

**NANDHA ENGINEERING COLLEGE**

**(Autonomous Institution)**

Erode-638 052



**LAB RECORD**

**22CSP09 - FULL STACK DEVELOPMENT**

**LABORATORY**

**V - Semester**

**B.E COMPUTER SCIENCE & ENGINEERING**

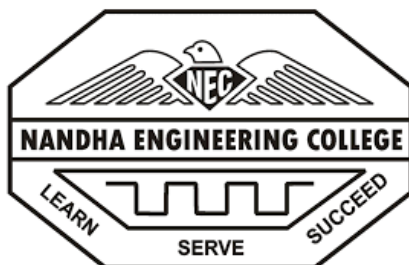
Department of Computer Science and Engineering

**NANDHA ENGINEERING COLLEGE, ERODE-52**

# NANDHA ENGINEERING COLLEGE

(Autonomous Institution)

Erode-638 052



## BONAFIDE CERTIFICATE

**REGISTER NUMBER:**

Certified that this is the Bonafide Record of work done  
by.....of the V Semester **B.E–  
COMPUTER SCIENCE AND ENGINEERING** branch during the  
Academic Year 2024 – 2025 in the **22CSP09 - FULL STACK  
DEVELOPMENT LABORATORY.**

.....

Faculty incharge

.....

Head of the Department

Submitted for the End Semester Practical Examination Held on .....

.....

Internal Examiner

.....

External Examiner

# INDEX

EX. NO	DATE	TITLE OF THE EXPERIMENT	PAGE NO.	MARKS	SIGN
1		Build a Basic React APP that display custom message from users	2		
2		Create a Login form using React JS	6		
3		Write a program to upload Single/Multiple images to cloundinary using Node JS	11		
4		Write a program to create router using Node.js with Express	14		
5		Design a program to create Single Responsive Page using Bootstrap	20		
6		Establish Connection to MongoDB with Node JS	24		
7		Performing CRUD operations to MongoDB with Node.js	29		
<b>ADDITIONAL EXERCISES</b>					
8		Creating a Node.js Server with Routes for an E-commerce Application	32		
9		Setting up a Node.js Server to Handle Bookstore Requests	37		
10		Creating a Node.js Web Server to Handle Dynamic Content and Serve Files	40		
11		Developing a User Registration and Login System Using Express.js	44		
<b>AVERAGE MARKS AWARDED</b>					

## **NODE JS :**

**To install Node.js, follow these steps based on your operating system:**

### **1. Install on Windows:**

#### **Download the Installer:**

- Visit the official Node.js website.
- Download the LTS (Long-Term Support) version for most stable features.

#### **Run the Installer:**

- Open the downloaded .msi file.
- Follow the installation prompts, ensuring you check the option to install npm (Node Package Manager) and add Node.js to the system PATH.

#### **Verify Installation:**

- Open Commands Prompt and run:  
**node -v**
- This displays the installed Node.js version.

#### **Run:**

**npm -v**

- This checks the npm version.

## **REACT JS:**

**To install a simple React app, follow these steps:**

### **1. Install Node.js**

**Ensure Node.js is installed (required for npm/yarn). Verify by running:**

`node -v`

`npm -v`

### **2. Create a New React App**

#### **Using Create React App (CRA):**

- Open your terminal and run:  
**npx create-react-app my-app**
- Replace my-app with your project name.
- Navigate to the project folder:

**cd my-app**

### **3. Start the Development Server**

- Run the app:  
**npm start**
- This starts a local server at <http://localhost:3000>.

**AIM:**

To build a basic React app that allows users to submit custom messages and displays each submitted message in a list format.

**PROCEDURE:**

- Initialize a React app using `create-react-app`.
- Create an input form with a button for users to submit messages.
- Store each message in the React state and update the list each time a new message is submitted.
- Display all submitted messages in a list below the form.
- Optionally, style the form and message list for better user experience.

**CODING:****1)Setup Project**

Run the following commands in your terminal to create a React app:

```
npx create-react-app message-app  
cd message-app
```

**2)App Component**

Replace the content of **App . js** with the following code:

```
// App.js  
import React, { useState } from 'react';  
import './App.css';  
  
function App() {  
  const [message, setMessage] = useState("");  
  const [messages, setMessages] = useState([]);  
  
  // Function to handle message submission  
  const handleSubmit = (e) => {
```

```

    e.preventDefault();
    if (message.trim()) {
      setMessages([...messages, message]);
      setMessage(""); // Clear the input field
    }
  };

  return (
    <div className="App">
      <h1>Message Display App</h1>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Enter your message"
          value={message}
          onChange={(e) => setMessage(e.target.value)}
        />
        <button type="submit">Submit</button>
      </form>
      <div>
        <h2>Messages</h2>
        <ul>
          {messages.map((msg, index) => (
            <li key={index}>{msg}</li>
          ))}
        </ul>
      </div>
    </div>
  );
}

export default App;

```

### 3)Styling:

Add the following CSS to `App.css` for basic styling:

**/\* App.css \*/**

```

.App {
  text-align: center;
  padding: 20px;
}

input[type="text"] {
  padding: 10px;
  margin-right: 10px;
}

```

```
width: 200px;
}

button {
  padding: 10px;
}

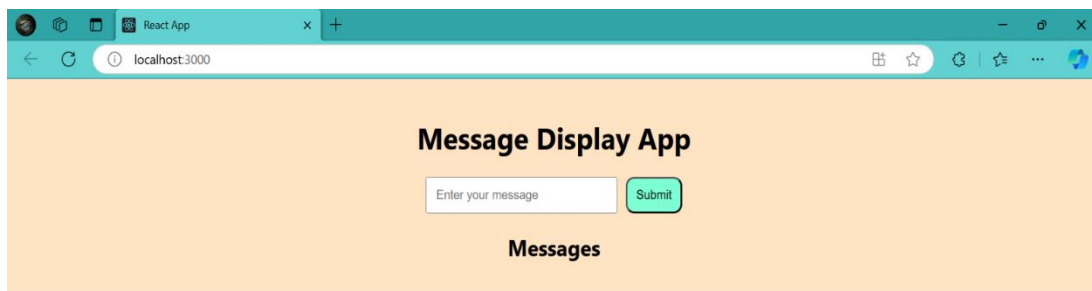
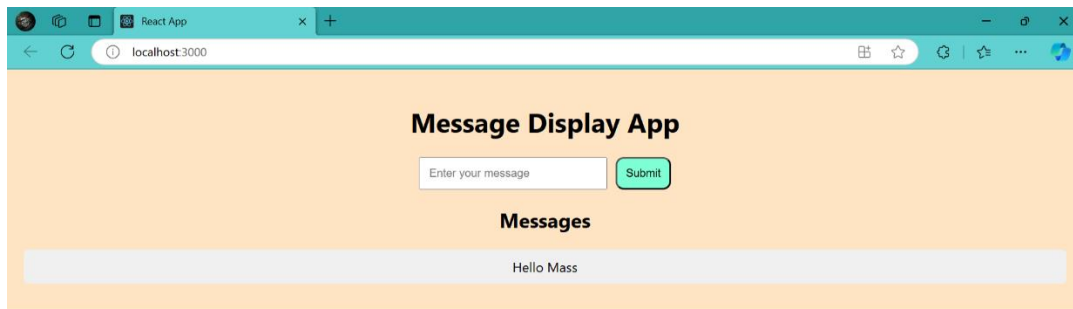
ul {
  list-style-type: none;
  padding: 0;
}

li {
  background: #f0f0f0;
  margin: 5px 0;
  padding: 10px;
  border-radius: 5px;
}
```

#### 4)Run the App

Start the React app:

### OUTPUT:



<b>MARK ALLOCATION</b>		
Preparation & conduct of experiment	50	
Observation & Result	30	
Record	10	
Viva voice	10	
Total	100	

## **RESULT:**

The app now successfully captures and displays custom messages from users in a clear and organized list. Each new message appears instantly after submission, demonstrating a functional React application for displaying user-generated content.

**FACULTY SIGNATURE**



**AIM:**

To create a reusable login form component in React that captures username and password inputs.

**Procedure:**

1. Initialize a React app or add this component to an existing app.
2. Create a `LoginForm` component with input fields for username and password, and a submit button.
3. Handle the form submission and validation.
4. Optionally, style the form for better user experience.

**CODING:****1)Setup Project**

If you haven't already, create a new React project:

```
npx create-react-app login-app
```

```
cd login-app
```

**2)Create Login Form Component**

Inside the src directory, create a new file called **LoginForm.js** and add the following code:

```
// LoginForm.js
import React, { useState } from 'react';

function LoginForm() {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");

  // Handle form submission
  const handleSubmit = (e) => {
    e.preventDefault();
    // Basic validation
    if (!username || !password) {
      setError('Please enter both username and password');
    }
  };
}
```

```

    return;
  }

  setError(""); // Clear error
  console.log('Username:', username);
  console.log('Password:', password);
  // Further authentication logic here
};

return (
  <div className="login-form">
    <h2>Login</h2>
    {error && <p className="error">{error}</p>}
    <form onSubmit={handleSubmit}>
      <div>
        <label>Username:</label>
        <input
          type="text"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          placeholder="Enter username"
        />
      </div>
      <div>
        <label>Password:</label>
        <input
          type="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          placeholder="Enter password"
        />
      </div>
      <button type="submit">Login</button>
    </form>
  </div>
);
}

export default LoginForm;

```

### 3) Import and Use LoginForm in App Component

In `App.js`, import and use the `LoginForm` component:

## // App.js

```
import React from 'react';
import LoginForm from './LoginForm';

function App() {
  return (
    <div className="App">
      <h1>Welcome to the Login Page</h1>
      <LoginForm />
    </div>
  );
}

export default App;
```

## /\* App.css \*/

```
.login-form {
  width: 300px;
  margin: auto;
  padding: 20px;
  border: 1px solid #ccc;
  border-radius: 10px;
}

.login-form h2 {
  margin-bottom: 20px;
}

.login-form label {
  display: block;
  margin-top: 10px;
}

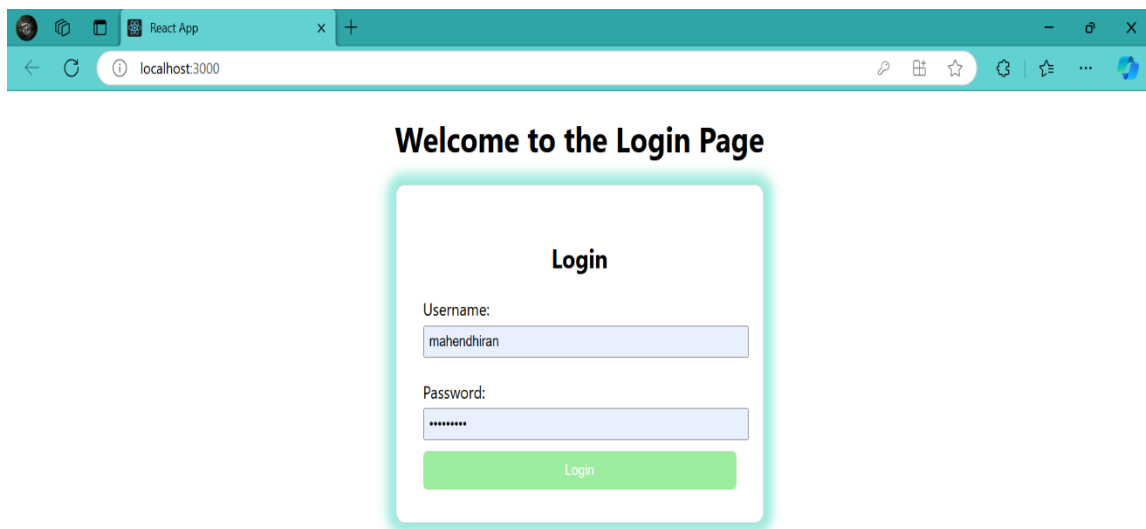
.login-form input {
  width: 100%;
  padding: 8px;
  margin-top: 5px;
  margin-bottom: 10px;
}

.login-form button {
  width: 100%;
```

```
padding: 10px;
background-color: #4CAF50;
color: white;
border: none;
border-radius: 5px;
cursor: pointer;
}

.error {
  color: red;
}
```

## OUTPUT:





## Welcome to the Login Page

**Login**

**Username:**

**Password:**

Login

MARK ALLOCATION		
Preparation & conduct of experiment	50	
Observation & Result	30	
Record	10	
Viva voice	10	
Total	100	

## RESULT:

The functional and reusable login form component in React. It captures username and password inputs, performs basic validation, and can be easily customized for more advanced authentication.

**FACULTY SIGNATURE**

**AIM:**

To create a router in Node.js using Express that handles various endpoints, such as home, about, and contact.

**PROCEDURE**

1. Set up a Node.js and Express project.
2. Create the main server file to initialize Express.
3. Define routes using Express Router.
4. Use the router in the server file to handle different endpoints.

**CODING:****1)Setup Project**

Initialize a Node.js project and install Express:

```
mkdir express-router-app
```

```
cd express-router-app
```

```
npm init -y
```

```
npm install express
```

**2)Create the Server File**

In the project root, create a file called `server.js`. This file will set up the Express server and use the router.

**// server.js**

```
const express = require('express');  
const app = express();  
const port = 3000;
```

```
const mainRouter = require('./routes/mainRoutes');
```

```
app.use('/', mainRouter);
```

```
app.listen(port, () => {
```

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

## Create the Router

Inside the project, create a **routes directory**, and inside it, create a file named **mainRoutes.js**. Define routes for different endpoints in this file using Express Router.

**// routes/mainRoutes.js**

```
const express = require('express');  
const router = express.Router();  
  
router.get('/', (req, res) => {  
  res.send('Welcome to the Home Page');  
});  
  
router.get('/about', (req, res) => {  
  res.send('Welcome to the About Page');  
});  
  
router.get('/contact', (req, res) => {  
  res.send('Welcome to the Contact Page');  
});  
  
module.exports = router;
```

## OUTPUT:





Welcome to the About Page

MARK ALLOCATION		
Preparation & conduct of experiment	50	
Observation & Result	30	
Record	10	
Viva voice	10	
Total	100	

## RESULT:

The successfully created an Express router in Node.js that organizes and handles requests for different routes. This structure allows you to add more routes easily and organize them into different modules if needed.

**FACULTY SIGNATURE**



**AIM:**

To upload single or multiple images to Cloudinary using Node.js with Express, multer for handling file uploads, and Cloudinary's SDK for cloud storage.

**PROCEDURE**

1. Set up a Node.js project and install necessary packages.
2. Configure Cloudinary with your credentials.
3. Create routes to handle single and multiple image uploads.
4. Use multer to handle file uploads and Cloudinary SDK to upload them to Cloudinary.

**CODING :**

```
require('dotenv').config();
const http = require('http');
const cloudinary = require('cloudinary').v2;
const formidable = require('formidable');
const fs = require('fs');
const path = require('path');
cloudinary.config({
  cloud_name: 'dqdl7sugb',
  api_key: '281571532867932',
  api_secret: 'Hrxv6UfwJokArW_cgf8ZViIu5S4',
  secure: true,
});

const server = http.createServer((req, res) => {
  if (req.method.toLowerCase() === 'post') {
    const form = new formidable.IncomingForm({ multiples: true });
```

```

form.parse(req, async (err, fields, files) => {
  if (err) {
    console.error('Formidable Error:', err);
    res.writeHead(400, { 'Content-Type': 'text/plain' });
    res.end('Error parsing the files.');
    return;
  }
  const uploadedFiles = Array.isArray(files.image) ? files.image :
[files.image];
  if (!uploadedFiles || uploadedFiles.length === 0) {
    res.writeHead(400, { 'Content-Type': 'text/plain' });
    res.end('No files were uploaded.');
    return;
  }
  try {
    const uploadPromises = uploadedFiles.map((file) => {
      const mimeType = file.mimetype || file.type;
      if (!mimeType.startsWith('image/')) {
        throw new Error('Only image files are allowed.');
      }

      return cloudinary.uploader.upload(file.filepath, {
        resource_type: 'image',
        folder: 'uploads',
      });
    });

    const results = await Promise.all(uploadPromises);
    const urls = results.map(result => result.secure_url);
  }
});

```

```

    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify({
      message: 'Images uploaded successfully!',
      urls: urls
    }));
  } catch (error) {
    console.error('Error uploading to Cloudinary:', error);
    res.writeHead(500, { 'Content-Type': 'text/plain' });
    res.end('Error uploading to Cloudinary. ' + error.message);
  }
});
} else {
  res.writeHead(200, { 'Content-Type': 'text/html' });
  res.end(`
    <!DOCTYPE html>
    <html>
      <head>
        <title>Upload Image(s)</title>
      </head>
      <body>
        <h2>Upload Single or Multiple Images to Cloudinary</h2>
        <form action="/" enctype="multipart/form-data" method="post">
          <input type="file" name="image" multiple><br><br>
          <button type="submit">Upload Image(s)</button>
        </form>
      </body>
    </html>
  `);
}

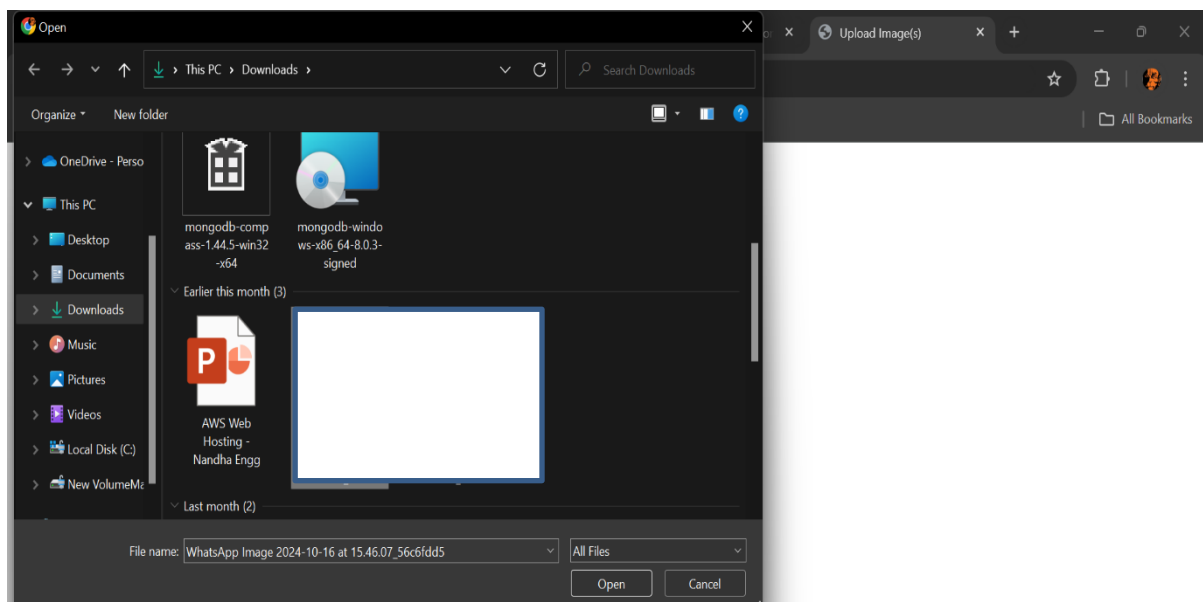
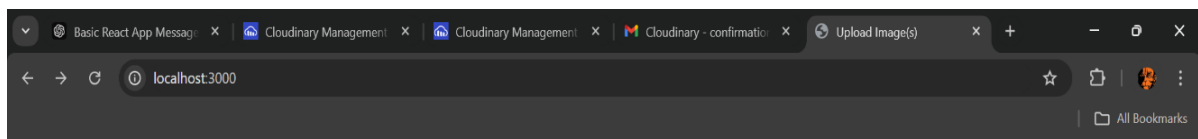
```

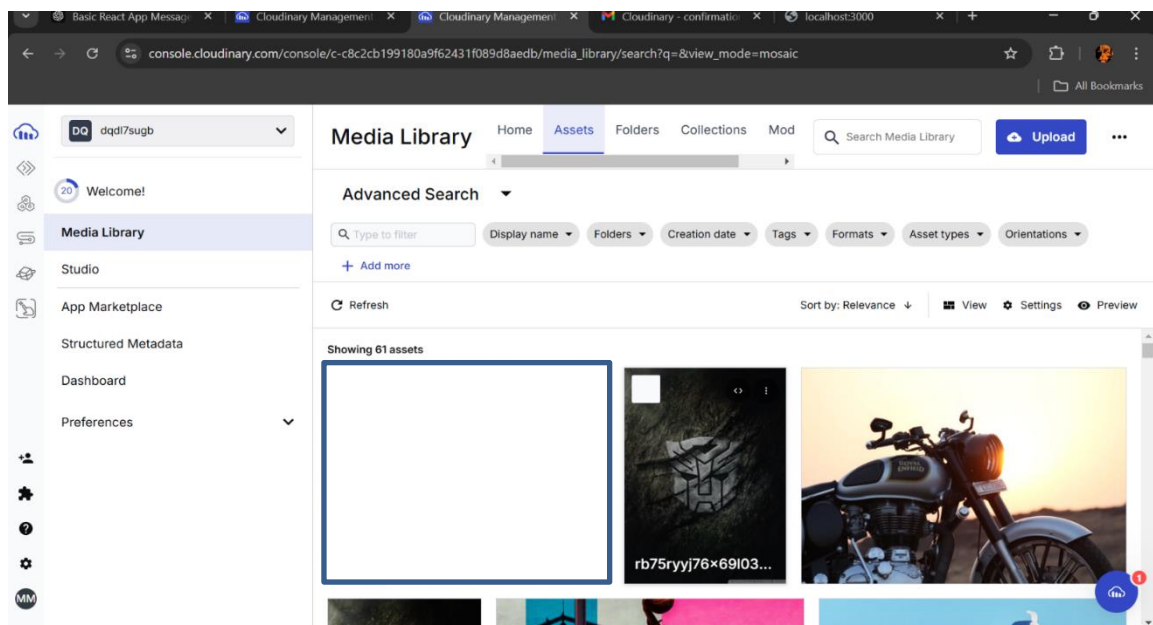
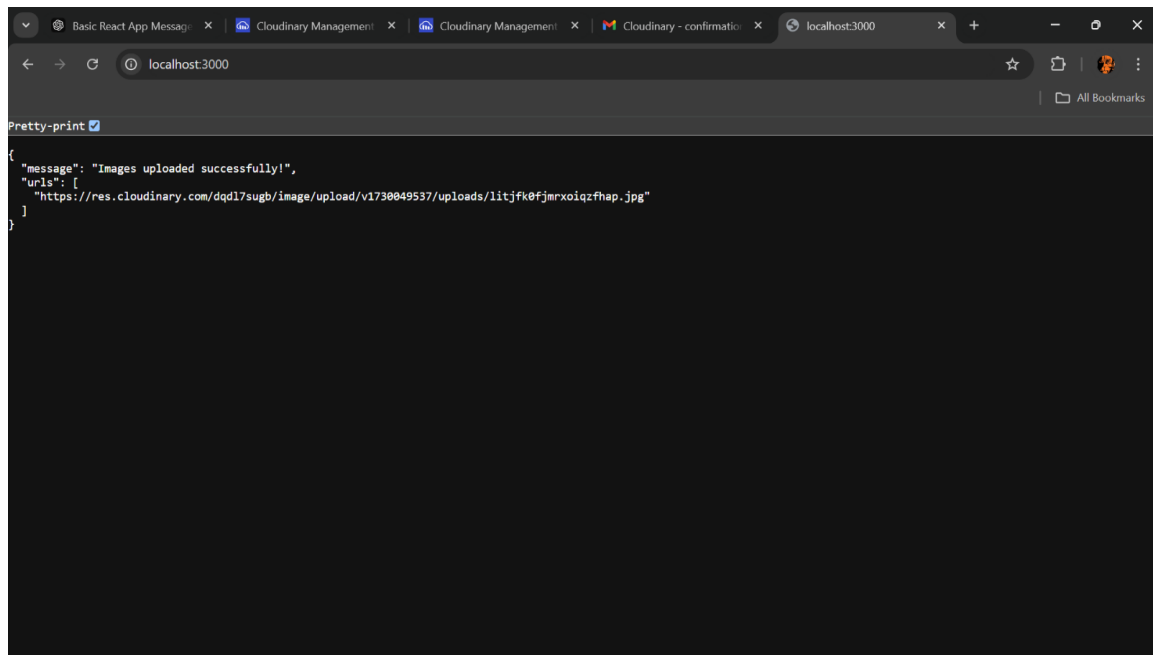
```
});

const PORT = process.env.PORT || 3000;

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

## OUTPUT:





<b>MARK ALLOCATION</b>		
Preparation & conduct of experiment	50	
Observation & Result	30	
Record	10	
Viva voice	10	
Total	100	

## **RESULT:**

The created a Node.js application to handle single and multiple image uploads to Cloudinary using Express, multer, and Cloudinary's SDK. Each uploaded image is stored in Cloudinary, and the server returns the image URLs in response.

**FACULTY SIGNATURE**

**AIM:**

To design a single responsive webpage using Bootstrap that is visually appealing and functional.

**PROCEDURE:**

1. Set up a basic HTML structure.
2. Include Bootstrap CSS and JS files.
3. Create a responsive navigation bar.
4. Add a hero section.
5. Create a section for features or content.
6. Add a footer.
7. Ensure responsiveness using Bootstrap's grid system and utility classes.

**CODING:**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Simple Responsive Page</title>
  <link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.mi
n.css" rel="stylesheet">
  <style>
    body {
      padding-top: 56px; /* Space for fixed navbar */
    }
    .hero {
      background: #007bff; /* Blue background */
      color: white;
      height: 300px;
      display: flex;
      align-items: center;
      justify-content: center;
      text-align: center;
```

```

    }
  </style>
</head>
<body>

  <!-- Navigation Bar -->
  <nav class="navbar navbar-expand-lg navbar-light bg-light fixed-top">
    <div class="container">
      <a class="navbar-brand" href="#">My Simple Page</a>
      <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav"
aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav ml-auto">
          <li class="nav-item">
            <a class="nav-link" href="#about">About</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#contact">Contact</a>
          </li>
        </ul>
      </div>
    </div>
  </nav>

  <!-- Hero Section -->
  <div class="hero">
    <h1>Welcome to My Simple Page</h1>
    <p>This is a simple responsive webpage using Bootstrap.</p>
  </div>

  <!-- About Section -->
  <div class="container my-5" id="about">
    <h2 class="text-center">About Us</h2>
    <p class="text-center">We are dedicated to providing the best
services to our customers.</p>
  </div>

  <!-- Contact Section -->
  <div class="container my-5" id="contact">
    <h2 class="text-center">Contact Us</h2>
    <form>

```

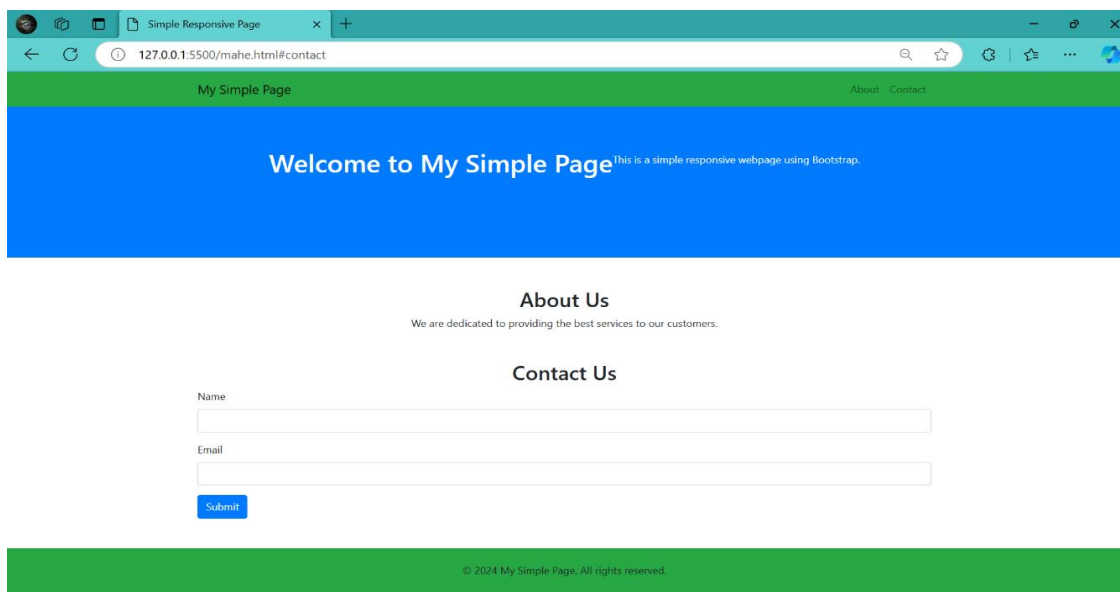


```

<div class="form-group">
  <label for="name">Name</label>
  <input type="text" class="form-control" id="name" required>
</div>
<div class="form-group">
  <label for="email">Email</label>
  <input type="email" class="form-control" id="email" required>
</div>
<button type="submit" class="btn btn-primary">Submit</button>
</form>
</div>
<!-- Footer -->
<footer class="text-center py-4">
  <div class="container">
    <p>&copy; 2024 My Simple Page. All rights reserved.</p>
  </div>
</footer>
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper
.min.js"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>

```

## OUTPUT:



<b>MARK ALLOCATION</b>		
Preparation & conduct of experiment	50	
Observation & Result	30	
Record	10	
Viva voice	10	
Total	100	

## **RESULT:**

This simple responsive page is fully functional and adapts to different screen sizes thanks to Bootstrap's responsive grid system. You can customize the content, styles, and sections further based on your needs.

**FACULTY SIGNATURE**

**AIM:**

To perform CRUD operations on MongoDB compass and connect to MongoDB through node.js

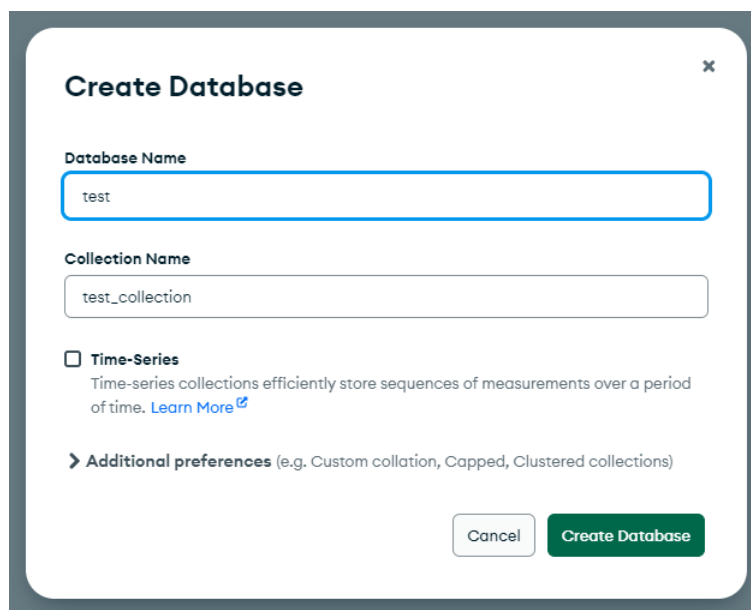
**PROCEDURE:**

1. Create Database: Open MongoDB Compass, click "Create Database", name it test, and add a collection (e.g., test\_collection).
2. Insert Data: Go to test\_collection, click "Insert Document", and add:  

```
{ "name": "John Doe", "age": 19, "department": "Computer Science" }
```
3. Update Data: Click the pencil icon, modify fields (e.g., change age to 20), and save changes.
4. Delete Data: Select a document, click the trash icon, and confirm deletion.
5. To connect MongoDB from node.js, install package using **npm install mongodb**.
6. Insert some temporary data and connect to MongoDB using the connection string through node.js.
7. View the temporary data by logging all the data into console.

**PROCESS:**

- 1.



**Create Database**

Database Name  
test

Collection Name  
test\_collection

☐ **Time-Series**  
Time-series collections efficiently store sequences of measurements over a period of time. [Learn More](#)

[Additional preferences](#) (e.g. Custom collation, Capped, Clustered collections)

Cancel Create Database

2.

### Insert Document

To collection test.test\_collection

VIEW { } ≡

```
1  /**
2  * Paste one or more documents here
3  */
4  {
5  ▾  "_id": {
6    "$oid": "6741dd1cd827e777d4ed6026"
7  },
8    "name" : "John Doe",
9    "age" : 19,
10   "department": "Computer Science"
11  }
12
```

Cancel Insert

### Insert Document

To collection test.test\_collection

VIEW { } ≡

```
1  _id: ObjectId('6741dd1cd827e777d4ed6026')
2  name : "John Doe"
3  age : 19
4  department : "Computer Science"
```

ObjectId

String

Int32

String

Cancel Insert

3.

+ ↶ ✎ 🗑

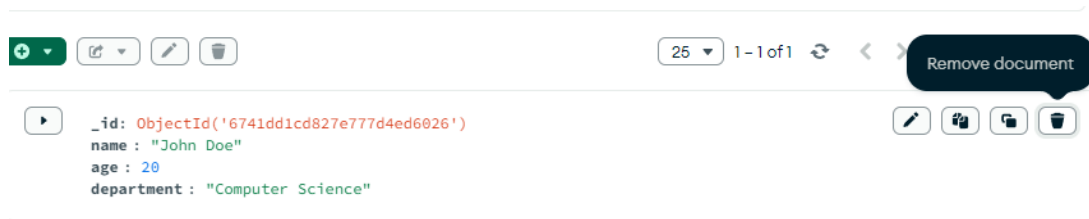
25 ▾ 1 - 1 of 1 ↺ Edit document { } ≡

▶

```
_id: ObjectId('6741dd1cd827e777d4ed6026')
name : "John Doe"
age : 19
department : "Computer Science"
```

✎ ↶ ↷ 🗑

4.



5.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS E:\Development\MERN> npm install mongodb

added 12 packages, and audited 87 packages in 1s

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

found 0 vulnerabilities
PS E:\Development\MERN> |
```

## 6. CODE:

```
const { MongoClient } = require('mongodb');

const uri = "mongodb://localhost:27017";

const databaseName = "test";

const collectionName = "test_collection";

async function fetchData() {

  const client = new MongoClient(uri);

  try {

    await client.connect();

    console.log("Connected to MongoDB!");

    const db = client.db(databaseName);

    const collection = db.collection(collectionName);
```

```
    const documents = await collection.find({}).toArray();

    console.log(documents);

  } catch (error) {

    console.error("Error connecting to MongoDB:", error);

  } finally {

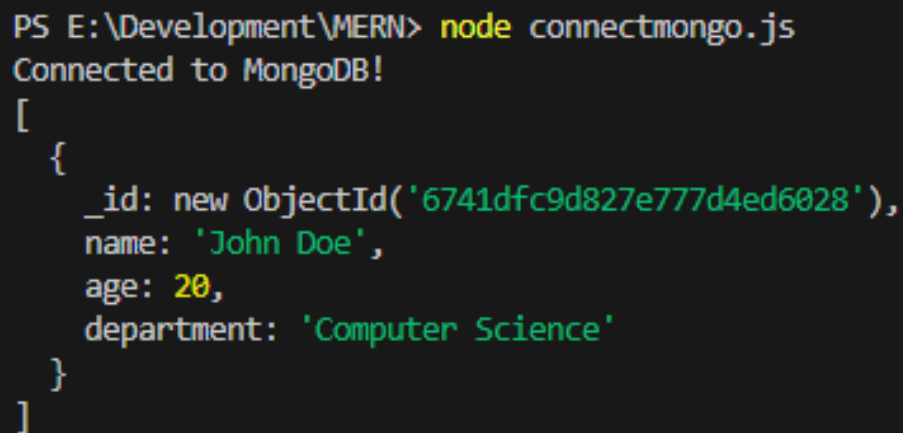
    await client.close();

  }

}

fetchData();
```

## OUTPUT:

A terminal window with a dark background and light-colored text. The prompt is 'PS E:\Development\MERN>'. The command 'node connectmongo.js' is entered. The output is 'Connected to MongoDB!' followed by a JSON array containing one object. The object has fields: '\_id' with a new ObjectId value, 'name' as 'John Doe', 'age' as 20, and 'department' as 'Computer Science'.

```
PS E:\Development\MERN> node connectmongo.js
Connected to MongoDB!
[
  {
    _id: new ObjectId('6741dfc9d827e777d4ed6028'),
    name: 'John Doe',
    age: 20,
    department: 'Computer Science'
  }
]
```

MARK ALLOCATION		
Preparation & conduct of experiment	50	
Observation & Result	30	
Record	10	
Viva voice	10	
Total	100	

## RESULT:

CRUD operations were successfully performed on **MongoDB Compass**, including creating a database, inserting, updating, and deleting documents. A Node.js script connected to MongoDB using the **mongodb** package, inserted temporary data, and logged it to the console, verifying successful integration.

**FACULTY SIGNATURE**

**AIM:**

To perform CRUD operations on MongoDB using Node.js and view the results via the console.

**PROCEDURE:**

1. Connect to MongoDB: Connects to MongoDB using the MongoClient.
2. Insert Data: Inserts a document ({ name: "Rahul", age: 21, department: "Information Technology" }) into the test\_collection.
3. Read Data: Fetches and logs all documents in the collection using .find({}).toArray().
4. Update Data: Updates the age field of Rahul from 21 to 19.
5. Delete Data: Deletes the document where name: "Rahul"

**CODE:**

```
const { MongoClient } = require('mongodb');

const uri = "mongodb://localhost:27017"; // MongoDB connection string

const databaseName = "test";

const collectionName = "test_collection";

async function performCRUD() {

  const client = new MongoClient(uri);

  try {

    // Connect to MongoDB

    await client.connect();

    console.log("Connected to MongoDB!");

    const db = client.db(databaseName);

    const collection = db.collection(collectionName);
```



```

    const insertResult = await collection.insertOne({ name: "Rahul", age: 21,
department: "Computer Science" });

    console.log("Insert Acknowledgement:", insertResult.acknowledged);

    console.log("All Documents:", await collection.find({ }).toArray());

    const updateResult = await collection.updateOne({ name: "Rahul" }, {
    $set: { age: 19 } });

    console.log("Updated Document Count:", updateResult.modifiedCount);

    console.log("All Documents:", await collection.find({ }).toArray());

    const deleteResult = await collection.deleteOne({ name: "Rahul" });

    console.log("Deleted Document Count:", deleteResult.deletedCount);

} catch (error) {

    console.error("Error performing CRUD operations:", error);

} finally {

    await client.close();

}} performCRUD();

```

## OUTPUT:

```

PS E:\Development\MERN> node connectmongo.js
Connected to MongoDB!
Insert Acknowledgement: true
All Documents: [
  {
    _id: new ObjectId('6741eb1a78a9181821cb565b'),
    name: 'Rahul',
    age: 21,
    department: 'Computer Science'
  }
]
Updated Document Count: 1
All Documents: [
  {
    _id: new ObjectId('6741eb1a78a9181821cb565b'),
    name: 'Rahul',
    age: 19,
    department: 'Computer Science'
  }
]
Deleted Document Count: 1

```

MARK ALLOCATION		
Preparation & conduct of experiment	50	
Observation & Result	30	
Record	10	
Viva voice	10	
Total	100	

### **RESULT:**

CRUD operations on MongoDB using Node.js were successfully performed and verified through the console.

**FACULTY SIGNATURE**

## **AIM**

To create a simple Node.js server for an e-commerce application with routes for managing products, orders, and users.

## **PROCEDURE**

1. Set Up Environment: Install Node.js and create a project folder.
2. Initialize the Project: Use npm init to initialize a Node.js project and install dependencies like express for building the server.
3. Create Routes:
4. Define routes for handling product-related requests (/products).
5. Define routes for user-related actions (/users).
6. Define routes for order management (/orders).
7. Start the Server: Use the listen method to run the server on a specified port.
8. Test the Routes: Use a browser or tools like Postman to check the functionality of each route.

## **CODING:**

Install Dependencies

```
npm init -y
```

```
npm install express
```

```
//Index.js
```

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

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

```
// Middleware to parse JSON
```

```
app.use(express.json());
```

```
// Sample Users
```

```
let users = [
```

```
    { id: 1, name: "Alice Johnson", email: "alice.johnson@example.com",  
      address: "123 Main St, Springfield", phone: "123-456-7890" }
```

```
];
```

```
// Sample Products
```

```
let products = [
```

```
    { id: 101, name: "Smartphone", description: "Latest 5G smartphone  
    with 128GB storage", price: 599.99, category: "Electronics", stock: 50 }
```

```
];
```

```
// Sample Orders
```

```
let orders = [
```

```
    { orderId: 1001, userId: 1, products: [{ productId: 101, quantity: 1 }, {  
      productId: 103, quantity: 2 }], totalAmount: 899.97, status: "Shipped",  
      orderDate: "2024-11-20" }
```

```
];
```

```
// Product Routes
```

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

```
    res.json({ message: "List of products", data: products });
```

```
});
```

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

```
    const product = req.body;
```

```
    products.push(product);
```

```
    res.json({ message: "Product added", product });
```

```
});
```

```
// User Routes
```

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

```
    res.json({ message: "List of users", data: users });
  });

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

    const user = req.body;

    users.push(user);

    res.json({ message: "User registered", user });

  });

  // Order Routes

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

    res.json({ message: "List of orders", data: orders });

  });

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

    const order = req.body;

    orders.push(order);

    res.json({ message: "Order placed", order });

  });

  // Start Server

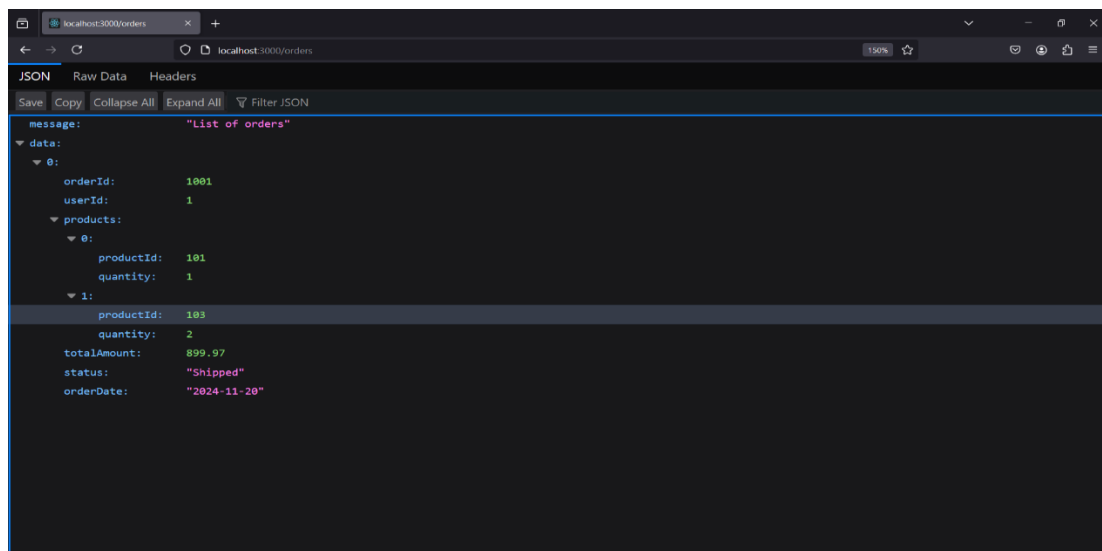
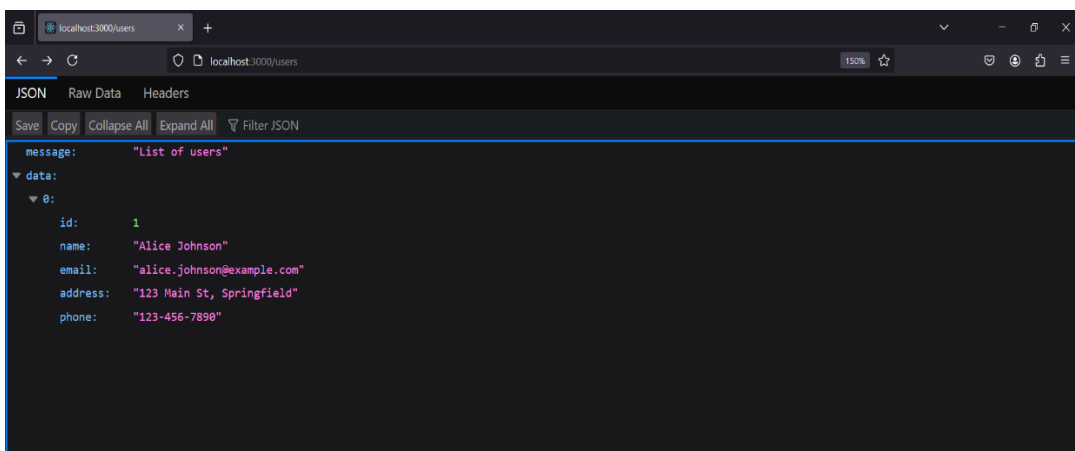
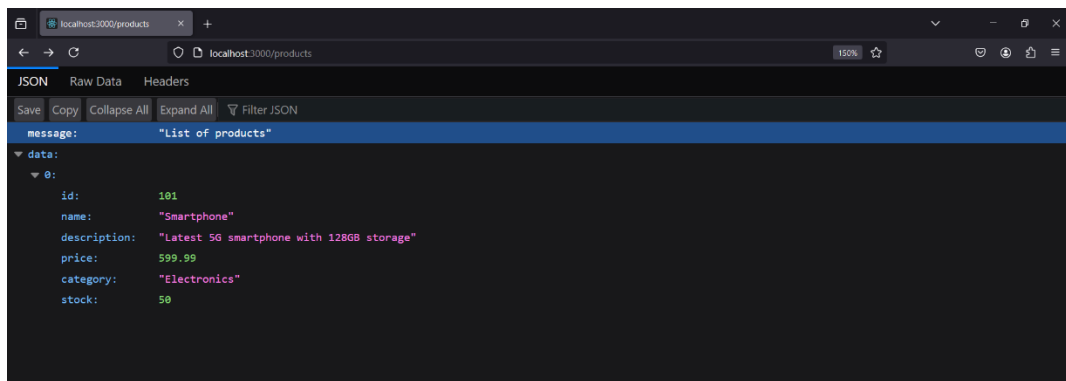
  const PORT = 3000;

  app.listen(PORT, () => {

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

  });
```

## OUTPUT:



MARK ALLOCATION		
Preparation & conduct of experiment	50	
Observation & Result	30	
Record	10	
Viva voice	10	
Total	100	

### **RESULT:**

A working Node.js server for an e-commerce application has been created. It includes routes to manage products, users, and orders, and provides responses for each request successfully.

**FACULTY SIGNATURE**

### AIM

To build a Node.js server for handling basic bookstore operations, including retrieving book information and adding new books.

### PROCEDURE:

1. Install Node.js from the official website.
2. Create a project folder and initialize it with npm init.
3. Install required packages like express and body-parser.
4. Write code for the server with basic GET and POST routes.
5. Start the server and test the endpoints using a browser or Postman.

### CODING:

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const PORT = 3000;

// Middleware to parse JSON
app.use(bodyParser.json());
// In-memory "database" for books
let books = [
  { id: 1, title: 'To Kill a Mockingbird', author: 'Harper Lee' },
  { id: 2, title: '1984', author: 'George Orwell' },
];

// GET route to retrieve all books
app.get('/books', (req, res) => {
  res.status(200).json(books);
});

// POST route to add a new book
app.post('/books', (req, res) => {
  const { title, author } = req.body;
  if (!title || !author) {
```



```

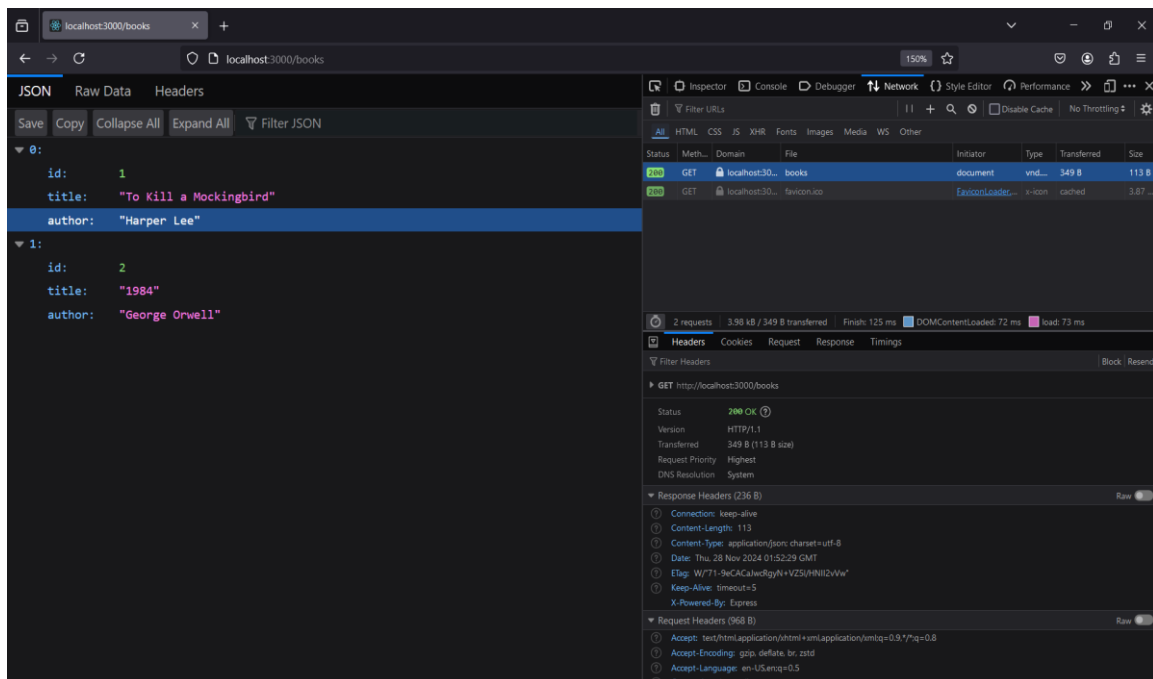
    return res.status(400).json({ error: 'Title and Author are required' });
  }
  const newBook = { id: books.length + 1, title, author };
  books.push(newBook);
  res.status(201).json(newBook);
});

// Root endpoint
app.get('/', (req, res) => {
  res.send('Welcome to the Simple Bookstore API!');
});

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

```

## OUTPUT:



<b>MARK ALLOCATION</b>		
Preparation & conduct of experiment	50	
Observation & Result	30	
Record	10	
Viva voice	10	
Total	100	

### **RESULT:**

This exercise demonstrates the ability to set up a Node.js server with basic functionality to manage a bookstore's book inventory.

**FACULTY SIGNATURE**

**AIM:**

To create a Node.js web server that handles dynamic content and serves static files.

**PROCEDURE:**

1. Install Node.js\*\*: Ensure Node.js is installed on your system.
2. Setup Project\*\*: Create a new folder for the project and initialize it using `npm init -y`.
3. Install Dependencies\*\*: Install the `http` and `fs` modules (built-in in Node.js).
4. Write the Server Code\*\*: Use the `http` module to create a server and the `fs` module to serve static files.
5. Add Dynamic Content\*\*: Incorporate dynamic content by responding with data based on the request URL.
6. Run the Server\*\*: Start the server using `node server.js`.
7. Test the Server\*\*: Access the server via a browser or Postman using `http://localhost:3000`.

**CODE:**

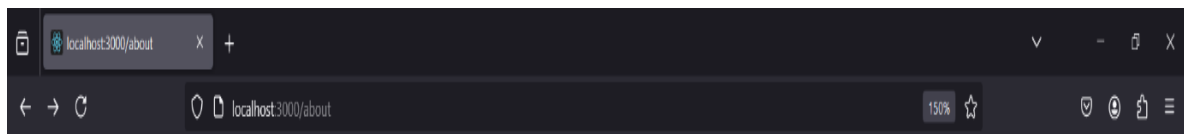
```
javascript
const http = require('http');
const fs = require('fs');
const path = require('path');

// Create server
const server = http.createServer((req, res) => {
  if (req.url === '/') {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.end('<h1>Welcome to the Node.js Web Server</h1>');
  } else if (req.url === '/about') {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.end('<h1>About Us</h1><p>This is a Node.js web server
example.</p>');
  } else if (req.url === '/file') {
    const filePath = path.join(__dirname, 'example.txt');
    fs.readFile(filePath, (err, data) => {
      if (err) {
```

```
        res.writeHead(404, { 'Content-Type': 'text/plain' });
        res.end('File not found');
    } else {
        res.writeHead(200, { 'Content-Type': 'text/plain' });
        res.end(data);
    }
});
} else {
    res.writeHead(404, { 'Content-Type': 'text/html' });
    res.end('<h1>404 Not Found</h1>');
}
});

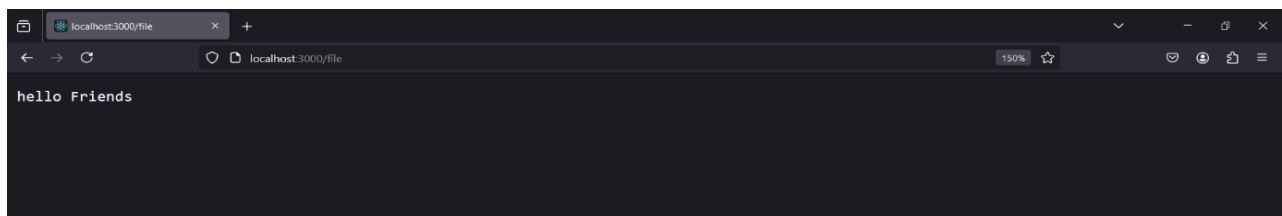
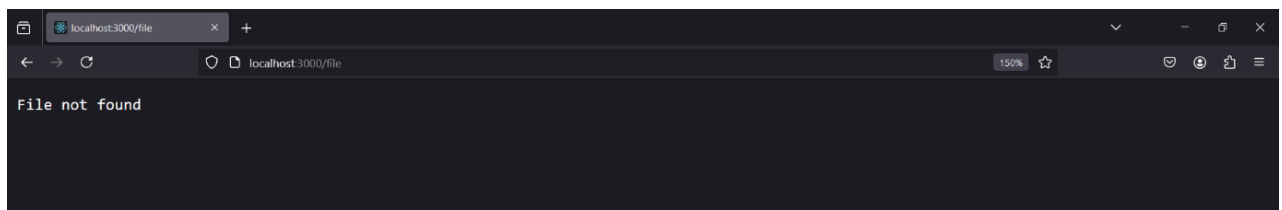
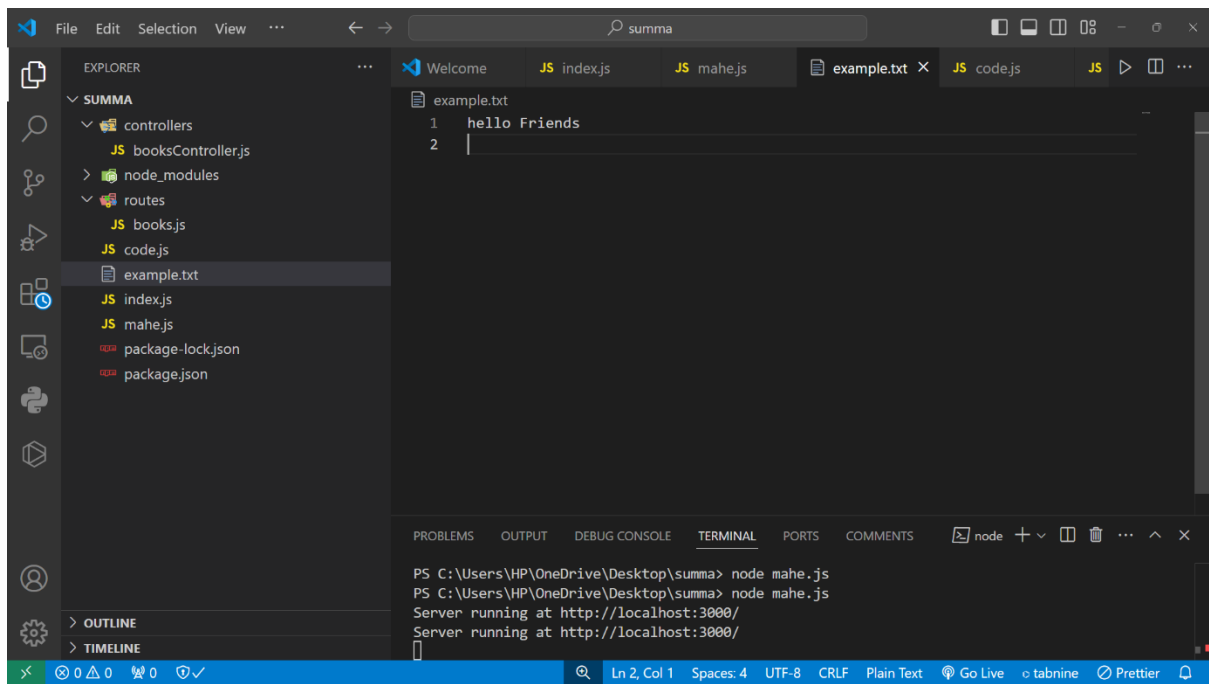
// Start server
server.listen(3000, () => {
    console.log('Server running at http://localhost:3000/');
});
```

## OUTPUT:



## About Us

This is a Node.js web server example.



MARK ALLOCATION		
Preparation & conduct of experiment	50	
Observation & Result	30	
Record	10	
Viva voice	10	
Total	100	

## RESULT:

The Node.js web server was successfully created. It handles dynamic content and serves static files, demonstrating the ability to handle multiple routes and file system operations.

**FACULTY SIGNATURE**

**AIM:**

To develop a user registration and login system using Express.js with secure password handling and session management.

**PROCEDURE:**

Install Dependencies:

Initialize the project and install required modules:

**npm init -y**

**npm install express body-parser bcryptjs express-session**

Setup Project Structure:

Create the following files:

server.js (main server script)

users.json (to simulate a database for user data)

Server Configuration:

Configure the Express server to handle routes for registration and login.

Password Hashing:

Use bcryptjs to hash passwords securely during registration and verify them during login.

Session Management:

Use express-session to manage user sessions after login.

Run and Test:

Start the server and test registration and login endpoints using Postman or a browser.

## **CODING:**

### **//Server.js**

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

const bodyParser = require('body-parser');

const bcrypt = require('bcryptjs');

const session = require('express-session');

const fs = require('fs');

const path = require('path');


const app = express();

const PORT = 3000;


// Middleware

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

app.use(

  session({

    secret: 'secret-key',

    resave: false,

    saveUninitialized: false,

  })

);

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



```

// File paths

const usersFile = './users.json';

// Helper functions to read/write users

const getUsers = () => {

  if (!fs.existsSync(usersFile)) {

    fs.writeFileSync(usersFile, JSON.stringify([], null, 2)); // Create file
    if it doesn't exist

  }

  return JSON.parse(fs.readFileSync(usersFile, 'utf8'));

};

const saveUsers = (users) => {

  fs.writeFileSync(usersFile, JSON.stringify(users, null, 2)); // Save
  users to JSON file

};

// Routes

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

  res.sendFile(path.join(__dirname, 'public', 'index.html'));

});

// Register route

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

```

```

const { username, password } = req.body;

if (!username || !password) {
  return res.status(400).send('Username and password are required.');
```

```

}
```

```

const users = getUsers();

const existingUser = users.find((user) => user.username ===
username);

if (existingUser) {
  return res.status(400).send('Username already exists.');
```

```

}
```

```

const hashedPassword = await bcrypt.hash(password, 10);
users.push({ username, password: hashedPassword });
saveUsers(users);

res.status(201).send('User registered successfully.');
```

```

});

// Login route

app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  const users = getUsers();

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

  if (!user) {
```

```

        return res.status(400).send('Invalid username or password.');
```

```

    }

    const isPasswordValid = await bcrypt.compare(password,
user.password);

    if (!isPasswordValid) {

        return res.status(400).send('Invalid username or password.');
```

```

    }

    req.session.user = username;

    res.status(200).send('Login successful.');
```

```

});

// Protected dashboard route

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

    if (!req.session.user) {

        return res.status(401).send('Unauthorized. Please log in.');
```

```

    }

    res.send(`Welcome to the dashboard, ${req.session.user}!`);

});

// Logout route

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

    req.session.destroy(() => {

        res.send('Logged out successfully.');
```

```

    });

});

```

```
// Start the server

app.listen(PORT, () => {

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

});

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-
scale=1.0">

  <title>User Registration and Login</title>

</head>

<body>

  <h1>User Registration</h1>

  <form id="registerForm">

    <label>Username:</label>

    <input type="text" id="regUsername" required>

    <label>Password:</label>

    <input type="password" id="regPassword" required>

    <button type="submit">Register</button>

  </form>

  <h1>User Login</h1>

  <form id="loginForm">

    <label>Username:</label>
```

```

    <input type="text" id="loginUsername" required>

    <label>Password:</label>

    <input type="password" id="loginPassword" required>

    <button type="submit">Login</button>

</form>

<h1>Dashboard</h1>

<button id="dashboard">Go to Dashboard</button>

<h1>Logout</h1>

<button id="logout">Logout</button>

<script>

    document.getElementById('registerForm').onsubmit = async (e) =>
    {

        e.preventDefault();

        const username =
document.getElementById('regUsername').value;

        const password =
document.getElementById('regPassword').value;

        const res = await fetch('/register', {

            method: 'POST',

            headers: { 'Content-Type': 'application/json' },

            body: JSON.stringify({ username, password })

        });

        alert(await res.text());

    };

    document.getElementById('loginForm').onsubmit = async (e) => {

```

```

        e.preventDefault();

        const username =
document.getElementById('loginUsername').value;

        const password =
document.getElementById('loginPassword').value;

        const res = await fetch('/login', {

            method: 'POST',

            headers: { 'Content-Type': 'application/json' },

            body: JSON.stringify({ username, password })

        });

        alert(await res.text());

    };

document.getElementById('dashboard').onclick = async () => {

    const res = await fetch('/dashboard');

    alert(await res.text());

};

document.getElementById('logout').onclick = async () => {

    const res = await fetch('/logout');

    alert(await res.text());

};

</script></body></html>

```

# OUTPUT:

User Registration and Login

localhost:3000

### User Registration

Username:  Password:  Register

### User Login

Username:  Password:  Login

### Dashboard

Go to Dashboard

### Logout

Logout

User Registration and Login

localhost:3000 says  
User registered successfully.

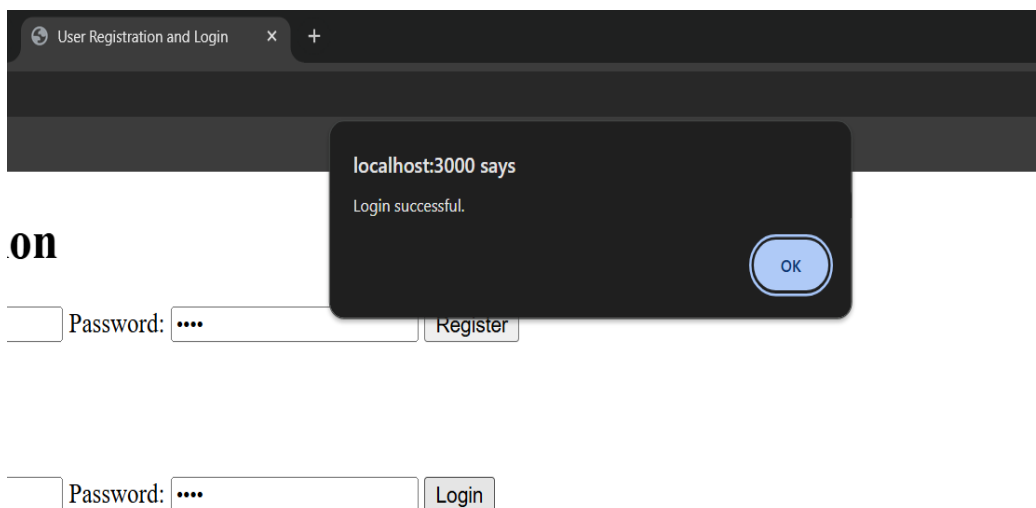
Password:  Register

Password:  Login

```
1 {
2   "username": "ravi",
3   "password": "$2a$10$18BquvRKP0HvL0HL750QJst5fW6LULx0BwF9u0aPmKycC3MC"
4 }
5 {
6   "username": "rahi",
7   "password": "$2a$10$1y85ovT55vY3a7hka15q6..OK9SC7uhaa..0bTV7X/0cPK110VQu66"
8 }
9 {
10  "username": "friends",
11  "password": "$2a$10$jpx1Pyvghu02oZ1..A22u4k80/19e0ntFes4E0c770vaf04..p0"
12 }
13 {
14  "username": "john ",
15  "password": "$2a$10$u3CL1XGJLLAcv7/JVH/HJdo/VKky1gu1Pm1V0mhJ11gcXU10gtJy"
16 }
17 {
18  "username": "john1",
19  "password": "$2a$10$3ATV15qA56Qwh30F4dnturvhca0u95FUI1k07d6CET0Tf0k4dq"
20 }
21 }
22 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

Server is running on http://localhost:3000  
C:\Users\VP\OneDrive\Desktop\summa> node server.js  
saveUsers(users);  
ReferenceError: saveUsers is not defined  
at C:\Users\VP\OneDrive\Desktop\summa\server.js:57:5  
Node.js v0.13.1  
PS C:\Users\VP\OneDrive\Desktop\summa> node server.js  
Server is running on http://localhost:3000  
PS C:\Users\VP\OneDrive\Desktop\summa> node server.js  
Server is running on http://localhost:3000



MARK ALLOCATION		
Preparation & conduct of experiment	50	
Observation & Result	30	
Record	10	
Viva voice	10	
Total	100	

## RESULT:

The user registration and login system was successfully developed using Express.js with secure password handling and session management.

**FACULTY SIGNATURE**