**PROGRAM PLANNING AND ORGANIZING EVENTS**

TEAM MEMBERS:

|  |  |
| --- | --- |
| ROLL NUMBER: | NAME: |
| SALINIYAN P | 22ALR083 |
| SANJAY A | 22ALR084 |
| YESHWANTH A | 22ALR115 |

**INTRODUCTION:**

Our innovative application leverages React, CSS, MongoDB, and Node.js to provide a seamless solution for program planning and event organization. Designed for efficiency and user-friendliness, it empowers users to effortlessly manage and book events with just a few clicks. The intuitive interface ensures a smooth experience for creating, editing, and tracking events, while the robust backend guarantees data integrity and swift processing. Users can customize their events, manage guest lists, and receive real-time updates, making event management hassle-free. Additionally, the application offers insightful analytics and reporting tools to help users evaluate the success of their events. With our application, planning and organizing events has never been easier or more efficient.

**Key Features:**

1. User Logins:

- Different login roles for Central users, Students, and Event Organizers, each with specific access and functions.

2. Event Creation:

- Event Organizers can easily add event details such as name, date, time, location, and description.

3. Event Approval:

- Central users can review and either approve or reject event submissions. Approved events are then made visible to students.

4. Event Display and Registration:

- Students can browse approved events and register for them directly through the application.

5. Participant Management:

- Event Organizers can access and manage information about registered participants.

6. Calendar Scheduling:

- Integrated calendar system helps to schedule events without conflicts, ensuring proper timing.

7. Secure Login:

- Features a two-step verification process to ensure secure access for all users.

**REQUIREMENTS:**

**Front-End Programming:**

1. **CSS (Cascading Style Sheets)**:
   * CSS is used to define the visual appearance and layout of web pages. It allows developers to apply styles, such as colors, fonts, spacing, and positioning, to HTML elements, creating a visually appealing and consistent user interface.
2. **JavaScript**:
   * JavaScript is a powerful programming language that adds interactivity and dynamic behavior to web pages. It enables developers to manipulate and modify HTML and CSS, handle user interactions, perform calculations, validate forms, and make asynchronous requests to servers.
3. **React**:
   * React is a popular JavaScript library for building user interfaces. It provides a component-based approach to web development, allowing developers to create reusable UI components that update efficiently based on changes in data. React uses a virtual DOM (Document Object Model) to optimize performance and facilitate the building of complex web applications.

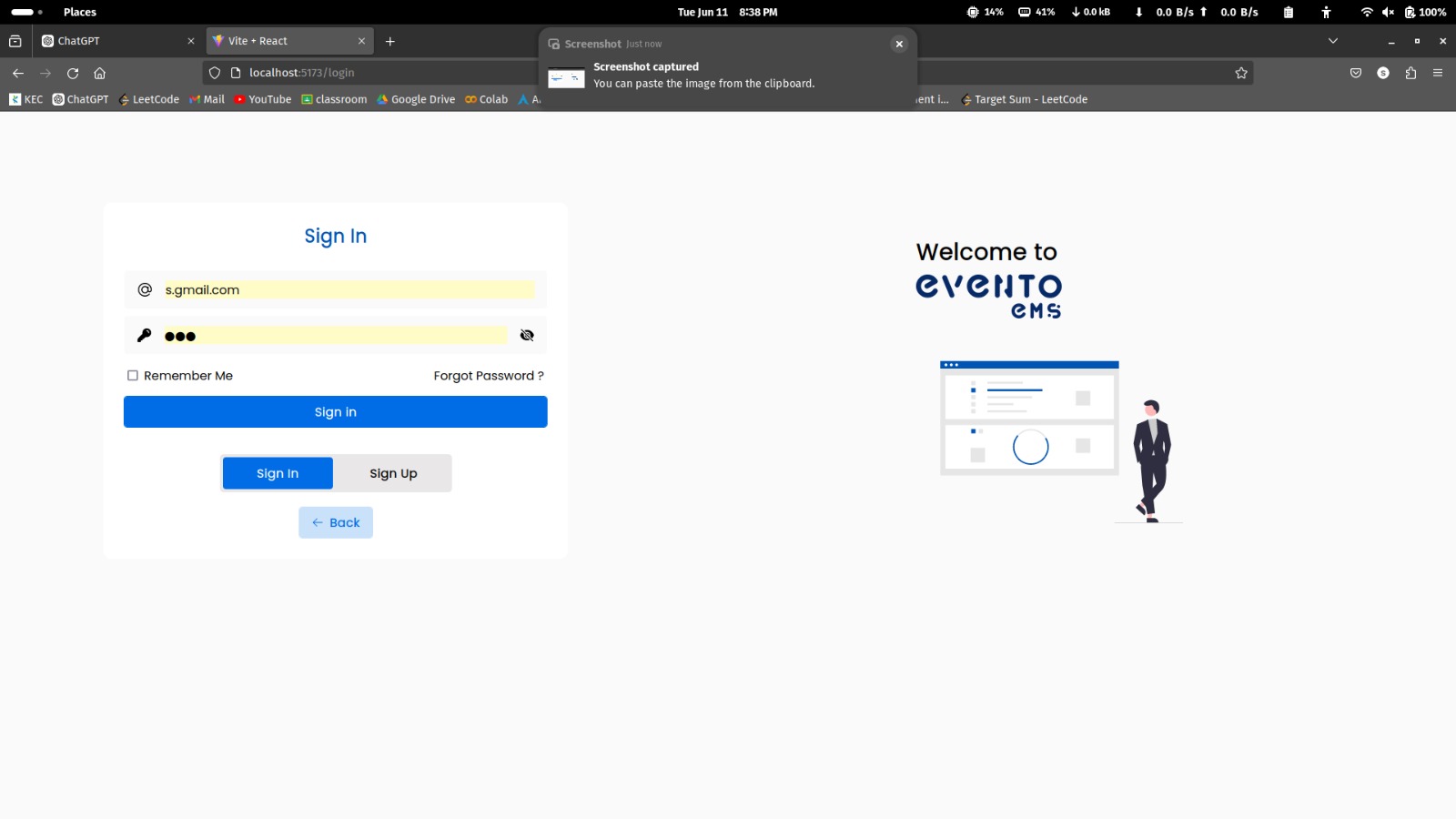
**Back-End Programming**

**Database Management**:

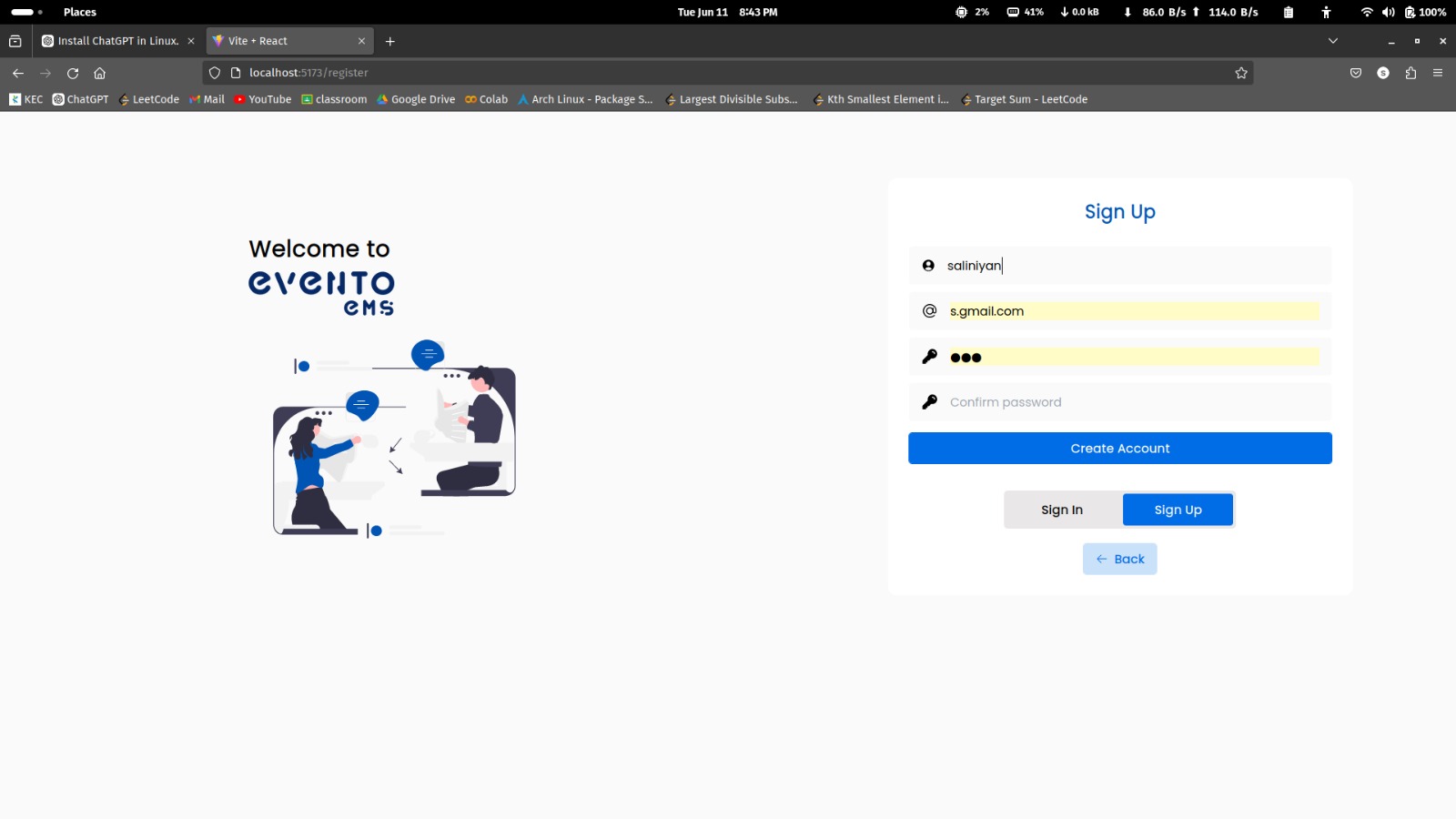
* + Backend programming involves working with databases to store and retrieve data. Developers interact with databases using query languages like SQL or NoSQL databases. For this system, MongoDB is used to design and implement database schemas, optimize queries, and ensure data integrity and security.

**UI DESIGN:**

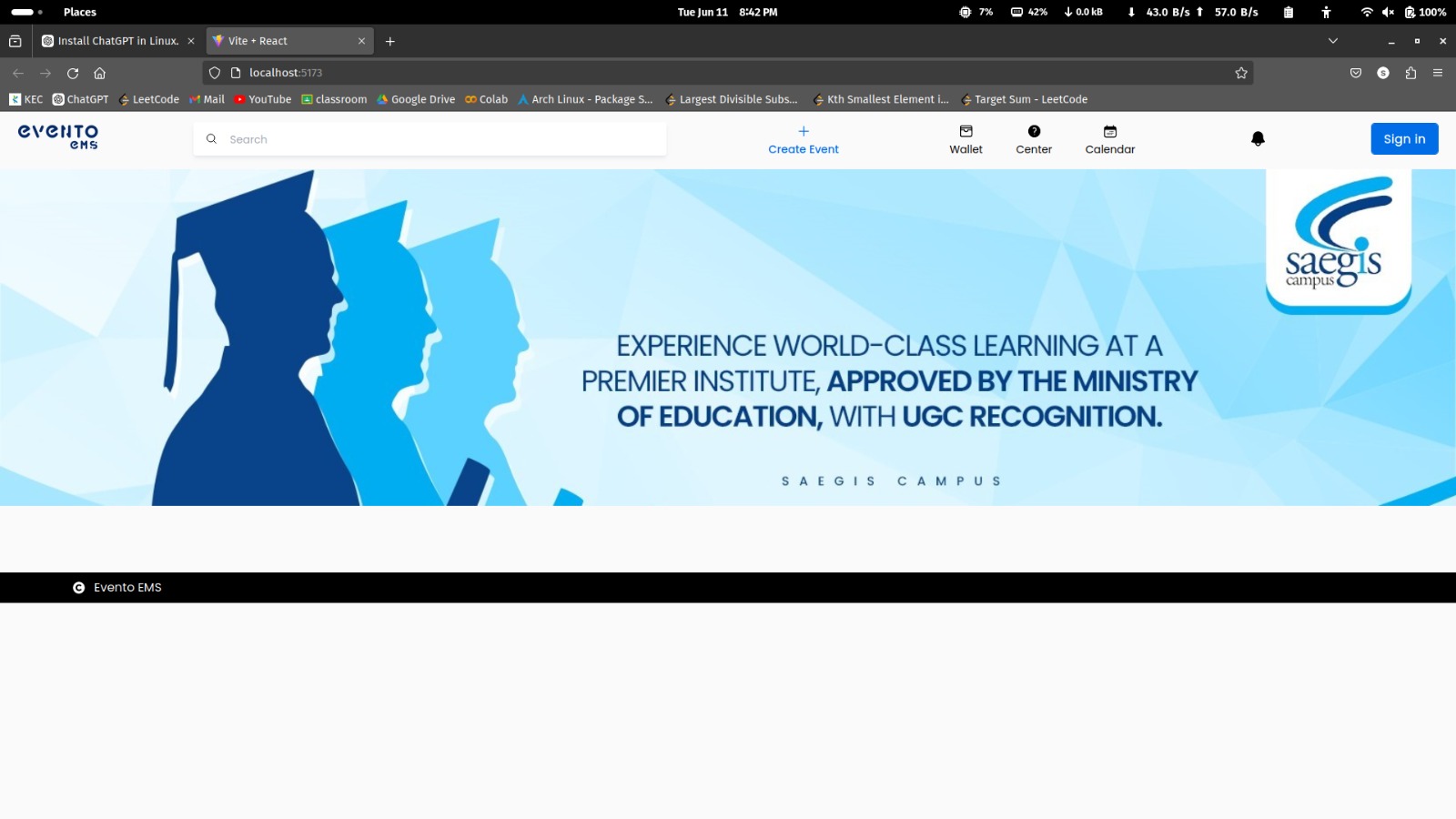
**LOGIN PAGE:**

****

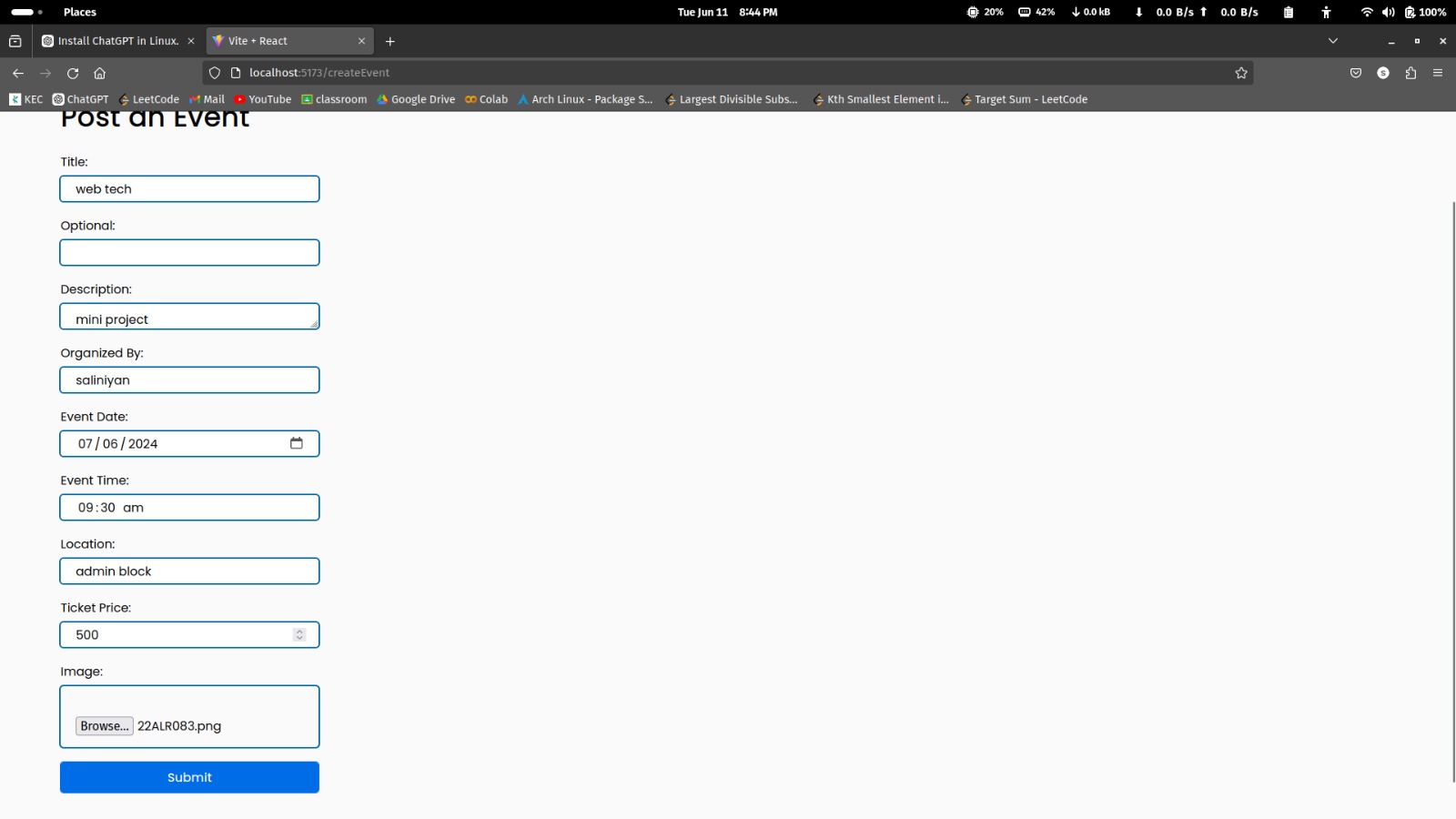
**SIGN UP PAGE:**

****

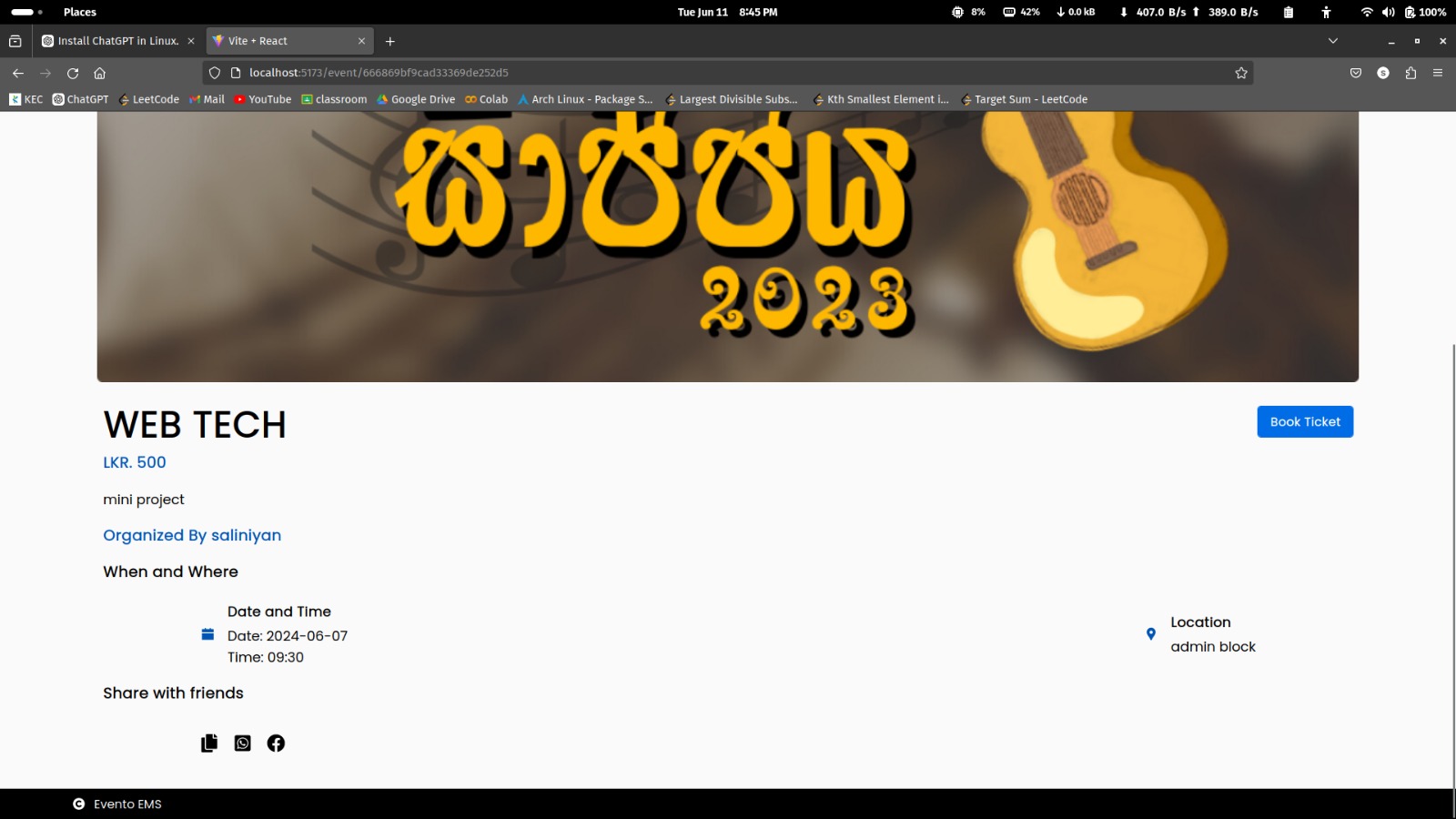
**HOME PAGE:**

****

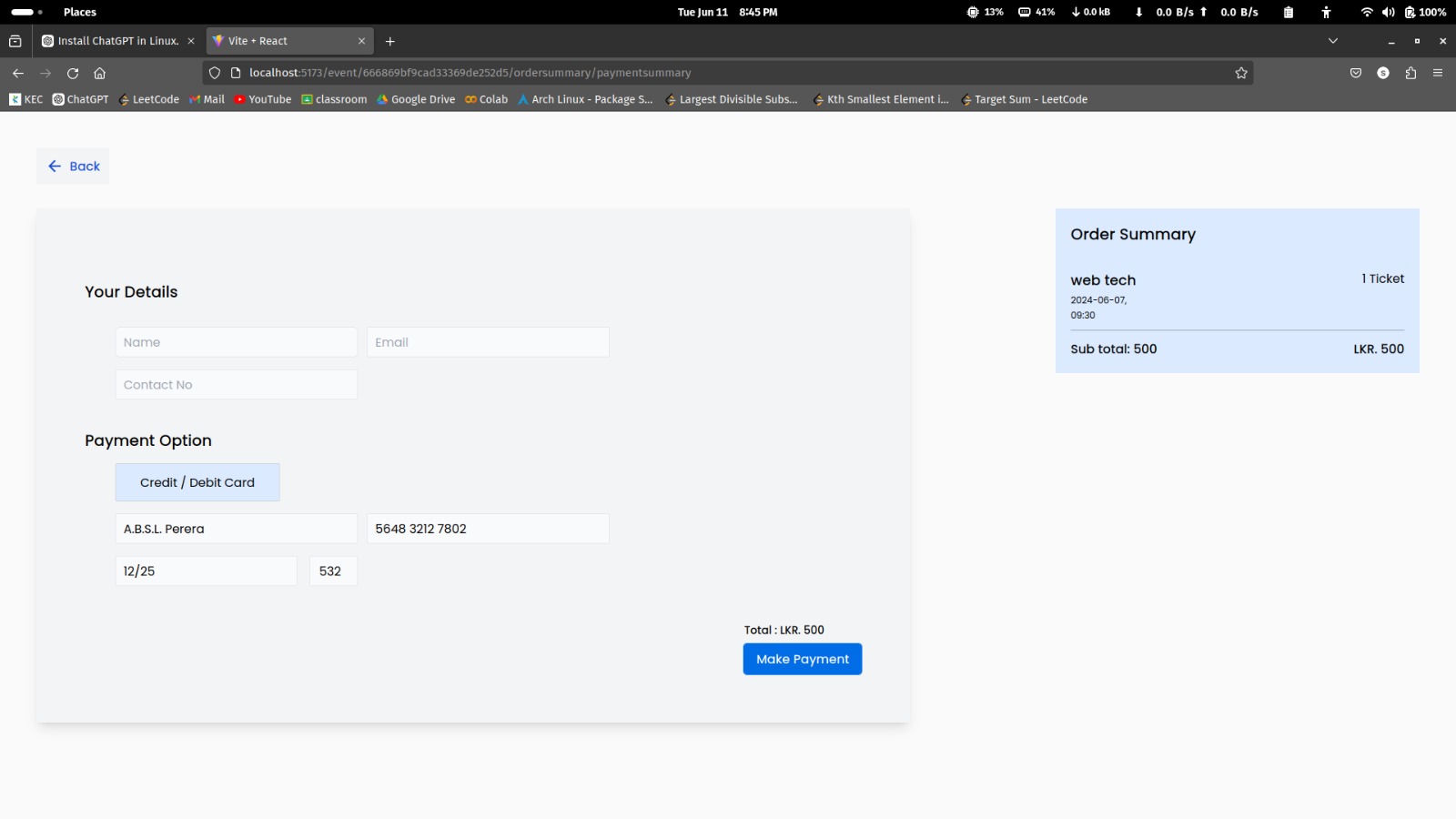
**EVENT ORGANIZING PAGE:**

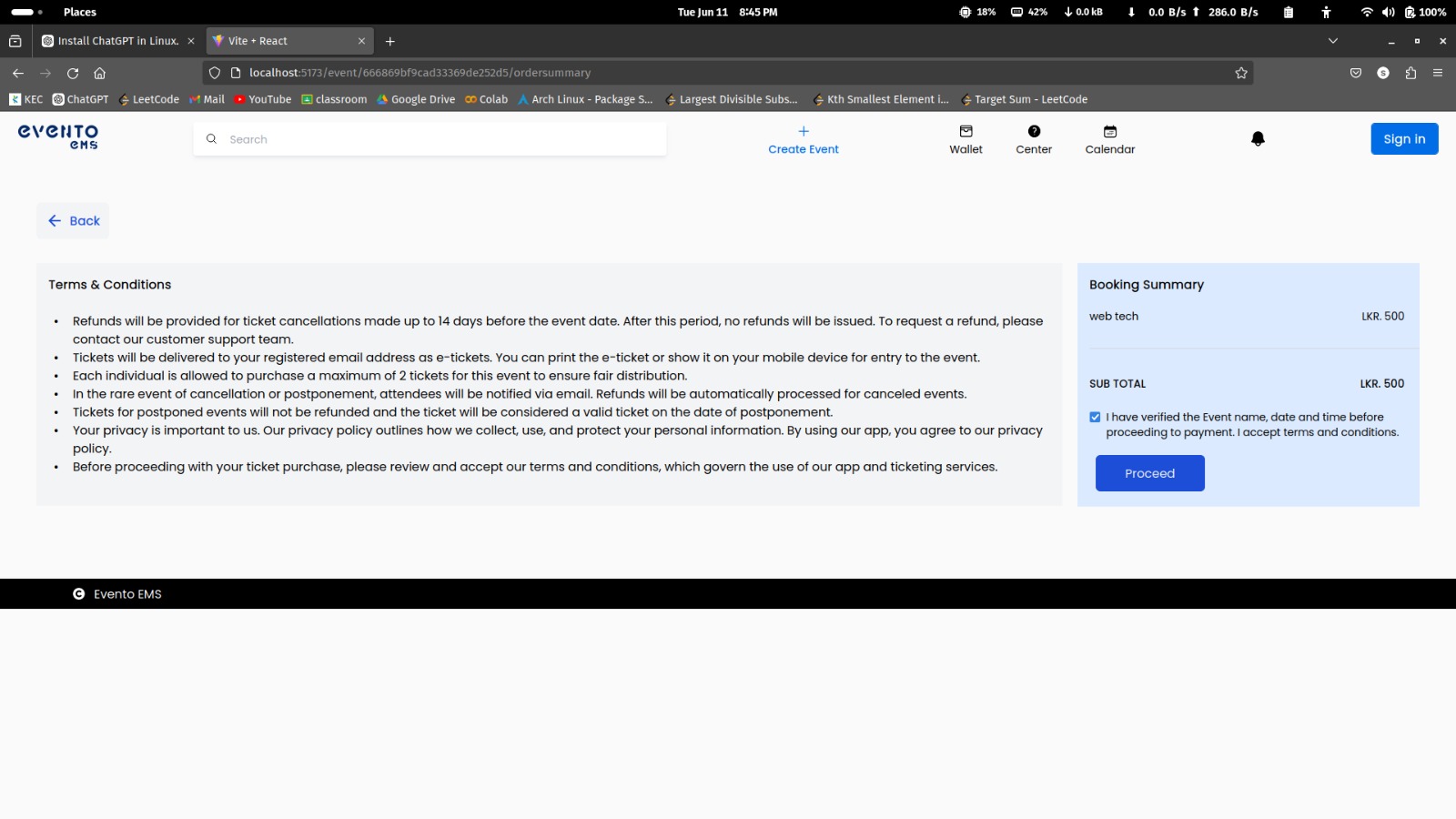
****

**BOOKING PAGE:**

****

**CONFIRMATION PAGE:**

****

**FINAL PAGE:**

**CODING:**

**INDEX.CSS:**

**index.css**

**@import url('https://fonts.googleapis.com/css2?family=Poppins&display=swap');**

**@tailwind base;**

**@tailwind components;**

**@tailwind utilities;**

**body{**

**background-color: #fafafa;**

**font-family: Poppins;**

**}**

**button.primary{**

**@apply text-white py-2 px-4 bg-primary cursor-pointer ring-1 ring-primary rounded font-bold hover:bg-white hover:shadow-lg duration-75 hover:ring-primarydark hover:text-primary;**

**}**

**button.secondary{**

**@apply flex items-center gap-2 text-primary py-2 px-4 bg-primarylight cursor-pointer ring-1 font-bold ring-primarylight rounded hover:bg-primarydark hover:shadow-lg duration-75 hover:ring-primarydark hover:text-white;**

**}**

**div.input{**

**@apply flex bg-background rounded w-full py-2.5 px-4 gap-4 items-center ;**

**}**

**input.input-et{**

**@apply text-base bg-inherit text-black outline-none w-full;**

**}**

**.custom-list {**

**list-style-type: none;**

**padding-left: 2.0em;**

**}**

**.custom-list li:before {**

**content: "•";**

**font-size: 17px;**

**position: absolute;**

**margin-left: -1.5em;**

**}**

**.truncate-text {**

**display: -webkit-box;**

**-webkit-line-clamp:3 ;**

**-webkit-box-orient: vertical;**

**overflow: hidden;**

**text-overflow: ellipsis;**

**}**

**APP.CSS:**

.container{

width: 80vw; /\* 80% of the viewport width \*/

max-width: 1200px; /\* maximum width for the container \*/

height: 650px;

/\* add flex style \*/

display: flex;

justify-content: space-around;

align-content: center;

position: absolute ;

top: 50%;

left: 50%;

transform: translate(-50%, -50%);

}

.left-box{

width: auto;

background: white;

margin: 20px;

display: flex;

justify-content: center;

align-items: center;

}

.right-box {

width: 50%;

height: 93%;

display: flex;

justify-content: center;

align-items: center;

margin: 20px;

}

.form {

width: 100%;

max-width: 400px;

padding: 20px;

}

.imgs{

justify-content: center;

align-items: center;

width: 50%;

}

.sign-pic{

width: fit-content;

height: 100%;

}

.h1 {

font-weight: 800;

color: #0054B3;

text-align: left;

font-size: 40px;

margin-bottom: 20px;

}

.input {

width: 100%;

height: 50px;

padding-left: 70px;

font-size: 15px;

border-radius: 5px;

border: none;

outline: none;

background-color: #f0f0f0;

margin-top: 10px;

}

/\* icons positoning starts\*/

.name,

.email,

.passw,

.pass2 {

height: 25px;

width: 25px;

position: absolute;

padding: 14px 0 0 20px;

}

/\* icon positioning ends \*/

.form-group{

padding-top: 20px;

}

.create-btn {

width: 100%;

height: 50px;

font-size: 18px;

font-weight: bold;

font-family: 'Poppins', sans-serif;

color: white;

border-radius: 5px;

border: none;

background-color: #006CE6;

cursor: pointer;

margin-top: 10px;

}

.container2 {

display: flex;

justify-content: center;

align-items: center;

background-color: rgb(233, 231, 231);

width: 306px;

height: 50px;

margin: 20px auto;

border-radius: 5px;

overflow: hidden;

}

.slider{

width: 100%;

height: 100%;

transition: transform 0.3s ease-in-out;

transform: translateX(0%);

}

.slider.active{

transform: translateX(-50%);

}

.slider-signup-btn{

margin-top: 2px;

margin-bottom: 2px;

margin-right: 1px;

margin-left: 1px;

width: 150px;

height: 46px;

font-size: 15px;

font-weight: 800;

border: none;

border-radius: 2px;

color: white;

background-color: #006CE6;

outline: none;

cursor: pointer;

}

.slider-signin-btn{

margin-top: 2px;

margin-bottom: 2px;

margin-right: 1px;

margin-left: 1px;

width: 150px;

height: 48px;

font-size: 15px;

font-weight: 800;

border: none;

border-radius: 2px;

color: rgb(8, 1, 1);

outline: none;

cursor: pointer;

}

.slider-btn:last-child {

background-color: #b3b3b3;

}

.slider-btn:focus {

box-shadow: 0px 0px 5px #0054b3;

}

/\* Media Queries \*/

@media (max-width: 768px) {

.container {

flex-direction: column;

align-items: center;

height: auto;

}

.left-box {

width: 100%;

margin: 20px 0;

}

.sign-pic {

width: 100%;

height: auto;

margin: 0;

}

.input {

width: 100%;

padding-left: 50px;

}

.h1 {

font-size: 30px;

text-align: center;

}

.slider-signup-btn,

.slider-signin-btn {

width: 120px;

font-size: 12px;

}

}

@media (max-width: 550px) {

.h1 {

font-size: 30px;

text-align: center;

}

.h1 {

font-size: 20px;

}

.slider-signup-btn,

.slider-signin-btn {

width: 100px;

font-size: 10px;

}

}

/\* Login CSS temp \*/

.outermainDiv{

display: flex;

justify-content: center;

align-items: center;

margin-top: 50px;

width:98% ;

height: 90vh;

}

.outerdiv{

display: flex;

justify-content: center;

align-items: center;

position: relative;

width: 80%;

height: 80vh;

background-color: rgb(243, 243, 243);

}

.welocme{

margin-left: 5px;

font-size: 30px;

font-family: sans-serif;

margin-top: 10px;

font-style: normal;

font-weight: bold;

color: #00070E;

}

.welcomImage{

width: 350px;

height: 200px;

margin-left: 5px;

margin-top: 100px;

}

.logoImage{

width: 200px;

height: 75px;

margin-left: 5px;

}

.imageHolder{

position: relative;

width: 40%;

height: 80%;

background-color: rgb(243, 243, 243);

}

.inforHolder{

background-color: white;

width: 40%;

box-shadow: 1px 1px 10px 10px rgba(0, 0, 0, 0.122);

height: 80%;

position: relative;

}

.signText{

margin-left: 20px;

font-family: sans-serif;

font-weight: bold;

font-size: 30px;

padding: 15px;

margin-top: 25px;

color: #0054B3;

left: 44.38%;

right: 44.38%;

top: 15.23%;

bottom: 78.71%;

}

.InputHolder{

position: relative;

width: 100%;

padding: 15px;

height: 100%;

margin-left: 20px;

align-items: center;

}

.inputText{

font-size: 12px;

font-family: sans-serif;

margin: 5px;

}

.EmailPut{

background: #FAFAFA;

left: 180px;

outline: none;

text-decoration: none;

border: none;

}

.passwordPut{

background: #FAFAFA;

left: 180px;

outline: none;

text-decoration: none;

border: none;

}

.fa-envelope{

position: relative;

margin: 5px;

}

.fa-lock{

position: relative;

margin: 5px;

}

.fa-eye{

position: relative;

width: 50px;

height: 50px;

margin: 5px;

right: 0px;

left: 130px;

cursor: pointer;

}

.forgetpasswordDiv{

display: flex;

justify-content: space-evenly;

align-items: center;

width: 80%;

}

.rememberMe{

position: absolute;

float: left;

left: 25px;

margin-left: 21px;

font-family: sans-serif;

font-size: 15px;

top: 150px;

}

.cjeckBtInput{

position: absolute;

float: left;

left: 5px;

margin-left: 21px;

top: 150px;

}

.forgetpassword{

position: relative;

float: right;

left: 150px;

margin-right: 25px;

font-family: sans-serif;

font-size: 15px;

color: #0054B3;

top: 22px;

}

.loinbutton{

position: relative;

width: 80%;

background-color: blue;

color: white;

font-size: 17px;

font-weight: bold;

border-radius: 10px;

height: 50px;

margin: 10px;

cursor: pointer;

border: none;

padding: 5px;

font-family: sans-serif;

font-size: 18px;

top: 30px;

}

.buttonHolder{

display: flex;

justify-content: space-between;

position: relative;

align-items: center;

width: 50%;

bottom: 150px;

position: relative;

border-radius: 8px;

margin: 13px;

background: #EEEEEE;

top: 20px;

left: 70px;

}

.actionBtns{

width: 50%;

background: #EEEEEE;

margin: 2em;

display: flex;

justify-content: space-between;

position: relative;

border-radius: 8px;

box-shadow:rgba(0,0,0,0.09) 0px 3px 12px;

left: 45px;

font-family: sans-serif;

height: 50px;

}

.actionBtn{

padding: 1em;

width: 50%;

outline: none;

border: none;

background-color: transparent;

background: #EEEEEE;

font-family: sans-serif;

font-size: 18px;

font-weight: 600;

cursor: pointer;

height: 50px;

}

.moveBtn{

position: absolute;

width: 50%;

height: 90%;

margin: 2px;

border: none;

outline: none;

border-radius: 8px;

font-family: sans-serif;

background: #006CE6;

font-size: 18px;

font-weight: 600;

box-shadow:rgba(0,0,0,0.1) 0px 4px 12px;

transform: translateX(0);

transition: all 0.2s ease-in-out;

}

.rightBtn{

transform: translateX(98%);

transition: all 0.2s ease-in-out;

}

.inputGroup{

width: 80%;

height: 50px;

margin: 2px;

background: #FAFAFA;

border-radius: 8px;

font-weight: 500;

justify-content: space-between;

}

.truncate-3-lines {

display: -webkit-box;

-webkit-box-orient: vertical;

-webkit-line-clamp: 3; /\* Number of lines to show \*/

overflow: hidden;

text-overflow: ellipsis;

}

APP.JS:

app.js

/\* eslint-disable no-unused-vars \*/

import { Route, Routes } from 'react-router-dom'

import './App.css'

import IndexPage from './pages/IndexPage'

import RegisterPage from './pages/RegisterPage'

import Layout from './Layout'

import LoginPage from './pages/LoginPage'

import axios from 'axios'

import { UserContextProvider } from './UserContext'

import UserAccountPage from './pages/UserAccountPage'

import ForgotPassword from './pages/ForgotPassword'

import ResetPassword from './pages/ResetPassword'

import AddEvent from './pages/AddEvent'

import EventPage from './pages/EventPage'

import CalendarView from './pages/CalendarView'

import OrderSummary from './pages/OrderSummary'

import PaymentSummary from './pages/PaymentSummary'

import TicketPage from './pages/TicketPage'

import CreatEvent from './pages/CreateEvent'

axios.defaults.baseURL = 'http://localhost:4000/';

axios.defaults.withCredentials=true;

function App() {

return (

<UserContextProvider>

<Routes>

<Route path='/' element={<Layout />}>

<Route index element = {<IndexPage />} />

<Route path='/useraccount' element = {<UserAccountPage />}/>

<Route path='/createEvent' element = {<AddEvent/>} />

<Route path='/event/:id' element= {<EventPage/>} />

<Route path='/calendar' element={<CalendarView />} />

<Route path='/wallet' element={<TicketPage />}/>

<Route path='/event/:id/ordersummary' element = {<OrderSummary />} />

</Route>

<Route path='/register' element={<RegisterPage />}/>

<Route path='/login' element={<LoginPage />}/>

<Route path='/forgotpassword' element = {<ForgotPassword/>} />

<Route path='/resetpassword' element = {<ResetPassword/>} />

<Route path='/event/:id/ordersummary/paymentsummary' element = {<PaymentSummary />} />

</Routes>

</UserContextProvider>

)

}

export default App;

INDEX.HTML:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<link rel="icon" type="image/svg+xml" href="/vite.svg" />

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

<title>Vite + React</title>

</head>

<body>

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

<script type="module" src="/src/main.jsx"></script>

</body>

</html>

**.ENV:**

.env

MONGO\_URL=mongodb://localhost:27017/saliniyan

MONGODB\_URI=mongodb://localhost:27017/saliniyan

**index.js**

const express = require("express");

const cors = require("cors");

require("dotenv").config();

const mongoose = require("mongoose");

const UserModel = require("./models/User");

const bcrypt = require("bcryptjs");

const jwt = require("jsonwebtoken");

const cookieParser = require("cookie-parser");

const multer = require("multer");

const path = require("path");

const Ticket = require("./models/Ticket");

const app = express();

const bcryptSalt = bcrypt.genSaltSync(10);

const jwtSecret = "bsbsfbrnsftentwnnwnwn";

app.use(express.json());

app.use(cookieParser());

app.use(

cors({

credentials: true,

origin: "http://localhost:5173",

})

);

mongoose.connect(process.env.MONGO\_URL);

const storage = multer.diskStorage({

destination: (req, file, cb) => {

cb(null, "uploads/");

},

filename: (req, file, cb) => {

cb(null, file.originalname);

},

});

const upload = multer({ storage });

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

res.json("test ok");

});

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

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

try {

const userDoc = await UserModel.create({

name,

email,

password: bcrypt.hashSync(password, bcryptSalt),

});

res.json(userDoc);

} catch (e) {

res.status(422).json(e);

}

});

app.post("/login", async (req, res) => {

const { email, password } = req.body;

const userDoc = await UserModel.findOne({ email });

if (!userDoc) {

return res.status(404).json({ error: "User not found" });

}

const passOk = bcrypt.compareSync(password, userDoc.password);

if (!passOk) {

return res.status(401).json({ error: "Invalid password" });

}

jwt.sign(

{

email: userDoc.email,

id: userDoc.\_id,

},

jwtSecret,

{},

(err, token) => {

if (err) {

return res.status(500).json({ error: "Failed to generate token" });

}

res.cookie("token", token).json(userDoc);

}

);

});

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

const { token } = req.cookies;

if (token) {

jwt.verify(token, jwtSecret, {}, async (err, userData) => {

if (err) throw err;

const { name, email, \_id } = await UserModel.findById(userData.id);

res.json({ name, email, \_id });

});

} else {

res.json(null);

}

});

app.post("/logout", (req, res) => {

res.cookie("token", "").json(true);

});

const eventSchema = new mongoose.Schema({

owner: String,

title: String,

description: String,

organizedBy: String,

eventDate: Date,

eventTime: String,

location: String,

Participants: Number,

Count: Number,

Income: Number,

ticketPrice: Number,

Quantity: Number,

image: String,

likes: Number,

Comment: [String],

});

const Event = mongoose.model("Event", eventSchema);

app.post("/createEvent", upload.single("image"), async (req, res) => {

try {

const eventData = req.body;

eventData.image = req.file ? req.file.path : "";

const newEvent = new Event(eventData);

await newEvent.save();

res.status(201).json(newEvent);

} catch (error) {

res.status(500).json({ error: "Failed to save the event to MongoDB" });

}

});

app.get("/createEvent", async (req, res) => {

try {

const events = await Event.find();

res.status(200).json(events);

} catch (error) {

res.status(500).json({ error: "Failed to fetch events from MongoDB" });

}

});

app.get("/event/:id", async (req, res) => {

const { id } = req.params;

try {

const event = await Event.findById(id);

res.json(event);

} catch (error) {

res.status(500).json({ error: "Failed to fetch event from MongoDB" });

}

});

app.post("/event/:eventId", (req, res) => {

const eventId = req.params.eventId;

Event.findById(eventId)

.then((event) => {

if (!event) {

return res.status(404).json({ message: "Event not found" });

}

event.likes += 1;

return event.save();

})

.then((updatedEvent) => {

res.json(updatedEvent);

})

.catch((error) => {

console.error("Error liking the event:", error);

res.status(500).json({ message: "Server error" });

});

});

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

Event.find()

.then((events) => {

res.json(events);

})

.catch((error) => {

console.error("Error fetching events:", error);

res.status(500).json({ message: "Server error" });

});

});

app.get("/event/:id/ordersummary", async (req, res) => {

const { id } = req.params;

try {

const event = await Event.findById(id);

res.json(event);

} catch (error) {

res.status(500).json({ error: "Failed to fetch event from MongoDB" });

}

});

app.get("/event/:id/ordersummary/paymentsummary", async (req, res) => {

const { id } = req.params;

try {

const event = await Event.findById(id);

res.json(event);

} catch (error) {

res.status(500).json({ error: "Failed to fetch event from MongoDB" });

}

});

app.post("/tickets", async (req, res) => {

try {

const ticketDetails = req.body;

const newTicket = new Ticket(ticketDetails);

await newTicket.save();

return res.status(201).json({ ticket: newTicket });

} catch (error) {

console.error("Error creating ticket:", error);

return res.status(500).json({ error: "Failed to create ticket" });

}

});

app.get("/tickets/:id", async (req, res) => {

try {

const tickets = await Ticket.find();

res.json(tickets);

} catch (error) {

console.error("Error fetching tickets:", error);

res.status(500).json({ error: "Failed to fetch tickets" });

}

});

app.get("/tickets/user/:userId", (req, res) => {

const userId = req.params.userId;

Ticket.find({ userid: userId })

.then((tickets) => {

res.json(tickets);

})

.catch((error) => {

console.error("Error fetching user tickets:", error);

res.status(500).json({ error: "Failed to fetch user tickets" });

});

});

app.delete("/tickets/:id", async (req, res) => {

try {

const ticketId = req.params.id;

await Ticket.findByIdAndDelete(ticketId);

res.status(204).send();

} catch (error) {

console.error("Error deleting ticket:", error);

res.status(500).json({ error: "Failed to delete ticket" });

}

});

const PORT = process.env.PORT || 4000;

app.listen(PORT, () => {

console.log(Server is running on port ${PORT});

});

**CONCLUSION:**

In conclusion, an Event Management System (EMS) is a comprehensive software solution that plays a crucial role in transforming the management and execution of events. By digitizing and automating administrative and operational processes, an EMS improves the efficiency, accuracy, and quality of event management and also ensures the security of data. With features such as user logins for central, student, and event organizers, event creation and approval, participant management, calendar integration, secure login, and real-time notifications, an EMS streamlines workflows, reduces manual tasks, enhances event coordination, and promotes better engagement among participants.

Implementing an EMS brings numerous benefits, including improved operational efficiency, reduced errors, optimized resource utilization, enhanced participant experience, and increased satisfaction for all stakeholders involved. It empowers event organizers to focus more on delivering successful events while the system takes care of administrative tasks and data management. Overall, an EMS revolutionizes the way events are managed by leveraging technology to streamline processes, improve decision-making, and drive better outcomes. It serves as a central platform for managing event information, participant data, and operational details, ultimately leading to more efficient and effective event execution.