THEME - EVENT REGISTRATION {BookNow@24.7}

What's the prototype/overview of our website?

1) Initially we have the registration form for the event, where the event is fixed as of now in the frontend.
2) For event registration we gave user the option to choose what type of booking he/she want's to make:
    a. Self -
        - If the person is booking for himself/herself.
    b. Group -
        - If the person wants to book tickets for a group.
        - User has the choice to either include him/her or not.
        - Total tickets have to be more than 1.
        - From the backend we will check if any user among the group has any previous booking of any type already made.
        - In this case skipping that member, the rest of the other people will be considered for booking.
    c. Corporate-
        - It's much like a group booking.
        - For verification we may add the functionality to verify the organization email id in future.
        - User has the choice to either include him/her or not.
        - Total tickets have to be more than 1.
        - From the backend we will check if any user among the group has any previous booking of any type already made.
        - In this case skipping that member, the rest of the other people will be considered for booking.
    d. Others -
        - This is a special case where users can make booking for someone else too.
        - The person making the booking might or might not include him/her in the booking process.
        - Users can book even 1 ticket under this category.
3) An account will be automatically created whenever a new user makes any booking for any event.
4) Account related credentials will be mailed to the person who does the bookings.
5) User has to upload his/her IdProof (with a restriction that the file should be a jpeg or png and file size less than 500kb).
5) We preview the details along with the Id proof that the user uploaded.
6) Upon confirmation the booking is made and tickets along with a QR (that could be used at the event location to verify the user) will be mailed to the person.
7) Also a text message will be sent to the user's mobile number showing the minimal necessary details of the event.
8) We have implemented the login forms for users and admin.
9) Admin can login and check their dashboard for detailed information of every booking or every event.
10) A chart is shown to the admin for every event.

11) That chart represents the total count of the registrations made for that corresponding event based on the types (self, group, corporate, others).
12) Admin can click on each bar to populate tables for each type.
13) Table shows the unique registration Id, date and name of the person who made the booking.
14) Upon clicking on any row a modal will open up showing the details related to that registration Id.
15) When user tries to log in:
    a. If a user account doesn't exist with the entered mobile number he/she will be notified to create an account. (user signup form will be created in future)
    b. If a user enters incorrect password the password field is highlighted with error and incorrect password message will be displayed.
    c. If a user enters correct credentials he will be redirected to their dashboard.
16) Almost all the edge cases for event registration form are covered:
    a. Validations at every necessary field.
    b. Users can't proceed further without filling out all the details including the nested group details.
    c. For booking type "Other" there can be a booking for even 1 user, whereas for group and corporate bookings there is restriction of ticket count more than 1.
    d. Checkbox at the end of the form is used to verify that all the details are filled correctly.
    e. In any way if users try to manipulate the booking form it is well handled in the frontend and backend to their levels.
    f. Incorrect login attempts are notified.
    g. Details of login cleared out once the user logout.

What's our vision?

1. The type of events our site provides are awareness, social cause, hackathons, seminars, educational and training camps.
2. This Site consists of two portals Admin and User.
3. Admin manages the site by adding events, updating events, and analyzing the bookings being made for events on their dashboard.
4. Admin accounts can't be created by any other person, it can only be created by other admins and the credentials to that account will be forwarded to the person's email ID whose account was created.
5. Users can make bookings for various ongoing events. And view their booking history by logging into their account on our site. If the person does not create an account and directly makes a booking the account will be created automatically and the credentials will be mailed to the person‘s email. The user can login and reset the password.
6. Most of the edge cases are covered related to event registration from backend as well as frontend.

FRONTEND PART

Tool/Software: Visual studio code, codesandbox
Technologies: reactJS, materialUI, http-proxy-middleware, axios, react-chartjs-2

Basic requirements:

1. Details to be taken from user (fullName, mobile number, e-mail, ID card, gender, registration type, no. of tickets)
2. Registration type should have - **self/group/corporate/others.**
3. Preview details to the user before submitting.
4. Upon submission show a unique auto generated **registration number** to the user.
5. Maintain an **admin login**.

For admin UI:

1. Chart showing count of registration types.
2. List all registrations showing **registration number, date** and **name.**
3. Hyperlink on registration number.

How to Implement/run frontend:

1. Open the downloaded folder in visual studio code.
2. Go to the terminal and type “npm update” - in case any compatibility issues occur, they will be resolved on its own.
3. Type “npm start” - files will be compiled and development server will begin.
4. Wait for output on the browser at http://localhost:3000/
5. Ensure the backend is running fine before doing anything on the frontend.

Functionalities for people of different roles:

Person making the booking:

1. Has the option to select the registration type from self/group/corporate/others.
2. For self the total number of tickets is fixed to 1.
3. For rest types the user has to enter a total number of tickets he/she wants.
4. Group button is enabled once the user enters the correct number of tickets.
5. Upon checking the checkbox all fields are checked if any is left empty.
6. Details are previewed to the user before making the final submission.
7. Upon confirmation the final booking component is loaded with registrationId and booking date retrieved from the backend.
8. Success message with some additional information is displayed.
9. An account of the person making the booking is created with a random password and mailed to the user.

Admin:

1. Upon successful login admin dashboard consists of all the necessary details.
2. A chart is used to show the count of total booking made for each registration type (self/group/corporate/others).
3. Multiple charts will be populated when there are bookings for multiple events.
4. When admin clicks on each bar the corresponding data is displayed in a table beneath the chart.
5. In cases of group/corporate/others a nested table will be displayed when admin clicks on each row.
6. Hyperlink to each registrationId is added to fetch the booking information associated with that registrationId.

User(person who made the booking):

1. After successful booking an account of the person is created.
2. User logs in with the credentials mailed to him/her.
3. Users have to reset their account password.
4. Users can see for what events they have made any bookings for.
5. Users can see/download their ticket they booked, for every event.
6. Users can see their group details (in case the booking was made for the same).

Workflow of frontend:

1. Index.js file renders App.js
2. App.js loads LandingPage.jsx as the initial web page being displayed.
3. LandingPage.jsx returns “login/logout button”, “user login form”, “admin login form” and FormDataInput.jsx
4. If admin logs in, a child component called AdminDashboard.jsx is rendered within LandingPage.jsx else if the user logs in, a child component called UserDashboard.jsx is rendered.
5. FormDataInput.jsx manages the rendering of UserDetails.jsx or DetailsPreview.jsx or ConfirmBooking.jsx
6. FormDataInput.jsx is the parent for all those three components.
7. UserDetails.jsx is taking form values input from the user.
8. DetailsPreview.jsx is previewing all the details filled by the user and the Id Card too.
9. ConfirmBooking.jsx sends the request to the backend with the details the user filled in the form and receives a response/error.
10. When a successful response is received the registration ID and booking Date is displayed to the user with some extra useful information.
11. An option to make another booking is given that reloads the main page again as a fresh entry point.
12. On AdminDashboard.jsx we have a bar chart that shows the total count of registrations made.
13. On clicking each specific bar the corresponding table is loaded showing all registrations made for that type.

BACKEND PART

Tool/Software: MySql Workbench , Eclipse IDE, Postman
Frameworks: Maven, Spring Framework,Hibernate ORM, Spring MVC ,Restful-web services
JRE library: JavaSE-1.8
Server: apache-tomcat-9.0.34
Technologies: MySQL,J2EE
Website connection: Gmail (for sending booking details and account creation on mail) ,Fast2sms (for sending booking confirmed notification on sms)
To be added: Spring Security

Basic Storage Structure:

1. Details to be stored and maintained in the database about the user (Full Name, Phone number, Email, Password, Booking details (booking history)).
2. Details to be stored and maintained in the database about booking details(booking ID, Full Name, Phone number, Gender, Email, Booking date, ID card, Booking type (self/group/corporate/others),Event for which booking is being done, Number of people for which booking is being done, List of details of people whose booking is being done).
3. Details to be stored and maintained in database about the event(ID, Name, About, Total number of seats available, Number of seats left, Event date, Event type (online/live) , List of bookings made for the event, List of people registered for event)
4. Details to be stored and maintained in a database about admin(ID, name, mobile, email, password).
5. Details to be stored and maintained in a database about People(Phone number, name, gender).
   Key points: Gender was taken for future purpose when the website may organize gender specific events. Like Women hackathon.

Functionalities/Methods used by various Modules:
(Functionalities to be implemented in Frontend : Highlighted blue)
(Functionalities Implemented Frontend: Highlighted green)

Admin Called Methods:

1. Login using AdminId and Password.
2. To get the list of details of bookings for an event grouped by registration types.
   The above booking list consists of details **Booking number, date** and **name.**
   The admin can view the booking details in a pop-up by clicking on the booking ID visible on the admin dashboard.
3. Change password.
4. Create a new admin account for new admin's.
5. Admin can add new events.
6. Admin can update details of the events.

User Called Methods:

1. Sign up on the site with basic requirements(Name, Email, Phone number, Password, Gender)
2. Login using Phone number and Password.
3. View details of all events available for booking on home page
4. Change password.
5. Make a new booking to the event.
6. View previous bookings

How to Install Backend :

1. Make sure you have the following(Or equivalent) softwares MySql Workbench , Eclipse IDE, Postman.
2. Start by running the table script (table_script.txt) provided for MySql.
3. For connecting MySql with your project add the url, username and password of Mysql to the file “SpringConfig.java” inside the package “com.project.configuration”.
4. For SMS connection create an account on Fast2sms and copy paste the API of your account to the file “MobileMessage.java” inside the package “com.project.utilities”.
5. For Email configuration you need to add the email (Gmail) and password to the file “JavaMailUtil.java” inside the package “com.project.utilities”.
6. Create a server using Tomcat. We used Tomcat 9, you can use any.
7. Check the server is running properly without running the project on it.
8. If it runs fine, run the application on the server and copy the port number and call the methods using that port number.
9. You can use postman for calling the backend methods.

Methods

1. adminLogin()

This method is used for logging in by admin and viewing the admin dashboard that gives the analysis of event bookings in a graphical view. also he can view the minor details like bookingId, bookingDate and the person’s name who made the booking. Clicking on the bookingId he can have the entire details view about booking.

How to call the method:
URL: http://localhost:{Port Number}/eventRegistration/credential/adminLogin
Method: POST
Request Consumed: JSON
Body: {

```
"adminId":1001,
"password":"Work@24/7"
}
```

Mandatory Fields: adminId, password
Response:

- This method returns an object with fullName(Admin's name), message " LOGGED IN SUCCESSFULLY ", If the data entered was correct.
- If no admin with the id exists returns a null object with message "NO SUCH ADMIN EXIST".
- If the password entered is incorrect, it returns null object with message " INCORRECT PASSWORD ".

2. **userLogin()**

This method is used by users to log In to the account and view the events going on and register in them. Also users can view booking history to verify.
How to call the method:
URL: http://localhost:{Port Number}/eventRegistration/credential/userLogin
Method: POST
Request Consumed: JSON
Body: {

```
"phoneNo":826*********,
"password":"usersPassword"
}
```

Mandatory Fields: phoneNo, password
Response:

- This method returns an object with fullName(User's name), message " LOGGED IN SUCCESSFULLY ", if the data entered was correct.
- If no user with the phoneNo exists returns a null object with message " NO SUCH USER EXIST ".
- If the password entered is incorrect, it returns null object with message " INCORRECT PASSWORD ".

3. **addNewAdmin()** {This Method was created for future expansion of website}

This method allows admins to add new admins. After account creation the new account holder receives the account credentials on their mails registered with us.
How to call the method:
URL: http://localhost:{Port Number}/eventRegistration/credential/addNewAdmin
Method: POST
Request Consumed: JSON
Body: {

```
"fullName":"Khyati Gupta",
"phoneNo":826*******,
"email":"khyati********@gmail.com"
}
```

Mandatory Fields: phoneNo, email
Response:

- Sends A mail of account creation to the admin's email, provided while creating account. The mail consists of the adminId and password. And returns " ADMIN ACCOUNT CREATED SUCCESSFULLY " message as response.
- This method will be accessible by other Admin's only they will be able to add a new admin.
- On violating Mandatory fields message sent back " PHONE NUMBER IS MANDATORY " or " EMAIL IS MANDATORY "
- If unable to create account message sent " ACCOUNT CREATION FOR ADMIN FAILED ".

4. **userSignUp()**

This method is used by new users to create accounts. After successful account creation the user receives an automated welcome mail. The mail consists of credentials if the password was not provided by the user.

How to call the method:

URL: http://localhost:{Port Number}/eventRegistration/credential/userSignUp

Method: POST

Request Consumed: JSON

Body: {

"phoneNo": 826*******,
"fullName":"Khyati Gupta",
"password": "userPassword",
"email": "khyati********@gmail.com",
"gender": "female"

}

Mandatory Fields: phoneNo, email

Response:

- This method is used to create a new account for users.
- When the Account is created a welcome mail is sent to the user's email if the password is unknown to the user(this case occurs when the user doesn't create an account and directly makes a booking in this case his account is created automatically) the password is sent in the mail.
- And returns " USER ACCOUNT CREATED SUCCESSFULLY " message as response.
- If the details already exist an null object with the message " USER ACCOUNT ALREADY EXIST " is returned.
- If mandatory fields are violated an null object with the message " EMAIL IS MANDATORY " or " PHONE NUMBER IS MANDATORY " is returned.
- returns null if unable to store and add the exception thrown by compiler to message, else add message as " USER ACCOUNT CREATION FAILED ".

5. **getUserDetails()** {This Method was created for future expansion of website}

This method will be used by the admin to view the user details(Other than password).

How to call the method:

URL: http://localhost:{Port Number}/eventRegistration/credential/getUserDetails/{user's phoneNo}

Method: GET

Mandatory Fields: user's phoneNo

Response Body : {

"phoneNo": 826*******,
"fullName": "khyati",

```
"password": null,
"email": "khyati*******@gmail.com",
"gender": "female",
"booking": [],
"message": null
}
```


Response:

- Returns the User details (password not included because of privacy concerns).
- If user does not exist returns null object of user with message "NO SUCH USER EXIST".

6. **adminPasswordChange()** {This Method was created for future expansion of website}

This method will be used to help admin to reset their password.

How to call the method:

URL: http://localhost:{Port Number}/eventRegistration/credential/adminPasswordChange

Method: POST

Request Consumed: JSON

Body: {

```
"adminId": 1001,
"password":"oldPassword",
"message":"newPassword"
}
```

Mandatory Fields: adminId, password, message

Response:

- Uses password sent in the message field to set a new password.
- If the passwords are entered incorrectly an null object with message "INCORRECT OLD PASSWORD" or " NEW PASSWORD AND OLD PASSWORD ARE SAME " is returned.
- If the adminId does not exist an null object with message " NO SUCH ADMIN EXIST " is returned.
- If the mandatory fields are missing an null object with a message " NEW PASSWORD WAS NOT ENTERED " or " OLD PASSWORD WAS NOT ENTERED " is returned.
- On successful update message sent " PASSWORD UPDATED SUCCESSFULLY ".

7. **userPasswordChange()** {This Method was created for future expansion of website}

This method will be used by users to reset their password.

How to call the method:

URL: http://localhost:{Port Number}/eventRegistration/credential/userPasswordChange

Method: POST

Request Consumed: JSON

Body: {

```
"phoneNo": 826*******,
"password":"oldPassword",
"message":"newPassword"

}
```

Mandatory Fields: phoneNo, password, message
Response:

- Uses password sent in the message field to set a new password.
- If the passwords are entered incorrectly an null object with message "INCORRECT OLD PASSWORD" or " NEW PASSWORD AND OLD PASSWORD ARE SAME " is returned.
- If the adminId does not exist an null object with message " NO SUCH USER EXIST " is returned.
- If the mandatory fields are missing an null object with a message " NEW PASSWORD WAS NOT ENTERED " or " OLD PASSWORD WAS NOT ENTERED " is returned.
- On successful update message sent " PASSWORD UPDATED SUCCESSFULLY ".

8. **getEventInfo()** {This Method was created for future expansion of website}

This method will help users and admin get the details about the event (other Than who made the bookings and who is attending details).
How to call the method:
URL: http://localhost:{Port Number}/eventRegistration/booking/getEventInfo/{eventId}
Method: GET
Mandatory Fields: eventID
Response Body:


```
{
        "eventId": 1,
        "eventName": "Event Name",
        "description": "About the Event",
        "dateOfEvent": [
                2020,
                5,
                22
        ],
        "totalBookings": 40,
        "remainingBooking": 36,
        "eventType": "online",
        "listOfBookings": [],
        "peopleAttending": [],
        "message": "EVENT INFORMATION EXTRACTED SUCCESSFULLY"
}
```


Response:

- Returns details that are not confidential (like who is attending and who has booked).
- This method will be used to have details of an event, these details are shown on the user's home Page to know about the event.
- If an event does not exist null object with message " NO SUCH EVENT EXIST " .
- Else returns event details in the event object with the message " EVENT INFORMATION EXTRACTED SUCCESSFULLY ".

9. **addNewEvent()** {This Method was created for future expansion of website}

This method allows admins to add new events. That would be displayed on the homepage of the user.
How to call the method:

URL: http://localhost:{Port Number}/eventRegistration/booking/addNewEvent
Method: POST
Request Consumed: JSON
Body: {

"eventName": “Event Name”,
"description": “few lines about the event”,
"dateOfEvent": [YEAR,MONTH,DAY],
"totalBookings": Number of seats for events,
"eventType": "live/online" //this is send from frontend so it is expected to be lower case

}

Mandatory Fields: eventName, dateOfEvent, totalBookings, eventType(case sensitive: lower case)
Response:

- This method can be used by admin to add new events on the site.
- On violation of mandatory fields an null object with message " EVENT NAME IS MANDATORY " or " EVENT DATE IS MANDATORY " or " NUMBER OF BOOKINGS ALLOWED IS MANDATORY " or " EVENT TYPE IS MANDATORY " is returned.
- If event type is not entered correctly "EVENT TYPE CAN BE ONLINE OR LIVE"
- This method adds new events details and returns a message " EVENT ADDED SUCCESSFULLY ".
- If the method is unable to set event details returns null object with message message returned " UNABLE TO ADD EVENT ".

10. **duplicateNumberCheck()** {This Method was created for future expansion of website}

This method was developed to verify phone numbers being entered by users while booking.
How to call the method:
URL: http://localhost:{Port Number}/eventRegistration/booking/duplicateNumberCheck
Method: POST
Request Consumed: JSON
Body: {

"phoneNo": 826*******,
"fullName": "Khyati/Khyati Gupta"

}

Mandatory Fields: phoneNo, fullName
Response:

- This method returns true or false in the message field.
- If the number doesn't exist In database or the number belongs to the same person only it returns a message with “true”.
- If the number belongs to some other person then a message with “false” is returned .
- Exceptional cases handled if database has name "khyati gupta" and request sent has "khyati" vice versa will all return true as it is referring to the same person.
- On violation of mandatory fields " PHONE NUMBER IS MANDATORY " or " FULL NAME IS MANDATORY ".

11. **viewTypeViseBooking()**

This method returns Lists according to the event for the admin dashboard, that takes the input for event vise graphical view for booking made(Grouped in booking types) for event.

How to call the method:

URL: http://localhost:{Port Number}/eventRegistration/booking/viewTypeViseBooking/{eventId}

Method: GET

Mandatory Fields: eventId

Response Body :


```
{
    "other": [
        {
            "bookingId": 1,                    //Booking 1
            "event": null,
            "phoneNo": 0,
            "email": null,
            "gender": null,
            "fullName": "user1",
            "bookingDate": [
                2020,
                6,
                2
            ],
            "noOfPeople": null,
            "bookingType": null,
            "detailsOfPeople": [],
            "idProof": null,
            "message": null
        },
        {
            "bookingId": 3,                    //Booking 2
            "event": null,
            "phoneNo": 0,
            "email": null,
            "gender": null,
            "fullName": "user2",
            "bookingDate": [
                2020,
                6,
                2
            ],
            "noOfPeople": null,
            "bookingType": null,
            "detailsOfPeople": [],
            "idProof": null,
            "message": null
        }
    ],
    "corporate": [],
    "self": [
        {
            "bookingId": 2,                    //Booking 3
```

```
                    "event": null,
                    "phoneNo": 0,
                    "email": null,
                    "gender": null,
                    "fullName": "user3",
                    "bookingDate": [
                                2020,
                                6,
                                2
                            ],
                    "noOfPeople": null,
                    "bookingType": null,
                    "detailsOfPeople": [],
                    "idProof": null,
                    "message": null
                }
            ],
        “group": []
}
```


Response:

- This method is used to provide data for events bookings analysis on admin’s dashboard in the form of bar graphs.
- The data returned is bookingID, Date, Name as only that is to be displayed on the admin dashboard.
- Returns Map of lists of bookings made for particular events sorted according to the type of booking i.e. self,group,corporate and other.
- The returned List is of type:
  [“self” : [{Object of booking of type:self} , {Object of booking of type:self} , {Object of booking of type:self }] ,
  “other” : [{Object of booking of type:other} , {Object of booking of type:other} , {Object of booking of type:other }] ,
  “corporate” : [{Object of booking of type:corporate} , {Object of booking of type:corporate} , {Object of booking of type:corporate }] ,
  “group” : [{Object of booking of type:group} , {Object of booking of type:group} , {Object of booking of type:group }]]
- If the event does not exist one element will be added to the map being returned
  ["NO SUCH EVENT EXIST":null].

12. viewBookingDetails()

This method is used to get the booking details by admin. For admin the method is linked on the dashboard when he clicks on bookingId a page with booking details gets loaded.

How to call the method:

URL: http://localhost:{Port Number}/eventRegistration/booking/viewBookingDetails/{bookingId}

Method: GET

Mandatory Fields: bookingId

Response Body:


```
{
    "bookingId": 1,
```

```
"event": {
        "eventId": 1,                                   //Details of event which is booked
        "eventName": "Event Name",
        "description": "Few lines about the event",
        "dateOfEvent": [ 2020,  5, 22 ] ,
        "totalBookings": 40,
        "remainingBooking": 36,
        "eventType": "online",
        "listOfBookings": [],
        "peopleAttending": [],
        "message": null
    },
"phoneNo": 826*******,                                  //Details of person made the booking
"email": "khyati******@gmail.com",
 "gender": "female",
"fullName": "khyati gupta",
"bookingDate": [ 2020, 6, 2 ],
"noOfPeople": 1,
"bookingType": "other",
"detailsOfPeople": [
        {
            "phoneNo": 971*******,          //Details of person whose booking was done
            "fullName": "kushal",
            "gender": "male",
            "message": null
        }
    ],
"idProof": "64 bit code of picture converted to String",
"message": "DETAILS RETURNED SUCCESSFULLY"
}
```


Response:

- This Method Returns the complete booking details of a particular booking.
- If the booking does not exist null object returned with message "NO SUCH BOOKING EXIST".

13. getLiveEvents() {This Method was created for future expansion of website}

This method is used to get all the events that are open for booking. Events are open for booking before the event date and on the event day. This list of events will be used to display live events on users' home page, and users can view events available for booking and book them if they are interested in them.

How to call the method:

URL: http://localhost:{Port Number}/eventRegistration/booking/getLiveEvents

Method: GET

Mandatory Fields: none

Response Body:


```
[
    {
        "eventId": 2,                                   //Event 1
        "eventName": "Event Name 1",
        "description": "About event",
        "dateOfEvent": [
            2020,
```

```
                    7,
                    7
                ],
            "totalBookings": 50,
            "remainingBooking": 50,
            "eventType": "online",
            "listOfBookings": [],
            "peopleAttending": [],
            "message": null
        },
        {
            "eventId": 3,                                   //Event 2
            "eventName": "Event Name2",
            "description": "About event",
            "dateOfEvent": [
                    2020,
                    7,
                    7
                ],
            "totalBookings": 150,
            "remainingBooking": 150,
            "eventType": "live",
            "listOfBookings": [],
            "peopleAttending": [],
            "message": null
        }
    ]
```


Response:

- This Method returns the List of Events that are open for booking And the event is today or upcoming.
- " NO ON GOING EVENT " message returned if no upcoming event existed.

14. getBookingHistory()

{This Method was created for future expansion of website}

This method is used to get a list of all the bookings done by the user. Each user will have access to their own booking history.

How to call the method:

URL: http://localhost:{Port Number}/eventRegistration/booking/getBookingHistory/{phoneNo user's}

Method: GET

Mandatory Fields: phoneNo

Response Body:


```
{
    "phoneNo": 826*******,                          //user's phone no
    "fullName": null,
    "password": null,
    "email": null,
    "gender": null,
    "booking": [
        {                                                   //booking 1
            "bookingId": 1,
```

```
    "event": {                                  //event details for which booking was made
            "eventId": 1,
            "eventName": "Event Name1",
            "description": "About Event",
            "dateOfEvent": [
                            2020,
                            5,
                            22
                           ],
            "totalBookings": 40,
            "remainingBooking": 36,
             "eventType": "online",
            "listOfBookings": [],
            "peopleAttending": [],
             "message": null
          },
    "phoneNo": 827*******,
   “email": "khyati*******@gmail.com",          //email of the person made the booking
   "gender": "female",
   "fullName": "Khyati Gupta",                  //name  of user made booking
   "bookingDate": [                             //date when the booking was made
                    2020,
                    6,
                    2
                  ],
    "noOfPeople": 1,                    //no of people whose booking are made
    "bookingType": "other",             //booking type:self/group/corporate/other
    "detailsOfPeople": [
                        {
                            "phoneNo": 971*******,   //phoneNo of person attending event
                            "fullName": "person1",   //name of person attending event
                            "gender": "female",
                            "message": null
                         }
                      ],
   "idProof": “64 bit code of id proof image converted to string”,
   "message": null
 },
{
    "bookingId": 3,                     //Booking 2
     "event": {
                "eventId": 2,               //event for which booking was made
                "eventName": "Event Name",
                "description": "About the event",
                "dateOfEvent": [
                                2020,
                                5,
                                22
                              ],
```

```
                                    "totalBookings": 40,
                                    "remainingBooking": 36,
                                    "eventType": "online",
                                    "listOfBookings": [],
                                    "peopleAttending": [],
                                    "message": null
                                },
                            "phoneNo": 826*******,
                            "email": "khyati******@gmail.com",
                            "gender": "female",
                            "fullName": "khyati Guptar",
                            "bookingDate": [
                                            2020,
                                            6,
                                            2
                                           ],
                            "noOfPeople": 2,                                    //no of people attending event
                            "bookingType": "other",
                            "detailsOfPeople": [
                                                {
                                                    "phoneNo": 977*******,          //person1 attending
                                                    "fullName": "kushal verma",
                                                    "gender": "male",
                                                    "message": null
                                                },
                                                {
                                                    "phoneNo": 826*******,          //person2 attending
                                                    "fullName": "Khyati",
                                                    "gender": "female",
                                                    "message": null
                                                 }
                                               ],
                            "idProof": null,
                            "message": null
                        }
                    ],
                "message": "HISTORY SENT SUCCESSFULLY"
}
```


Response:

- Returns list of all the Booking made by a user inside user object type with message " HISTORY SENT SUCCESSFULLY ".
- If no user exists on the corresponding phone no, null object with message " NO SUCH USER EXIST " is returned.
- If No booking history exists a null object with message " NO BOOKINGS EXIST " is returned.
- Return object consists of all bookings made and events for which booking was made and people whose booking was done inside the user object.

15. updateEvent() {This Method was created for future expansion of website}

This method is used by admin's for updating minor details of events.

How to call the method:

URL: http://localhost:{Port Number}/eventRegistration/booking/updateEvent

Method: POST

Request Consumed: JSON

Body: {

```
"eventId": 1,
"eventName": "Event Name",
"description": "About event",
"dateOfEvent": [
                2020,
                5,
                22
            ],
"eventType": "online"

}
```

Mandatory Fields: phoneNo, anyone of: {name,description,date,eventType}

Response:

- If no data to update found a null object with message " NO DATA FOR UPDATE " is returned.
- If the eventId is missing an null object with the message " NO EVENT ID SENT FOR UPDATE " is returned.
- If the eventId is incorrect null object with message " NO SUCH EVENT EXIST " is returned.
- If the server fails to update null object with message " UNABLE TO UPDATE THE DETAILS " is returned.
- If the details are updated properly the received object with message " EVENT ADDED SUCCESSFULLY " is returned.

16. bookEvent()

This method is used for making bookings by user. Users can choose a booking type as self when he is making his own booking. This method automatically creates the account of the person making the booking only if there is no corresponding user account with the phone number provided while making the booking. After the Bookings are made successfully a mail and sms is sent for successful booking. And If the account is made automatically by the method a mail with credentials for the account (phone No, password of 6 digit random number) is sent to the mail Id of the user. This method allows one person to be registered only once. So if the same person tries to make duplicate bookings the booking will fail.If a person(i.e. phoneNo) is already registered in an event he won't be able to get re-registered for the same event but he can be registered in other events. If people details consist of new registrations and re-registration, people not registered will only be registered and the generated ticket will be only for newly registered(for the event) people.

How to call the method:

URL: http://localhost:{Port Number}/eventRegistration/booking/bookEvent

Method: POST

Request Consumed: JSON
Body: {

```
"message": "1",          //sent from front end when user doesn't want to include his details in list of booking details of people(As the front-end by default has his details in list).

"event": {
        "eventId":1      // This is the event ID of the event for which booking is being done
    },
"phoneNo": 826*******,          //Details of the person making booking
"email": "khyati*******@gmail.com",
"gender": "female",
"fullName": "Khyati Gupta",
"bookingType": "group",
"detailsOfPeople": [            // Details of the people whose booking is being made
        {"phoneNo":971*******,"fullName":"Kushal Verma","gender":"male"}
    ],
"idProof": "64 bit code of id proof sent from front-end"
}
```

Mandatory Fields: phoneNo, name, email, eventId, Id Proof, booking type
Response:

- This method makes booking for the people and returns details like no of people actually registered (this number may change only if any person was already registered for the same event), booking id and list of people registered.
- If the person making the booking is not a user his account will be automatically created and the password (Random 6 digit number) will be mailed to him on his registered email ID.
- Mail and SMS for successful booking will be sent on the person making the booking's email and phone number.
- If any of the mandatory details are missing null object with message " DETAILS OF PERSON MAKING BOOKINGS ARE MISSING (PHONE NO/EMAIL/NAME) " or " DETAILS OF BOOKING, REQUIRED FOR MAKING BOOKINGS ARE MISSING (EVENTID/BOOKING TYPE/ID PROOF) " or " NO DETAILS OF PERSON ATTENDING THE EVENT FOUND " is returned.
- If no event exists on eventId null object with message "NO SUCH EVENT EXIST" is returned.
- If the method is unable to find the details of people willing to make booking becoz their bookings are already done for the event " NO VALID DETAILS OF PEOPLE WILLING TO ATTEND THE EVENT EXISTS " + 826*******+ ": IS ALREADY REGISTERED FOR EVENT, SO CAN NOT RE-REGISTER " is returned (All the phone numbers already registered are sent in message).
- If the phone number entered by the user for group details already exists with some other name (example 826******* has name khyati stored in backend and the user enters kushal) a null object with message " PHONE NO ALREADY REGISTERED WITH OTHER USER NAME : " + 826******* is returned.
  The two cases where the person is same are handled internally (also the checking done is case insensitive, so Khyati and khyati are treated same):
  Case 1: when the name in the database is Khyati Gupta and the user enters Khyati only. In this case the booking detail updates with full name.
  Cade 2: when the name in the database is Khyati and the user enters Khyati Gupta. The database is updated to full name.

- If the no of bookings required are more than no of seats left the booking fails and a null object with message " BOOKING FAILED BECAUSE REMAINING SEATS ARE NOT ENOUGH : "+ Number of seats available for event is returned.
- If the booking type is specified to “self” and the number of people is more than one OR the number of people is one only but the details of user do not match the details of person who is making the booking in such cases a null object with message " FOR BOOKING TYPE SELF, DETAILS OF USER SHOULD MATCH DETAILS OF PERSON ATTENDING " is returned.
- If the booking type is other than “self” and the person tries to make his own booking only then the booking type is changed to self by the method.
- On successful the object with message " BOOKING MADE SUCCESSFULLY " is returned or in case some of the phone numbers were already registered the message returned " BOOKING MADE SUCCESSFULLY, PHONE NO ALREADY REGISTERED WITH OTHER USER NAME : " + 826******* this message consists of all the phone numbers who were previously registered.
- After the booking is done the method adds this booking to users booking history and to the list of booking for the event. And the people attending the event are added to the list of people attending the event maintained by the event.