LATE BHAUSAHEB HIRAY S.S. TRUST'S INSTITUTE OF COMPUTER APPLICATION

ISO 9001-2008
CERTIFIED
S.N. 341, Next to New English School, Govt. Colony, Bandra (East), Mumbai – 400051, Tel: 91-22-26570892/3181

A PROJECT REPORT ON

User Management System

SUBMITTED
BY

Rohit Raosaheb Phatangare

(MCA SEM-1)

ACADEMIC YEAR 2023-2025

UNDER THE GUIDANCE OF
Prof. Avantika Mahadik

LATE BHAUSAHEB HIRAY S.S. TRUST'S INSTITUTE OF COMPUTER APPLICATION
(Affiliated to University of Mumbai)
MUMBAI-MAHARASHTRA-400051

CERTIFICATE

This is to certify that the project entitled, "**User Management System**", is bonifide work of **Rohit Raosaheb Phatangare** bearing Roll. No: 2023044 submitted in partial fulfilment of the requirements for the award of Master Degree of MCA from University of Mumbai.

Internal Guide

Coordinator **External Examiner**

Date: 14/02/2024

ACKNOWLEDGEMENT

First, I would like to thank Prof. Avantika Mahadik, my project advisor, for guiding me through each and every step of the process with knowledge and support. Thank you for your advice, guidance and assistance. Without their diligent and hard work it would have not been possible for us to complete the project in record time.

I also own a debt of gratitude to all those who helped us in the development of this book. I am grateful to all the teaching and non-teaching staff, those who directly and indirectly helped us to make this report a successful one. I am also thankful to friends and family for helping us out in making this project a big success.

This project imparted a great deal of practical experience, which will be helpful for us in the near future

Rohit Raosaheb Phatangare

2023044

ABSTRACT

The User Management System is a comprehensive software solution designed to streamline the management of user accounts within organizations and applications. The system provides a centralized platform for administrators to add, view, update, delete, and search for users, enhancing operational efficiency and data security.

Key features of the User Management System include secure authentication mechanisms, role-based access control, and customizable user profiles. The system is built on a scalable architecture, allowing it to accommodate growing user bases and increasing workload demands.

By leveraging modern technologies such as Node.js and SQL, the User Management System offers a robust and user-friendly interface for both administrators and users. It empowers organizations to manage user data effectively, improve user experiences, and ensure compliance with data protection regulations.

Overall, the User Management System is a versatile and adaptable solution that caters to the diverse user management needs of organizations across various industries. Its flexibility, scalability, and security features make it a valuable asset for enhancing user management processes and driving organizational growth.

Table of Contents

CHAPTER1: INTRODUCTION

1.1 Background

The background of the User Management System project stems from the growing need for organizations to effectively manage user data and authentication processes in a digital environment. With the proliferation of online services, applications, and platforms, the management of user accounts, permissions, and access controls has become increasingly complex. Inadequate user management systems often lead to security vulnerabilities, inefficiencies, and poor user experiences.

Recognizing these challenges, the project was conceived to provide a centralized solution for managing user data, authentication, and authorization within organizations. Leveraging modern technologies such as Node.js, the project aims to streamline administrative tasks, enhance security measures, and improve user experiences. Its development is driven by the desire to address the shortcomings of existing user management systems and empower organizations to effectively manage their user base in a secure and efficient manner.

Additionally, the project's background is influenced by the broader context of technological advancements and evolving regulatory landscapes, such as the need to comply with data protection regulations like GDPR. By understanding the challenges and opportunities presented by these factors, the project seeks to deliver a robust and future-proof solution that meets the needs of organizations operating in today's digital age.

1.2 Objectives

- **Secure Authentication**: Implementing industry-standard authentication mechanisms, we ensure the integrity of user credentials and safeguard against unauthorized access.

- **Flexible User Roles and Permissions**: Administrators can define custom roles and permissions tailored to the specific needs of their application, allowing for granular control over user access to resources and functionalities.

- **User Profile Management**: Users have the ability to manage their profiles, update personal information, and adjust preferences, enhancing their overall experience and engagement.

- **Efficient User Administration**: Administrators can easily add, modify, or deactivate user accounts, as well as perform bulk operations when necessary, streamlining administrative tasks and reducing overhead.

- **Scalability and Performance**: Built on Node.js, our system is designed to be highly scalable and performant, capable of handling large volumes of user data and concurrent requests with ease.

- **Customizable and Extendable**: Our system is built with modularity and extensibility in mind, allowing for easy customization and integration with other components or third-party services as needed.

1.3 Purpose, Scope, and Applicability

1.3.1 Purpose

- **Centralized User Data Management:** Provide a centralized platform for managing user accounts, permissions, and access controls, streamlining administrative tasks and ensuring data consistency.
- **Enhanced Security Measures**: Implement robust authentication mechanisms, access controls, and encryption protocols to safeguard sensitive user data and protect against unauthorized access or breaches.
- **Improved Operational Efficiency**: Automate repetitive administrative tasks such as user registration, account updates, and access control, freeing up resources and enhancing productivity within the organization.
- **User Empowerment and Experience**: Empower users with intuitive interfaces for managing their profiles, preferences, and privacy settings, fostering a positive user experience and promoting digital trust and engagement.

1.3.2 Scope

- **Scalable Architecture**: Design the system with scalability in mind, ensuring it can handle increasing volumes of user data and concurrent requests as the organization grows.
- **Flexibility for Integration**: Build the system with modular components and API integrations, enabling seamless integration with existing applications, services, and third-party platforms.
- **Continuous Improvement and Adaptation**: Maintain a roadmap for ongoing enhancements and updates, incorporating feedback from users and stakeholders to continuously improve security, functionality, and user experience.

1.3.3 Applicability

- **Enterprise Solutions**: Suitable for businesses of all sizes, from startups to large enterprises, seeking to centralize and streamline user data management, authentication, and authorization processes.

- **Software as a Service (SaaS) Platforms**: Ideal for SaaS providers looking to enhance security, scalability, and user experience by integrating a robust user management system into their offerings.

- Online Platforms and Applications: Relevant for any online platform or application, including e-commerce websites, social media platforms, and

educational portals, requiring efficient user authentication, profile management, and access controls.

1.4 Organization of report

Till now we have seen the 1st Chapter that is the Introduction. It covered the Background, Objectives, Purpose, Scope, and Applicability, and Organization of Report. This Chapter gave us a basic idea of the project and why was this topic selected.

CHAPTER 2: SURVEY OF TECHNOLOGIES

Nodejs :-

Node.js is an open source server environment
A common task for a web server can be to open a file on the server and return the content to the client.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

Visual Studio –

I chose the Visual Studio platform because it is one of the fastest and popular IDE used by developers worldwide for creating, debugging, and deploying software applications. It offers a rich and customizable environment that supports various programming languages, frameworks, and platforms.

Features:

- Code Editor
- Debugger
- Project Management
- Integrated Tools
- Extensibility
- Collaborative Development

SQL -

SQL (Structured Query Language) is a standard programming language designed for managing and manipulating relational databases. It provides a powerful set of commands for querying, updating, and administering databases, making it a fundamental tool for data management in modern applications.

- Data Querying
- Data Manipulation
- Data Definition
- Data Control

CHAPTER 3. REQUIREMENT & ANALYSIS

3.1 Problem Definition

In today's digital landscape, the management of user data within applications and organizations poses significant challenges. As businesses expand and digital interactions proliferate, the need for a robust and efficient User Management System becomes paramount. The absence of such a system can lead to security vulnerabilities, data inconsistencies, and operational inefficiencies.

.

3.2 Requirement Specific

It requires any windows operating system and normal storage to store data.

3.3 Planning and Scheduling

Planning is the process of determining the guidelines on which actions should be taken. In the planning process we must define the time when to start and when to complete a given task.

3.4 Software and Hardware Requirements

- Hardware requirements :-

Hard disk space :-	32 GB or more
Processor :-	2 GHz Intel Pentium Dual Core or later
RAM :-	2 GB or more
Internet : -	Ethernet Cable, Wi-Fi or Data connection

- Software requirements :-

For Front End: -	HTML-CSS-JS
For Back End: -	SQL
Language: -	Nodejs.
Tool:-	Visual Studio Code.

- Requirements for user Any android,Windows,IOS version or greater with RAM 2GB.

CHAPTER 4. SYSTEM DESIGN

4.1 Basic Modules.

Add User:

- Prompt the administrator to enter the user's details (e.g., name, email, password, role).
- Validate the input data to ensure it meets any specified criteria (e.g., required fields, email format).
- Create a new user record in the database with the provided information.

View User:

- Display a list of all users currently registered in the system, along with their basic details (e.g., name, email, role).
- Allow the administrator to click on a specific user to view more detailed information about that user (e.g., profile picture, contact details).

Update User:

- Allow the administrator to select a user from the user list to update.
- Display a form pre-filled with the user's current information, allowing the administrator to make changes as needed.
- Validate the updated data to ensure it meets any specified criteria (e.g., required fields, email format).

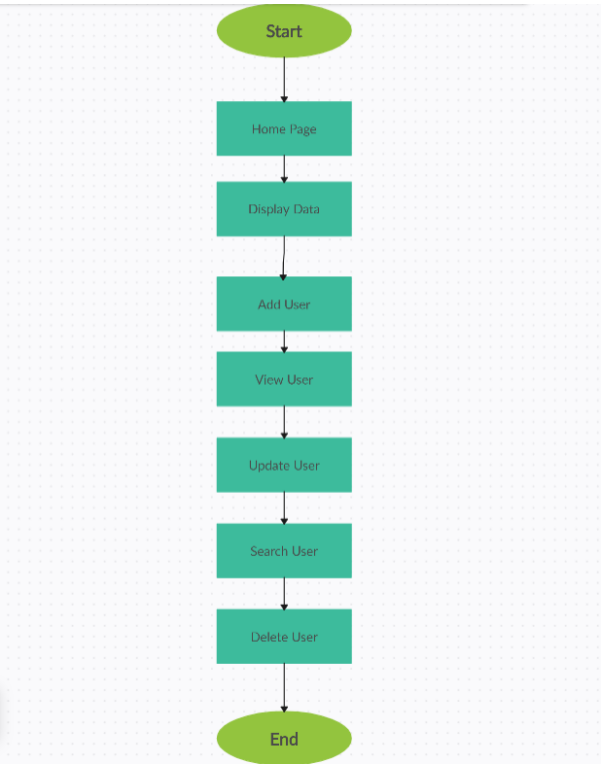
Delete User:

- Allow the administrator to select one or more users from the user list to delete.
- Display a confirmation prompt to confirm the deletion action.
- If confirmed, remove the selected user(s) from the database.
- Display a success message to the administrator confirming that the user(s) have been deleted.

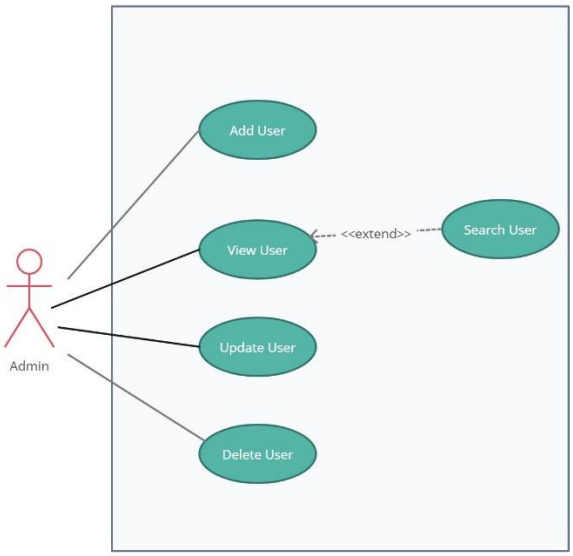
Search User:

- Provide a search bar or form where the administrator can enter search criteria (e.g., name, email) to find specific users.
- Implement a search functionality that filters the user list based on the entered criteria.
- Display the search results to the administrator, showing users that match the search criteria.
- Allow the administrator to further refine the search or reset the search criteria as needed.

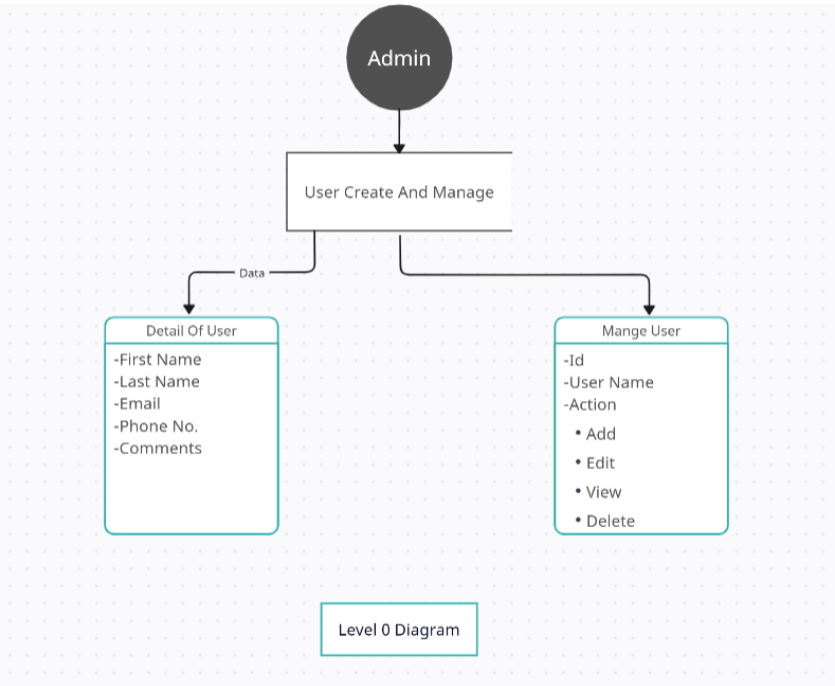
Block Diagram :-



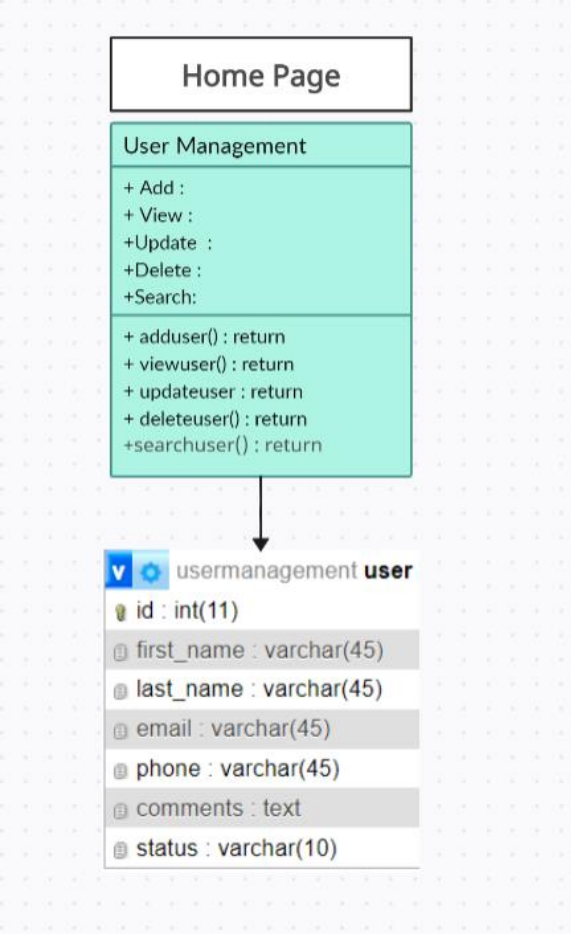
Use Case Diagram :-



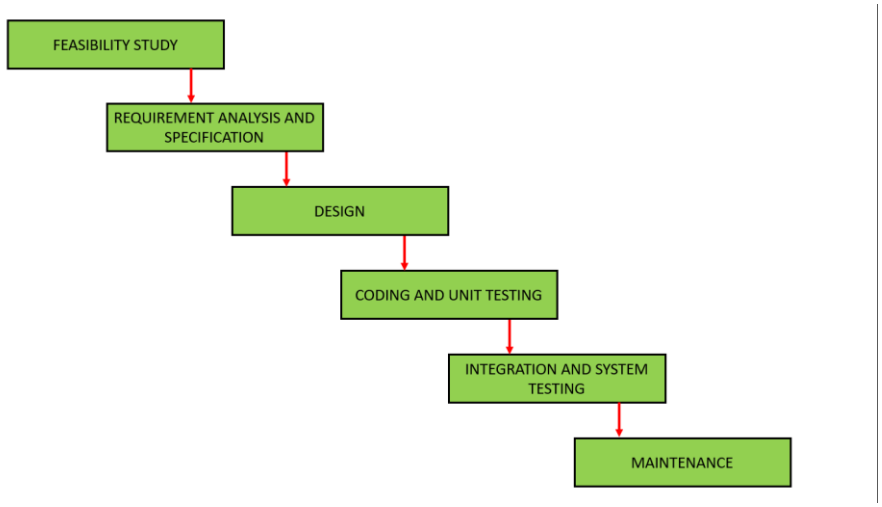
Data Flow Diagram (DFD):-



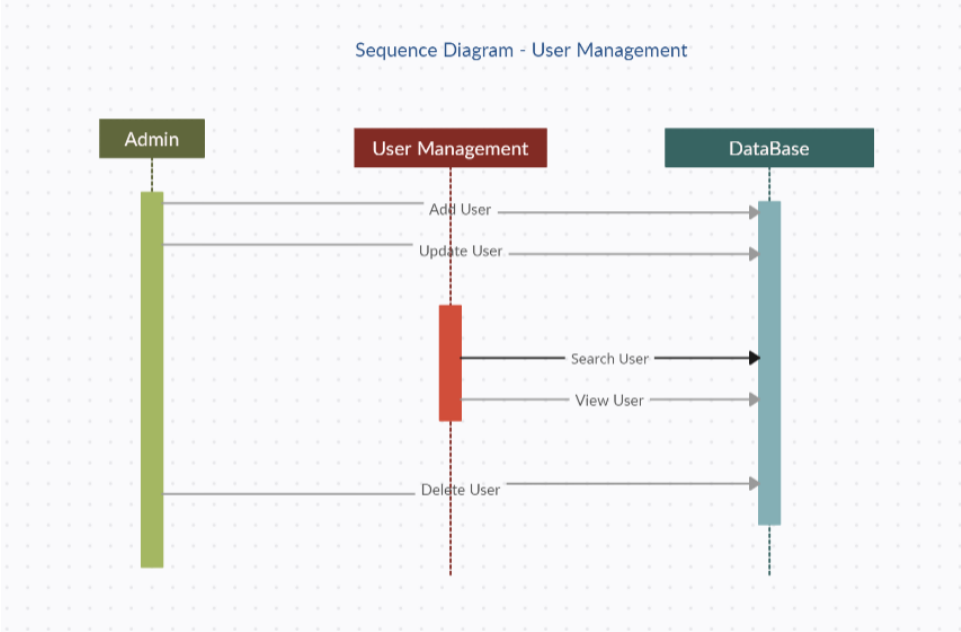
Class Diagram:-



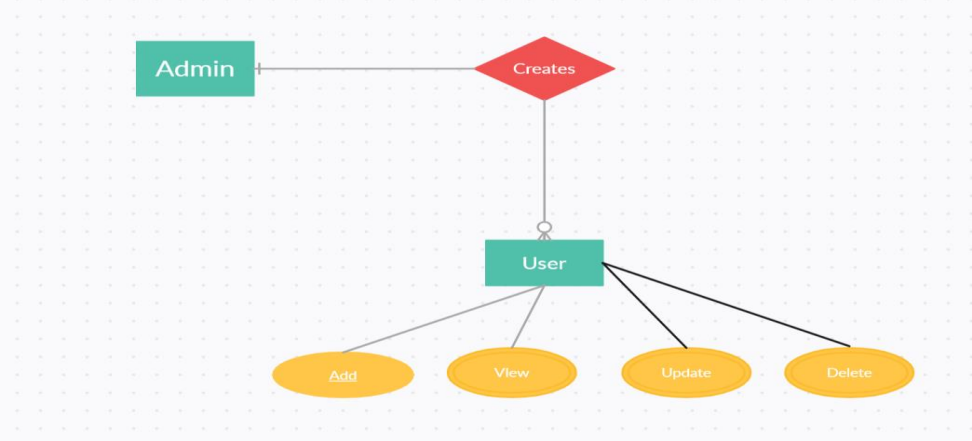
WaterFall Model Diagram :-



Sequence Diagram :-



ER Diagram :-



Gantt Chart :-

A Gantt chart is constructed with a horizontal axis representing the total time span of the project, broken down into increments and a vertical axis representing the task that make up the project. Horizontal bars of varying axis representing that make up the project. Horizontal bars of varying lengths represent the sequences, timing, and time span for each task.

The Gant chart is represented by the horizontal bar the milestone achieved are marked on the bar with arrows represented by the horizontal bar and the milestone achieved are marked on the bar with the arrows representing the completion or the start of an event. The horizontal bar is divided into the time intervals or duration period, month by month, day by day, month by years, etc.

Step In Creating A Timeline Chart

- Identify the milestones.
- Set the start date.
- Set the end date.
- Identify the events.
- Mark the dates according to start and completion of event

Chapter 5 : Implementation And Testing

5.1 Coding :-

Main.html :-

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>User Management System</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1"
crossorigin="anonymous">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.3.0/font/bootstrap-icons.css">
</head>

<body>
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
      <a class="navbar-brand" href="/">User Management System</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
        aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav me-auto mb-2 mb-lg-0">
          <li class="nav-item">
            <a class="nav-link active" aria-current="page" href="/">Home</a>
          </li>
        </ul>
        <form class="d-flex" method="POST" action="/" novalidate>
          <input class="form-control me-2" type="search" placeholder="Search" name="search" aria-label="Search">
          <button class="btn btn-outline-light" type="submit">Search</button>
        </form>

      </div>
    </div>
  </nav>

  <div class="container pt-5 pb-5">
      {{{body}}}
  </div>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-ygbV9kiqUc6oa4msXn9868pTtWMgiQaeYH7/t7LECLbyPA2x65Kgf80OJFdroafW"
    crossorigin="anonymous"></script>
</body>
</html>
```

Home.html:

```
{{#if removedUser}}
<div class="alert alert-success alert-dismissible fade show" role="alert">
  User has been removed.
  <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
</div>
{{/if}}

<div class="row">
  <div class="col-6">
    <h1>Users</h1>
  </div>
  <div class="col-6 d-flex justify-content-end">
    <a href="/adduser" type="button" class="btn btn-primary align-self-center">+ add user</a>
  </div>
</div>

<table class="table table-bordered">
  <thead class="thead-dark">
    <tr>
      <th scope="col">#</th>
      <th scope="col">First Name</th>
      <th scope="col">Last Name</th>
      <th scope="col">Email</th>
      <th scope="col">Phone</th>
      <th scope="col" class="text-end">Action</th>
    </tr>
  </thead>
  <tbody>

    {{#each rows}}
    <tr>
      <th scope="row">{{this.id}}</th>
      <td>{{this.first_name}}</td>
      <td>{{this.last_name}}</td>
      <td>{{this.email}}</td>
      <td>{{this.phone}}</td>
      <td class="text-end">
        <a href="/viewuser/{{this.id}}" type="button" class="btn btn-light btn-small"><i class="bi bi-eye"></i> View</a>
        <a href="/edituser/{{this.id}}" type="button" class="btn btn-light btn-small"><i class="bi bi-pencil"></i>
          Edit</a>
        <a href="/{{this.id}}" type="button" class="btn btn-light btn-small"><i class="bi bi-person-x"></i> Delete</a>
      </td>
    </tr>
    {{/each}}
  </tbody>
</table>
```

Add.html:-

```
<nav aria-label="breadcrumb">
 <ol class="breadcrumb">
  <li class="breadcrumb-item"><a href="/">Home</a></li>
  <li class="breadcrumb-item active" aria-current="page">New User</li>
 </ol>
</nav>

{{#if alert}}
<div class="alert alert-success alert-dismissible fade show" role="alert">
 {{alert}}
 <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
</div>
{{/if}}

<form class="row g-3 needs-validation" method="POST" action="/adduser" novalidate>
 {{> user-forms}}
</form>
```

Edit.html:-

```
<nav aria-label="breadcrumb">
  <ol class="breadcrumb">
    <li class="breadcrumb-item"><a href="/">Home</a></li>
    <li class="breadcrumb-item active" aria-current="page">Edit User</li>
  </ol>
</nav>

{{#if alert}}
  <div class="alert alert-success alert-dismissible fade show" role="alert">
    {{alert}}
    <button type="button" class="btn-close" data-bs-dismiss="alert" aria-
label="Close"></button>
  </div>
{{/if}}

{{#each rows}}
<form class="row g-3 needs-validation" method="POST" action="/edituser/{{this.id}}"
novalidate>
  {{> user-forms}}
</form>
{{/each}}
```

View.html:-

```
<nav aria-label="breadcrumb">
  <ol class="breadcrumb">
    <li class="breadcrumb-item"><a href="/">Home</a></li>
    <li class="breadcrumb-item active" aria-current="page">View User</li>
  </ol>
</nav>

<div class="view-user p-5">
  {{#each rows}}
  <div class="row mb-5">
    <div class="col text-center">
      <h3>{{this.first_name}} {{this.last_name}}</h3>
    </div>
  </div>
  <div class="row">
    <div class="col">
      <table class="table">
        <thead>
          <tr>
            <th scope="col">First Name</th>
            <th scope="col">Last Name</th>
            <th scope="col">Email</th>
            <th scope="col">Phone</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <th scope="row">{{this.first_name}}</th>
            <td>{{this.last_name}}</td>
            <td>{{this.email}}</td>
            <td>{{this.phone}}</td>
          </tr>
        </tbody>
      </table>
    </div>
    <div class="row">
      <div class="col">
        <b>Comments</b>
      </div>
    </div>
    <div class="row">
      <div class="col">
        {{this.comments}}
      </div>
    </div>
  </div>
  {{/each}}
</div>
```

UserController.js:-

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

// Connection Pool
let connection = mysql.createConnection({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASS,
  database: process.env.DB_NAME,
});

// View Users
exports.view = (req, res) => {

    // pool.getConnection((err,connection)=>{
    //     if (err) throw err; //NOT Connected
    //     console.log("Connected as ID " + connection.threadId);
    // });

  // User the connection
  connection.query(
    'SELECT * FROM user WHERE status = "active"',
    (err, rows) => {
      // When done with the connection, release it
      if (!err) {
        let removedUser = req.query.removed;
        res.render("home", { rows, removedUser });
      } else {
        console.log(err);
      }
      console.log("The data from user table: \n", rows);
    }
  );
};

// Find User by Search
exports.find = (req, res) => {

    // pool.getConnection((err,connection)=>{
    //     if (err) throw err; //NOT Connected
    //     console.log("Connected as ID " + connection.threadId);
    // });
```

```
  let searchTerm = req.body.search;
  // User the connection
  connection.query(
    "SELECT * FROM user WHERE first_name LIKE ? OR last_name LIKE ?",
    ["%" + searchTerm + "%", "%" + searchTerm + "%"],
    (err, rows) => {
      if (!err) {
        res.render("home", { rows });
      } else {
        console.log(err);
      }
      console.log("The data from user table: \n", rows);
    }
  );
};

exports.form = (req, res) => {
  res.render("add-user");
};

// Add new user
exports.create = (req, res) => {
  const { first_name, last_name, email, phone, comments } = req.body;
  let searchTerm = req.body.search;

  // User the connection
  connection.query(
    "INSERT INTO user SET first_name = ?, last_name = ?, email = ?, phone = ?, comments = ?",
    [first_name, last_name, email, phone, comments],
    (err, rows) => {
      if (!err) {
        res.render("add-user", { alert: "User added successfully." });
      } else {
        console.log(err);
      }
      console.log("The data from user table: \n", rows);
    }
  );
};

// Edit user
exports.edit = (req, res) => {
  // User the connection
  connection.query(
```

```
    "SELECT * FROM user WHERE id = ?",
    [req.params.id],
    (err, rows) => {
      if (!err) {
        res.render("edit-user", { rows });
      } else {
        console.log(err);
      }
      console.log("The data from user table: \n", rows);
    }
  );
};

// Update User
exports.update = (req, res) => {
  const { first_name, last_name, email, phone, comments } = req.body;
  // User the connection
  connection.query(
    "UPDATE user SET first_name = ?, last_name = ?, email = ?, phone = ?, comments = ? WHERE id = ?",
    [first_name, last_name, email, phone, comments, req.params.id],
    (err, rows) => {
      if (!err) {
        // User the connection
        connection.query(
          "SELECT * FROM user WHERE id = ?",
          [req.params.id],
          (err, rows) => {
            // When done with the connection, release it

            if (!err) {
              res.render("edit-user", {
                rows,
                alert: `${first_name} has been updated.`,
              });
            } else {
              console.log(err);
            }
            console.log("The data from user table: \n", rows);
          }
        );
      } else {
        console.log(err);
      }
      console.log("The data from user table: \n", rows);
```

```
    }
  );
};

// Delete User
exports.delete = (req, res) => {
  // Delete a record

  // User the connection
  // connection.query('DELETE FROM user WHERE id = ?', [req.params.id], (err, rows) => {

  //   if(!err) {
  //     res.redirect('/');
  //   } else {
  //     console.log(err);
  //   }
  //   console.log('The data from user table: \n', rows);

  // });

  // Hide a record

  connection.query(
    "UPDATE user SET status = ? WHERE id = ?",
    ["removed", req.params.id],
    (err, rows) => {
      if (!err) {
        let removedUser = encodeURIComponent("User successeflly removed.");
        res.redirect("/?removed=" + removedUser);
      } else {
        console.log(err);
      }
      console.log("The data from beer table are: \n", rows);
    }
  );
};

// View Users
exports.viewall = (req, res) => {
  // User the connection
  connection.query(
    "SELECT * FROM user WHERE id = ?",
    [req.params.id],
    (err, rows) => {
```

```
    if (!err) {
      res.render("view-user", { rows });
    } else {
      console.log(err);
    }
    console.log("The data from user table: \n", rows);
  }
 );
};
```

User.js:-

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

// Routes : create / update / find / delete
router.get('/', userController.view);
router.post('/', userController.find);
router.get('/adduser', userController.form);
router.post('/adduser', userController.create);
router.get('/edituser/:id', userController.edit);
router.post('/edituser/:id', userController.update);
router.get('/viewuser/:id', userController.viewall);
router.get('/:id',userController.delete);

module.exports = router;
```

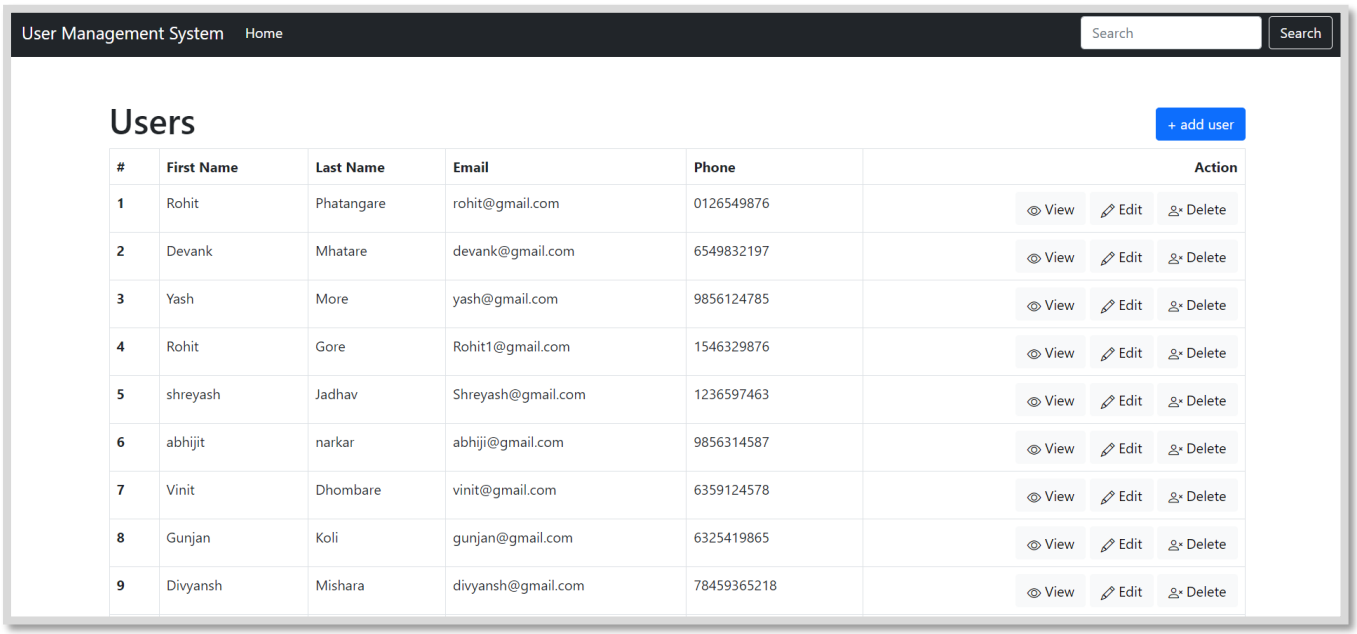
App.js:-

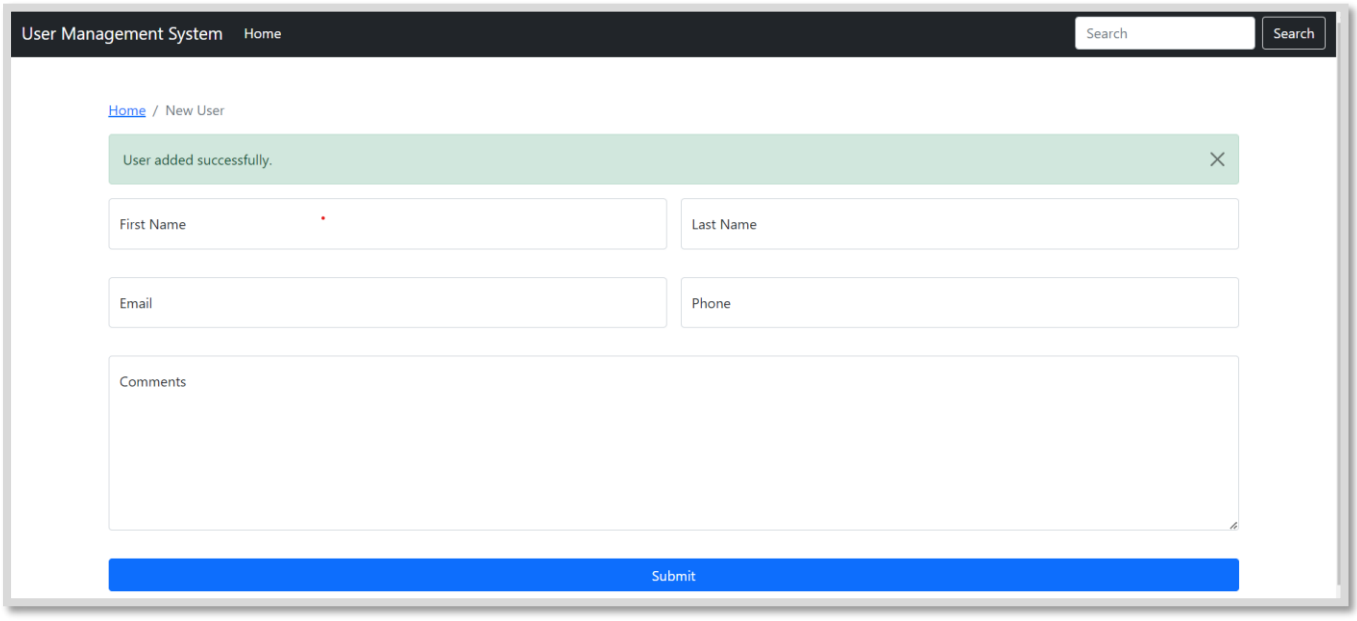
```
const express = require('express');
const exphbs = require('express-handlebars');
const bodyParser = require('body-parser'); // No longer Required
const mysql = require('mysql'); // Not required -> moved to userController

require('dotenv').config();

const app = express();
const port = process.env.PORT || 5000;

// Parsing middleware
// Parse application/x-www-form-urlencoded
```

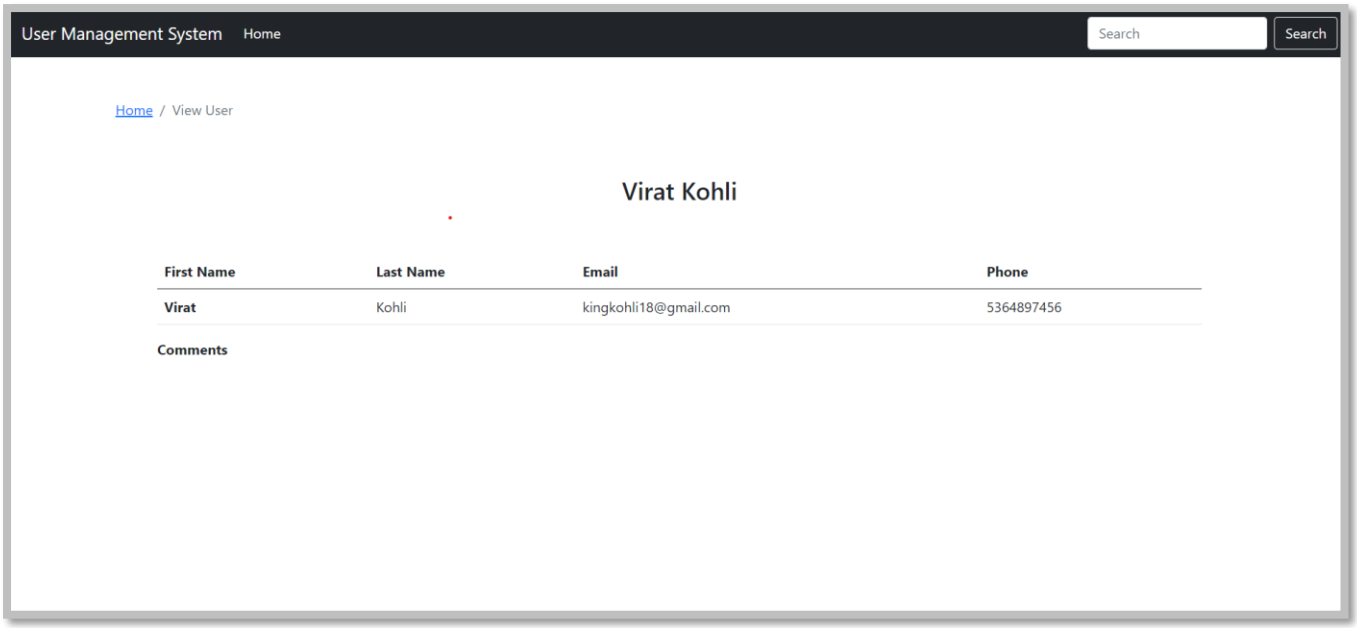
```
app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.urlencoded({ extended: true })); // New

// Parse application/json
app.use(bodyParser.json());
app.use(express.json()); // New

// Static Files
app.use(express.static('public'));

// Templating Engine
const handlebars = exphbs.create({ extname: '.hbs', });
app.engine('.hbs', handlebars.engine);
app.set('view engine', '.hbs');

// You don't need the connection here as we have it in userController
// let connection = mysql.createConnection({
//   host: process.env.DB_HOST,
//   user: process.env.DB_USER,
//   password: process.env.DB_PASS,
//   database: process.env.DB_NAME
// });

//Connection pool:
const pool=mysql.createPool({
   connectionLimit :  100,
   host          :  process.env.DB_HOST,
   user          :  process.env.DB_USER,
   password      :  process.env.DB_PASS,
   database      :  process.env.DB_NAME
});

//Connect to DB
pool.getConnection((err,connection)=>{
   if (err) throw err; //NOT Connected~
   console.log("Connected as ID " + connection.threadId);
});

// Router
const routes = require('./server/routes/user');
app.use('/', routes);

//use before only for check ~
// app.get('',(req,res)=>{
//     res.render('home');
// });

app.listen(port, () => console.log(`Listening on port ${port}`));
```

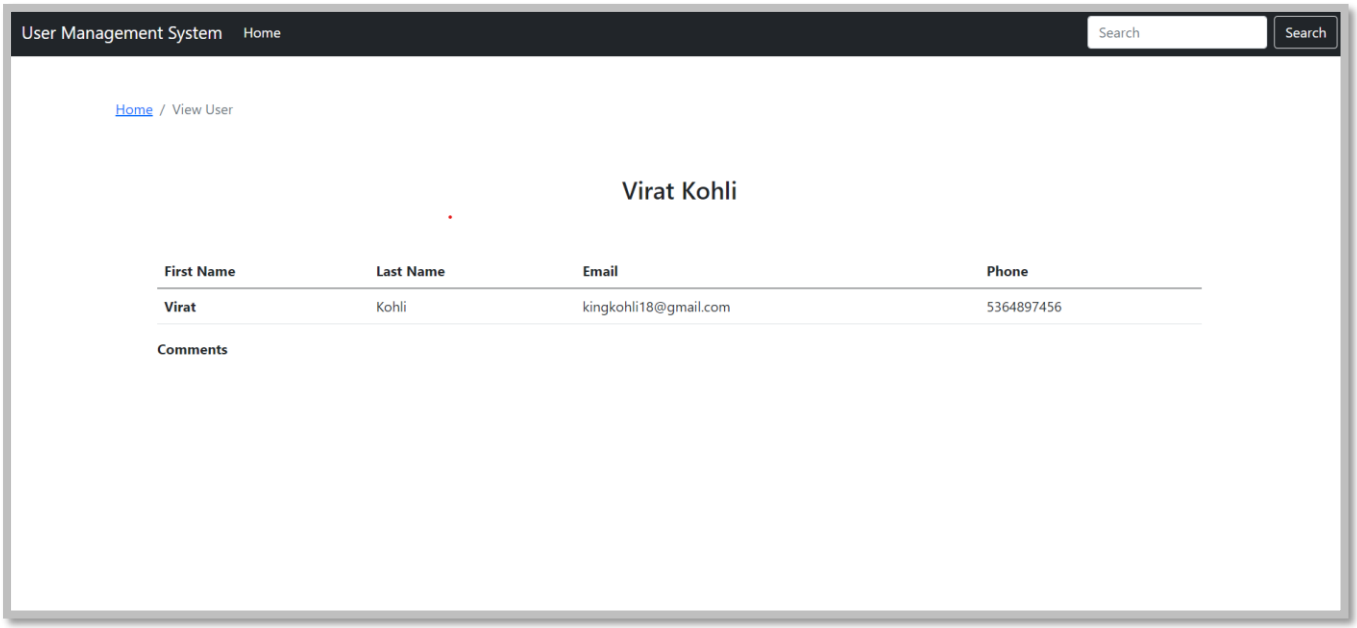
RESULT :- Successful.

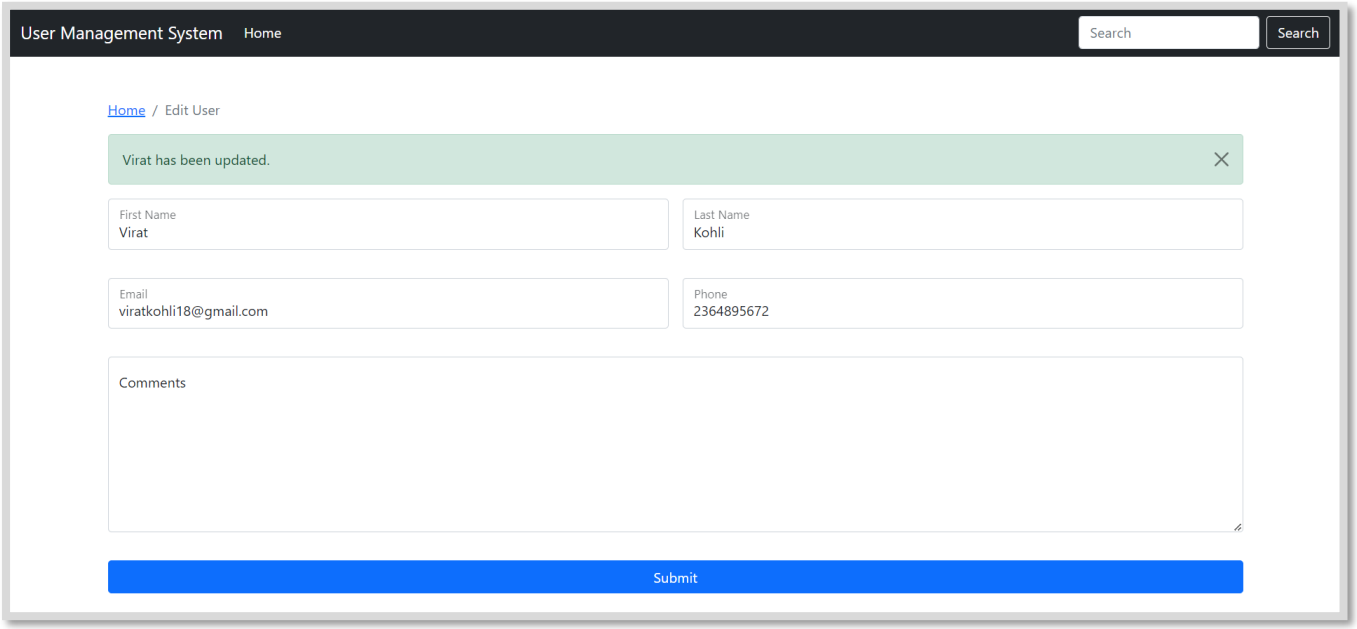
Screen Shots :-

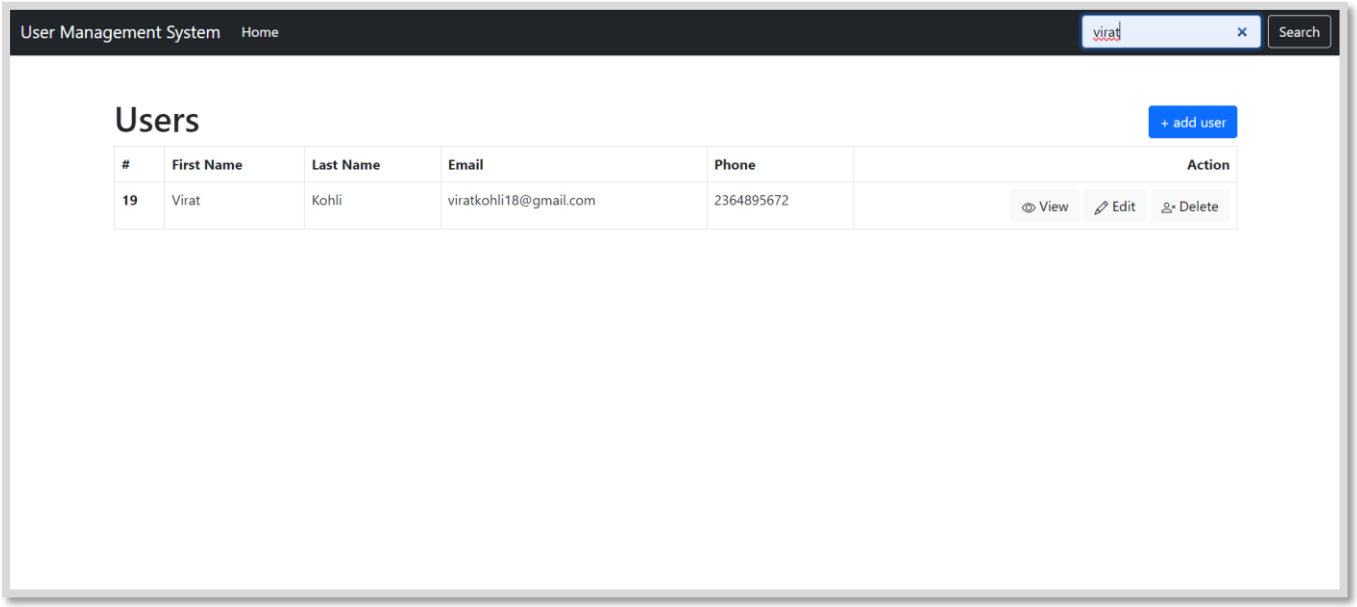


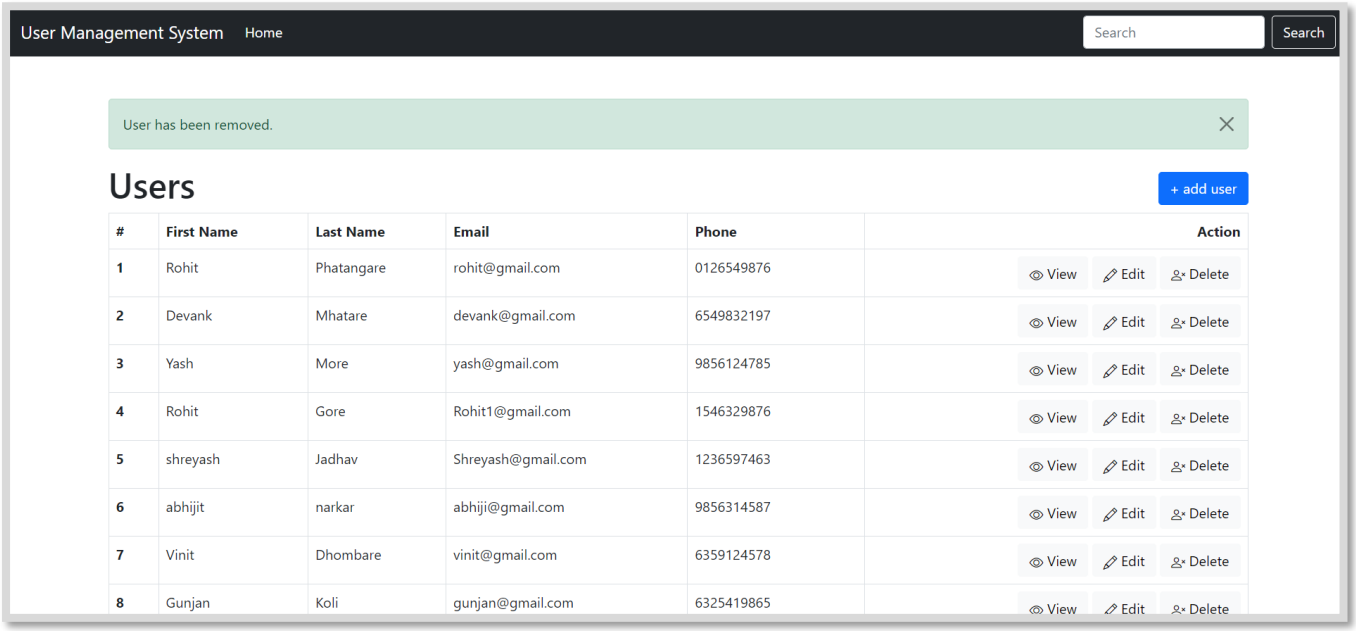



User Management System
Home
Search
Search
Home / View User
Virat Kohli
First Name
Last Name
Email
Phone
Virat
Kohli
kingkohli18@gmail.com
5364897456
Comments


User Management System
Home
Search
Search
Home / View User
Virat Kohli
First Name
Last Name
Email
Phone
Virat
Kohli
kingkohli18@gmail.com
5364897456
Comments


User Management System
Home
Search
Search
Home / Edit User
Virat has been updated.
First Name
Virat
Last Name
Kohli
Email
viratkohli18@gmail.com
Phone
2364895672
Comments
Submit


User Management System
Home
virat
Search
Users
+ add user
# First Name Last Name Email Phone Action
19 Virat Kohli viratkohli18@gmail.com 2364895672
View
Edit
Delete


User Management System
Home
Search
Search
User has been removed.
Users
+ add user
# First Name Last Name Email Phone Action
1 Rohit Phatangare rohit@gmail.com 0126549876 View Edit Delete
2 Devank Mhatare devank@gmail.com 6549832197 View Edit Delete
3 Yash More yash@gmail.com 9856124785 View Edit Delete
4 Rohit Gore Rohit1@gmail.com 1546329876 View Edit Delete
5 shreyash Jadhav Shreyash@gmail.com 1236597463 View Edit Delete
6 abhijit narkar abhiji@gmail.com 9856314587 View Edit Delete
7 Vinit Dhombare vinit@gmail.com 6359124578 View Edit Delete
8 Gunjan Koli gunjan@gmail.com 6325419865 View Edit Delete

5.2 Testing

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, **testing** is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

Software Testing is necessary because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. Software testing is really required to point out the defects and errors that were made during the development phases.

Software testing is an eminent tool and has a significant role in today's business. Some of the foremost **advantages** are

- Highly efficient
- Quality
- Satisfies customer
- Good product, good revenue
- User experience
- Business optimization

Some of the **disadvantages** are:

- Appropriate communication and coordination with the tester
- Competition among similar service providers
- Lack of experienced professionals
- Finding the right service provider

Types of Testing

Unit Testing

In unit testing, each component or individual units of the software shall be tested. The aim of the unit testing is to check internal data structures, logic, boundary conditions for input and output data as per the design.

Integration Testing

Integration Testing is defined as a type of software testing carried out in an integrated hardware and software environment to verify the behaviour of the complete system. For Example, software and/or hardware components are combined and tested progressively until the entire system has been **integrated.**

Regression Testing

Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

Smoke Testing

Smoke Testing is a software testing process that determines whether the deployed software build is stable or not. Smoke testing is a confirmation for QA team to proceed with further software testing.

Sanity Testing

Sanity testing is a kind of Software Testing performed after receiving a software build, with minor changes in code, or functionality, to ascertain that the bugs have been fixed and no further issues are introduced due to these changes. The goal is to determine that the proposed functionality works roughly as expected.

The App which we are implementing mainly falls on 2 types, Smoke testing and Integration testing

Smoke Testing: In this we are verifying whether the important features are working and there are no showstoppers in the build that is under testing.

- Easy to perform testing
- Defects will be identified in early stages.
- Improves the quality of the system

Integration Testing: In integration testing, individual units are integrated and tested to understand if the integrated components work efficiently.

1. Integration testing for different modules at the same time is easy.
2. Can be used in the early as well as later stages of the testing process.
3. Code length coverage is more as compared with other software testing techniques as both, the bottom-up and top-down approaches can be used.

TEST CASE: -

Test Case id	Test Scenario	Test Steps	Expected Result	Actual Result	Case Status
TC_001	Admin can Add user data(First Name,Last Name, Email,Phone Number.)	App -- Home -- Add user --	Add User success	As same as expected	Pass
TC_002	Admin can Edit user data(First Name,Last Name, Email,Phone Number.)	App -- Home -- Edit user --	Edit User Successfuly	As same as expected	Pass
TC_003	Admin can Update user data(First Name,Last Name, Email,Phone Number.)	App -- Update -- Add user --	Update User Successfuly	As same as expected	pass
TC_004	Admin can Search user data(First Name,Last Name, Email,Phone Number.)	App -- Home Page	Search User Successfuly	As same as expected	Pass
TC_005	Admin can Delete user data(First Name,Last Name, Email,Phone Number.)	App -- Home -- Delete user --	Delete User Successfuly	As same as expected	Pass

CHAPTER 6: CONCLUSION

Simplicity is never simple. As we have seen in this project, the process of creating a user-friendly and straightforward platform that facilitates the administrator's job is one filled with complexity. From understanding user requirements to system design and finally system prototype and finalization, every step requires in-depth understanding and commitment towards achieving the objectives of the project.

Although the user Management system web App module is not fully integrated to the system and used on real time, the system prototype demonstrates easy navigation and data are stored in a systematic way. Overall, efficiency has improved and work processes simplified. Although all the objectives have been met, the system still has room for improvement. The system is robust and flexible enough for future upgrade using advanced technology and devices.

CHAPTER 7: Reference

- https://nodejs.org/en/learn/getting-started/introduction-to-nodejs
- https://www.geeksforgeeks.org/nodejs/
- https://www.w3schools.com/nodejs/nodejs_mysql.asp
- Book: Node.js Design Patterns by Mario Casciaro, 2014 (Author)