Shiromani Gurdwara Prabandhak Committee's (S.G.P.C.)

GURU NANAK INSTITUTE OF MANAGEMENT STUDIES

(Management Institute of G N Khalsa College),
Matunga, Mumbai – 400 019
Affiliated to University of Mumbai

Name : Khan Tabish Mujeeb

Roll No : 25MCA-33

Subject code : MCAL14

Subject Name : Web Technologies Lab

Shiromani Gurdwara Prabandhak Committee's (S.G.P.C.)
GURU NANAK INSTITUTE OF MANAGEMENT STUDIES
(Management Institute of G N Khalsa College),
Matunga, Mumbai – 400 019
Affiliated to University of Mumbai

Web Technologies Lab

Index

Practical Number	Problem Statement	Date
1	Create an application to demonstrate Node.js Modules.	26/09/25
2	Create an application to demonstrate various Node.js Events.	03/10/25
3	Create an application to demonstrate Node.js Functions.	06/10/25
4	Using File Handling demonstrate all basic file operations (Create,write,read,delete).	
	• File Handling without modules	15/10/25
	• File Handling with modules	24/10/25
5	Create an HTTP Server and perform operations on it.	
	PART 1 : GCD and Reverse	31/10/25
	PART 2 : Create http server to home,contact and about pages.	07/11/25
6	Create an application to establish a connection with the MySQL database and perform basic database operations on it.	
	PART 1 : Create an application to establish a connection with the MySQL database and perform CRUD operations on student data.	14/11/25
	PART 2 Create a http website using node js having the following: • home page • enquiry page • contact us • login page • registration page • view page make sure to use port number 9091 for nodejs and make use of CSS to make your website more attractive.	16/11/25
7	Create an application in ReactJS to implement component life cycle	
	1. Using ReactJs print 'Hello World' on the web browser.	28/11/25
	2. Using ReactJS components (Hooks) display different emojis on the screen. print the name of atleast 10 emoji	28/11/25

	using array and 3 without arrays. When the user clicks the emoji the name of the emoji should popup as an alert using DOM objects.	
8	**Create an application to implement class, functional component, state and props in ReactJS**	
	1. Create a basic program by implementing class,state and props in ReactJS.	**05/12/25**
	2. Design a Profile Card component in React that displays a user's information.	**18/12/25**
9	**Create an application in ReactJSto import and export the files (components) and in ReactJSto use DOM events**	
	1. Create a basic application using ReactSto.	**05/12/25**
	2. Design a ReactJS webpage that displays the resumes of six students.	**05/12/25**
10	**Create an application to implement React Hooks.**	**11/12/25**
11	**Practical No. 11**	
	Part 1	**18/12/25**
	Part 2	**22/12/25**

Practical-1

Aim: Create an application to demonstrate Node.js Modules

Theory:

In NodeJS, modules play an important role in organizing, structuring, and reusing code efficiently. A module is a self-contained block of code that can be exported and imported into different parts of an application. This modular approach helps developers manage large projects, making them more scalable and maintainable.

Types of modules:

- Built-in: These are modules that come with Node.js, such as http, fs, and path.
- Local: These are files you create yourself within your project.
- Third-party: These are modules created by other developers and installed from a package manager like npm, such as validator.

Programming Code for Node.js Modules:

app.js

```
const factorial = require('./factorial');

const isPrime = require('./prime');

const isArmstrong = require('./armstrong');

console.log("Factorial of 10:", factorial(10));

console.log("Is 8 Prime?", isPrime(8));

console.log("Is 153 Armstrong?", isArmstrong(153));

console.log("Is 123 Armstrong?", isArmstrong(123));
```

prime.js

```
function isPrime(n) {
    if (n <= 1) return false;
    for (let i = 2; i <= Math.sqrt(n); i++) {
        if (n % i === 0) return false;
    }
    return true;
}
module.exports = isPrime;
```

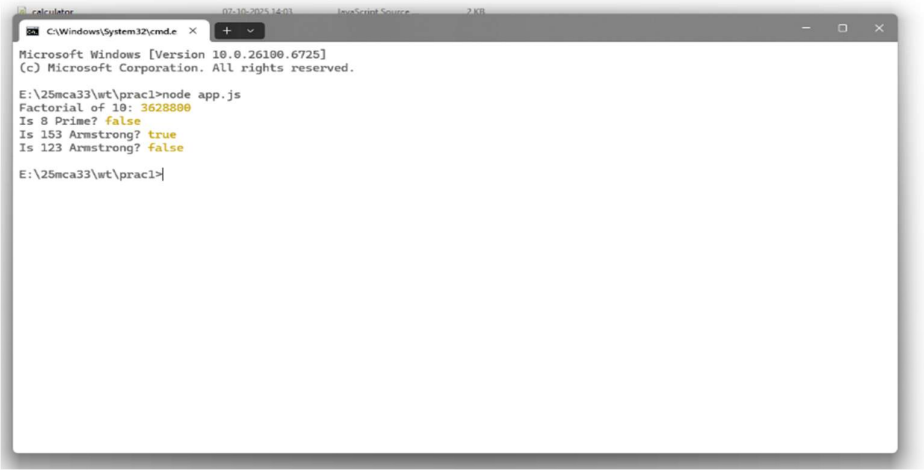
armstong.js

```
function isArmstrong(n) {
    let digits = n.toString().split('').map(Number);
    let sum = digits.reduce((acc, digit) => acc + Math.pow(digit, digits.length), 0);
    return sum === n;
}
module.exports = isArmstrong;
```

factorial.js

```
function factorial(n) {
    if (n < 0) {
        return "Factorial is not defined for negative numbers";
    }
    let result = 1;
    for (let i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
module.exports = factorial;
```

Implementation (Output of the Program) :



Practical -2

Aim: Create an application to demonstrate various Node.js Events.

Theory: NodeJS Events Concept

▪ In Node.js, events are powerful mechanisms that allow objects to emit and listen for certain actions or occurrences.

▪ Events are a core part of the Node.js architecture, enabling a non-blocking I/O model, which is crucial for building scalable applications.

▪ The EventEmitter class is part of the built-in events module in Node.js.

▪ It allows objects to emit events and register callbacks to handle those events.
on(eventName, listener): Adds a listener function to the end of the listeners array for the specified event.

emit(eventName, [...args]): Emits an event, invoking all listeners for that event.

once(eventName, listener): Adds a one-time listener for the specified event. The listener is invoked only the next time the event is emitted, after which it is removed.

removeListener(eventName, listener): Removes a listener from the specified event.

removeAllListeners([eventName]): Removes all listeners for the specified event, or all listeners if no event is specified.

▪ Key Methods of EventEmitter:

▪ The events module is fundamental to the asynchronous nature of Node.js, enabling developers to create interactive and responsive applications. Understanding how to effectively use the EventEmitter class helps manage asynchronous flows and improve application performance.

▪ The Node.js Event module facilitates a robust event-driven architecture, allowing you to create applications that can effectively handle asynchronous operations, user interactions, and system events.

Create an application to demonstrate various Node.js Events

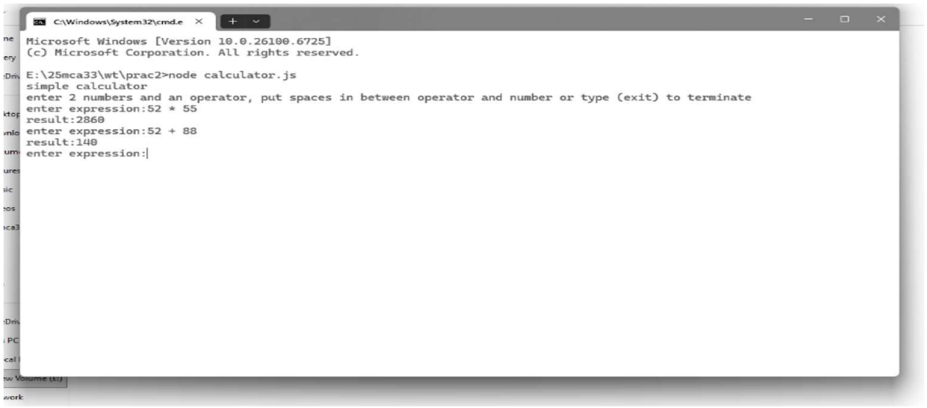
calculator.js

```
const prompt= require('prompt-sync')();

const EventEmitter=require('events');

const eventEmitter=new EventEmitter();

function performOperation(operator, num1 , num2)

{

   let result;
```

```
    switch(operator)
    {
        case '+':
            result=num1 + num2;
            break;
        case '-':
            result=num1 - num2;
            break;
        case '*':
            result=num1 * num2;
            break;
        case '/':
            result=num1 / num2;
            break;
        default:
            console.log('invalid operator');
            return;
    }
    eventEmitter.emit('calculationResult',result);
}
eventEmitter.on('calculationResult',(result)=>{
    console.log('result:'+result);
    askForInput();
});
function askForInput()
{
    const input=prompt('enter expression:');
    if(input=='exit')
```

```
    {
        process.exit(0);
    }
    else
    {
        const [num1, operator, num2]=input.split(' ');
            if(num1 && operator && num2)
        {
                        performOperation(operator,parseFloat(num1),parseFloat(num2));
        }
        else
        {
        console.log('invalid input. please enter two numbers and an operator with
spaces in between');
        }
    }
}
console.log('simple calculator');
console.log('enter 2 numbers and an operator, put spaces in between operator and
number or type (exit) to terminate');
askForInput();
```

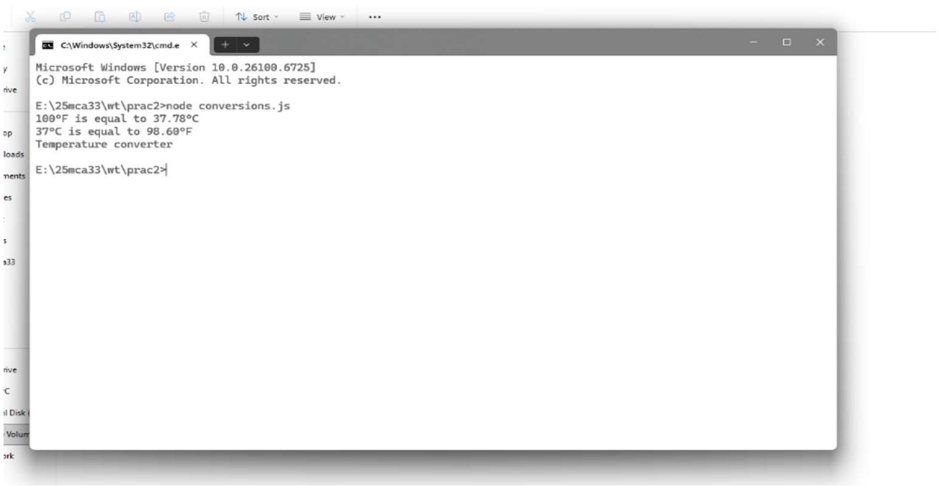
Implementation (Output of the Program) :



conversion.js

```
const EventEmitter = require('events');
const eventEmitter = new EventEmitter();
const conversions = {
    toCelsius: (fahrenheit) => (fahrenheit - 32) * 5 / 9,
    toFahrenheit: (celsius) => (celsius * 9 / 5) + 32,
};
eventEmitter.on('toCelsius', (fahrenheit) => {
    const celsius = conversions.toCelsius(fahrenheit);
    console.log(`${fahrenheit}°F is equal to ${celsius.toFixed(2)}°C`);
});
eventEmitter.on('toFahrenheit', (celsius) => {
    const fahrenheit = conversions.toFahrenheit(celsius);
    console.log(`${celsius}°C is equal to ${fahrenheit.toFixed(2)}°F`);
});
eventEmitter.emit('toCelsius', 100);
eventEmitter.emit('toFahrenheit', 37);
console.log('Temperature converter');
```

Implementation (Output of the Program) :

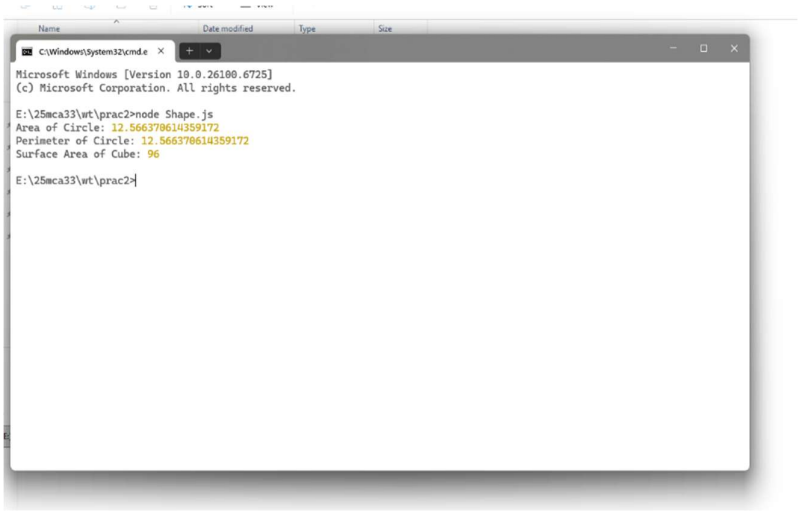


Shape.js

```
const EventEmitter = require('events');
class GeometryEmitter extends EventEmitter {
```

```
    calculateAreaOfCircle(radius) {
        const area = Math.PI * Math.pow(radius, 2);
        this.emit('areaOfCircle', area);
    }
    calculatePerimeterOfCircle(radius) {
        const perimeter = 2 * Math.PI * radius;
        this.emit('perimeterOfCircle', perimeter);
    }
    calculateAreaOfCube(side) {
        const surfaceArea = 6 * Math.pow(side, 2);
        this.emit('areaOfCube', surfaceArea);
    }
}
const geometryEmitter = new GeometryEmitter();
geometryEmitter.on('areaOfCircle', (area) => {
    console.log('Area of Circle:', area);
});
geometryEmitter.on('perimeterOfCircle', (perimeter) => {
    console.log('Perimeter of Circle:', perimeter);
});
geometryEmitter.on('areaOfCube', (surfaceArea) => {
    console.log('Surface Area of Cube:', surfaceArea);
});
const radius = 2;
const side = 4;
geometryEmitter.calculateAreaOfCircle(radius);
geometryEmitter.calculatePerimeterOfCircle(radius);
geometryEmitter.calculateAreaOfCube(side);
```

Implementation (Output of the Program) :



Practical 3

Aim: Create an application to demonstrate Node.js Functions

Theory:

Functions in Node.js are essentially JavaScript functions executed within the Node.js runtime environment. They are fundamental building blocks for organizing and executing code, and they play a crucial role in Node.js's asynchronous, event-driven architecture.

Key Aspects of Functions in Node.js:

JavaScript Syntax: Node.js functions adhere to standard JavaScript function syntax, including: Function Declarations:

main.js

```
const readlineSync = require('readline-sync');
const isPrime = require('./isprime');
const factorial = require('./factorial');
const fibonacci = require('./fibonacci');
function main() {
 console.log("Select an option:");
 console.log("1. Check if a number is prime");
 console.log("2. Calculate factorial of a number");
 console.log("3. Display Fibonacci series");
 const choice = readlineSync.questionInt("Enter your choice (1/2/3): ");
 switch (choice) {
  case 1:
   const num1 = readlineSync.questionInt("Enter a number to check if it's prime: ");
   const resultPrime = isPrime(num1);
   console.log(resultPrime ? "It's a prime number." : "It's not a prime number.");
   console.log(`Prime check result: ${resultPrime}`);
   break
  case 2:
   const num2 = readlineSync.questionInt("Enter a number to find its factorial: ");
   const resultFactorial = factorial(num2);
   console.log(`Factorial of ${num2} is ${resultFactorial}`);
```

```
      break;
    case 3:
      const num3 = readlineSync.questionInt("Enter how many Fibonacci numbers to display:
");
      const resultFibonacci = fibonacci(num3);
      console.log(`First ${num3} Fibonacci numbers:`, resultFibonacci.join(", "));
      break;
    default:
      console.log("Invalid choice. Please enter 1, 2, or 3.");
  }
}
main();
```

isprime.js

```
function isPrime(num) {
  if (num <= 1) return false;
  for (let i = 2; i <= Math.sqrt(num); i++) {
    if (num % i === 0) return false;
  }
  return true;
}
module.exports = isPrime;
```

factorial.js

```
function factorial(num) {
  if (num < 0) return 'Factorial is not defined for negative numbers';
  let fact = 1;
  for (let i = 1; i <= num; i++) {
    fact *= i;
  }
  return fact;
}
module.exports = factorial;
```

fibonacci.js

```
function fibonacci(n) {
  if (n <= 0) return 'Please enter a positive integer';
  let fib = [0, 1];
  for (let i = 2; i < n; i++) {
    fib.push(fib[i - 1] + fib[i - 2]);
  }
  return fib;
}
module.exports = fibonacci;
```

Implementation (Output of the Program) :

Practical 4

AIM: Using File Handling, demonstrate all basic file operations: Create, Write, Read, Delete in Node.js.

FILE HANDLING IN NODE.JS – CONCEPT

File handling in Node.js is an essential feature that allows developers to interact with the file system. Node.js provides a built-in module called fs (File System), which enables reading, writing, updating, deleting, and managing files and directories. The fs module supports both asynchronous (non-blocking) and synchronous (blocking) methods.

File System Module (fs)

The fs module is part of Node.js core and provides an API for performing file operations, such as:

- Creating files
- Writing to files
- Reading files
- Updating files
- Deleting files
- Getting file metadata
- Working with directories

Types of File Operations

1. Reading Files

Node.js supports both asynchronous and synchronous reading.

- Asynchronous:

fs.readFile() – reads file content without blocking the main thread.

- Synchronous:

fs.readFileSync() – reads file content and blocks execution until complete.

2. Writing Files

Writing data to files can also be done asynchronously or synchronously.

- Asynchronous:

fs.writeFile() – writes data to a file. Creates the file if it does not exist.

- Synchronous:

fs.writeFileSync() – writes data to a file synchronously.

3. Updating Files

- Appending Data Asynchronously:

fs.appendFile() – adds data to an existing file.

4. Deleting Files

- Asynchronous Delete:

fs.unlink() – deletes a file asynchronously.

- Synchronous Delete:

fs.unlinkSync() – deletes a file synchronously.

5. File Metadata Operations

- Asynchronous:

fs.stat() – retrieves metadata (size, creation time, modification time).

- Synchronous:

fs.statSync() – retrieves metadata synchronously.

6. Working with Directories

Node.js supports:

- Creating directories
- Reading directory contents
- Deleting directories

Using fs methods like fs.mkdir(), fs.readdir(), and fs.rmdir().

Error Handling

Node.js uses callback-based error handling for asynchronous operations.

Always check the err parameter in callback functions.

For synchronous methods, use try–catch blocks to manage exceptions.

Conclusion

File handling in Node.js is efficient and powerful due to its non-blocking nature. Developers frequently use it for tasks such as logging, configuration management, data persistence, and general file manipulation.

Prac4.A

Q.1 Program in Node.js to Perform Basic File Operations (Create, Read, Write, Delete)

Code:

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

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

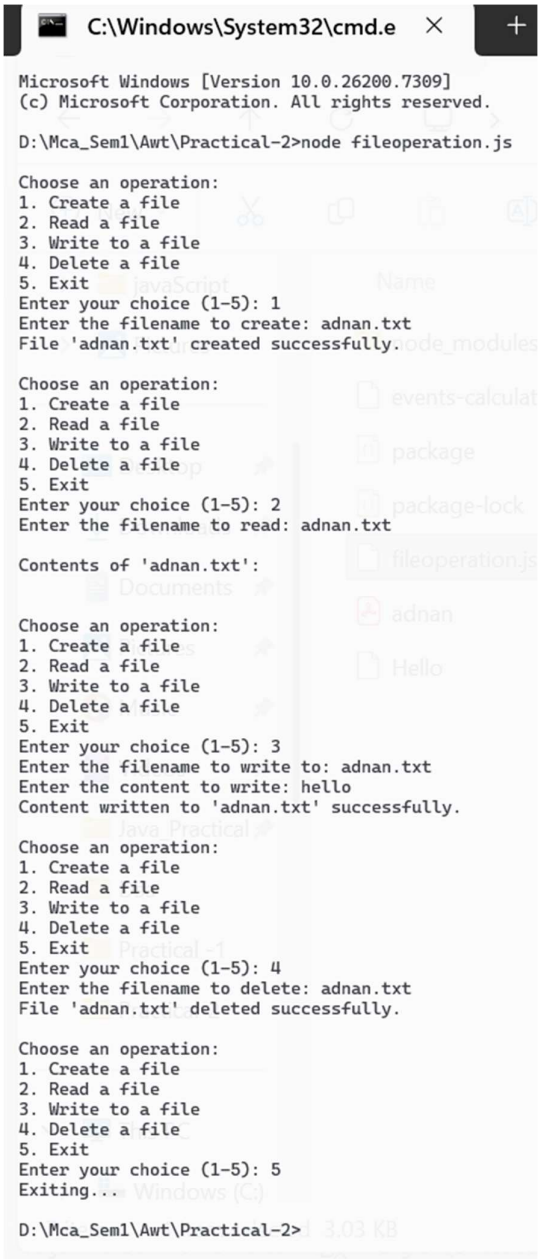
```
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});
function showMenu() {
    console.log("\nChoose an operation:");
    console.log("1. Create a file");
    console.log("2. Read a file");
    console.log("3. Write to a file");
    console.log("4. Delete a file");
    console.log("5. Exit");
}
function createFile() {
    rl.question('Enter the filename to create: ', (filename) => {
        fs.writeFile(filename, '', (err) => {
            if (err) {
                console.error('Error creating file:', err);
            } else {
                console.log(`File '${filename}' created successfully.`);
            }
            showMenu();
            getUserChoice();
        });
    });
}
function readFile() {
    rl.question('Enter the filename to read: ', (filename) => {
        fs.readFile(filename, 'utf8', (err, data) => {
            if (err) {
```

```
            console.error('Error reading file:', err);
        } else {
            console.log(`\nContents of '${filename}':\n${data}`);
        }
        showMenu();
        getUserChoice();
    });
  });
}
function writeFile() {
  rl.question('Enter the filename to write to: ', (filename) => {
    rl.question('Enter the content to write: ', (content) => {
      fs.writeFile(filename, content, (err) => {
        if (err) {
          console.error('Error writing to file:', err);
        } else {
          console.log(`Content written to '${filename}' successfully.`);
        }
        showMenu();
        getUserChoice();
      });
    });
  });
}
function deleteFile() {
  rl.question('Enter the filename to delete: ', (filename) => {
    fs.unlink(filename, (err) => {
      if (err) {
        console.error('Error deleting file:', err);
      } else {
```

```
            console.log(`File '${filename}' deleted successfully.`);
        }
        showMenu();
        getUserChoice();
    });
  });
}
function getUserChoice() {
  rl.question('Enter your choice (1-5): ', (choice) => {
    switch (choice) {
      case '1':
        createFile();
        break;
      case '2':
        readFile();
        break;
      case '3':
        writeFile();
        break;
      case '4':
        deleteFile();
        break;
      case '5':
        console.log('Exiting...');
        rl.close();
        break;
      default:
        console.log('Invalid choice. Please try again.');
        showMenu();
        getUserChoice();
```

```
            break;
        }
    });
}
showMenu();
getUserChoice();
```

Implementation (Output of the Program) :



Prac4.B

Aim: File Handling with modules

App.js

```
const readline = require('readline');
const feedback = require('./feedback');
const rl = readline.createInterface({
   input: process.stdin,
   output: process.stdout
});
feedback.on('feedback-received', (message) => {
   console.log(`Feedback received: ${message}`);
});
feedback.on('feedback-saved', () => {
   console.log('Feedback has been saved successfully!');
});
feedback.on('feedback-read', (data) => {
   console.log('Feedback Read:\n', data);
});
feedback.on('feedback-deleted', () => {
   console.log('Feedback file has been deleted.');
});
const showMenu = () => {
   rl.question(
      'Select an action:\n1. Add Feedback\n2. Read Feedback\n3. Delete Feedback\n4.
Exit\nYour choice: ',
      (choice) => {
         switch (choice) {
            case '1':
               rl.question('Please enter your feedback: ', (feedbackMessage) => {
                  feedback.addFeedback(feedbackMessage);
                  showMenu();
```

```
                });
                break;
            case '2':
                feedback.readFeedback();
                showMenu();
                break;
            case '3':
                feedback.deleteFeedbackFile();
                showMenu();
                break;
            case '4':
                console.log('Goodbye!');
                rl.close();
                break;
            default:
                console.log('Invalid choice, please try again.');
                showMenu();
        }
    }
  );
};
showMenu();
```

feedback.js

```
const fs = require('fs');
const EventEmitter = require('events');
const path = './feedback.txt';
class Feedback extends EventEmitter {
  constructor() {
    super();
```

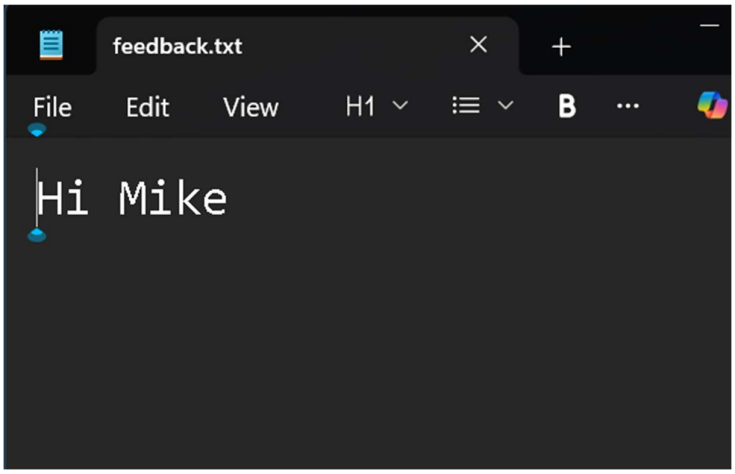
```
    this.feedbackMessages = [];
  }
  addFeedback(feedbackMessage) {
    this.feedbackMessages.push(feedbackMessage);
    this.emit('feedback-received', feedbackMessage);
    this.saveFeedback();
  }
  saveFeedback() {
    fs.writeFile(path, this.feedbackMessages.join('\n'), (err) => {
      if (err) {
        console.error('Error writing feedback to file:', err);
      } else {
        this.emit('feedback-saved');
      }
    });
  }
  readFeedback() {
    fs.readFile(path, 'utf8', (err, data) => {
      if (err) {
        console.error('Error reading feedback from file:', err);
      } else {
        this.emit('feedback-read', data);
        console.log('All Feedback:\n', data);
      }
    });
  }
  deleteFeedbackFile() {
    fs.unlink(path, (err) => {
      if (err) {
        console.error('Error deleting feedback file:', err);
```

```
        } else {
            this.emit('feedback-deleted');
            console.log('Feedback file deleted.');
        }
    });
  }
}
module.exports = new Feedback();
```

Implementation(Output Of The Code):


```
Microsoft Windows [Version 10.0.26220.7523]
(c) Microsoft Corporation. All rights reserved.

D:\MCA\sem4\Activity-20260104T145913Z-1-001>node app.js
Select an action:
1. Add Feedback
2. Read Feedback
3. Delete Feedback
4. Exit
Your choice: 1
Please enter your feedback: Hi Mike
Feedback received: Hi Mike
Select an action:
1. Add Feedback
2. Read Feedback
3. Delete Feedback
4. Exit
Your choice: Feedback has been saved successfully!
2
Select an action:
1. Add Feedback
2. Read Feedback
3. Delete Feedback
4. Exit
Your choice: Feedback Read:
 Hi Mike
All Feedback:
 Hi Mike
```

```
All Feedback:
 Hi Mike

Invalid choice, please try again.
Select an action:
1. Add Feedback
2. Read Feedback
3. Delete Feedback
4. Exit
Your choice: 3
Select an action:
1. Add Feedback
2. Read Feedback
3. Delete Feedback
4. Exit
Your choice: Feedback file has been deleted.
Feedback file deleted.

Invalid choice, please try again.
Select an action:
1. Add Feedback
2. Read Feedback
3. Delete Feedback
4. Exit
Your choice: 4
Goodbye!
```



Practical 5

Aim : Create an HTTP Server and perform operations on it

Practical 5.A

Server.js

```
const http = require('http');
const url = require('url');
const fs = require('fs');
function calculateGCD(num1, num2) {
   while (num2 !== 0) {
      const temp = num2;
      num2 = num1 % num2;
      num1 = temp;
   }
   return num1;
}
function reverseNumber(num) {
   let reversed = 0;
   while (num !== 0) {
      reversed = reversed * 10 + num % 10;
      num = Math.floor(num / 10);
   }
   return reversed;
}
const server = http.createServer((req, res) => {
   const urlPath = req.url.split("?")[0];
   if (urlPath === "/") {
      fs.readFile("index.html", (err, data) => {
         res.writeHead(200, { "Content-Type": "text/html" });
         res.end(data);
      });
```

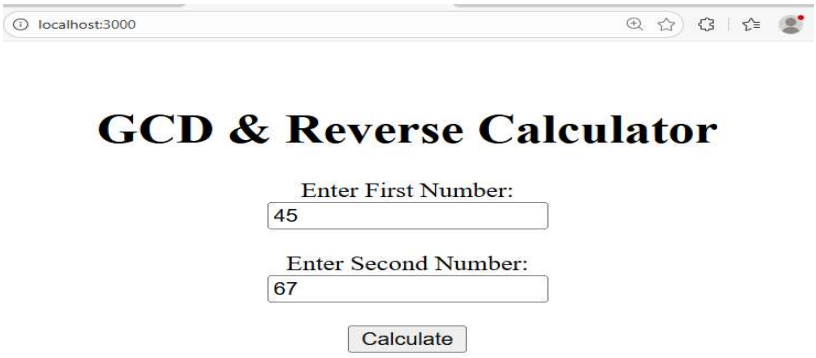
```
        return;
    }
    if (urlPath === "/calculate") {
        const queryObject = url.parse(req.url, true).query;
        const num1 = parseInt(queryObject.num1);
        const num2 = parseInt(queryObject.num2);
        const gcd = calculateGCD(num1, num2);
        const reversedNum1 = reverseNumber(num1);
        const reversedNum2 = reverseNumber(num2);
        const resultPage = `
        <h1><center>Results</center></h1>
        <p>GCD of ${num1} and ${num2} is <b>${gcd}</b></p>
        <p>Reverse of ${num1} is <b>${reversedNum1}</b></p>
        <p>Reverse of ${num2} is <b>${reversedNum2}</b></p>
        <br><a href="/">Go Back</a>
        `;
        res.writeHead(200, { "Content-Type": "text/html" });
        res.end(resultPage);
        return;
    }
});
server.listen(3000, () => {
    console.log("Server running at http://localhost:3000/");
});
```

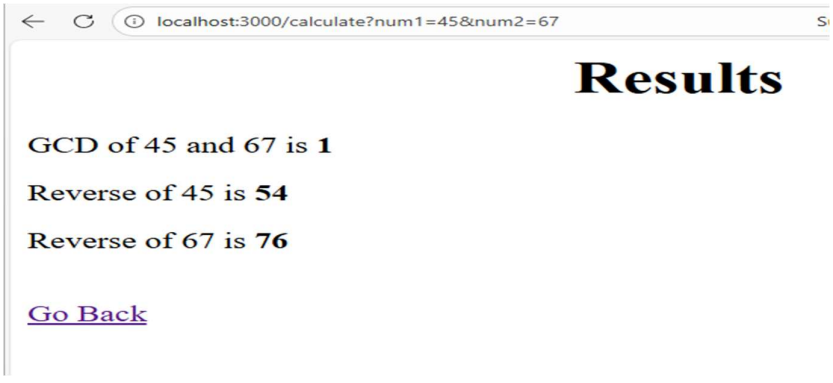
Index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>GCD & Reverse</title>
</head>
```

```
<body style="text-align:center; margin-top:50px;">
<h1>GCD & Reverse Calculator</h1>
<form action="/calculate" method="GET">
    <label>Enter First Number:</label><br>
    <input type="number" name="num1" required><br><br>
    <label>Enter Second Number:</label><br>
    <input type="number" name="num2" required><br><br>
    <button type="submit">Calculate</button>
</form>
</body>
</html>
```

Implementation of the code(output)





Practical 5b

Code:

Server.js

```
const http = require('http');
const fs = require('fs');
const path = require('path');
const server = http.createServer((req, res) => {
    res.setHeader("Content-Type", "text/html");
    if (req.method === "GET" && req.url === "/") {
        const homePage = fs.readFileSync(path.join(__dirname, "index.html"));
        res.writeHead(200);
        return res.end(homePage);
    }
    else if (req.method === "POST" && req.url === "/about") {
        let data = "";
        req.on("data", chunk => {
            data += chunk;
        });
        req.on("end", () => {
            res.writeHead(200);
            res.end(`<h1>About Page Submitted</h1><p>You sent: ${data}</p>`);
        });
    }
    else if (req.method === "GET" && req.url === "/contact") {
        const contactPage = fs.readFileSync(path.join(__dirname, "contact.html"));
        res.writeHead(200);
        return res.end(contactPage);
    }
    else if (req.method === "GET" && req.url === "/about") {
        const aboutPage = fs.readFileSync(path.join(__dirname, "about.html"));
```

```
        res.writeHead(200);
        return res.end(aboutPage);
    }
    else {
        res.writeHead(404);
        res.end("<h1>404 - Page Not Found</h1>");
    }
});
server.listen(3000, () => {
    console.log("Server is running on http://localhost:3000");
});
```

Index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1>Welcome to Home Page</h1>
    <p>This is the Home page served using GET request.</p>
    <a href="/about">Go to About Page</a><br><br>
    <a href="/contact">Go to Contact Page</a>
</body>
</html>
```

Contact.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Contact</title>
</head>
```

```
<body>
    <h1>Contact Page</h1>
    <p>Contact Number: 789456123</p>
    <a href="/">Back to Home</a>
</body>
</html>
```

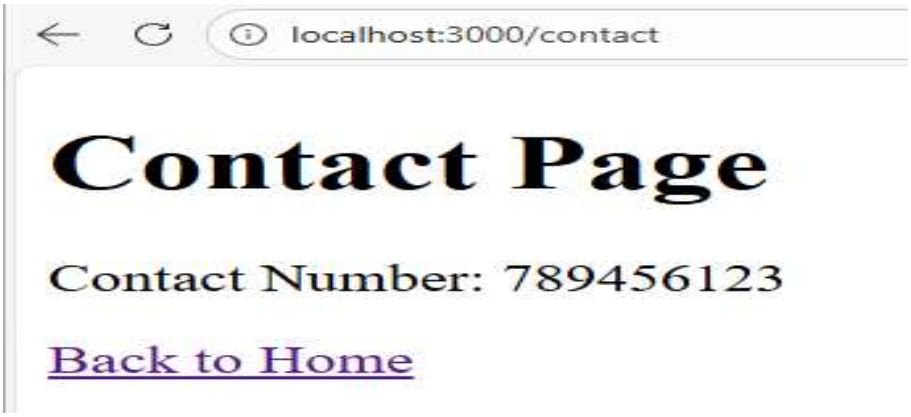
about.html

```
<!DOCTYPE html>
<html>
<head>
    <title>About</title>
</head>
<body>
    <h1>About Page</h1>
    <form action="/about" method="POST">
        <label>Enter any message:</label><br>
        <input type="text" name="info" required><br><br>
        <button type="submit">Submit</button>
    </form>
    <br><a href="/">Back to Home</a>
</body>
</html>
```

Implementation of the code(output)


localhost:3000/about
About Page
Enter any message:
Hello
Submit
Back to Home


localhost:3000/about
About Page Submitted
You sent: info=Hello


localhost:3000/contact
Contact Page
Contact Number: 789456123
Back to Home

Practical 6

Aim: Create an application to establish a connection with the MySQL database and perform basic database operations on it.

Aim:

PART 1 : Create an application to establish a connection with the MySQL database and perform CRUD operations on student data.

MySQL Database Concept

ReactJS and Database Interaction

- ReactJS is a front-end library, which means it runs in the user's browser.
- For security reasons and due to the architecture of web applications, ReactJS cannot directly interact with a MySQL database.
- Instead, a ReactJS application communicates with a database through a backend server.

Backend Server

- A backend server is created using technologies such as Node.js, Express.js, or any other server-side language/framework.
- The backend server acts as an intermediary between the ReactJS application and the MySQL database.

Database Connection

- On the backend, a MySQL driver or an ORM (Object-Relational Mapping) library is used to connect to the MySQL database.
- Commonly used libraries include:
  - mysql2
  - sequelize
  - Knex.js
- These libraries help establish and manage database connections efficiently.

API Endpoints

- The backend exposes RESTful API endpoints using Express.js.
- The ReactJS application interacts with these endpoints using HTTP methods such as GET, POST, PUT, and DELETE.

Basic Database Operations (CRUD)

Once the database connection is established, the following basic operations can be performed:

1. Create (Insert Data)

- Used to add new records to the database.
- Endpoint Example:
  - POST /api/data

2. Read (Fetch Data)

- Used to retrieve records from the database.
- Endpoint Example:
  - GET /api/data

3. Update (Modify Data)

- Used to update existing records in the database.
- Endpoint Example:
  - PUT /api/data/:id

4. Delete (Remove Data)

- Used to delete records from the database.
- Endpoint Example:
  - DELETE /api/data/:id

Conclusion

By creating a backend server that connects to a MySQL database, a ReactJS application can safely and efficiently perform various database operations using RESTful APIs. This architecture ensures a clear separation of frontend and backend, resulting in improved security, performance, and scalability of web applications.

//npm install mysql

Code:

SQL QUERY:

```
create table student(s_id int,s_name varchar(30),s_mobile bigint,s_emailid varchar(30));
```

db.js

```
const mysql = require('mysql');
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'gnims',
  database: 'penta',
```

```
});
connection.connect((err) => {
        if (err)
        {
                 console.error('Error connecting to MySQL: ' + err.stack);
                return;
        }
        console.log('Connected to MySQL as id ' + connection.threadId);
});
module.exports = connection;
```

delete-test.js

```
// update-test.js
function delete_student()
{
                        const prompt = require('prompt-sync')();
                        const st_id = prompt('Enter the id of student to be deleted: ');
                        const db = require('./db');
                        db.query('delete from student WHERE s_id = ?', [st_id], (err, results)
=> {
                        if (err)
                        {
                                console.error('Error Deleting data: ' + err);
                        }
                        else
                        {
                                console.log('Data deleted:', results.affectedRows);
                         }
                        });
}
module.exports = {
```

```
  delete_student,
};
```

insert-test.js

```
// insert-test.js
function insert_student()
{
const prompt = require('prompt-sync')();
st_id=prompt('Enter the student id: ');
st_name=prompt('Enter the student name: ');
st_mobile=prompt('Enter the mobile number: ');
st_emailid=prompt('Enter the email id: ');
const newData = {
  s_id: st_id,
  s_name: st_name,
  s_mobile: st_mobile,
  s_emailid: st_emailid
};
const db = require('./db');
db.query('INSERT INTO student SET ?', newData, (err, results) => {
  if (err) {
    console.error('Error inserting data: ' + err);
  } else {
    console.log('Data inserted. ID:', results.insertId);
  }
});
}
module.exports = {
  insert_student,
};
```

main.js

```
const prompt = require('prompt-sync')();
function displayMenu()
{
  console.log('Menu');
  console.log('1. Update data in student table');
  console.log('2. Insert data in student table');
  console.log('3. Select data in student table');
  console.log('4. Delete data in student table');
  console.log('5. Exit');
}
function student_data()
{
        var choice;
        displayMenu();
// Get the user's choice
        choice = prompt('Enter your choice: ');
        switch (choice)
        {
                case '1':
                        const ud = require('./update-test');
                        ud.update_student();
                        setTimeout(() => {student_data();}, 1000);
                        break;
                case '2':
                        const ins = require('./insert-test');
                        ins.insert_student();
                        setTimeout(() => {student_data();}, 1000);
                        break;
                case '3':
```

```
                const sd = require('./select-test');
                sd.select_student();
                setTimeout(() => {student_data();}, 1000);
                break;
        case '4':
                const dd = require('./delete-test');
                dd.delete_student();
                setTimeout(() => {student_data();}, 1000);
                break;
        case '5':
                console.log('Goodbye!');
                process.exit(0);
                break;
        default:
                console.log('Invalid choice');
                break;
    }
}
student_data();
```

select-test.js

```
// update-test.js
function select_student()
{
                const prompt = require('prompt-sync')();
                const st_id = prompt('Enter the id of student to be displayed: ');
                const db = require('./db');
                db.query('select * from student where s_id = ?',[st_id],  (err, results) =>
{
                if (err)
                {
```

```
                        console.error('Error selecting data: ' + err);
                }
                else
                {
                        console.log('Data select:', results);
                 }
                });
}
module.exports = {
  select_student,
};

update-test.js
// update-test.js
function update_student()
{
                const prompt = require('prompt-sync')();
                st_name=prompt('Enter the student name: ');
                st_mobile=prompt('Enter the mobile number: ');
                st_email=prompt('Enter the email address: ');
                const updatedData = {
                                s_name: st_name,
                                s_mobile: st_mobile,
                                s_emailid: st_email,
                        };
                const st_id = prompt('Enter the student id of student whose name and
mobile has to be updated: ');
                const db = require('./db');
                db.query('UPDATE student SET ? WHERE s_id = ?', [updatedData,
st_id], (err, results) => {
                if (err)
```

```
                    {
                        console.error('Error updating data: ' + err);
                    }
                    else
                    {
                        console.log('Data updated:', results.affectedRows);
                     }


                    });
}
module.exports = {
  update_student,
};
```

Implementation of the code(output)

Inserting Data in Student Table

```
E:\25mca33\Sem1\wt\CRUD>node main.js
Menu
1. Update data in student table
2. Insert data in student table
3. Select data in student table
4. Delete data in student table
5. Exit
Enter your choice: 2
Enter the student id: 31
Enter the student name: Adnan Khan
Enter the mobile number: 9876
Enter the email id:  adnan.khan@gmail.com
Connected to MySQL as id 8
Data inserted. ID: 0
```

```
mysql> select * from student;
+------+-----------+------------+----------------------+
| s_id | s_name    | s_mobile   | s_emailid            |
+------+-----------+------------+----------------------+
|    1 | tabish    | 1234567890 | a@gmail.com          |
|    3 | adnan khan | 1234567890 | a@gmail.com          |
|   31 | Adnan Khan |       9876 |  adnan.khan@gmail.com |
+------+-----------+------------+----------------------+
3 rows in set (0.00 sec)
```

After Updating Student Table

```
mysql> select * from student;
+------+-----------+------------+------------------------+
| s_id | s_name    | s_mobile   | s_emailid              |
+------+-----------+------------+------------------------+
|    1 | tabish    | 1234567890 | a@gmail.com            |
|    3 | adnan khan | 1234567890 | a@gmail.com            |
|   31 | Adnan Khan | 9988776655 | adnan.khan123@gmail.com |
+------+-----------+------------+------------------------+
3 rows in set (0.00 sec)
```

```
1. Update data in student table
2. Insert data in student table
3. Select data in student table
4. Delete data in student table
5. Exit
Enter your choice: 1
Enter the student name: Adnan Khan
Enter the mobile number: 9988776655
Enter the email address: adnan.khan123@gmail.com
Enter the student id of student whose name and mobile has to be updated: 31
Data updated: 1
```

Select data In Student Table

```
Enter your choice: 3
Enter the id of student to be displayed: 31
Data select: [
  RowDataPacket {
    s_id: 31,
    s_name: 'Adnan Khan',
    s_mobile: 9988776655,
    s_emailid: 'adnan.khan123@gmail.com'
  }
]
```

Delete data in student table

```
Enter your choice: 4
Enter the id of student to be deleted: 31
Data deleted: 1
Menu
1. Update data in student table
2. Insert data in student table
3. Select data in student table
4. Delete data in student table
5. Exit
Enter your choice:
```

```
mysql> select * from student;
+------+------------+------------+-------------+
| s_id | s_name     | s_mobile   | s_emailid   |
+------+------------+------------+-------------+
|    1 | tabish     | 1234567890 | a@gmail.com |
|    3 | adnan khan | 1234567890 | a@gmail.com |
+------+------------+------------+-------------+
2 rows in set (0.00 sec)

mysql>
```

Exit

```
Enter your choice: 5
Goodbye!
```

Practical 6.2

PART 2

Create a http website using node js having the following:

home page

enquiry page

contact us

login page

registration page

view page

make sure to use port number 9091 for nodejs and make use of CSS to make your website more attractive.

Code:

App.js

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const PORT = 9091;
app.use(bodyParser.urlencoded({ extended: true }));
const css = `
body {
  margin: 0;
  font-family: "Comic Sans MS", "Arial Rounded MT Bold", sans-serif;
  background: url("https://wallpapercave.com/wp/wp1914515.jpg") center/cover fixed;
  color: #fff;
  text-align: center;
}
body::before {
  content: "";
  position: fixed;
  top: 0; left: 0;
  width: 100%; height: 100%;
```

```
  background: rgba(0,0,0,0.45);
  pointer-events: none;
}
h1, h2 {
  color: #fff;
  text-shadow: 3px 3px 0 #000, 6px 6px 0 #ff0000;
  letter-spacing: 3px;
  margin-top: 10px;
  text-transform: uppercase;
  font-weight: bold;
}
nav a {
  color: white;
  margin: 15px;
  font-size: 22px;
  text-decoration: none;
  padding: 14px 26px;
  background: linear-gradient(45deg, #ff3b30, #ffcc00);
  border-radius: 30px;
  box-shadow: 0 5px 0 #b30000;
  transition: 0.2s;
  border: 3px solid #fff;
}
nav a:hover {
  transform: translateY(-4px);
  box-shadow: 0 9px 0 #b30000;
}
form {
  background: rgba(255,255,255,0.9);
  width: 380px;
```

```
  margin: auto;
  padding: 25px;
  border-radius: 15px;
  border: 4px solid #ffcc00;
  box-shadow: 0 0 25px #ffcc00, inset 0 0 25px #ff3b30;
}
input, button {
  width: 90%;
  padding: 12px;
  margin-top: 12px;
  border-radius: 10px;
  border: 3px solid #ffcc00;
  font-size: 16px;
  font-family: "Comic Sans MS", sans-serif;
}
input {
  background: #ffffff;
  color: #000;
  box-shadow: inset 0 0 10px #ffcc00;
}
input:focus {
  outline: none;
  box-shadow: 0 0 15px #ff3b30, inset 0 0 20px #ffcc00;
}
button {
  background: linear-gradient(45deg, #ff3b30, #ffcc00);
  color: #fff;
  font-weight: bold;
  box-shadow: 0 5px 0 #b30000;
  cursor: pointer;
```

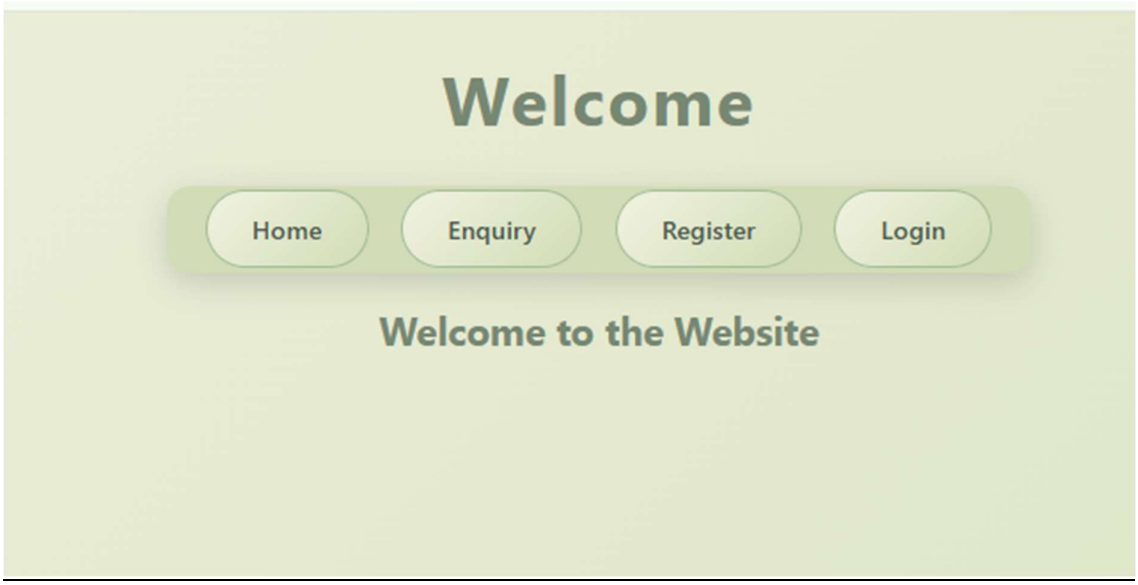
```
  transition: 0.25s;
  text-shadow: 2px 2px #000;
}
button:hover {
  transform: translateY(-3px);
  box-shadow: 0 8px 0 #b30000;
}
.box {
  background: rgba(255,255,255,0.9);
  padding: 20px;
  width: 350px;
  margin: auto;
  border-radius: 20px;
  border: 4px solid #ffcc00;
  color: #000;
  box-shadow: 0 0 30px #ffcc00, inset 0 0 40px #ff3b30;
  font-size: 20px;
}
`;
function layout(title, body) {
  return `
  <html>
  <head>
    <title>${title}</title>
    <style>${css}</style>
  </head>
  <body>
    <h1>${title}</h1>
    <nav>
      <a href="/">Home</a>
```

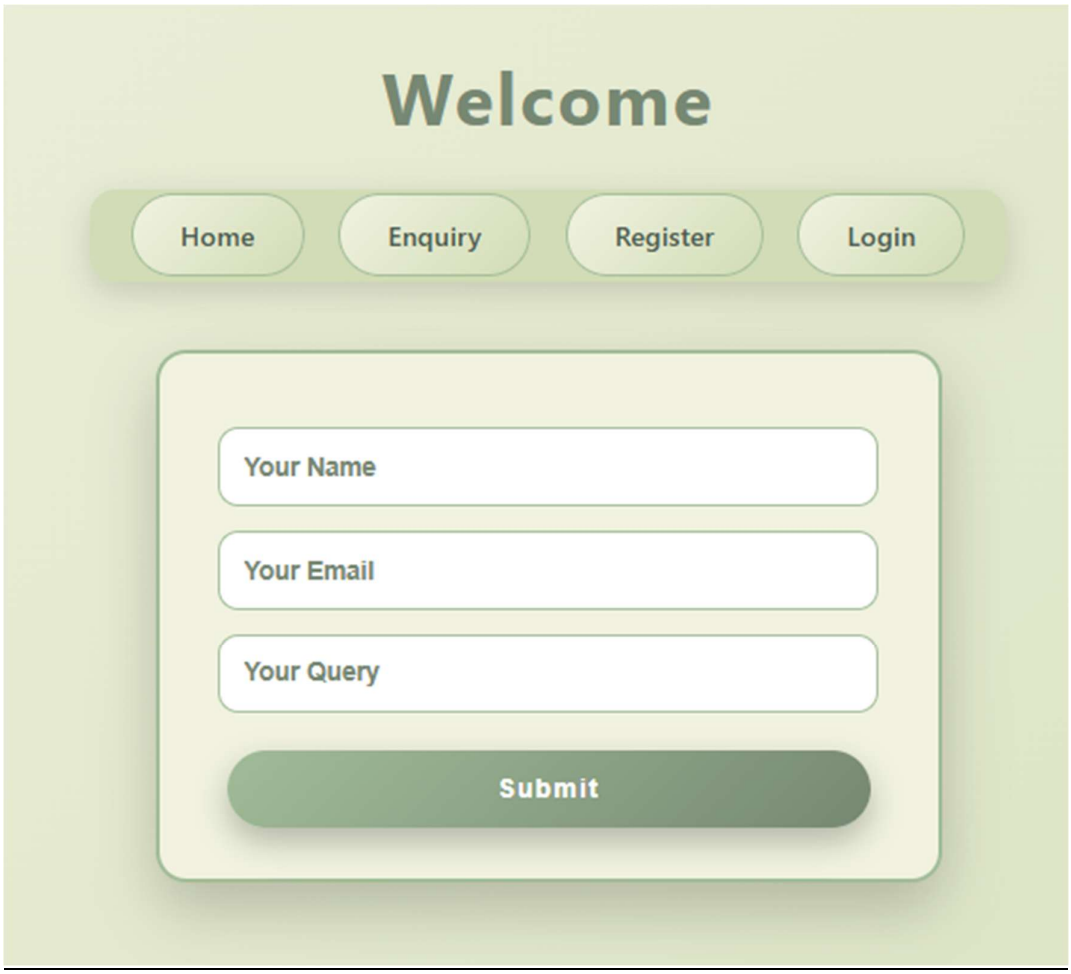
```
      <a href="/enquiry">Enquiry</a>
      <a href="/contact">Contact</a>
      <a href="/login">Login</a>
      <a href="/register">Register</a>
    </nav>
    <br><br>
    ${body}
  </body>
  </html>`;
}
app.get('/', (req, res) => {
  res.send(layout('GTA Home', `<h2>Welcome to the GTA Styled Website</h2>`));
});
app.get('/enquiry', (req, res) => {
  res.send(layout('GTA Enquiry Form', `
    <form action="/enquiry-submit" method="POST">
      <input type="text" name="name" placeholder="Your Name" required>
      <input type="email" name="email" placeholder="Your Email" required>
      <input type="text" name="query" placeholder="Your Query" required>
      <button type="submit">Submit</button>
    </form>
  `));
});
app.post('/enquiry-submit', (req, res) => {
  const { name, email, query } = req.body;
  res.send(layout('Enquiry Received', `
    <div class="box">
      <p><b>Name:</b> ${name}</p>
      <p><b>Email:</b> ${email}</p>
      <p><b>Query:</b> ${query}</p>
```

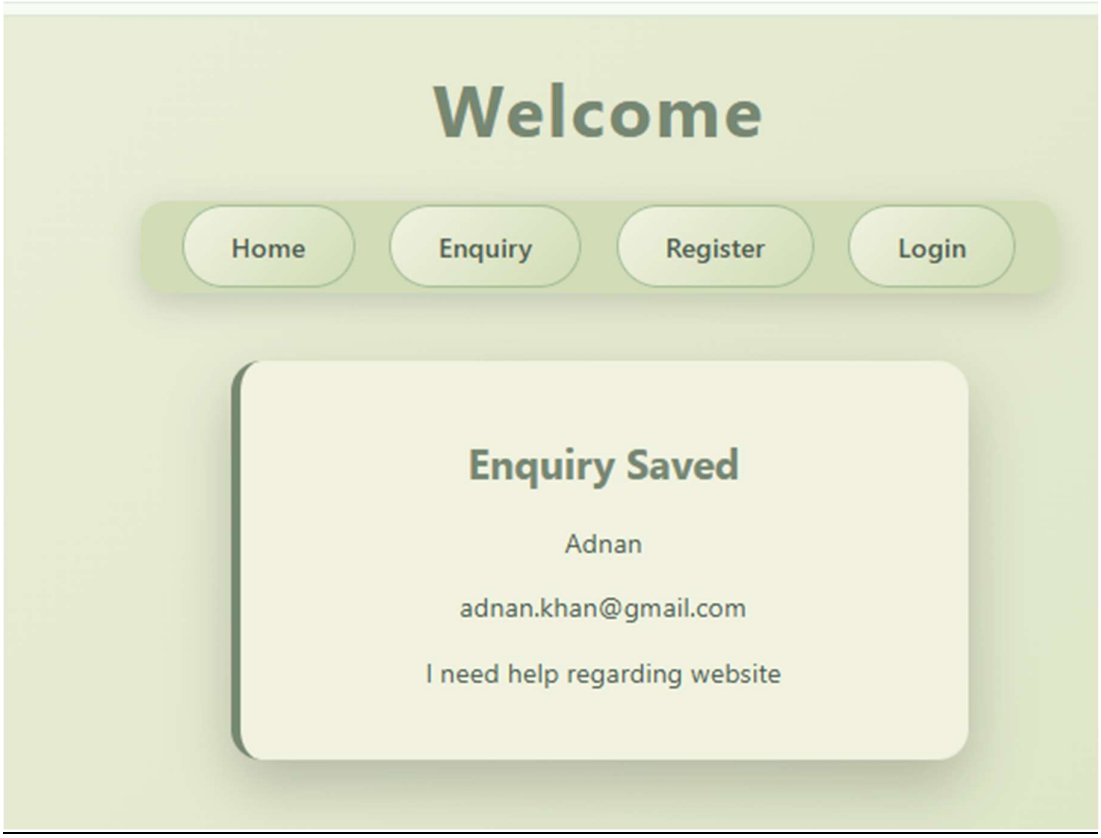
```
    </div>
  `));
});
app.get('/contact', (req, res) => {
  res.send(layout('Contact Us (GTA Style)', `
    <h2>Email: support@gtaweb.com</h2>
    <h2>Phone: +123 456 7890</h2>
  `));
});
const USER = { username: 'admin', password: '1234' };
app.get('/login', (req, res) => {
  res.send(layout('GTA Login', `
    <form action="/login-check" method="POST">
      <input type="text" name="username" placeholder="Username" required>
      <input type="password" name="password" placeholder="Password" required>
      <button type="submit">Login</button>
    </form>
  `));
});
app.post('/login-check', (req, res) => {
  const { username, password } = req.body;
  const msg =
    username === USER.username && password === USER.password
      ? `<h2>Login Successful — Welcome ${username}</h2>`
      : `<h2>Login Failed — Wrong Credentials</h2>`;
  res.send(layout('Login Result', msg));
});
app.get('/register', (req, res) => {
  res.send(layout('GTA Registration', `
    <form action="/register-submit" method="POST">
```

```
    <input type="text" name="name" placeholder="Full Name" required>
    <input type="email" name="email" placeholder="Email" required>
    <input type="tel" name="phone" placeholder="Phone Number" required>
    <input type="text" name="username" placeholder="Choose Username" required>
    <input type="password" name="password" placeholder="Password" required>
    <input type="password" name="confirm" placeholder="Confirm Password" required>
    <input type="date" name="dob" required>
    <button type="submit">Register</button>
  </form>
 `));
});
app.post('/register-submit', (req, res) => {
 const { name, email, username } = req.body;
 res.send(layout('Registration Complete', `
  <div class="box">
   <p><b>Name:</b> ${name}</p>
   <p><b>Email:</b> ${email}</p>
   <p><b>Username:</b> ${username}</p>
  </div>
 `));
});
app.listen(PORT, () =>
 console.log(`GTA Website running at http://localhost:${PORT}`)
);
```

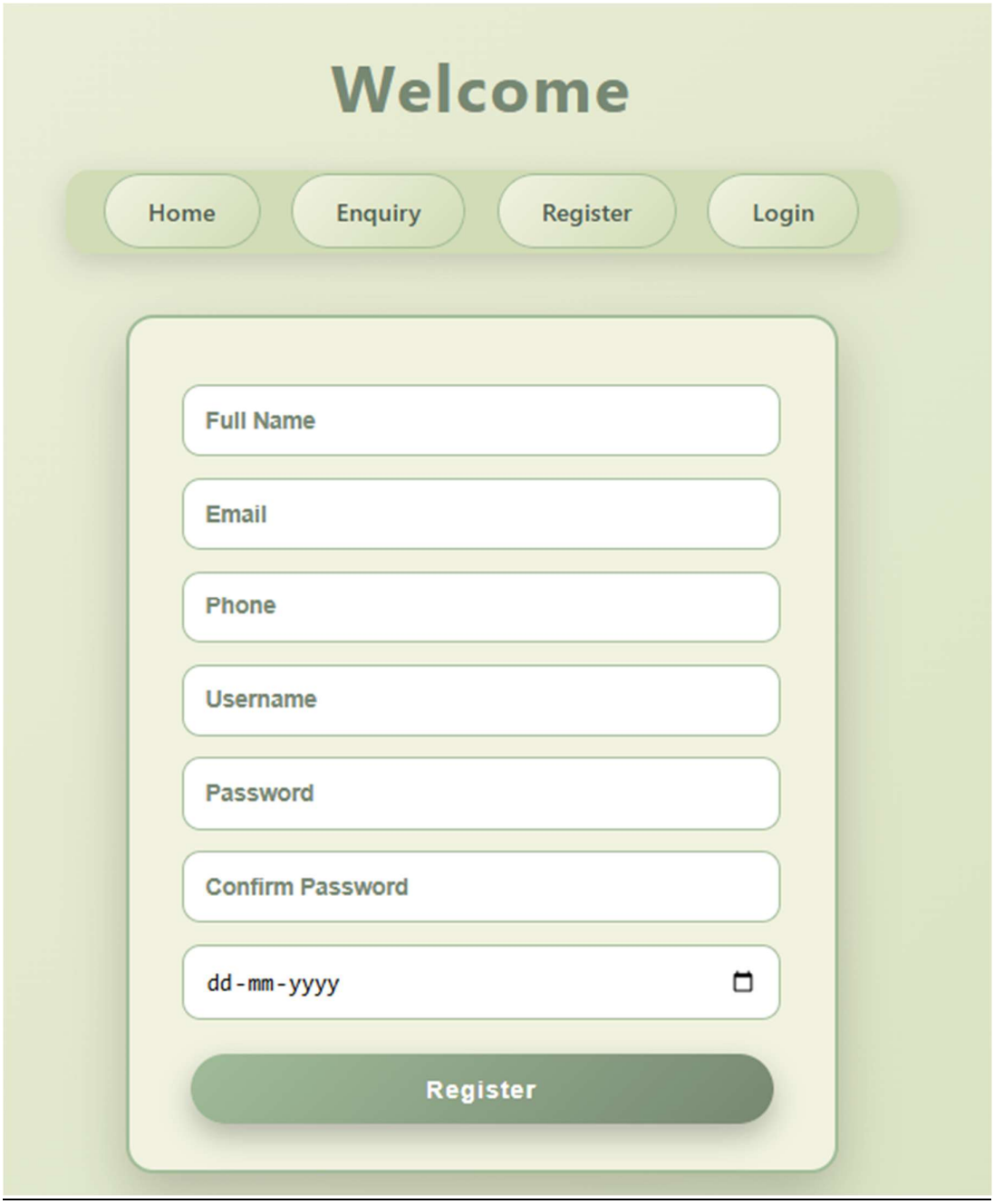
Implementation of the code(output)

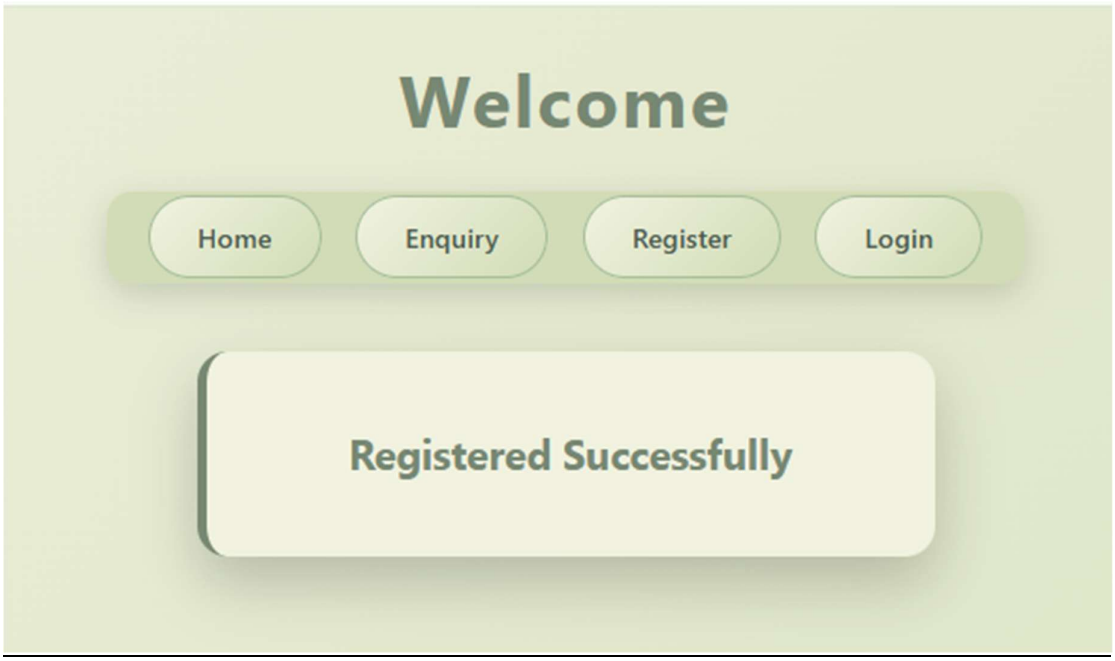

Welcome
Home
Enquiry
Register
Login
Welcome to the Website


Welcome
Home
Enquiry
Register
Login
Your Name
Your Email
Your Query
Submit

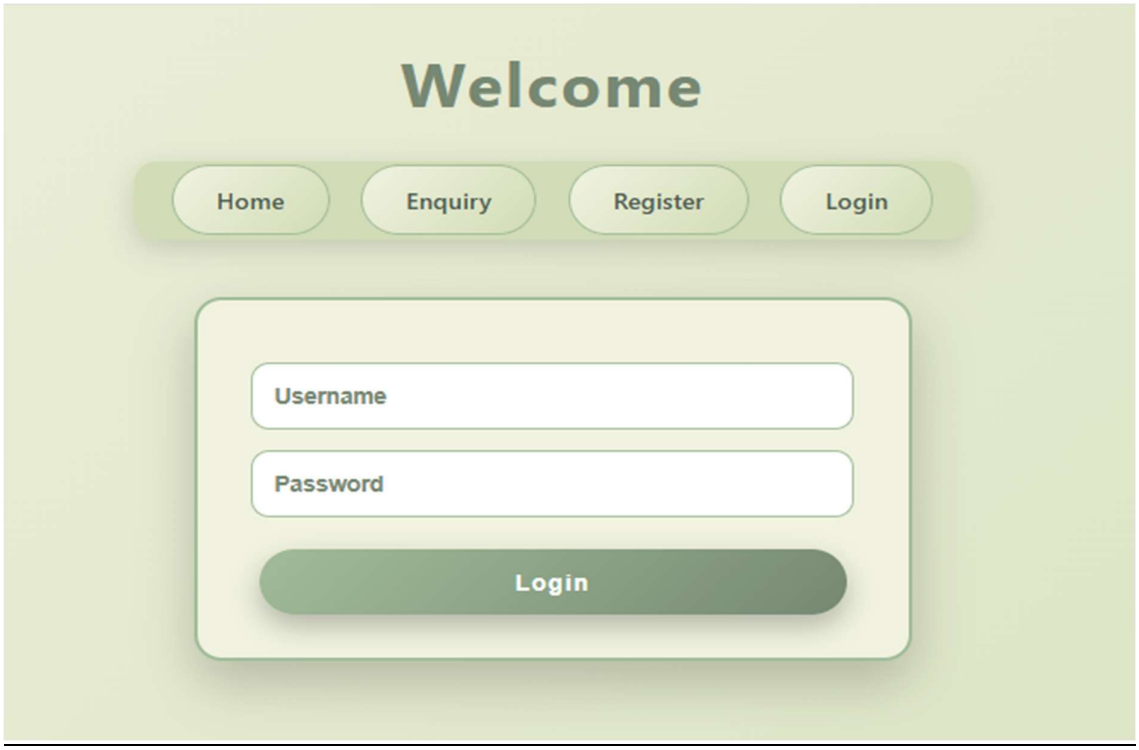



```
mysql> select * from enquiries;
+----+-------+---------------------------+-------------------------------+
| id | name  | email                     | query                         |
+----+-------+---------------------------+-------------------------------+
|  1 | tk    | A@gmail.com               | hi                            |
|  2 | admin | ishitashetty127@gmail.com | hi mq                         |
|  3 | patil | az@gmail.com              | hi mq                         |
|  4 | ak    | AK@gmail.com              | hi ak                         |
|  5 | ak    | chetanmaurya02@gmail.com  | hi mq                         |
|  6 | Adnan | adnan.khan@gmail.com      | I need help regarding website |
+----+-------+---------------------------+-------------------------------+
```


Welcome
Home
Enquiry
Register
Login
Full Name
Email
Phone
Username
Password
Confirm Password
dd-mm-yyyy
Register




```
mysql> select * from user;
+----+-------+--------------------------+------------+----------+----------+------------+
| id | name  | email                    | phone      | username | password | dob        |
+----+-------+--------------------------+------------+----------+----------+------------+
|  1 | tk    | chetanmaurya02@gmail.com | 1234567890 | admin    | 123456   | 2025-11-01 |
|  2 | admin | A@gmail.com              | 1234567890 | admin    | 123456   | 2025-10-29 |
|  3 | Adnan | adnan.khan1@gmail.com    | 9876543211 | ad123    | 123      | 2025-01-01 |
+----+-------+--------------------------+------------+----------+----------+------------+
3 rows in set (0.00 sec)
```



Practical 7

Create an application in ReactJS to implement component life cycle

ReactJS Component Life Cycle – Concept

• In React, component lifecycle methods provide a way to run code at specific points in a component's lifetime.

• This concept is mainly applicable to class components, which were the original way to manage lifecycle events in React.

• With the introduction of React Hooks, similar lifecycle behaviors can now be implemented in functional components.

Class Components Lifecycle

• Mounting Phase

• This phase covers the life of a component from its initial creation until it is rendered in the DOM.

• constructor(props):

- Called before the component is mounted.
- Used to initialize state and bind methods.

• static getDerivedStateFromProps(nextProps, prevState):

- Used to derive state from props.
- Called before rendering during initial mount and updates.

• render():

- Returns the JSX that defines the UI of the component.
- It is a pure method and does not modify the component state.

• componentDidMount():

- Called immediately after the component is mounted.
- Used for DOM-related operations or data fetching.

• Updating Phase

• This phase occurs when there is a change in component state or props.

• static getDerivedStateFromProps(nextProps, prevState):

- Invoked when the component receives new props.

• shouldComponentUpdate(nextProps, nextState):

- Returns a boolean value.
- Helps optimize performance by avoiding unnecessary re-rendering.

• render():

- Re-renders the component with updated data.

• getSnapshotBeforeUpdate(prevProps, prevState):

- Called just before the DOM is updated.
- Captures information from the DOM (snapshot).

• componentDidUpdate(prevProps, prevState, snapshot):

- Called after the component is updated.
- Used for operations that require updated DOM data.

• Unmounting Phase

• This phase occurs when a component is removed from the DOM.

• componentWillUnmount():

- Called just before the component is destroyed.
- Used for cleanup operations like removing event listeners or cancelling API calls.

• Error Handling Phase

• React provides special lifecycle methods for handling errors in components.

• static getDerivedStateFromError(error):

- Triggered when an error occurs in a child component.
- Used to render fallback UI.

• componentDidCatch(error, info):

- Used to log error details or report errors.

Steps to Create React Application (Terminal – VS Code)

• Open terminal in VS Code

• Run the following commands:

• npx create-react-app firstapp

• npm install -g npx

• npm install web-vitals

• cd firstapp

• npm start

Result

• Thus, a ReactJS application demonstrating the component life cycle was successfully created and executed.

Aim:1.Create an application in ReactJS to implement component life cycle

Using ReactJs print 'Hello World' on the web browser.

Code:

App.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
//import App2 from './app2';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
reportWebVitals();
```

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
```

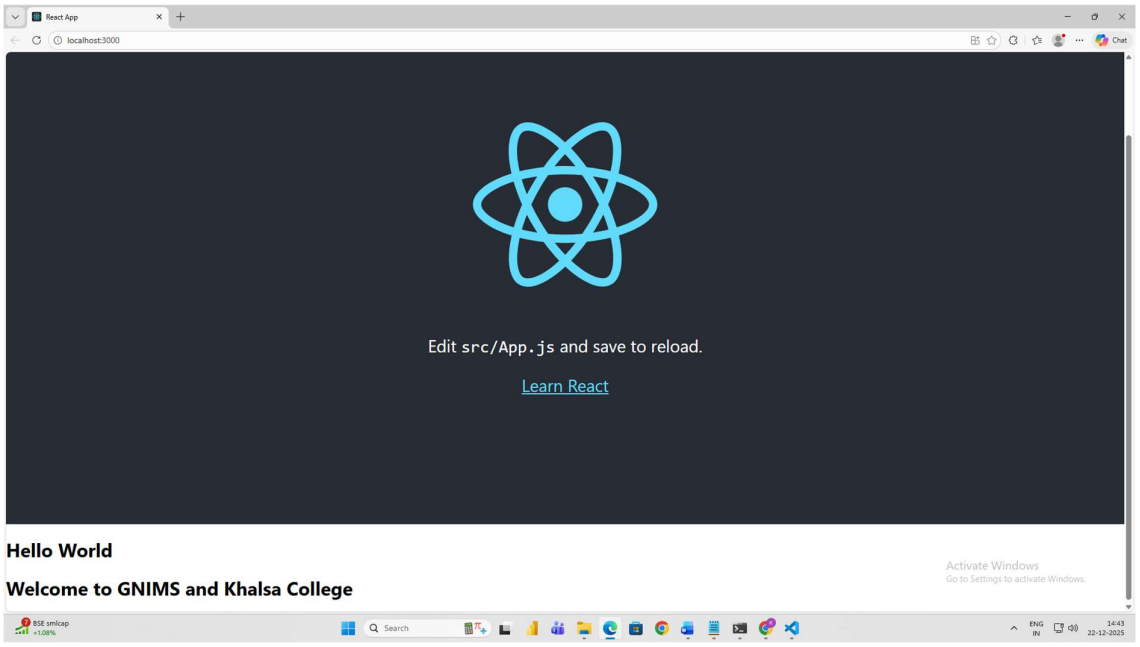
```
    content="Web site created using create-react-app"
  />
  <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
  <!--
    manifest.json provides metadata used when your web app is installed on a
    user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
  -->
  <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
  <!--
    Notice the use of %PUBLIC_URL% in the tags above.
    It will be replaced with the URL of the `public` folder during the build.
    Only files inside the `public` folder can be referenced from the HTML.

    Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
    work correctly both with client-side routing and a non-root public URL.
    Learn how to configure a non-root public URL by running `npm run build`.
  -->
  <title>React App</title>
 </head>
 <body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
```

```
    To create a production bundle, use `npm run build` or `yarn build`.
  -->

  <h1>Hello World</h1>
    <h1>Welcome to GNIMS and Khalsa College</h1>

 </body>
</html>
```

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
//import App2 from './app2';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
 <React.StrictMode>
  <App />
 </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Output:


React App
localhost:3000
Edit src/App.js and save to reload.
Learn React
Hello World
Welcome to GNIMS and Khalsa College
Activate Windows

Practical 7.B

Aim: 2.Using ReactJS components (Hooks) display different emojis on the screen. print the name of atleast 10 emoji

Code:

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App2 from './myapp2'
import reportWebVitals from './reportWebVitals';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App2 />
  </React.StrictMode>
);
reportWebVitals();
```

myapp2.js

```
import React, { useState } from "react";
import "./App.css";

function App() {
  const [selectedEmoji, setSelectedEmoji] = useState({
    em: " 🎉 ",
    name: "Party Popper",
    category: "Celebration",
    code: "U+1F389",
    description: "A party popper used to celebrate events."
  });
```

```
const [darkMode, setDarkMode] = useState(false);


const emojiArray = [
  // Smileys & Emotion
  { em: "😀", name: "Grinning Face", category: "Smileys & Emotion", code: "U+1F600", description: "A happy smiling face." },
  { em: "😃", name: "Smiling Face with Open Mouth", category: "Smileys & Emotion", code: "U+1F603", description: "A joyful smiling face." },
  { em: "😄", name: "Smiling Face with Open Mouth and Smiling Eyes", category: "Smileys & Emotion", code: "U+1F604", description: "Happily smiling with eyes closed." },
  { em: "😁", name: "Grinning Face with Smiling Eyes", category: "Smileys & Emotion", code: "U+1F601", description: "Wide grin showing happiness." },
  { em: "😆", name: "Laughing Face", category: "Smileys & Emotion", code: "U+1F606", description: "Laughing heartily with closed eyes." },
  { em: "😅", name: "Smiling Face with Sweat", category: "Smileys & Emotion", code: "U+1F605", description: "A smile with a bead of sweat." },
  { em: "😂", name: "Tears of Joy", category: "Smileys & Emotion", code: "U+1F602", description: "Laughing so hard that tears come out." },
  { em: "🤣", name: "Rolling on the Floor Laughing", category: "Smileys & Emotion", code: "U+1F923", description: "Extreme laughter." },
  { em: "🥰", name: "Smiling Face with Hearts", category: "Smileys & Emotion", code: "U+1F970", description: "Smiling face surrounded by hearts." },
  { em: "😍", name: "Heart Eyes", category: "Smileys & Emotion", code: "U+1F60D", description: "Face with heart-shaped eyes." },
  { em: "😎", name: "Smiling with Sunglasses", category: "Smileys & Emotion", code: "U+1F60E", description: "Cool face with sunglasses." },
  { em: "😡", name: "Pouting Face", category: "Smileys & Emotion", code: "U+1F621", description: "An angry or frustrated face." },
  { em: "😱", name: "Screaming Face", category: "Smileys & Emotion", code: "U+1F631", description: "Face screaming in fear or shock." },
  { em: "🤯", name: "Exploding Head", category: "Smileys & Emotion", code: "U+1F92F", description: "Mind blown!" },
  { em: "🥳", name: "Partying Face", category: "Smileys & Emotion", code: "U+1F973", description: "A party celebration face." },
```

```
// People & Body
{ em: "🤦", name: "Face Palm", category: "People & Body", code: "U+1F486", description: "Expressing frustration or embarrassment." },
{ em: "🤦‍♀️", name: "Woman Pouting", category: "People & Body", code: "U+1F64E-200D-2640", description: "A woman showing disappointment or disapproval." },
{ em: "💪", name: "Flexed Biceps", category: "People & Body", code: "U+1F4AA", description: "Showing strength." },
{ em: "🙏", name: "Folded Hands", category: "People & Body", code: "U+1F64F", description: "Praying or thank you gesture." },
{ em: "🤝", name: "Handshake", category: "People & Body", code: "U+1F91D", description: "Handshake gesture." },

// Animals & Nature
{ em: "🐶", name: "Dog Face", category: "Animals & Nature", code: "U+1F436", description: "A cute dog face." },
{ em: "🐱", name: "Cat Face", category: "Animals & Nature", code: "U+1F431", description: "A cute cat face." },
{ em: "🐭", name: "Mouse Face", category: "Animals & Nature", code: "U+1F42D", description: "A small mouse face." },
{ em: "🐹", name: "Hamster Face", category: "Animals & Nature", code: "U+1F439", description: "A hamster face." },
{ em: "🐰", name: "Rabbit Face", category: "Animals & Nature", code: "U+1F430", description: "A rabbit face." },
{ em: "🦊", name: "Fox Face", category: "Animals & Nature", code: "U+1F98A", description: "A sly fox face." },
{ em: "🐻", name: "Bear Face", category: "Animals & Nature", code: "U+1F43B", description: "A bear face." },
{ em: "🐼", name: "Panda Face", category: "Animals & Nature", code: "U+1F43C", description: "A cute panda face." },
{ em: "🦁", name: "Lion Face", category: "Animals & Nature", code: "U+1F981", description: "A majestic lion face." },
{ em: "🦄", name: "Unicorn", category: "Animals & Nature", code: "U+1F984", description: "A mythical unicorn." },
```

{ em: "🌞", name: "Sun with Face", category: "Animals & Nature", code: "U+1F31E", description: "The sun with a smiling face." },

{ em: "🌙", name: "Crescent Moon", category: "Animals & Nature", code: "U+1F319", description: "A crescent moon." },

{ em: "⭐", name: "Star", category: "Animals & Nature", code: "U+2B50", description: "A bright star." },

{ em: "🌈", name: "Rainbow", category: "Animals & Nature", code: "U+1F308", description: "A rainbow." },

// Food & Drink

{ em: "🍎", name: "Red Apple", category: "Food & Drink", code: "U+1F34E", description: "A red apple fruit." },

{ em: "🍌", name: "Banana", category: "Food & Drink", code: "U+1F34C", description: "A ripe yellow banana." },

{ em: "🍕", name: "Pizza", category: "Food & Drink", code: "U+1F355", description: "A slice of pizza." },

{ em: "🍔", name: "Hamburger", category: "Food & Drink", code: "U+1F354", description: "A classic burger." },

{ em: "🍣", name: "Sushi", category: "Food & Drink", code: "U+1F363", description: "Sushi rolls." },

{ em: "🍩", name: "Doughnut", category: "Food & Drink", code: "U+1F369", description: "A sweet doughnut." },

{ em: "🍪", name: "Cookie", category: "Food & Drink", code: "U+1F36A", description: "A chocolate chip cookie." },

{ em: "🍫", name: "Chocolate", category: "Food & Drink", code: "U+1F36B", description: "Chocolate bar." },

// Activity & Sports

{ em: "⚽", name: "Soccer Ball", category: "Activity", code: "U+26BD", description: "A soccer ball." },

{ em: "🏀", name: "Basketball", category: "Activity", code: "U+1F3C0", description: "A basketball." },

{ em: "🏈", name: "Football", category: "Activity", code: "U+1F3C8", description: "An American football." },

{ em: " 🎾 ", name: "Tennis Ball", category: "Activity", code: "U+1F3BE", description: "A tennis ball." },

{ em: " 🏓 ", name: "Ping Pong", category: "Activity", code: "U+1F3D3", description: "Table tennis paddle and ball." },

{ em: " 🎳 ", name: "Bowling", category: "Activity", code: "U+1F3B3", description: "A bowling ball with pins." },

{ em: " 🎮 ", name: "Video Game", category: "Activity", code: "U+1F3AE", description: "A game controller." },

{ em: " 🏊 ", name: "Swimmer", category: "Activity", code: "U+1F3CA-200D-2642", description: "A person swimming." },

{ em: " 🚴 ", name: "Cyclist", category: "Activity", code: "U+1F6B4-200D-2640", description: "A person riding a bicycle." },

{ em: " 🎯 ", name: "Bullseye", category: "Activity", code: "U+1F3AF", description: "A target for accuracy." },

{ em: " 🏹 ", name: "Bow and Arrow", category: "Activity", code: "U+1F3F9", description: "Archery or Cupid's bow." },

{ em: " 🛹 ", name: "Skateboard", category: "Activity", code: "U+1F6F9", description: "A skateboard." },

{ em: " 🎲 ", name: "Dice", category: "Activity", code: "U+1F3B2", description: "A dice for games." },

// Travel & Places

{ em: " 🚀 ", name: "Rocket", category: "Travel & Places", code: "U+1F680", description: "A rocket launching into space." },

{ em: " ✈️ ", name: "Airplane", category: "Travel & Places", code: "U+2708", description: "An airplane." },

{ em: " 🚗 ", name: "Car", category: "Travel & Places", code: "U+1F697", description: "A car emoji." },

{ em: " 🛸 ", name: "Flying Saucer", category: "Travel & Places", code: "U+1F6F8", description: "A UFO or flying saucer." },

// Objects & Symbols

{ em: " 🎤 ", name: "Microphone", category: "Objects", code: "U+1F3A4", description: "A microphone." },

```
  { em: " 🎧 ", name: "Headphones", category: "Objects", code: "U+1F3A7", description: "A pair of headphones." },

  { em: " 📷 ", name: "Camera", category: "Objects", code: "U+1F4F7", description: "A camera." },

  { em: " 📺 ", name: "Television", category: "Objects", code: "U+1F4FA", description: "A TV set." },

  { em: " ⚡ ", name: "High Voltage", category: "Symbols", code: "U+26A1", description: "Lightning bolt." },

  { em: " 🔥 ", name: "Fire", category: "Symbols", code: "U+1F525", description: "A flame." },

  { em: " 💧 ", name: "Droplet", category: "Symbols", code: "U+1F4A7", description: "A water droplet." },

  { em: " ❄️ ", name: "Snowflake", category: "Symbols", code: "U+2744", description: "A snowflake." },

 ];


 // Hardcoded emojis outside array

 const hardcodedEmojis = [

  { em: " 🎯 ", name: "Bullseye", category: "Activity", code: "U+1F3AF", description: "A target for accuracy." },

  { em: " 🏹 ", name: "Bow and Arrow", category: "Activity", code: "U+1F3F9", description: "Archery or Cupid's bow." },

  { em: " 🛹 ", name: "Skateboard", category: "Activity", code: "U+1F6F9", description: "A skateboard." },

  { em: " 🎲 ", name: "Dice", category: "Activity", code: "U+1F3B2", description: "A dice for games." },

 ];


 return (

  <div className={darkMode ? "app-container dark" : "app-container"}>

   <header>

    <h1>Emoji Picker 🎉</h1>

    <button className="theme-toggle" onClick={() => setDarkMode(!darkMode)}>
```

```
        {darkMode ? "Switch to Light Theme" : "Switch to Dark Theme"}
      </button>
    </header>

    <div className="main-content">
      <div className="emoji-section">
        <h3>Array Emojis</h3>
        <table className="emoji-table">
          <tbody>
            {Array.from({ length: Math.ceil(emojiArray.length / 6) }).map((_, rowIdx) => (
              <tr key={rowIdx}>
                {emojiArray.slice(rowIdx * 6, rowIdx * 6 + 6).map((emoji, colIdx) => (
                  <td key={colIdx}>
                    <button onClick={() => setSelectedEmoji(emoji)}>
                      <span role="img" aria-label={emoji.name}>{emoji.em}</span>
                    </button>
                  </td>
                ))}
              </tr>
            ))}
          </tbody>
        </table>

        <h3>Hardcoded Emojis</h3>
        <ul className="emoji-ul">
          {hardcodedEmojis.map((emoji, idx) => (
            <li key={idx}>
              <button onClick={() => setSelectedEmoji(emoji)}>
                <span role="img" aria-label={emoji.name}>{emoji.em}</span>
              </button>
```

```
          </li>
        ))}
      </ul>
    </div>

    <div className="info-section">
      <h2>Selected Emoji Info</h2>
      <table className="info-table">
        <tbody>
          <tr>
            <td>Emoji:</td>
            <td><span className="emoji-image">{selectedEmoji.em}</span></td>
          </tr>
          <tr>
            <td>Name:</td>
            <td><span>{selectedEmoji.name}</span></td>
          </tr>
          <tr>
            <td>Category:</td>
            <td><span>{selectedEmoji.category}</span></td>
          </tr>
          <tr>
            <td>Codepoint:</td>
            <td><span>{selectedEmoji.code}</span></td>
          </tr>
          <tr>
            <td>Description:</td>
            <td><span>{selectedEmoji.description}</span></td>
          </tr>
        </tbody>
```

```
          </table>
        </div>
      </div>
    </div>
  );
}

export default App;
```

app.css

```
/* Reset & Base Styles */
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
  font-family: 'Arial', sans-serif;
}

body {
  background: #f0f4f8;
  color: #222;
  transition: background 0.3s, color 0.3s;
}

.app-container {
  max-width: 1200px;
  margin: auto;
  padding: 20px;
  transition: background 0.3s, color 0.3s;
```

```
}

/* Dark Theme */
.app-container.dark {
  background: #1e1e1e;
  color: #fff;
}

/* Header */
header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 20px;
}

header h1 {
  font-size: 2rem;
  color: #4f46e5;
}

.theme-toggle {
  padding: 10px 16px;
  cursor: pointer;
  border-radius: 8px;
  border: none;
  background: #4f46e5;
  color: #fff;
  font-weight: bold;
  font-size: 0.9rem;
```

```
  transition: all 0.3s;
}

.theme-toggle:hover {
  opacity: 0.8;
  transform: scale(1.05);
}

/* Main Content Layout */
.main-content {
  display: flex;
  gap: 30px;
  flex-wrap: wrap;
}

/* Emoji Section */
.emoji-section {
  flex: 2;
}

/* Emoji Table */
.emoji-table {
  width: 100%;
  border-collapse: collapse;
  margin-bottom: 20px;
}

.emoji-table td {
  padding: 10px;
  text-align: center;
```

```
}

.emoji-table button {
  font-size: 32px;
  border: none;
  background: #fff;
  cursor: pointer;
  border-radius: 10px;
  padding: 10px;
  transition: 0.2s;
}

.app-container.dark .emoji-table button {
  background: #333;
  color: #fff;
}

.emoji-table button:hover {
  transform: scale(1.2);
  border: 2px solid #4f46e5;
}

/* Emoji UL List */
.emoji-ul {
  list-style: none;
  padding: 0;
  display: flex;
  flex-wrap: wrap;
  margin-top: 10px;
}
```

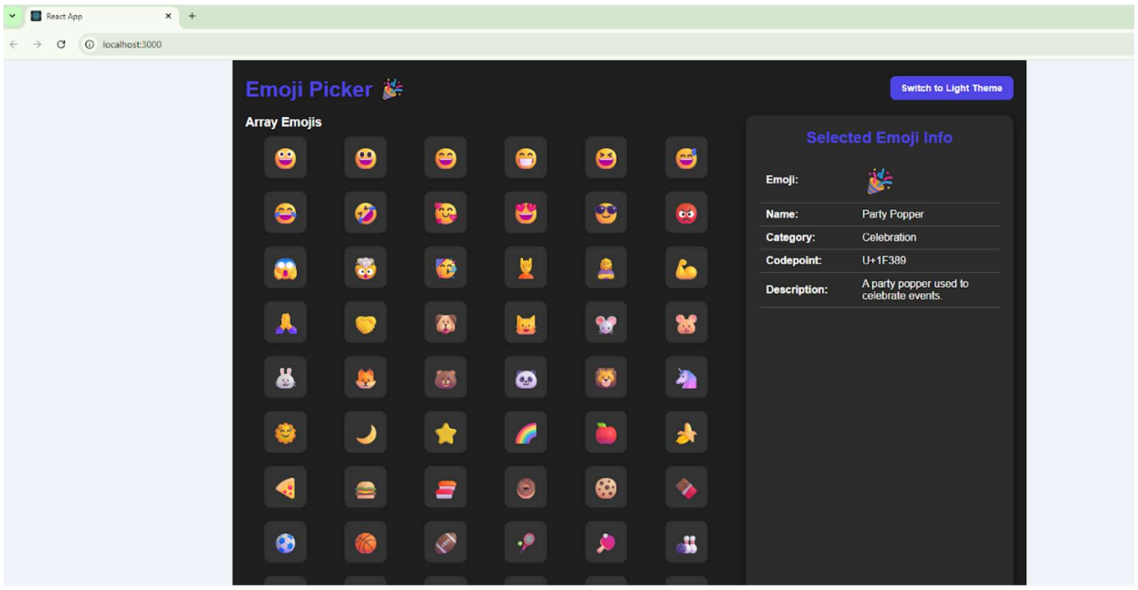
```
.emoji-ul li {
  margin: 8px;
}

.emoji-ul button {
  font-size: 32px;
  border: none;
  background: #fff;
  padding: 8px 12px;
  border-radius: 10px;
  cursor: pointer;
  transition: 0.2s;
}

.app-container.dark .emoji-ul button {
  background: #333;
  color: #fff;
}

.emoji-ul button:hover {
  transform: scale(1.2);
  border: 2px solid #4f46e5;
}

/* Info Section */
.info-section {
  flex: 1;
  background: #fff;
  border-radius: 12px;
```

```
  padding: 20px;
  box-shadow: 0 6px 18px rgba(0, 0, 0, 0.1);
  min-width: 250px;
  transition: background 0.3s, color 0.3s;
}

.app-container.dark .info-section {
  background: #2c2c2c;
  color: #fff;
  box-shadow: 0 6px 18px rgba(0, 0, 0, 0.4);
}

.info-section h2 {
  margin-bottom: 15px;
  color: #4f46e5;
  text-align: center;
}

.info-table {
  width: 100%;
  border-collapse: collapse;
}

.info-table td {
  padding: 8px 10px;
  border-bottom: 1px solid #ddd;
}

.app-container.dark .info-table td {
  border-bottom: 1px solid #555;
```

```
}

.info-table td:first-child {
  font-weight: bold;
  width: 40%;
}

.emoji-image {
  font-size: 40px;
}

/* Responsive Design */
@media (max-width: 900px) {
  .main-content {
    flex-direction: column;
  }

  .emoji-table td {
    padding: 5px;
  }

  .emoji-table button {
    font-size: 28px;
  }

  .emoji-ul button {
    font-size: 28px;
  }

  .info-section {
```

```
    min-width: 100%;
    margin-top: 20px;
  }
}
```

Implementation of the code(output)



Practical 8

Aim: Create an application to implement class and functional component in ReactJS

React JS Class and Functional Component Concept

In React, components can be defined in two primary ways: class components and functional components. Each has its own characteristics, advantages, and use cases.

Class Components Class components are ES6 classes that extend from React.Component. They are capable of holding and managing their own local state and lifecycle methods.

- Key Features of Class Components:

  - State Management: Class components can maintain internal state using the this.state object.

  - Lifecycle Methods: These components can utilize lifecycle methods (e.g., componentDidMount, componentDidUpdate, componentWillUnmount) to hook into different points in a component's lifecycle.

  - Render Method: Class components must implement a render() method that returns JSX.

Functional Components Functional components are simple JavaScript functions that accept props as an argument and return a React element. Initially, they were stateless but have evolved to support Hooks, which allow them to manage state and side effects.

- Key Features of Functional Components:

  - Simplicity: Easier to read and understand due to their functional nature without managing this context.

  - Hooks: Functional components support React Hooks, such as useState and useEffect, which allow for state management and side effects.

  - No Lifecycle Methods: Instead of lifecycle methods, functional components leverage the useEffect hook to handle component lifecycle events.

At Terminal (VS Code)

To set up your environment and run the application, execute the following commands in your terminal:

1. npx create-react-app firstapp
2. npm install -g npx
3. npm install web-vitals
4. cd firstapp
5. npm start

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
//import App from './App';
import reportWebVitals from './reportWebVitals';
import App from './product';
//import App from './App2'
//import App from './App3'

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

reportWebVitals();
```

counter.js

```
import React, { Component } from 'react'
export class Counter extends Component {
    constructor(props) {
      super(props)

      this.state = {
         count: 0
      }
    }
    updateClick=()=>{
       this.setState({count:this.state.count+1})
```

```
  }
 render() {
  const {count}=this.state;
  return (
   <div>
    <button onClick={this.updateClick}>Visited {count} times</button>
   </div>
  )
 }
}

export default Counter;
```

product.js

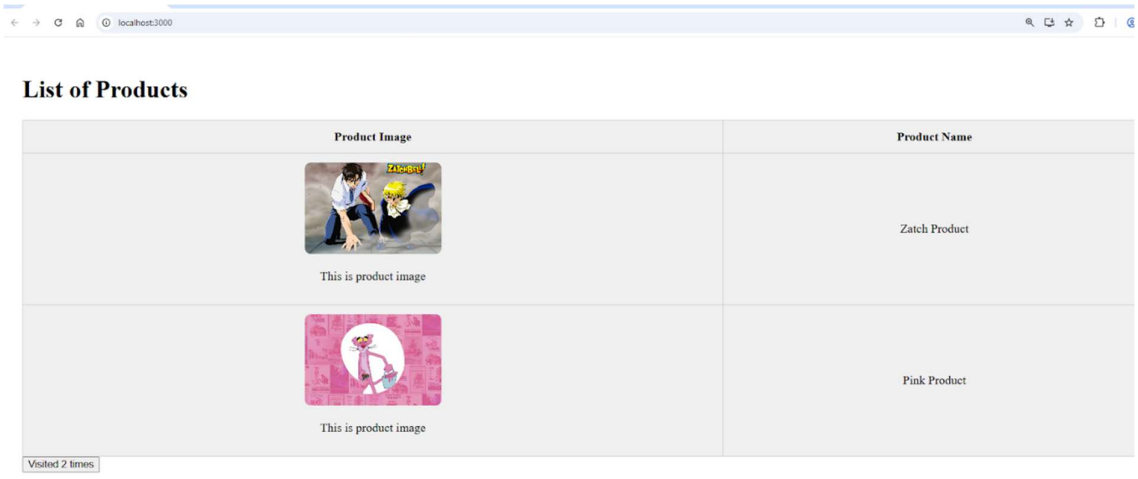
```
import React from 'react';
import './App.css';
import ProductImg from './productimg.js';
import pimg from './zatch.jpg';
import pimg2 from './pink.jpg';
import Counter from './Counter.js';

function App() {
 return (
  <div className="container">
   <h1>List of Products</h1>

   <table className="product-table">
    <thead>
     <tr>
      <th>Product Image</th>
```

```
            <th>Product Name</th>
          </tr>
        </thead>

        <tbody>
          <tr>
            <td>
              <ProductImg im={pimg} />
            </td>
            <td>Zatch Product</td>
          </tr>

          <tr>
            <td>
              <ProductImg im={pimg2} />
            </td>
            <td>Pink Product</td>
          </tr>
        </tbody>
      </table>

      <div>
        <Counter />
      </div>
    </div>
  );
}

export default App;
```

productimg.js

```
import React, { Component } from 'react';
import './App.css'; // to use the table image styles
export class ProductImg extends Component {
 render() {
   return (
     <div className="product-img-box">
       <img src={this.props.im} alt="Product" className="product-img" />
       <p className="product-text">This is product image</p>
     </div>
   );
 }
}
export default ProductImg;
```

Implementation of the code(output)



Practical 8.2

Code:

App.js

```
import './style.css';
import ProfileCard from './ProfileCard';
import messi from './messi.jpg';
import ronaldo from './ronaldo.webp';
function App() {
 const users = [
   { name: "Messi", profession: "Frontend Developer:Footballer", email:
"messi@example.com", profileImage: messi },
   { name: "Cristiano Ronaldo", profession: "Backend Developer :Footballer", email:
"ronaldo@example.com", profileImage: ronaldo },
 ];

 return (
  <div className="container">
    <h1>List of Users</h1>

    <table className="profile-table">
      <thead>
        <tr>
          <th>Profile Image</th>
          <th>Name</th>
          <th>Profession</th>
          <th>Email</th>
        </tr>
      </thead>
      <tbody>
        {users.map((user, index) => (
          <tr key={index}>
```

```
            <td>
              <ProfileCard
                name={user.name}
                profession={user.profession}
                email={user.email}
                profileImage={user.profileImage}
              />
            </td>
            <td>{user.name}</td>
            <td>{user.profession}</td>
            <td>{user.email}</td>
          </tr>
        ))}
      </tbody>
    </table>
  </div>
 );
}

export default App;
```

ProfileCard.js

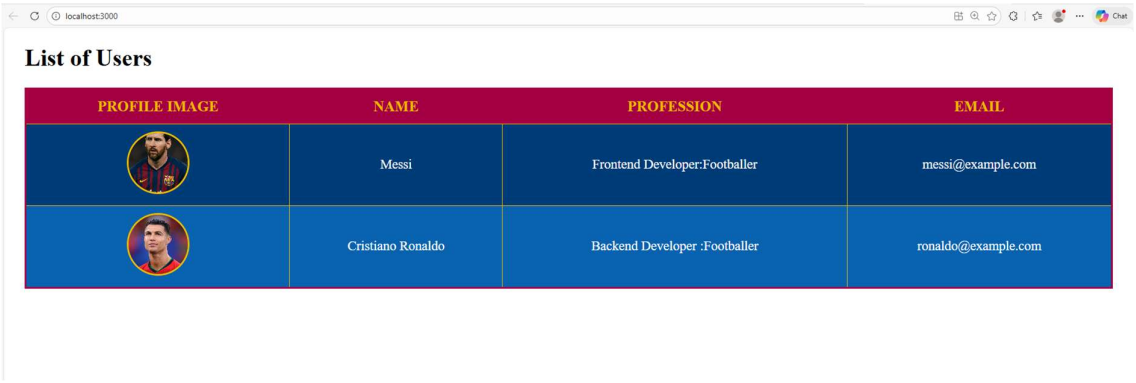
```
import React from 'react';
function ProfileCard({ profileImage }) {
 return (
  <img
    src={profileImage}
    alt="Profile"
    className="profile-image"
  />
```

```
  );
}

export default ProfileCard;
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './style.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
reportWebVitals();
```

Implementation of the code(output)



Practical 9

Aim: Create an application in ReactJS to import and export the files

ReactJS Application for File Import and Export - Overview Objective: Build a ReactJS application to allow users to import files (e.g., text, JSON, or CSV) for viewing or processing and export data as downloadable files.

Setup: Create a new React app using create-react-app. Install dependencies like file-saver for exporting files and optionally xlsx for handling Excel files.

File Import:

- Use an <input type="file" /> element to let users upload files.
- Read the file content using the FileReader API, which supports text, binary, and data URLs.
- Process or display the file content within the application as needed.

File Export:

- Prepare the data to be exported, converting it into a string or binary format.
- Use the Blob API to create a downloadable file.
- Leverage the file-saver package or browser APIs to trigger the download, specifying the file name and type.

Integration: Combine the import and export functionalities into reusable React components. Manage file data using State and Props to ensure seamless interaction.

Benefits:

1. Simplifies data handling for users.
2. Flexible: Can handle various file formats.
3. Enhances user experience by enabling efficient file management directly in the application.

Practical 10

Aim: Create an application to implement state and props

Understanding State and Props in React

In React, State and Props are fundamental concepts used to manage and pass data within applications. Here's a quick overview of both:

1. State:

- Definition: State refers to data that changes over time within a component. It is managed and controlled by the component itself.
- Usage: State is used to store dynamic data that can be modified, causing the component to re-render when the state changes.

- Key Points:
  - State is mutable and local to the component.
  - It is initialized using the useState hook in functional components or within the class component constructor.
  - Changes to state trigger re-renders of the component.
  - Example: A counter that increments when a button is clicked would use state to keep track of the current count.

2. Props:

- Definition: Props (short for properties) are immutable data passed from a parent component to a child component. They are used to provide data to child components.
- Usage: Props allow components to communicate and share data in a unidirectional way. Child components receive props from their parent components and can use them but cannot modify them.
- Key Points:
  - Props are read-only and cannot be changed by the component receiving them.
  - They help in making components reusable and modular by passing dynamic data from parent to child components.
  - Example: A parent component might pass a "name" prop to a child component to display the name dynamically.

Key Differences Between State and Props:

- State is managed within the component and can be changed, leading to re-renders.
- Props are passed to a component from its parent and cannot be altered by the receiving component.

In summary, State is for managing dynamic data within a component, while Props are for passing data between components.

Code:

index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
//import App from './App';
import reportWebVitals from './reportWebVitals';
import App from './mycart';
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
reportWebVitals();
```

Data.js

```
import SamsungImg from './images/SamsungImg.jpg';
import OneplusImg from './images/OneplusImg.jpg';
import RealmeImg from './images/RealmeImg.jpg';
import NokiaImg from './images/NokiaImg.jpg';

const Data= [
    {
        pname: 'Samsung Galaxy S22 5G',
        price: 72900,
        rating: 4.5,
        features: ['Android', '6.1 inch display', '8GB Ram'],
        additional: 'A smartphone with difference',
        img: SamsungImg
    },
    {
        pname: 'OnePlus Nord 2T 5G',
        price: 33999,
        rating: 4.2,
        features: ['Android', '6.1 inch display', '8GB Ram'],
```

```
      additional: 'A smartphone with difference',
      img: OneplusImg
   },
   {
      pname: 'Realme C11 2021',
price: 8900,
      rating: 3.2,
      features: ['Android', '6.1 inch display', '8GB Ram'],
      additional: 'A smartphone with difference',
      img: RealmeImg
   },
   {
      pname: 'NOKIA 555',
price: 6000,
      rating: 3.2,
      features: ['Sumbiosis', '4.1 inch display', '2GB Ram'],
      additional: 'A old fashioned phone',
      img: NokiaImg
   }
]

export default Data;
```

ProductCart.js

```
import React from 'react';
import './ProductCart.css';
function ProductCard({ img, additional, pname, price, rating, features }) {
 return (
```

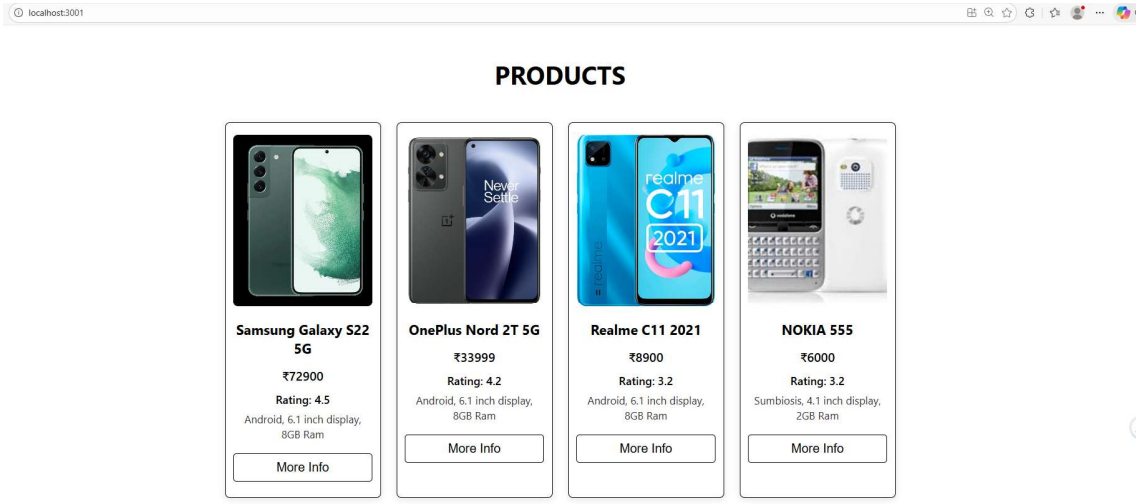
```
    <div className='product-wrapper'>
      <img src={img} alt={pname} className="product-img" />
      <h2>{pname}</h2>
      <h3>₹{price}</h3>
      <h4>Rating: {rating}</h4>
      <div>{features.join(', ')}</div>
      <button onClick={() => alert(additional)}>More Info</button>
    </div>
  );
}

export default ProductCard;
```

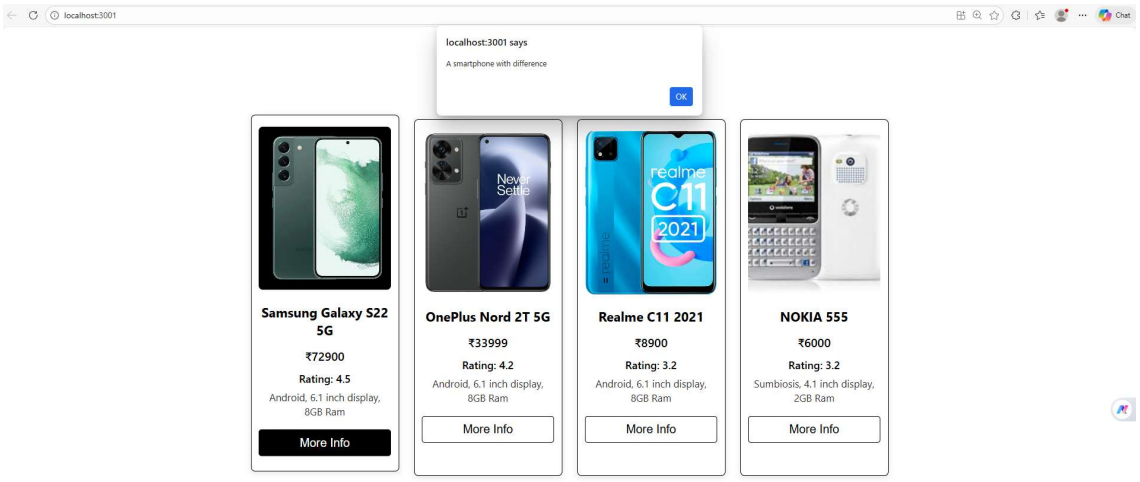
mycart.js

```
import './App.css';
import ProductCard from './ProductCart';
import Data from './Data';

function MyCart() {
  return (
    <div className="wrapper">
      <h1>PRODUCTS</h1>
      {Data.map(product => (
        <ProductCard
          key={product.pname}
          pname={product.pname}
          price={product.price}
          rating={product.rating}
          features={product.features}
          additional={product.additional}
```

```
        img={product.img}
      />
    ))}
  </div>
 );
}

export default MyCart;
```

Implementation of the code(output)





Practical 9.2

Code:

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
//import App from './App';
import App from './App2';
import reportWebVitals from './reportWebVitals';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
reportWebVitals();
```

App2.js

```
import React from "react";
import StudentResume from "./StudentResume";
import students from "./data";
import "./Style2.css";

function App() {
  return (
    <div className="container">
      <h1 className="title">Student Resumes</h1>

      <div className="grid">
        {students.map((student, index) => (
```

```
        <StudentResume key={index} student={student} />
      ))}
    </div>
  </div>
 );
}
export default App;
```

Style2.css

```
body {
 margin: 0;
 padding: 0;
 font-family: "Arial", sans-serif;
 background-color: #E6E9AF;
}

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

.title {
 color: #22177A;
 font-size: 36px;
 margin-bottom: 20px;
}

.grid {
 display: grid;
 grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
```

```
  gap: 20px;
  padding: 20px;
}

.resume-card {
  background: #605EA1;
  color: white;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0px 4px 10px rgba(0,0,0,0.2);
  transition: 0.3s;
}

.resume-card:hover {
  background: #22177A;
  transform: translateY(-5px);
}

.toggle-btn {
  margin-top: 10px;
  padding: 10px 20px;
  background: #8EA3A6;
  border: none;
  color: #22177A;
  font-weight: bold;
  border-radius: 5px;
  cursor: pointer;
}

.toggle-btn:hover {
```

```
  background: #E6E9AF;
}

.extra-info {
  margin-top: 15px;
  background: rgba(230, 233, 175, 0.2);
  padding: 10px;
  border-radius: 8px;
}
```

StudentResume.jsx

```
import React, { useState } from "react";
import "./Style2.css";

function StudentResume({ student }) {
  const [showMore, setShowMore] = useState(false);

  return (
    <div className="resume-card">
      <h2>{student.firstName} {student.middleName} {student.lastName}</h2>

      <p><strong>Email:</strong> {student.email}</p>
      <p><strong>Area of Interest:</strong> {student.areaOfInterest}</p>
      <p><strong>Technical Knowledge:</strong> {student.technicalKnowledge}</p>

      <button onClick={() => setShowMore(!showMore)} className="toggle-btn">
        {showMore ? "Hide Details" : "Show More"}
      </button>

      {showMore && (
```

```
        <div className="extra-info">
          <p><strong>Hobbies:</strong> {student.hobbies}</p>
          <p><strong>Additional Work:</strong> {student.additionalWork}</p>
        </div>
      )}
    </div>
  );
}

export default StudentResume;
```

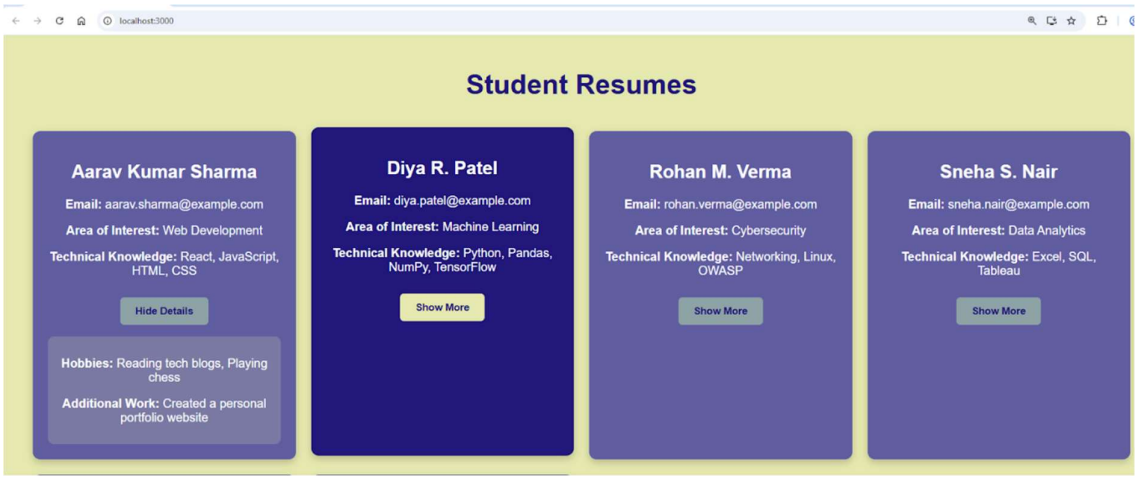
data.js

```
// src/data.js
const students = [
  {
    firstName: "Aarav",
    middleName: "Kumar",
    lastName: "Sharma",
    email: "aarav.sharma@example.com",
    areaOfInterest: "Web Development",
    technicalKnowledge: "React, JavaScript, HTML, CSS",
    hobbies: "Reading tech blogs, Playing chess",
    additionalWork: "Created a personal portfolio website"
  },
  {
    firstName: "Diya",
    middleName: "R.",
    lastName: "Patel",
    email: "diya.patel@example.com",
    areaOfInterest: "Machine Learning",
```

```
  technicalKnowledge: "Python, Pandas, NumPy, TensorFlow",
  hobbies: "Singing, photography",
  additionalWork: "Completed an ML internship at ABC Corp"
 },
 {
  firstName: "Rohan",
  middleName: "M.",
  lastName: "Verma",
  email: "rohan.verma@example.com",
  areaOfInterest: "Cybersecurity",
  technicalKnowledge: "Networking, Linux, OWASP",
  hobbies: "Cricket, Gym",
  additionalWork: "Conducted a workshop on Ethical Hacking"
 },
 {
  firstName: "Sneha",
  middleName: "S.",
  lastName: "Nair",
  email: "sneha.nair@example.com",
  areaOfInterest: "Data Analytics",
  technicalKnowledge: "Excel, SQL, Tableau",
  hobbies: "Drawing, Reading",
  additionalWork: "Analyzed survey data for college fest"
 },
 {
  firstName: "Karan",
  middleName: "J.",
  lastName: "Singh",
  email: "karan.singh@example.com",
  areaOfInterest: "Mobile App Development",
```

```
    technicalKnowledge: "Flutter, Dart",
    hobbies: "Traveling, Gaming",
    additionalWork: "Developed a college event app"
  },
  {
    firstName: "Meera",
    middleName: "P.",
    lastName: "Joseph",
    email: "meera.joseph@example.com",
    areaOfInterest: "AI & Robotics",
    technicalKnowledge: "C++, ROS, Arduino",
    hobbies: "Robotics competitions, Crafting",
    additionalWork: "Built an autonomous line-following robot"
  }
];

export default students;
```

Implementation of the code(output)



Practical 10

Aim: To create a React application that demonstrates the implementation of State and Props.

Write-up

Understanding State and Props in React

In React, State and Props are fundamental concepts used to manage and pass data within an application. The following sections explain each concept clearly.

1. State

Definition:

State refers to data that can change over time within a component. It is managed and controlled by the component itself.

Usage:

State is used to store dynamic data. Whenever the state changes, the component automatically re-renders to reflect the updated data.

Key Points:

- State is mutable and local to the component.
- It is initialized using the useState hook in functional components or in the constructor of class components.
- Any change in state triggers a re-render of the component.

Example:

A counter application where the displayed number increases when a button is clicked uses state to store and update the count value.

2. Props

Definition:

Props (short for properties) are read-only data passed from a parent component to a child component.

Usage:

Props enable communication between components by allowing a parent component to pass data to its child components.

Key Points:

- Props are immutable and cannot be modified by the receiving component.
- They help make components reusable and modular.
- Data flows in a one-way (unidirectional) manner, from parent to child.

Example:

A parent component passing a name prop to a child component so that the child can display the name dynamically.

Key Differences Between State and Props

State Props

Managed within the component Passed from parent to child

Mutable Immutable

Used for dynamic data Used for component communication

Changes cause re-render Cannot be changed by child

Conclusion

State is used for managing dynamic data within a component, whereas Props are used to pass data between components. Together, they form the foundation for building dynamic, reusable, and well-structured React applications.

Code:

src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
reportWebVitals();
```

src/index.css

```
body {
  margin: 0;
  font-family: 'Segoe UI', Roboto, Oxygen, Ubuntu, Cantarell, sans-serif;
  background: #FEEE91;
```

```
  color: #333;
}
code {
  font-family: Menlo, Monaco, Consolas, 'Courier New', monospace;
}
```

src/Product.js

```
import React, { Component } from 'react';
import './product.css';
const products = [
  { pr: ' 🍦 ', name: 'Ice Cream', price: 50 },
  { pr: ' 🍩 ', name: 'Donuts', price: 190 },
  { pr: '🍉', name: 'Watermelon', price: 30 }
];
class Product extends Component {
  state = {
    cart: [],
    total: 0
  };
  currencyOptions = {
    minimumFractionDigits: 2,
    maximumFractionDigits: 2,
  };
  getTotal = () => {
    return this.state.total.toLocaleString(undefined, this.currencyOptions);
  };
  add = (product) => {
    this.setState(state => ({
      cart: [...state.cart, product.name],
      total: state.total + product.price
    }));
```

```
};
remove = (product) => {
  this.setState(state => {
    const cart = [...state.cart];
    const index = cart.indexOf(product.name);
    if (index !== -1) {
      cart.splice(index, 1);
      return {
        cart,
        total: state.total - product.price
      };
    }
    return state;
  });
};
render() {
  return (
    <div className="wrapper">
      <h1>Shopping Cart</h1>
      <div className="cart-info">
        Items: {this.state.cart.length}
        <br />
        Total: ₹{this.getTotal()}
      </div>
      <div className="product-list">
        {products.map(product => (
          <div className="product-card" key={product.name}>
            <div className="product">
              <span role="img" aria-label={product.name}>{product.pr}</span>
            </div>
```

```
            <h3>{product.name}</h3>
            <p className="price">₹{product.price}</p>
            <button className="add" onClick={() => this.add(product)}>
              Add
            </button>
            <button className="remove" onClick={() => this.remove(product)}>
              Remove
            </button>
          </div>
        ))}
      </div>
    </div>
  );
 }
}
export default Product;
```

📌 **src/product.css**

```
.wrapper {
 padding: 20px;
 font-size: 20px;
 text-align: center;
 background: #FEEE91;
}
h1 {
 color: #FF5656;
}
.cart-info {
 margin-bottom: 20px;
 padding: 10px;
 background: #8CE4FF;
```

```
  border-radius: 8px;
  font-weight: bold;
}
.product-list {
  display: flex;
  justify-content: center;
  gap: 20px;
}
.product-card {
  background: #FFF;
  padding: 20px;
  width: 180px;
  border-radius: 12px;
  box-shadow: 0 4px 10px rgba(0,0,0,0.15);
}
.product span {
  font-size: 80px;
}
.price {
  font-weight: bold;
  color: #FFA239;
}
button {
  padding: 8px 15px;
  border: none;
  cursor: pointer;
  margin: 5px;
  color: white;
  font-size: 16px;
  border-radius: 6px;
```

```
}
.add {
  background: #FFA239;
}
.remove {
  background: #FF5656;
}
```

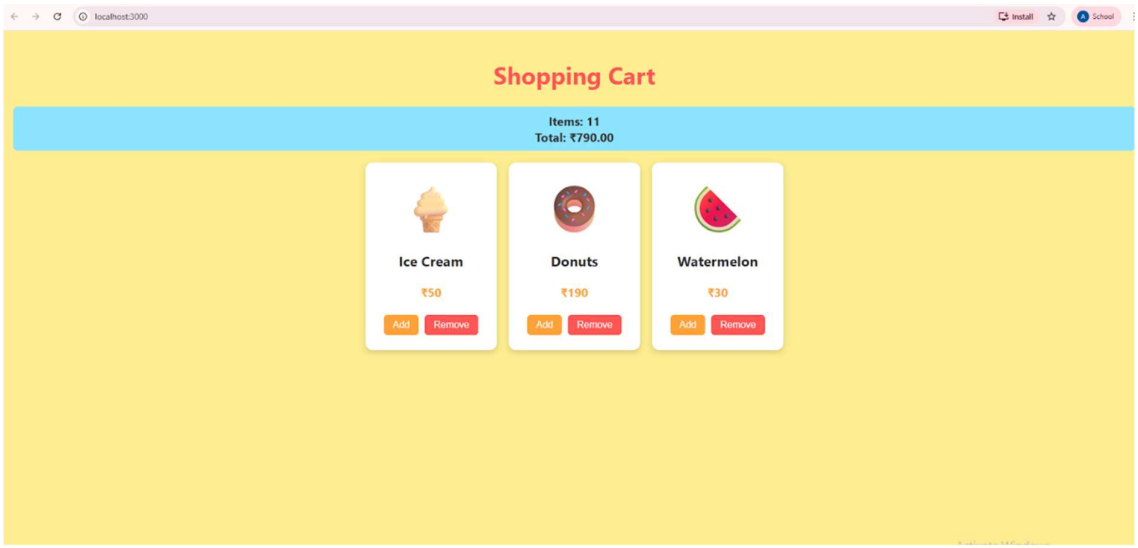
src/App.js

```
import './App.css';
import Product from './Product';
function App() {
  return <Product />;
}
export default App;
```

src/App.css

```
.App {
  text-align: center;
}
```

Implementation of the code(output)



Practical 11

Part 1

Create a website using react.js and node.js with the following specification

1. A company name - cyber IT need a solution to get their client data, enquiries , when it client fires the enquiry on the web page an alert should go to the admin.

2. Create the following web page for the web application :-

a. Home page

b. enquiry page

c. client data page (the client data should come from client database)

3. Create a the following admin web pages:-

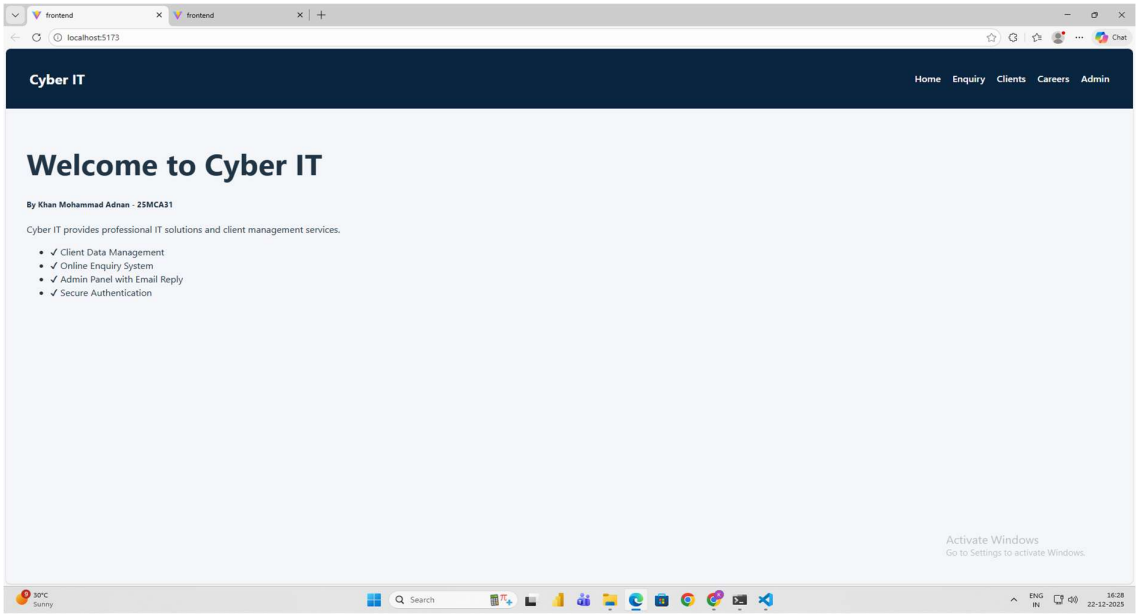
a. client data entry page (Should include client company logo)

b. Page to display the enquiries

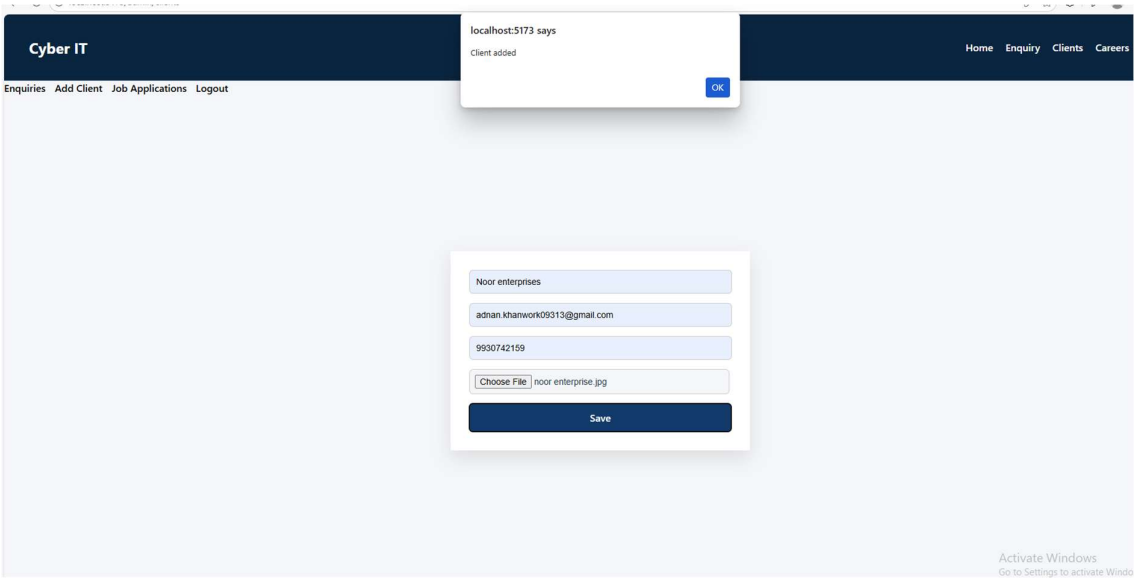
NOTE: the above to page mention should be username/password Authenticated

Cyber-IT logo

Home page

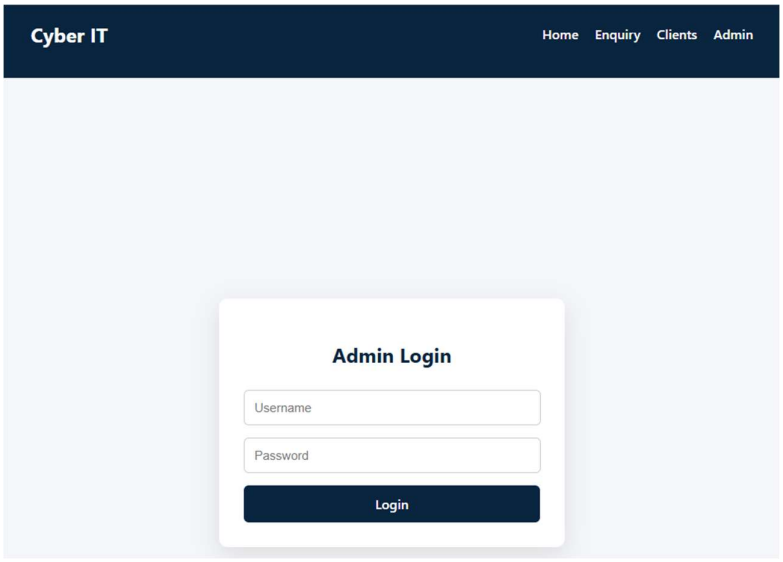


Add client

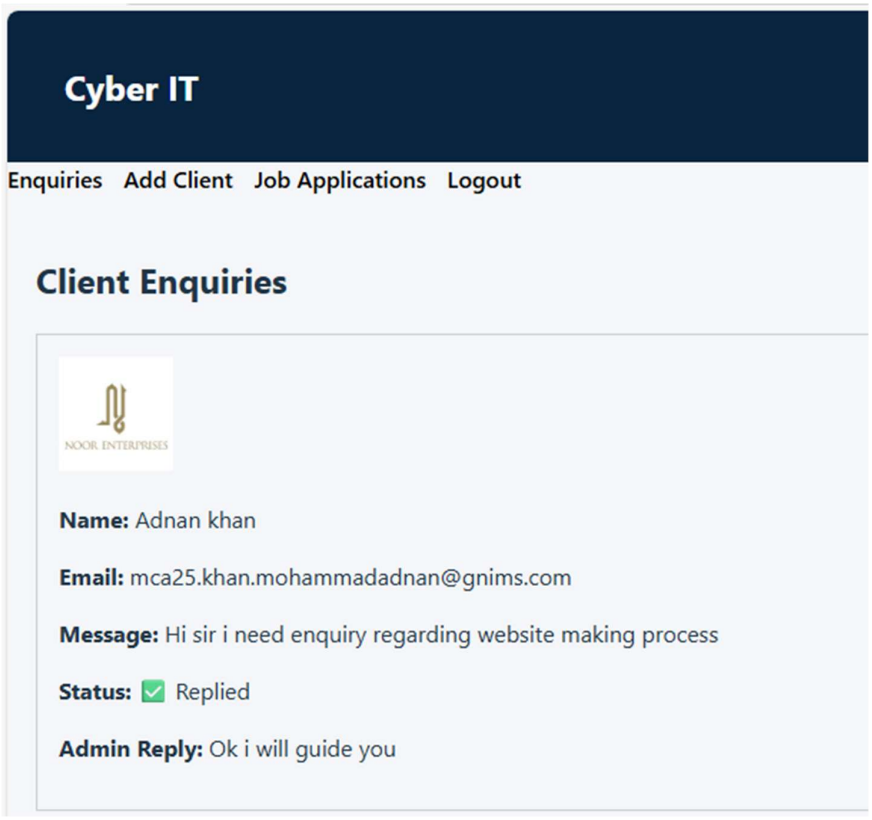



```
mysql> select * from clients;
+----+-----------------+-------------------------------+------------+-----------------------------------+
| id | company_name    | email                         | phone      | logo                              |
+----+-----------------+-------------------------------+------------+-----------------------------------+
|  1 | Noor enterprises | adnan.khanwork09313@gmail.com | 9930742159 | 1766398260659_noor enterprise.jpg |
+----+-----------------+-------------------------------+------------+-----------------------------------+
1 row in set (0.00 sec)
```

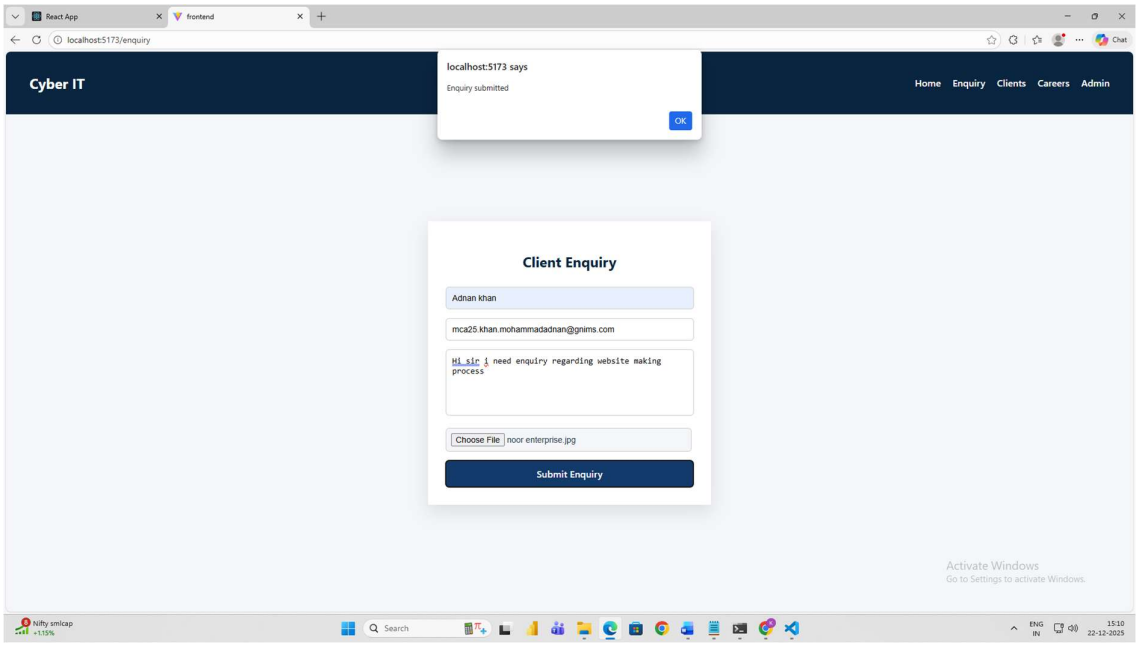
Admin login page

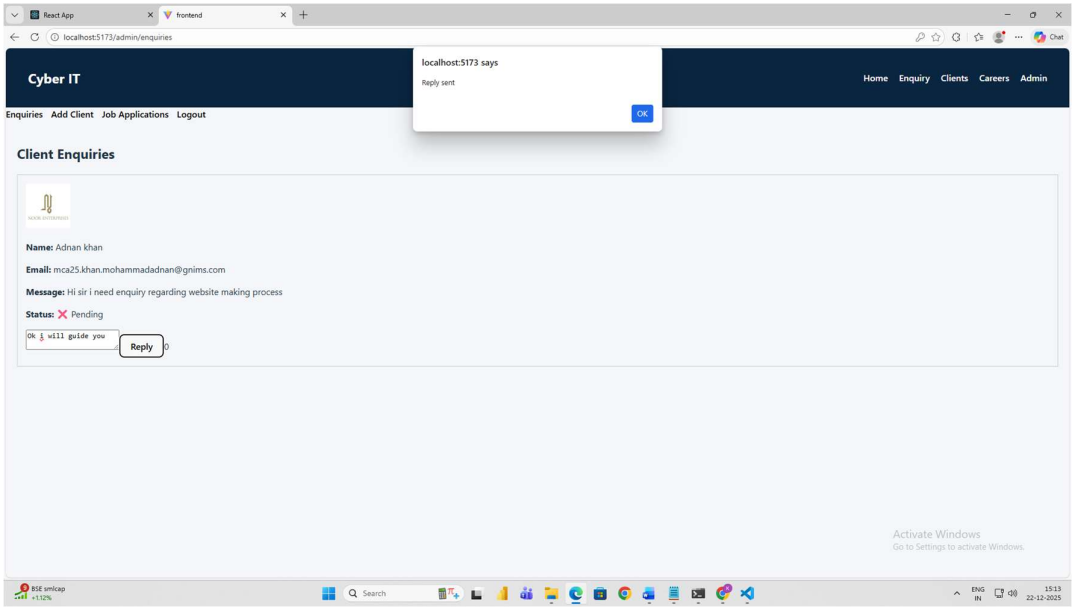


Admin details

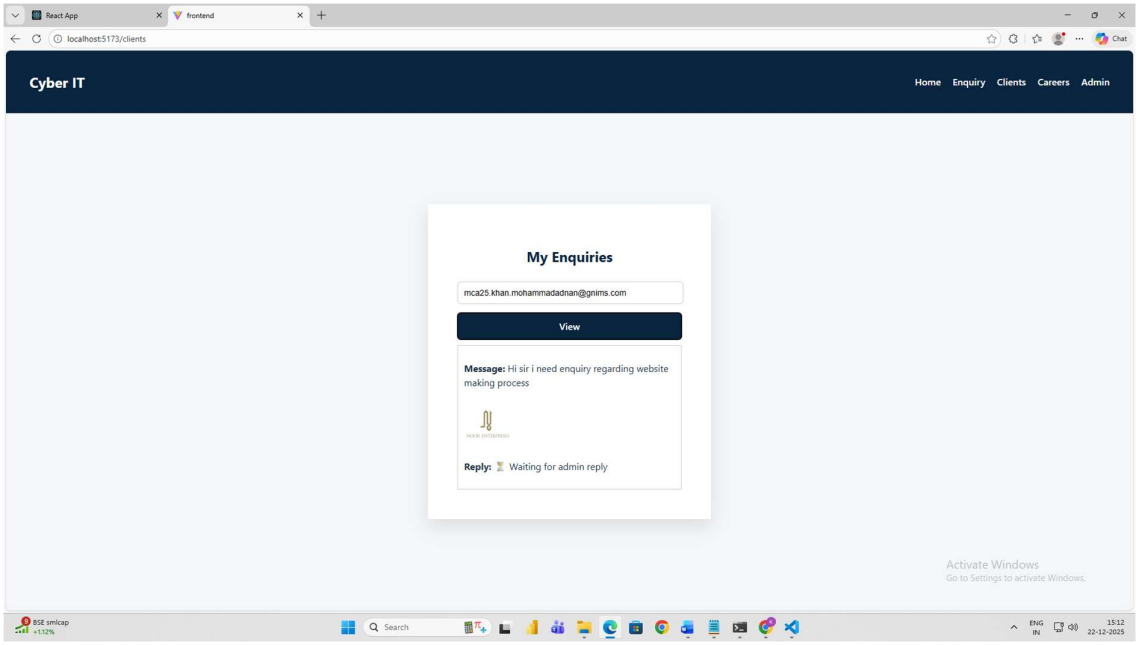


Client enquiry



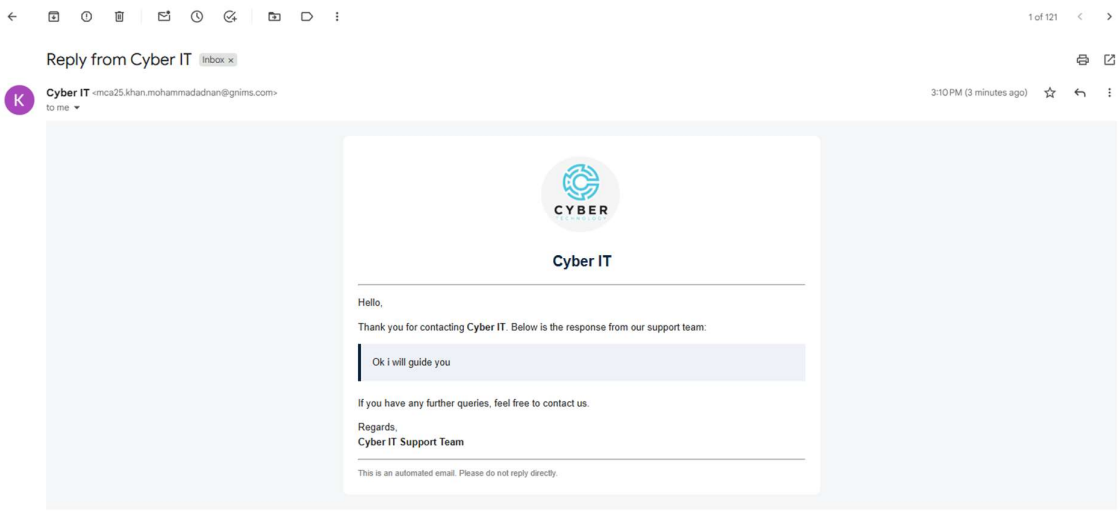


Client Enquiries Status




```
mysql> select * from enquiries;
+----+------------+-----------------------------------+-----------------------------------------+-----------------------------------+---------+-----------------------+
| id | name       | email                             | message                                 | logo                              | replied | reply                 |
+----+------------+-----------------------------------+-----------------------------------------+-----------------------------------+---------+-----------------------+
|  1 | Adnan khan | mca25.khan.mohammadadnan@gnims.com | Hi sir i need enquiry regarding website making process | 1766396453600_noor enterprise.jpg |       1 | Ok i will guide you |
+----+------------+-----------------------------------+-----------------------------------------+-----------------------------------+---------+-----------------------+
1 row in set (0.00 sec)
```

Email Integration


1 of 121
Reply from Cyber IT
Inbox ×
Cyber IT <mca25.khan.mohammadadnan@gnims.com>
to me
3:10 PM (3 minutes ago)
CYBER
Cyber IT
Hello,
Thank you for contacting Cyber IT. Below is the response from our support team:
Ok i will guide you
If you have any further queries, feel free to contact us.
Regards,
Cyber IT Support Team
This is an automated email. Please do not reply directly.

Practical 11.2

Part 2

In conjunction with the previous project (CYBERIT).

Add a page careers which will show all the openings in the company create a registration form for the applicants to apply. Make sure the form is capable of accepting passport photo(jpg/png) and resume in pdf format apart from other details

admin tab-> as admin logs in -> there should be tab as (show job applications)

when admin clicks on show job applications tab it should show all the details of the job appliers.

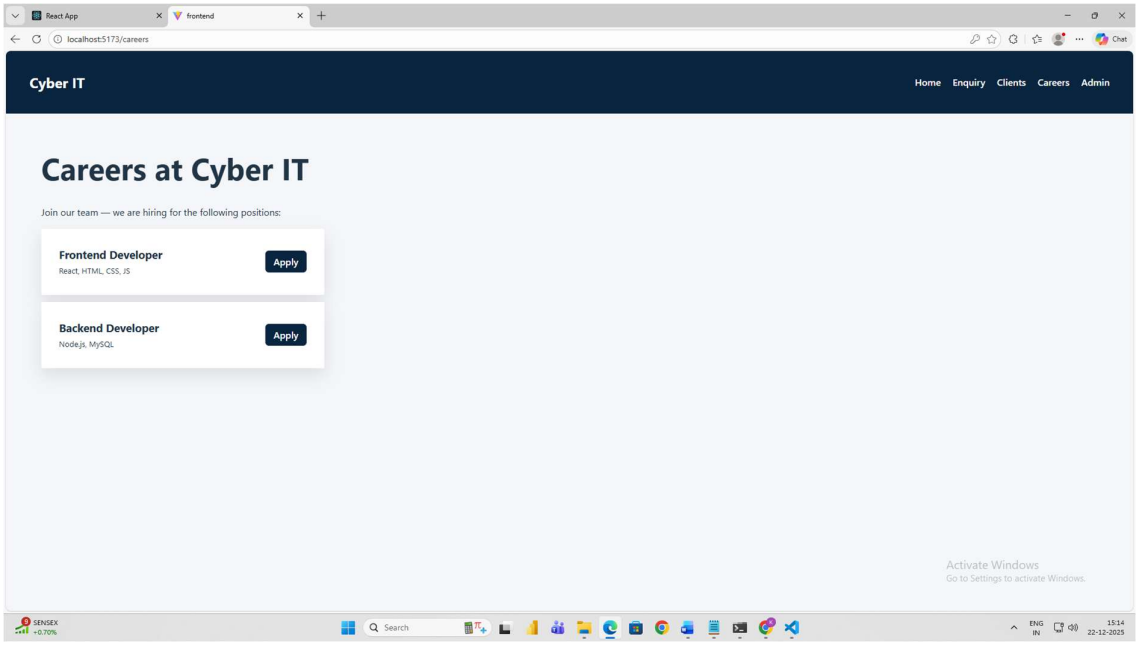
As admin clicks on the button beside every record should have 3 columns click here, accept or reject toggle button and sent email

click here - should open cv uploaded by job appliers.

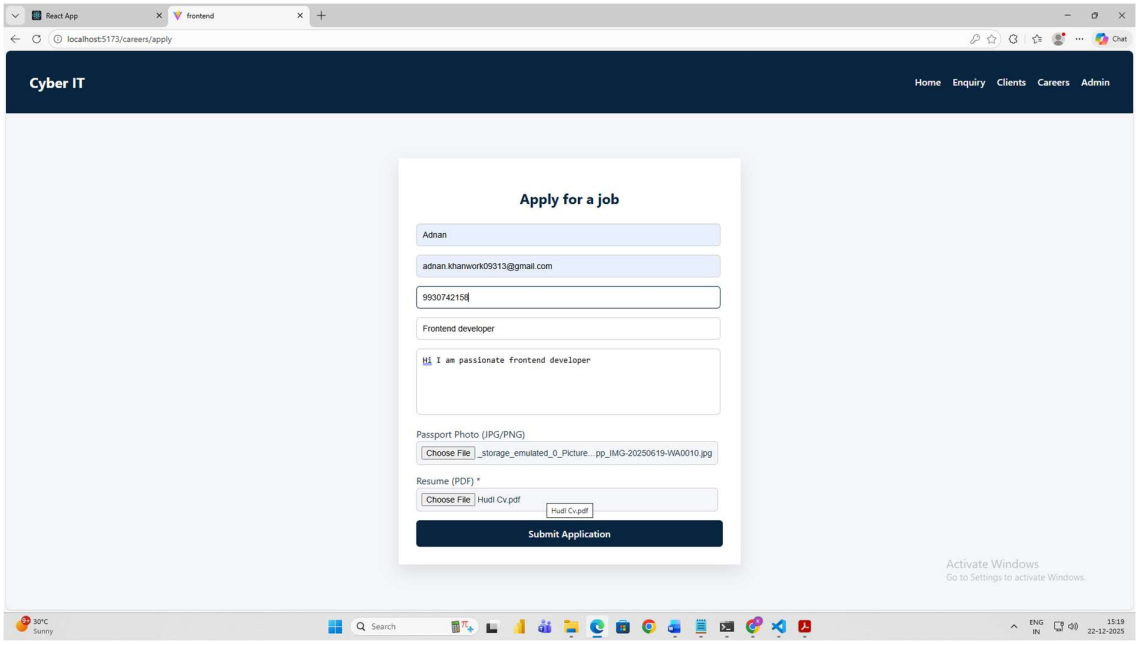
toggle button - To accept or reject the application

sent email - The email should be sent to that job applier once resume is accepted or rejected as per the selection by the hr.

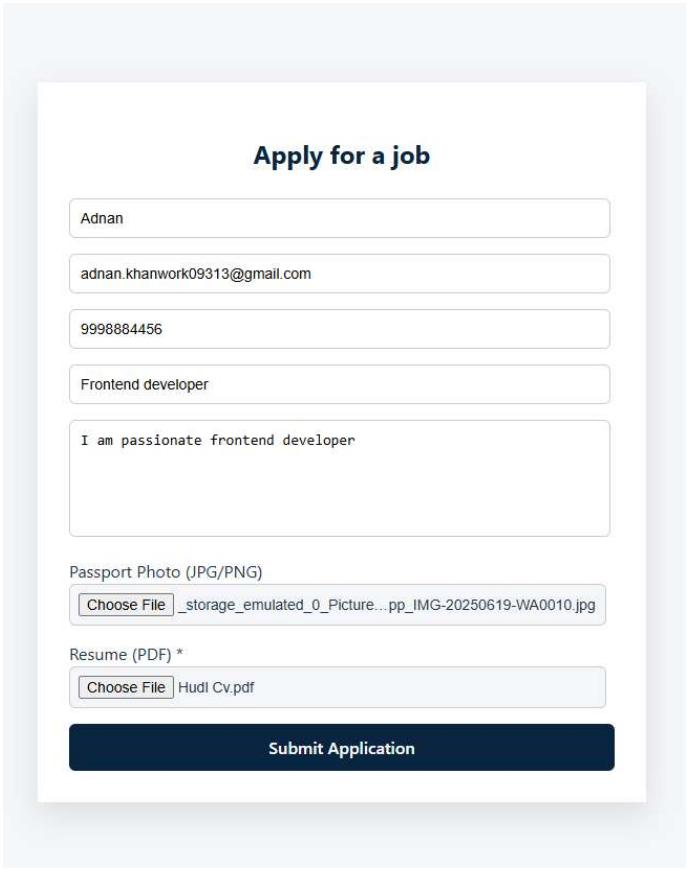
Job Portal Page



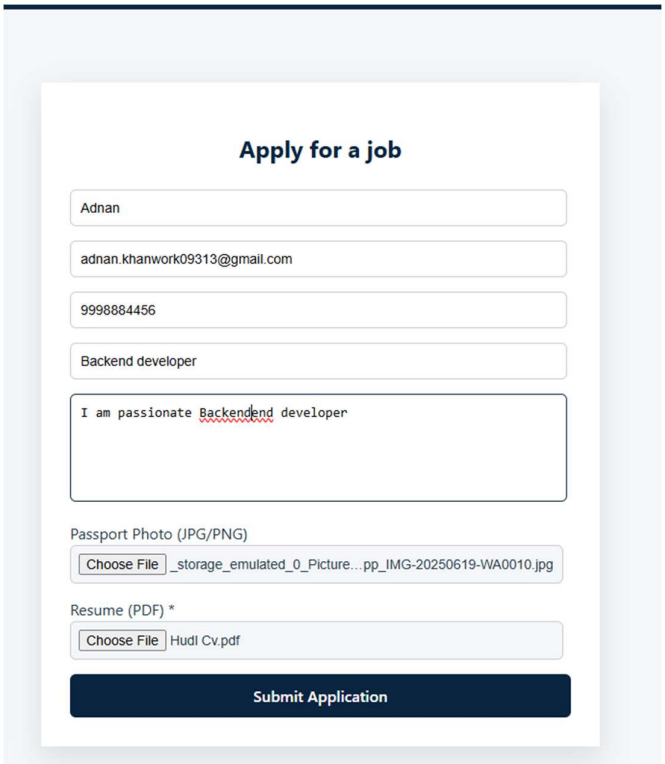
Apply for a Backend Dev



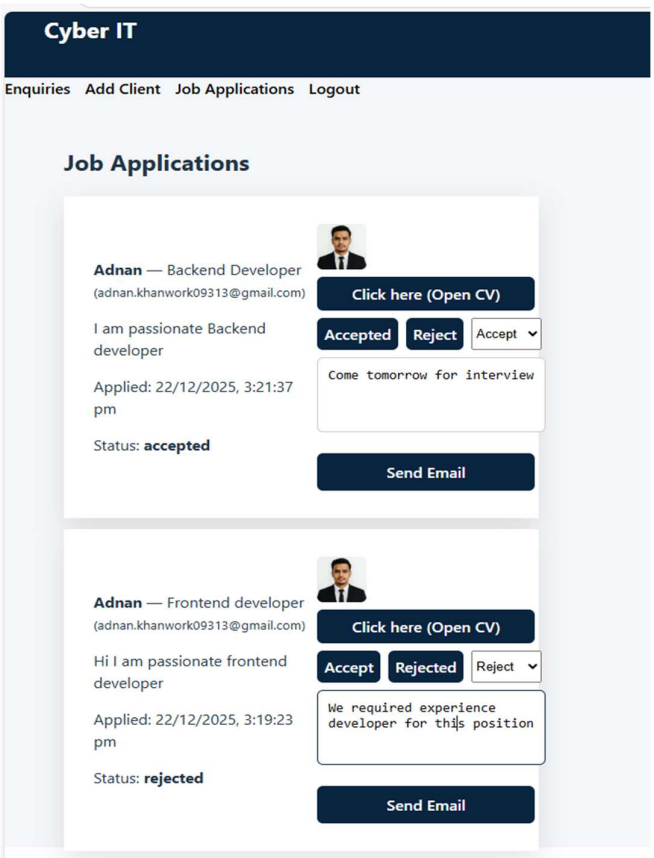
Apply for a Frontend Dev



Apply for a Backend Dev



Job application status



```
mysql> select * from job_applications;
```

id	name	email	phone	position	cover_letter	photo	resume	status	applied_at
1	Adnan	adnan.khanwork09313@gmail.com	9930742158	Frontend developer	Hi I am passionate frontend developer	1766396963862__storage_emulated_0_Pictures_WhatsApp_IMG-20250619-WA0010.jpg	1766396963863_Hudl_Cv.pdf	rejected	2025-12-22 15:19:23
2	Adnan	adnan.khanwork09313@gmail.com	9930742158	Backend Developer	I am passionate Backend developer	1766397097512__storage_emulated_0_Pictures_WhatsApp_IMG-20250619-WA0010.jpg	1766397097512_Hudl_Cv.pdf	accepted	2025-12-22 15:21:37

Email reply of Job Application

Your Job Application - Update Inbox ×

Cyber IT <mca25.khan.mohammadadnan@gnims.com>
to me

Cyber IT - Application Update

Dear Adnan,

I review your profile we need experience developer

Position: Frontend developer

Regards,
Cyber IT HR Team

One attachment • Scanned by Gmail · Add to Drive

Cyber IT
Cyber IT - Application Accepted Dear Adnan, Come tomorrow for interview Position: Backend Developer Regards, Cyber IT HR Team

Cyber IT <mca25.khan.mohammadadnan@gnims.com>
to me

Cyber IT - Application Accepted

Dear Adnan,

Come tomorrow for interview

Position: Backend Developer

Regards,
Cyber IT HR Team

One attachment • Scanned by Gmail · Add to Drive