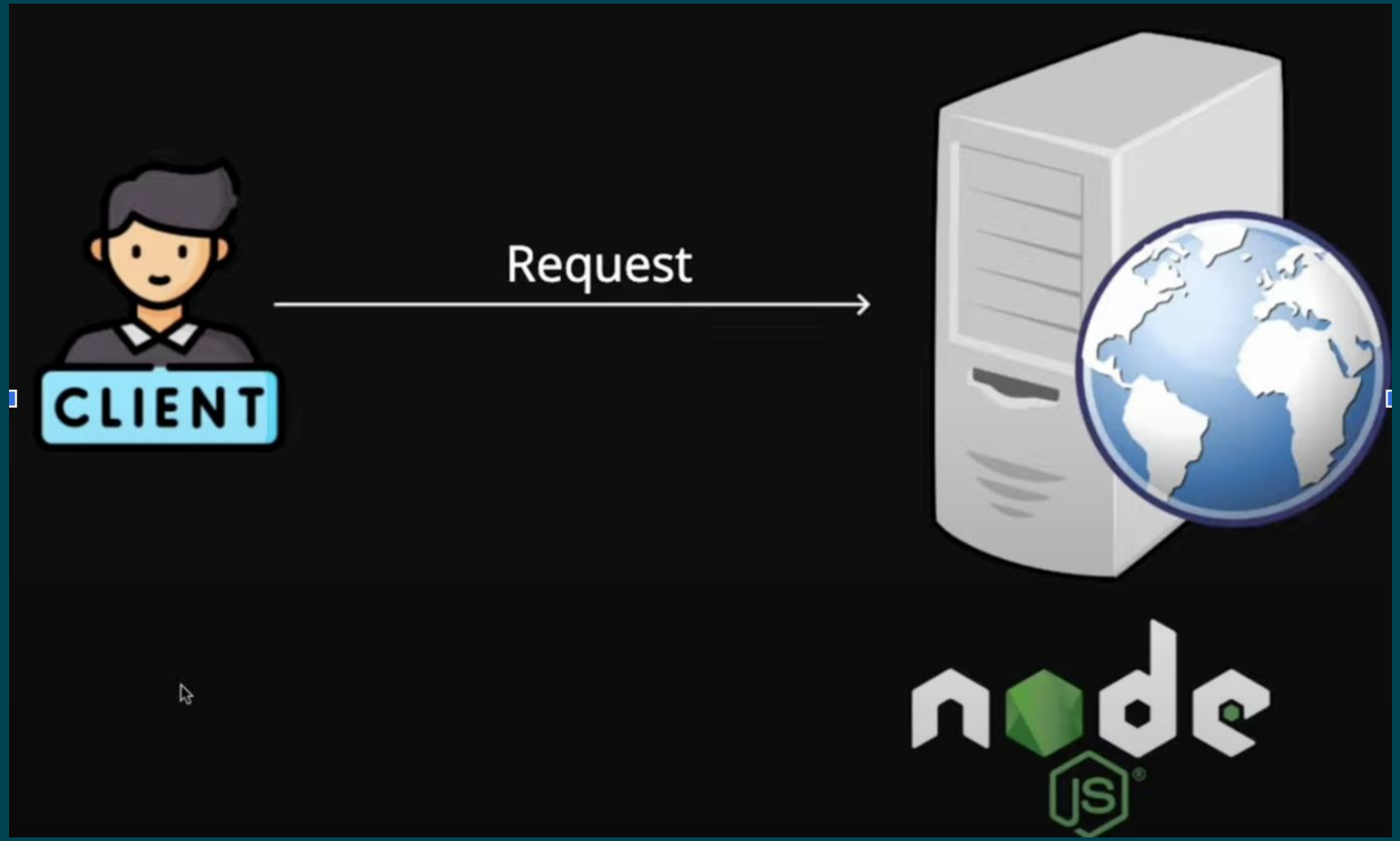
EXPRESS JS

By Gajanan Kharat - CODEMIND Technology

Contact us 8600 2550 66

CODEMIND
Your Success Partner

Client Server Architecture



What is Express JS ?

Express.js is a lightweight and flexible web framework for Node.js that helps you build web applications and APIs easily. It provides a set of tools and features that make it simpler to handle HTTP requests (like GET, POST, etc.), manage routes, and handle server-side logic.

Key Points:

- Simplifies Web Development: Express simplifies tasks like routing, handling requests, and managing middleware in Node.js.
- Flexible: You can add extra functionality with middleware and build both small and large applications.
- Minimalistic: It provides just the basic tools, so you have the flexibility to structure your application however you like.

Express application creation

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

const app = express();
app.get('/home', (req, res) => {
    return res.send('Hello from Home Page');
});
app.listen(8080, () => {
    console.log('Server Started...');
});
```

Import express module

Create an application

Handler with GET method and path is /home

Express sets up a simple server that listens on port 3000

In the browser enter the url → localhost:8080/home

Routing in Express JS

In Express.js, routing refers to how an application responds to client requests to specific endpoints (URLs). These endpoints could be different HTTP methods like GET, POST, PUT, and DELETE, which represent different operations. The syntax for routing is simple and intuitive.

Basic Routing Syntax:

```
app.METHOD(PATH, HANDLER)
```

- app: The Express application object.
- METHOD: The HTTP method (GET, POST, PUT, DELETE, etc.).
- PATH: The endpoint/URL path (e.g., /home, /about).
- HANDLER: The function that gets executed when the route is matched. It usually takes req (request) and res (response) objects as parameters.

Example of Simple Routing

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

// Basic GET route at the root URL ('/')
app.get('/', (req, res) => {
    res.send('Welcome to the homepage!');
});

// GET route at '/about'
app.get('/about', (req, res) => {
    res.send('This is the about page.');
});

// POST route at '/submit'
app.post('/submit', (req, res) => {
    res.send('Form submitted successfully!');
});
```

```
// PUT route at '/update'
app.put('/update', (req, res) => {
    res.send('Update was successful!');
});

// DELETE route at '/delete'
app.delete('/delete', (req, res) => {
    res.send('Item deleted successfully!');
});

// Start the Express server on port 3000
app.listen(3000, () => {
    console.log('Server is running on http://localhost:3000');
});
```

Http-status-codes → Dependency for Status Codes

→ npm install http-status-codes

We can use this dependency as well, if we want to use the predefined status codes

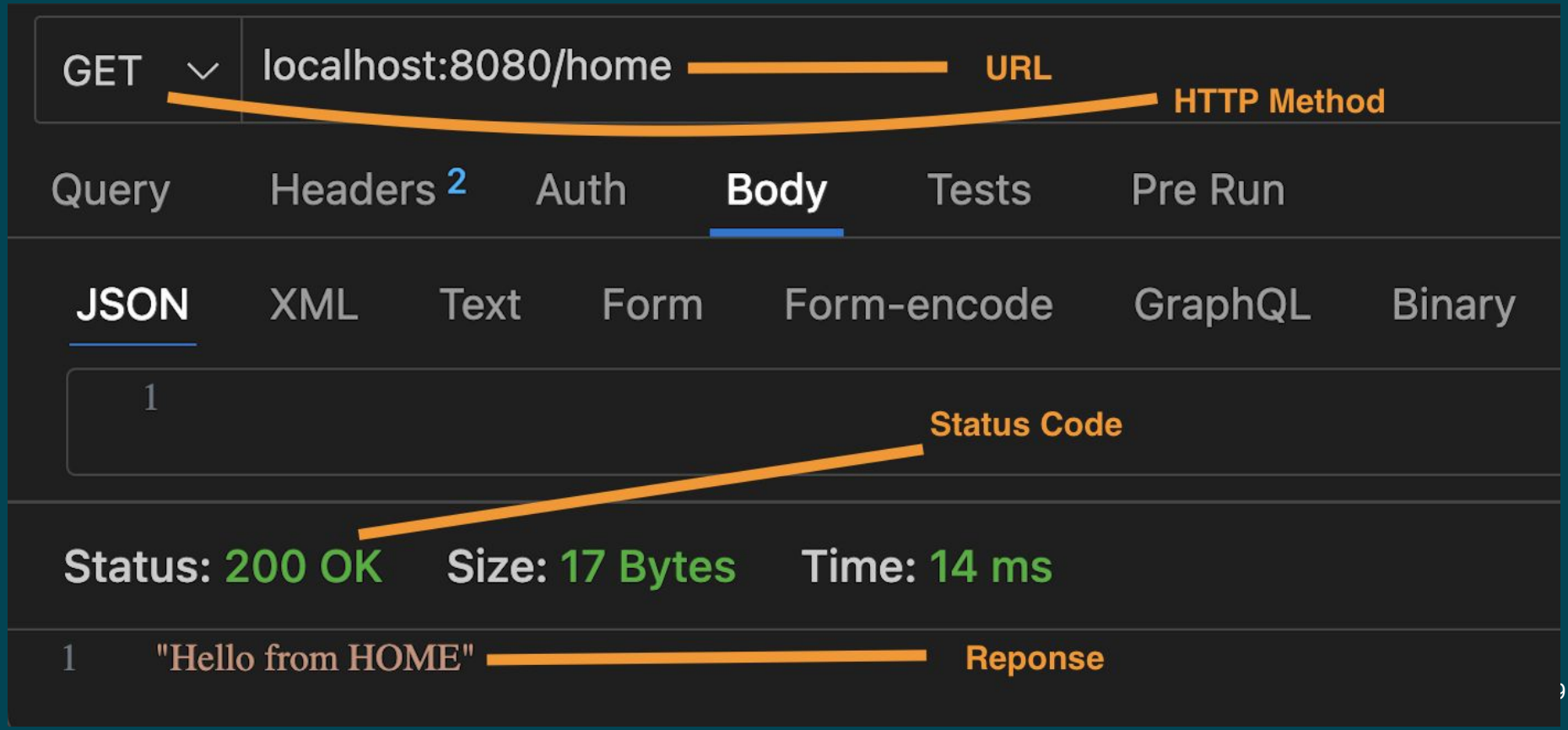
Return the response with status code and in json format

JS crud-app.js U × TC New Request *

JS crud-app.js > app.get("/home") callback

```js
const express = require("express");
const app = express();
app.listen(8080, () => {
  console.log("Server Started..");
});
app.get("/home", (req, res) => {
  return res.status(200)
            .json("Hello from HOME");
});
```

Create the request using thunder client



mongosh

Enter the command → mongosh

- mongosh is the MongoDB Shell command. It provides an interactive command-line interface for interacting with your MongoDB databases.
- ‘mongosh’ command will also give us the complete connection details.

```
Last login: Tue Sep 17 22:42:39 on console
gajanan@192 ~ % mongosh
Current Mongosh Log ID: 66efc27f52e512ac0bbe3fca
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&s
Using MongoDB:          7.0.2
Using Mongosh:          2.3.0

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/
```

mongoose

Mongoose is a tool that helps you work with MongoDB easily in Node.js. It acts as a bridge between your Node.js application and the MongoDB database.

Install the mongoose dependency

```
npm install mongoose
```

Steps to Store, Read, Delete and Update the data in Database

1. Establish the MongoDB Connection: Use Mongoose to connect to your MongoDB database.
2. Define the Schema: Define the structure of documents using Mongoose schemas.
3. Create the Model: Create a model from the schema to interact with the MongoDB collection.
4. Perform CRUD Operations: Use the model to perform Create, Read, Update, and Delete operations.

Mongodb connection string format without authentication

Syntax → protocol://host:port/database_name

Example: mongodb://localhost:27017/student_db

Establish connection with mongodb

- Install the mongoose dependency
- use the connect( ) to establish the connection. Which return the promises
- Handle this promises using then( ) and catch( ) methods approach

JS crud-app-express.js > ...

```
3   const mongoose = require("mongoose");
4   mongoose.connect("mongodb://localhost:27017/student_db")
5     .then(() => {
6       console.log("Connection is established successfully..");
7     })
8     .catch((error) => {
9       console.log("Failed to connect:", error);
10    });
11
```

Require mongoose dependency

mongodb protocol

database name

Host

Port number

Async and await approach to handle the promises return by connect( )

JS crud-app-express.js > ...

```
3   const mongoose = require("mongoose");
4   async function mongoConnection() {
5     try {
6       await mongoose.connect("mongodb://localhost:27017/student_db");
7       console.log("Connection established successfully..");
8     } catch (error) {
9       console.log(error);
10    }
11  }
12  mongoConnection();
```

Define Schema and Model in Mongo Database

Model: A Model is like a blueprint for creating, reading, updating, and deleting documents (data) in a specific collection of your MongoDB database.

- Model Provides methods to interact with the MongoDB collection that stores the documents following the schema.
- Using Model we do all the CRUD operations.
- Provides methods to interact with the MongoDB collection that stores the documents following the schema.

Schema: Defines the structure of the documents.

Define the Schema and Model

```
const studentSchema = new mongoose.Schema({
    rollNo: { type: Number },
    name: { type: String },
    city: { type: String },
    graduation: { type: String },
},
  { collection: "student" },
  { timestamps: true }
);

const Student = mongoose.model("Student", studentSchema);
```

Define Schema

Defining Collection name explicitly

To preserve timestamps while insertion and update

Creating Model --> Students

Model Methods to read data from mongodb provided by mongoose

1. Read (Retrieve Data)
   a. find({ }) → Retrieves all documents or students
   b. findOne({ rollNo: 11 }); → Retrieves only one student whose rollNo is 11
   c. findById("60e2ee10344b56043c9e3b5f") → Find By ID
   d. countDocuments({ graduation: "BSc" }) → Returns the count of documents that match the filter criteria.

Note: All Model method return the Promise

find({ }) Model Method to read all students

```
app.get("/students", async (request, response) => {
  console.log("Fetching all Students");
  const studentsAll = await Student.find({});
  return response.status(200).json(studentsAll);
});
```

find({ }) Method to find all students

Assignment. File Name → crud-adhar.js

Database Name: adhar_database

Collection Name: adhar_collection

Fields:

- adharNo → Number
- fullName → String
- isMarried → Boolean
- city → String
- pin → Number
- country → String

Note: Manually insert minum 5 documents using mongodb compass.

Retrieve all these documents and show in the API as response (Thunder client)

Parse request body to json format

express.json( ): This middleware parses incoming requests with JSON payloads

```
const app = express();

// Use express.json() to parse JSON request bodies
app.use(express.json())
```

Save or store the student data received in request

Steps:

- Get the request body using → request.body
- Extract the student data like rollNo, name, city and graduation from the body
- Create the new student document using Model
- Save the new student to the database

Post method : Store the new student object

```
app.post("/create", async (request, response) => {
    const { rollNo, name, city, graduation } = request.body;
    const student = await Student.create( {
      rollNo,
      name,
      city,
      graduation
});
    return response.status(200).json(student);
});
```

Extract data from request body using object destructuring

Creates a new document and saves it directly to the database in one step.

Model methods to save data in the database

insertMany( ): Inserts multiple documents at once into the database.

```
const students = await Student.insertMany([
  { rollNo: 103, name: 'Alice', city: 'Los Angeles', graduation: 'BA' },
  { rollNo: 104, name: 'Bob', city: 'Boston', graduation: 'MA' }
]);
```

Put method: Update the student data

findOneAndUpdate( ): Finds a document by a condition, updates it, and saves the changes.

```
app.put("/update", async (request, response) => {
    const { rollNo, name, city, graduation } = request.body;
   const student = await Student.findOneAndUpdate(
     { rollNo: 22 }, // Find condition
     { name: name, city: city }, // Update fields
     { new: true, upsert: true }
   );
    return response.status(200).json(student);
});
```

upsert: true → upsert stands for "update" + "insert". This option tells Mongoose to create a new document if one does not already exist that matches the query criteria

new: true → This option tells Mongoose to return the updated document instead of the original one before the update

URL Path parameter: Get the student using rollNo field

A URL path parameter is a part of the URL that is used to pass information to the backend server. It helps identify specific resources or data when making requests.

findOne({ rollNo: rollNo }): This queries the MongoDB collection to find a document where the rollNo matches the one provided. It returns the first matching document

GET http://localhost:8080/student/33

URL with path parameter and GET method

This captures the roll number from the URL path parameter

```
app.get('/student/:rollNumber', async (request, response) => {
    const rollNumber = request.params.rollNumber;
    const student = await Student.findOne({rollNo: rollNumber})
    return response.status(200).json(student);
});
```

Delete method→ findOneAndDelete( ): This method will find the document based on the given roll number and delete it from the collection.

```
app.delete('/delete/:rollNumber', async (request, response) => {
    const rollNumber = request.params.rollNumber;
    const student = await Student.findOneAndDelete({rollNo: rollNumber});
    if (!student) {
        return response.status(400).json("Student Not Found");
    }
    return response.status(200).json("Student Deleted Successfully");
});
```

Assignment

1. Repository name: crud-product
2. Create the product → Laptop, Desktop, Pen, Notebook, Bag
3. Read the product using Thunder client
4. Update any one product
5. Delete any one product

Database name: product-database

Collection Name: product-collection

Thank You

Success is not a milestone, it's a journey. And we have vowed to help you in yours.

CODEMIND
Your Success Partner

www.codemindtechnology.com