1. **Write a Node.js program that reads a user’s name from the console and prints a greeting message asynchronously.**

const readline = require('readline');

// Create readline interface

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

// Ask for the user's name

rl.question('What is your name? ', (name) => {

// Asynchronous greeting (using setTimeout to simulate async operation)

setTimeout(() => {

console.log(`Hello, ${name}! Welcome to Node.js.`);

rl.close(); // Close the readline interface

}, 1000); // Delay of 1 second

});

1. **Write a Node.js script to create, read, update, and delete (CRUD) a text file using the fs module.**

const fs = require('fs');

const filePath = 'example.txt';

// CREATE

fs.writeFile(filePath, 'Hello, this is the initial content.\n', (err) => {

if (err) return console.error('Create Error:', err);

console.log('File created successfully.');

// READ

fs.readFile(filePath, 'utf8', (err, data) => {

if (err) return console.error('Read Error:', err);

console.log('File content after creation:\n', data);

// UPDATE (Append to the file)

fs.appendFile(filePath, 'This is the updated content.\n', (err) => {

if (err) return console.error('Update Error:', err);

console.log('File updated successfully.');

// READ again to verify update

fs.readFile(filePath, 'utf8', (err, updatedData) => {

if (err) return console.error('Read Error:', err);

console.log('File content after update:\n', updatedData);

// DELETE

fs.unlink(filePath, (err) => {

if (err) return console.error('Delete Error:', err);

console.log('File deleted successfully.');

});

});

});

});

});

1. **Install Express.js and set up a basic Express server.**

Step 1: Initialize a Node.js Project

mkdir my-express-app

cd my-express-app

npm init -y

This will create a package.json file.

Step 2: Install [Express.js](http://express.js)

npm install express

### **Step 3: Create the Express Server**

Create a file named server.js and add the following code:

const express = require('express');

const app = express();

const port = 3000;

// Basic route

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

res.send('Hello, Express!');

});

// Start the server

app.listen(port, () => {

console.log(`Server is running at http://localhost:${port}`);

});

### **Step 4: Run the Server**

In your terminal, run:

node server.js

Server is running at http://localhost:3000

Hello, Express!

1. **Create an Express.js application with routes for handling GET and POST requests for user data.**

Step 1: Set Up Project and Install Express

mkdir express-user-app

cd express-user-app

npm init -y

npm install express

Step 2: Create app.js with GET & POST Routes

const express = require('express');

const app = express();

const port = 3000;

// Middleware to parse JSON request body

app.use(express.json());

// In-memory array to store users

let users = [];

// GET route to retrieve all users

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

res.json({

message: 'List of users',

users: users

});

});

// POST route to add a new user

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

const { name, email } = req.body;

if (!name || !email) {

return res.status(400).json({ error: 'Name and email are required.' });

}

const newUser = {

id: users.length + 1,

name,

email

};

users.push(newUser);

res.status(201).json({

message: 'User added successfully',

user: newUser

});

});

// Start the server

app.listen(port, () => {

console.log(`Server running at http://localhost:${port}`);

});

Step 3: Run the Server

node app.js

### **Test the API**

#### **🔹 GET Request**

Open your browser or use curl / Postman:

GET <http://localhost:3000/users>

#### **POST Request**

Use Postman or curl:

curl -X POST http://localhost:3000/users \

-H "Content-Type: application/json" \

-d '{"name": "Alice", "email": "alice@example.com"}'

### **Output Examples**

**GET /users** Response:

{

"message": "List of users",

"users": [

{

"id": 1,

"name": "Alice",

"email": "alice@example.com"

}

]

}

**POST /users** Response:

{

"message": "User added successfully",

"user": {

"id": 1,

"name": "Alice",

"email": "alice@example.com"

}

}

**5.Create RESTful routes (GET, POST, PUT, DELETE) for a user management system.**

### **Step 1: Setup Project**

If you haven’t already:

mkdir user-management-api

cd user-management-api

npm init -y

npm install express

Step 2: Create app.js with RESTful Routes

const express = require('express');

const app = express();

const port = 3000;

app.use(express.json()); // Middleware to parse JSON

// In-memory "database"

let users = [];

// GET all users

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

res.json({ users });

});

// GET a single user by ID

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

const user = users.find(u => u.id === parseInt(req.params.id));

if (!user) return res.status(404).json({ error: 'User not found' });

res.json(user);

});

// POST a new user

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

const { name, email } = req.body;

if (!name || !email) {

return res.status(400).json({ error: 'Name and email are required' });

}

const newUser = {

id: users.length + 1,

name,

email

};

users.push(newUser);

res.status(201).json({ message: 'User created', user: newUser });

});

// PUT (update) a user by ID

app.put('/users/:id', (req, res) => {

const id = parseInt(req.params.id);

const user = users.find(u => u.id === id);

if (!user) return res.status(404).json({ error: 'User not found' });

const { name, email } = req.body;

if (name) user.name = name;

if (email) user.email = email;

res.json({ message: 'User updated', user });

});

// DELETE a user by ID

app.delete('/users/:id', (req, res) => {

const id = parseInt(req.params.id);

const index = users.findIndex(u => u.id === id);

if (index === -1) return res.status(404).json({ error: 'User not found' });

const deletedUser = users.splice(index, 1);

res.json({ message: 'User deleted', user: deletedUser[0] });

});

// Start the server

app.listen(port, () => {

console.log(`User Management API running at http://localhost:${port}`);

});

| **Method** | **Route** | **Description** |
| --- | --- | --- |
| GET | /users | Get all users |
| GET | /users/:id | Get a specific user |
| POST | /users | Create a new user |
| PUT | /users/:id | Update an existing user |
| DELETE | /users/:id | Delete a user |

Example POST Body (JSON)

{

"name": "Alice",

"email": "alice@example.com"

}

To Run the App:

node app.js

6.Create a real-time chat application using **WebSocket and** [**Socket.io**](http://socket.io).

### **Requirements:**

* express — web framework
* socket.io — enables real-time bi-directional communication

Step 1: Initialize Project

mkdir socket-chat-app

cd socket-chat-app

npm init -y

npm install express socket.io

Step 2: Create Project Structure

socket-chat-app/

├── public/

│ └── index.html

└── server.js

Step 3: server.js — Express + Socket.io Server

const express = require('express');

const http = require('http');

const socketIo = require('socket.io');

const app = express();

const server = http.createServer(app);

const io = socketIo(server);

// Serve static files

app.use(express.static('public'));

// Handle socket connections

io.on('connection', (socket) => {

console.log(' New user connected');

socket.on('chat message', (msg) => {

io.emit('chat message', msg); // broadcast to all users

});

socket.on('disconnect', () => {

console.log('User disconnected');

});

});

// Start server

const PORT = 3000;

server.listen(PORT, () => {

console.log(`Chat server running at http://localhost:${PORT}`);

});

Step 4: public/index.html — Frontend Chat Page

<!DOCTYPE html>

<html>

<head>

<title>Socket.io Chat</title>

<style>

body { font-family: Arial; margin: 0; padding: 0; background: #f2f2f2; }

#chat { padding: 20px; max-width: 600px; margin: auto; }

#messages { list-style: none; padding: 0; }

#messages li { padding: 8px; margin-bottom: 5px; background: #fff; border-radius: 4px; }

#form { display: flex; }

#input { flex: 1; padding: 10px; border: 1px solid #ccc; border-radius: 4px; }

#send { padding: 10px 20px; margin-left: 10px; }

</style>

</head>

<body>

<div id="chat">

<h2>💬 Real-Time Chat</h2>

<ul id="messages"></ul>

<form id="form" action="">

<input id="input" autocomplete="off" placeholder="Type a message..." />

<button id="send">Send</button>

</form>

</div>

<script src="/socket.io/socket.io.js"></script>

<script>

const socket = io();

const form = document.getElementById('form');

const input = document.getElementById('input');

const messages = document.getElementById('messages');

form.addEventListener('submit', (e) => {

e.preventDefault();

if (input.value) {

socket.emit('chat message', input.value);

input.value = '';

}

});

socket.on('chat message', (msg) => {

const li = document.createElement('li');

li.textContent = msg;

messages.appendChild(li);

window.scrollTo(0, document.body.scrollHeight);

});

</script>

</body>

</html>

Run the App

node server.js