Node A to Z

By Laeeq Khan Niazi

Learning Objective

- Students should be able to develop a backend services (APIs)
- Student should be able to manage project directory structure for scalable projects
- Student should be able to write manageable code
- Students should be able to store the data in the database using APIs

Prerequisites

- Programming
- Java Script Syntax
- Database

Software's Required

- Node (Server that run JS code)
- NPM (Package Manager)
- MongoDb (No SQL Database)
- MongoDbCompass (Software to visualize the database data)
- VS Code (Text Editor)
- Postman (Software for testing APIs)

What is node?

- Node.js is an open-source, cross-platform runtime environment built on Chrome’s V8 Engine.
- It is used to develop highly scalable backend as well as server-side applications.
- Node.js uses a single-threaded event loop architecture.

Check version

- **Check Node Version**
  - node –v make sure you have minimum 16 version installed
- **Check NPM Version**
  - npm –v 8 or higher must be installed

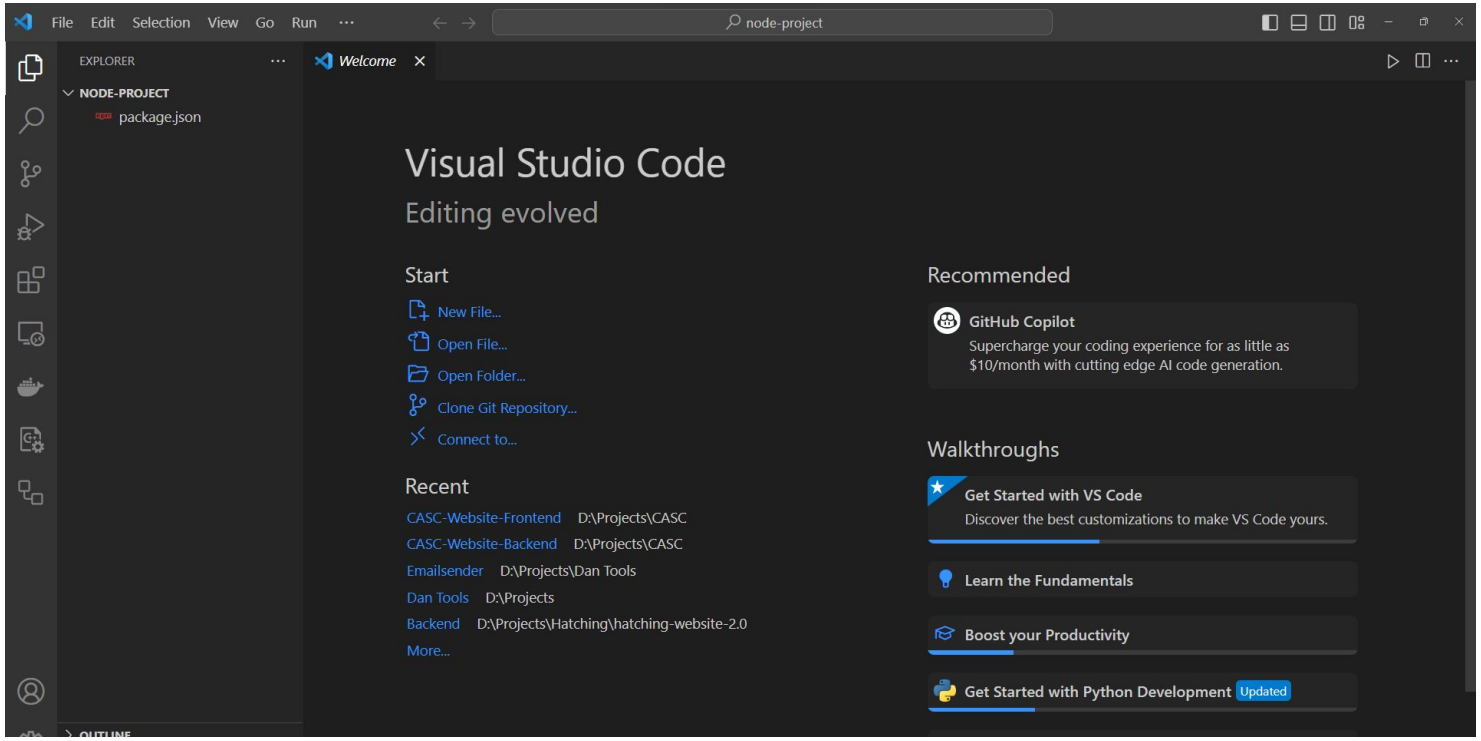
Create your node project

- Navigate to your required destination where you want to create your node project
- Use **npm init** command to create a new project
- It will create a project and ask you to enter some project details

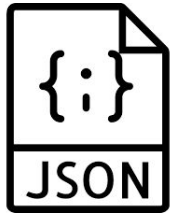
```
package name: (project) my-first-project
version: (1.0.0)
description: online courses website backend
entry point: (index.js) server.js
test command:
git repository:
keywords:
author: laeeq
license: (ISC)
```

Open your project

- Use **code .** Command to open your code into your default editor or first open **VS Code** and then from the file use open folder option to open the project

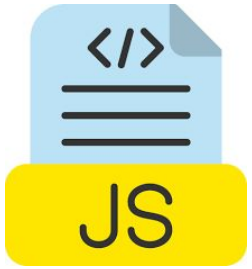


Package.json

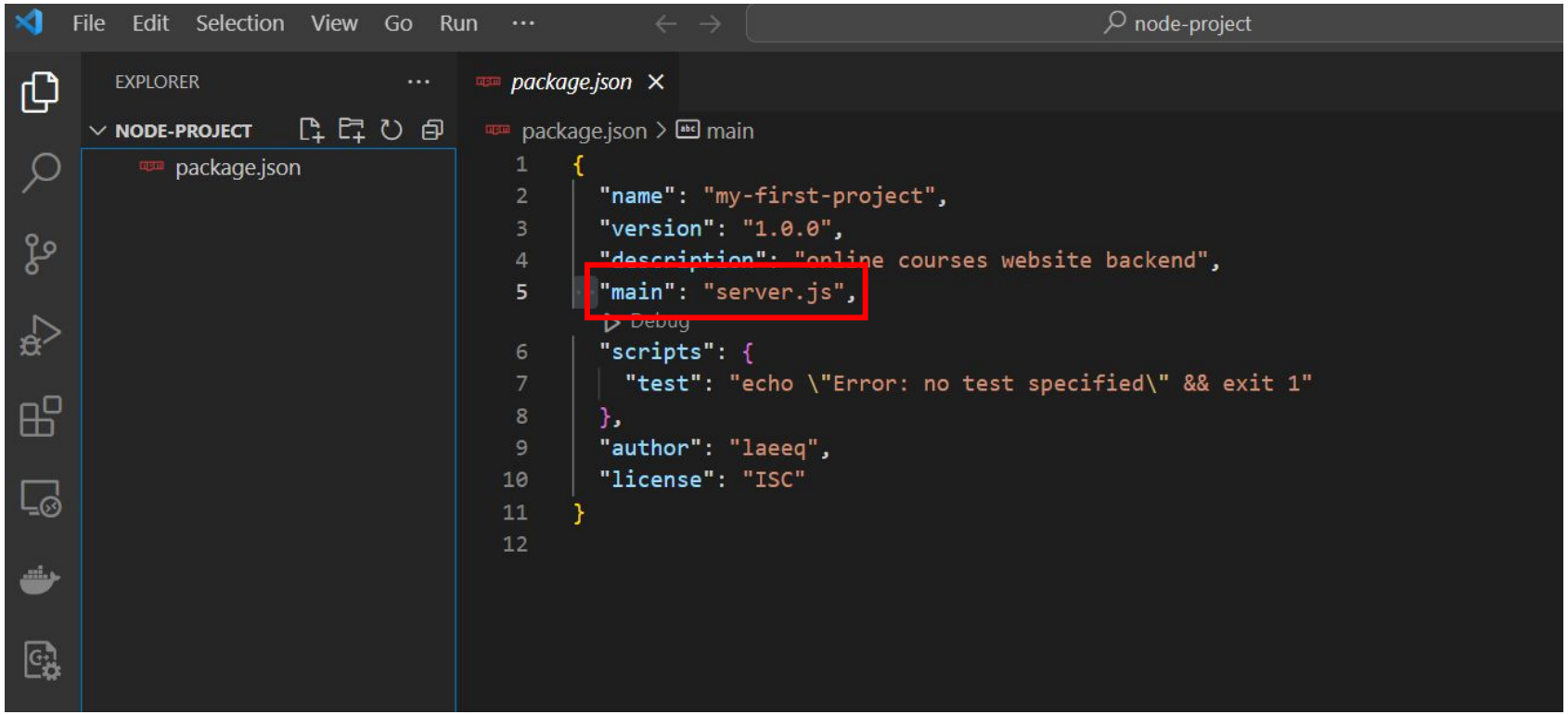


- This file is used as a manifest, storing information about applications, modules, packages, and more.
- Major roles of Package.json file
  - Dependency Management : It contains the packages
  - Script Definitions : npm run or other custom commands for start and test and build
  - Project Metadata: contains metadata about your project, including its name, description, version, author, and license information
  - Project Configuration: such as the entry point for your application, environment variables, and other project-specific settings.

Let's create your first js file

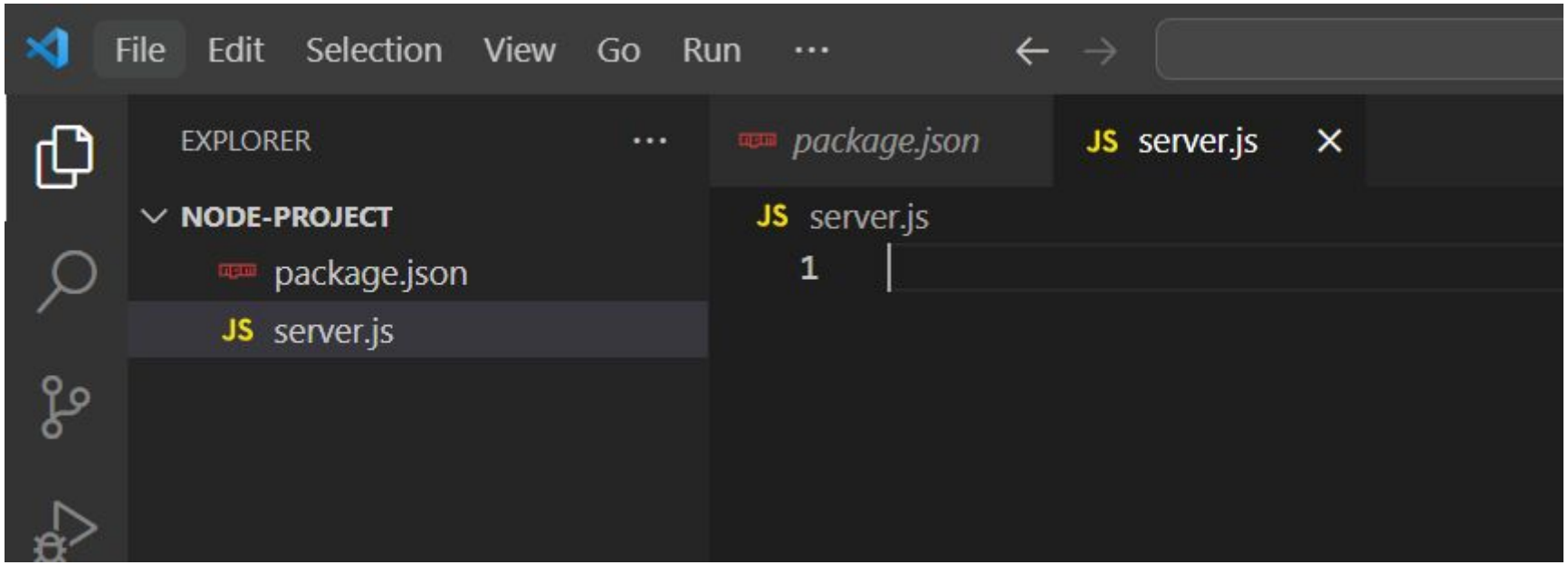


- Create the file with same name that is available in package.json file with main key



Project

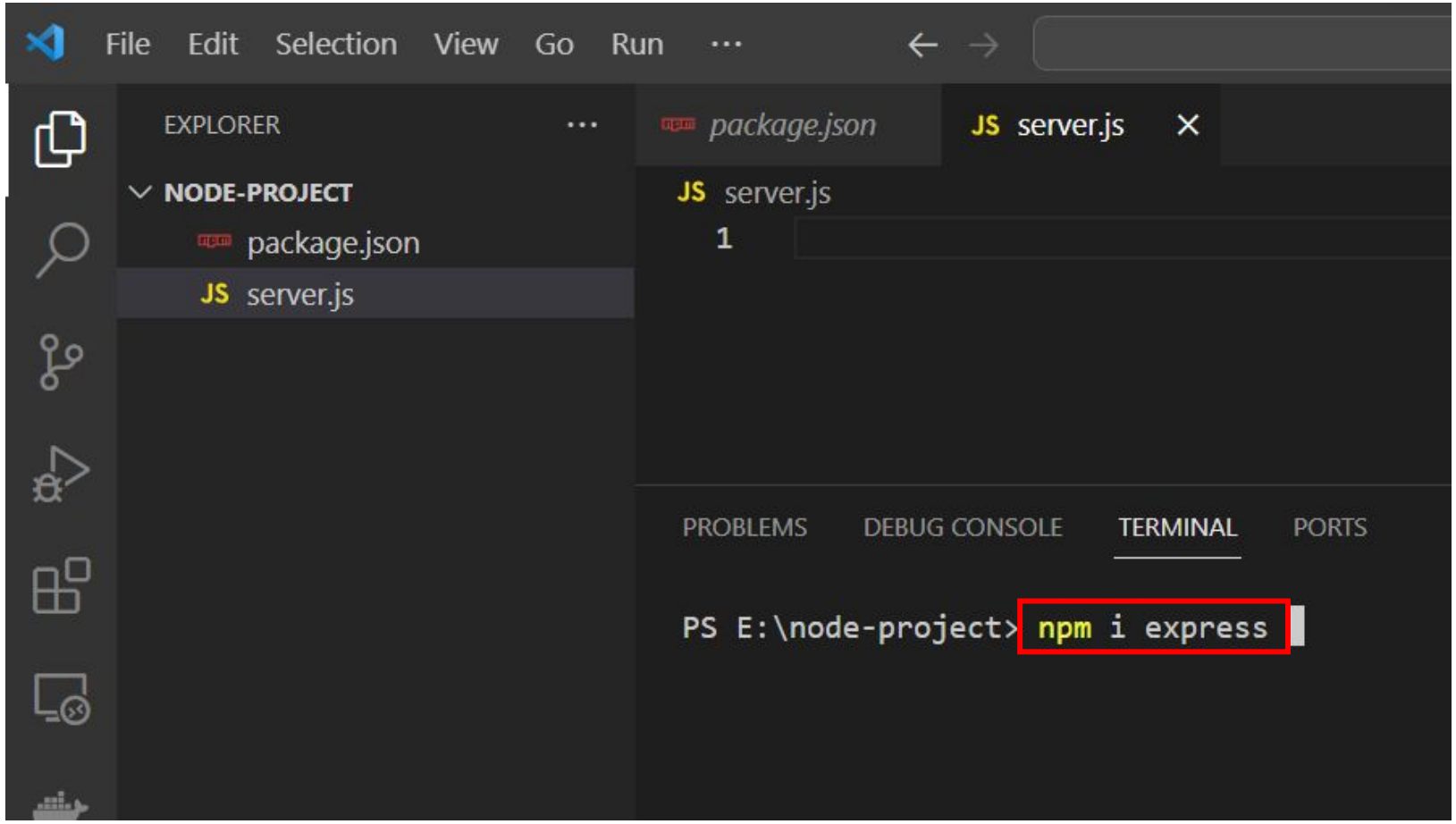
- Make sure your project directory structure looks like this



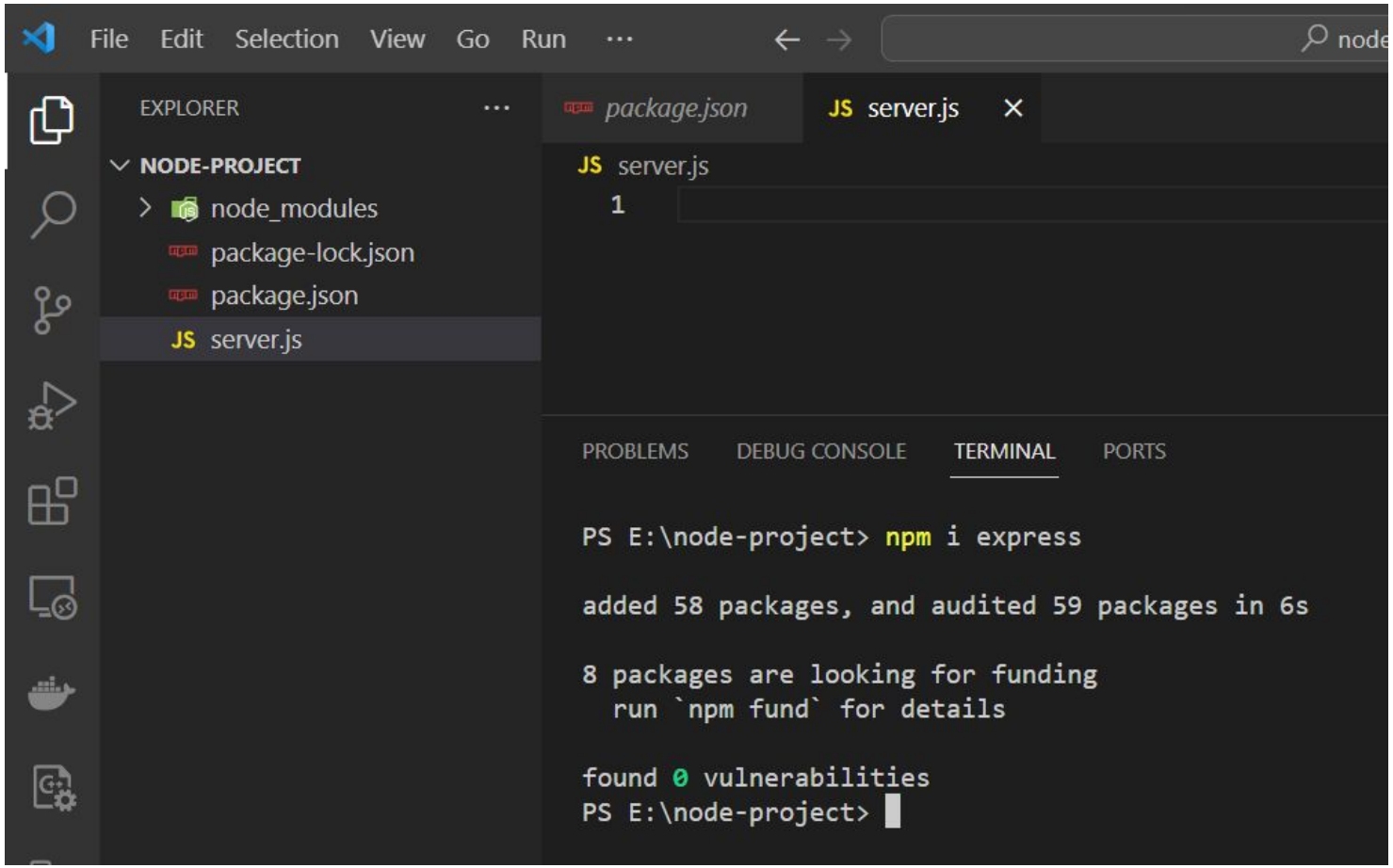
Now install your first library express.js

- We can certainly write the node server in pure JavaScript, but it can be a bit challenging and may not offer the flexibility we need. Therefore, we've opted to utilize Express.js for developing our APIs, as it provides a more streamlined and adaptable approach.
- Here is command to install the express in your node project

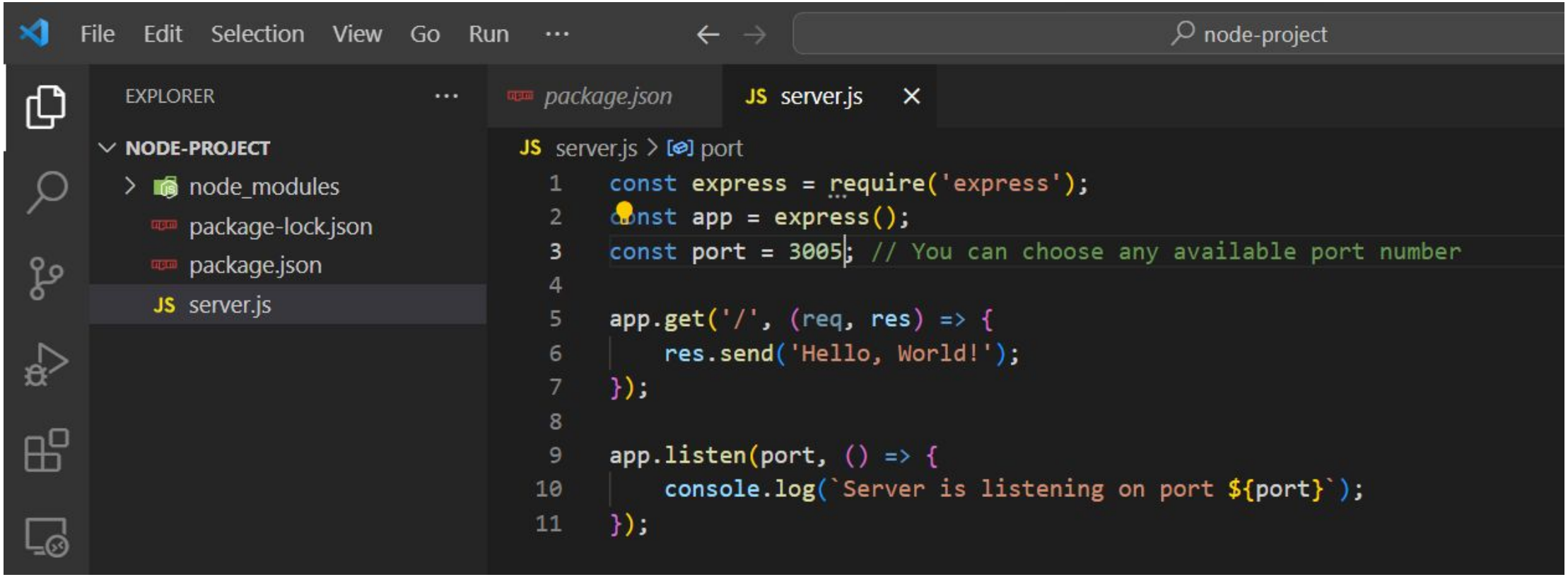
Make sure you run this command in your node project folder



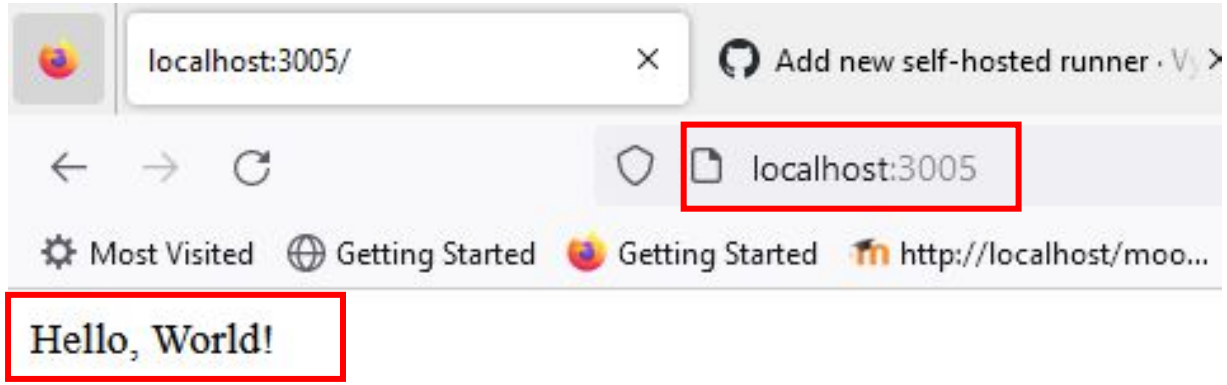
This command will create some file and folder to store the package files and information



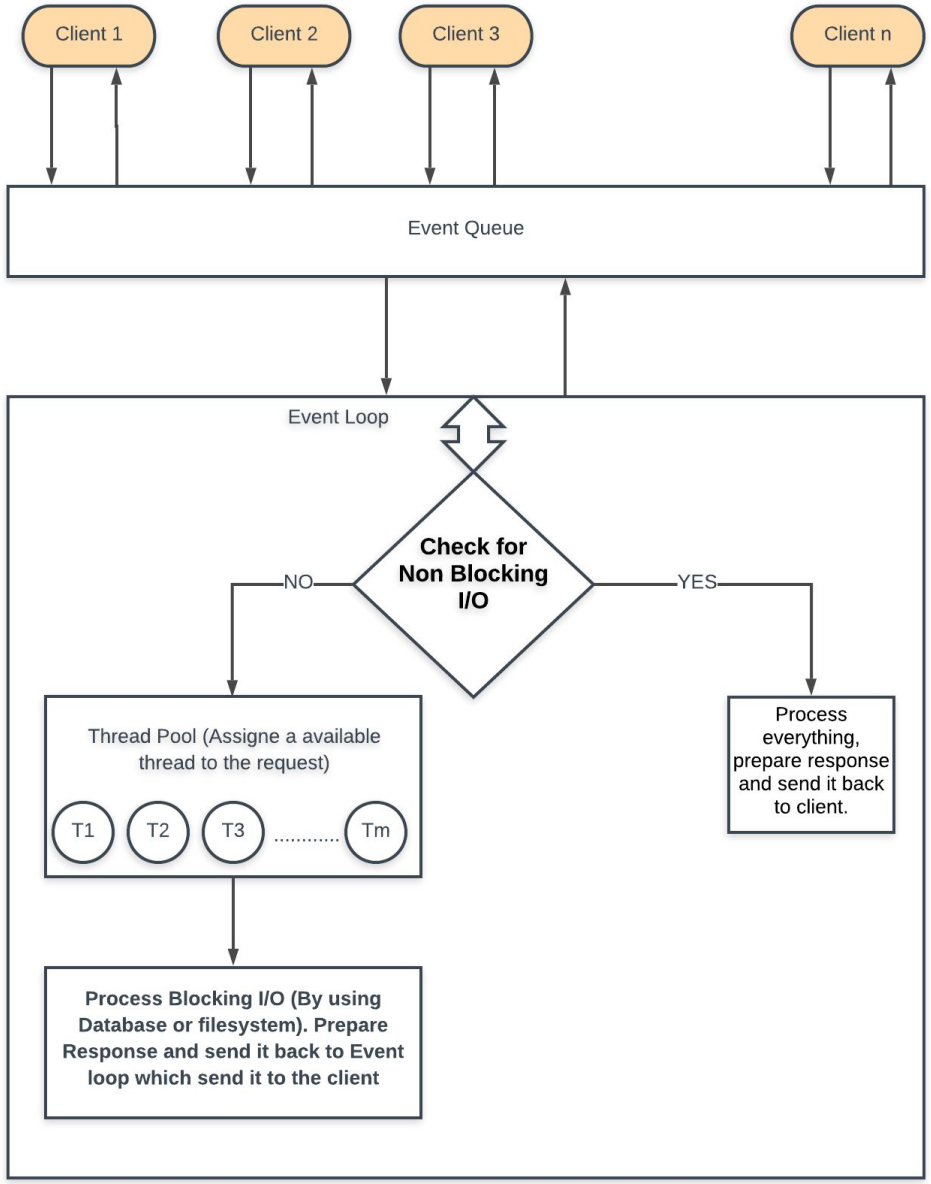
Express **hello word** write, run and see



In your browser type the localhost with your project port number and you will see the result of your application



What is happening?



Class Task

- Make another API /welcome
- If you hit localhost:3000/welcome it in browser it send you response like Welcome yourname

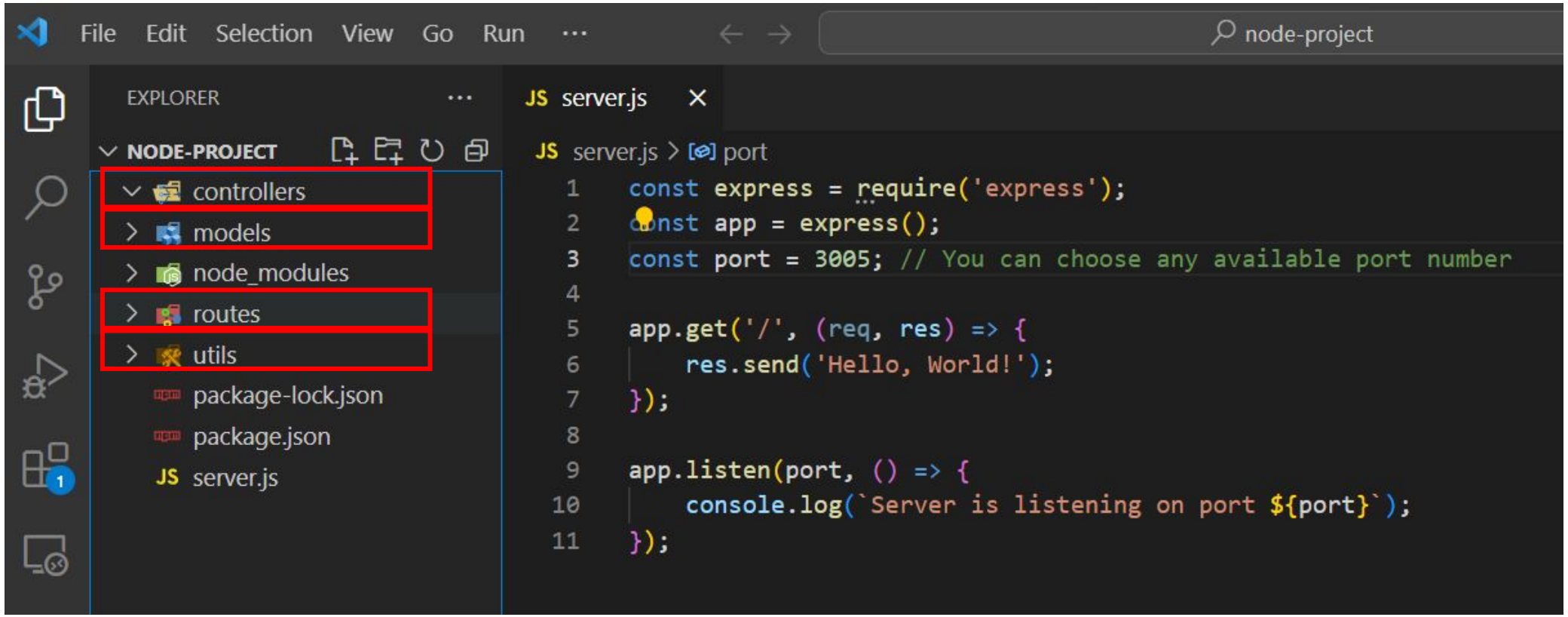
Discussion

- This was a simple hello word API example
- But if we have 20 or 50 or 100 API we will make all the APIs in the server.js file????

- How we can do it or we can manage large projects??

For Professional Web Development

- We make following directors mostly but still its your choice you can chose any other design pattern to complete project



Controllers

- Controllers handle the application's logic and act as intermediaries between the routes (HTTP endpoints) and the models (data layer). They contain functions that define how the application responds to different HTTP requests.
- **Example**: userController.js, productController.js, etc.

Typical Responsibilities:

- Parsing and validating request data.
- Calling appropriate functions from the models.
- Formatting and sending responses to the client.

Models

Models represent the data structures and database interactions of your application. They encapsulate the data schema, database queries, and business logic related to data.

Example: userModel.js, productModel.js, etc.

Typical Responsibilities:

- Defining the data schema using Object-Relational Mapping (ORM) or database libraries (e.g., Mongoose for MongoDB, Sequelize for SQL databases).
- Executing database queries and transactions.
- Enforcing data validation and business rules.

Routes:

Routes define the HTTP endpoints and map them to specific controller functions. They determine how incoming requests are routed to the appropriate controller methods.

Example: userRoutes.js, productRoutes.js, etc.

Typical Responsibilities:

- Defining the URL routes and HTTP methods (GET, POST, PUT, DELETE).
- Binding route handlers (controller functions) to specific routes and methods.

Utils (Utilities)

The "utils" folder typically contains utility functions or modules that can be used throughout the application. These utilities are not directly related to models, controllers, or routes but provide general-purpose functionality.

Example: helper.js, validation.js, auth.js, etc.

Typical Responsibilities:

- Reusable functions such as date formatting, string manipulation, or encryption.
- Custom validation functions.
- Authentication and authorization middleware.

Directory Structure Example

```
project/
|-- controllers/
|   |-- userController.js
|   |-- productController.js
|-- models/
|   |-- userModel.js
|   |-- productModel.js
|-- routes/
|   |-- userRoutes.js
|   |-- productRoutes.js
|-- utils/
|   |-- helper.js
|   |-- validation.js
|   |-- auth.js
|-- app.js (or server.js)
|-- package.json
```

Lets install some dependences

- npm i mongoose body-parser

- **mongooes** is used to mongodb operations
- The **body-parser** middleware plays a crucial role in your Node.js application when dealing with HTTP requests that have a request body, such as POST and PUT requests. Its primary purpose is to parse the request body and make it available under req.body in a more convenient format, such as JSON or URL-encoded data, for your route handlers to work with.

Middleware??

- Middleware in Express.js is a fundamental concept that refers to functions that are executed during the request-response cycle. These functions have access to the **request (req)** and **response (res)** objects and can perform various tasks or modifications to the request or response or even terminate the request-response cycle.

Your database connection code

utils > JS db.js > ...

```javascript
const mongoose = require('mongoose');
mongoose.set('strictQuery', true); // or true

mongoose.connect('mongodb://127.0.0.1:27017/product-apis', {
    useNewUrlParser: true,
    useUnifiedTopology: true,
});
const db = mongoose.connection;
db.on('error', (err) => {
    console.log('Failed to connect with db');
});
db.once('open', () => {
    console.log('Connected with db');
});
```

Use following library and middleware call in your server.js file

- const bodyParser = require('body-parser’);

- app.use(bodyParser.json());

Server.js file should look like that

JS server.js > ...

```
1   const express = require('express');
2   const bodyParser = require('body-parser')
3   const app = express();
4   const port = 3005; // You can choose any available port number
5
6   // Middle wares
7   app.use(bodyParser.json());
8
9
10  app.listen(port, () => {
11      console.log(`Server is listening on port ${port}`);
12  });
```

We are going to develop API for Products

- We will be working with product name, description and price

Create product model in model's directory

models > JS product.js > ...

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

const productSchema = new mongoose.Schema({
    name: String,
    description: String,
    price: Number,
}, { timestamps: true });

module.exports = mongoose.model('Product', productSchema);
```

Create your productController.js file

controllers > JS productController.js > updateProduct

```javascript
// controllers/productController.js
const Product = require('../models/product');

// Create a new product
async function createProduct(req, res) {
    try {
        const product = await Product.create(req.body);
        res.status(201).json(product);
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
}
```

This function will take the product data from the API and store into the database

Get all products

```
14  // Get all products
15  async function getAllProducts(req, res) {
16      try {
17          const products = await Product.find();
18          res.json(products);
19      } catch (err) {
20          res.status(500).json({ error: err.message });
21      }
22  }
23
```

Export the function to make it visible to other js files

controllers > JS productController.js > ...

```javascript
// controllers/productController.js
const Product = require('../models/product');

// Create a new product
async function createProduct(req, res) {
    try {
        const product = await Product.create(req.body);
        res.status(201).json(product);
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
}


module.exports = {
    createProduct,
};
```

Create the router file and map your function API end point

```
// routes/productRoutes.js
const express = require('express');
const router = express.Router();
const productController = require('../controllers/productController');
// Create a new product
router.post('/products', productController.createProduct);

module.exports = router;
```

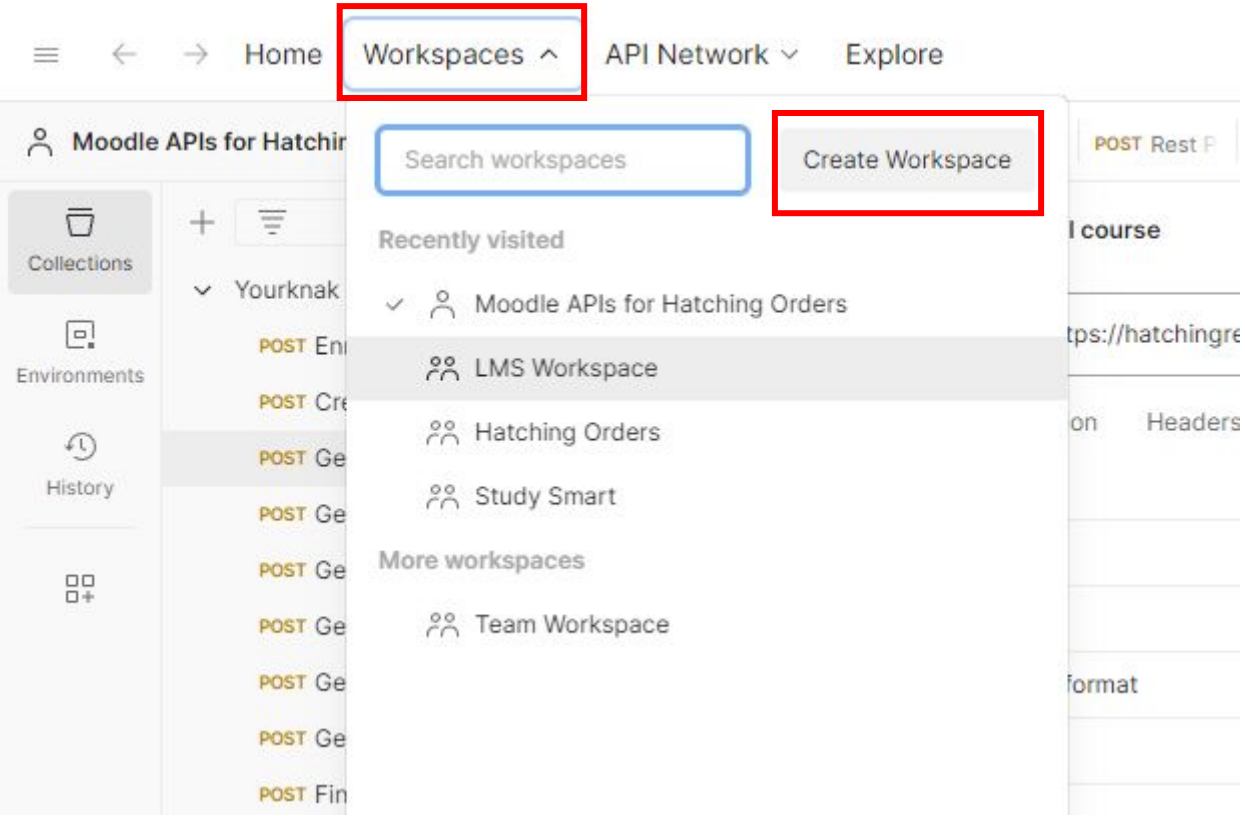
Here use the middle where and pass the route and the routers

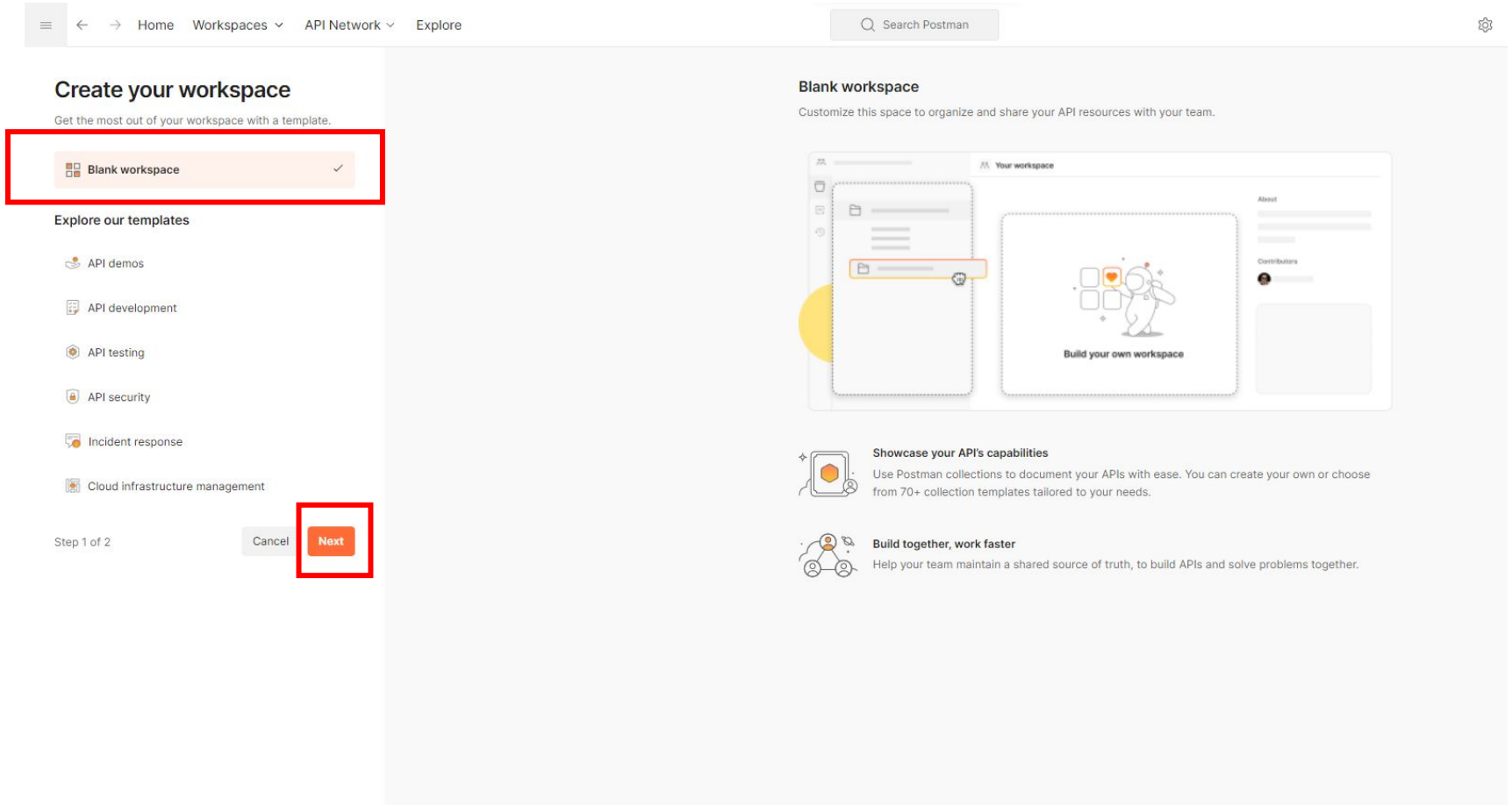
JS server.js > ...

```javascript
const express = require('express');
const productRoutes = require('./routes/productRoutes');
const bodyParser = require('body-parser')
const app = express();
const port = 3005; // You can choose any available port number

// Middle wares
app.use(bodyParser.json());

//Apis
app.use('/api', productRoutes);


app.listen(port, () => {
    console.log(`Server is listening on port ${port}`);
});
```

Use postman to test the API

- Open postman
- Create your workspace
- Setup the global variable to store the base URL
- Create your post request and pass the data to API get the response and check the data into the mongodb database.


Home
Workspaces
API Network
Explore
Moodle APIs for Hatchir
Search workspaces
Create Workspace
Recently visited
Moodle APIs for Hatching Orders
LMS Workspace
Hatching Orders
Study Smart
More workspaces
Team Workspace
Collections
Environments
History
Yourknak
POST Rest
l course
tps://hatchingre
on
Headers
format


Home
Workspaces
API Network
Explore
Search Postman
Create your workspace
Get the most out of your workspace with a template.
Blank workspace
Explore our templates
API demos
API development
API testing
API security
Incident response
Cloud infrastructure management
Step 1 of 2
Cancel
Next
Blank workspace
Customize this space to organize and share your API resources with your team.
Your workspace
Build your own workspace
Showcase your API's capabilities
Use Postman collections to document your APIs with ease. You can create your own or choose from 70+ collection templates tailored to your needs.
Build together, work faster
Help your team maintain a shared source of truth, to build APIs and solve problems together.

Home Workspaces API Network Explore

Search Postman

Create your workspace

Name

Lab6

Summary

Test APIs

Who can access your workspace?

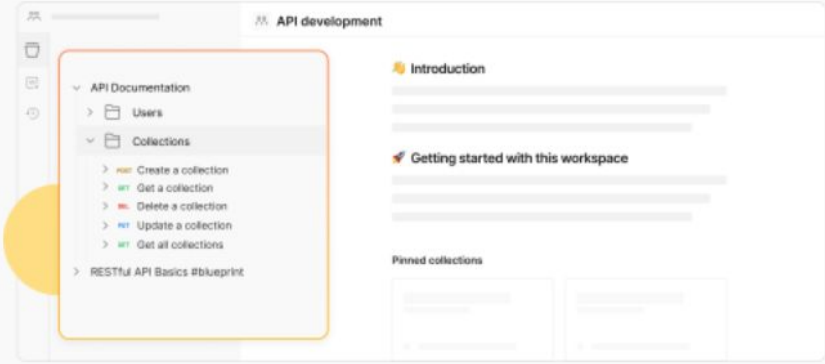
- Personal
  Only you can access
- Private
  Only invited team members can access
- Team
  All team members can access
- Partner
  Only invited partners and team members can access
- Public
  Everyone can view

Step 2 of 2

Back Create

API development

Helps you onboard new engineers to your team's API landscape faster.



Share knowledge in one place

Showcase collections with thorough documentation of your APIs to help your team stay on top of services you own.

Help developers catch up in no time

Curate ready-to-use endpoints for new developers to quickly send requests and understand workflows.

Home Workspaces API Network Explore

Search Postman

Create your workspace

Name

Lab6

Summary

Who can access your workspace?

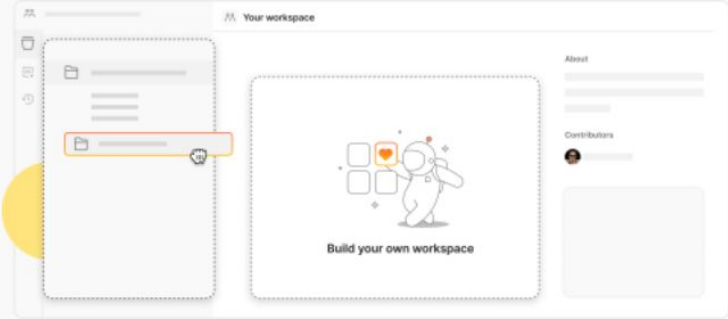
- Personal
  Only you can access
- Private
  Only invited team members can access
- Team
  All team members can access
- Partner
  Only invited partners and team members can access
- Public
  Everyone can view

Step 2 of 2

Back Create

Blank workspace

Customize this space to organize and share your API resources with your team.



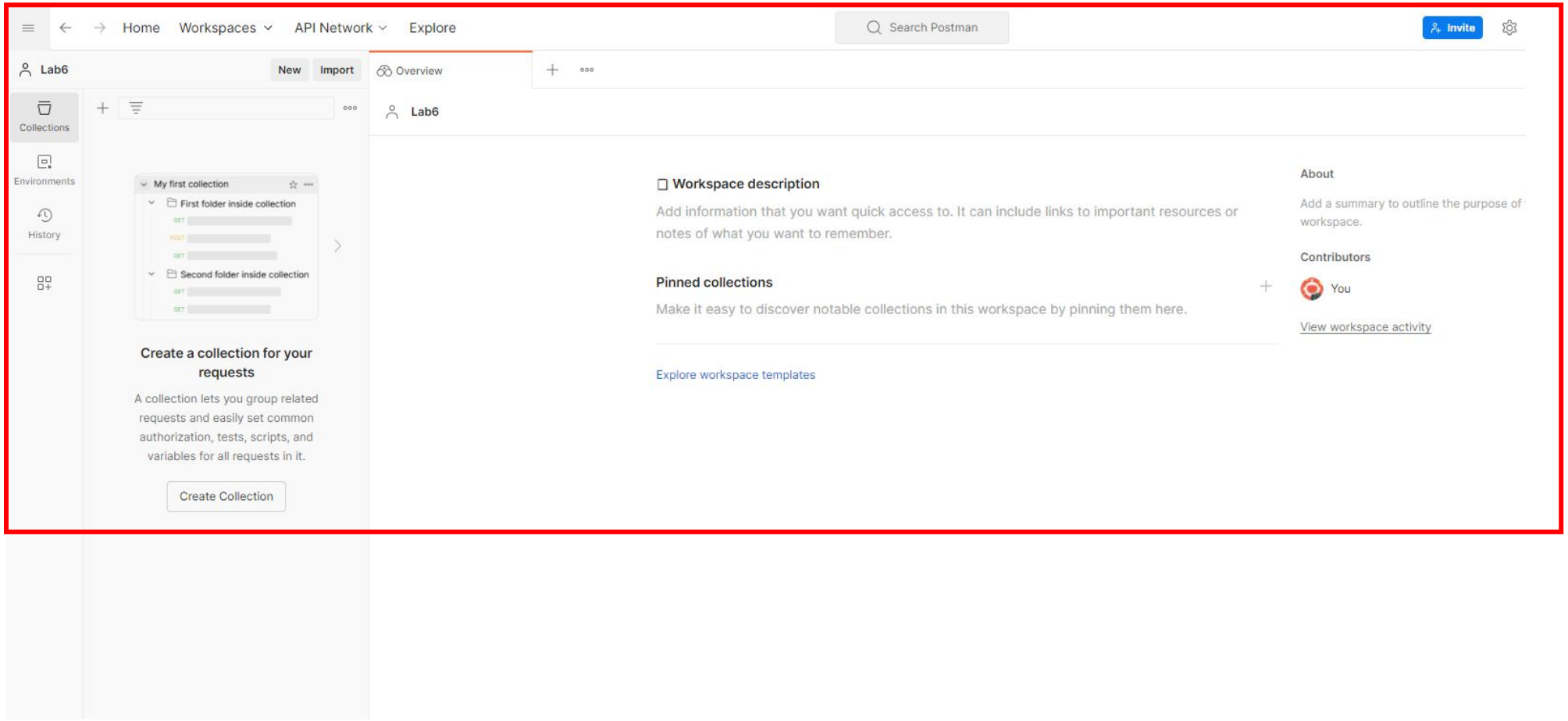
Showcase your API's capabilities

Use Postman collections to document your APIs with ease. You can create your own or choose from 70+ collection templates tailored to your needs.

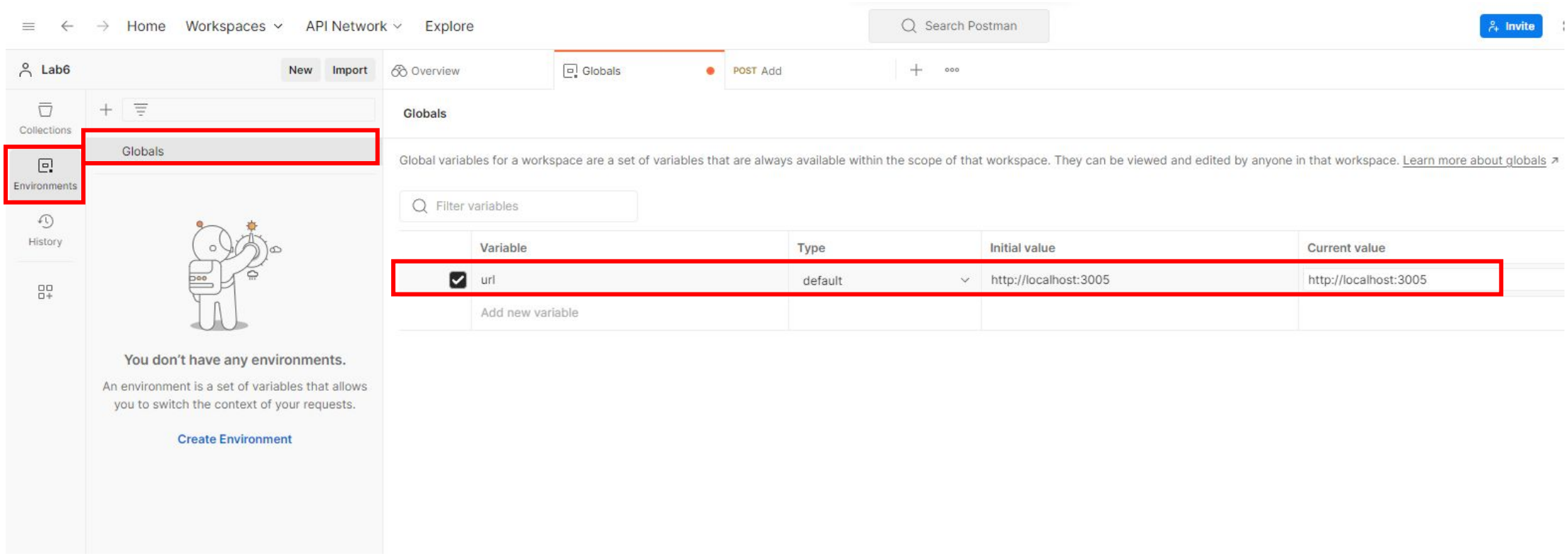
Build together, work faster

Help your team maintain a shared source of truth, to build APIs and solve problems together.

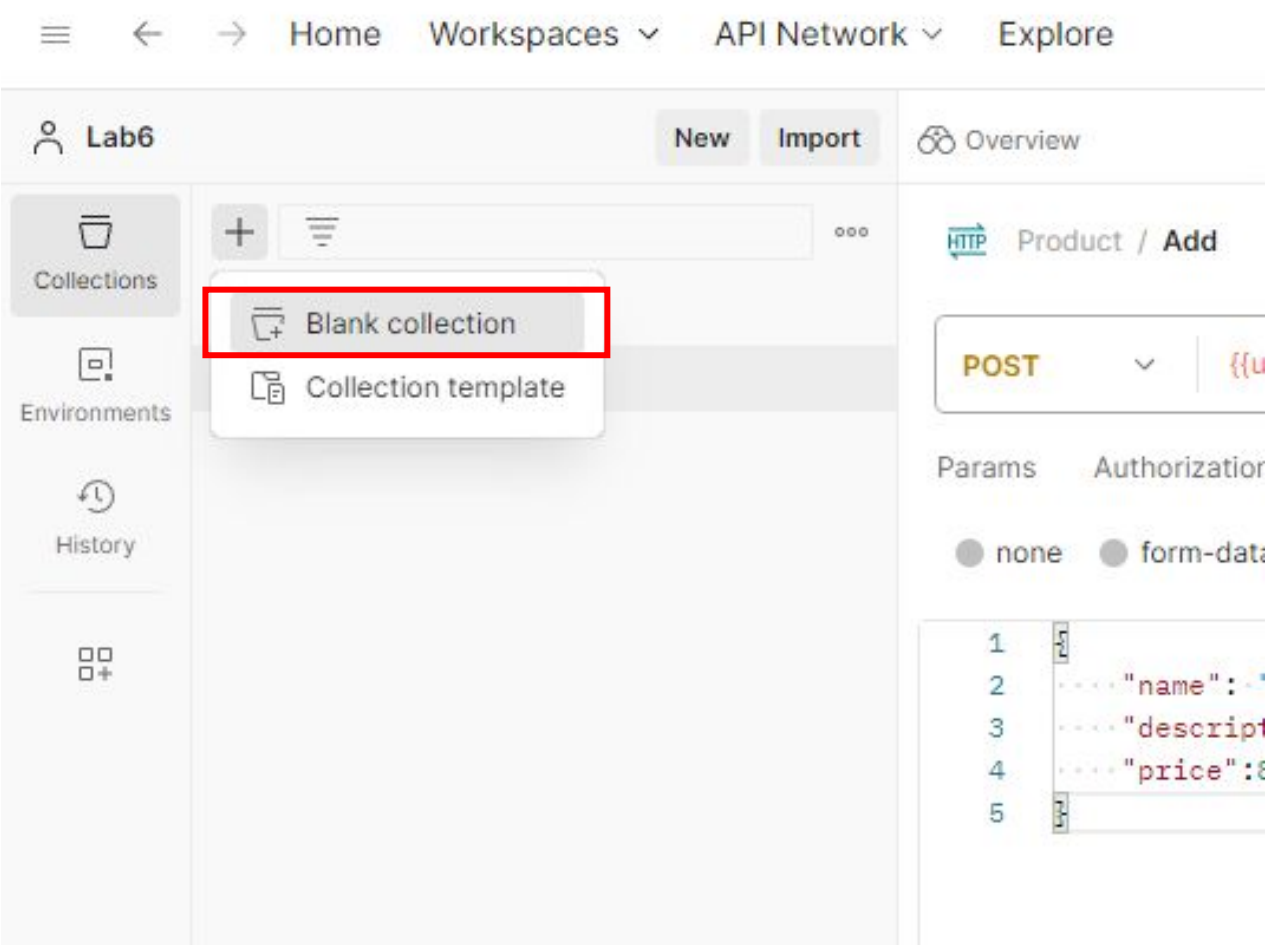
Create


Home
Workspaces
API Network
Explore
Search Postman
Invite
Lab6
New
Import
Overview
Collections
Environments
History
My first collection
First folder inside collection
Second folder inside collection
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
Lab6
Workspace description
Add information that you want quick access to. It can include links to important resources or notes of what you want to remember.
Pinned collections
Make it easy to discover notable collections in this workspace by pinning them here.
Explore workspace templates
About
Add a summary to outline the purpose of workspace.
Contributors
You
View workspace activity

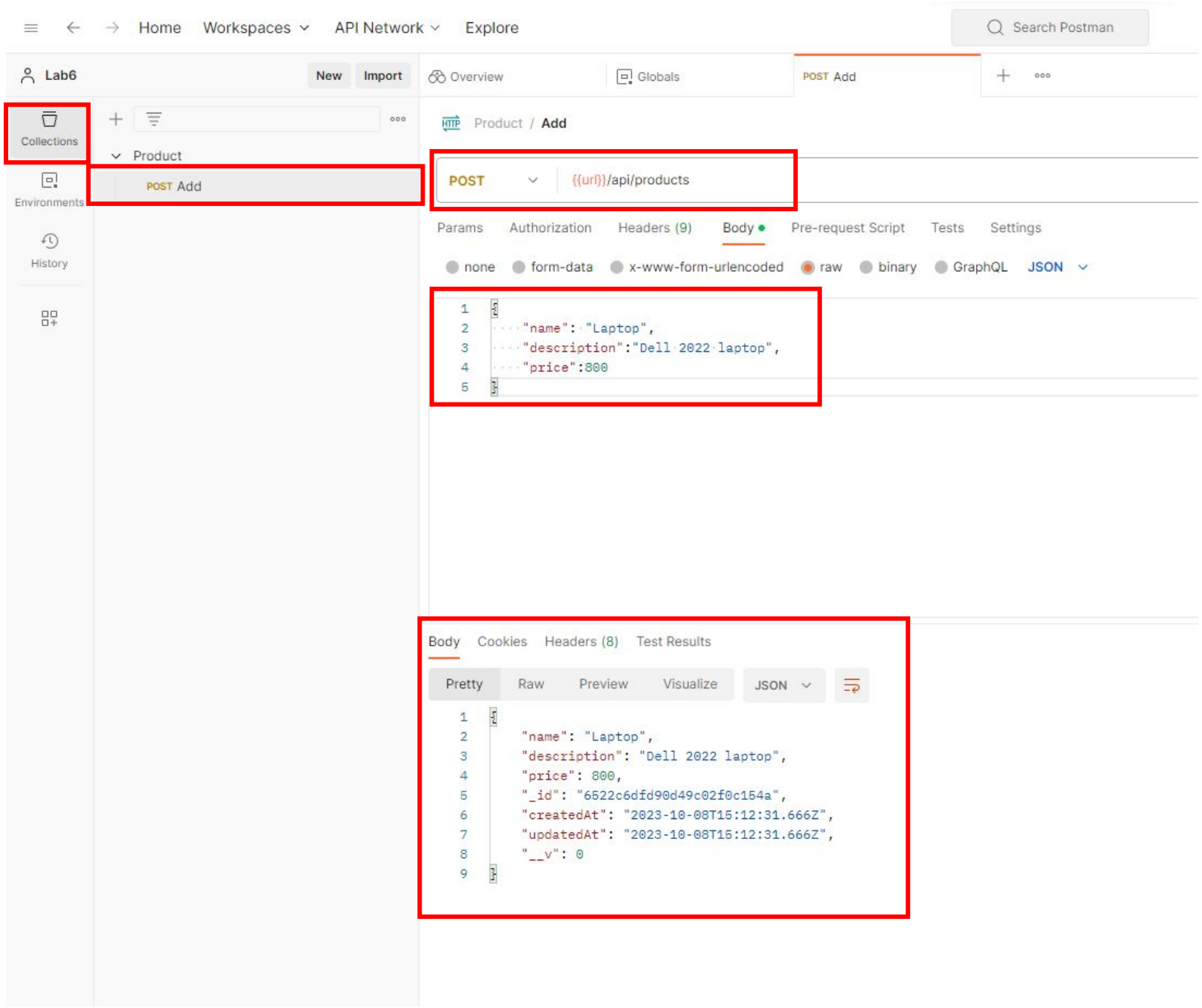
Setup the URL as global variable



Create a collection and add a request in the collection



Send the API request and check response



Install cors module and use it as middleware

- npm i cors

- What is cors?

CORS stands for Cross-Origin Resource Sharing. It is a security feature implemented by web browsers that controls and restricts web pages' ability to make requests to a different domain (origin) than the one that served the web page. CORS is an important security measure to prevent cross-origin attacks, like Cross-Site Request Forgery (CSRF) and Cross-Site Scripting (XSS).

Server.js code will look like that

JS server.js > ...

```
1   const express = require('express');
2   const productRoutes = require('./routes/productRoutes');
3   const bodyParser = require('body-parser')
4   const cors = require('cors');
5   const app = express();
6   const port = 3005; // You can choose any available port number
7
8   // Middle wares
9   app.use(bodyParser.json());
10  app.use(cors());
```

Let's make all APIs for product

- Update
- Delete
- Get All

Update product by Id

```
// Update a product by ID
async function updateProduct(req, res) {
    try {
        const { id } = req.params;
        const updatedProduct = await Product.findByIdAndUpdate(id, req.body, { new: true });
        res.json(updatedProduct);
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
}
```

Get All Products

```
// Get all products
async function getAllProducts(req, res) {
    try {
        const products = await Product.find();
        res.json(products);
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
}
```

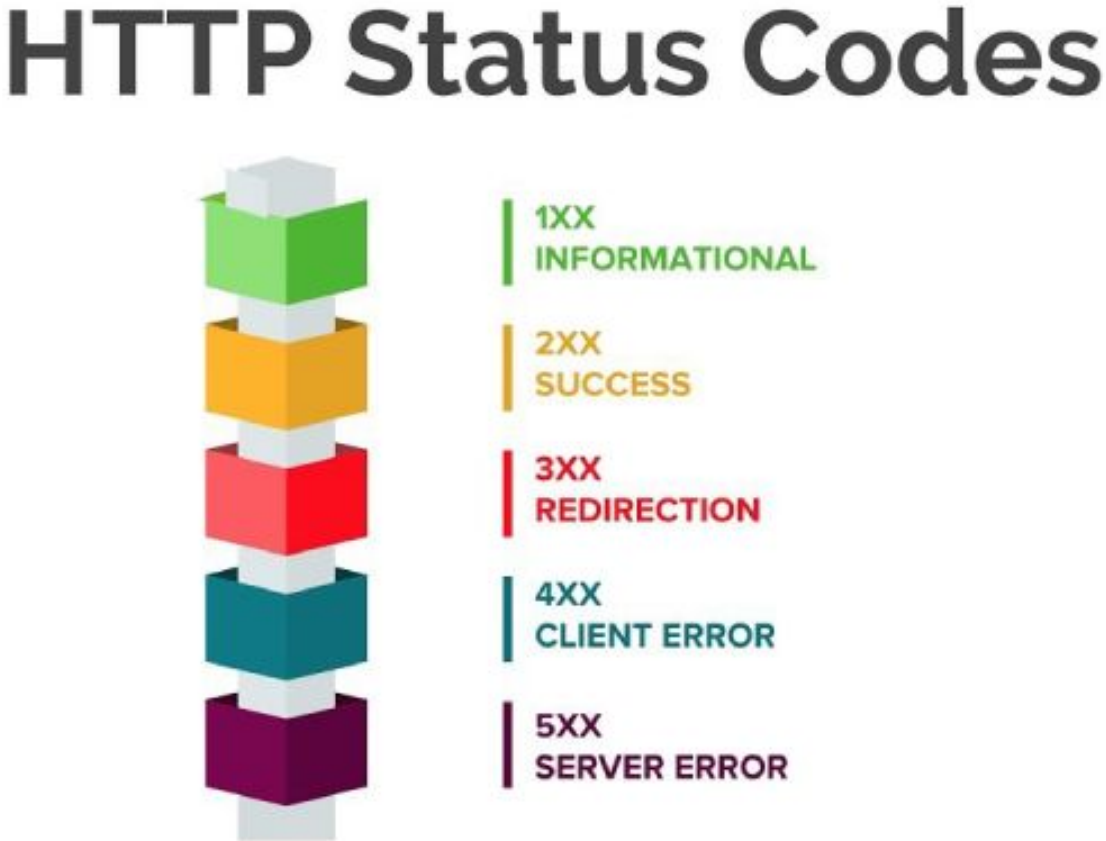
Delete product by Id

```
// Delete a product by ID
async function deleteProduct(req, res) {
    try {
        const { id } = req.params;
        await Product.findByIdAndRemove(id);
        res.sendStatus(204);
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
}
```

Export modules

```
45
46  module.exports = {
47      createProduct,
48      getAllProducts,
49      updateProduct,
50      deleteProduct,
51  };
52
```

Status Codes



Create **productRoutes** File in Routes directory

routes > JS productRoutes.js > ...

```
1   // routes/productRoutes.js
2   const express = require('express');
3   const router = express.Router();
4   const productController = require('../controllers/productController');
5   // Create a new product
6   router.post('/products', productController.createProduct);
7   // Get all products
8   router.get('/products', productController.getAllProducts);
9   // Update a product by ID
10  router.put('/products/:id', productController.updateProduct);
11  // Delete a product by ID
12  router.delete('/products/:id', productController.deleteProduct);
13
14  module.exports = router;
15
```

import productController

Server.js file

JS server.js > ...

```javascript
const express = require('express');
const productRoutes = require('./routes/productRoutes');
const bodyParser = require('body-parser')
const app = express();
const port = 3005; // You can choose any available port number

// Middle wares
app.use(bodyParser.json());

//Apis
app.use('/api', productRoutes);


app.listen(port, () => {
    console.log(`Server is listening on port ${port}`);
});
```

Test your all APIs

Your task

- Create 4 APIs for Student with data student name, registration no, father name, date of birth and contact

Let’s make your life easier

- Install nodemon and it will help you to restart your application on every save.

```
npm install --save-dev nodemon
```

- It will install the nodemon as development library it will not work in production because we don’t need it in production.

Add following line in your package.json

JS server.js | package.json × | JS db.js | JS productController.js

package.json > {} scripts > serve

```
1   {
2     "name": "my-first-project",
3     "version": "1.0.0",
4     "description": "online courses website backend",
5     "main": "server.js",
      ▷ Debug
6     "scripts": {
7       "test": "echo \"Error: no test specified\" && exit 1",
8       "serve": "nodemon server.js"
9     },
10    "author": "laeeq",
```

Use following command to run the application

npm run serve

Now when you will save your file you don’t need to restart the node server it will restart automatically and help you to speed up your development process