

PROGRAM:

Exercise 6.html

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Customer CRUD App</title>

  <link rel="stylesheet" href="ex_6_style.css">

</head>

<body>

  <h2>Customer Management System</h2>

  <form id="customerForm">

    <input type="text" id="name" placeholder="Enter Name" required>

    <input type="text" id="city" placeholder="Enter City" required>

    <input type="text" id="mobile" placeholder="Enter Mobile No" required>

    <button type="submit">Add Customer</button>

  </form>

  <table>

    <thead>

      <tr>

        <th>Name</th>

        <th>City</th>

        <th>Mobile No</th>

        <th>Actions</th>

      </tr>

    </thead>

    <tbody id="customersList"></tbody>

  </table>

  <script>

    const API_URL = 'http://localhost:5000/customers';

    async function fetchCustomers() {

      const res = await fetch(API_URL);
```

```

const customers = await res.json();

document.getElementById('customersList').innerHTML = customers.map(customer
=> `
    <tr>
        <td><input type="text" value="${customer.name}" id="name-
${customer.id}"></td>
        <td><input type="text" value="${customer.city}" id="city-
${customer.id}"></td>
        <td><input type="text" value="${customer.mobile}" id="mobile-
${customer.id}"></td>
        <td>
            <button onclick="updateCustomer('${customer.id}')">Update</button>
            <button onclick="deleteCustomer('${customer.id}')">Delete</button>
        </td>
    </tr>
`
).join("");
}

async function addCustomer(event) {
    event.preventDefault();

    const name = document.getElementById('name').value;
    const city = document.getElementById('city').value;
    const mobile = document.getElementById('mobile').value;

    await fetch(API_URL, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ name, city, mobile })
    });

    document.getElementById('customerForm').reset();

    fetchCustomers();
}

async function updateCustomer(id) {
    const name = document.getElementById(`name-${id}`).value;
    const city = document.getElementById(`city-${id}`).value;
    const mobile = document.getElementById(`mobile-${id}`).value;

    await fetch(`${API_URL}/${id}`, {

```

```

        method: 'PUT',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ name, city, mobile })
    });
    fetchCustomers();
}

async function deleteCustomer(id) {
    await fetch(`${API_URL}/${id}`, { method: 'DELETE' });
    fetchCustomers();
}

document.getElementById('customerForm').addEventListener('submit', addCustomer);
fetchCustomers();
</script>
</body>
</html>

```

Exercise 6 style.css

```

body {
    font-family: Arial, sans-serif;
    text-align: center;
    background-color: #f4f4f4;
}

form {
    margin: 20px auto;
    width: 50%;
    padding: 15px;
    background: white;
    border-radius: 8px;
}

input {
    padding: 10px;
    margin: 5px;
    width: 80%;
    border: 1px solid #ccc;
}

```

```

}
table {
  width: 60%;
  margin: 20px auto;
  border-collapse: collapse;
}
th, td {
  padding: 10px;
  border: 1px solid #ddd;
  text-align: center;
}
button {
  padding: 8px 12px;
  border: none;
  color: white;
  border-radius: 5px;
}
button:nth-child(1) { background-color: #28a745; }
button:nth-child(2) { background-color: #dc3545; }

```

Ex_6_server.js

```

require('dotenv').config();
const express = require('express');
const mysql = require('mysql2');
const cors = require('cors');
const app = express();
app.use(express.json());
app.use(cors());
const db = mysql.createConnection({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASS,
  database: process.env.DB_NAME
});

```

```

db.connect(err => {
  if (err) {
    console.error("✖ MySQL Connection Error:", err.message);
    process.exit(1);
  }
  console.log('✔ MySQL Connected');
});

app.post('/customers', (req, res) => {
  const { name, city, mobile } = req.body;
  const sql = 'INSERT INTO customers (name, city, mobile) VALUES (?, ?, ?)';
  db.query(sql, [name, city, mobile], (err, result) => {
    if (err) return res.status(500).json({ error: err.message });
    res.json({ id: result.insertId, name, city, mobile });
  });
});

app.get('/customers', (req, res) => {
  const sql = 'SELECT * FROM customers';
  db.query(sql, (err, results) => {
    if (err) return res.status(500).json({ error: err.message });
    res.json(results);
  });
});

app.put('/customers/:id', (req, res) => {
  const { name, city, mobile } = req.body;
  const sql = 'UPDATE customers SET name = ?, city = ?, mobile = ? WHERE id = ?';
  db.query(sql, [name, city, mobile, req.params.id], (err) => {
    if (err) return res.status(500).json({ error: err.message });
    res.json({ message: 'Customer updated successfully' });
  });
});

app.delete('/customers/:id', (req, res) => {
  const sql = 'DELETE FROM customers WHERE id = ?';
  db.query(sql, [req.params.id], (err) => {

```

```
    if (err) return res.status(500).json({ error: err.message });  
    res.json({ message: 'Customer deleted successfully' });  
  });  
});  
app.listen(5000, () => console.log('🚀 Server running on port 5000'));
```

Output:

Customer Management System

Enter Name

Enter City

Enter Mobile No

Add Customer

Name	City	Mobile No	Actions
<input type="text" value="sunil"/>	<input type="text" value="chennai"/>	<input type="text" value="9791989723"/>	<button>Update</button> <button>Delete</button>
<input type="text" value="sibi"/>	<input type="text" value="Mahendra city"/>	<input type="text" value="8376836387"/>	<button>Update</button> <button>Delete</button>
<input type="text" value="ragul"/>	<input type="text" value="potheri"/>	<input type="text" value="8765636783"/>	<button>Update</button> <button>Delete</button>
<input type="text" value="raghav"/>	<input type="text" value="Mahendra city"/>	<input type="text" value="8376834641"/>	<button>Update</button> <button>Delete</button>
<input type="text" value="sp"/>	<input type="text" value="perungalthur"/>	<input type="text" value="9672738373"/>	<button>Update</button> <button>Delete</button>

PROGRAM:

Exercise 7.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" />

    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

    <title>React App</title>

  </head>

  <body>

    <noscript>You need to enable JavaScript to run this app.</noscript>

    <div id="root"></div>

  </body>

</html>
```

App.css

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

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
```



```

    }
  }
.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}
.App-link {
  color: #61dafb;
}
@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

```

App.js

```

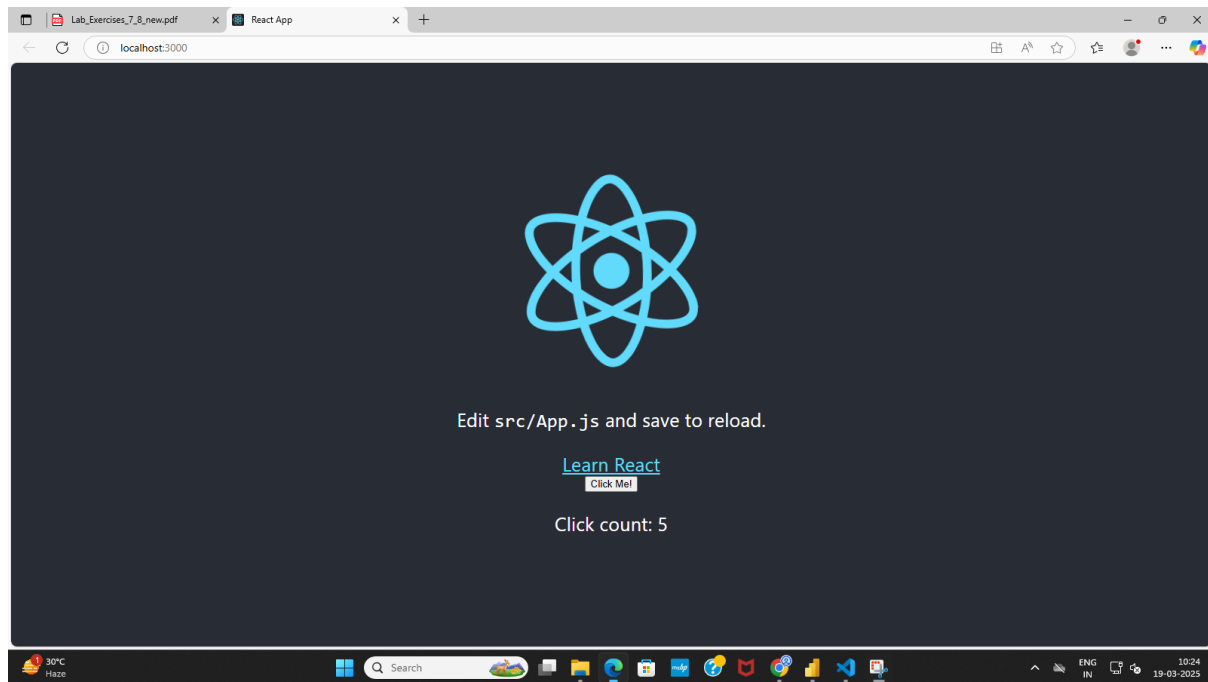
import React, { useState } from 'react'; // Import useState from React
import logo from './logo.svg';
import './App.css';
function App() {
  const [count, setCount] = useState(0);
  const handleClick = () => {
    setCount(count + 1); // Increase the count by 1 each time the button is clicked
  };

```

```
return (  
  <div className="App">  
    <header className="App-header">  
      <img src={logo} className="App-logo" alt="logo" />  
      <p>  
        Edit <code>src/App.js</code> and save to reload.  
      </p>  
      <a  
        className="App-link"  
        href="https://reactjs.org"  
        target="_blank"  
        rel="noopener noreferrer"  
      >  
        Learn React  
      </a>  
      { /* Button to trigger count update */}  
      <button onClick={ handleClick }>Click Me!</button>  
      { /* Display the click count */}  
      <p>Click count: {count}</p>  
    </header>  
  </div>  
);  
}
```

export default App;

Output:



PROGRAM:

Exercise 8:

App.js

```
import React, { useState } from 'react';
import './App.css';
function App() {
  const [todos, setTodos] = useState([]);
  const [input, setInput] = useState("");
  const [editIndex, setEditIndex] = useState(null);
  const [editText, setEditText] = useState("");
  const handleInputChange = (e) => {
    setInput(e.target.value);
  };
  const addTodo = () => {
    if (input.trim() !== "") {
      setTodos([...todos, { text: input, completed: false }]);
      setInput("");
    }
  };
  const toggleComplete = (index) => {
    const updatedTodos = todos.map((todo, i) =>
      i === index ? { ...todo, completed: !todo.completed } : todo
    );
    setTodos(updatedTodos);
  };
  const deleteTodo = (index) => {
    const updatedTodos = todos.filter((_, i) => i !== index);
    setTodos(updatedTodos);
  };
  const editTodo = (index) => {
    setEditIndex(index);
    setEditText(todos[index].text);
  };
}
```

```

const saveEdit = () => {
  if (editText.trim() !== "") {
    const updatedTodos = todos.map((todo, i) =>
      i === editIndex ? { ...todo, text: editText } : todo
    );
    setTodos(updatedTodos);
    setEditIndex(null);
    setEditText("");
  }
};

return (
  <div className="App">
    <h1>To-Do App</h1>
    <div className="todo-input">
      <input
        type="text"
        value={input}
        onChange={handleInputChange}
        placeholder="Enter a new task"
      />
      <button className="add-btn" onClick={addTodo}>Add</button>
    </div>
    { /* Edit todo */ }
    {editIndex !== null && (
      <div className="edit-todo">
        <input
          type="text"
          value={editText}
          onChange={(e) => setEditText(e.target.value)}
          placeholder="Edit your task"
        />
        <button className="save-btn" onClick={saveEdit}>Save</button>
      </div>
    )}
  </div>
)

```

```

    })
    <table className="todo-table">
      <thead>
        <tr>
          <th>Task</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        {todos.map((todo, index) => (
          <tr key={index}>
            <td className={todo.completed ? 'completed' : ''}>
              <span onClick={() => toggleComplete(index)}>{todo.text}</span>
            </td>
            <td>
              <button className="edit-btn" onClick={() => editTodo(index)}>Edit</button>
              <button className="delete-btn" onClick={() =>
deleteTodo(index)}>Delete</button>
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  </div>
);
}

export default App;

```

App.css

```

.App {
  text-align: center;
  margin-top: 20px;
  font-family: Arial, sans-serif;

```

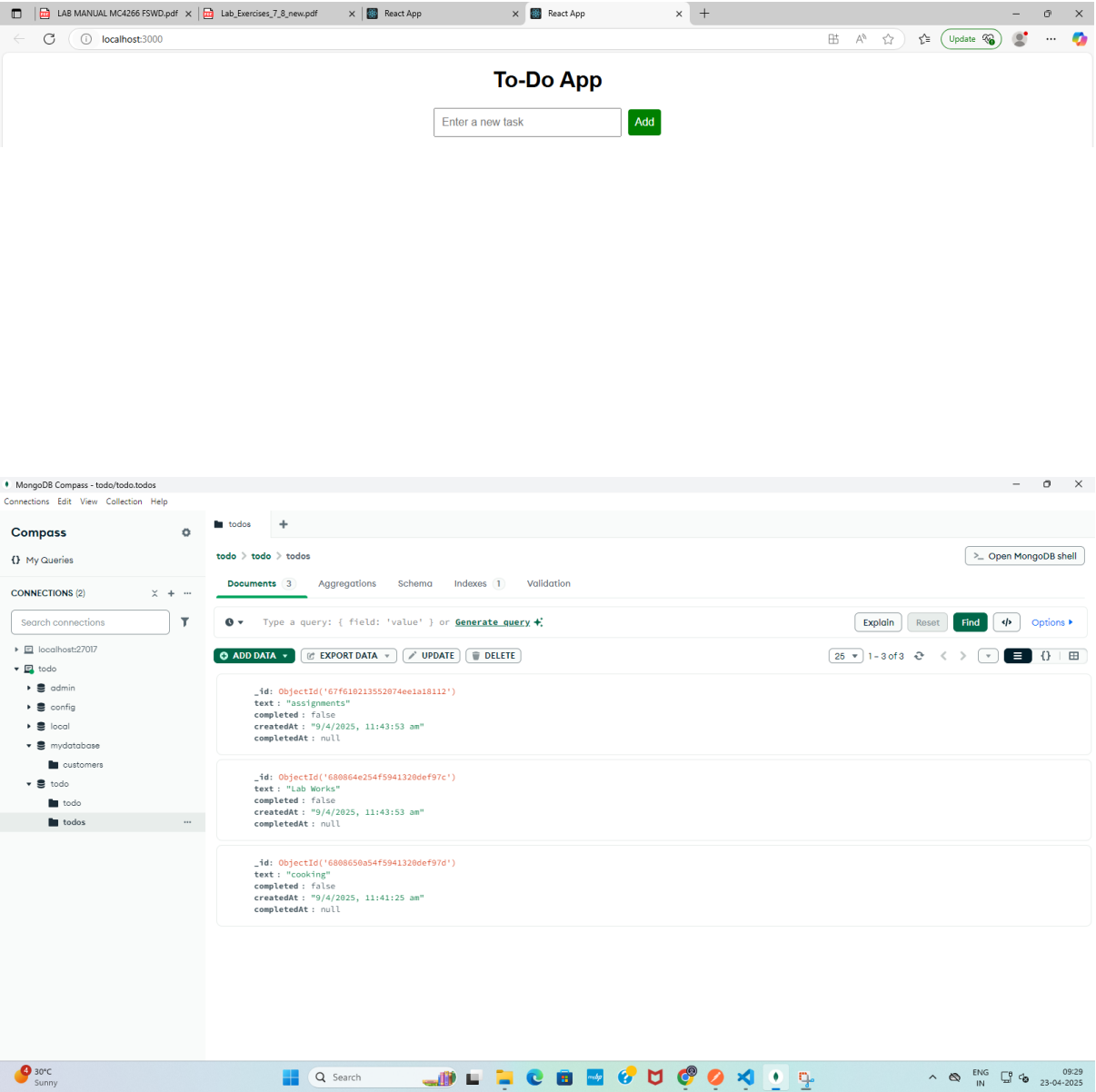
```
}  
.todo-input {  
  margin-bottom: 20px;  
}  
input {  
  padding: 10px;  
  font-size: 16px;  
  width: 250px;  
}  
button {  
  padding: 10px;  
  font-size: 16px;  
  cursor: pointer;  
  margin-left: 10px;  
  border: none;  
  border-radius: 4px;  
}  
.add-btn {  
  background-color: green;  
  color: white;  
}  
.add-btn:hover {  
  background-color: darkgreen;  
}  
.delete-btn {  
  background-color: red;  
  color: white;  
}  
.delete-btn:hover {  
  background-color: darkred;  
}  
.edit-btn {  
  background-color: orange;
```

```
    color: white;
}
.edit-btn:hover {
    background-color: darkorange;
}
.todo-table {
    width: 80%;
    margin: 0 auto;
    border-collapse: collapse;
}
th, td {
    padding: 10px;
    text-align: left;
    border: 1px solid #ddd;
}
th {
    background-color: #f2f2f2;
}
.completed {
    text-decoration: line-through;
    color: gray;
}
.edit-todo {
    margin-top: 20px;
}
.edit-todo input {
    padding: 10px;
    font-size: 16px;
    width: 250px;
}
.save-btn {
    background-color: blue;
    color: white;
```



```
    cursor: pointer;
    padding: 10px;
    margin-left: 10px;
}
.save-btn:hover {
    background-color: darkblue;
}
```

Output:



PROGRAM

Exercise 9

models/user.js

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

const UserSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true }
});

module.exports = mongoose.model('User', UserSchema);
```

routes/user.js

```
import express from 'express';
import { getAllUsers, login, logout, signUp } from "../controllers/user.js";
import { checkRole, checkToken } from '../middlewares/middlewares.js';
const router = express.Router();

router.post("/signUp", signUp);
router.post("/login", login);
router.post("/logout", checkToken, logout);
router.get('/getAllUsers', checkToken, checkRole(['admin', 'manager']), getAllUsers);

export default router;
```

Server.js

```
const express = require('express');
const connectDB = require('./config/db');
const cookieParser = require('cookie-parser');
const authRoutes = require('./routes/auth');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 2222;

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

```
app.use('/api/auth', authRoutes);
```

```
connectDB();
```

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

controller/user.js

```
import bcrypt from 'bcrypt';
```

```
import User from "../models/user.js";
```

```
import { CreateToken } from '../middlewares/middlewares.js';
```

```
import jwt from 'jsonwebtoken';
```

```
export const signUp = async (req, res) => {
```

```
  const { name, mobile, email, password, role } = req.body;
```

```
  if (!name || !mobile || !email || !password) {
```

```
    return res.status(422).json({ message: "All feilds should be filled" });
```

```
  }
```

```
  try {
```

```
    let existingUser;
```

```
    try {
```

```
      existingUser = await User.findOne({ $or: [{ email: email }, { mobile: mobile }] });
```

```
    } catch (err) {
```

```
      console.error(err);
```

```
    }
```

```
  if (existingUser) {
```

```
    if (existingUser.email === email) {
```

```
      return res.status(409).json({ message: "A User is already signUp with this email" });
```

```
    }
```

```
    else if (existingUser.mobile === mobile) {
```

```
      return res.status(409).json({ message: "A User is already signUp with this mobile" });
```

```
    }
```

```
  }
```

```
  const salt = await bcrypt.genSalt(6)
```

```
  const hashedpassword = await bcrypt.hash(password, salt);
```

```
  const user = new User({
```

```
    name,
```

```
    mobile,
```

```

    email,
    password: hashedpassword,
    role: role,
  });

  await user.save();
  return res.status(201).json({ message: "Account Creation is success, Login to your account", User: user
  })

  } catch (err) {
    console.error(err)
    return res.status(400).json({ message: "Error in saving user in DB" });
  }

}

export const login = async (req, res) => {

  const { email, password } = req.body;

  if (!email || !password) {
    return res.status(422).json({ message: "All feilds should be filled" })
  }

  let loggedUser;

  try {
    loggedUser = await User.findOne({ email: email });

    if (!loggedUser) {
      return res.status(404).json({ message: "Email is not found, Check it and try again" })
    }

    const isPasswordCorrect = bcrypt.compareSync(password, loggedUser.password);
    if (!isPasswordCorrect) {
      return res.status(400).json({ message: "Invalid password, Check it and try again" })
    }

    const token = CreateToken(loggedUser._id);
    res.cookie(String(loggedUser._id), token, {
      path: "/",
      expires: new Date(Date.now() + 1000 * 59),
      httpOnly: true    sameSite: "lax"
    })
  }

```

```

    return res.status(200).json({ message: "Successfully logged in", User: loggedUser })
  } catch (err) {
    console.log(err)
  }
}

```

```

export const logout = (req, res) => {
  const cookies = req.headers.cookie
  const previousToken = cookies.split("=")[1];

  if (!previousToken) {
    return res.status(400).json({ message: "Couldn't find token" });
  }
  jsonwebtoken.verify(String(previousToken), process.env.JWTAUTHSECRET, (err, user) => {
    if (err) {
      console.log(err);
      return res.status(403).json({ message: "Authentication failed" });
    }
    res.clearCookie(`${user.id}`);
    req.cookies[`${user.id}`] = "";
    return res.status(200).json({ message: "Successfully Logged Out" });
  });
};

```

```

export const getAllUsers = async (req, res) => {
  try {
    const allusers = await User.find();
    if (!allusers) {
      return res.status(404).json({ message: "There are not any users" });
    }
    else {
      res.status(200).json({ allusers })
    }
  } catch (error) {
    console.log(error);
    return res.status(500).json({ message: "Error in getting the Users" })
  }
}

```

Output

The screenshot shows the Postman application interface. The left sidebar displays a history of requests, with the selected request being a POST to `http://localhost:2222/auth/signup`. The main panel shows the request details for this endpoint. The request method is POST, and the body is a JSON object with the following fields: `name` (Sunil), `mobile` (8894941230), `email` (sunil@gmail.com), and `password` (12345). The response status is 201 Created, with a time of 13 ms and a size of 511 B. The response body is a JSON object with the following fields: `message` (Account Creation is success. Login to your account.), `User` (an object with name, mobile, email, and password), and `role` (custor).

```
{
  "message": "Account Creation is success. Login to your account.",
  "User": {
    "name": "Sunil",
    "mobile": "8894941230",
    "email": "sunil@gmail.com",
    "password": "12345"
  },
  "role": "custor"
}
```

The screenshot shows the Postman application interface. The left sidebar displays a history of requests, with the selected request being a POST to `http://localhost:2222/auth/login`. The main panel shows the request details for this endpoint. The request method is POST, and the body is a JSON object with the following fields: `email` (sunil@gmail.com) and `password` (12345). The response status is 200 OK, with a time of 38 ms and a size of 758 B. The response body is a JSON object with the following fields: `message` (Successfully logged in.), `User` (an object with _id, name, mobile, email, and password), and `role` (custor).

```
{
  "message": "Successfully logged in.",
  "User": {
    "_id": "68088351ef69fdb9b2ebb",
    "name": "Sunil",
    "mobile": "8894941230",
    "email": "sunil@gmail.com",
    "password": "12345"
  },
  "role": "custor"
}
```

PROGRAM

Ex10:

Step 1: Prepare your computer for Virtualization:

- Enable Processor Virtualization: Ensure Virtualization is enabled on your computer.

See the Virtualization Error (VT-d/VT-x or AMD-V) for troubleshooting support.

- Review File Sync Services for tools like OneDrive, Nextcloud, DropBox Sync, iCloud, etc. If you are using a data synchronization service, make sure it DOES NOT (or at least not frequently) synchronize the folder in which your hypervisor imports and installs the Virtual Machines.

- File sync services can cause a dramatic fall-off in performance for your entire system as these services try to synchronize these massive files that are getting updated constantly while you are using the Virtual Machines.

- Sufficient Disk Space: Virtual Machines require a significant amount of Disk space (10 GB or more each is typical). Ensure you have sufficient space on your computer.

- Admin Privileges: Installing a hypervisor on a host in most cases requires admin privileges.

Step 2: Install Hypervisor (Virtualization Tool):

Installing a hypervisor on your host is usually quite simple. In most cases, the install program will ask only a couple of questions, such as where to install the hypervisor software.

Step 3: Import a Virtual Machine:

- The first step is to download the Virtual Machine for your course from our Course Virtual Machines page. This will download an .ova file. The .ova file is actually a compressed (zipped) tarball of a Virtual Machine exported from Virtual Box.
- Once the Virtual Machine has been imported, it will normally show up in the guest list within your hypervisor tool.

Step 4: Start the Virtual Machine:

To start up a Virtual Machine guest in most hypervisors, you simply click on the desired guest and click the Start button (often double-clicking the guest icon will work as well).

Step 5: Using the Virtual Machine:

- Sharing files between the guest and host: To learn about different ways of sharing files, check out this guide.
- Run a command with sudo (root) privileges: Open a terminal and type any command with sudo in front to run that command as root.
- Example: `sudo apt-get install vim` – will install the vim text editor package on an Ubuntu Linux Virtual Machine.
- Find the IP address of your guest: Open a terminal and type `ifconfig | more` – The `| more` (pronounced “pipe more”) will “pipe” the output of the `ifconfig` command to the `more` command, which will show the results one page at a time, so it doesn’t scroll by before you see it all.
- If you have a Host-Only Network IP address, you will see an IP of 192.168.56.101 (or something similar). Check the Trouble-Shooting section below for more information about the Host-Only Network.

Step 6: Shut down the Virtual Machine:

When you are done using a guest Virtual Machine, regardless of hypervisor, you need to shut it down properly. This can be done in three ways:

1. Press the shutdown button found on the desktop, taskbar, or task menu of the guest operating system.
2. Open a terminal and type the command: `sudo shutdown -h now`
3. In the guest window, click Machine (menu) -> ACPI Shut down – This will simulate the power button being pressed

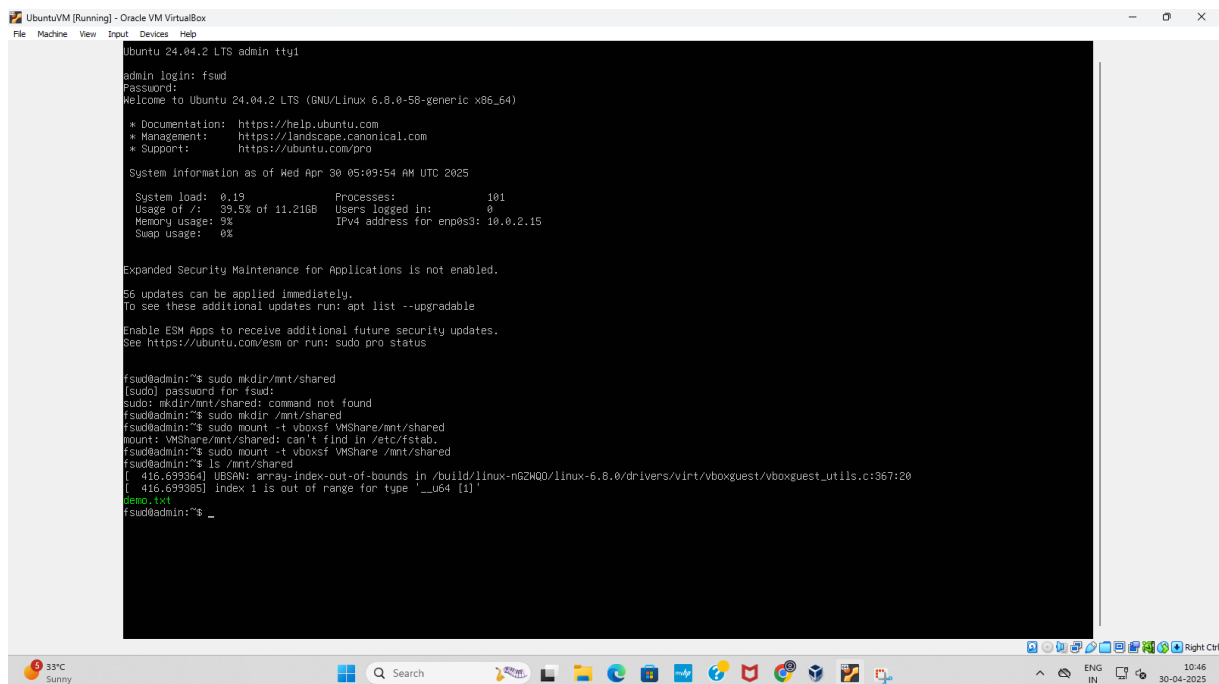
Output:

Bash

ssh [USERNAME]@[IP_ADDRESS]

Enter Password: Enter the password for the specified user.

SSH Connection: A screenshot of the terminal window on the host computer, displaying a successful SSH connection to the guest OS.



```
Ubuntu 24.04.2 LTS admin tty1
admin login: fsud
Password:
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-58-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

System information as of Wed Apr 30 05:09:54 AM UTC 2025

System load:  0.19           Processes:    101
Usage of /:   39.5% of 11.21GB Users logged in:   0
Memory usage: 9%            IPv4 address for enp0s3: 10.0.2.15
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

56 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

fsud@admin:~$ sudo mkdir /mnt/shared
[sudo] password for fsud:
fsud@admin:~$ sudo mkdir /mnt/shared
fsud@admin:~$ sudo mount -t vboxsf VMShare /mnt/shared
mount: VMShare/mnt/shared: can't find in /etc/fstab.
fsud@admin:~$ sudo mount -t vboxsf VMShare /mnt/shared
fsud@admin:~$ ls /mnt/shared
[ 416.699364] UBSAN: array-index-out-of-bounds in /build/linux-n62WQ0/linux-6.8.0/drivers/virt/vboxguest/vboxguest_utils.c:367:20
[ 416.699385] index 1 is out of range for type '___u64 [1]'
fsud@admin:~$
```

PROGRAM**Ex11_server.js:**

```

const http = require('http');
const hostname = '0.0.0.0';
const port = 8080;
const server = http.createServer((req, res) => {
  if (req.method === 'GET' && req.url === '/ping') {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'application/json');
    res.end(JSON.stringify({ message: 'pong' }));
  } else {
    res.statusCode = 404;
    res.end('Not Found');
  }
});
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});

```

Dockerfile:

```

# Use official Node.js image from Docker Hub
FROM node:16

# Set the working directory inside the container
WORKDIR /usr/src/app

# Copy the server.js file to the working directory
COPY server.js .

# Expose the port the app will run on
EXPOSE 8080

# Run the Node.js server
CMD ["node", "server.js"]

```

Output:

After running the server, you should be able to access the endpoint via

<http://localhost:8080/ping>, and it should respond with the following JSON message:

json

```
{  
  "message": "pong"  
}
```

