SIES GST, NERUL, NAVI MUMBAI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (INTERNET OF THINGS AND CYBER SECURITY INCLUDING BLOCK CHAIN TECHNOLOGY)

LAB MANUAL

WEB X.0 LAB

(IOTCSBCC604)

T.E.

SEMESTER-VI
(R-2019)-Ver1

Branch: Computer Science& Engineering (IOT & CSIBCT)

DEPARTMENT'S VISION

To be a centre of excellence in IoT, Cybersecurity and Blockchain Technologies to cater growing demands of society.

DEPARTMENT'S MISSION

1. To Impart quality education to satisfy the advanced requirements in the fields of IoT, Cybersecurity and Block chain.
2. To provide a platform for developing research, professional, social and innovative skills.
3. To develop entrepreneurial, technical and life-long learning skills for socio-economic growth.
4. To strengthen the alumni and industrial association for development of students leading to technical and socio-economic growth.

PROGRAMME EDUCATIONAL OBJECTIVES(PEOs)

- PEO1: Develop leadership abilities for professional advancement.
- PEO2: Pursue higher education to learn and advance their careers.
- PEO3: Promote services in the field of Computer Science and Engineering through excellent technical and communication skills.

PROGRAMME OUTCOMES(POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics science engineering fundamentals and an mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual and as a member or leader in diverse teams and individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write

effective reports and design documentation, and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding ofthe engineering and knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: long learning: Recognize the need for and have the Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

PROGRAMME SPECIFIC OUTCOMS (PSOs)

- To work in multidisciplinary domains associated with IoT, Cyber security and Blockchain technologies.
- To solve real life problems related to IoT and Cyber Security.

GENERAL INSTRUCTIONS(Do's And Dont's)

1. Wearing ID-Card is compulsory.
2. Keep your bag at the specified place.
3. Shut down the system after use.
4. Place the chairs in proper position before leaving the laboratory.
5. Report failure/Non-working of equipment to Faculty In-charge / Technical Support staff immediately.
6. Know the location of the fire extinguisher and the FIRST-AID Box and how to use then in case of an emergency.
7. Do not eat or drink in the laboratory.
8. Do not litter in the laboratory.
9. Avoid stepping on electrical wires or any other computer cables.

10. Do not open the system unit casing or monitor casing particularly when the power in turn
11. Do not insert metal objects such as clips, pins and needles into the computer casing. They may cause fire.
12. Do not remove anything from laboratory without permissions.
13. Do not touch, connect or disconnect any plug or cable without permission.

LAB OUTCOMES (LO)

Sr.No.	Lab Outcomes	Blooms' taxonomy level (L1-L6)
1.	Understand open source tools for web analytics and semantic web apps development and deployment.	L1, L2
2.	Understand the basic concepts of TypeScript for designing web applications.	L1, L2, L3
3.	Construct back-end applications using Node.js	L1, L2,L3
4.	Construct back end applications using Express	L1, L2,L3
5.	Implement Single Page Applications using AngularJS Framework	L1, L2,L3
6.	Develop REST web services using MongoDB	L1, L2,L3

LABARTICULATIONMATRIX(MAPPINGWITHPO& PSO)

Course Outcomes / Program Outcomes	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**PSO1**	**PSO2**
L01	3	3			3									2
L02	3	3	3		3									2
LO3	3	3	3		3									2
LO4	3	3	3		3									2
LO5	3	3	3		3									2
LO6	3	3	3		3									2

Laboratory Assessment

Academicv Year: 2025-2026 **Class/Sem:** TE/VI **Div:** H **Batch**: H2
Name: Hrishikesh Garige **Roll No:** 122AX024
Course Name: Web Lab **CourseCode**: IoTCSBCL604

1. Preparedness and Efforts/ Preparation and Knowledge		
3: Well prepared and puts efforts.	2: Not prepared but puts efforts or prepared but doesn't put efforts	1: Neither prepared nor puts efforts
2. Presentation of output/Accuracy and Neatness of Documentation		
3: Uses all perfect Instructions /Interrupts/Presented well.	2: Uses some perfect Instructions/Interrupts/ moderate presentation.	1: Doesn't use any of the proper Instruction/Interrupt/Not presented properly.
3. Results/Participation in Practical Performance		
3: Participate and gets proper results	2: Participate but doesn't get proper result or gets result but with the help of faculty in-charge	1:Neither Participate nor gets the results
4. Punctuality		
3: Get the experiment checked in-time and is always in-time to the lab sessions	2: Some time delays the experiment checking or is late to the lab sessions for few times	1: Most of the time delays experiment checking and / or comes late for lab sessions
5. Lab Ethics		
3: Follows proper lab ethics by keeping the lab clean and placing things at their right place	2: Sometimes doesn't follow the lab ethics	1: Most of the times makes the lab untidy and keeps things at wrong place

EVALUATION:

PerformanceIndicator/ Expt.No	1	2	3	4	5	6	7	8	9	10	Total
	CO1, 2	CO1, 2	CO1 ,2	CO1 ,2	CO1 ,2	CO1, 2	CO1, 2	CO1, 2	CO1, 2	CO1 ,2	
1. Preparedness and Efforts/ Preparation and Knowledge	3	3	3	3	3	3	3	3	3	3	30
2. Presentation of output/ Accuracy and Neatness of Documentation	3	3	3	3	3	3	3	3	3	3	30
3. Debugging and results/Participation in Practical Performance	3	3	3	3	2	3	3	2	3	2	27
4. Punctuality	3	3	3	3	3	3	3	3	3	3	30
5. Lab Ethics	3	3	3	3	3	3	3	3	3	3	30
Total	15	15	15	15	14	15	15	14	15	14	147
Average	3	3	3	3	3	3	3	3	3	3	

Exceed Expectations(3), Meet Expectations(2), Below Expectations(1)

Faculty In-charge: Prof.Swapnil B Wani

GRADUATE SCHOOL OF TECHNOLOGY

NERUL, NAVI MUMBAI.

DEPARTMENT OF CSE(IOT)

SEM: - VI **BRANCH:- CSE(IOT)**

Web X.0 Lab

LIST OF EXPERIMENTS

No	Name of Experiment
1	Study web analytics using open source tools like Matomo, Open Web Analytics, AWStats, Countly, Plausible.
2	Small code snippets for programs like Hello World, Calculator using TypeScript.
3	Inheritance example using TypeScript.
4	Build Hello World App in Node.js.
5	Stream and Buffer in Node.js.
6	Configuring Express Settings and creating Express application using request and response objects.
7	Build Express application by Sending and Receiving Cookies.
8	Create a simple HTML “Hello World” Project using AngularJS Framework and apply ng-controller, ngmodel,expression and filters.
9	Create an application for like Students Record using AngularJS.
10	Connect MongoDB withNode.js and perform CRUD operations.
11	Build a RESTful API using MongoDB.

Name: Hrishikesh Garige

PRN: 122AX024
Branch: CSE(IOT)
Batch: H2

Experiment No. 1

Aim: Case Study Semantic Web Open Source Tools like Apache Jena.

Theory:
Apache Jena Case Study: A Comprehensive Analysis

Introduction: Apache Jena is a powerful open-source Java framework for building Semantic Web and Linked Data applications. It offers a comprehensive suite of tools for working with RDF (Resource Description Framework) data, SPARQL querying, ontology management, and reasoning over data.

Background: Apache Jena originated as a part of HP Labs and was later contributed to the Apache Software Foundation, ensuring its development and maintenance under an open-source license. Jena is widely used in academic research, data integration, and enterprise-level knowledge graph applications.

Key Features:

1. **RDF Support:**
    - Provides an API for creating, reading, and manipulating RDF data.
    - RDF/XML, Turtle, N-Triples, and JSON-LD serialization formats supported.
2. **SPARQL Support:**
    - Implements SPARQL, a powerful query language for RDF data.
    - Offers a SPARQL server (Fuseki) for querying RDF datasets over HTTP.
3. **Ontology Management:**
    - Supports OWL (Web Ontology Language) for complex data modeling.
    - Built-in tools for inference and reasoning over RDF graphs.
4. **Inference and Reasoning:**
    - Rule-based reasoning for both forward and backward chaining.
    - Built-in inference engines for RDFS and OWL reasoning.
5. **Data Storage and Integration:**
    - TDB: Native, high-performance triple store.
    - TDB2: Advanced storage system with improved scalability.
    - Fuseki: SPARQL server for web-based RDF querying and management.

Case Study: Implementing a Knowledge Graph for Healthcare

Objective: To create a knowledge graph integrating patient data, research articles, and disease ontologies to facilitate better decision-making in healthcare.

Implementation Steps:

1. **Data Collection:**
    - Patient records in RDF format.
    - Disease ontologies (e.g., SNOMED CT, ICD-10) integrated using OWL.
    - Research papers annotated using RDF triples.
2. **Data Storage:**
    - TDB2 was selected for local storage of RDF datasets due to its high performance and scalability.
3. **Data Integration:**
    - RDF datasets were merged using Apache Jena's RDF API.
4. **Querying and Reasoning:**
    - SPARQL queries were used for information retrieval.
    - Reasoning was applied to detect relationships between symptoms and diseases.
5. **Visualization:**
    - Fuseki was used to provide a web interface for executing SPARQL queries.

Results:

- Improved data interoperability between patient records and disease ontologies.
- Enhanced decision-making through query-based insights.
- Automated detection of potential health risks using reasoning capabilities.

Advantages of Apache Jena:

- Open-source and community-driven.
- Scalable and suitable for enterprise-level data integration.
- Strong support for RDF, SPARQL, and OWL standards.

Challenges:

- Steep learning curve for beginners.
- Java dependency might be a limitation for non-Java environments.

Conclusion: We have done the Case Study Semantic Web Open Source Tools like Apache Jena.

Name: Hrishikesh Garige
PRN: 122AX024
Branch: CSE(IOT)
Batch: H2

Experiment No.2

Aim: To execute small code snippets for programs like Hello

World, Calculator using TypeScript.

Thoery:

1. Install Node.js and TypeScript

1. **Install Node.js**
   Download and install Node.js. This will also install npm (Node Package Manager).
2. **Install TypeScript**
   Open a terminal and run:
3. npm install -g typescript

   This will install TypeScript globally on your system.

```
npm install -g typescript
```

2. Write Your TypeScript Code

1. Save the code in a file with a .ts extension. For example, program.ts.

3. Compile the TypeScript File

TypeScript code needs to be compiled into JavaScript before it can be run.

1. Open a terminal and navigate to the directory where your .ts file is located.
2. Compile the file using:
3. tsc program.ts

   This will generate a program.js file in the same directory.

4. Run the Compiled JavaScript File

Run the generated program.js file using Node.js:

node program.js

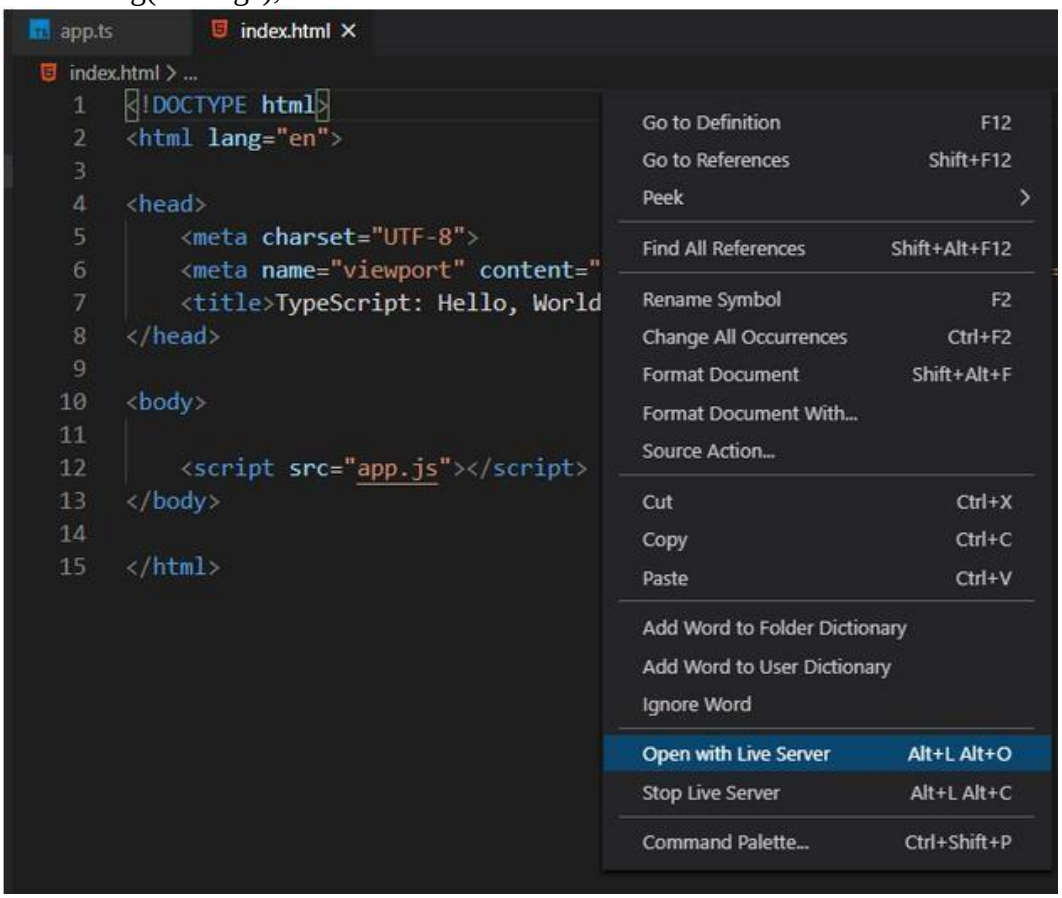
Example

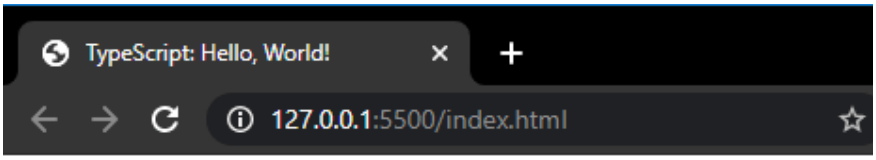
For the Hello World program:

1. Save this code in hello.ts:
2. const message: string = "Hello, World!";
3. console.log(message);
4. Compile the file:
5. tsc hello.ts
6. Run the compiled file:
7. node hello.js

1. Hello World Program

```
const message: string = "Hello, World!";
console.log(message);
```





Hello, World!

2. Simple Calculator

```
function calculator(a: number, b: number, operator: string): number | string {
    switch (operator) {
        case '+':
            return a + b;
        case '-':
```

```
            return a - b;
        case '*':
            return a * b;
        case '/':
            return b !== 0 ? a / b : "Cannot divide by zero";
        default:
            return "Invalid operator";
    }
}
const num1: number = 10;
const num2: number = 5;
const op: string = '+';
console.log(`Result: ${calculator(num1, num2, op)}`);
```

HelloWorld.ts 43asjx2jd

```
class Calculator {
    add(num1: number, num2: number): number {
        return num1 + num2;
    }

    subtract(num1: number, num2: number): number {
        return num1 - num2;
    }

    multiply(num1: number, num2: number): number {
        return num1 * num2;
    }

    divide(num1: number, num2: number): number | string {
        if (num2 === 0) {
            return "Cannot divide by zero";
        }
        return num1 / num2;
    }
}

// Example usage
const calculator = new Calculator();

console.log("Addition: ", calculator.add(10, 5));
console.log("Subtraction: ", calculator.subtract(10, 5));
console.log("Multiplication: ", calculator.multiply(10, 5));
console.log("Division: ", calculator.divide(10, 5));
console.log("Division by zero: ", calculator.divide(10, 0));
```

Output:

```
Addition:  15
Subtraction:  5
Multiplication:  50
Division:  2
Division by zero:  Cannot divide by zero
```

3. Factorial of a Number

```
function factorial(n: number): number {
    if (n < 0) return -1; // Error for negative numbers
    return n === 0 || n === 1 ? 1 : n * factorial(n - 1);
}
const num: number = 5;
console.log(`Factorial of ${num}: ${factorial(num)}`);
```

HelloWorld.ts 43asjx2jd

```
function factorial(n: number): number {
    if (n === 0 || n === 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}

// Function to get user input
function getUserInput(): number {
    const readline = require('readline').createInterface({
        input: process.stdin,
        output: process.stdout
    });

    return new Promise((resolve) => {
        readline.question('Enter a number to calculate its factorial: ', (input) => {
            const num = parseInt(input);
            if (isNaN(num)) {
                console.log("Invalid input. Please enter a number.");
                resolve(getUserInput()); // Recursively ask for input if it's not a number
            } else if (num < 0) {
                console.log("Factorial is not defined for negative numbers.");
                resolve(getUserInput()); // Recursively ask for input if it's negative
            } else {
                readline.close();
                resolve(num);
            }
        });
    });
}

// Main function
async function main() {
    const number = await getUserInput();
    const result = factorial(number);
    console.log(`The factorial of ${number} is: ${result}`);
}

main().catch(console.error);
```

4. Check if a Number is Even or Odd

```
function isEvenOrOdd(n: number): string {
   return n % 2 === 0 ? "Even" : "Odd";
}
const num: number = 7;
console.log(`${num} is ${isEvenOrOdd(num)}`);
```

```
function isEvenOrOdd(num: number): string {
    if (num % 2 === 0) {
        return `${num} is even`;
    } else {
        return `${num} is odd`;
    }
}

// Example usage
console.log(isEvenOrOdd(10)); // Output: 10 is even
console.log(isEvenOrOdd(11)); // Output: 11 is odd
```

Output:

```
10 is even
11 is odd
```

Conclusion: We have executed small code snippets for programs like Hello World, Calculator using TypeScript.

Name: Hrishikesh Garige
PRN: 122AX024
Branch: CSE(IOT)
Batch: H2

Experiment No 3

Aim: Inheritance example using TypeScript.

Theory:

Inheritance is a fundamental concept in object-oriented programming that allows a class to inherit properties and methods from another class. This promotes code reusability and establishes a hierarchy between classes. TypeScript, being a superset of JavaScript, supports classical inheritance, allowing developers to create a new class (subclass or derived class) based on an existing class (superclass or base class).

Key Concepts of Inheritance

1. **Base Class (Superclass)**: The class that is being inherited from. It contains properties and methods that can be shared with other classes.

2. **Derived Class (Subclass)**: The class that inherits from the base class. It can have additional properties and methods, as well as override existing ones.

3. **extends Keyword**: In TypeScript, the extends keyword is used to create a subclass that inherits from a base class.

4. **super Keyword**: This keyword is used to call the constructor of the base class and access its properties and methods from the derived class.

5. **Method Overriding**: A subclass can provide a specific implementation of a method that is already defined in its superclass.

Benefits of Inheritance

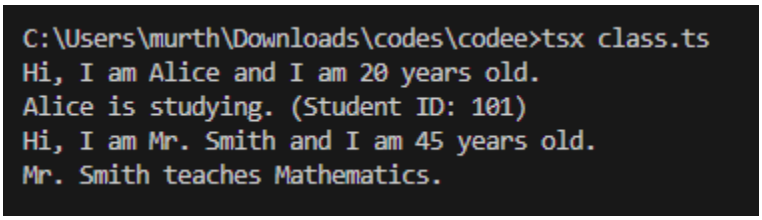
- **Code Reusability**: Common functionality can be defined in a base class and reused in derived classes.
- **Modularity**: Classes can be organized into a hierarchy, making the codebase easier to understand and maintain.
- **Extensibility**: New functionality can be added with minimal changes to existing code.

```
// Base class (Parent)
class Person
{
  name: string;
  age: number;
  constructor(name: string, age: number)
 {
    this.name = name;
    this.age = age;
  }
  greet(): void
{
    console.log(`Hi, I am ${this.name} and I am ${this.age} years old.`);
 }
}

// Derived class (Child) inheriting from the Person class
class Student extends Person
 {
  studentId: number;
  constructor(name: string, age: number, studentId: number) {
    // Call the parent class constructor
    super(name, age);
    this.studentId = studentId;
  }

  // Method specific to the Student class
  study(): void {
    console.log(`${this.name} is studying. (Student ID: ${this.studentId})`);
  }
}

// Derived class (Child) inheriting from the Person class
class Teacher extends Person {
  subject: string;

  constructor(name: string, age: number, subject: string) {
    // Call the parent class constructor
    super(name, age);
    this.subject = subject;
  }
```

```
  // Method specific to the Teacher class
  teach(): void {
    console.log(`${this.name} teaches ${this.subject}.`);
  }
}

// Example usage:
const student = new Student("Alice", 20, 101);
student.greet(); // Inherited method
student.study(); // Specific to Student class

const teacher = new Teacher("Mr. Smith", 45, "Mathematics");
teacher.greet(); // Inherited method
teacher.teach(); // Specific to Teacher class
```

Output:

```
Hi, I am Alice and I am 20 years old.
Alice is studying. (Student ID: 101)
Hi, I am Mr. Smith and I am 45 years old.
Mr. Smith teaches Mathematics.
```



Explanation:

1. The `Person` class is the base (parent) class.
2. The `Student` and `Teacher` classes are derived (child) classes, inheriting from `Person`.
3. The `super()` keyword is used to call the constructor of the parent class.
4. Both derived classes can use methods and properties from the parent class while adding their specific methods.

Conclusion: We have implemented Inheritance example using TypeScript.

Name: Hrishikesh Garige

PRN: 122AX024
Branch: CSE(IOT)
Batch: H2

Experiment No 4

Aim: Build Hello World App in Node.js

Theory:

Node.js is an open-source, cross-platform runtime environment that allows developers to execute JavaScript code on the server side. It uses the V8 JavaScript engine, developed by Google, to convert JavaScript code into machine code. Node.js is designed for building scalable network applications, leveraging an event-driven, non-blocking I/O model that makes it efficient and suitable for handling numerous simultaneous connections.

One of the key features of Node.js is its asynchronous nature, which enables it to perform non-blocking operations. This means that while a task is being completed (like reading files or querying a database), the application can continue processing other requests, resulting in improved performance and responsiveness.

Node.js has a rich ecosystem of libraries and frameworks, primarily accessible through npm (Node Package Manager), which allows developers to easily incorporate third-party modules into their applications. It is particularly popular for building APIs, real-time applications, and microservices due to its lightweight architecture and ability to handle high concurrency.

- **Steps to Create server.js and Paste Code**

1. **Open Command Prompt and Navigate to Your Folder**

cd "C:\Users\swapn\OneDrive\Documents\node js project"

2. **Create the server.js File**

type nul > server.js

3. **Open server.js in Notepad**

notepad server.js

4. **Paste the Following Code into server.js**

```
// Import the HTTP module
const http = require('http');

// Create the server
const server = http.createServer((req, res) => {
   // Set response headers
   res.writeHead(200, { 'Content-Type': 'text/plain' });

   // Send the response
   res.end('Hello to my World!\n');
});

// Define the port
const PORT = 3000;

// Start the server
server.listen(PORT, () => {
   console.log(`Server is running on http://localhost:${PORT}`);
});
```

5. **Save the File**

- **For Notepad:** Click **File → Save** and close it.

Steps to Run the Server

1. **Open Command Prompt** again.
2. Navigate to your project folder: cd "C:\Users\swapn\OneDrive\Documents\node js project"
3. **Run the server:** node server.js
4. If everything is correct, you'll see: Server is running on http://localhost:3000

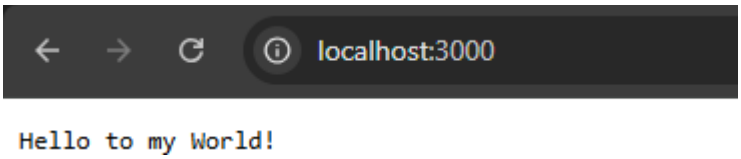
Test in Browser

1. Open **Google Chrome** (or any browser).
2. Go to: http://localhost:3000
3. You should see **"Hello to myWorld!"** displayed.

 How to Stop the Server

If you need to **stop the server**, go to the terminal and press:

CTRL + C



Conclusion: We have built Hello World App in Node.js

Name: Hrishikesh Garige
PRN: 122AX024
Branch: CSE(IOT)
Batch: H2

Experiment No. 5

Aim: To implement Stream and Buffer in Node.js program.

Theory:

1. What are Streams?

Streams in Node.js are used to handle large amounts of data efficiently. They allow reading or writing data in chunks rather than loading the entire data at once. Streams can be:

- **Readable**: Read data from a source (e.g., file, network).
- **Writable**: Write data to a destination.
- **Duplex**: Both readable and writable (e.g., TCP sockets).
- **Transform**: A special type of Duplex stream that can modify data.

2. What is a Buffer?

Buffers in Node.js are temporary storage areas used to handle binary data before processing. They are used when working with streams, especially when dealing with files, network requests, etc.

Step-by-Step Example: Reading and Writing Using Streams and Buffers

Step 1: Create a File for Testing

Create a file named input.txt and add some text to it.

Step 2: Create a Node.js Script

Create a new file stream_example.js and add the following code:

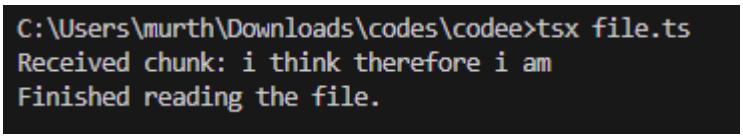
```
const fs = require("fs");

// Creating a readable stream
const readableStream = fs.createReadStream("input.txt");
```

```
// Creating a writable stream
const writableStream = fs.createWriteStream("output.txt");

// Handling the 'data' event to read chunks
readableStream.on("data", (chunk) => {
    console.log("Received chunk:", chunk.toString());
    writableStream.write(chunk); // Writing chunk to output file
});

// Handling the 'end' event
readableStream.on("end", () => {
    console.log("Finished reading the file.");
    writableStream.end();
});

// Handling errors
readableStream.on("error", (err) => {
    console.error("Error:", err);
});
writableStream.on("error", (err) => {
    console.error("Error:", err);
});
```

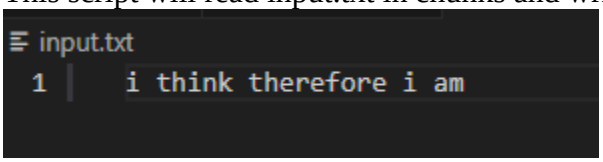
Step 3: Run the Script

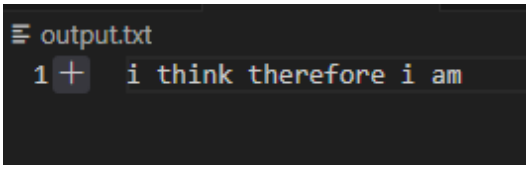
Open the terminal and run:

tsx file.ts



This script will read input.txt in chunks and write it to output.txt.





Step 4: Explanation

1. **fs.createReadStream("input.txt")**: Creates a readable stream to read the file in chunks.
2. **fs.createWriteStream("output.txt")**: Creates a writable stream.
3. **Event Listeners**:
    - "data": Fires when a chunk of data is read.
    - "end": Fires when the file reading is completed.
    - "error": Catches any errors.
4. **Using Buffers**:

    - Chunks are stored in a buffer before processing.
    - Buffers help in handling binary data efficiently.

Error
C:\Users\swapn\OneDrive\Documents\node js project>tsc stream_example.ts
stream_example.ts:1:12 - error TS2580: Cannot find name 'require'. Do you need to install type definitions for node? Try npm i --save-dev @types/node. 1 const fs = require("fs"); ~~~~~~~
Found 1 error in stream_example.ts:1

Fix the Issue

Run the following command in your project directory:

bash
CopyEdit
npm i --save-dev @types/node

This will install the necessary type definitions for Node.js.

Error
The warning npm WARN config dev Please use --include=dev instead. appears because newer versions of npm recommend using --include=dev instead of --save-dev.

Run the following command to install @types/node:

bash
CopyEdit
npm i @types/node --include=dev OR

npm i @types/node --include=dev run this in Command prompt

Explanation of the Node.js Stream Program

Your program reads data from input.txt using a **Readable Stream** and writes it to output.txt using a **Writable Stream**. Let's break it down step by step:

Step 1: Import the File System Module

```
import * as fs from "fs";
```

- The fs (File System) module is required to work with files in Node.js.

Step 2: Create a Readable Stream

```
const readableStream = fs.createReadStream("input.txt");
```

- fs.createReadStream("input.txt") creates a stream to read the file in small chunks.
- Instead of loading the entire file into memory, it reads data **piece by piece** (useful for large files).

Step 3: Create a Writable Stream

```
const writableStream = fs.createWriteStream("output.txt");
```

- fs.createWriteStream("output.txt") creates a writable stream to write data into output.txt.

Step 4: Read and Write Data Using Streams

```
readableStream.on("data", (chunk) => {
   console.log("Received chunk:", chunk.toString());
   writableStream.write(chunk);
});
```

- The "data" event fires when a chunk of data is read.
- chunk.toString() converts the binary Buffer data to readable text.
- writableStream.write(chunk) writes the data to output.txt.

Step 5: Handle End of File Read

```
readableStream.on("end", () => {
   console.log("Finished reading the file.");
   writableStream.end();
});
```

- The "end" event fires when input.txt is fully read.
- writableStream.end(); closes the writable stream.

Step 6: Handle Errors

```
readableStream.on("error", (err) => {
   console.error("Error:", err);
});
writableStream.on("error", (err) => {
   console.error("Error:", err);
});
```

- If there's any issue (e.g., file not found), the error is logged to the console.

Final Output Example

If input.txt contains:

Hello, this is a test file for Node.js stream example!

Terminal Output:

Received chunk: Hello, this is a test file for Node.js stream example!
Finished reading the file.

Content of output.txt:

Hello, this is a test file for Node.js stream example!

Key Benefits of Using Streams

Memory Efficient: Streams handle large files without consuming too much memory.

Fast Processing: Streams process data as soon as it's available, rather than waiting for the whole file.

Scalable: Suitable for handling real-time data processing like video streaming, file uploads, etc.

Conclusion: We have implemented Stream and Buffer in Node.js program.

Name: Hrishikesh Garige

PRN: 122AX024

Branch: CSE(IOT)

Batch: H2

Experiment No. 6

Aim: Configuring Express Settings and creating Express application using request and response objects.

Theory:

Express Framework

Express is a web framework for Node.js that simplifies the creation of web applications and APIs. It provides robust features for web and mobile applications, including routing, middleware, templates, and more. Express makes working with HTTP requests and responses easier than using the built-in Node.js http module directly.

Step 1: Install Express

1. Initialize a new Node.js project: npm init –y

```
PS C:\Users\Dell\Downloads\P> npm init -y
Wrote to C:\Users\Dell\Downloads\P\package.json:

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

2. Install Express:npm install express: npm install express

```
PS C:\Users\Dell\Downloads\P> npm install express

added 69 packages, and audited 70 packages in 2s

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

found 0 vulnerabilities
```

Step 2: Set Up the Basic Express Application

CODE:

```
// app.js
const express = require('express');
const app = express();
const port = 3000;
// Middleware to configure settings (e.g., body parsing, logging)
app.use(express.json());  // for parsing application/json
app.use(express.urlencoded({ extended: true }));  // for parsing application/x-www-form-urlencoded
// Set up a route
app.get('/', (req, res) => {
  res.send('Hello, World! I am Student');
});
// Handling POST request with request and response objects
app.post('/data', (req, res) => {
  const requestData = req.body; // Access data from the request body
  console.log(requestData); // Output the data in the console
  res.json({
    message: 'Data received successfully!',
    data: requestData,
  });
});
 // Start the server
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
```

});

Step 3: Running the Application

node app.js

Output:

Example Request:

Execute the following command in **PowerShell**:

$headers = @{ "Content-Type" = "application/json" }

Invoke-WebRequest -Uri http://localhost:3000/data -Method Post -Headers $headers -Body '{"name": "John", "age": 30}' -ContentType "application/json"

Conclusion: We have configured Express Settings and creating Express application using request and response objects.

Name: Hrishikesh Garige
PRN: 122AX024
Branch: CSE(IOT)
Batch: H2

Experiment No. 7

Aim: Build express application by sending and receiving cookies.

Theory:

Express.js and Cookies – General Theory

What is Express.js?

Express.js is a lightweight, flexible, and minimal web application framework for **Node.js**. It is widely used to build web servers and APIs quickly and efficiently. Express simplifies many of the complexities involved in setting up a server, routing requests, handling middleware, and serving static files.

Key Features of Express.js:

- Easy routing system
- Middleware support
- Fast setup for RESTful APIs
- Integration with template engines (like EJS, Pug)
- Robust ecosystem with plugins

What are Cookies?

Cookies are small pieces of data stored on the client-side (browser) that help websites remember information about a user across requests.

They are usually used for:

- Session management (login status, preferences)
- Tracking user behavior
- Storing user-specific settings or tokens

Each HTTP request and response can carry cookies:

- **Request**: Sends cookies to the server

- **Response**: Server can set cookies on the client

Cookie Properties

Some commonly used properties when setting a cookie:

- name: Name of the cookie
- value: Value of the cookie
- maxAge: Duration in milliseconds before the cookie expires
- httpOnly: Cookie not accessible via JavaScript (more secure)
- secure: Cookie sent only over HTTPS
- path: URL path that must exist for the cookie to be sent

Using Cookies in Express.js

To handle cookies in Express, we use a middleware called **cookie-parser**.

1. Installing cookie-parser:

npm install cookie-parser

2. Using it in an Express app:

const cookieParser = require('cookie-parser');

app.use(cookieParser());

Common Cookie Operations in Express

Setting a Cookie:

res.cookie('key', 'value', options);

Reading a Cookie:

const value = req.cookies.key;

Deleting a Cookie:

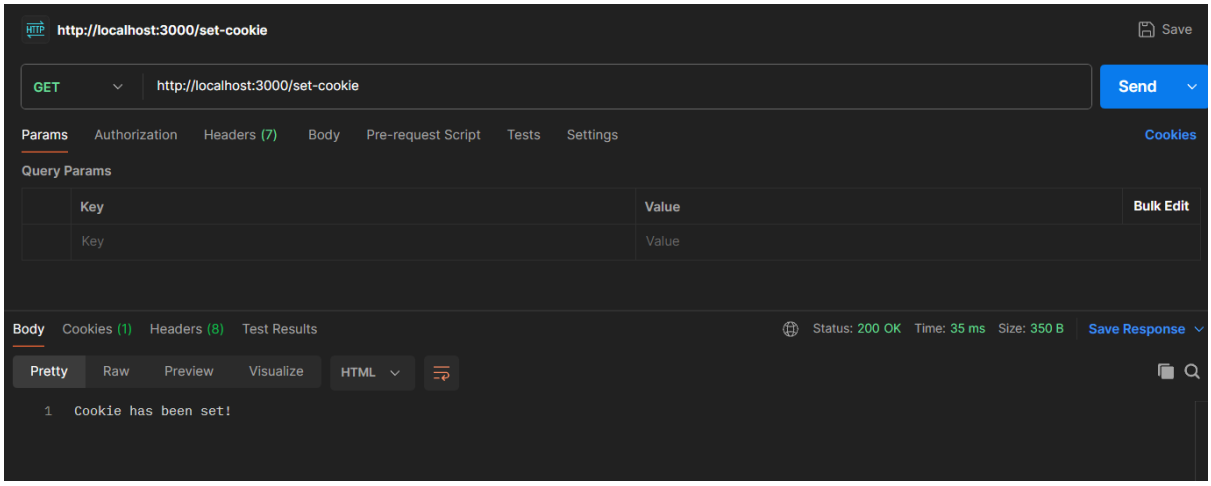
res.clearCookie('key');

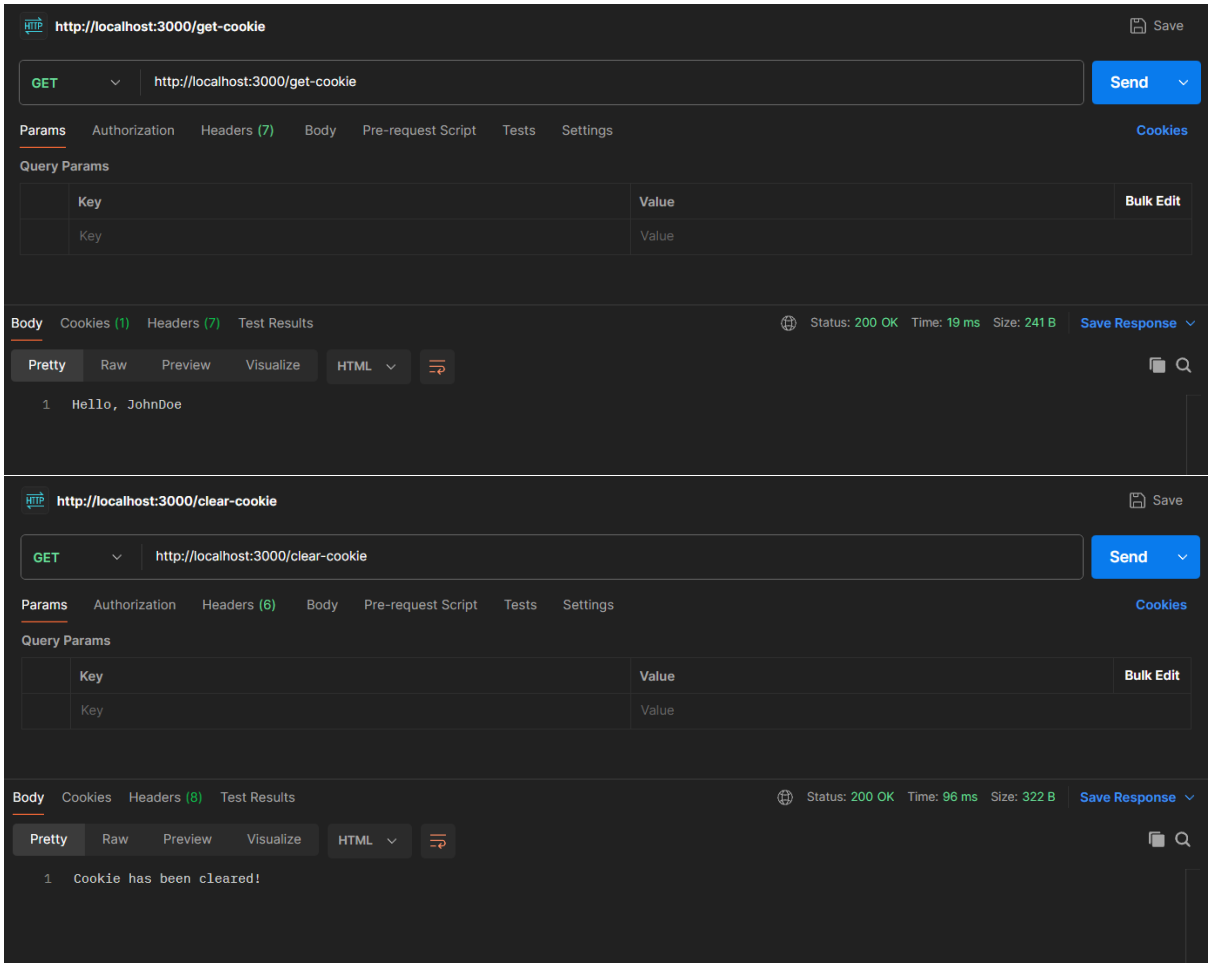
Code:
index.js

```
const express = require('express');
const cookieParser = require('cookie-parser');


const app = express();
const PORT = 3000;


// Middleware
app.use(cookieParser());
app.use(express.json());


// Route to set a cookie
app.get('/set-cookie', (req, res) => {
  res.cookie('username', 'JohnDoe', {
    maxAge: 24 * 60 * 60 * 1000, // 1 day
    httpOnly: true,
    secure: false, // set to true if using HTTPS
  });
  res.send('Cookie has been set!');
});


// Route to get the cookie
app.get('/get-cookie', (req, res) => {
  const username = req.cookies.username;
  if (username) {
    res.send(`Hello, ${username}`);
  } else {
    res.send('No cookie found!');
```

```
  }
});

// Route to clear the cookie
app.get('/clear-cookie', (req, res) => {
  res.clearCookie('username');
  res.send('Cookie has been cleared!');
});

app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

Output:





Conclusion: We have built express application by sending and receiving cookies.

Name: Hrishikesh Garige
PRN: 122AX024
Branch: CSE(IOT)
Batch: H2

Experiment No. 8

Aim: Create a simple HTML “Hello World” Project using AngularJS Framework and apply ng-controller, ngmodel, expression and filters.

Theory:

Introduction to AngularJS Concepts

AngularJS is a JavaScript framework used to build dynamic, single-page applications. It extends HTML with features like two-way data binding, reusable components, and dependency injection, making it easier to develop interactive web applications.

Core Concepts in AngularJS

1. **ng-app**: Initializes an AngularJS application. It marks the root of the app where AngularJS functionality starts.

   html
   Copy
   <body ng-app="myApp">

2. **ng-controller**: Defines a controller to manage data and logic for a specific part of the view. It binds the controller to a section of the HTML.

   html
   Copy

3. **ng-model**: Creates a two-way data binding between the model (JavaScript variable) and the view (HTML element). It updates both automatically when either changes.

   html
   Copy
   <input type="text" ng-model="name">

4. **Expressions**: AngularJS expressions are used to display dynamic content by evaluating JavaScript code in HTML. They are enclosed in {{ }}.

html
Copy
Hello, {{ name }}!

5. **Filters**: Filters format data before displaying it. They are applied with the pipe (|) symbol.

   html
   Copy
   {{ name | uppercase }}

Key Concepts

1. **Services**:
   - **Services** in AngularJS are used to manage and share data across controllers. In the example, the userService handles storing and updating user data (like name and email).
   - The service provides methods like getUserData() to retrieve data and updateUserData() to update it.
2. **Controllers**:
   - The mainCtrl controller manages the user input and form validation.
   - It interacts with the userService to fetch and update the data and provides the logic for submitting the form.
3. **Directives & Validations**:
   - **ng-model** binds the form fields to the controller's model (username and email), enabling two-way data binding.
   - **Validation** is done using built-in directives such as ng-required, ng-minlength, and ng-pattern. The form only submits if it is valid, ensuring correct data entry.
4. **Events**:
   - **ng-click** is used to trigger events, such as the custom triggerEvent() function, which shows an alert when clicked.
5. **Form Submission**:
   - The form uses ng-submit to submit only if the form is valid. The $valid property checks if the form passes validation.

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AngularJS Hello World</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-app="myApp">

    <!-- Using ng-controller to bind AngularJS functionality -->
    <div ng-controller="helloCtrl">
        <!-- Input field bound to 'name' model using ng-model -->
        <h2>Welcome to AngularJS!</h2>
        <label for="name">Enter your name: </label>
        <input type="text" id="name" ng-model="name">

        <!-- Displaying dynamic content using AngularJS expression -->
        <p>Hello, {{ name }}!</p>

        <!-- Using AngularJS filter to format the date -->
        <p>Current date and time: {{ currentDate | date:'fullDate' }}</p>

        <!-- Showing filtered input with uppercase -->
```

```
        <p>Your name in uppercase: {{ name | uppercase }}</p>
    </div>

    <script>
        // Define an AngularJS module named 'myApp'
        var app = angular.module('myApp', []);

        // Define a controller named 'helloCtrl'
        app.controller('helloCtrl', function($scope) {
            // Initialize model variables
            $scope.name = '';
            $scope.currentDate = new Date();
        });
    </script>
</body>
</html>
```

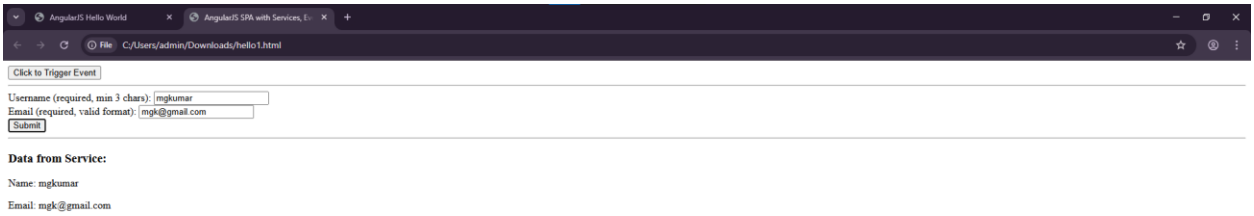
```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>AngularJS SPA with Services, Events, and Validations</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body>

    <div ng-controller="mainCtrl">

        <!-- Event Triggered Button -->
        <button ng-click="triggerEvent()">Click to Trigger Event</button>

        <hr>

        <!-- Form with Validations -->
        <form name="userForm" ng-submit="submitForm(userForm.$valid)" novalidate>
            <label for="username">Username (required, min 3 chars): </label>
            <input type="text" id="username" name="username" ng-model="user.username" ng-required="true" ng-minlength="3">
            <span style="color: red" ng-show="userForm.username.$dirty && userForm.username.$invalid">
                Username is required and should have at least 3 characters.
            </span>
            <br>

            <label for="email">Email (required, valid format): </label>
            <input type="email" id="email" name="email" ng-model="user.email" ng-required="true" ng-pattern="/^[a-zA-Z0-9._%+-]+@[a-zA0-9.-]+\.[a-zA-Z]{2,4}$/">
```

```
        <span style="color: red" ng-show="userForm.email.$dirty &&
userForm.email.$invalid">
            Valid email is required.
        </span>
        <br>

        <button type="submit" ng-disabled="userForm.$invalid">Submit</button>
    </form>

    <hr>

    <div>
        <h3>Data from Service:</h3>
        <p>Name: {{ userData.name }}</p>
        <p>Email: {{ userData.email }}</p>
    </div>

</div>

<script>
    // Define the AngularJS module
    var app = angular.module('myApp', []);

    // Service to store and retrieve user data
    app.service('userService', function() {
```

```
    var userData = {
        name: 'John Doe',
        email: 'johndoe@example.com'
    };

    this.getUserData = function() {
        return userData;
    };

    this.updateUserData = function(newData) {
        userData.name = newData.username;
        userData.email = newData.email;
    };
});

// Controller for the application
app.controller('mainCtrl', function($scope, userService) {
    // Load data from service
    $scope.userData = userService.getUserData();

    // User model for the form
    $scope.user = {
        username: '',
        email: ''
    };
```

```
        // Triggering an event
        $scope.triggerEvent = function() {
            alert("Custom event triggered!");
        };

        // Function to handle form submission
        $scope.submitForm = function(isValid) {
            if (isValid) {
                userService.updateUserData($scope.user); // Update service with form data
                $scope.userData = userService.getUserData(); // Refresh user data in the view
                alert("Form submitted successfully!");
            } else {
                alert("Please correct the form errors before submitting.");
            }
        };
    });
  </script>

</body>
</html>
```



Conclusion: We have created a simple HTML “Hello World” Project using AngularJS Framework and apply ng-controller, ngmodel, expression and filters.

Name: Hrishikesh Garige
PRN: 122AX024
Branch: CSE(IOT)
Batch: H2

Experiment No. 9

Aim: Create an Application for like Studnet Record using AngularJS.

Theory:

Student Record Management Application – Theory

Introduction

This project is a simple **Student Record Management System** built using **AngularJS (v1.x)**. It demonstrates basic concepts of AngularJS such as **data binding**, **controllers**, **models**, and **directives**. The application runs completely on the client side and allows the user to **add**, **view**, and **delete** student records dynamically in the browser.

Technologies Used

- **HTML**: For structuring the web page.
- **CSS**: For basic styling.
- **AngularJS**: A JavaScript framework for building dynamic, single-page applications (SPAs).

What is AngularJS?

AngularJS is a structural JavaScript framework maintained by Google, used for building dynamic web applications. It extends HTML with new attributes and binds data to HTML using expressions.

Key Features:

- Two-way data binding
- MVC (Model-View-Controller) pattern
- Dependency Injection
- Directives and custom components
- Built-in services and filters

Project Overview

The application provides a simple interface to manage student records. Each student record contains the following fields:

- **Name**
- **Age**
- **Branch**

Functionalities:

1. **Add Student**: Users can input student details and add them to a list.
2. **Display Students**: All added student records are displayed in a table format.
3. **Delete Student**: Each student entry has a delete button to remove the record from the list.

Application Flow

1. The application is initialized with an empty student list.
2. When the user enters a new student's name, age, and branch, and clicks the "Add Student" button, the information is added to the list using AngularJS's **two-way data binding** and **controller logic**.
3. The table is automatically updated with the new student data.
4. Each row has a delete button, which removes the student from the list when clicked.

AngularJS Concepts Used

- **Module**: Declared using angular.module('studentApp', []), it acts as the main container for different parts of the app.
- **Controller**: StudentController handles the logic for adding and removing students.
- **Scope ($scope)**: Acts as a bridge between the controller and the view (HTML). It holds the student data and functions.
- **ng-model**: Binds the input fields to the controller's scope.
- **ng-repeat**: Iterates through the list of students and displays them in a table.
- **ng-submit** and **ng-click**: Handle form submission and button click events.

Advantages of Using AngularJS

- Clean separation between logic and UI
- Easy to manage data and DOM updates
- Less boilerplate code compared to vanilla JavaScript

Code:

index.html

```
<!DOCTYPE html>
<html ng-app="studentApp">
<head>
  <meta charset="UTF-8">
  <title>Student Record App</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <script src="app.js"></script>
  <style>
    body { font-family: Arial; padding: 20px; }
    input, button { margin: 5px 0; }
    table { width: 100%; border-collapse: collapse; margin-top: 20px; }
    th, td { border: 1px solid #ccc; padding: 8px; text-align: left; }
  </style>
</head>
<body ng-controller="StudentController">

  <h1>Student Record Management</h1>

  <form ng-submit="addStudent()">
    <input type="text" ng-model="newStudent.name" placeholder="Student Name" required>
    <input type="number" ng-model="newStudent.age" placeholder="Age" required>
    <input type="text" ng-model="newStudent.branch" placeholder="Branch" required>
    <button type="submit">Add Student</button>
  </form>

  <table>
```

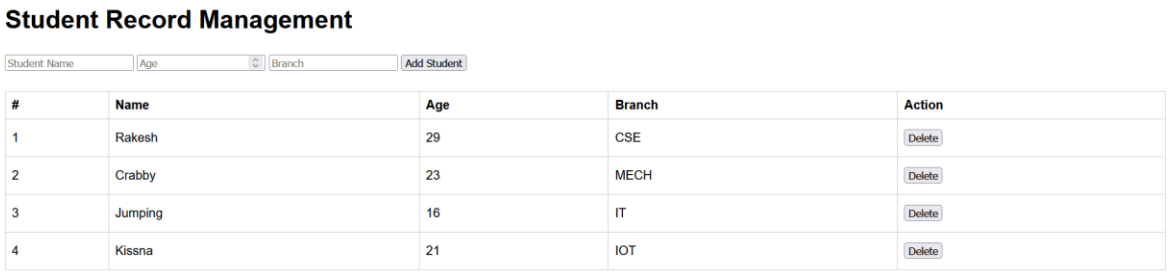
```
    <tr>
      <th>#</th>
      <th>Name</th>
      <th>Age</th>
      <th>Branch</th>
      <th>Action</th>
    </tr>
    <tr ng-repeat="student in students track by $index">
      <td>{{$index + 1}}</td>
      <td>{{student.name}}</td>
      <td>{{student.age}}</td>
      <td>{{student.branch}}</td>
      <td><button ng-click="removeStudent($index)">Delete</button></td>
    </tr>
  </table>

</body>
</html>
```

app.js

```
var app = angular.module('studentApp', []);

app.controller('StudentController', function ($scope) {
  $scope.students = [];

  $scope.newStudent = {};

  $scope.addStudent = function () {
```

```
    if ($scope.newStudent.name && $scope.newStudent.age && $scope.newStudent.branch)
{
      $scope.students.push(angular.copy($scope.newStudent));
      $scope.newStudent = {};
    }
  };

  $scope.removeStudent = function (index) {
    $scope.students.splice(index, 1);
  };
});
```

Output:



Conclusion: We have created an Application for like Studnet Record using AngularJS.

Name: Hrishikesh Garige
PRN: 122AX024
Branch: CSE(IOT)
Batch: H2

Experiment No. 10

Aim: Connect MongoDB with NodeJS and perform CRUD operations.

Theory:

What is MongoDB?

MongoDB is a NoSQL database that stores data in flexible, JSON-like documents. Unlike traditional relational databases, MongoDB does not require a fixed schema, making it suitable for modern applications where data structure may change frequently.

Key Features:

- Document-oriented storage
- High scalability
- Flexible schemas
- Uses BSON (Binary JSON) format internally
- Supports a powerful query language for data retrieval and manipulation

Connecting MongoDB with Node.js:

To interact with MongoDB in a Node.js application, Mongoose is commonly used. Mongoose is an Object Data Modeling (ODM) library that provides a structured way to define data models and interact with MongoDB collections.

The basic steps for connection involve:

- Installing Mongoose via npm
- Importing it into the application
- Connecting to the MongoDB instance using a connection URI

Once connected, Mongoose enables the developer to define schemas and models for consistent interaction with MongoDB.

CRUD Operations in MongoDB:

CRUD stands for Create, Read, Update, and Delete. These are the four basic operations for managing data in any database.

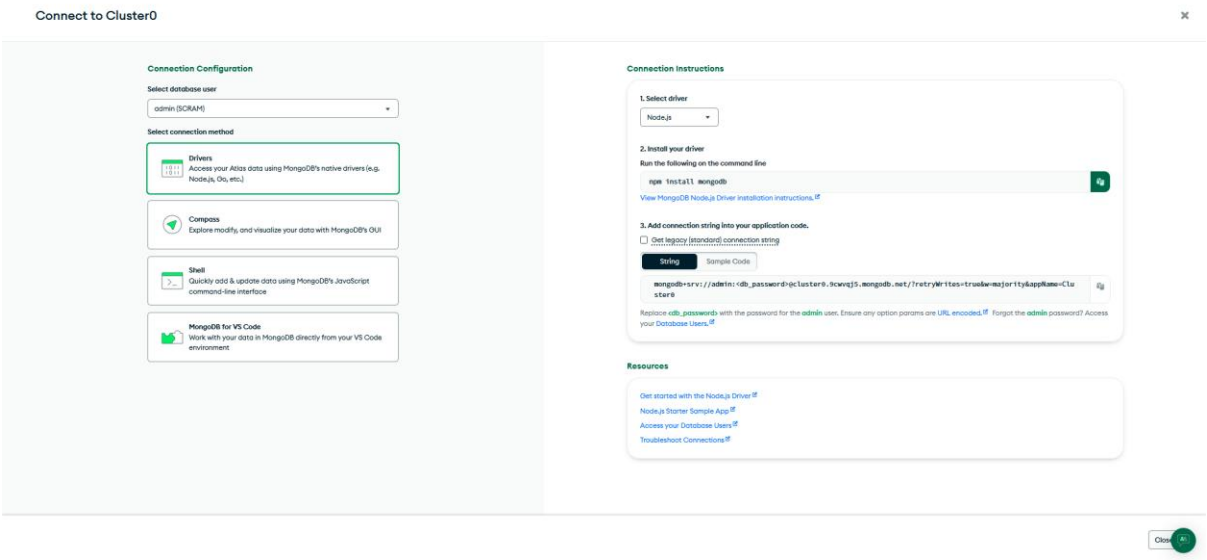
1. **Create**: Inserting new documents into a collection. This is typically done using the .save() method or the .create() method in Mongoose.
2. **Read**: Retrieving documents from a collection using queries. Mongoose provides methods like .find(), .findOne(), and others to filter and fetch records.
3. **Update**: Modifying existing documents. This can be performed using methods such as .updateOne(), .updateMany(), or .findByIdAndUpdate().
4. **Delete**: Removing documents from a collection. This is done using methods like .deleteOne(), .deleteMany(), or .findByIdAndDelete().

Schema and Model in Mongoose:

A schema defines the structure of a document, specifying fields and their data types. Once a schema is defined, a model is created from it. The model acts as a constructor for creating and managing documents within the collection that the schema represents.

Using schemas and models ensures consistency, validation, and provides built-in methods for performing CRUD operations in a structured and efficient way.

Create a Free Cluster and Connect MongoDB to get the URI



Create an .env file with the following parameters:
PORT=4545

MONGO_URI= <your_mongo_uri>

password = <your_password>

Code:
app.js

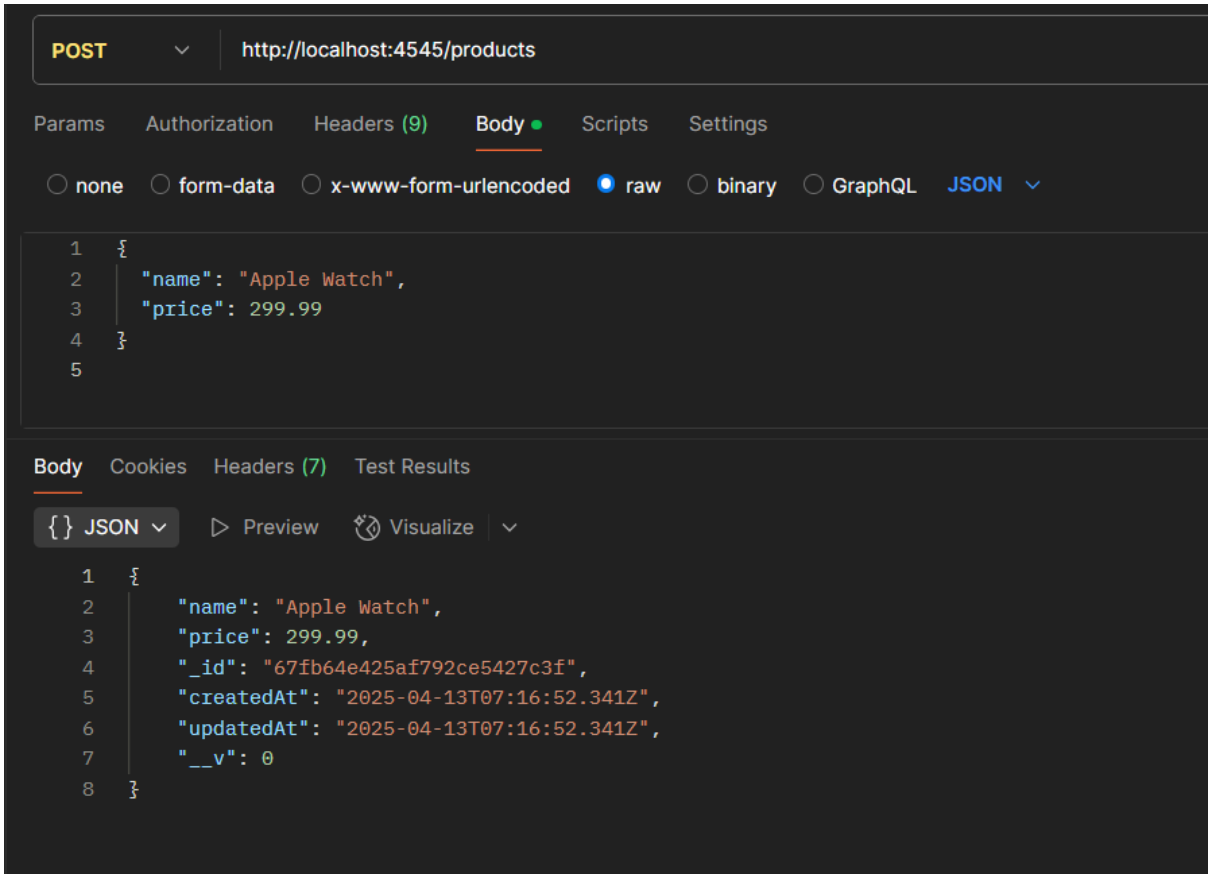
```
import express from "express";
import mongoose from "mongoose";
import dotenv from "dotenv";
import Product from "./schema.js";

dotenv.config();

const app = express();

app.use(express.json());

// MongoDB connection
mongoose
  .connect(process.env.MONGO_URI)
  .then(() => console.log("MongoDB connected"))
  .catch((err) => console.error("MongoDB connection error:", err));

// CRUD APIs

// GET all products
app.get("/products", async (req, res) => {
  try {
    const products = await Product.find();
    res.json(products);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
```

```
});

// POST create new product
app.post("/products", async (req, res) => {
  try {
    const { name, price } = req.body;
    const newProduct = new Product({ name, price });
    await newProduct.save();
    res.status(201).json(newProduct);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

// PUT update product by id
app.put("/products/:id", async (req, res) => {
  try {
    const { name, price } = req.body;
    const updated = await Product.findByIdAndUpdate(
      req.params.id,
      { name, price },
      { new: true }
    );
    res.json(updated);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});
```

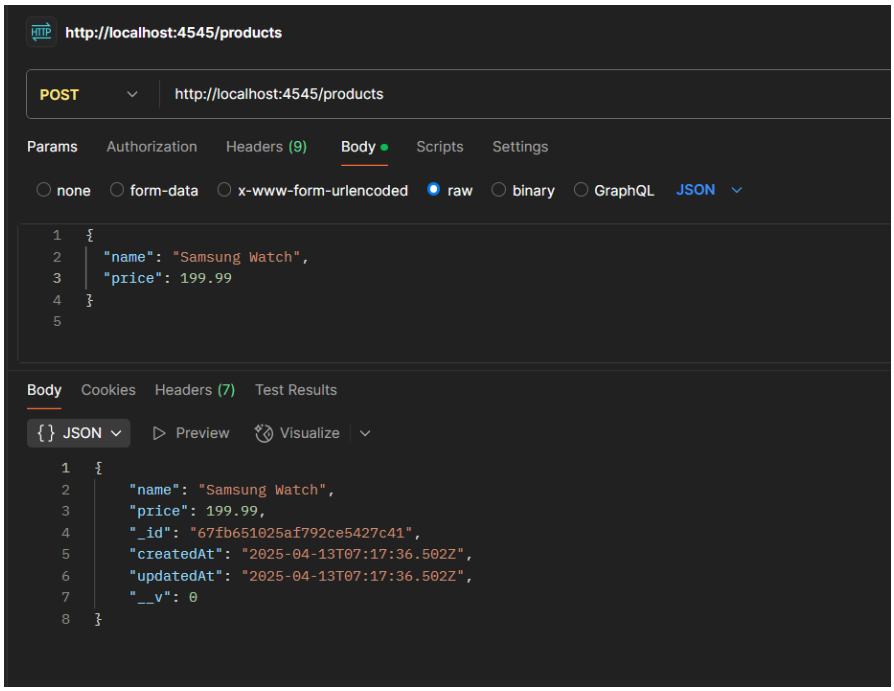
```
// DELETE product by id
app.delete("/products/:id", async (req, res) => {
  try {
    await Product.findByIdAndDelete(req.params.id);
    res.json({ message: "Product deleted" });
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

export default app;
```

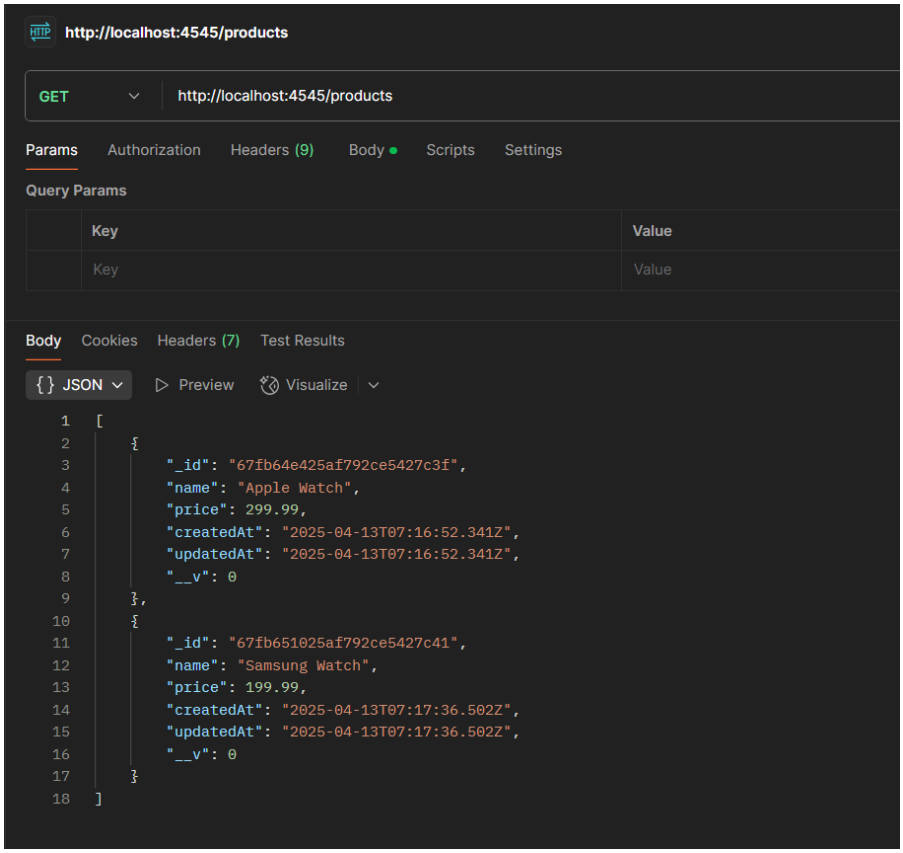
Ouput:

Creating a Product

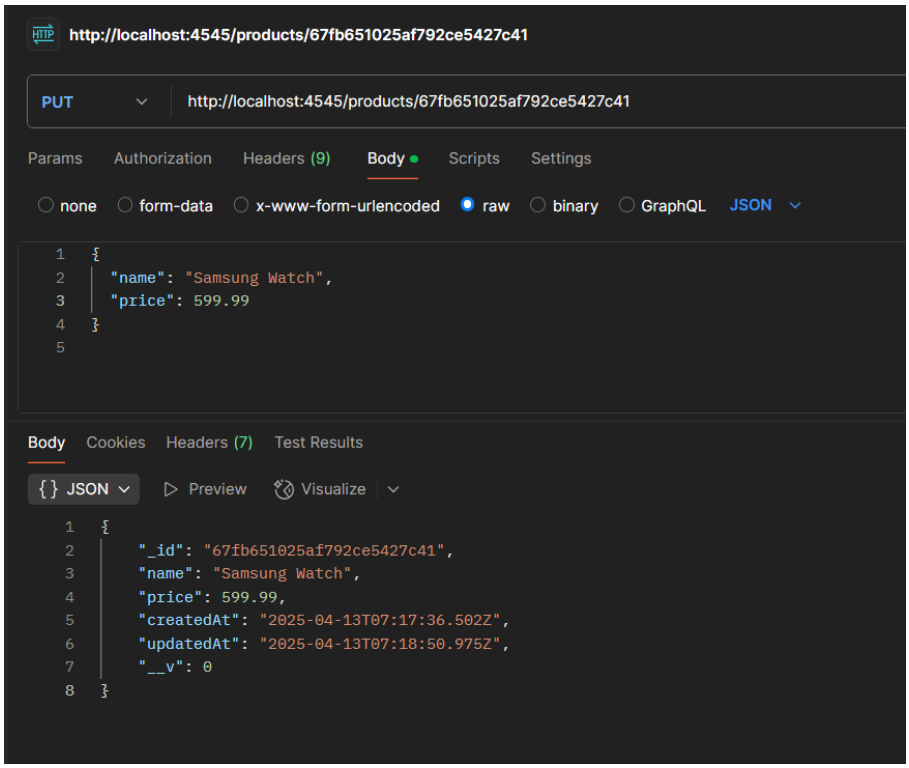


Adding Product

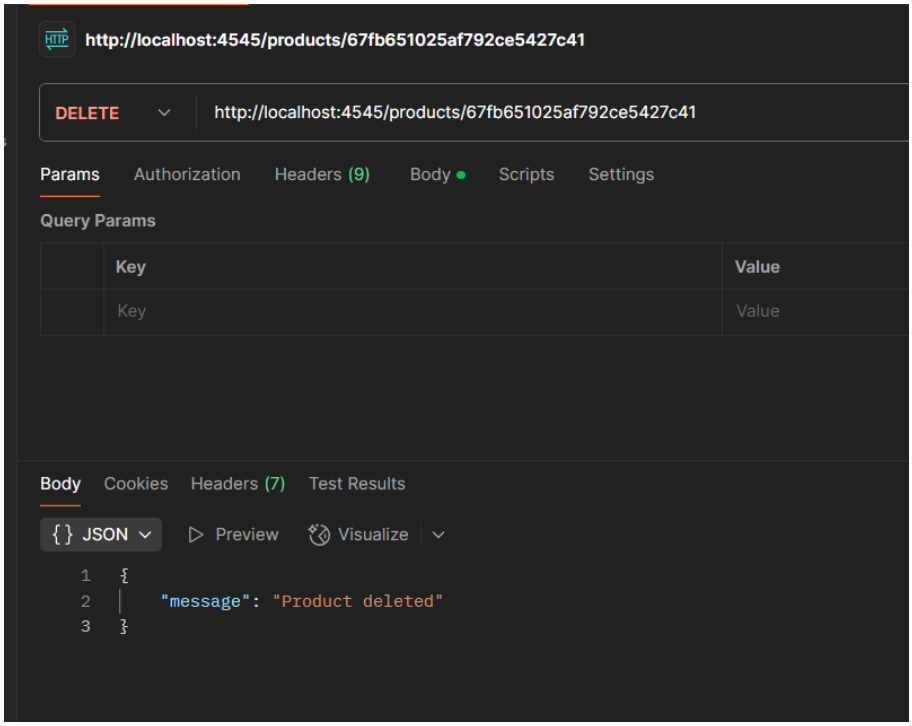




Updating Price



Deleting Product



Conclusion: We have connected MongoDB with NodeJS and perform CRUD operations.

Name: Hrishikesh Garige
PRN: 122AX024
Branch: CSE(IOT)
Batch: H2

Experiment No. 11

Aim: Build a RESTful API using MongoDB

Theory:

What is a RESTful API?

A RESTful API (Representational State Transfer) is an architectural style for designing networked applications. It uses standard HTTP methods to perform operations on resources, which are represented as URLs.

Key principles of REST:

- **Stateless:** Each request from the client contains all the information needed to process it.
- **Client-Server architecture:** The client and server operate independently.
- **Uniform Interface:** Resources are accessed through consistent and predictable URIs.
- **HTTP methods used:**
    - GET: Retrieve data
    - POST: Create new data
    - PUT or PATCH: Update existing data
    - DELETE: Remove data

MongoDB in RESTful APIs:

MongoDB, a NoSQL database, pairs well with REST APIs for storing and retrieving data in a flexible, schema-less format. It is commonly used with Mongoose to handle data modeling and interaction with collections.

Building the API:

To build a RESTful API using MongoDB, the following technologies are used:

- **Node.js:** A JavaScript runtime that allows building scalable server-side applications.
- **Express.js:** A web framework for Node.js that simplifies routing and handling HTTP requests.
- **Mongoose:** An ODM library that helps interact with MongoDB using schemas and models.

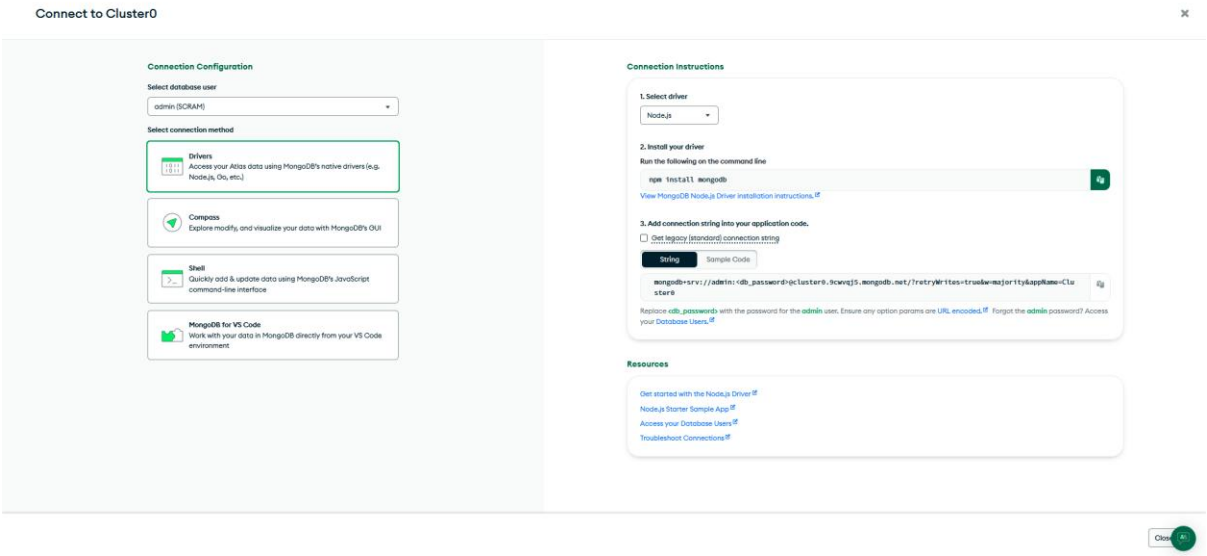
Steps involved:

1. **Set up the project environment:** Initialize a Node.js application and install dependencies like Express and Mongoose.
2. **Connect to MongoDB:** Use Mongoose to establish a connection to the database.
3. **Define the schema and model:** Represent the structure of your data.
4. **Create RESTful routes:**
    - GET /resource: List all records
    - GET /resource/:id: Get a single record by ID
    - POST /resource: Create a new record
    - PUT /resource/:id: Update a record
    - DELETE /resource/:id: Delete a record
5. **Handle responses and errors:** Return appropriate status codes and messages based on the request outcome.

Advantages of Using MongoDB with REST APIs:

- **Scalability:** MongoDB is designed for horizontal scaling.
- **Flexibility:** No strict schema, making it easy to adapt as requirements change.
- **JSON-based format:** Data is stored in BSON, which is easily converted to JSON used in APIs.
- **High performance:** MongoDB supports high read and write throughput for large volumes of data.

Create a Free Cluster and Connect MongoDB to get the URI



Create an .env file with the following parameters:
PORT=4545

MONGO_URI= <your_mongo_uri>

password = <your_password>

Code:
app.js

```
import express from "express";

import mongoose from "mongoose";

import dotenv from "dotenv";

import Product from "./schema.js";


dotenv.config();


const app = express();


app.use(express.json());


// MongoDB connection

mongoose
```

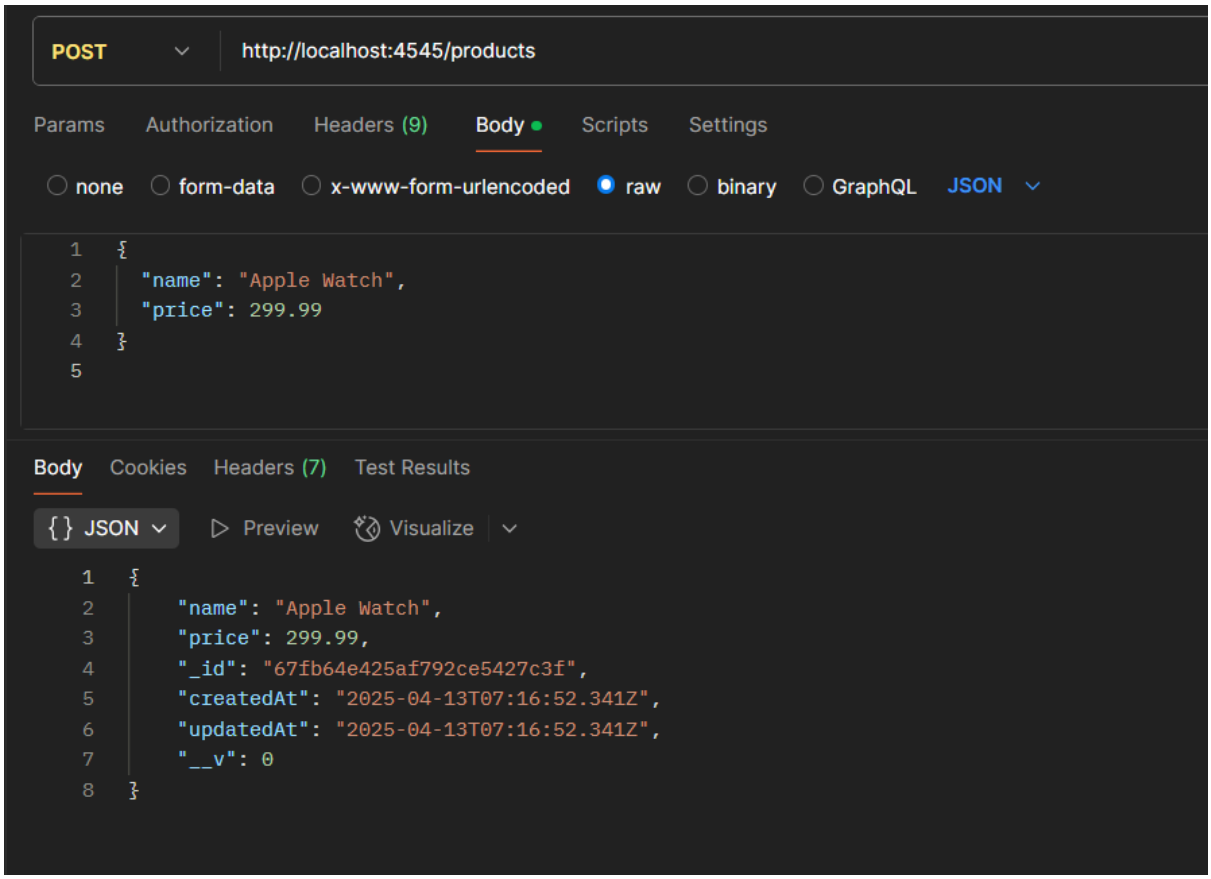
```
  .connect(process.env.MONGO_URI)
  .then(() => console.log("MongoDB connected"))
  .catch((err) => console.error("MongoDB connection error:", err));

// CRUD APIs

// GET all products
app.get("/products", async (req, res) => {
  try {
    const products = await Product.find();
    res.json(products);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// POST create new product
app.post("/products", async (req, res) => {
  try {
    const { name, price } = req.body;
    const newProduct = new Product({ name, price });
    await newProduct.save();
    res.status(201).json(newProduct);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

// PUT update product by id
```

```
app.put("/products/:id", async (req, res) => {
  try {
    const { name, price } = req.body;
    const updated = await Product.findByIdAndUpdate(
      req.params.id,
      { name, price },
      { new: true }
    );
    res.json(updated);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

// DELETE product by id
app.delete("/products/:id", async (req, res) => {
  try {
    await Product.findByIdAndDelete(req.params.id);
    res.json({ message: "Product deleted" });
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

export default app;
```

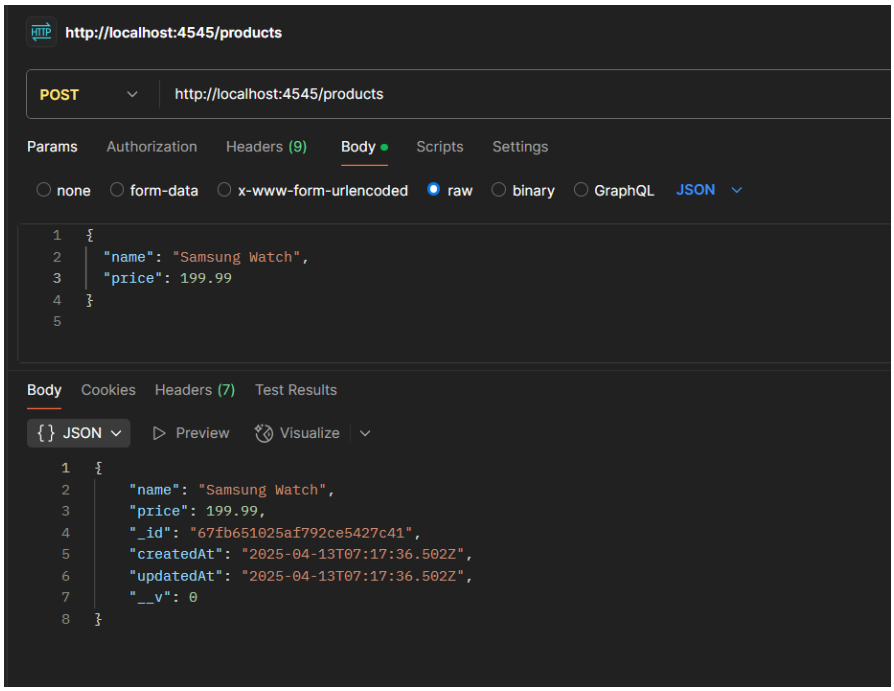
```
PS C:\Users\yadav\OneDrive\Desktop\This\PROJECT> cd webx
PS C:\Users\yadav\OneDrive\Desktop\This\PROJECT\webx> npm install

added 113 packages, and audited 114 packages in 3s

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

found 0 vulnerabilities
PS C:\Users\yadav\OneDrive\Desktop\This\PROJECT\webx> npm run dev

> webx@1.0.0 dev
> nodemon server.js

[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server running on http://localhost:4545
MongoDB connected
```

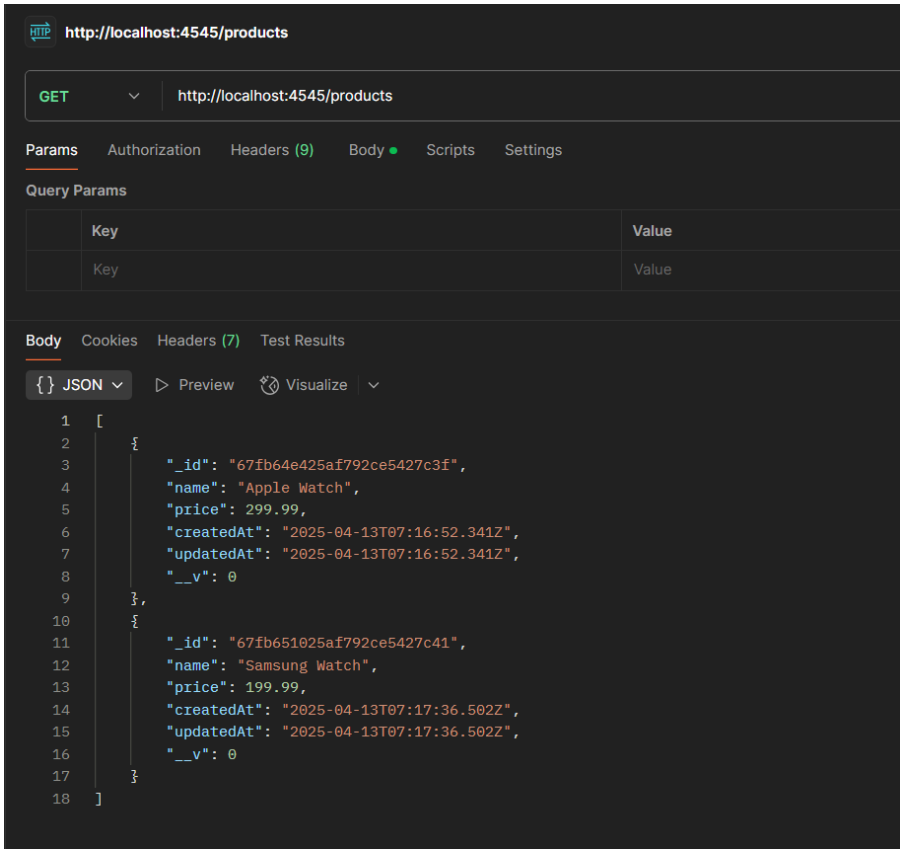
Ouput:

Creating

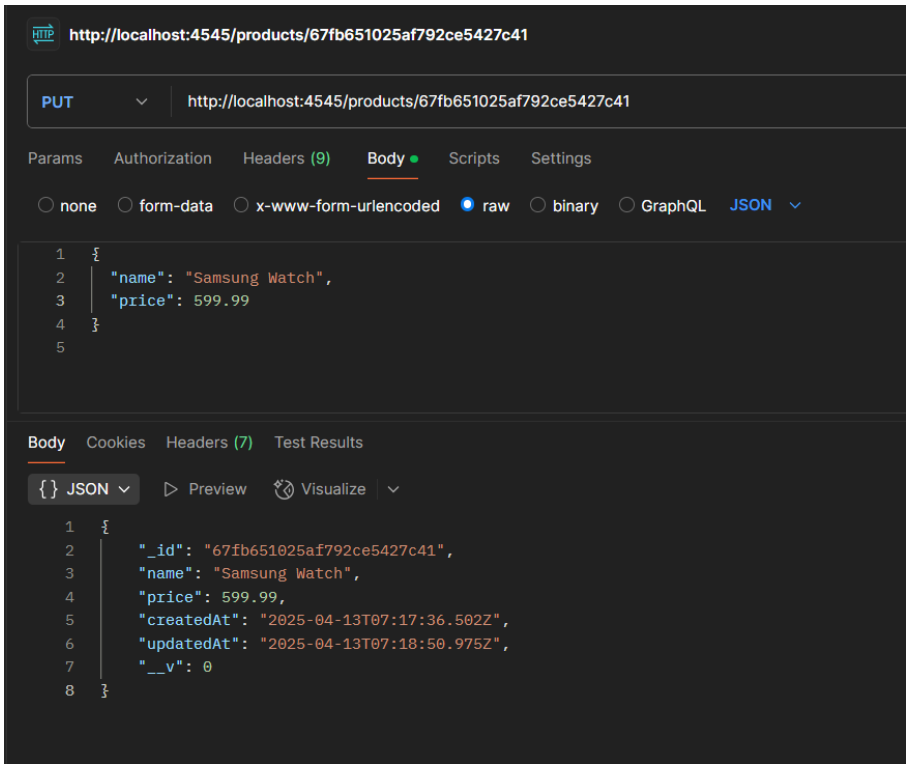


Adding

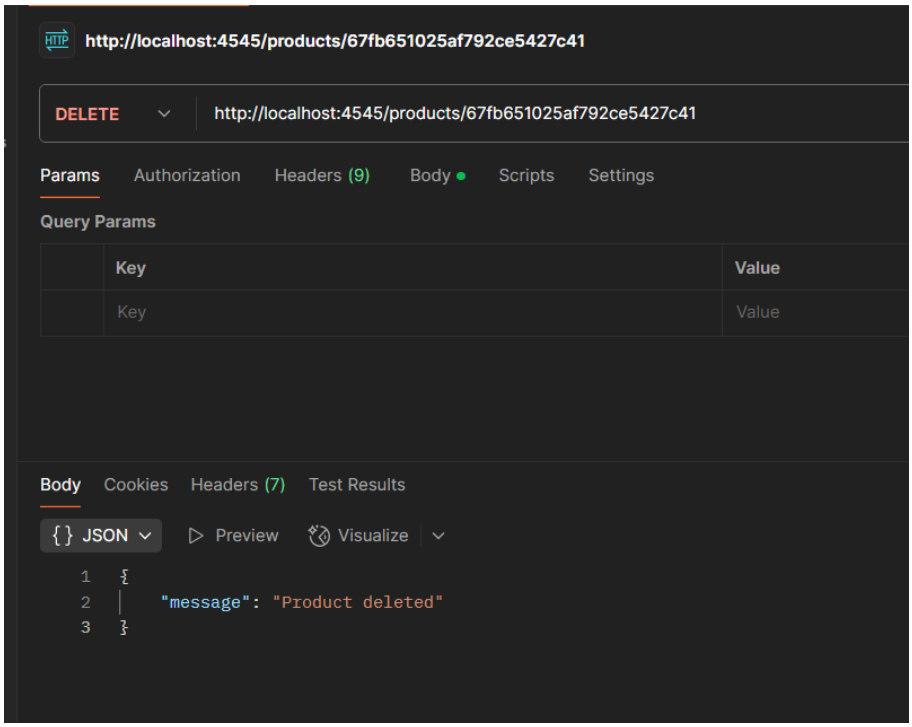

http://localhost:4545/products
POST
http://localhost:4545/products
Params
Authorization
Headers (9)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"name": "Samsung Watch",
"price": 199.99
}
Body
Cookies
Headers (7)
Test Results
JSON
Preview
Visualize
{
"name": "Samsung Watch",
"price": 199.99,
"_id": "67fb651025af792ce5427c41",
"createdAt": "2025-04-13T07:17:36.502Z",
"updatedAt": "2025-04-13T07:17:36.502Z",
"__v": 0
}


http://localhost:4545/products
GET
http://localhost:4545/products
Params
Authorization
Headers (9)
Body
Scripts
Settings
Query Params
Key
Value
Key
Value
Body
Cookies
Headers (7)
Test Results
JSON
Preview
Visualize
[
{
"_id": "67fb64e425af792ce5427c3f",
"name": "Apple Watch",
"price": 299.99,
"createdAt": "2025-04-13T07:16:52.341Z",
"updatedAt": "2025-04-13T07:16:52.341Z",
"__v": 0
},
{
"_id": "67fb651025af792ce5427c41",
"name": "Samsung Watch",
"price": 199.99,
"createdAt": "2025-04-13T07:17:36.502Z",
"updatedAt": "2025-04-13T07:17:36.502Z",
"__v": 0
}
]

Updating



Deleting



Conclusion: We have built a RESTful API using MongoDB.