JEPPIAAR
COLLEGE OF ARTS AND SCIENCE

IBM

JEPPIAAR
COLLEGE OF ARTS AND SCIENCE

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

IBM FINAL PROJECT
EVENT BOOKING AND TICKET MANAGEMENT PLATFORM USING MICROSERVICES

Submitted by

ALFI JAMES (222312119)

NITHYAA SRI. S (222312126)

JESHWANTH. G (222312108)

BACHELOR OF SCIENCE IN COMPUTER SCIENCE WITH ARTIFICIAL INTELLIGENCE

Under the guidance of

Mr AADHI SIVA VAIRAVAN. CT- Corporate Trainer

2025 - 2026

DECLARATION

We, **Alfi James, Nithyaa Sri. S and Jeshwanth. G,** hereby declare that this project report on "Event booking and ticket management platform using microservices" submitted to the University of Madras in partial fulfilment of the requirement for the award of the Degree Bachelor of Computer Science with Artificial Intelligence under the guidance of **Mrs S. CAROLIN JOSHIBA, HEAD OF DEPARTMENT** and **Mr AADHI SIVA VAIRAVAN. CT** has not been submitted earlier to any other university or institute for the award of any degree.

ALFI JAMES

NITHYAA SRI. S

JESHWANTH. G

Place:

Date:

JEPPIAAR
COLLEGE OF ARTS AND SCIENCE

IBM

BONAFIDE CERTIFICATE

This is to certify that the project titled "Event booking and ticket management platform" is the bonafide work done by, **Alfi James, Nithyaa Sri. S and Jeshwanth. G** hereby and second-year student of Jeppiaar College of Arts and Science, Padur, Chennai in partial fulfillment of the requirement for the award of the Degree of Bachelor of Computer Science with Artificial Intelligence 2023-2024.

PROJECT GUIDE:

Mr AADHI SIVA VAIRAVAN. CT

HEAD OF THE DEPARTMENT:

Mrs S. CAROLIN JOSHIBA

Date:

1. ABSTRACTION:

This project is developed to simplify the process of event booking and ticket management through a user-friendly web application. It allows users to view available events, book tickets, and receive booking confirmations, while administrators can manage events, schedules, and ticket availability. The system helps reduce manual work, avoid booking conflicts, and improve the overall event management experience. It ensures secure data handling and efficient booking operations. The application is built using Node.js for backend API development, Express.js for server management, MongoDB for database storage, and HTML, CSS, and JavaScript for the frontend interface.

2. PROBLEM STATEMENT:

Many event organizers and users face difficulties in managing event registrations and ticket bookings using manual or unorganized systems. Traditional booking methods often lead to overbooking, data errors, delayed confirmations, and lack of proper record maintenance. Users also struggle to get real-time information about event availability and booking status. Therefore, a centralized event booking and ticket management system is required to efficiently manage events, automate ticket reservations, track bookings, and ensure secure access for both users and administrators.

3. OBJECTIVES:

- To design and develop a centralized event booking and ticket management system.
- To provide efficient user and administrator role management with secure access control.
- To store and manage event details, user information, and booking data efficiently using a database system.
- To automate the event registration and ticket booking process to reduce manual effort and errors.
- To enable real-time tracking of ticket availability and booking status.
- To generate booking reports and event summaries for administrative analysis and decision-making.

4. SCOPE

The scope of the Event Booking and Ticket Management Platform defines the functional boundaries and features supported by the system. The project covers the following aspects:

• User Management

The system supports user registration and login functionality. Users can create accounts, authenticate themselves, and access event booking features securely. User details are stored and managed efficiently in the database.

- Data Storage

All system data, including user information, event details, ticket bookings, and booking history, are stored in a centralized MongoDB database. This ensures data persistence, reliability, and easy retrieval.

- CRUD Operations

The platform supports Create, Read, Update, and Delete (CRUD) operations:

a) Create new user accounts and events
b) Read event listings and booking history
c) Update user and event information (if required)
d) Delete or manage outdated records

• Authentication (if included)

Authentication is implemented to ensure that only registered users can book tickets and view their booking history. Users must log in before accessing protected features such as ticket booking.

• Validation

Input validation is applied to ensure that user-provided data such as email, password, and booking details are correct and complete. This prevents invalid or duplicate entries in the system.

• Logging / Alerts (Optional)

The system provides alerts and notifications for important actions such as successful registration, login, ticket booking confirmation, and error messages. These alerts enhance user experience and system transparency.

5. SOFTWARE REQUIREMENTS

Frontend (optional)

- HTML / CSS / JavaScript

Backend

- Node.js
- Express JS

Database

- MongoDB
- MongoDB atlas

Tools

- Notepad
- GitHub

6. SYSTEM ARCHITECTURE:

Client Layer (Frontend)

The client layer is developed using **HTML, CSS, and JavaScript**.
It provides user interfaces for registration, login, event listing, ticket booking, and booking history.

Methods Used in Client Layer:

- fetch() method for sending HTTP requests to backend APIs
- Form submission handling using JavaScript
- Client-side input validation
- Dynamic DOM manipulation to display events, QR codes, and booking history
- Alert-based user notifications

Application Layer (Backend – Node.js & Express)

The backend layer consists of **multiple microservices**, each responsible for a specific function such as authentication, event management, and booking management.

Methods Used in Backend Layer:

- RESTful API methods (POST, GET)
- Express.js routing methods (app.post(), app.get())
- Middleware usage (express.json(), cors())
- Authentication validation (login verification)
- QR code generation logic after successful booking
- Error handling and response management

Data Layer (MongoDB Database)

The data layer uses **MongoDB** to store all application data in collections such as users, events, and bookings.

Methods Used in Data Layer:

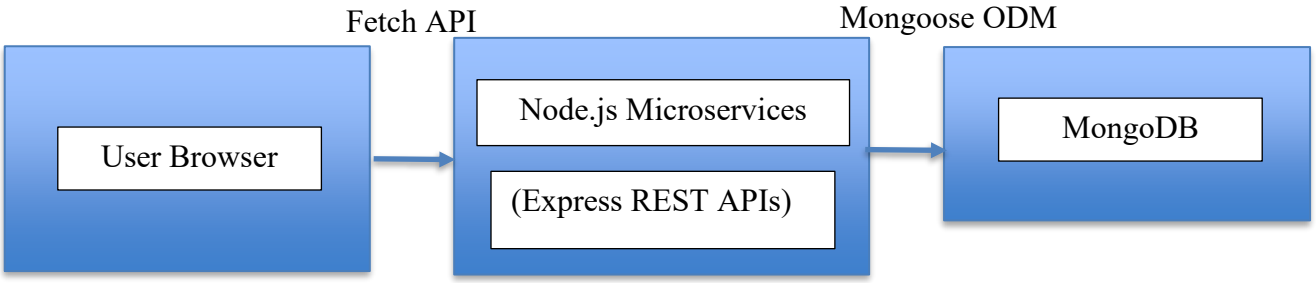
- Mongoose schema definitions
- findOne() to verify existing users
- save() to insert new records
- find() to retrieve events and booking history
- Database-level uniqueness validation (email-based)

Microservice Communication

Each microservice runs independently on a separate port and communicates with the frontend through HTTP requests.

Communication Methods Used:

- JSON-based data exchange
- Asynchronous request handling
- Service-specific APIs for authentication, events, and bookings



7. Folder Structure (Recommended)

```
event-booking-microservices/
├── auth-service/
│   ├── index.js
│   ├── db.js
│   └── User.js
├── event-service/
│   ├── index.js
│   ├── db.js
│   └── Event.js
├── booking-service/
│   ├── index.js
│   ├── db.js
│   └── Booking.js
├── ticket-service/
│   └── index.js
└── frontend/
    ├── register.html
    ├── login.html
    ├── events.html
    ├──book.html
    ├──history.html
    ├──index.html
    ├──style.css
```

8. DATABASE DESIGN (MONGODB SCHEMA)

The Event Booking and Ticket Management Platform uses **MongoDB** as a NoSQL database. Data is stored in collections using **Mongoose schemas** to define the structure of documents.

User Schema

This schema stores registered user details required for authentication and booking access.

Fields:

- name – Name of the user
- email – User email address (unique)
- password – User login password
- createdAt – Date and time of user registration

Event Schema

This schema stores event details created by the event organizer.

Fields:

- title – Event name
- description – Event details
- date – Event date
- location – Event venue
- price – Ticket price
- createdAt – Event creation timestamp

Booking Schema

This schema stores ticket booking details for users.

Fields:

- email – Email of the user who booked the ticket
- eventId – Reference to the booked event
- eventTitle – Name of booked event
- qr – QR code data (base64 or URL)
- date – Booking date and time

9. API Endpoints

The Event Booking and Ticket Management Platform exposes RESTful API endpoints to handle user authentication, event management, ticket booking, and booking history.

Authentication Service APIs

Method	Endpoint	Description
POST	/register	Register a new user
POST	/login	Authenticate user login

Event Management Service APIs

Method	Endpoint	Description
GET	/events	Retrieve all available events
POST	/events	Create a new event (organizer)

Ticket Booking Service APIs

Method	Endpoint	Description
POST	/book	Book a ticket for an event
GET	/history/:email	Retrieve booking history for a user

10. IMPLEMENTATION (SAMPLE CODE):

STEP 1: MongoDB CONNECTION (COMMON)

#db.js (same in all services)

```
const mongoose = require("mongoose");
mongoose.connect("mongodb+srv://purplenithu10_db_user:tAudbDNS3uqubUkM
@cluster1.uvqi7cd.mongodb.net/?appName=Cluster1")
.then(() => console.log("MongoDB Connected"))
.catch(err => console.log(err));
```

STEP 2: AUTH SERVICE (REGISTER + LOGIN)

#Auth-service\user.js

```
const mongoose = require("mongoose");
const UserSchema = new mongoose.Schema({
name: String,
email: { type: String, unique: true },
password: String
});
module.exports = mongoose.model("User", UserSchema);
```

#Auth-service\Index.js

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const User = require("./User");
const app = express();
app.use(express.json());
app.use(cors());
require("./db");
// Register
app.post("/register", async (req, res) => {
const { name, email, password } = req.body;
```

```
try {
// check if user already exists
const existingUser = await User.findOne({ email: email });
if (existingUser) {
return res.json({ success: false, message: "User already exists" });
}
// create new user
const newUser = new User({name,email,password});
await newUser.save();
res.json({ success: true });}
catch (err) {console.error(err);
res.status(500).json({ success: false });}});
// Login (Keep This)
app.post("/login", async (req, res) => {const user = await User.findOne(req.body);
res.json({ success: !!user });});
app.listen(5000, () => {console.log("Auth Service running on 5000");});
```

STEP 3: EVENT SERVICE (CREATE + LIST EVENTS)

#event-service\event.js

```
const mongoose = require("mongoose");
const eventSchema = new mongoose.Schema({
  title: String,
  date: String,
  location: String,
  price: Number
});
module.exports = mongoose.model("Event", eventSchema);
```

#event-service\index.js

```
console.log("INDEX FILE LOADED");
const express = require("express");
const cors = require("cors");
require("./db");
const Event = require("./Event")
```

```
const app = express();
app.use(cors());
app.use(express.json());
app.get("/", (req, res) => {res.send("Event Service is running");});
app.get("/events", async (req, res) => {const events = await Event.find();
res.json(events);});
async function insertSampleEvents()
{const events = await Event.find();
if (events.length === 0) {
await Event.insertMany([
{title: "Tech Conference 2025",date: "2025-03-15",location: "Chennai",
  price: 500},
{title: "Music Concert Night",date: "2025-04-10",location: "Bangalore",
  price: 800}]);
console.log("Sample events inserted");}}
insertSampleEvents();
app.listen(5001,"0.0.0.0", () => {console.log("Event Service running on 5001");
});
```

STEP 4: BOOKING SERVICE (BOOK + HISTORY)

#booking-service\booking.js

```
const mongoose = require("mongoose");
const bookingSchema = new mongoose.Schema({
email: String,
  eventTitle: String,
  qr: String,
  date: { type: Date, default: Date.now }
});
module.exports = mongoose.model("Booking", bookingSchema);
```

#booking-service\index.js

```
const express = require("express");
const cors = require("cors");
const QRCode = require("qrcode");
```

```
require("./db");
const Booking = require("./Booking");
const app = express();
app.use(cors());
app.use(express.json());
// Book Ticket + Generate QR
app.post("/book", async (req, res) => {const { email, eventTitle } = req.body;
// Generate QR
const qr = await QRCode.toDataURL(`Event: ${eventTitle}\nEmail: ${email}`);
// Save booking
await Booking.create({email, eventTitle, qr });
res.json({ success: true });});
// Booking history
app.get("/history/:email", async (req, res) => {
  const bookings = await Booking.find({ email: req.params.email });
  res.json(bookings);});
app.listen(5002, () => {console.log("Booking Service running on 5002");});
```

STEP 5: TICKET SERVICE (QR CODE)

#ticket-service\index.js

```
const express = require("express");
const QRCode = require("qrcode");
const app = express();
app.use(express.json());
app.post("/qr", async (req, res) => {
  const qr = await QRCode.toDataURL(JSON.stringify(req.body));
  res.json({ qr });
});
app.listen(5003, () => console.log("Ticket Service 5003"));
```

STEP 6: FRONTEND (REGISTER → MONGODB)

#index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Event Booking System</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
<h2 class="center">🎟️ Event Booking & Ticket Management</h2>
<div class="menu">
  <a href="register.html">📝 Register</a>
  <a href="login.html">🔐 Login</a>
  <a href="events.html">🎉 View Events</a>
  <a href="history.html">📜 Booking History</a>
</div>
<p class="center" style="margin-top:40px;color:white;">Microservices Based Web Application</p>
</body>
</html>
```

#register.html

```
<html>
<head>
  <title>Register</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
<div class="container">
<h2>User Registration</h2>
<input id="name" placeholder="Full Name">
<input id="email" placeholder="Email">
<input id="password" type="password" placeholder="Password">
```

```
<button onclick="register()">Register</button>
<p class="center">Already registered? <a href="login.html">Login</a></p>
</div>
<div class="navbar">
<h2>🎟 Event Booking & Ticket Management</h2>
<div class="menu">
<a href="events.html">Events</a>
<a href="history.html">My Bookings</a>
<a href="#" onclick="logout()">Logout</a>
</div></div>
<script>
function register(){
 fetch("http://localhost:5000/register",{
  method:"POST",
  headers:{ "Content-Type":"application/json" },
  body:JSON.stringify({
    name: name.value,
    email: email.value,
    password: password.value}) })
 .then(r=>r.json())
 .then(d=>{
  if(d.success){
    alert("Registration successful");
    window.location="login.html";}
else {
    alert("User already exists");
  }});}
function logout(){
  localStorage.clear();
  window.location="login.html";}
</script>
</body>
</html>
```

#login.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Login</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
<div class="container">
  <h2>Login</h2>
  <input id="email" placeholder="Email">
  <input id="password" type="password" placeholder="Password">
  <button onclick="login()">Login</button>
  <p class="center">New user?<a href="register.html">Register</a></p>
</div>
<script>
function login(){
 fetch("http://localhost:5000/login",{
  method:"POST",
  headers:{ "Content-Type":"application/json" },
  body:JSON.stringify({
    email: email.value,
    password: password.value}) })
 .then(r=>r.json())
 .then(d=>{
  if(d.success){
    localStorage.setItem("email", email.value);
    window.location="events.html";
  } else {
    alert("Invalid credentials");}});}

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

#events.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>🎟️ Event Booking & Ticket Management</title>
<style>
/* ===== GLOBAL STYLE (SAME FEEL AS LOGIN PAGE) ===== */
body{margin:0;
  font-family: "Segoe UI", Arial, sans-serif;
  background: linear-gradient(135deg, #141e30, #243b55);
  min-height:100vh;
  color:#333;}
/* ===== NAVBAR ===== */
.navbar{
  background:#0f172a;
  color:white;
  padding:15px 25px;
  display:flex;
  justify-content:space-between;
  align-items:center;
  box-shadow:0 4px 10px rgba(0,0,0,0.4);
}
.navbar h2{
  margin:0;
  font-size:22px;
}
.navbar a{
  color:white;
  text-decoration:none;
  margin-left:20px;
  font-size:14px;
}
```

```
.navbar a:hover{
  text-decoration:underline;
}
/* ===== MAIN CONTENT ===== */
.container{
  padding:30px;
  display:flex;
  flex-wrap:wrap;
  justify-content:center;
}
/* ===== EVENT CARD ===== */
.card{
  background:white;
  width:280px;
  padding:18px;
  margin:15px;
  border-radius:12px;
  box-shadow:0 10px 25px rgba(0,0,0,0.25);
  transition:transform 0.3s;
}
.card:hover{
  transform:translateY(-5px);
}
.card h3{
  margin-top:0;
  color:#0f172a;
}
.card p{
  margin:6px 0;
  font-size:14px;
  color:#555;
}
```

```
/* ===== BUTTON ===== */
button{
  width:100%;
  margin-top:10px;
  padding:10px;
  border:none;
  border-radius:6px;
  background:#0ea5e9;
  color:white;
  font-size:15px;
  cursor:pointer;
}
button:hover{
  background:#0284c7;
}
/* ===== FOOTER MESSAGE ===== */
.footer{
  text-align:center;
  color:#cbd5e1;
  font-size:13px;
  padding-bottom:20px;
}
</style>
</head>
<body>
<!-- LOGIN CHECK -->
<script>
if(!localStorage.getItem("email")){
  alert("Please login first");
  window.location="login.html";
}
</script>
```

```
<!-- NAVBAR -->
<div class="navbar">
  <h2>🎟 Event Booking & Ticket Management</h2>
  <div>
    <a href="events.html">Events</a>
    <a href="history.html">My Bookings</a>
    <a href="#" onclick="logout()">Logout</a>
  </div>
</div>
<!-- EVENTS LIST -->
<div class="container" id="events">
  <!-- Events will load here -->
</div>
<div class="footer">
  © 2025 Event Booking System
</div>
<script>
const email = localStorage.getItem("email");
/* LOAD EVENTS */
fetch("http://127.0.0.1:5001/events")
.then(res => res.json())
.then(events => {
  let html = "";
  events.forEach(e => {
    html += `
      <div class="card">
        <h3>${e.title}</h3>
        <p><b>Date:</b> ${e.date}</p>
        <p><b>Location:</b> ${e.location}</p>
        <p><b>Price:</b> ₹${e.price}</p>
        <button onclick="bookTicket('${e.title}')">Book Ticket</button>
      </div>
    `; });
```

```
  document.getElementById("events").innerHTML = html;
})
.catch(() => {
  document.getElementById("events").innerHTML = "<p
style='color:white'>Cannot load events</p>";
});
/* BOOK TICKET */
function bookTicket(title){
  fetch("http://127.0.0.1:5002/book",{
    method:"POST",
    headers:{ "Content-Type":"application/json" },
    body:JSON.stringify({
      email: email,
      eventTitle: title    })  })
  .then(res => res.json())
  .then(() => {
    alert("🎉 Ticket booked successfully");
    window.location = "history.html";  });}
/* LOGOUT */
function logout(){
  localStorage.clear();
  window.location = "login.html";
}
</script>
</body>
</html>
```

#book.html

```
<html>
<head>
  <title>Book Ticket</title>
  <link rel="stylesheet" href="style.css">
</head>
```

```
<body>
<div class="container">
  <h2>Confirm Booking</h2>
  <button onclick="confirm()">Confirm Ticket</button>
</div>
<script>
function confirm(){
 fetch("http://localhost:5002/book",{
  method:"POST",
  headers:{ "Content-Type":"application/json" },
  body:JSON.stringify({
    email: localStorage.getItem("email"),
    eventId: localStorage.getItem("eventId")  }) })
 .then(r=>r.json())
 .then(d=>{
  if(d.success){
    alert("Ticket booked!");
    window.location="history.html";  } });}
</script>
</body>
</html>
```

#history.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>🎟️ Event Booking & Ticket Management</title>
<style>
/* RESET */
*{
  margin:0;
  padding:0;
```

```
  box-sizing:border-box;
  font-family:Arial, Helvetica, sans-serif;
}
/* BODY */
body{
  background: linear-gradient(rgba(0,0,0,.6),rgba(0,0,0,.6)),
  url("https://images.unsplash.com/photo-1515165562835-c4c7b0b1b6c4");
  background-size:cover;
  background-attachment:fixed;
  color:white;
  min-height:100vh;}
/* NAVBAR */
.navbar{
  display:flex;
  justify-content:space-between;
  align-items:center;
  padding:15px 40px;
  background:rgba(0,0,0,0.85);}
.navbar h2{
  font-size:22px;}
.navbar ul{
  list-style:none;
  display:flex;
  gap:25px;
}
.navbar a{
  text-decoration:none;
  color:white;
  font-weight:bold;
}
.navbar a:hover{
  color:#00ffd5;
}
```

```
/* PAGE TITLE */
.title{
  text-align:center;
  margin:30px 0;
  font-size:28px;
}

/* HISTORY CONTAINER */
.history-container{
  display:flex;
  flex-wrap:wrap;
  justify-content:center;
  gap:25px;
  padding-bottom:40px;
}
/* BOOKING CARD */
.card{
  background:rgba(255,255,255,0.95);
  color:#000;
  width:300px;
  padding:20px;
  border-radius:15px;
  box-shadow:0 10px 25px rgba(0,0,0,0.4);
  text-align:center;
  transition:0.3s;
}
.card:hover{
  transform:scale(1.05);
}
/* QR IMAGE */
.card img{
  margin:15px 0;
  width:160px;
}
```

```
/* DATE TEXT */
.card p{
  font-size:14px;
  color:#444;
}
/* NO BOOKINGS */
.empty{
  text-align:center;
  font-size:20px;
  opacity:0.9;
}
</style>
</head>
<body>
<!-- NAVBAR -->
<div class="navbar">
  <h2>🎟 Event Booking & Ticket Management</h2>
  <ul>
    <li><a href="events.html">Events</a></li>
    <li><a href="history.html">My Bookings</a></li>
    <li><a href="login.html" onclick="logout()">Logout</a></li>
  </ul>
</div>
<!-- TITLE -->
<div class="title">My Booking History</div>
<!-- BOOKINGS -->
<div class="history-container" id="history"></div>
<script>
function logout(){
  localStorage.clear();}
const email = localStorage.getItem("email");
if(!email){
  alert("Please login first");
```

```
  window.location="login.html";}
fetch("http://127.0.0.1:5002/history/" + email)
.then(res => res.json())
.then(data => {
  let html = "";
  if(data.length === 0){
    html = "<div class='empty'>No bookings yet</div>";  }
  data.forEach(b => {
    html += `
      <div class="card">
        <h3>${b.eventTitle}</h3>
        <img src="${b.qr}">
        <p>Booked on:<br>${new Date(b.date).toLocaleString()}</p>
      </div>    `;  });
document.getElementById("history").innerHTML = html;});
</script>
</body>
</html>
```

#style.css

```
* {
  box-sizing: border-box;
  font-family: Arial, Helvetica, sans-serif;
}
body {
  background: linear-gradient(120deg, #667eea, #764ba2);
  margin: 0;
  padding: 0;
}
.container {
  width: 350px;
  margin: 80px auto;
  background: #fff;
  padding: 25px;
```

```
  border-radius: 10px;
  box-shadow: 0 10px 25px rgba(0,0,0,0.2);
}
h2 {
  text-align: center;
  margin-bottom: 20px;
  color: black;
}
input {
  width: 100%;
  padding: 10px;
  margin: 8px 0;
  border-radius: 6px;
  border: 1px solid #ccc;
  font-size: 14px;
}
button {
  width: 100%;
  padding: 10px;
  background: #667eea;
  color: #fff;
  border: none;
  border-radius: 6px;
  font-size: 16px;
  cursor: pointer;
  margin-top: 10px;
}
button:hover {
  background: #5a67d8;
}
.event {
  background: #fff;
  margin: 15px;
  padding: 15px;
```

```
  border-radius: 10px;
  box-shadow: 0 5px 15px rgba(0,0,0,0.1);
}
.event h3 {
  margin: 0;
  color: #333;
}
.event p {
  margin: 5px 0;
}
.nav {
  background: #333;
  padding: 10px;
  color: white;
  text-align: center;
}
.nav a {
  color: white;
  margin: 0 10px;
  text-decoration: none;
}
.nav a:hover {
  text-decoration: underline;
}
.center {
  text-align: center;
}
.menu {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
  gap: 20px;
  margin-top: 40px;
}
```

```
.menu a {
  width: 220px;
  text-align: center;
  padding: 20px;
  background: white;
  color: #333;
  text-decoration: none;
  font-size: 18px;
  border-radius: 12px;
  box-shadow: 0 8px 20px rgba(0,0,0,0.15);
  transition: 0.3s;
}
.menu a:hover {
  transform: scale(1.05);
  background: #667eea;
  color: white;
}
```

11. OUTPUT SCREENSHOTS

Creating folders:

```
C:\Users\purpl>node -v
v24.12.0

C:\Users\purpl>mkdir event-booking

C:\Users\purpl>cd event-booking

C:\Users\purpl\event-booking>mkdir auth-service

C:\Users\purpl\event-booking>mkdir event-service

C:\Users\purpl\event-booking>mkdir booking-service

C:\Users\purpl\event-booking>mkdir ticket-service

C:\Users\purpl\event-booking>mkdir frontend
```

Installing required packages according to their service:

```
C:\Users\purpl\event-booking\event-service>cd ../booking-service

C:\Users\purpl\event-booking\booking-service>npm init -y
Wrote to C:\Users\purpl\event-booking\booking-service\package.json:

{
  "name": "booking-service",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "Nithya",
  "license": "MIT",
  "type": "commonjs"
}



C:\Users\purpl\event-booking\booking-service>npm install express mongoose co
rs

added 84 packages, and audited 85 packages in 5s

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

found 0 vulnerabilities
```

```
C:\Users\purpl\event-booking\auth-service>cd..

C:\Users\purpl\event-booking>cd event-service

C:\Users\purpl\event-booking\event-service>npm init -y
Wrote to C:\Users\purpl\event-booking\event-service\package.json:

{
  "name": "event-service",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "Nithya",
  "license": "MIT",
  "type": "commonjs"
}



C:\Users\purpl\event-booking\event-service>npm install express mongoose cors


added 84 packages, and audited 85 packages in 5s

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

found 0 vulnerabilities
```

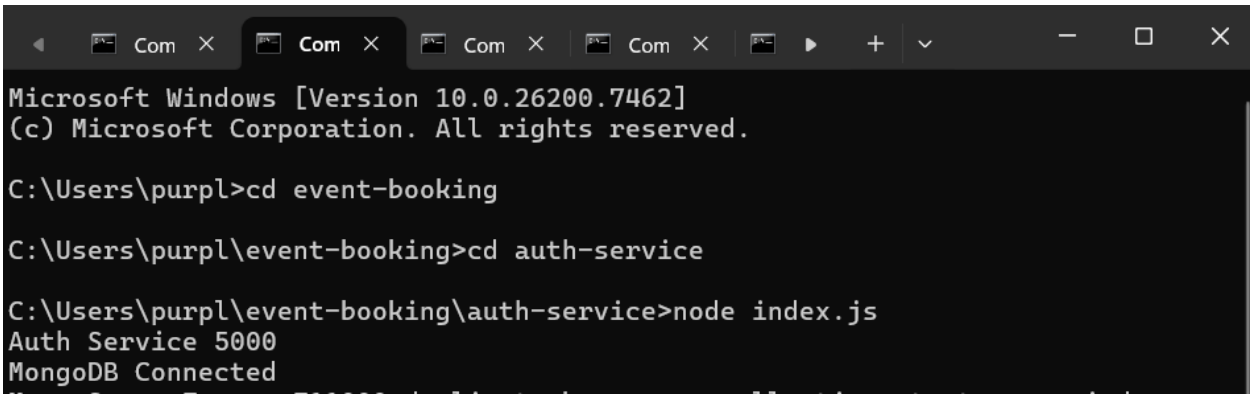
```
C:\Users\purpl\event-booking\booking-service>cd ../ticket-service

C:\Users\purpl\event-booking\ticket-service>npm init -y
Wrote to C:\Users\purpl\event-booking\ticket-service\package.json:

{
  "name": "ticket-service",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "Nithya",
  "license": "MIT",
  "type": "commonjs"
}



C:\Users\purpl\event-booking\ticket-service>npm install express qrcode

added 94 packages, and audited 95 packages in 5s

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

found 0 vulnerabilities
```

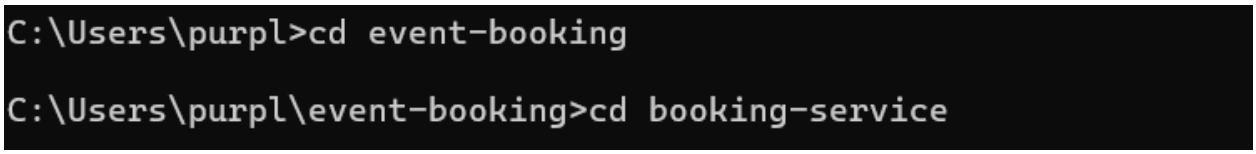
```
C:\Users\purpl\event-booking>cd auth-service

C:\Users\purpl\event-booking\auth-service>npm init -y
Wrote to C:\Users\purpl\event-booking\auth-service\package.json:

{
  "name": "auth-service",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "Nithya",
  "license": "MIT",
  "type": "commonjs"
}



C:\Users\purpl\event-booking\auth-service>npm install express mongoose cors

added 84 packages, and audited 85 packages in 8s

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

found 0 vulnerabilities
```

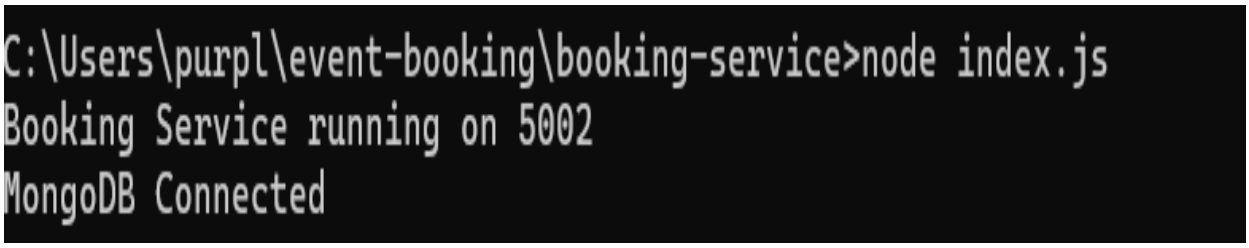
```
C:\Users\purpl\event-booking\booking-service>npm install express mongo
ose cors

added 84 packages, and audited 85 packages in 5s

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

found 0 vulnerabilities

C:\Users\purpl\event-booking\booking-service>npm install express mongo
ose cors axios

added 11 packages, and audited 96 packages in 3s

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

found 0 vulnerabilities
```

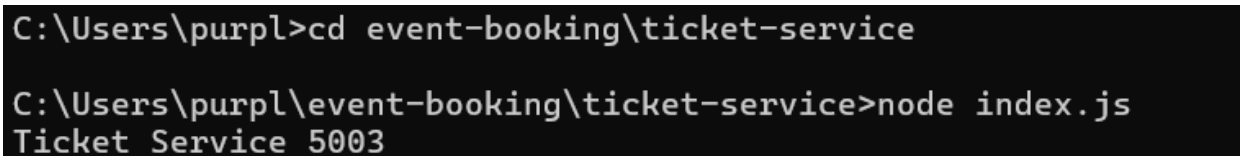
```
C:\Users\purpl\event-booking\booking-service>npm install qrcode

added 29 packages, and audited 125 packages in 4s

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

found 0 vulnerabilities
```

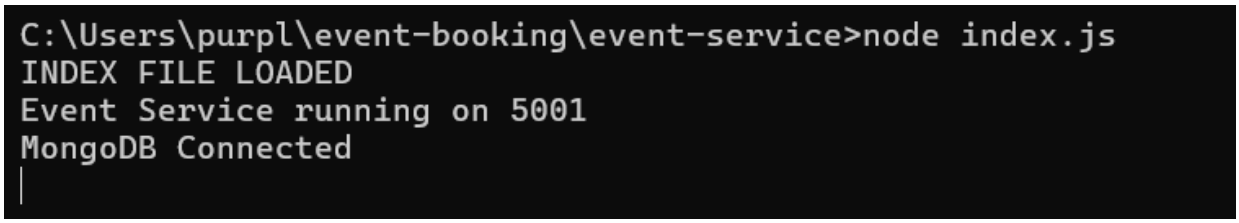
Connecting MongoDB:



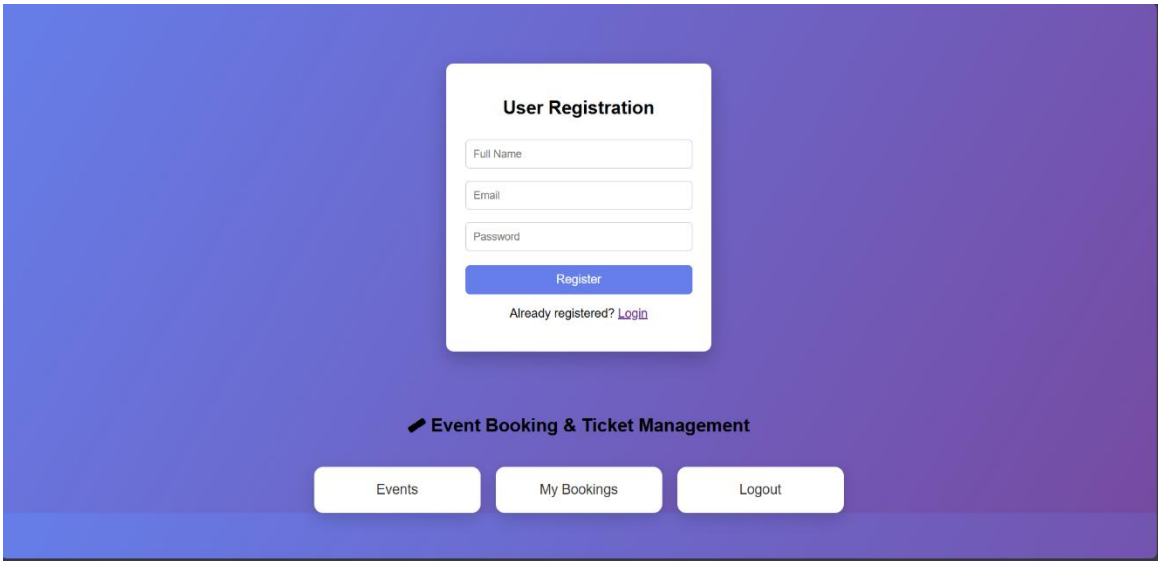


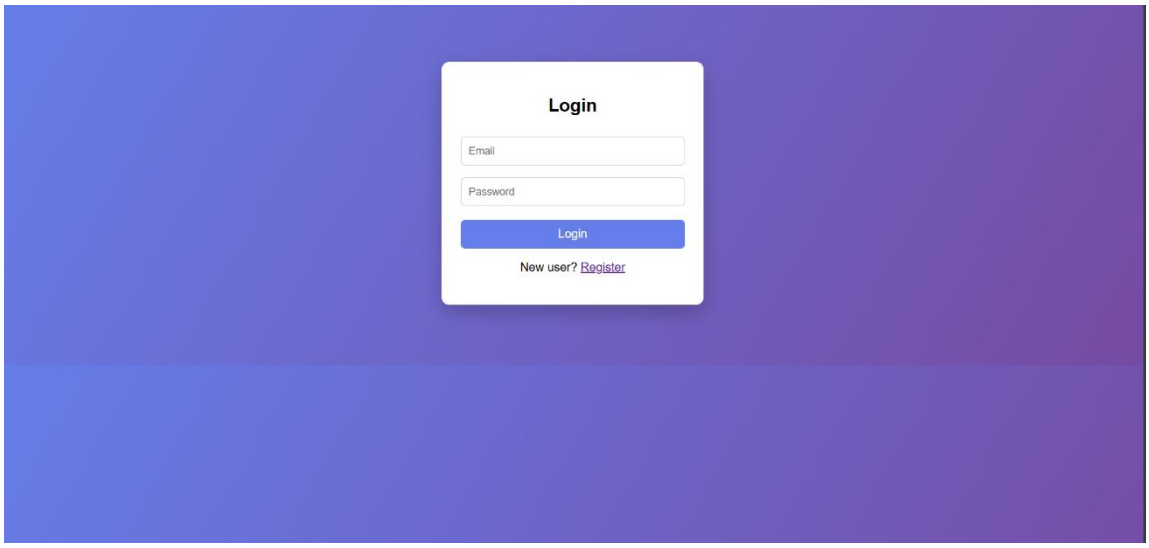


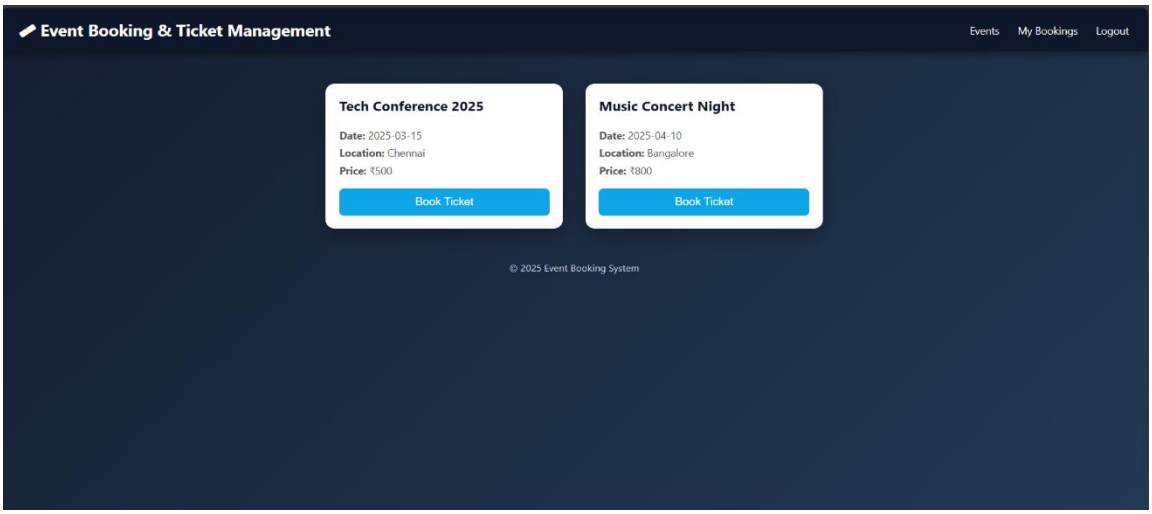


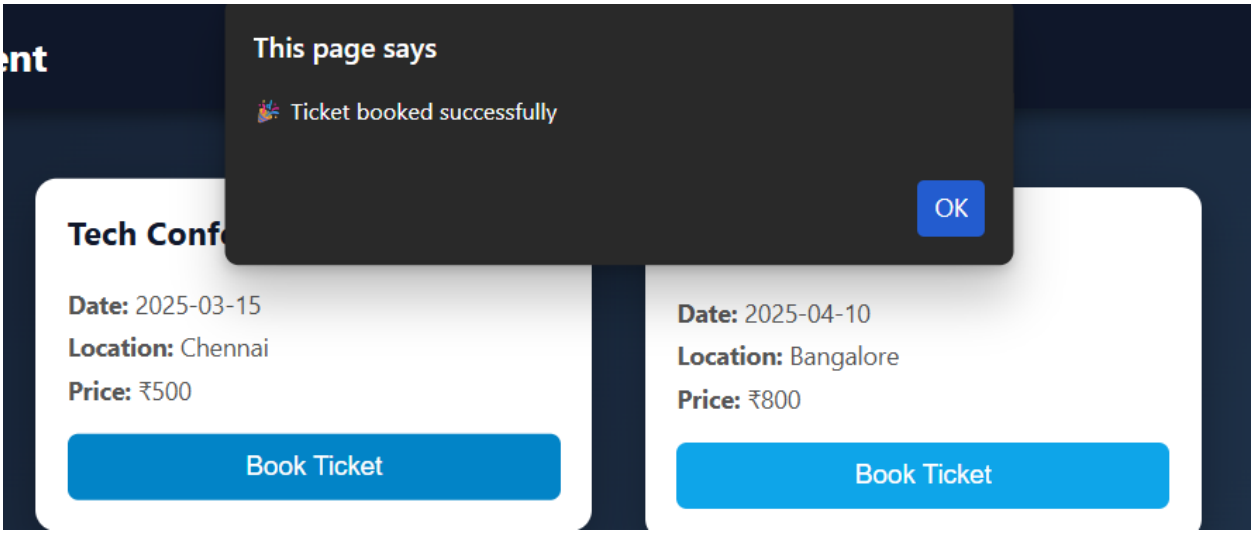


Frontend output:



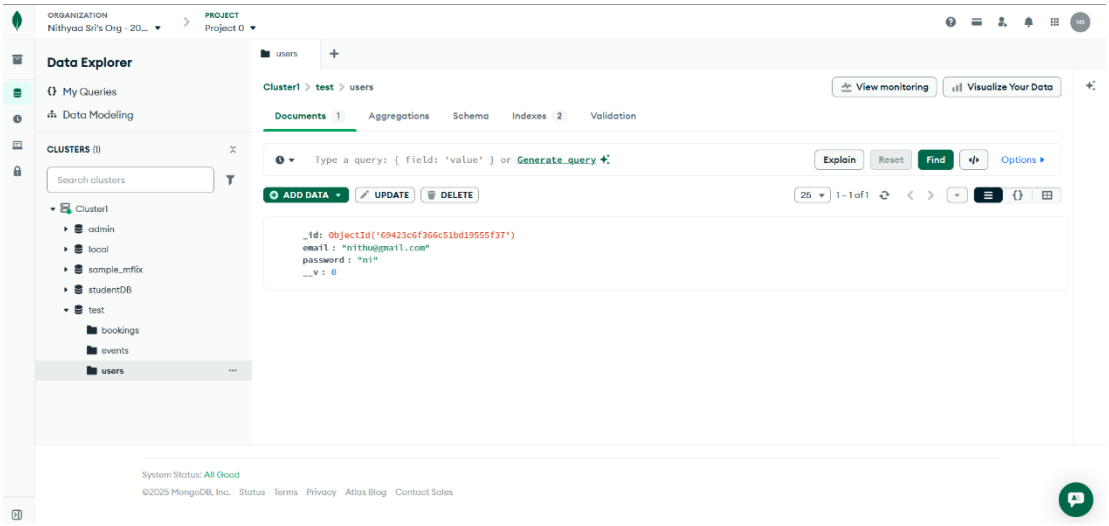


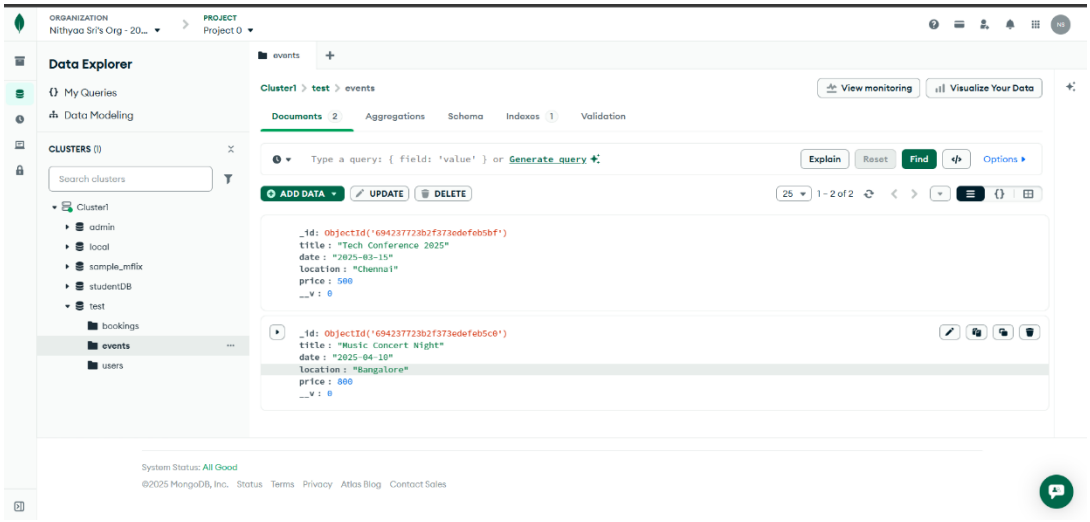


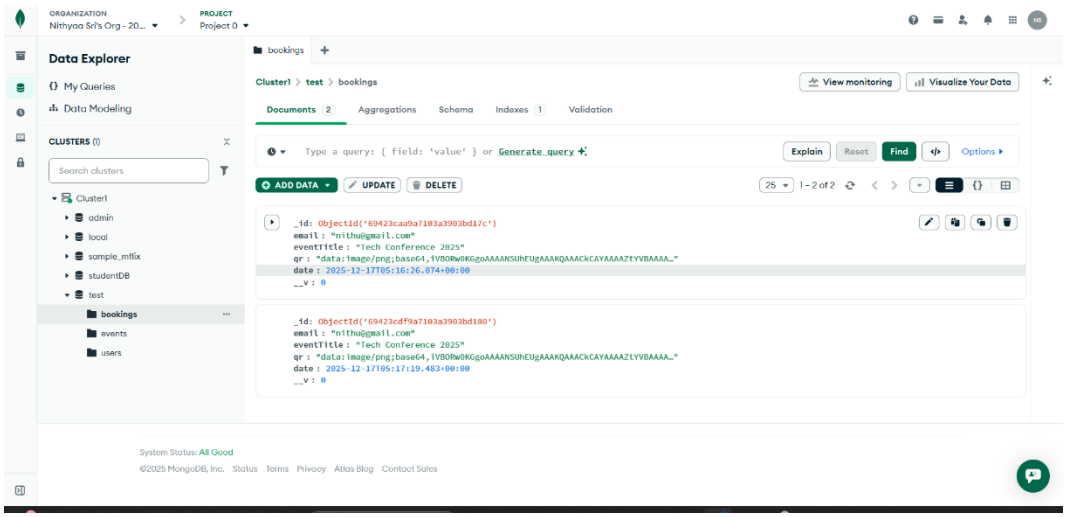

This page says
Ticket booked successfully
OK
Tech Conf
Date: 2025-03-15
Location: Chennai
Price: ₹500
Book Ticket
Date: 2025-04-10
Location: Bangalore
Price: ₹800
Book Ticket


Event Booking & Ticket Management
Events
My Bookings
Logout
My Booking History
Tech Conference 2025
Booked on:
17/12/2025, 10:46:26 am
Tech Conference 2025
Booked on:
17/12/2025, 10:47:19 am

Data stored in MongoDB Atlas:







12. FUTURE ENHANCEMENTS:

The **Event Booking and Ticket Management Platform** can be enhanced by adding **secure authentication** using JWT, an **admin panel** for managing users and events, and **charts and dashboards** for analyzing bookings. **Email and SMS alerts** can be integrated for notifications, and the system can be deployed on **cloud platforms** like AWS, Render, or Heroku for better scalability and availability.

13. Conclusion

The **Event Booking and Ticket Management Platform** provides an efficient solution for managing event registration, ticket booking, and booking history through a user-friendly web interface. Using **Node.js, Express, and MongoDB**, the system ensures reliable data storage and smooth communication between frontend and backend services. The project successfully fulfills the problem statement by simplifying event management and improving the overall user experience.

14. REFERENCE:

Git-Hub link: