Welcome to the CoGrammar Tutorial: Express.js and MongoDB

The session will start shortly...

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

CoGrammar

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**

CoGrammar

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

CoGrammar

SKILLS FOR LIFE
SKILLS BOOTCAMPS
Department for Education

CoGrammar

Express and MongoDB

May 2024

Express.js Recap

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.

CoGrammar

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

CoGrammar

Installation and Configuration

Setting up Express.js

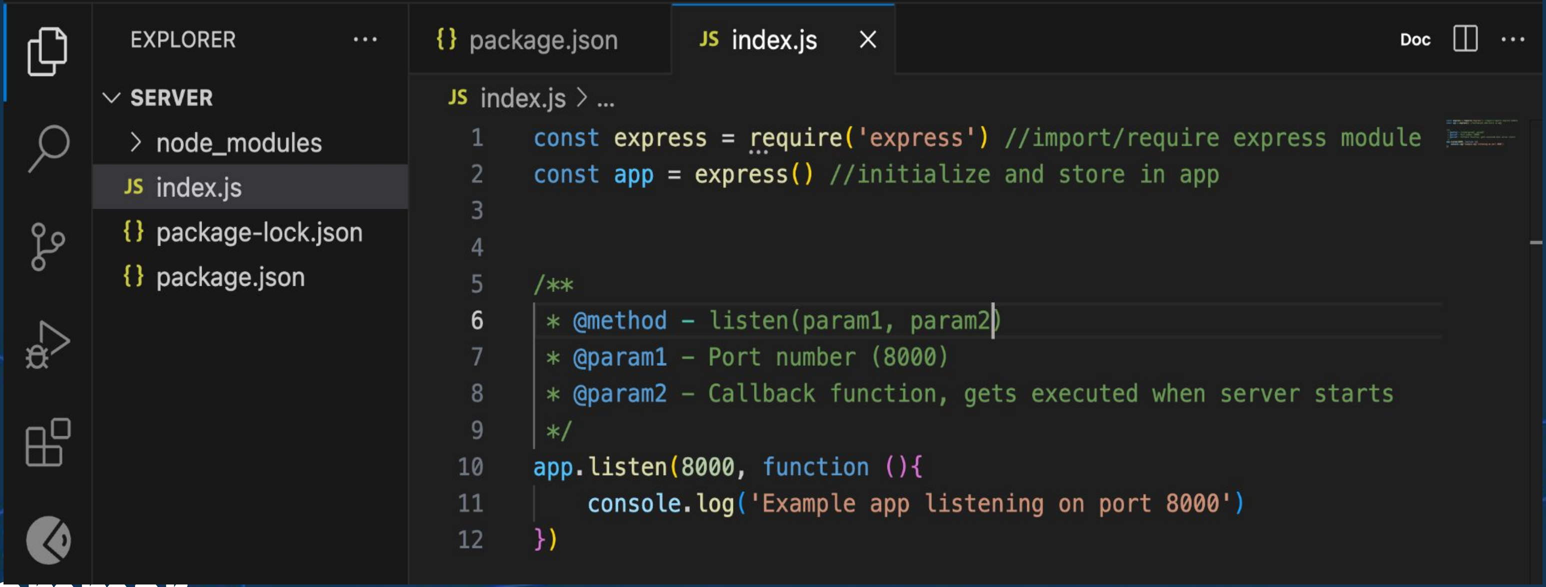
- Create a folder where your application will live and change directory to it:
  - mkdir server
  - cd server
- Initialize your package.json file with the default settings:
  - npm init -y (The y is optional if you need to skip prompts)
- Install express.js:
  - npm install express

CoGrammar

Creating a server

Running a port on your local machine

❖ 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.



```
const express = require('express') //import/require express module
const app = express() //initialize and store in app


/**
 * @method - listen(param1, param2)
 * @param1 - Port number (8000)
 * @param2 - Callback function, gets executed when server starts
 */
app.listen(8000, function (){
    console.log('Example app listening on port 8000')
})
```

CoGrammar

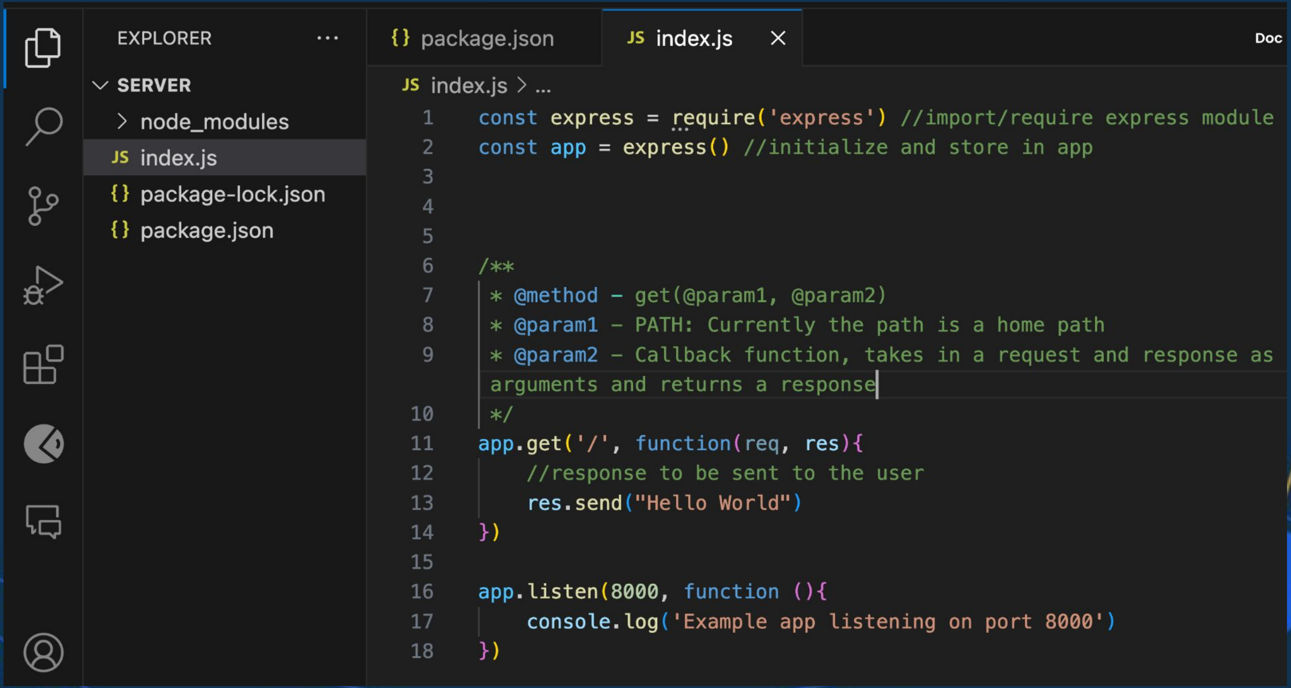
Creating a route for your application

- ❖ 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.

CoGrammar

Creating a route

Creating a home path



```javascript
const express = require('express') //import/require express module
const app = express() //initialize and store in app



/**
 * @method - get(@param1, @param2)
 * @param1 - PATH: Currently the path is a home path
 * @param2 - Callback function, takes in a request and response as arguments and returns a response
 */
app.get('/', function(req, res){
    //response to be sent to the user
    res.send("Hello World")
})

app.listen(8000, function (){
    console.log('Example app listening on port 8000')
})
```

CoGrammar

Creating a server

Adding a start script to the server

- 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:

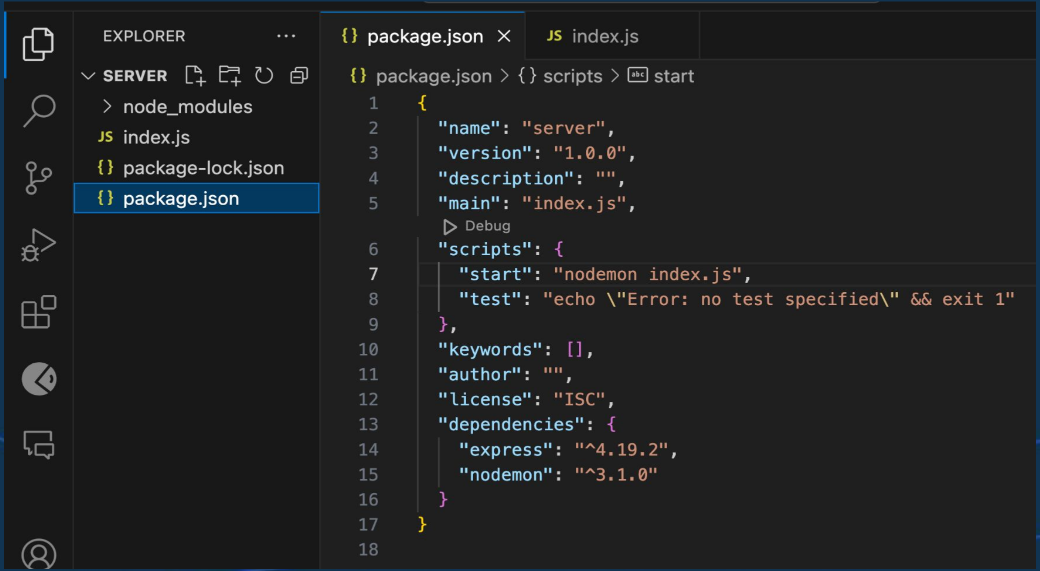
```
npm install nodemon
```

CoGrammar

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}**



```json
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.19.2",
    "nodemon": "^3.1.0"
  }
}
```

CoGrammar

Serving static files

Rendering HTML, CSS or JS using express.js

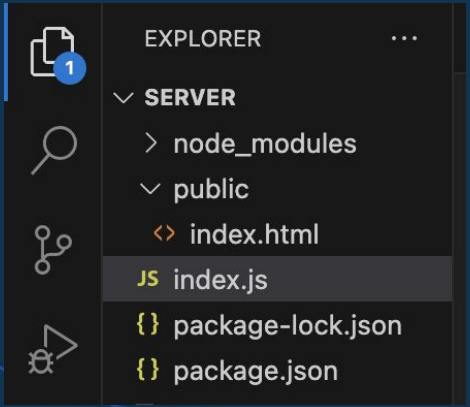
- Static files like HTML, CSS, JavaScript, images, and other files that don't change dynamically, can be served by Express using a built in **middleware** (**express.static**).
- A common convention for creating the static files is having them in a directory called **public** (You can name it to any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

CoGrammar

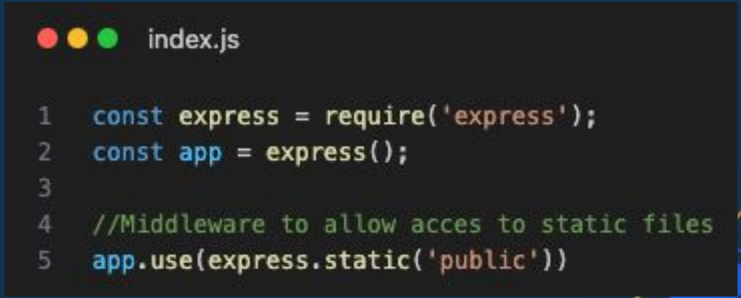
Serving static files

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.



```
EXPLORER
SERVER
  > node_modules
  v public
      index.html
    index.js
    package-lock.json
    package.json
```



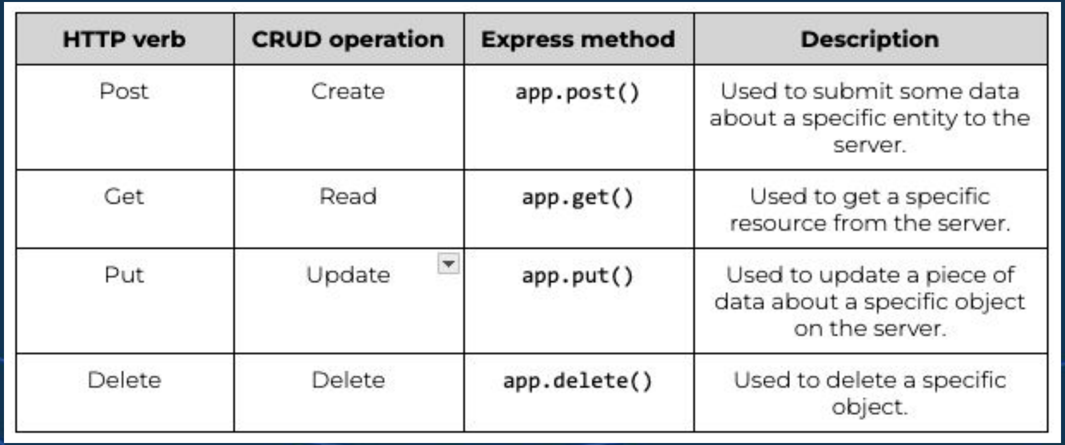
index.js

```javascript
const express = require('express');
const app = express();

//Middleware to allow acces to static files
app.use(express.static('public'))
```

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.

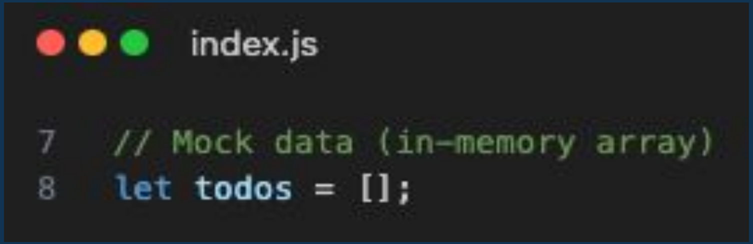


HTTP verb	CRUD operation	Express method	Description
Post	Create	`app.post()`	Used to submit some data about a specific entity to the server.
Get	Read	`app.get()`	Used to get a specific resource from the server.
Put	Update	`app.put()`	Used to update a piece of data about a specific object on the server.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

CoGrammar

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.

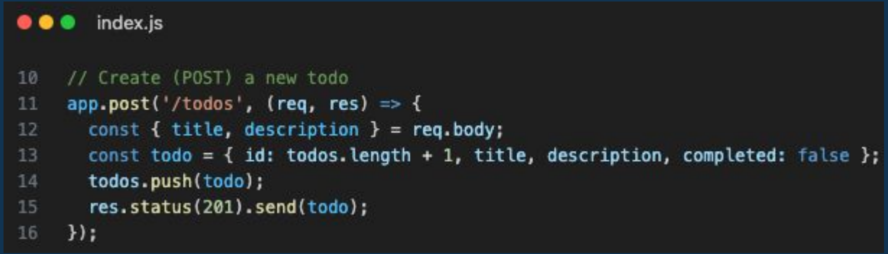


```javascript
// Mock data (in-memory array)
let todos = [];
```

CoGrammar

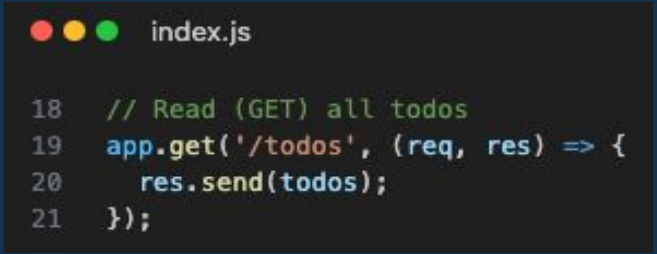
CRUD Operations

- **C** - Create functionality, create a new todo item.
- **R**- Read functionality, returns all todo items



index.js

```javascript
// 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);
});
```



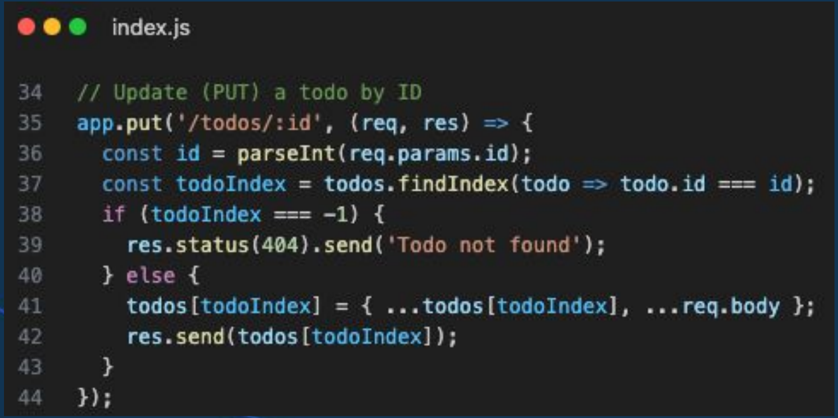
index.js

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

CoGrammar

CRUD Operations

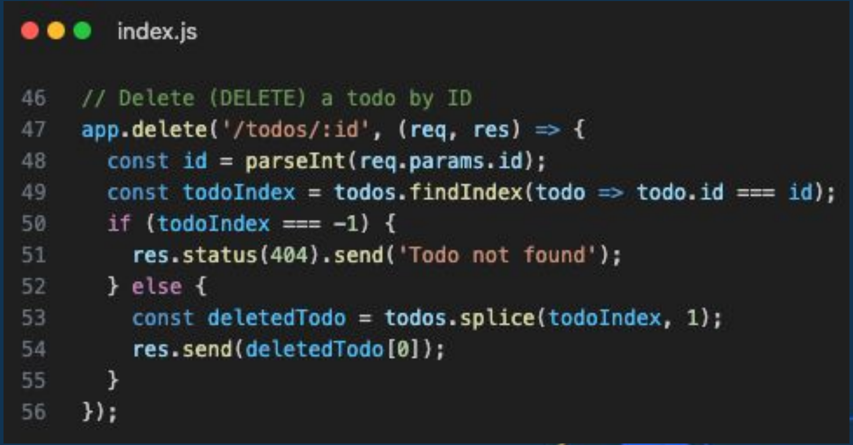
- **U** - Update functionality, updates an existing todo item



index.js

```javascript
// Update (PUT) a todo by ID
app.put('/todos/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const todoIndex = todos.findIndex(todo => todo.id === id);
  if (todoIndex === -1) {
    res.status(404).send('Todo not found');
  } else {
    todos[todoIndex] = { ...todos[todoIndex], ...req.body };
    res.send(todos[todoIndex]);
  }
});
```

- **D-** deletes an existing todo item



index.js

```javascript
// Delete (DELETE) a todo by ID
app.delete('/todos/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const todoIndex = todos.findIndex(todo => todo.id === id);
  if (todoIndex === -1) {
    res.status(404).send('Todo not found');
  } else {
    const deletedTodo = todos.splice(todoIndex, 1);
    res.send(deletedTodo[0]);
  }
});
```

CoGrammar

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).

CoGrammar

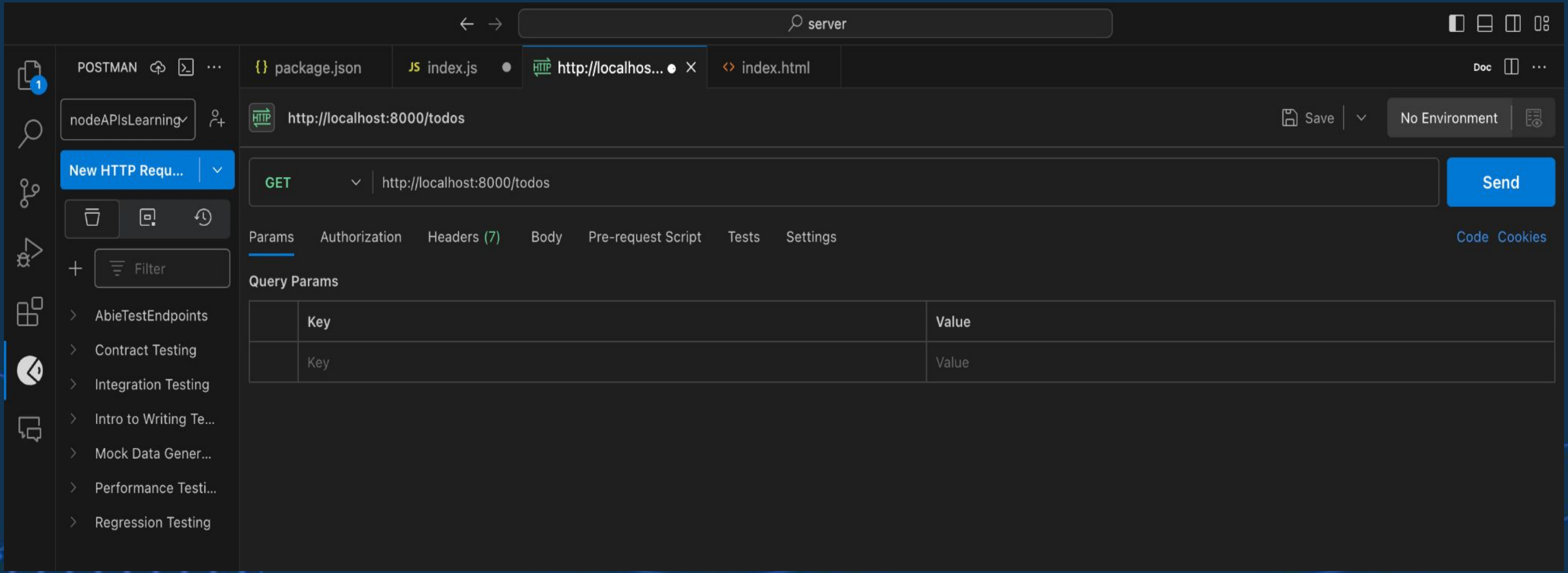
Testing the API Endpoints Created

CoGrammar

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.



http://localhost:8000/todos

GET http://localhost:8000/todos

Key	Value
Key	Value

CoGrammar

Databases

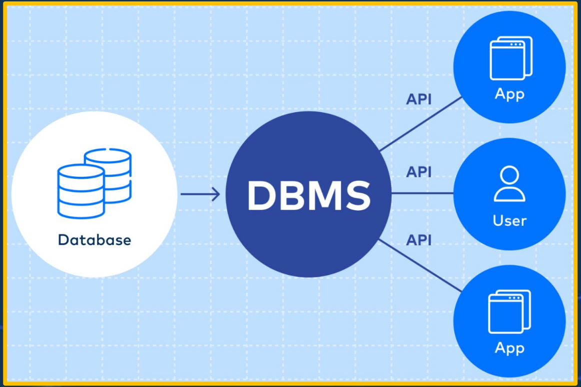
A large container of data with the ability to order the data in multiple ways, while providing access to the data itself.

- **Data** refers to **raw, unprocessed** facts. Once data has been processed, we call it **information**.
- The production of accurate, timely and relevant information is the key to good **decision-making**, which is the key to a **business' survival** in a competitive global environment.
- Timely and useful information requires accurate data, which must be captured properly and stored in a format that is easy to access and process

CoGrammar

DBMS

- A database is usually controlled by a database engine, commonly known as a **Database Management System (DBMS).**
- DBMSs serve as a **tool** between a user and their data, **organising** and **cataloging** the data for **quick and easy retrieval.**
- The data and the DBMS, and the applications associated with them are referred to as a **database system**, usually shortened to **database.**



CoGrammar

DBMS

- The **advantages** of the DBMS are:
  - **Data sharing:** Better access to more, better managed data across applications and users.
  - **Data integration:** Unified view of well-managed data combined from multiple sources.
  - **Data consistency:** Minimised risk of different versions of the same data stored in different places.
  - **Data access:** The DBMS makes it possible to produce quick answers to spur-of-the-moment requests for data.

CoGrammar

Types of Databases

Single/Multi-user Database	Refers to how many users can work on the database at the same time.
Enterprise Database	A multi-user database that supports more than 50 users and an entire organisation, across departments.
Centralised/Distributed Database	Refers to how many sites the database is distributed across.
Structured/Unstructured Database	Refers to whether data is stored in the form collected in or if it has been processed to facilitate operations.

CoGrammar

Relational Databases

Any database system that allows data to be associated and grouped by common attributes.

- ❖ Relational databases are comprised of a number of tables (**relations**), within each are:
  - ➢ Rows also known as records or tuples
  - ➢ Columns also known as attributes or fields
- ❖ Each record is identified with a **unique key**, known as the **primary key**.
- ❖ Records from one table can be references in other tables using their key, in this case they are called **foreign keys**.
- ❖ Each table/relation represents one **“entity type”**.

CoGrammar

NoSQL Databases

- ❖ The performance of relational databases degrades as the volume of data increases.
- ❖ Web applications usually have to store massive amounts of data, so NoSQL databases were developed to improve performance.
- ❖ NoSQL databases have the following characteristics:
  - ➢ Not based on the relational model.
  - ➢ Support distributed database architectures.
  - ➢ High scalability, high availability and fault tolerance.
  - ➢ Support large amounts of sparse data.
  - ➢ Geared toward performance rather than transactional consistency

Types of NoSQL DBs

Key-value store databases	Simplest form of the NoSQL DB. Every item is stored as a key and a value.
Column-oriented databases	A key is used to identify values but can identify multiple values instead of one.
Document-store databases	A key is used to identify a particular document (like XM, JSON, PDF, etc.)
Graph databases	Graph structure (nodes connected by links or edges) is used to store data.
Object-oriented databases	Combines OOP and database principles.

CoGrammar

MongoDB

A document store and NoSQL database, made up of collections and documents.

- Collections: A group of documents, similar to an entity or table in RDBs.
- Documents: Equivalent to a record in an RDB (or row in a RDB table).
- MongoDB uses **Binary JSON** (BSON) which uses JSON files and stores **type** information, which makes it **quicker** and **more efficient** to use.
- If a user wants to access, add, or change any information that needs to persist, they will need access to the MongoDB database.
- Clients interact with a web server that runs Node.js, which makes use of **MongoDB drivers** to communicate with MongoDB.

CoGrammar

Installation

Installing MongoDB to use Mongo and Atlas to host MongoDB on the cloud.

1. Install MongoDB's free Community Server.
2. Configure MongoDB Atlas:
   a. Enter your information here.
   b. On the Database Deployments page, click of Build a Database.
   c. Under 'Cloud provider and Region', select AWS and any free tier region.
   d. Under 'Cluster Tier', select the free M0 option.
   e. You can rename your cluster under 'Cluster Name'.
   f. Click 'Create' to create your cluster.
   g. Get the connection string to connect to the database server.

CoGrammar

Shell Commands

- `show dbs;`
  - List all the databases in your cluster.
- `use db_name;`
  - Select a database or create it if it does not exist.
- `show collections;`
  - Shows all the collections in the previously selected database.
- `db.dropDatabase();`
  - Deletes the selected database.

CoGrammar

Mongoose

A library that makes working with the MongoDB driver simpler.

1. Install Mongoose using NPM:
   a. `npm install mongoose`
2. Create a schema which outlines the data in our database and how it is organised and structured.
3. Create a controller file to perform data manipulation.
4. Connect to the database and execute operations.

CRUD Operations

Create, Read, Update and Delete

❖ These are the 4 basic operations which act as the **foundation** of any computer programming language.

❖ We need to understand CRUD in Mongoose to interact with databases.

1. **Create:** To add or insert collections or documents into it.
   a. `insertOne({document});`
   b. `insertMany([{document1}, {document2}]);`
2. **Read:** To retrieve or fetch documents from your collection.
   a. `find()`

CoGrammar

CRUD Operations

3. Update: To modify documents within a collection.

a. `updateOne({field}, { $set: {new_document}});`

b. `updateMany({field}, { $set: {new_document}});`

4. Delete: To remove or delete documents from a collection.

a. `deleteOne({field});`

b. `deleteMany({field});`

CoGrammar

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar