CHAPTER 1

INTRODUCTION

Welcome to our AngularJS project on an Event Organization Platform, powered by Node.js and MongoDB. In today's fast-paced world, organizing events efficiently is essential for businesses, communities, and individuals alike. Our platform aims to streamline the event management process, providing users with a seamless experience from planning to execution.
Utilizing AngularJS, a powerful front-end framework, we've developed an intuitive user interface that offers rich functionality and responsiveness. With AngularJS, users can effortlessly navigate through the platform, manage events, and engage with attendees.
Backing our front-end is Node.js, a versatile server-side JavaScript runtime, enabling real-time communication and handling concurrent requests efficiently. Node.js ensures the scalability and performance of our platform, allowing it to adapt to varying loads and user interactions seamlessly.
Furthermore, MongoDB serves as our database solution, providing a flexible and scalable storage option for event-related data. Leveraging MongoDB's document-oriented architecture, we can store and retrieve event details, user information, and other relevant data with ease, ensuring a smooth and efficient user experience.
In summary, our AngularJS project on Event Organization Platform combines the power of AngularJS for the front-end, Node.js for the server-side logic, and MongoDB for data storage, offering users a comprehensive solution for organizing and managing events effectively. Whether you're planning a small gathering or a large-scale conference, our platform is designed to meet your event management needs with simplicity and efficiency.

1.1 Scope of the project

The scope of a project on an event organization platform using Node.js and MongoDB would typically involve several key components:

1. User Management: Allow users to register, login, and manage their profiles. This includes features like profile editing, password reset, and account deletion.

2. Event Creation and Management: Enable users to create, edit, and delete events. This involves defining event details such as date, time, location, description, and possibly ticketing information.

3. Event Discovery: Implement features for users to browse and search for events based on different criteria such as location, date, category, or keyword.

4. Booking and Ticketing: Allow users to RSVP or purchase tickets for events. Implement features for managing bookings, issuing tickets, and handling payments securely.

5. Social Interaction: Incorporate social features such as comments, likes, ratings, and reviews to encourage engagement among users and promote events.

6. Notifications: Implement notification mechanisms to keep users informed about event updates, new events matching their preferences, or interactions related to their events.

7. Admin Panel: Provide administrators with a dashboard to manage users, events, and content moderation. This includes features for monitoring user activity, handling reported content, and managing system settings.

8. Analytics and Reporting: Include tools for tracking key metrics such as user engagement, event attendance, and revenue generation. Generate reports to analyze trends and make informed decisions.

9. Integration with External Services: Integrate with third-party services for functionalities such as geolocation, payment processing, email notifications, and social media sharing.

10. Security: Implement security measures to protect user data, prevent unauthorized access, and secure transactions. This includes techniques like encryption, authentication, and authorization.

11. Scalability and Performance: Design the application architecture to handle a large number of users and events efficiently. Optimize database queries, cache frequently accessed data, and deploy on scalable infrastructure.

12. Localization and Accessibility: Support multiple languages and ensure the platform is accessible to users with disabilities by following accessibility guidelines.

13. Testing and Quality Assurance: Conduct thorough testing to identify and fix bugs, ensure compatibility across different devices and browsers, and validate the platform's functionality and performance.

14. Documentation and Support: Provide comprehensive documentation for developers and users, including installation instructions, API documentation, and user guides. Offer support channels for users to seek assistance and report issues.

This scope outlines the main features and considerations for developing an event organization platform using Node.js and MongoDB. Depending on specific requirements and constraints, additional features or modifications may be necessary.

CHAPTER 2

SYSTEM REQUIREMENTS

To deploy an event organization platform using Node.js and MongoDB, you'll need to consider both the software and hardware requirements:

2.1 Software Requirements:

1. Node.js: Install Node.js runtime environment on your server. Ensure compatibility with the Node.js version specified in your project dependencies.

2. MongoDB: Set up MongoDB database server. Install and configure MongoDB to store event data. Make sure you have enough disk space to accommodate your data requirements.

3. NPM (Node Package Manager): Use npm to manage Node.js packages and dependencies. Install required packages for your project, such as Express.js for web server, Mongoose for MongoDB interaction, and other libraries for additional functionalities.

4. Operating System: Choose an operating system compatible with Node.js and MongoDB. Popular choices include Linux distributions like Ubuntu, CentOS, or Debian.

5. Web Server: Optionally, set up a web server like Nginx or Apache to serve your Node.js application to the internet. Configure the web server to proxy requests to your Node.js application.

6. SSL Certificate: If your platform handles sensitive data or payments, consider installing an SSL certificate to encrypt data transmitted over the network.

2.2 Hardware Requirements:

1. Processor: Choose a server with a suitable processor to handle the expected workload. The processor should have enough processing power to handle incoming requests efficiently.

2. RAM: Allocate sufficient RAM to run your Node.js application and MongoDB database comfortably. The amount of RAM required depends on factors like the number of concurrent users, database size, and complexity of your application.

3. Storage: Ensure you have enough disk space to store your MongoDB data files and Node.js application files. Consider using SSD storage for better performance, especially if your application has high read/write requirements.

4. Network: Ensure reliable network connectivity with sufficient bandwidth to handle incoming requests and data transfer between clients and servers.

CHAPTER 3

DESIGN

Designing an event organization platform using Node.js and MongoDB involves planning the architecture, data model, and user interface. Here's a high-level design outline:

3.1 Architecture:
1. Client-Server Architecture: Implement a client-server architecture where the Node.js server serves as the backend for handling requests and processing data, while the client (browser or mobile app) interacts with the server to access and display information.
2. RESTful API: Design a RESTful API for communication between the client and server. Define endpoints for user authentication, event management, user profiles, and other functionalities.
3. Microservices (Optional): Consider breaking down the application into microservices to improve scalability, maintainability, and flexibility. Each microservice can handle a specific aspect of the platform, such as user management, event management, or notifications.
4. Scalability: Design the architecture to be horizontally scalable, allowing for easy addition of more resources or nodes to handle increased traffic and load.

3.2 Data Model:
1. User Data: Define a user schema to store user information such as username, email, password (hashed), profile picture, and preferences.
2. Event Data: Create a schema to represent events, including attributes like title, description, date, time, location, organizer, and ticket information.
3. Booking Data: Design a schema to manage user bookings for events, including references to the user and event IDs, booking status, and ticket details.
4. Comments and Reviews: If implementing social features, design schemas for storing user comments, ratings, and reviews associated with events.

3.3 User Interface:
1. Authentication: Design user interfaces for registration, login, and password recovery. Implement forms for users to enter their credentials and handle authentication securely.
2. Event Creation and Management: Create intuitive interfaces for users to create, edit, and delete events. Include forms for specifying event details and options for uploading images or documents.
3. Event Discovery: Design search and browse interfaces for users to discover events based on different criteria such as location, date, category, or keyword. Implement filters and sorting options to refine search results.
4. Booking and Ticketing: Develop interfaces for users to view event details, RSVP, and purchase tickets. Provide options for selecting ticket types, quantity, and payment methods.
5. User Profile: Design user profiles where users can view and edit their personal information, manage their events, view past bookings, and interact with other users.

6. Notifications: Implement notification interfaces for users to receive alerts about event updates, new events matching their preferences, or interactions related to their events.
7. Admin Dashboard: Create an admin dashboard with interfaces for managing users, events, bookings, and content moderation. Include features for monitoring system activity and generating reports.

3.4 Security:
1. Authentication and Authorization: Implement secure authentication mechanisms such as JWT (JSON Web Tokens) for user authentication and authorization. Ensure proper validation and protection against common security vulnerabilities like XSS (Cross-Site Scripting) and CSRF (Cross-Site Request Forgery).
2. Data Validation: Validate user input and enforce data validation rules to prevent injection attacks and ensure data integrity.
3. Encryption: Use encryption techniques to protect sensitive data such as passwords and payment information stored in the database.
4. Rate Limiting: Implement rate limiting to prevent abuse and protect against brute force attacks on authentication endpoints.
5. Session Management: Manage user sessions securely to prevent session hijacking and enforce session expiration policies.

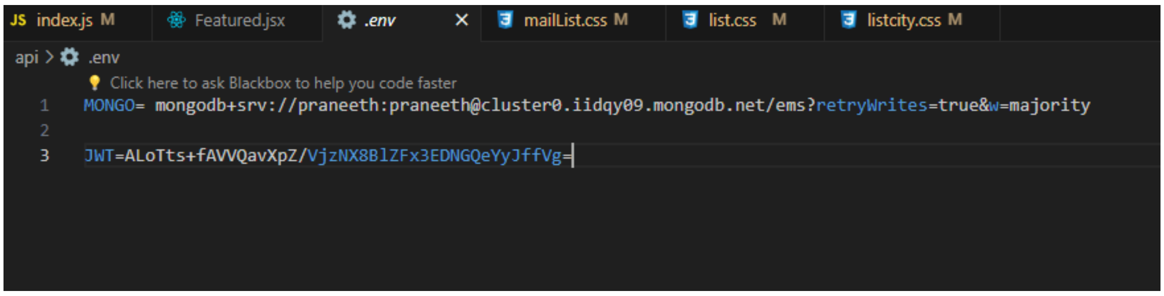
By following this design outline, you can develop a robust and user-friendly event organization platform using Node.js and MongoDB, meeting both functional and non-functional requirements.

CHAPTER 4

IMPLEMENTATION

This implementation covers basic user registration, login, event creation, and event listing functionalities using node.js, express, jwt for authentication, and bcrypt for password hashing. You'll need to set up a mongodb database and integrate it for data persistence. Additionally, consider adding error handling, input validation, and other security measures for a production-ready application.

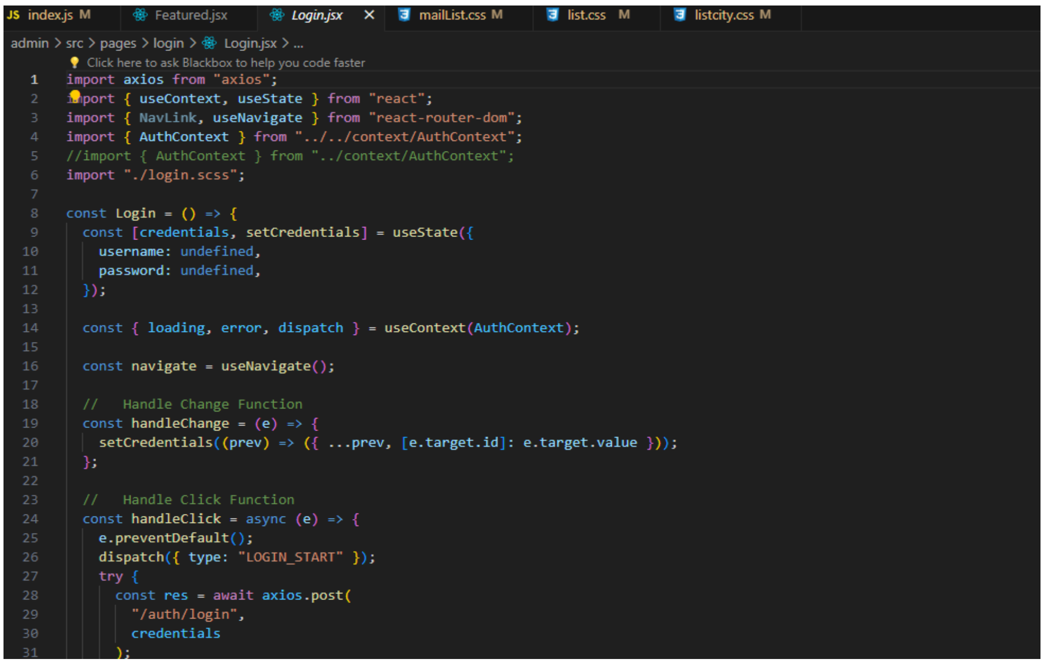
Mongodb :



```
MONGO= mongodb+srv://praneeth:praneeth@cluster0.iidqy09.mongodb.net/ems?retryWrites=true&w=majority

JWT=ALoTts+fAVVQavXpZ/VjzNX8BlZFx3EDNGQeYyJffVg=
```

CONNECTING TO MONGODB

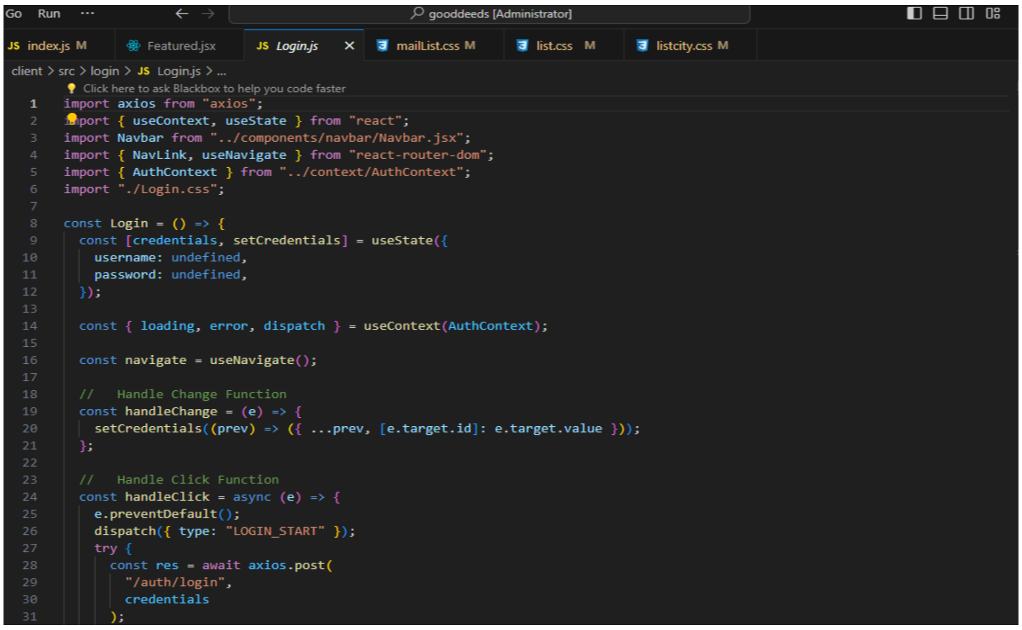
ADMIN LOGIN:



JS index.js M
Featured.jsx
Login.jsx
mailList.css M
list.css M
listcity.css M
admin > src > pages > login > Login.jsx > ...
Click here to ask Blackbox to help you code faster

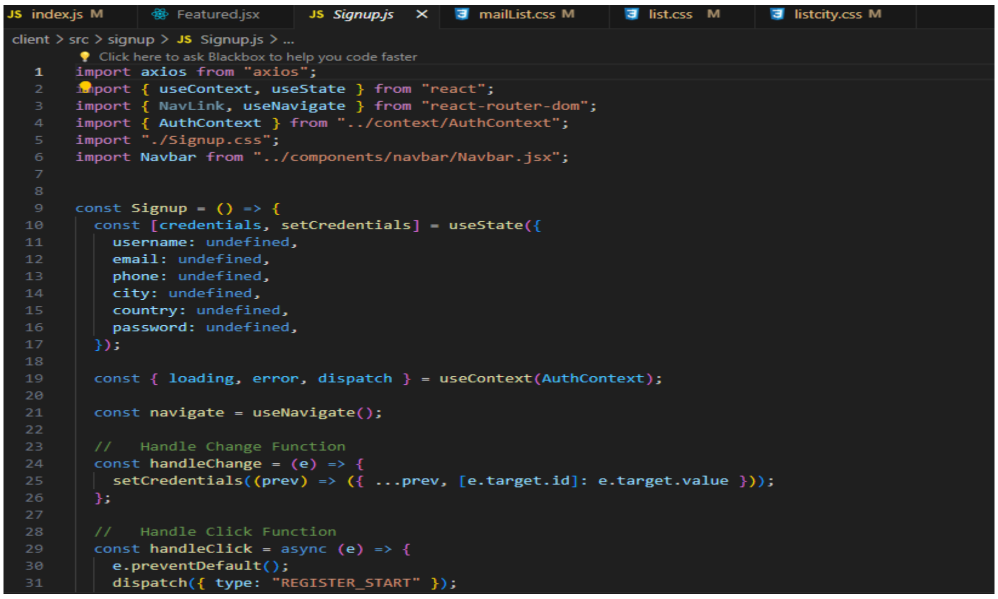
```
import axios from "axios";
import { useContext, useState } from "react";
import { NavLink, useNavigate } from "react-router-dom";
import { AuthContext } from "../../context/AuthContext";
//import { AuthContext } from "../context/AuthContext";
import "./login.scss";

const Login = () => {
  const [credentials, setCredentials] = useState({
    username: undefined,
    password: undefined,
  });

  const { loading, error, dispatch } = useContext(AuthContext);

  const navigate = useNavigate();

  //   Handle Change Function
  const handleChange = (e) => {
    setCredentials((prev) => ({ ...prev, [e.target.id]: e.target.value }));
  };

  //   Handle Click Function
  const handleClick = async (e) => {
    e.preventDefault();
    dispatch({ type: "LOGIN_START" });
    try {
      const res = await axios.post(
        "/auth/login",
        credentials
      );
```

Client Login:



```
import axios from "axios";
import { useContext, useState } from "react";
import Navbar from "../components/navbar/Navbar.jsx";
import { NavLink, useNavigate } from "react-router-dom";
import { AuthContext } from "../context/AuthContext";
import "./Login.css";

const Login = () => {
  const [credentials, setCredentials] = useState({
    username: undefined,
    password: undefined,
  });

  const { loading, error, dispatch } = useContext(AuthContext);

  const navigate = useNavigate();

  //   Handle Change Function
  const handleChange = (e) => {
    setCredentials((prev) => ({ ...prev, [e.target.id]: e.target.value }));
  };

  //   Handle Click Function
  const handleClick = async (e) => {
    e.preventDefault();
    dispatch({ type: "LOGIN_START" });
    try {
      const res = await axios.post(
        "/auth/login",
        credentials
      );
```

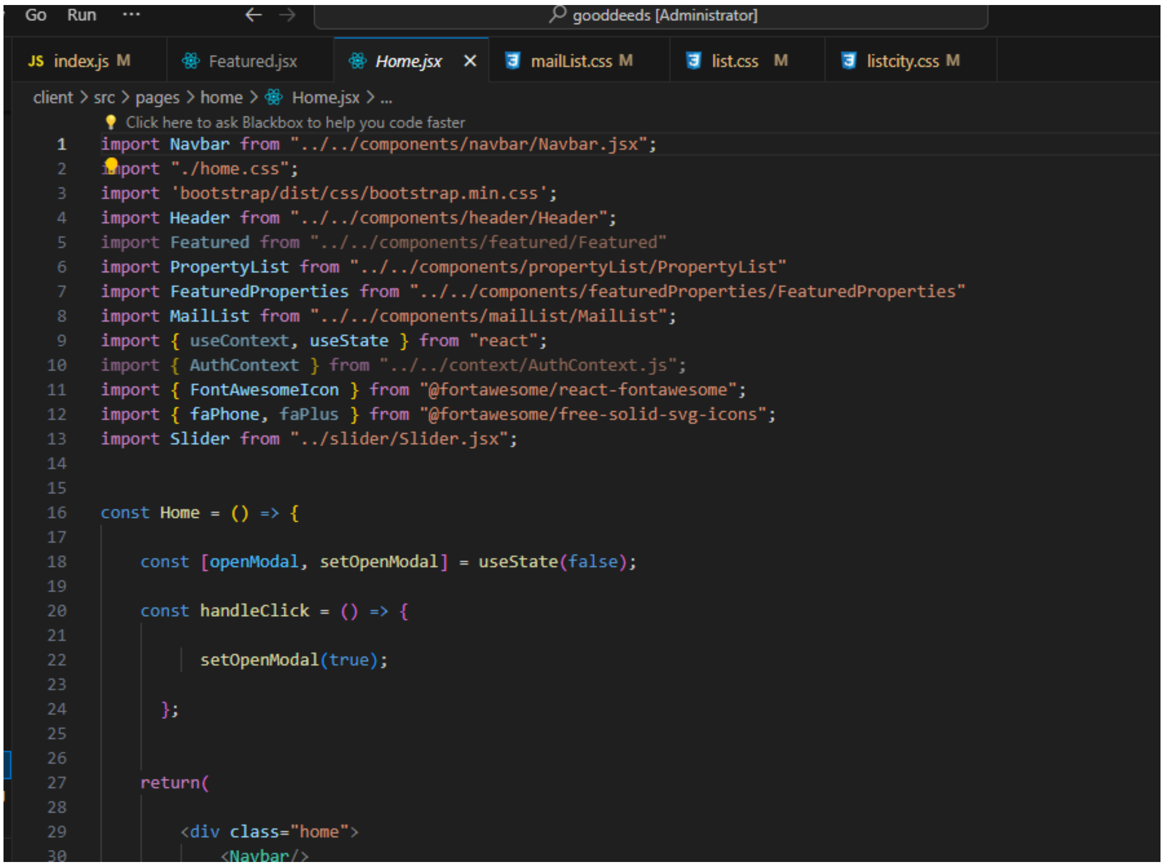
Client registration:



index.js M
Featured.jsx
Signup.js
mailList.css M
list.css M
listcity.css M
client > src > signup > Signup.js > ...
Click here to ask Blackbox to help you code faster

```
import axios from "axios";
import { useContext, useState } from "react";
import { NavLink, useNavigate } from "react-router-dom";
import { AuthContext } from "../context/AuthContext";
import "./Signup.css";
import Navbar from "../components/navbar/Navbar.jsx";


const Signup = () => {
  const [credentials, setCredentials] = useState({
    username: undefined,
    email: undefined,
    phone: undefined,
    city: undefined,
    country: undefined,
    password: undefined,
  });

  const { loading, error, dispatch } = useContext(AuthContext);

  const navigate = useNavigate();

  //   Handle Change Function
  const handleChange = (e) => {
    setCredentials((prev) => ({ ...prev, [e.target.id]: e.target.value }));
  };

  //   Handle Click Function
  const handleClick = async (e) => {
    e.preventDefault();
    dispatch({ type: "REGISTER_START" });
```

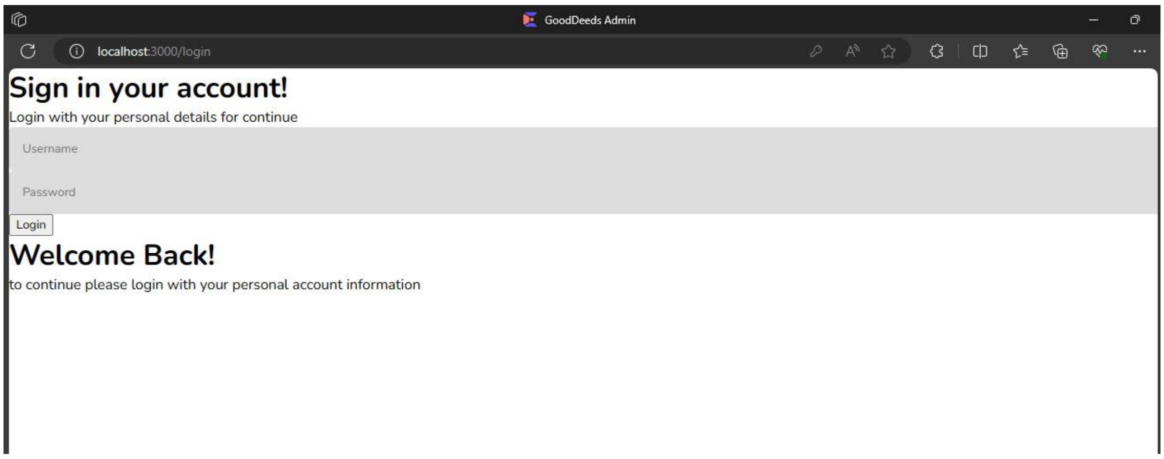
Home page:


Go Run ···
gooddeeds [Administrator]
index.js M
Featured.jsx
Home.jsx
mailList.css M
list.css M
listcity.css M
client > src > pages > home > Home.jsx > ...
Click here to ask Blackbox to help you code faster

```
import Navbar from "../../components/navbar/Navbar.jsx";
import "./home.css";
import 'bootstrap/dist/css/bootstrap.min.css';
import Header from "../../components/header/Header";
import Featured from "../../components/featured/Featured"
import PropertyList from "../../components/propertyList/PropertyList"
import FeaturedProperties from "../../components/featuredProperties/FeaturedProperties"
import MailList from "../../components/mailList/MailList";
import { useContext, useState } from "react";
import { AuthContext } from "../../context/AuthContext.js";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faPhone, faPlus } from "@fortawesome/free-solid-svg-icons";
import Slider from "../slider/Slider.jsx";


const Home = () => {

    const [openModal, setOpenModal] = useState(false);

    const handleClick = () => {

        setOpenModal(true);

      };


    return(

        <div class="home">
            <Navbar/>
```

CHAPTER 5

RESULT



GoodDeeds Admin

localhost:3000/login

Sign in your account!

Login with your personal details for continue

Username

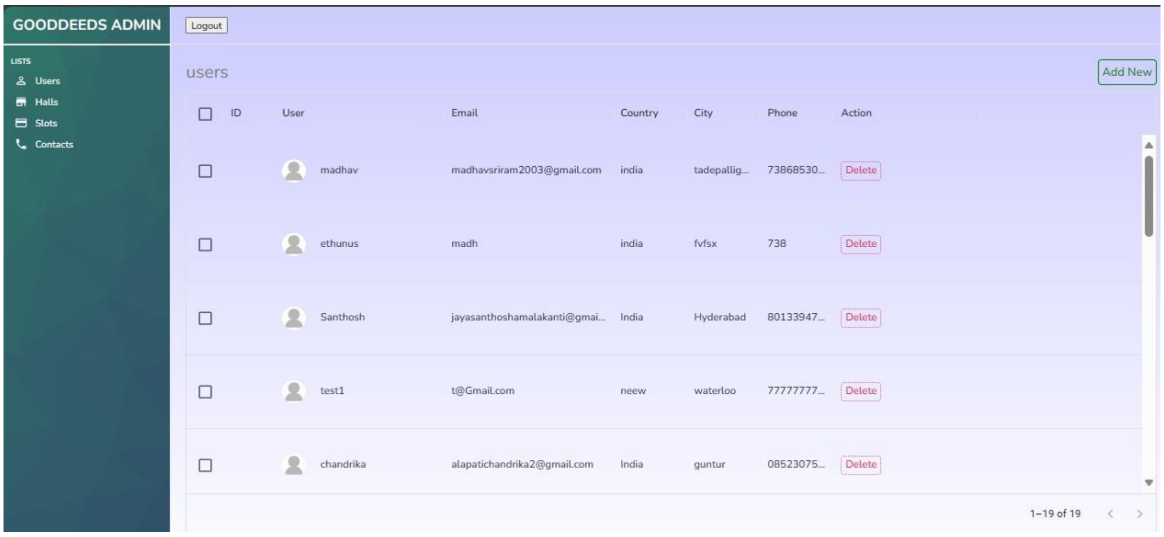
Password

Login

Welcome Back!

to continue please login with your personal account information

Admin login page



GOODDEEDS ADMIN

Logout

LISTS

Users

Halls

Slots

Contacts

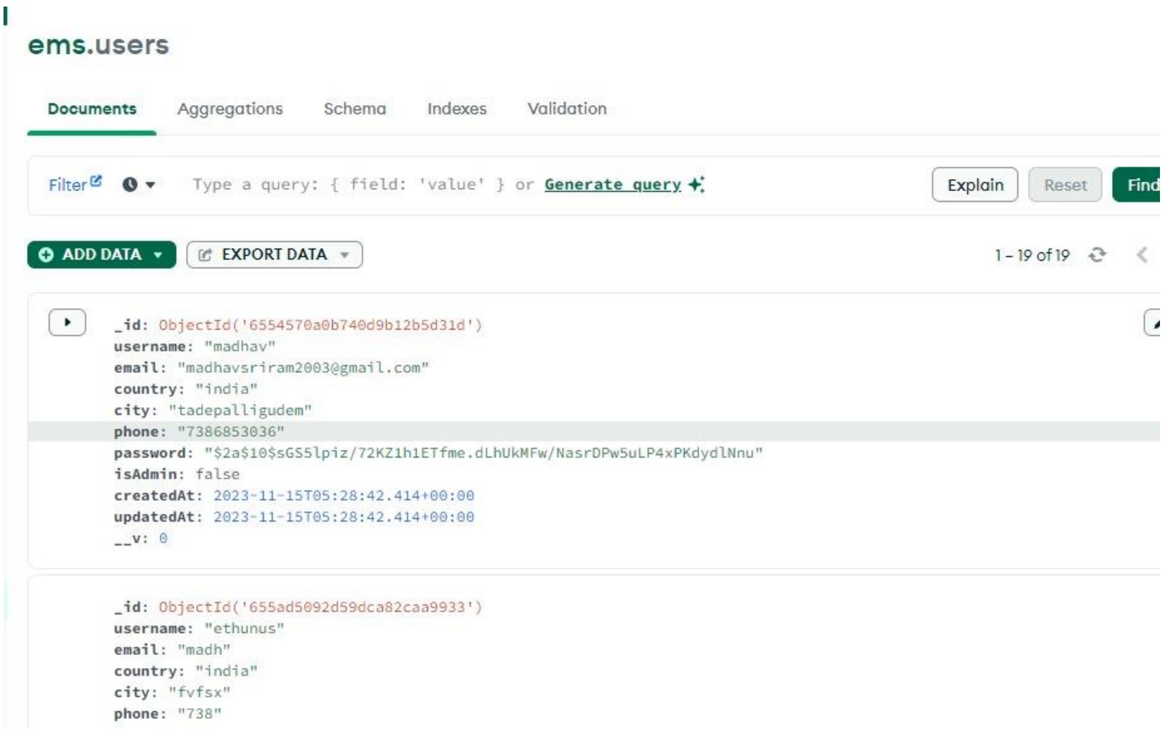
users

Add New

ID	User	Email	Country	City	Phone	Action
	madhav	madhavsriram2003@gmail.com	india	tadepallig...	73868530...	Delete
	ethunus	madh	india	fvfsx	738	Delete
	Santhosh	jayasanthoshamalakanti@gmai...	India	Hyderabad	80133947...	Delete
	test1	t@Gmail.com	neew	waterloo	77777777...	Delete
	chandrika	alapatichandrika2@gmail.com	India	guntur	08523075...	Delete

1–19 of 19

Total number of user


ems.users

Documents Aggregations Schema Indexes Validation

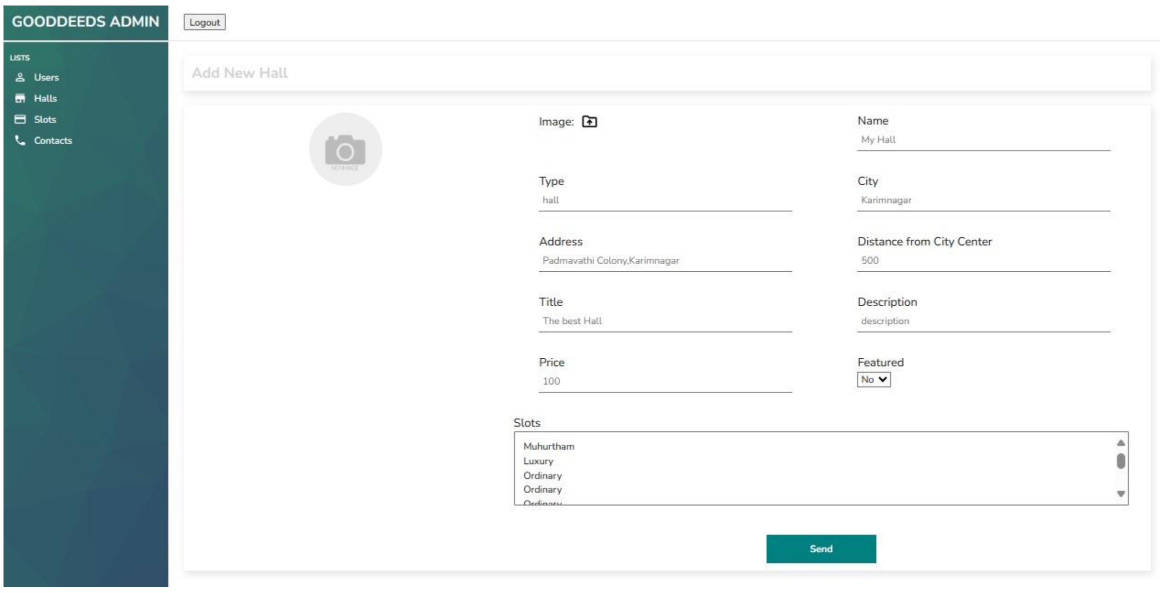
Filter Type a query: { field: 'value' } or Generate query Explain Reset Find

ADD DATA EXPORT DATA 1 – 19 of 19

```
_id: ObjectId('6554570a0b740d9b12b5d31d')
username: "madhav"
email: "madhavsriram2003@gmail.com"
country: "india"
city: "tadepalligudem"
phone: "7386853036"
password: "$2a$10$sGS5lpiz/72KZ1h1ETfme.dLhUkMFw/NasrDPw5uLP4xPKdydlNnu"
isAdmin: false
createdAt: 2023-11-15T05:28:42.414+00:00
updatedAt: 2023-11-15T05:28:42.414+00:00
__v: 0
```

```
_id: ObjectId('655ad5092d59dca82caa9933')
username: "ethunus"
email: "madh"
country: "india"
city: "fvfsx"
phone: "738"
```

Users details stored in mongodb

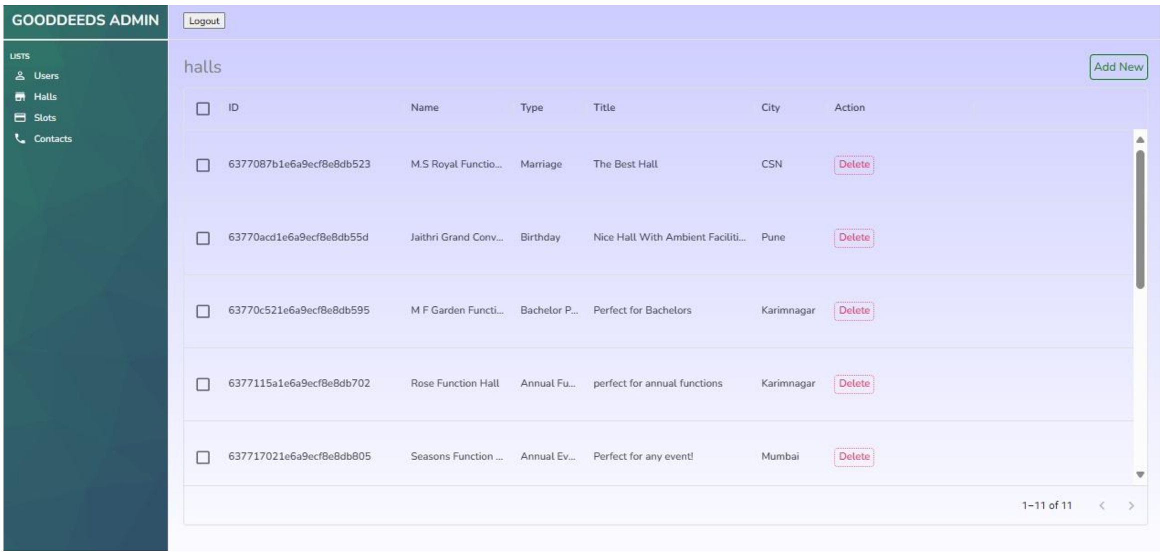


Add New Hall

Image:	Name: My Hall
Type: hall	City: Karimnagar
Address: Padmavathi Colony,Karimnagar	Distance from City Center: 500
Title: The best Hall	Description: description
Price: 100	Featured: No

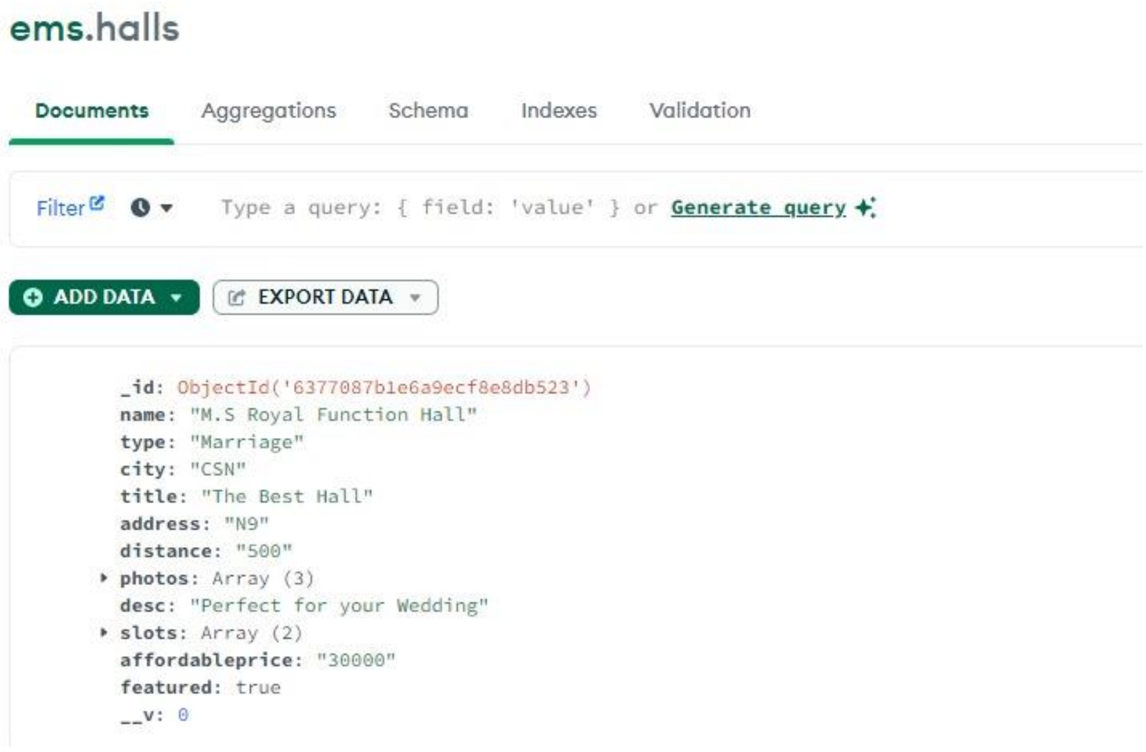
Slots
Muhurtham
Luxury
Ordinary
Ordinary

Adding new function halls



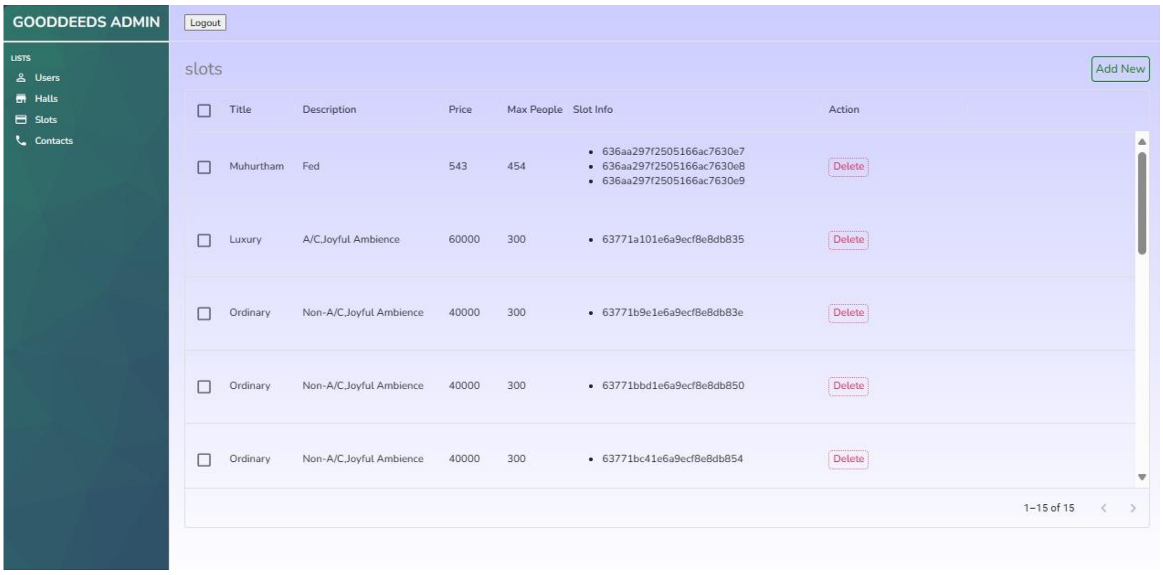
ID	Name	Type	Title	City	Action
6377087b1e6a9ecf8e8db523	M.S Royal Functio...	Marriage	The Best Hall	CSN	Delete
63770acd1e6a9ecf8e8db55d	Jaithri Grand Conv...	Birthday	Nice Hall With Ambient Faciliti...	Pune	Delete
63770c521e6a9ecf8e8db595	M F Garden Functi...	Bachelor P...	Perfect for Bachelors	Karimnagar	Delete
6377115a1e6a9ecf8e8db702	Rose Function Hall	Annual Fu...	perfect for annual functions	Karimnagar	Delete
637717021e6a9ecf8e8db805	Seasons Function ...	Annual Ev...	Perfect for any event!	Mumbai	Delete

Total number of halls



```
_id: ObjectId('6377087b1e6a9ecf8e8db523')
name: "M.S Royal Function Hall"
type: "Marriage"
city: "CSN"
title: "The Best Hall"
address: "N9"
distance: "500"
photos: Array (3)
desc: "Perfect for your Wedding"
slots: Array (2)
affordableprice: "30000"
featured: true
__v: 0
```

function halls in mogodb

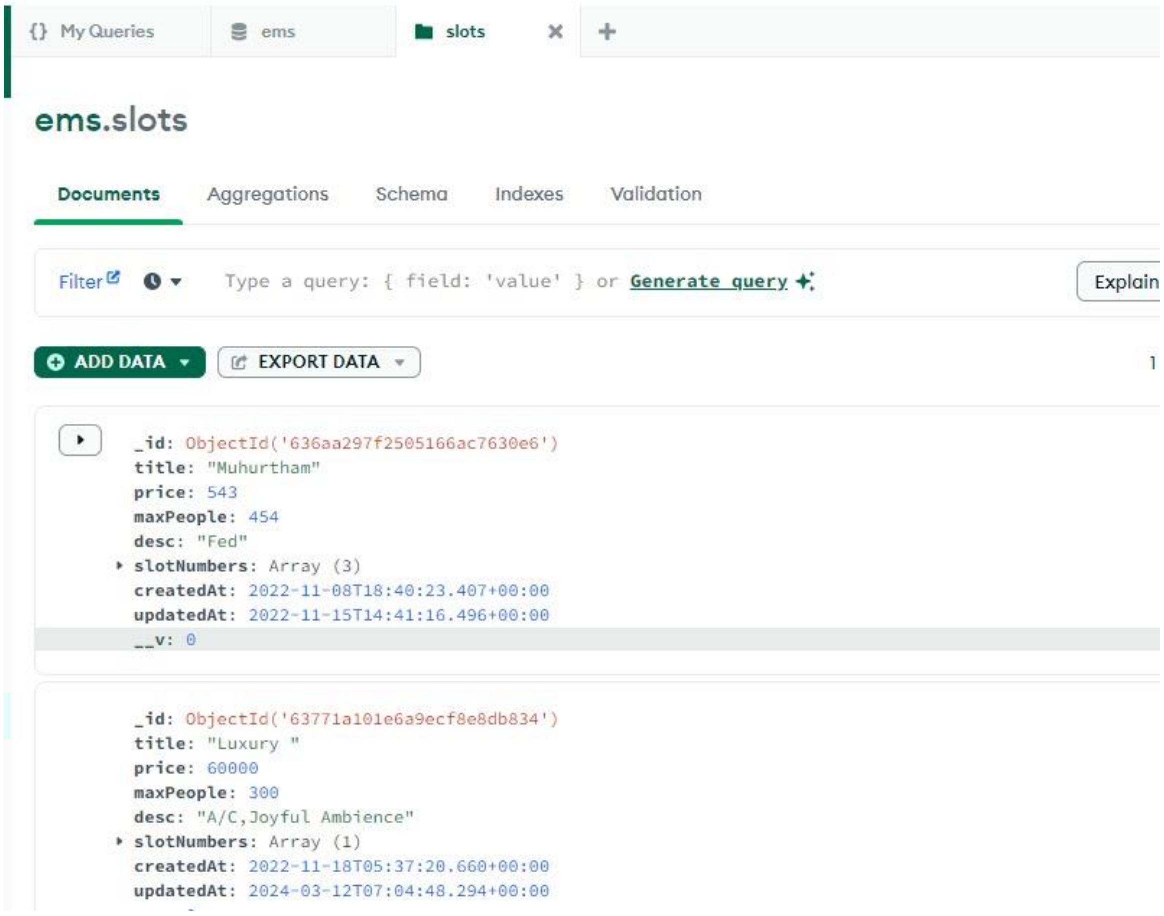


slots

Title	Description	Price	Max People	Slot Info	Action
Muhurtham	Fed	543	454	• 636aa297f2505166ac7630e7 • 636aa297f2505166ac7630e8 • 636aa297f2505166ac7630e9	Delete
Luxury	A/C,Joyful Ambience	60000	300	• 63771a101e6a9ecf8e8db835	Delete
Ordinary	Non-A/C,Joyful Ambience	40000	300	• 63771b9e1e6a9ecf8e8db83e	Delete
Ordinary	Non-A/C,Joyful Ambience	40000	300	• 63771bbd1e6a9ecf8e8db850	Delete
Ordinary	Non-A/C,Joyful Ambience	40000	300	• 63771bc41e6a9ecf8e8db854	Delete

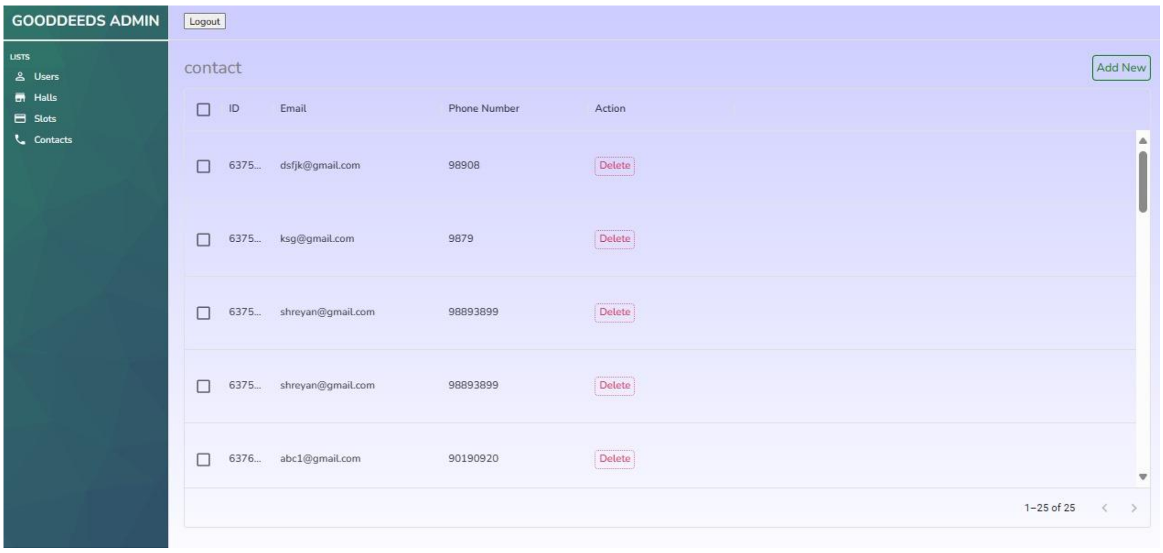
1–15 of 15

Slots booked



```
_id: ObjectId('636aa297f2505166ac7630e6')
title: "Muhurtham"
price: 543
maxPeople: 454
desc: "Fed"
slotNumbers: Array (3)
createdAt: 2022-11-08T18:40:23.407+00:00
updatedAt: 2022-11-15T14:41:16.496+00:00
__v: 0

_id: ObjectId('63771a101e6a9ecf8e8db834')
title: "Luxury "
price: 60000
maxPeople: 300
desc: "A/C,Joyful Ambience"
slotNumbers: Array (1)
createdAt: 2022-11-18T05:37:20.660+00:00
updatedAt: 2024-03-12T07:04:48.294+00:00
```

Slots in mongodb

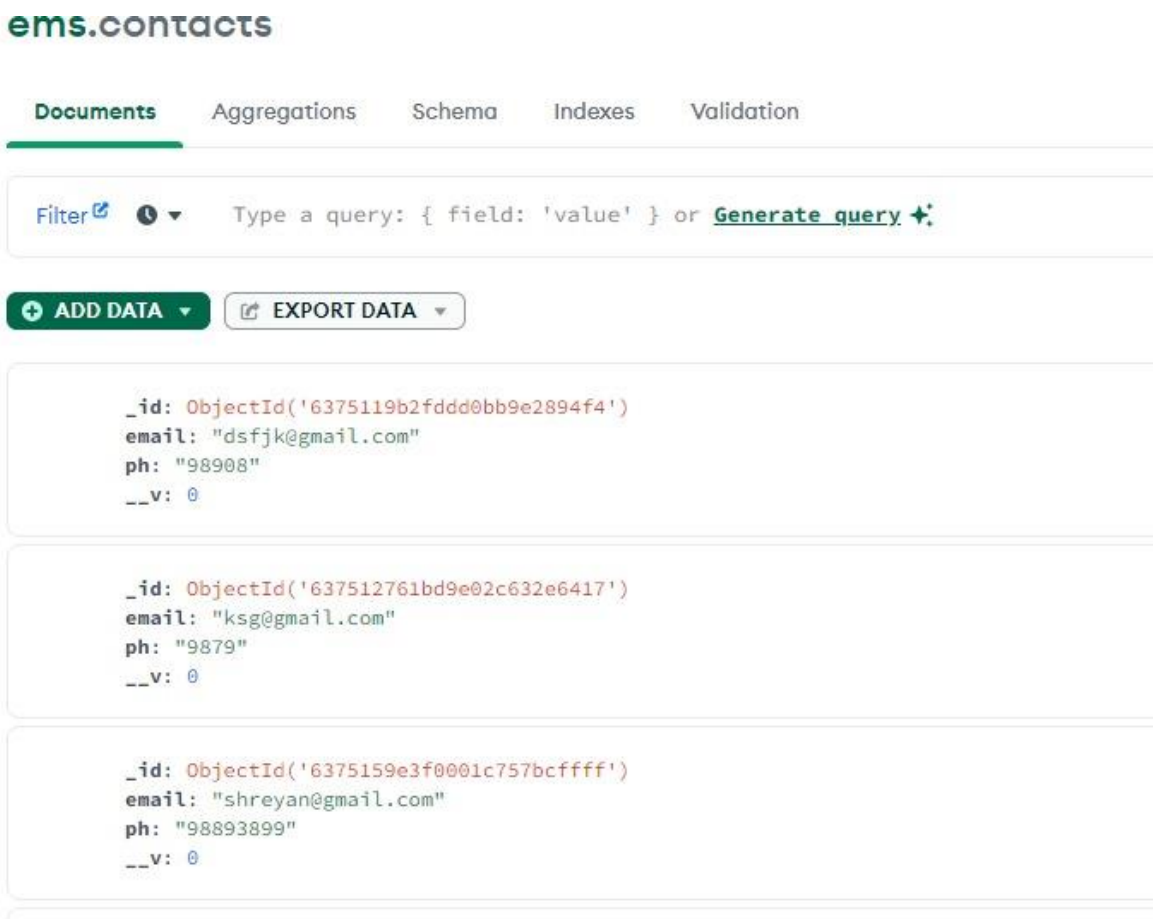


contact

ID	Email	Phone Number	Action
6375...	dsfjk@gmail.com	98908	Delete
6375...	ksg@gmail.com	9879	Delete
6375...	shreyan@gmail.com	98893899	Delete
6375...	shreyan@gmail.com	98893899	Delete
6376...	abc1@gmail.com	90190920	Delete

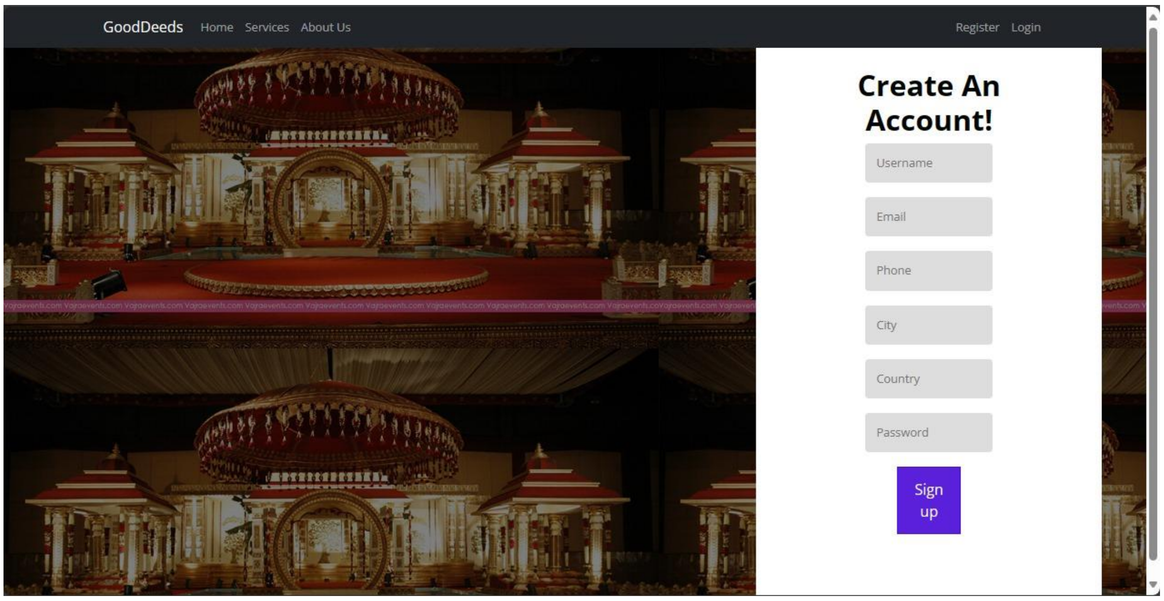
1–25 of 25

Contact info



```
_id: ObjectId('6375119b2fddd0bb9e2894f4')
email: "dsfjk@gmail.com"
ph: "98908"
__v: 0

_id: ObjectId('637512761bd9e02c632e6417')
email: "ksg@gmail.com"
ph: "9879"
__v: 0

_id: ObjectId('6375159e3f0001c757bcffff')
email: "shreyan@gmail.com"
ph: "98893899"
__v: 0
```

Contact info in mongodb



Create An Account!

Username

Email

Phone

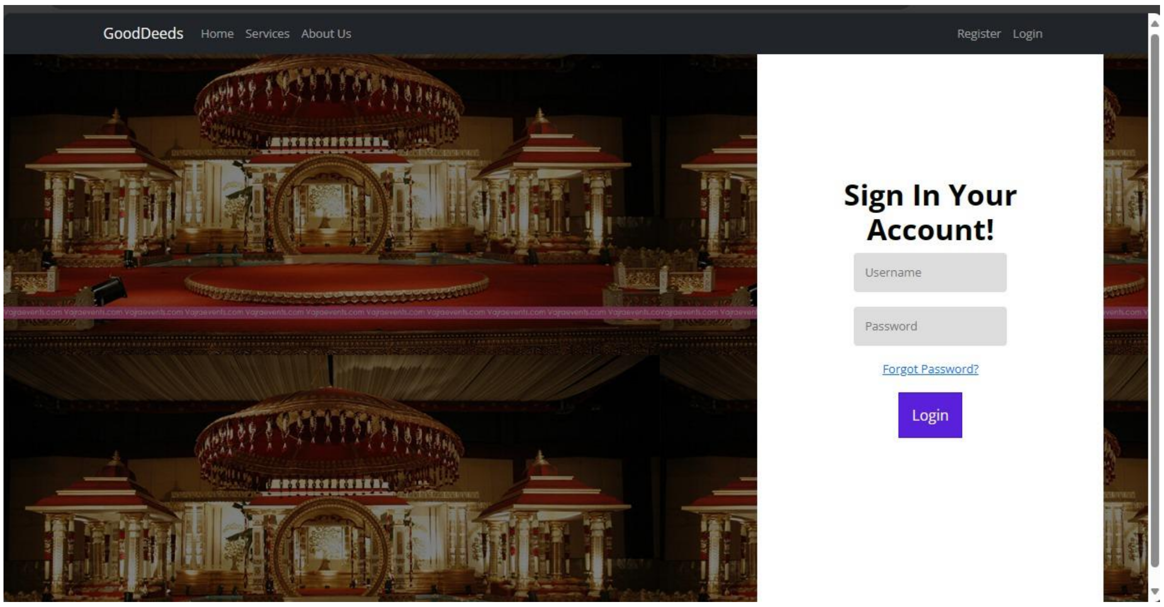
City

Country

Password

Sign up

User’s registeration page



Sign In Your Account!

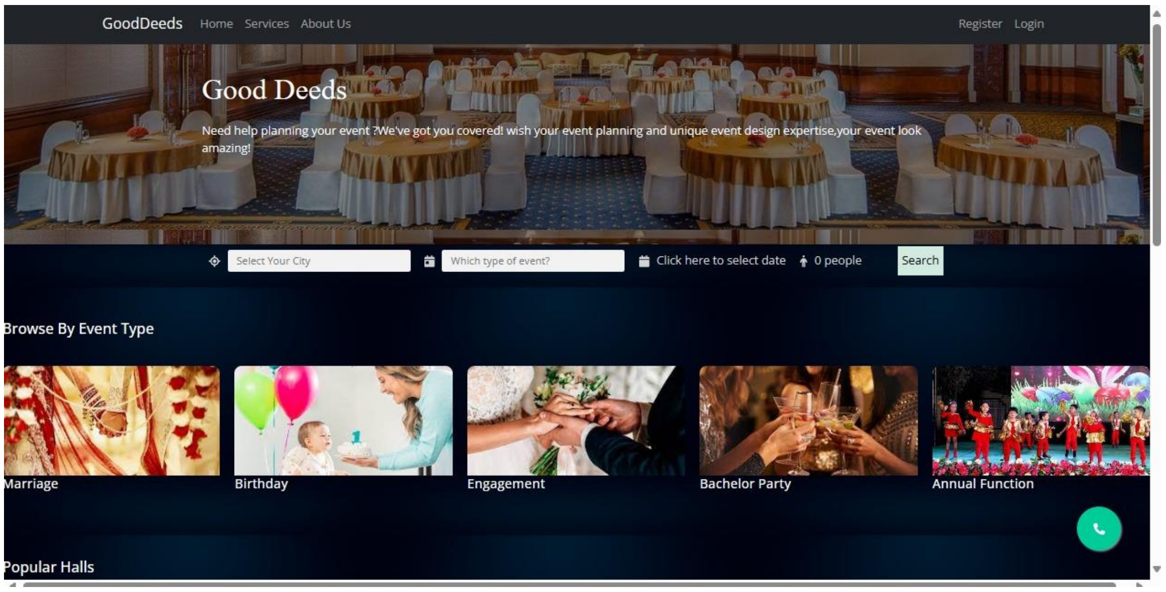
Username

Password

Forgot Password?

Login

User’s login page



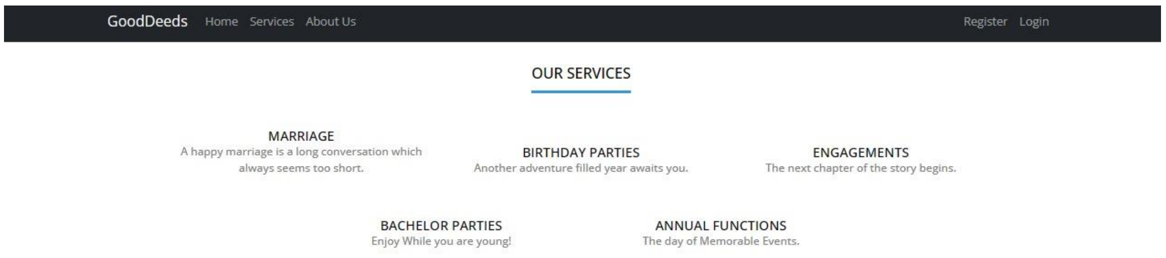
Good Deeds

Need help planning your event ?We've got you covered! wish your event planning and unique event design expertise,your event look amazing!

Browse By Event Type

Popular Halls

Homepage



OUR SERVICES

MARRIAGE
A happy marriage is a long conversation which always seems too short.

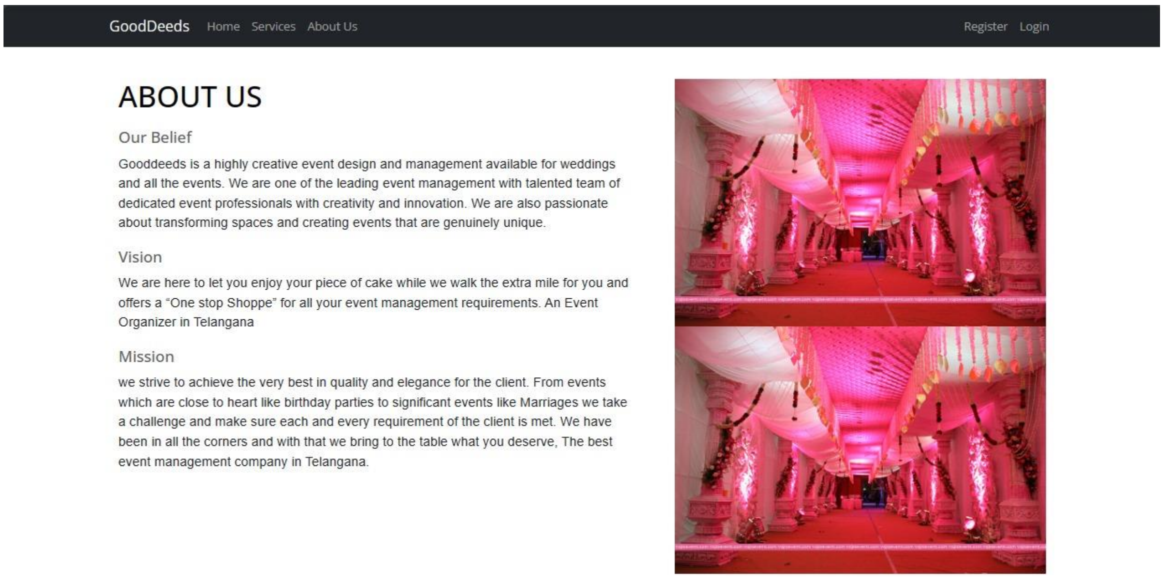
BIRTHDAY PARTIES
Another adventure filled year awaits you.

ENGAGEMENTS
The next chapter of the story begins.

BACHELOR PARTIES
Enjoy While you are young!

ANNUAL FUNCTIONS
The day of Memorable Events.

Services page



ABOUT US

Our Belief

Gooddeeds is a highly creative event design and management available for weddings and all the events. We are one of the leading event management with talented team of dedicated event professionals with creativity and innovation. We are also passionate about transforming spaces and creating events that are genuinely unique.

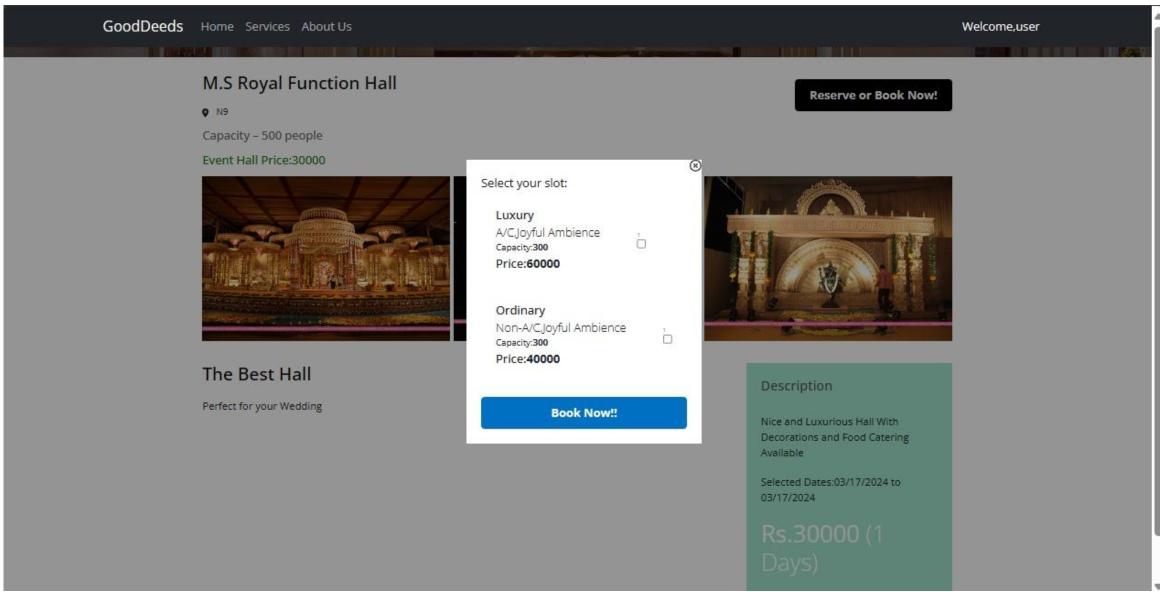
Vision

We are here to let you enjoy your piece of cake while we walk the extra mile for you and offers a "One stop Shoppe" for all your event management requirements. An Event Organizer in Telangana

Mission

we strive to achieve the very best in quality and elegance for the client. From events which are close to heart like birthday parties to significant events like Marriages we take a challenge and make sure each and every requirement of the client is met. We have been in all the corners and with that we bring to the table what you deserve, The best event management company in Telangana.

About us page



M.S Royal Function Hall

N9

Capacity – 500 people

Event Hall Price:30000

Select your slot:

Luxury
A/C,Joyful Ambience
Capacity:300
Price:60000

Ordinary
Non-A/C,Joyful Ambience
Capacity:300
Price:40000

The Best Hall

Perfect for your Wedding

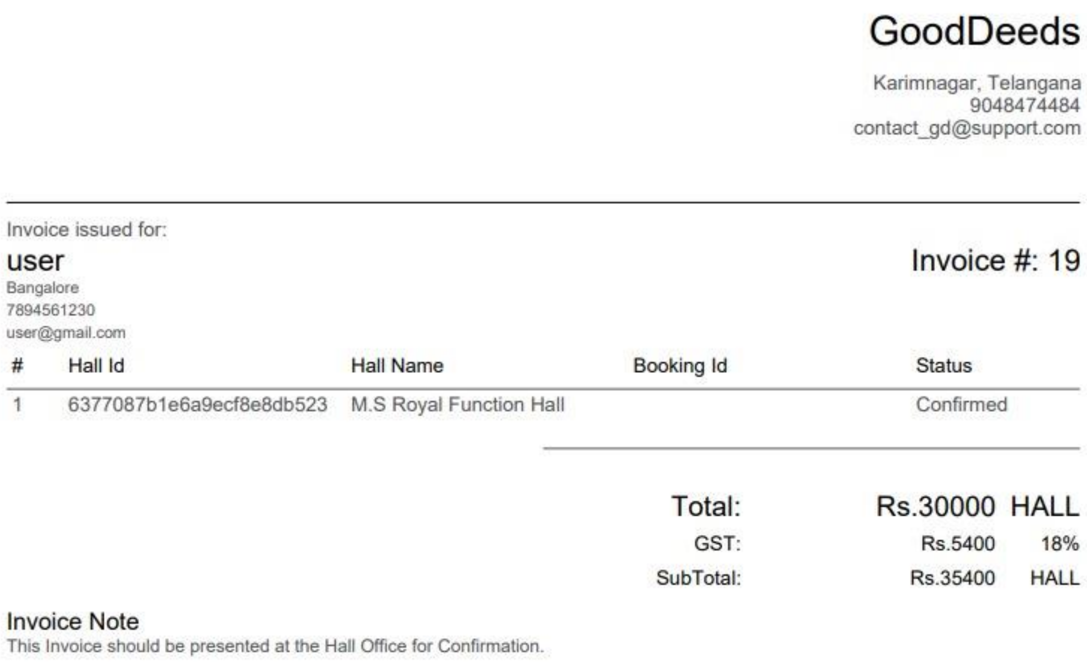
Description

Nice and Luxurious Hall With Decorations and Food Catering Available

Selected Dates:03/17/2024 to 03/17/2024

Rs.30000 (1 Days)

Booking slots



GoodDeeds

Karimnagar, Telangana
9048474484
contact_gd@support.com

Invoice issued for:
user
Bangalore
7894561230
user@gmail.com

Invoice #: 19

#	Hall Id	Hall Name	Booking Id	Status
1	6377087b1e6a9ecf8e8db523	M.S Royal Function Hall		Confirmed

Total:	Rs.30000	HALL
GST:	Rs.5400	18%
SubTotal:	Rs.35400	HALL

Invoice Note
This Invoice should be presented at the Hall Office for Confirmation.

Invoice of slot booked confirmation

CHAPTER 6

CONCLUSION

6.1 Conclusion:

The implementation outlined provides a foundation for an event organization platform using Node.js and MongoDB. It includes user authentication, event creation, and event listing functionalities, demonstrating the core features of the platform. However, there's still room for improvement and expansion in the future.

1. Core Functionality: The implemented features allow users to register, login, create events, and view a list of events.

2. Security: User authentication is implemented using JWT, with passwords securely hashed using bcrypt, enhancing security.

3. Scalability: The application architecture can be scaled horizontally to handle increased traffic and data volume.

4. Flexibility: The use of Node.js and MongoDB provides flexibility for future enhancements and modifications.

6.2 Future Scope:

1. User Profiles: Expand user functionality to include profile management, allowing users to update their information and preferences.

2. Event Details: Enhance event creation with additional fields such as date, time, location, and ticketing options.

3. Event Discovery: Implement advanced search and filtering functionalities to help users discover events more efficiently.

4. Social Features: Add features like comments, ratings, and reviews to encourage user engagement and interaction.

5. Notifications: Introduce real-time notifications to keep users informed about event updates and interactions.

6. Admin Panel: Develop an admin dashboard for managing users, events, and content moderation.

7. Analytics: Integrate analytics tools to track user activity, event attendance, and other key metrics for insights.

8. Accessibility: Ensure the platform is accessible to users with disabilities by following accessibility guidelines.

9. Localization: Support multiple languages to cater to a diverse user base.

10. Performance Optimization: Optimize database queries, caching, and server-side rendering to improve performance and user experience.

By continually iterating and enhancing the platform with these features, the event organization platform can evolve into a comprehensive and user-friendly solution, catering to the needs of event organizers and attendees alike

.

REFERENCES

- AngularJS Tutorial (w3schools.com)
- Angular 2 Tutorial - Javatpoint
- Speech to text angular(SpeechRecognition) - Vangarsushil - Medium