Name:	Dhanashree Gawai
Roll No:	21
Class/Sem:	TE/V
Experiment No.:	6
Title:	Perform NodeJS routing for method driven CRUD operations.
Date of Performance:	01/08/2025
Date of Submission:	12/08/2025
Marks:	
Sign of Faculty:	

Aim: Perform NodeJS routing for method driven CRUD operations.

Objective: Understanding the create, retrieve, update and delete operations and implementing them using NodeJS for corrective routing mechanism for data management.

Theory:

Node.js, a JavaScript runtime built on Chrome's V8 engine, has redefined the landscape of server-side development since its release in 2009. By enabling JavaScript to run on the server, Node.js allows developers to

use a single programming language across the entire development stack, streamlining the development process. Central to its operation are Node.js methods and its implementation of CRUD (Create, Read, Update, Delete) operations, which are fundamental to building robust and efficient web applications.

The Fundamentals of Node.js

Node.js is designed to be fast, scalable, and efficient, particularly for building network applications. It achieves this through its non-blocking, event-driven architecture. Unlike traditional server-side technologies that use multi-threaded request handling, Node.js employs a single-threaded event loop to manage asynchronous operations. This approach minimizes resource consumption and enhances performance, making Node.js ideal for real-time applications such as chat servers, online gaming, and collaborative tools.

Core Node.js Methods

Node.js provides a rich set of methods and modules that facilitate server-side development. Some of the core methods include:

1. **File System (fs) Methods**: Node.js includes a built-in fs module that provides an API for interacting with the file system. Methods such as readFile, writeFile, and unlink enable developers to read from, write to, and delete files asynchronously, which is crucial for tasks like logging, data storage, and configuration management.

2. **HTTP Methods**: The http module in Node.js allows developers to create HTTP servers and clients. Methods like createServer, listen, and request are essential for handling HTTP requests and responses, forming the backbone of web server development.

3. **Stream Methods**: Node.js supports streaming data, which is particularly useful for handling large files or real-time data transfer. The stream module provides methods for creating readable and writable streams, enabling efficient data processing without loading entire files into memory.

4. **Event Methods**: Node.js's events module allows developers to create and handle custom events. Methods such as on and emit facilitate the event-driven architecture of Node.js, enabling asynchronous handling of tasks and improving application responsiveness.

5. **Buffer Methods**: The buffer module is used for handling binary data directly, which is critical for applications that require manipulation of raw data, such as handling image files or implementing network protocols.

CRUD Operations in Node.js

CRUD operations are the foundation of most web applications, enabling users to interact with data through creating, reading, updating, and deleting records. Node.js, with its asynchronous nature and powerful libraries, provides efficient ways to implement CRUD functionality.

1. **Create**: Creating data typically involves inserting new records into a database. In Node.js, this can be accomplished using various database modules like mongoose for MongoDB or sequelize for SQL databases. The create operation often includes data validation and error handling to ensure data integrity.

2. **Read**: Reading data involves querying the database and retrieving records. Node.js's asynchronous methods make it easy to handle multiple read requests concurrently without blocking the event loop.

   Libraries like express can be used to define routes and handlers for fetching data, which is then sent back to the client in the form of JSON or HTML.

3. **Update**: Updating records requires locating the existing data and modifying it based on new inputs. Node.js provides robust methods for updating data efficiently, often utilizing query parameters or

request bodies to specify changes. Middleware functions in frameworks like express can be employed to handle authentication and validation before updating records.

4. **Delete**: Deleting records involves removing data from the database. This operation, while straightforward, must be handled with care to prevent accidental data loss. Node.js methods for deletion typically include confirmation steps or soft delete mechanisms, where records are marked as deleted but not removed entirely from the database.

Advantages of Node.js for CRUD Operations

Node.js offers several advantages that make it particularly well-suited for implementing CRUD operations:

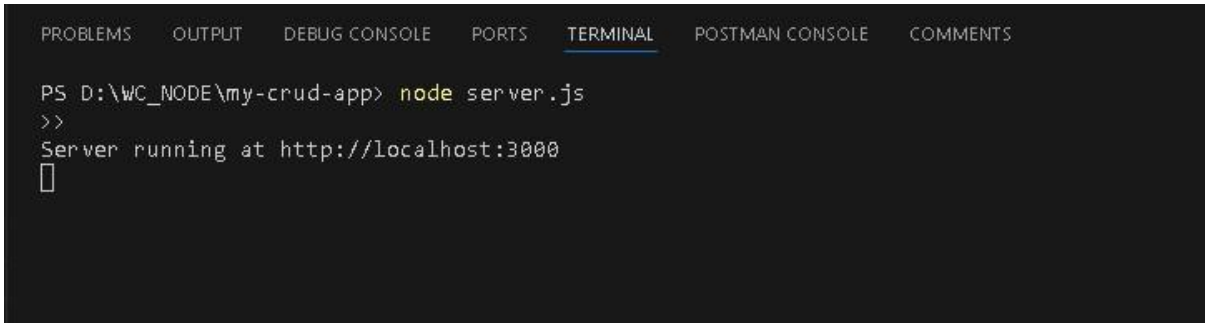
1. **Asynchronous Processing**: The non-blocking nature of Node.js allows it to handle multiple CRUD operations simultaneously, improving the performance and scalability of web applications.
2. **Unified Language Stack**: Using JavaScript for both client-side and server-side development simplifies the development process and reduces the learning curve, enabling developers to be more productive and maintain consistency across the application stack.
3. **Rich Ecosystem**: Node.js has a vast ecosystem of libraries and frameworks, such as express, mongoose, and sequelize, which provide powerful tools for building and managing CRUD operations efficiently.
4. **Scalability**: The event-driven architecture of Node.js makes it highly scalable, capable of handling a large number of concurrent connections with minimal resource consumption. This scalability is particularly beneficial for applications that require real-time data processing and high availability.

Code:

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

const app = express(); const port

= 3000; app.use(express.json());

let items = [];

app.post('/items', (req, res) => {

   const newItem = { id: items.length + 1, ...req.body };

items.push(newItem);     res.status(201).json(newItem);

});

app.get('/items', (req, res) => {

res.json(items);

});

app.get('/items/:id', (req, res) => {

   const item = items.find(i => i.id ===

parseInt(req.params.id)); if (!item) return

res.status(404).send('Item not found'); res.json(item);

});
```

```
app.put('/items/:id', (req, res) => {
   const item = items.find(i => i.id ===
parseInt(req.params.id)); if (!item) return
res.status(404).send('Item not found'); Object.assign(item,
req.body);                              res.json(item);
});
app.delete('/items/:id', (req, res) => {
   const index = items.findIndex(i => i.id === parseInt(req.params.id));
if (index === -1) return res.status(404).send('Item not found');     const
deletedItem = items.splice(index, 1);     res.json(deletedItem[0]);
});
app.listen(port, () => {
   console.log(`Server running at http://localhost:${port}`);
});
```

Output:



Conclusion

Node.js has revolutionized server-side web development with its event-driven, non-blocking architecture and extensive set of methods. Its ability to efficiently handle CRUD operations is central to its success, enabling developers to build robust, scalable, and high-performance web applications. By leveraging Node.js methods and its powerful ecosystem, developers can create dynamic and responsive applications that meet the demands of modern web users. As the Node.js community continues to grow and evolve, its impact on web development will undoubtedly remain profound, shaping the future of how we build and interact with web applications.