**MEETING NEST – MEETING SCHEDULING SYSTEM**

**PROJECT REPORT**

**PROJECT OVERVIEW :**

This project is a ready to deploy meetings scheduling system. The meeting can only be scheduled by the admin and the update is sent to the user via an email . It is a MEANSTACK based application.

**There are two separate parts of the application :**

* FRONTEND - ANGULAR 7, HTML5, CSS3, JS, Bootstrap
* BACKEND - NodeJS, ExpressJS and Socket.IO Database Allowed - MongoDB and Redis

Angular is a platform that makes it easy to build web based applications. It combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop

**How to create the angular boilerplate application ?**

Firstly create the boilerplate frontend application using following commands

* ng new meeting scheduling system
* cd meeting scheduling system
* ng serve

Create module folders according to the requirement of the project by the following command:

ng g module modulename

and inside module folder create components:

ng g component componentname

**Getting Started**

The instructions provided will get a copy of the project up and running on the local machine for development and testing purposes.

**Prerequisites**

**Make sure before creating an angular application following things are installed :**

* Visual studio code (IDE) (Download from the official site)
* Node JS and NPM (npm comes preinstalled in nodejs now a days)

To check you have node and npm installed on your system,run following commands:

* node -v

npm –v (NPM is a separate project from node js and tends to update itself more frequently.

Therefore we can update the npm with its latest version by this command , npm install npm@latest -g)

* Typescript installed- (npm install –g typescript)
* Install git
* Angular-CLI –(npm install –g @angular/cli)

**Running locally**

1. Create a folder named as meeting-scheduling-system
2. change directory to meeting-scheduling-system

cd meeting-scheduling-system

1. Fetch the source code from my github library

git init

git remote add origin <https://github.com/Surabhie/Meeting-Nest-Frontend.git>

git pull origin master

1. Install all the modules required to run the given application with following command

npm install (This command will install all the modules that is inside package.json)

1. Run the application by using following command

ng serve --open

1. This will navigate to <http://localhost:4200/> on the browser .

And then the application runs.

**REQUIREMENTS AND FLOW:­­­­­**

APPLICATION MODULES :

1. Dashboard Module
2. User Module
3. Meeting Module
4. Shared Module
5. Page-not-found Module
6. Server-error Module

Each module contains components based on the functionalities of the application.

COMPONENTS :

1. DASHBOARD MODULE

User-dashboard Admin- dashboard

1. USER MODULE

About

Login Sign up Reset-password Verify-email Forgot-password

1. MEETING

Create Meeting Update Meeting

1. SHARED MODULES

User-details my-nav

1. SERVER-ERROR

Server-error

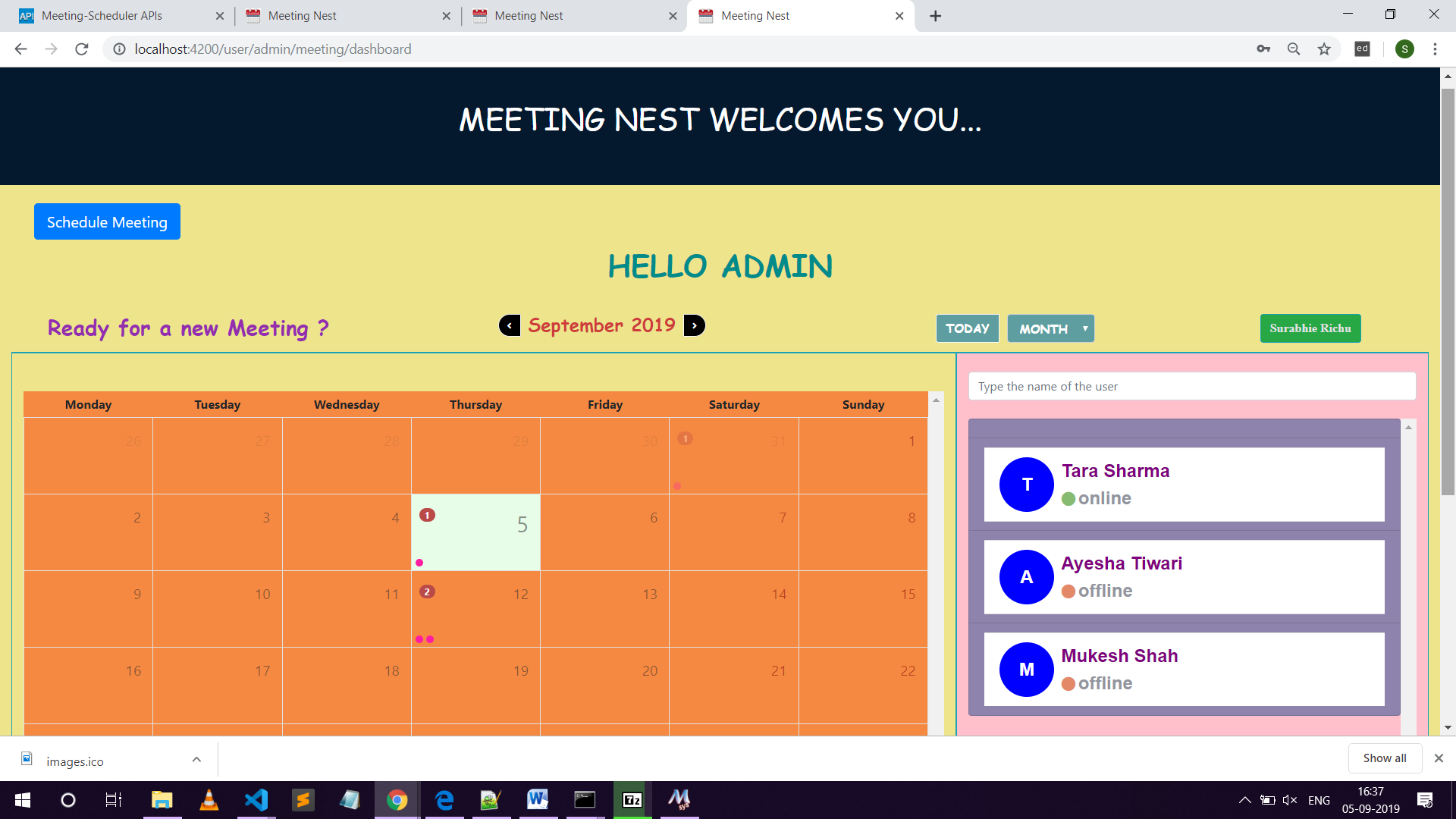
1. PAGE-NOT-FOUND-ERROR

Page-not-found

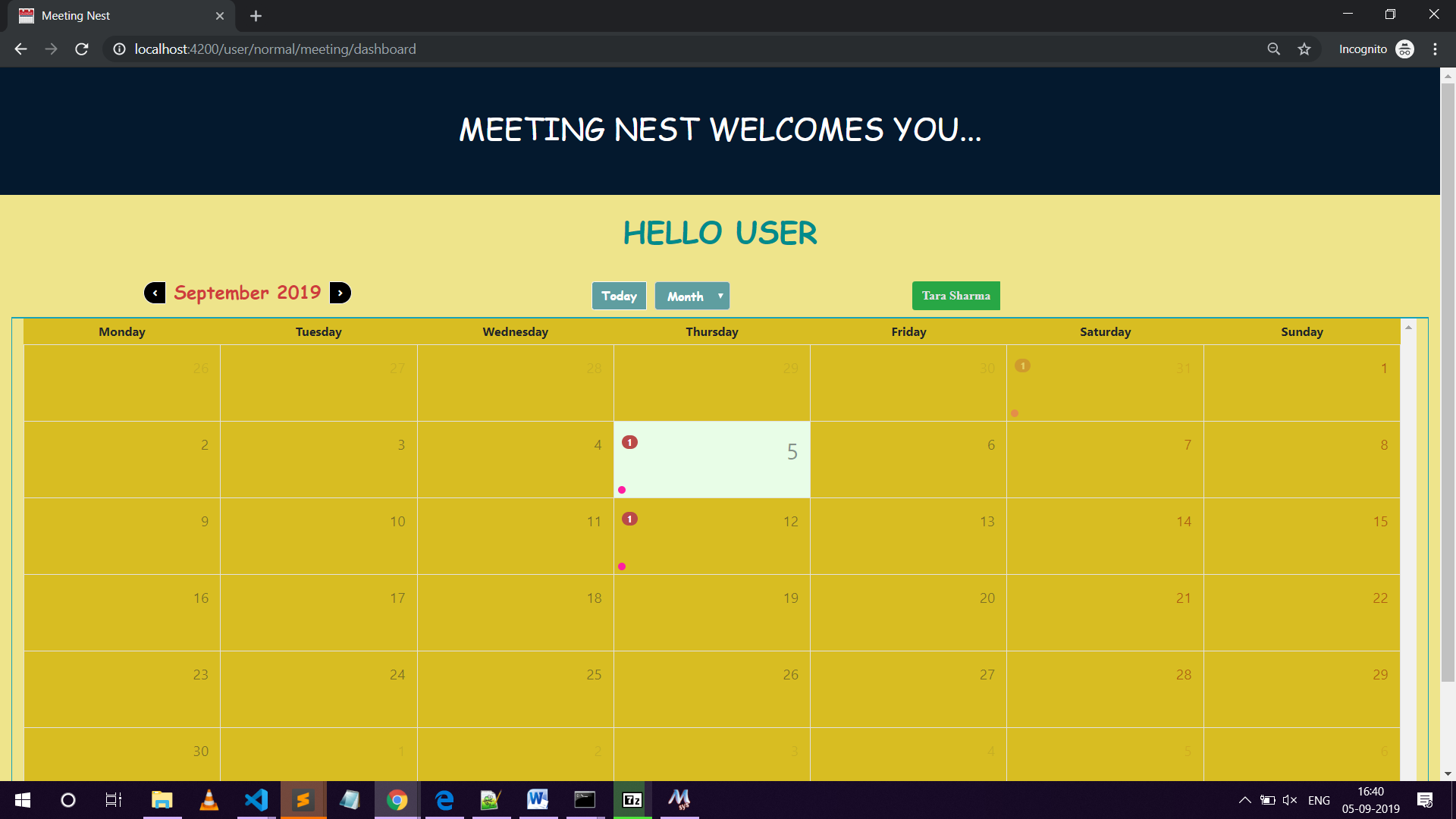
**Screenshots of Application**

**Login Page**

**Admin Dashboard**



**USER DASHBOARD :**



**­­**Application is having following views -­­­­

1. **User management System -**

a) Signup - User signs up on the platform providing details like First Name, Last Name, Email and Mobile number. Country code for mobile number (like 91 for India) will be automatically selected after selecting the Country.

b) Login - User logins using the credentials provided at signup.

c) Forgot password - User can recover password using a link on email.

1. **User Authorization system -**
   1. Two user roles, Normal and Admin.
   2. Admin will be identified with a username ending with "admin" .
   3. Select the option(I am an admin) while registering.
2. **User Slots management system (Flow for normal User) -**

a) Upon login, normal User will be taken to a dashboard showing his current months', planned meetings, in the form of a calendar.

b) User is able to view only his meeting slots and cannot make any changes as admin is the only person who can edit the meetings.

1. **User Slots management system (Flow for Admin)-**

a) Upon login, Admin will be taken to a dashboard, where he/she will be able to see all normal users.

b) Upon clicking on any user, admin will be taken to user's current calendar, with current date selected, by default.

c) Admin is able to add/delete/update meetings on any day, by clicking on a appropriate day-cell/timeline.

1. **User Alerts management system**

a) Normal User will get notifications in real time, though an alert if he is online, and email (irrespective of whether he is online or offline), when i) A meeting is created by admin ii) A meeting is changed by admin iii) 1 minute before meeting, with an option to snooze or dismiss.

1. **Planner Views**

a) Planner view is design and developed in a manner where it will look like a Google calendar.

b) The view have following Features –

i) Planner will show only current year, past and future years will be ignored.

ii) User will be able to change months, through an arrow button.

iii) Day Cells are filled with the dots and count to show the meetings.

iv) Upon click the day's cell, a view show all meetings, along a 24 hr timeline, with the slots covering the exact duration of each meeting.

v) Upon clicking on a meeting, its details should pop up in another view

c) View for Admin : i) For admin, a create button is there in calendar view, to create a meeting.

ii) Upon clicking on create button, admin will be able to fill the details of meeting and can schedule the meeting. I

ii) After scheduling the meeting, Admin will be able to see it on his calendar.

iv) Upon clicking on an already created meeting, meeting details view will display meeting details.

v) Admin can make changes in meeting details by clicking on Edit button associated with Meeting details.

vi) Admin can delete a meeting as well, with delete button.

Error Views and messages - Each major error response (like 404 or 500) are handled by different pages.

Suppose if a user tries to access the page that is not in the application he/she will get a 'Page Not Found Page'.

**SOLUTIONS :**

Features of the Application -

Inside app folder there are following modules with the components mentioned as per the project requirement.

a)User module-signup component,login component,forgot password component,reset password component

b)Dashboard module-admin dashboard component, user dashboard component

c)Meeting module-create meeting component,update meeting component

d)Shared module-user details component,my nav component

e)Page not found component

f)Server error component

FEATURES:

1. REGISTER - User signs up on the platform providing all details like FirstName, LastName, Email and Mobile number. Country code for mobile number (like 91 for India) should is also stored.

**SOLUTION:**

**User module -> signup component**

In signup.component.ts

Step 1- declare all the properties needed by the form

 public firstName: any;

  public lastName: any;

  public email: any;

  public password: any;

  public userName: any;

  public country: any;

  public mobileNumber: any;

  public isAdmin: any;

  public allCountries: any;

  public countryCode: string;

  public countryName: string;

  public countries: any[] = [];

  public countryCodes: string[];

step 2: Now in signupFunction if all the

 data is available,then use that data to build an obj as per the signup

        form that we have created and pass it to the service signUpFunction

let data = {

        firstName: this.firstName,

        lastName: this.lastName,

        mobileNumber: `${this.countryCode} ${this.mobileNumber}`,

        email: this.email,

        password: this.password,

        userName: this.userName,

        countryName: this.countryName,

        isAdmin: this.isAdmin

      }

step 3:  //method to validate username in case of admin

  public validateUserName = (name: string): boolean => {

    if (name.substr(name.length - 6, name.length - 1) != "-admin") {

      return true;

    }

    else {

      return false;

    }

  }//end validateUserName

Checks the string from last 6 places .If it is not equal to admin ,then returns true,normal user else false,Admin

ALL THE FUNCTIONS IN WHICH SERVICE IS USED:

      this.appService.signupFunction(data)

public getCountries() {

    this.appService.getCountryNames()

  public getCountryCodes() {

    this.appService.getCountryNumbers()

Apart from this other functions used are:

  public goToSignIn(): any {

  checkValue(event: any) {

this.isAdmin = event

  }

  public onChangeOfCountry() {

in HTML

Condition for admin:

<div \*ngIf="isAdmin == 'true' && userName">

                <div \*ngIf="validateUserName(userName)" class="alert alert-danger">

                  Admin user should have '-admin' at the end of User Name

                </div>

              </div>

<div class="input-group">

            <div class="checkbox">

              <label>

                <input id="inputAdmin" [(ngModel)]="isAdmin"

 (change)="checkValue(isAdmin?'true':'false')" type="checkbox" name="isAdmin" > I am an admin

              </label>

            </div>

          </div>

NOTE: For differentiating between normal user and an admin there is a flag isAdmin,which returns true if the user is an admin.

1. Login – user then logins using the credentials provided at signup.

SOLUTION:

Step 1- declare all the properties needed

 public email: any;

  public password: any;

step 2: Now in signupFunction if all the

 data is available,then use that data to build an obj as per the signup

        form that we have created and pass it to the service signinFunction

 let data = {

        email: this.email,

        password: this.password,

      }

this.appService.signinFunction(data)

        .subscribe((apiResponse) => {

setTimeout(() => {

              if(apiResponse.data.userDetails.isAdmin == "true"){

                this.goToAdminDashboard();

              }else{

                this.goToUserDashboard();

              }

ALL THE FUNCTIONS IN WHICH SERVICE IS USED:

this.appService.signinFunction(data)

           this.appService.setUserInfoInLocalStorage(apiResponse.data.

userDetails);

Apart from this other functions used in the signin component are:

  public goToAdminDashboard(): any {

    this.router.navigate(['/user/admin/meeting/dashboard']);

  }//end of goToAdminDashboard function

  public goToUserDashboard(): any {

    this.router.navigate(['/user/normal/meeting/dashboard']);

  }//end of goToUserDashboard function

public loginUsingKeypress: any = (event: any) => {

    if (event.keyCode === 13) { // 13 is keycode of enter.

      this.signinFunction();

    }

1. Forgot password - User recovers password using a link or code on email.

SOLUTION: If the user forgets the password,he will click on the forgot password button. He will be redirected to the forgot password page where he has to provide his email id on which the password reset link will be generated and provided.

Then an email will be sent on that email id where link to password reset is given. User clicks on that link and is redirected to password reset page.

Provides new password and submits.

All this process is associated with the backend part .

Used ​Nodemailer​ to send emails.

**AFTER LOGIN DASHBOARD COMES INTO PICTURE**

1. User is of two roles, normal and admin. - SOLUTION PROVIDED

Admin is identified with a username ending with "admin", like "alex-admin" is an admin, since it ends with "admin".

**Flow for normal User-**

1. Upon login, normal User is be taken to a dashboard showing his current months',

planned meetings, in the form of a calendar.

* Current day-cell should be selected by default.
* User can only view his meeting slots and cannot make any changes
* public getUserAllMeetingFunction = () => {//this function will get all the meetings of User.
* this.appService.getUserAllMeeting(this.receiverId,this.authToken).subscribe(
* (apiResponse) => {
* if (apiResponse.status == 200) {
* this.meetings = apiResponse.data;
* //console.log(this.meetings)
* for(let meetingEvent of this.meetings) {
* meetingEvent.title =  meetingEvent.meetingTopic;
* meetingEvent.start = new Date(meetingEvent.meetingStartDate);
* meetingEvent.end = new Date(meetingEvent.meetingEndDate);
* meetingEvent.color = colors.green;
* meetingEvent.remindMe = true
* }
* this.events = this.meetings;
* this.refresh.next();
* this.toastr.success("Calendar has been updated", `MEETINGS FOUND!`);
* //console.log(this.events)
* }
* else {
* this.toastr.error(apiResponse.message,"NO MEETING");
* this.events = [];
* }
* },
* (error) => {
* if(error.status == 400){
* this.toastr.warning("Calendar is not updated", "Either user or Meeting not found");
* this.events = []
* }
* else{
* this.toastr.error("Some Error Occurred");
* this.router.navigate(['/serverError']);
* }
* }
* );
* }//end getAllUserMeetings

NOTE :

INPUT VALUES used in components:

* **activeDay:** If set will be used to determine the day that should be open. If not set, the viewDate is used.
* **activeDayIsOpen**

|  |
| --- |
| Whether the events list for the day of the viewDate option is visible or not |
| *Type :*[boolean](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/boolean" \t "_blank) |
| *Default value :*false |

* **events**

|  |
| --- |
| An array of events to display on view. *Default value :*[] |
| *Type :*CalendarEvent[] |

* refresh

|  |
| --- |
| *Type :*Subject<any> |
| Defined in [projects/angular-calendar/src/modules/month/calendar-month-view.component.ts:173](https://mattlewis92.github.io/angular-calendar/docs/components/CalendarMonthViewComponent.html) | |
| An observable that when emitted on will re-render the current view |  |



|  |
| --- |
| **viewDate** |
| *Type :*[Date](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date) |
| Defined in [projects/angular-calendar/src/modules/month/calendar-month-view.component.ts:147](https://mattlewis92.github.io/angular-calendar/docs/components/CalendarMonthViewComponent.html) | |
| The current view date |  |



|  |
| --- |
| **weekendDays** |
| *Type :*number[] |
| Defined in [projects/angular-calendar/src/modules/month/calendar-month-view.component.ts:234](https://mattlewis92.github.io/angular-calendar/docs/components/CalendarMonthViewComponent.html) | |
| An array of day indexes (0 = sunday, 1 = monday etc) that indicate which days are weekends |  |

|  |
| --- |
| **weekStartsOn** |
| *Type :*[number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/number) |
| Defined in [projects/angular-calendar/src/modules/month/calendar-month-view.component.ts:204](https://mattlewis92.github.io/angular-calendar/docs/components/CalendarMonthViewComponent.html) | |
| The start number of the week |  |

**OUTPUTS:**

|  |
| --- |
| **DayClicked** |
| *Type :*[EventEmitter](https://angular.io/api/core/EventEmitter" \t "_blank) |
| Defined in [projects/angular-calendar/src/modules/month/calendar-month-view.component.ts:247](https://mattlewis92.github.io/angular-calendar/docs/components/CalendarMonthViewComponent.html) | |
| Called when the day cell is clicked |  |

|  |
| --- |
| **EventClicked** |
| *Type :*[EventEmitter](https://angular.io/api/core/EventEmitter" \t "_blank) |
| Defined in [projects/angular-calendar/src/modules/month/calendar-month-view.component.ts:256](https://mattlewis92.github.io/angular-calendar/docs/components/CalendarMonthViewComponent.html) | |
| Called when the event title is clicked |  |

|  |
| --- |
| **EventTimesChanged** |
| *Type :*[EventEmitter](https://angular.io/api/core/EventEmitter" \t "_blank) |
| Defined in [projects/angular-calendar/src/modules/month/calendar-month-view.component.ts:273](https://mattlewis92.github.io/angular-calendar/docs/components/CalendarMonthViewComponent.html) | |
| Called when an event is dragged and dropped |  |

# isSameDay

## Description

Are the given dates in the same day?

**FLOW OF ADMIN USER**

1. Upon login, admin User is taken to a dashboard, showing all normal users in a list format

**SOLUTION :**

**STEP 1- Upon login admin get onto the admin dashboard.**

  public goToAdminDashboard(): any {

    this.router.navigate(['/user/admin/meeting/dashboard']);

  }//end of goToAdminDashboard function

STEP 2: NORMAL USERS IN LIST FORMAT ( IN HTML FILE)

 <div class="people-list">

        <div class="list-group-item list-group list-group-flush"

          style="background-color:rgb(142, 131, 172);color:purple;font-family:sans-serif;font-size:x-large;font-weight:bold"

          \*ngFor="let user of allUsersData | filter:userName">

STEP-3: TO GET LIST OF MEETINGS :

<a (click)="getUserMeetings(user.userId,user.firstName,+ ' ' +user.lastName)"

            class="clearfix cursorPointer list-group-item list-group-item-action" \*ngIf="user.userId != adminId">

IN COMPONENT FILE :

  /\* Database functions \*/

  public getUserMeetings(userId, userName): any { //meetings of user ;

    this.receiverId = userId

    this.receiverName = userName

    this.getUserAllMeetingFunction()

  }//end of getUserMeetings function

Upon clicking on any user, admin should be taken to user's current calendar, with current date selected, by default.

 view: string = 'month';

viewDate: Date = new Date();

getUserAllMeetingFunction()

­­­­­­­­­­­ public getUserAllMeetingFunction = () => {//this function will get all the

meetings of User.

    if (this.receiverId != null && this.authToken != null) {

      this.appService.getUserAllMeeting(this.receiverId, this.authToken).subscribe((apiResponse) => {

        if (apiResponse.status == 200) {

          this.meetings = apiResponse.data;

          for (let meetingEvent of this.meetings) {

            meetingEvent.title = meetingEvent.meetingTopic;

            meetingEvent.start = new Date(meetingEvent.meetingStartDate);

            meetingEvent.end = new Date(meetingEvent.meetingEndDate);

            meetingEvent.color = colors.green;

            meetingEvent.actions = this.actions

            meetingEvent.remindMe = true

          }

          this.events = this.meetings;

          this.refresh.next();

          this.toastr.success("Calendar has been updated", `MEETINGS FOUND!`);

        }

        else {

          this.toastr.error(apiResponse.message, "NO MEETING");

          this.events = [];

        }

      },

        (error) => {

          if (error.status == 400) {

            this.toastr.warning("Calendar is not updated", "Either user or Meeting not found");

            this.events = []

          }

          else {

            this.toastr.error("Some Error Occurred");

            this.router.navigate(['/serverError']);

          }

        }//end error

      );//end appservice.getuserallmeeting

    }//end if

    else {

      this.toastr.info("Missing Authorization key,login again");

      this.router.navigate(['/user/login']);

    }

  }//end getUserAllMeetingFunction

**current date selected, by default :**

/\* Events of Calendar  \*/

  dayClicked(

    { date, events }: { date: Date; events: CalendarEvent[] }

  ): void {

    if (isSameMonth(date, this.viewDate)) {

      this.viewDate = date;

      if ((isSameDay(this.viewDate, date) && this.activeDayIsOpen === true) ||U events.length === 0) {

        this.activeDayIsOpen = false;

      } else {

        this.view = 'day'

      }

    }

  }

1. Admin can delete/update meetings on any day, by clicking on an appropriate day-cell or week cell.

These details are stored in database for every user.

**CREATE MEETING STEPS BY ADMIN**

* For admin, a create button should be there in calendar view, to create a meeting. Upon clicking on create button, details view should open.
* import { NgbModal } from '@ng-bootstrap/ng-bootstrap';// A service for opening modal windows.

NOTE:  when Angular calls ngOnInit it has finished creating a component DOM, injected all required dependencies through constructor and processed input bindings. So here you have all the required information available which makes it a good place to perform initialization logic.

It’s a common practice to use ngOnInit to perform initialization logic even if this logic doesn’t depend on DI, DOM or input bindings.

* Once created, it should appear on the calendar view.
* Upon clicking on an already created meeting, same details view should open.
* Details view should be a form
* Admin should be able to make changes in meeting details form, and submit. vii) Admin should be able to delete a meeting as well, with another button
* Meeting details, should cover when, where and purpose. Also, by default username of the admin, who kept the meeting, should also show in non-editable format.

So, in create meeting,there will be a form that admin has to fill.

In HTML FILE , For bringing the list of people to select one user in the form following code:

<div class="people-list">

                  <div class="card my-card" id="card-design" \*ngFor="let user of allUsersData | filter:userName">

                    <div class="card-body p-2" (click)="getSelected(user)" \*ngIf="user.userId != receiverId">

                      <span class="d-block font-adjust-heading clearfix cursorPointer list-group-item list-group-item-action">

                   <!-- FIRST CHAR -->

                        <user-details [userFirstName]="user.firstName" [userLastName]="user.lastName"

                   [userStatus]="user.status">

                      </user-details>

                    </span>

                   <span class="d-block font-adjust-heading">{{user.email}}</span>

                   i <span class="d-block font-adjust-heading" \*ngIf="user.isAdmin=='true'">Admin

                      </span>

                    </div>

                  </div>

                </div>

**AFTER FILLING ALL THE DETAILS**

**VIEW MEETING: A MODAL WILL OPEN**

<div class="row">

      <div class="col-md-12 text-center">

        <button type="button"style="font-display:cursive;font-weight:bolder" class="btn btn-info rounded" (click)="viewScheduledMeetingFunction()">

          VIEW SCHEDULED MEETING

        </button>

      </div>

    </div>

</div>

**FEW FUNCTIONS OF COMPONENT MENTIONED BELOW –**

  public getSelected = (user) => {

    this.selectedUser = user

  }

  /\* Events based Functions \*/

  //emitted

  public notifyUpdatesToUser: any = (data) => {

    this.socketService.notifyUpdates(data);

  }//end notifyUpdatesToUser

public viewScheduledMeetingFunction: any = () => {

    this.modal.open(this.modalContent, { size: 'lg' });// OPEN METHOD Opens a new modal window with the specified content and supplied options.

  }

**DELETE MEETING STEPS BY ADMIN :**

**Step 1:**

**In admin dashboard component :**

 deleteEvent(event: any): void {

    this.deleteMeetingFunction(event);

    this.events = this.events.filter(iEvent => iEvent !== event); //filter callback function

    this.refresh.next();

   this.activeDayIsOpen = false;

}

  public deleteMeetingFunction(meeting): any {

    this.appService.deleteMeeting(meeting.meetingId, this.authToken)

      .subscribe((apiResponse) => {

        if (apiResponse.status == 200) {

          this.toastr.success("Meeting is successfully deleted");

          let dataForNotify = {

            message: `Hello, ${this.receiverName} has cancelled the meeting - ${meeting.meetingTopic}. Please Check your Calendar/Email for further updates from the admin`,

            userId: meeting.participantId

          }

          this.notifyUpdatesToUser(dataForNotify);

        }

        else {

          this.toastr.error(apiResponse.message);

        }

      },

        (error) => {

          if (error.status == 404) {

            this.toastr.warning("Deletion Failed", "Meeting Not Found ");

          }

          else {

            this.toastr.error("Some Error Occurred");

            this.router.navigate(['/serverError']);

          }

        });//end calling deletemeeting

  }//end deletemeeting

**NOTE: The label for edit and delete for admin: adminDashboardComponent**

**­­**

//EDIT and DELETE label

  actions: CalendarEventAction[] = [

    {

      label: '<i class="fas fa-pencil-alt"></i>       ',

      onClick: (

        { event }: { event: CalendarEvent }

      ): void => {

        this.handleEvent('Edited', event);    //('Edited', event); event demonstrates edit (pop up)

      }

    },

    {

      label: '<i class="fa fa-fw fa-times"></i>        ',

      onClick: ({ event }: { event: CalendarEvent }): void => {

        this.handleEvent('Deleted', event);

      }

    }

  ];

**EDIT MEETING STEPS BY ADMIN : update component**

**Step 1:**

**User Alerts management system**

1. Normal User should also be notified in real time, though an alert if he is online,

and email (irrespective of whether he is online or offline),

**MEETING REMINDER**

  public meetingReminder(): any {

    let currentTime = new Date().getTime();

    for (let meetingEvent of this.meetings) {

      if (isSameDay(new Date(), meetingEvent.start) && new Date(meetingEvent.start).getTime() - currentTime <= 60000

        && new Date(meetingEvent.start).getTime() > currentTime) {

        if (meetingEvent.remindMe && this.gentleReminder) {

          this.modalData = { action: 'clicked', event: meetingEvent };

          this.modal.open(this.modalAlert, { size: 'sm' });

          this.gentleReminder = false

          break;

        }//end inner if

      }//end if

      else if (currentTime > new Date(meetingEvent.start).getTime() &&

        new Date(currentTime - meetingEvent.start).getTime() < 10000) {

        this.toastr.info(`Meeting ${meetingEvent.meetingTopic} has been started!`, `GENTLE REMINDER`);

      }

    }

  }//end of Reminder function

THROUGH EMAIL –REMINDERS

  public sentMeetingRemindersonEmailFunction = () => {//this function will send reminders through an email to the user.

    if (this.authToken != null && this.adminId != null) {

      this.appService.sentMeetingReminders(this.adminId, this.authToken).subscribe((apiResponse) => {

        if (apiResponse.status == 200) {

          this.toastr.success("Meeting Reminders are sent to the user");

        }

        else {

          this.toastr.error(apiResponse.message);

        }

      },

        (error) => {

          this.toastr.error('Server error occured');

          this.router.navigate(['/serverError']);

        }//end error

      );//end sentreminders

    }//end if

    else {

      this.toastr.info("Missing Authorization key,login again");

      this.router.navigate(['/user/login']);

    }//end else

  }//end sentRemindersFunction

when i) A meeting is created by admin

ii) A meeting is changed by admin

iii) 1 minute before meeting, with an option to snooze or dismiss. If snoozed, alert should come again in 5 seconds, if snoozed again, it should re-appear in next 5 seconds and so on.

Once dismissed, alert should no longer appear.

1. Planner Views

Similar to Google or outlook calendars.

The view must follow the following guidelines –

1. Planner should show only current year, past and future years to be ignored.

import { isSameDay, isSameMonth } from 'date-fns';

import { CalendarEvent, CalendarEventAction, CalendarEventTimesChangedEvent } from 'angular-calendar';

1. ii) User should be able to change months, through an arrow button(or prev/next button), each month should show all the dates in tabular format, like in actual calendar.
2. iii) Day Cells should be filled, if any meeting is kept, with a some design. There should also be a design for overlapping meetings.

1. Upon click the day's cell, a view should pop, showing all meetings, along a 24 hr timeline,

with the slots covering the exact duration of each meeting.

v) Upon clicking on a meeting, its details should pop up in another view

**Error Views and messages**​ –

You have to handle each major error response (like 404 or 500) with a different page. Also, all kind of errors, exceptions and messages should be handled properly on frontend. The user should be aware all the time on frontend about what is happening in the system.

BACKEND :

FOLDER STRUCTURE

1. Firstly create a folder meetingBackend
2. Then run npm init command. It basically tries to generate package.json file
3. npm install express –save

Now we have the node\_modules folder. In the similar manner, we will install all the packages that are required in our application.

In controller functions,the call back handles 3 cases: err,if the value is undefined.null or empty and the case of success

FOR HANDLING ERRORS

1-Created an appErrorHandler middleware and routeLevelMiddleware to log the information of the routes.

appErrorHandler- checks for errors and sends the error response. Route not found error handler is also executed .

LIBRARIES:

Created response format

WHILE GENERTAING TOKENS:

1. Token is generated after password validation.
2. token.generateToken (userDetails)where token specifies token library and generateToken is the method in the library.
3. Then passed the userDetails as it is expecting some data.

Why do we create Auth models:

JWT has a token that expires(we have set it).But prob is once token is set,it contains user info.you cannot delete it by going on everyone’s browser and deleting the cookie. So we create a Auth model.

And in login function we use it and add save token function

PASSWORD HASHING:

Password was encrypted before storage to prevent unauthorized access.

Two functions created for this: one to create hashing(used in sign up code later),other to compare the password(used in login code)

EVENT DRIVEN PROGRAMMING –

There is an event and an event handler.

Whenever an event is emitted, even handler function does its work.

The system is waiting for some input from the user. And that input is translated to an event. And when that event is trigerred,whatever associated with that event is called.

SOCKET :

Syntax of initialising socketio:

Let io= socketio.listen(server);

Let myIo=io.of(‘ ‘)

**Built With**

* [Angular](https://angular.io/) - The web based framework used for Frontend Design
* [NPM](https://www.npmjs.com/) - Most of the modules are used
* [nodemailer](https://nodemailer.com/about/) - NPM module to send the mails
* [apiDoc](http://apidocjs.com/) - NPM module to create the apiDoc and eventDoc

**Authors**

* Surabhi J – Application created
* Edwisor for providing instructions

**License**

Use of this source code is governed by an MIT-style license that can be found in the LICENSE file at https://angular.io/license.

**Acknowledgments**

* Problem statement provided by Edwisor.