Welcome to the CoGrammar Lecture: Express.js

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

Full Stack Web Development Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Full Stack Web Development Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: **www.hyperiondev.com/support**
- Report a **safeguarding** incident: **www.hyperiondev.com/safeguardreporting**
- We would love your **feedback** on lectures: **Feedback on Lectures**

Skills Bootcamp
8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✅ Criterion 1: Initial Requirements

Timeframe: First 2 Weeks
Guided Learning Hours (GLH): Minimum of 15 hours
Task Completion: First four tasks

Due Date: 24 March 2024

✅ Criterion 2: Mid-Course Progress

60 Guided Learning Hours

Data Science - **13 tasks**
Software Engineering - **13 tasks**
Web Development - **13 tasks**

Due Date: 28 April 2024

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3: Course Progress

Completion: All mandatory tasks, including Build Your Brand and resubmissions by study period end
Interview Invitation: Within 4 weeks post-course
Guided Learning Hours: Minimum of 112 hours by support end date (10.5 hours average, each week)

✅ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship Outcome: Document within 12 weeks post-graduation
Relevance: Progression to employment or related opportunity

Lesson Objectives

- Explain the purpose of Express.js as a Node.js framework
- Create and manage routes in Express.js applications
- Define the concept of middleware
- Designing RESTful APIs and perform CRUD functionalities with Express.js
- Testing RESTful APIs with Postman.

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar

Express

May 2024

Introduction to Express

CoGrammar

Express.js

Definition and Use Cases

- Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web applications.
- Express.js' main features include:
  - **Routing:** defines routes for handling different HTTP methods (GET, POST, PUT, DELETE).
  - **Middleware:** functions having access to request and response objects in the application.

Express.js

Definition and Use Cases

- **Static File Serving:** built in middlewares in place for serving static files (HTML, CSS, JS, Images).
- **Creating APIs:** Easy creation of API endpoints for web applications. The endpoints can perform tasks such as interacting with a database e.t.c.

- Express.js' lightweight and unopinionated nature makes it popular among developers for building scalable web solutions

Prerequisites for Express.js

- **Node.js:** make sure node.js is installed on your laptop
  - ➢ Confirm by running **`node -v`**
- **Code Editor:** preferably Visual Studio Code

CoGrammar

Configuring Node.js and Installing Express.js

CoGrammar

Installation and Configuration

Setting up Express.js

- Create a folder where your application will live and change directory to it:
  - mkdir server
  - cd server
- Initialize your package.json file with the default settings:
  - npm init -y (The y is optional if you need to skip prompts)
- Install express.js:
  - npm install express

Installation and Configuration

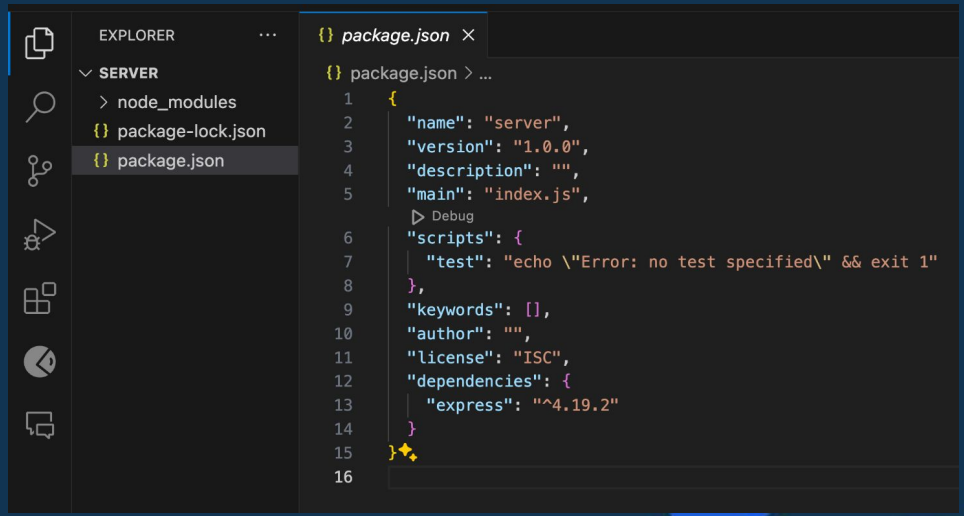
Setting up Express.js

- The commands executed should initialize a package.json file with predefined settings.
- After installing Express.js, the package name should be listed in the dependencies section of the package. json.
- All packages installed are stored in the node_modules folder.
  NOTE: Make sure the node_modules folder is .gitignored to avoid pushing it to github.

Installation and Configuration

Note the express inside the dependencies.

```
WalobwaD@users-MacBook-Pro Hyperion % mkdir server
WalobwaD@users-MacBook-Pro Hyperion % cd server
WalobwaD@users-MacBook-Pro server % npm init -y
Wrote to /Users/WalobwaD/coding/Hyperion/server/package.json:

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


WalobwaD@users-MacBook-Pro server % npm install express

added 64 packages, and audited 65 packages in 11s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
WalobwaD@users-MacBook-Pro server %
```



CoGrammar

Creating an Express.js Server

CoGrammar

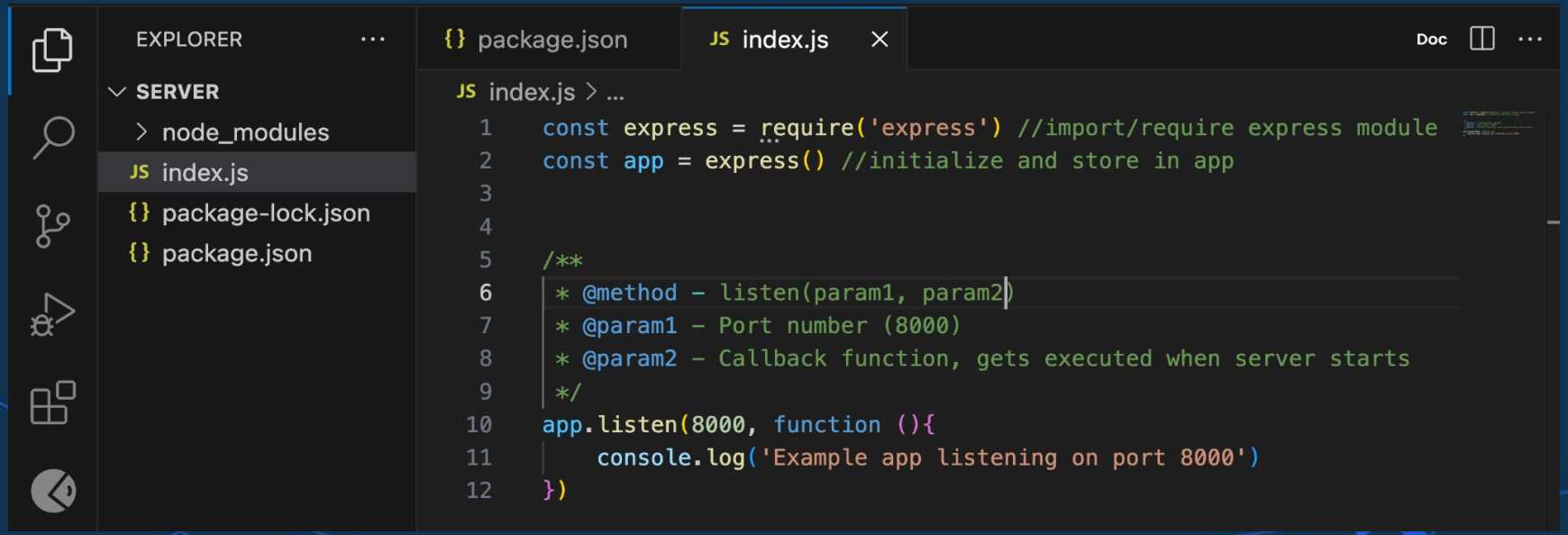
Creating a server

Running a port on your local machine

- From the configuration we just built, we can create an **index.js** file to act as your root file.
- We'll go ahead and import the express.js we just installed using common js syntax and reference it to a variable called app so whenever we need an express property, we'll use the app variable.
- The express module contains a **listen method** which takes in two arguments (**the port number** and **a callback function**). This will be the method to create the needed server for our app to run.

Creating a server

Running a port on your local machine



CoGrammar

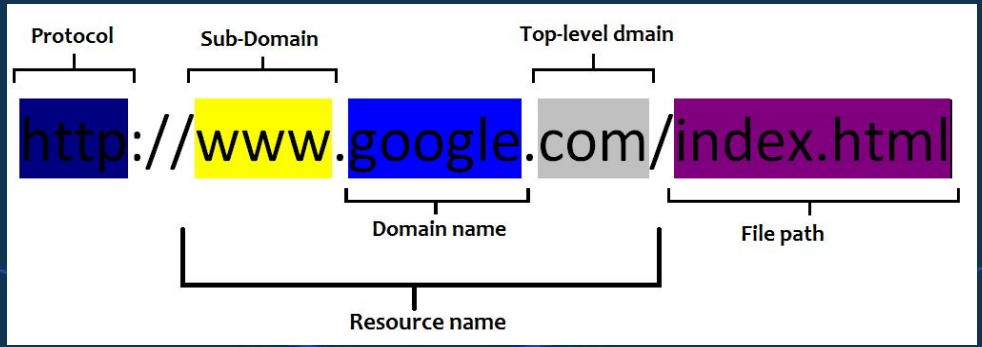
Routes in Express.js

CoGrammar

Routes

Determining how an application responds to a client's request to a particular endpoint.

- Since we have the server configured, we need a routing mechanism to react to users' requests.
- Having a brief of routes and URLs will assist. Below is an example of a full URL.



Routes

URL Parts

- **Protocol (Scheme):** Specifies the protocol or method used to access the resources such as **HTTP, HTTPS, FTP** e.t.c
- **Subdomain:** A prefix to the main domain name indication a specific department within the domain. **www**
- **Domain:** The main part of the URL that identifies the website or web service.
- **Top Level Domain:** Used to categorize the internet domain space into different groups based on their purpose of location.

Routes

URL Parts

- **Port:** Identifies a specific endpoint within a server separated from the domain by a colon. example.com**:8080**
- **Path:** specifies the location of a specific resource or file within the domain directory structure. E.g example.com**/path/file**
- We'll be more interested in the path section of the URL by identifying the path that was requested and providing a response based on that path.

Creating a route for your application

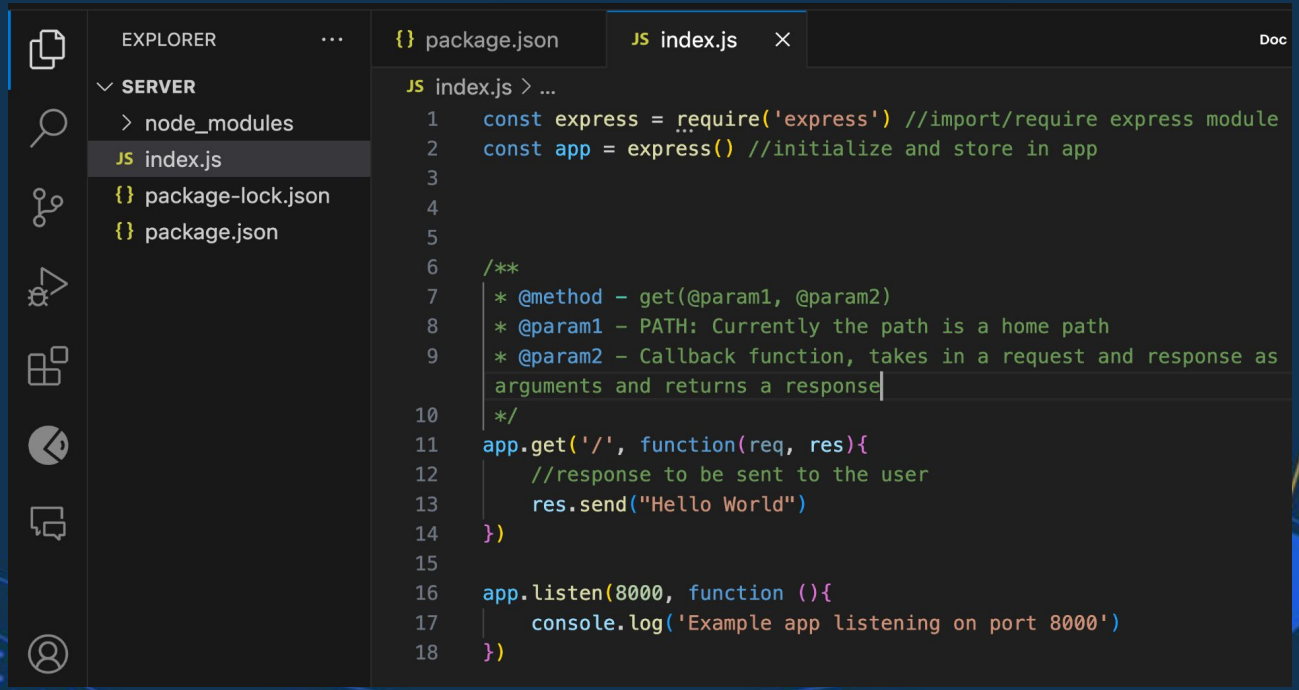
- In Express.js (or any backend frameworks) there are routing methods that specify the type of requests.
- Common routing methods we'll use in express.js:
  - **GET:** Retrieves data from server
  - **POST:** Submit data to be processed in the server
  - **PUT:** Updates or replaces data existing in the server with submitted data.
  - **DELETE:** Delete a specified resource from the server.

Creating a route for your application

- We'll create our first path with the GET method.
- From the app variable, we can call the app.get() which takes in two main arguments. (**The path** and **a callback function**).
- The callback function in this case becomes the route handler, it determined the kind of response the user will get after making a request to a specific path on the server.

Creating a server

Adding a start script to the server



CoGrammar

Creating a server

Adding a start script to the server

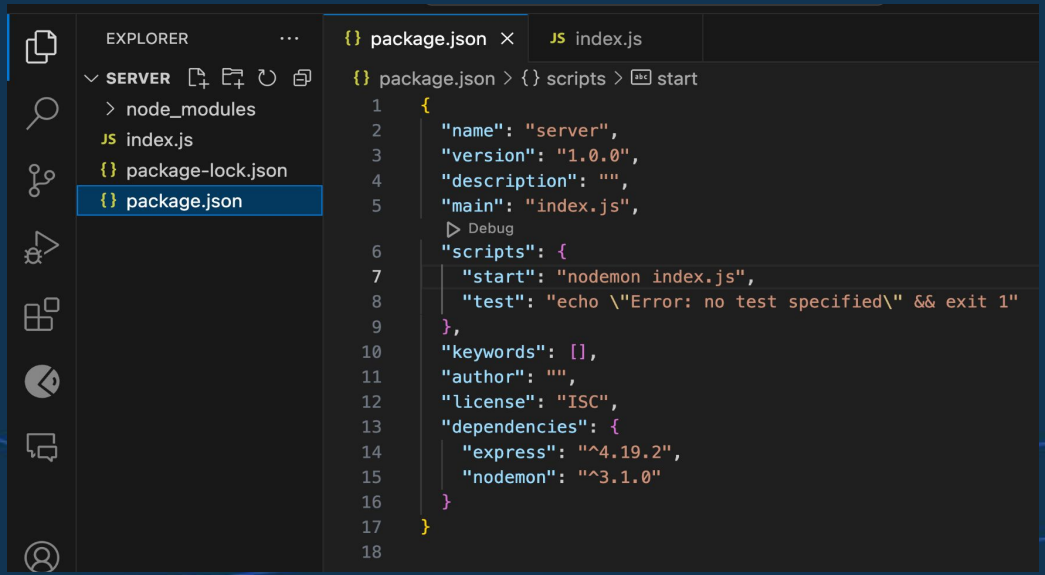
- We now need to start our server, you can run it directly using Node.js by executing: **`node index.js`** on the terminal.
- Instead we're going to use a library called nodemon to assist.
  - Nodemon is a tool that helps develop Node.js based applications by automatically restarting the node application when file changes in the directory are detected.
- We need to install it in order to use it using the command:

```
npm install nodemon
```

Creating a server

Adding a start script to the server

- After installing nodemon, in your package.json file, you can insert a “start” property inside your scripts object and include the text: **nodemon {nameOfFile}**



CoGrammar

Creating a server

Adding a start script to the server

- You can now run the project using

```
npm start
```

- At the moment from the configuration done so far, you'll be able to see a **"Hello World"** text being displayed on the UI.
- This means the server is rendering a response saying Hello World when the user requests for the home path of the website.

Serving static files with Express.js

CoGrammar

Serving static files

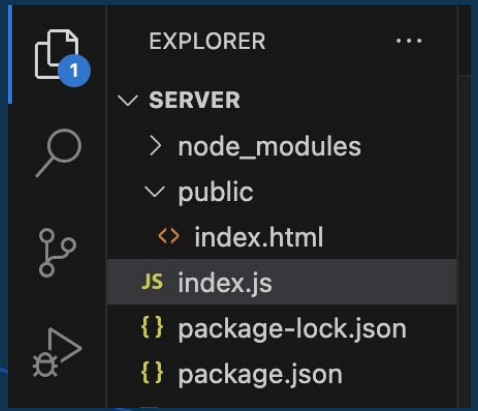
Rendering HTML, CSS or JS using express.js

- Static files like HTML, CSS, JavaScript, images, and other files that don't change dynamically, can be served by Express using a built in **middleware** (**express.static**).
- A common convention for creating the static files is having them in a directory called **public** (You can name it to any word).
- After creating the folder you simply call the static middleware with the name of the folder (in string format) as an argument.

Serving static files

Rendering HTML, CSS or JS using express.js

- From the code snippet, if you head over to **http://localhost:800/index.html**, you'll be able to access the HTML static file.



index.js

```
1  const express = require('express');
2  const app = express();
3
4  //Middleware to allow acces to static files
5  app.use(express.static('public'))
```

CRUD Operations with Express.js

CoGrammar

CRUD Operations

- CRUD operations are fundamental tasks when working with databases or managing resources.
- Here's an overview of how CRUD operations are implemented in Express.js and the respective description.

HTTP verb	CRUD operation	Express method	Description
Post	Create	`app.post()`	Used to submit some data about a specific entity to the server.
Get	Read	`app.get()`	Used to get a specific resource from the server.
Put	Update	`app.put()`	Used to update a piece of data about a specific object on the server.
Delete	Delete	`app.delete()`	Used to delete a specific object.

CRUD OPERATIONS

- For a start, we'll work with an in-memory array to act as our storage for a complete todo application having the CRUD functionalities.
- The code snippets on the next slides will show how you can perform the CRUD on the created array.

index.js

```
7   // Mock data (in-memory array)
8   let todos = [];
```

CoGrammar

CRUD Operations

- **C** - Create functionality, create a new todo item.

```
index.js

// Create (POST) a new todo
app.post('/todos', (req, res) => {
  const { title, description } = req.body;
  const todo = { id: todos.length + 1, title, description, completed: false };
  todos.push(todo);
  res.status(201).send(todo);
});
```

- **R**- Read functionality, returns all todo items

```
index.js

// Read (GET) all todos
app.get('/todos', (req, res) => {
  res.send(todos);
});
```

CoGrammar

CRUD Operations

- **U** - Update functionality, updates an existing todo item
- **D-** deletes an existing todo item

index.js

```
34  // Update (PUT) a todo by ID
35  app.put('/todos/:id', (req, res) => {
36    const id = parseInt(req.params.id);
37    const todoIndex = todos.findIndex(todo => todo.id === id);
38    if (todoIndex === -1) {
39      res.status(404).send('Todo not found');
40    } else {
41      todos[todoIndex] = { ...todos[todoIndex], ...req.body };
42      res.send(todos[todoIndex]);
43    }
44  });
```

index.js

```
46  // Delete (DELETE) a todo by ID
47  app.delete('/todos/:id', (req, res) => {
48    const id = parseInt(req.params.id);
49    const todoIndex = todos.findIndex(todo => todo.id === id);
50    if (todoIndex === -1) {
51      res.status(404).send('Todo not found');
52    } else {
53      const deletedTodo = todos.splice(todoIndex, 1);
54      res.send(deletedTodo[0]);
55    }
56  });
```

CRUD OPERATIONS

Passing data to the server using the request object.

- There are several ways of accepting data to the server from the user. This is made possible by utilizing the request **(req)** object.
- The **req** object is a mandatory parameter in the callback function of your request method. It has several properties like **body** and **params.**
- We access data passed to the body of the request using **req.body** (As observed from the POST/create method).
- We access data passed to the URL parameter of the request using **req.params** (As observed from the PUT/update method).

Testing the API Endpoints Created

CoGrammar

API testing

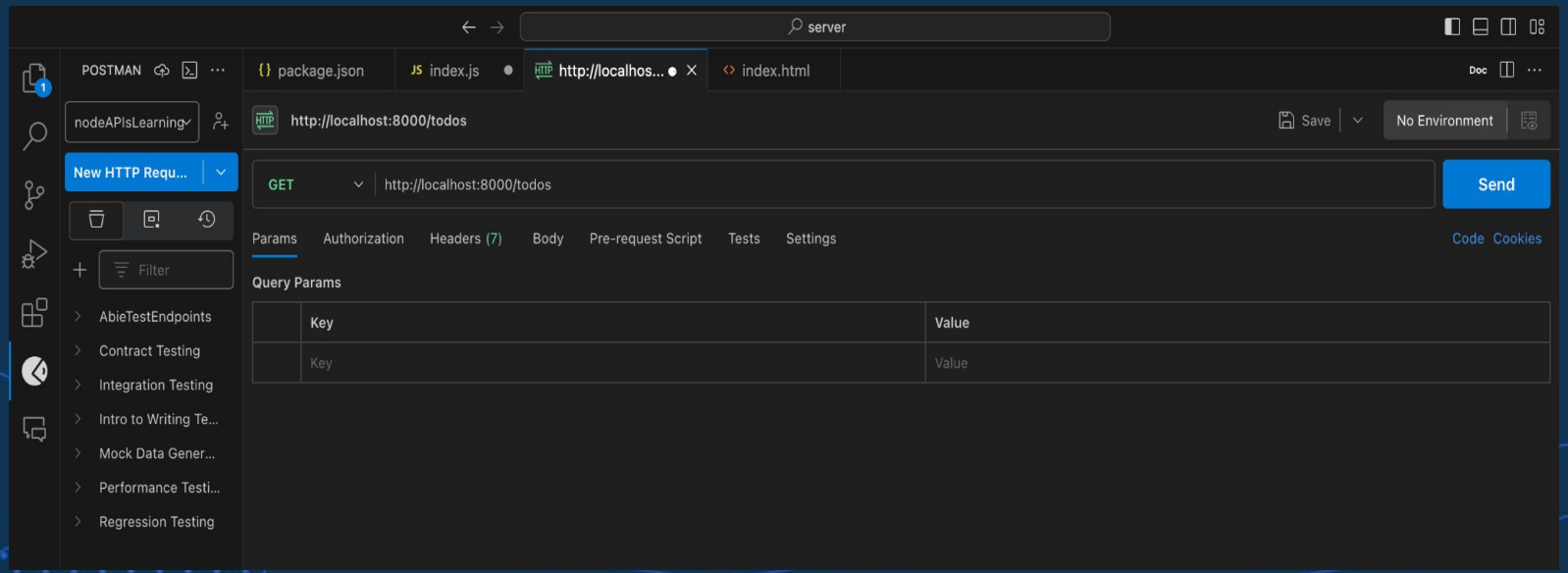
Using postman to access and send requests to the APIs created.

- After building your express endpoints, it is a best practice to test the APIs first before consuming them on the Frontend.
- We do this by use of tools like POSTMAN. Postman is an API platform for building and using APIs.
- After creating a postman account, you can install the application, use it on the browser or download it as a VS Code extension.
- The easiest way to get started with postman is by using the VSCode extension. You can get it by searching for postman in the extension marketplace.

API testing

Using postman to access and send requests to the APIs created.

- Testing the **/todos** endpoint as an example. Returns an empty list. You can make a POST request to the /todos and create a todo item.



CoGrammar

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar