|| Angular 11 Fundamentals ||

➞ https://jcoop.io/angular-practice-exercises/

■ Typescript :

```
interface ICat {
 name:string
 age?:number
}

let fluffy:ICat
```

```
fluffy =
  {
    name: 'Fluffy'
  }
```

TypeScript Interfaces

```
class Cat {
  name:string
  constructor (name) {
    this.name = name
  }
}
```

TypeScript Class Properties

```
class Cat {
  name:string
  color:string
  constructor (name) {
    this.name = name
  }
  speak() { console.log('My name is: ' + this.name) }
}
```

Public and Private Accessibility

```
class Cat {
  private name:string
  private speak() { console.log('My name is: ' + this.name) }
}

let fluffy = new Cat()
console.log(fluffy.name) //compile-time error
fluffy.speak() // compile-time error
```

Public and Private Accessibility

```
class Cat {
  private name:string
  private color:string
  constructor(name, color) {
    this.name = name
    this.color = color
  }
}
```

```
class Cat {
  constructor(private name, private color) {

  }
}

let fluffy = new Cat('Fluffy', 'White')
```

Public and Private Accessibility

- **Angular Vs AngularJS :**

Feature	Angular	AngularJS
Architecture	Uses a component-based architecture with modules	Follows a controller-based architecture
Language	Written in TypeScript	Written in JavaScript
Rendering	Uses real DOM and supports virtual DOM for efficiency	Uses real DOM
Binding	Supports two-way data binding (ngModel)	Uses two-way data binding extensively
Directives	Uses directives with a more modular syntax	Directives have a complex and verbose syntax
Dependency Injection	Hierarchical dependency injection system	Uses a simpler dependency injection system
Performance	Generally better performance due to optimized change detection	May experience performance issues with complex bindings
Tooling	Strong tooling support with Angular CLI	Limited tooling compared to Angular CLI
Mobile Support	Native mobile app development with Angular NativeScript or Ionic	Limited native mobile support
Community	Active and large community support	Mature community, but not as active as Angular
Learning Curve	Steeper learning curve due to TypeScript and complex concepts	Relatively easier learning curve, especially for beginners
Backward Compatibility	May have breaking changes between major versions	Generally backward compatible, but may require updates
Version History	Angular (Angular 2+)	AngularJS (Angular 1.x)

- **Communicating with Child Components Using @Input :**

events-list.component.html U

src > app > events > events-list.component.html > div

```
16  <div>
17      <!-- Heading for Upcoming Angular Events -->
18      <h2>Upcoming Angular Events</h2>
19      <!-- Horizontal line as a visual separator -->
20      <hr>
21
22      <!-- Include an event thumbnail component with event1 as the input -->
23      <!-- Note: This assumes that there is a component named 'event-thumbnail' -->
24      <event-thumbnail [event]="event1"></event-thumbnail>
25      <!-- event1 is corresponding to event1 object inside the events-list-component.ts file -->
26      <!-- [event] is corresponding to  our Input() property over there inside events-thumbnail-component.ts file-->
27      <!-- Line break for better spacing -->
28      <br>
29  </div>
```

events-list.component.ts U

src > app > events > events-list.component.ts > ...

Click here to ask Blackbox to help you code faster

```ts
// Importing Component decorator from Angular core module
import { Component } from '@angular/core';

// Component decorator configuration
@Component({ ···
})
// EventsListAppComponent class definition
export class EventsListAppComponent {
  // Sample event data
  event1 = {
    id: 1,
    name: 'Angular Connect',
    date: '09/03/2024',
    time: '10:00 AM',
    price: 998.0,
    imageUrl: '/assets/images/angularconnect-shield.jpg',
    location: {
      address: 'Pimple-Saudagar',
      city: 'Pune',
      country: 'India',
    },
  };
}
```

events-thumbnail.component.ts U

src > app > events > events-thumbnail.component.ts > ...

Click here to ask Blackbox to help you code faster

```ts
// Importing necessary decorators and modules from Angular core
import { Component, Input } from '@angular/core';

// Component decorator configuration
@Component({
  // Selector for the component to be used in HTML
  selector: 'event-thumbnail',
  // Template for the component
  template: `
    <!-- Well-styled container for event details -->
    <div class="well hoverwell thumbnail">
      <!-- Event name -->
      <h4>{{ event.name }}</h4>
      <!-- Event date -->
      <div>Date: {{ event.date }}</div>
      <!-- Event time -->
      <div>Time: {{ event.time }}</div>
      <!-- Event price -->
      <div>Price: \${{ event.price }}</div>
      <!-- Location details -->
      <div>
        <span>Location: {{ event.location.address }}</span>
        <!-- Adding padding to the left for better spacing -->
        <span class="pad-left"
          >{{ event.location.city }}, {{ event.location.country }}</span
        >
      </div>
    </div>
  `,
  // Styles for the component
  styles: [
    `
      /* Styling for the left-padding class */
      .pad-left {
        margin-left: 10px;
      }
      /* Styling for the well div */
      .well div {
        color: #fafad2; /* Setting text color */
      }
    `,
  ],
})
// EventsThumbnailAppComponent class definition
export class EventsThumbnailAppComponent {
  // Input property to receive event data from parent component
  @Input() event: any;
}
```

■ Communicating with Child Components Using @Output :

- When you click on click me button you are able to see event name data inside console output as event will be emitted when you click on btn & handleEventsClick() will call.

events-list.component.ts U

src > app > events > events-list.component.ts > ...

Click here to ask Blackbox to help you code faster

```ts
// Importing Component decorator from Angular core module
import { Component } from '@angular/core';

// Component decorator configuration
@Component({ ···
})
// EventsListAppComponent class definition
export class EventsListAppComponent {
  // Sample event data
  event1 = { ···
  };

  handleEventClicked(data: any) {
    console.log('Received:', data);
  }
}
```

events-list.component.html U

src > app > events > events-list.component.html > div

```html
<div>
    <!-- Heading for Upcoming Angular Events -->
    <h2>Upcoming Angular Events</h2>
    <!-- Horizontal line as a visual separator -->
    <hr>

    <!-- Include an event thumbnail component with event1 as the input -->
    <!-- Note: This assumes that there is a component named 'event-thumbnail' -->
    <event-thumbnail (eventClick)="handleEventClicked($event)" [event]="event1"></event-thumbnail>
    <!-- (eventClick) is corresponding to  our Output() property or event emitter over there inside events-thumbnail-compone
    <!-- $event refers to the data emitted with our event -->

    <!-- Line break for better spacing -->
    <br>
</div>
```

events-thumbnail.component.ts U

src > app > events > events-thumbnail.component.ts > ...

```ts
// Importing necessary decorators and modules from Angular core
import { Component, EventEmitter, Input, Output } from '@angular/core';

// Component decorator configuration
@Component({
  // Selector for the component to be used in HTML
  selector: 'event-thumbnail',
  // Template for the component
  template: `
    <!-- Well-styled container for event details -->
    <div class="well hoverwell thumbnail">
      <!-- Event name -->
      <h4>{{ event.name }}</h4>
      <!-- Event date -->
      <div>Date: {{ event.date }}</div>
      <!-- Event time -->
      <div>Time: {{ event.time }}</div>
      <!-- Event price -->
      <div>Price: \${{ event.price }}</div>
      <!-- Location details -->
      <div>
        <span>Location: {{ event.location.address }}</span>
        <!-- Adding padding to the left for better spacing -->
        <span class="pad-left"
          >{{ event.location.city }}, {{ event.location.country }}</span
        >
      </div>
      <button class="btn btn-primary" (click)="handleClickMe()">Click Me..!!</button>
    </div>
  `,
  // Styles for the component
  styles: [...
  ],
})
// EventsThumbnailAppComponent class definition
export class EventsThumbnailAppComponent {
  // Input property to receive event data from parent component
  @Input() event: any;
  @Output() eventClick = new EventEmitter();

  handleClickMe() {
    this.eventClick.emit(this.event.name);
  }
}
```

■ Using Template Variables to Interact with Child Components :

- use to access public properties & methods of a child components.

- when we click on Log me Demo String button we are able to see Demo String..!! inside console output.

events-list.component.html U

src > app > events > events-list.component.html > div

```html
<div>
    <!-- Heading for Upcoming Angular Events -->
    <h2>Upcoming Angular Events</h2>
    <!-- Horizontal line as a visual separator -->
    <hr>

    <!-- Include an event thumbnail component with event1 as the input -->
    <!-- Note: This assumes that there is a component named 'event-thumbnail' -->
    <event-thumbnail #thumbnail [event]="event1"></event-thumbnail>
    <!-- #thumbnail is the variable name  & we can use anywhere else in our template -->
    <!-- using thumbnail we are able to call any propertyor method from events-thumbnail.component.ts file -->

    <button class="btn btn-primary" (click)="thumbnail.logDemoStr()">Log me Demo String</button>
    <h3>{{thumbnail.someProperty}}</h3>
    <!-- Some value as a output for above h3 line -->

    <!-- Line break for better spacing -->
    <br>
</div>
```

(element) div: HTMLDivElement

events-thumbnail.component.ts U

src > app > events > events-thumbnail.component.ts > ...

```
45  export class EventsThumbnailAppComponent {
46    // Input property to receive event data from parent component
47    @Input() event: any;
48
49    logDemoStr() {
50      console.log('Demo String..!!');
51    }
52
53    someProperty: any = 'Some value';
54  }
```

■ Interpolation, Property Bindings, & Expressions :

- Interpolation is used when you want to display the data.

- property binding is used when you want to bind the data to the property of a DOM element.

- Here we're binding the user image URL to src property of this image tag.

- Interpolation & property binding both use expressions to specify the data from the component to bind to.

- To use interpolation we can enclose our expression inside double curly braces.

- To use binding we can put property inside square brackets & expression inside quotes.

- Inside interpolation we can also use {{ 2 + 2 }} , {{ getValue() }} , etc. But we cannot use expressions like below :

1. Assignments : (+=, =, ++, etc)

2. new keyword

3. expression chaining with ;

4. Global name space such as console, window , etc.

```
...
@Component({
  template: `
    <h2>{{user.name}}</h2>
    <img [src]="user.imageUrl"/>
    <button (click)="doSomething()"></button>`
})
export class ProfileComponent {
  user = {name: 'John Doe', imageUrl: 'doe.com/profile.jpg'}

  doSomething() {
  }
}
```

Interpolation

Property Binding

Event Binding

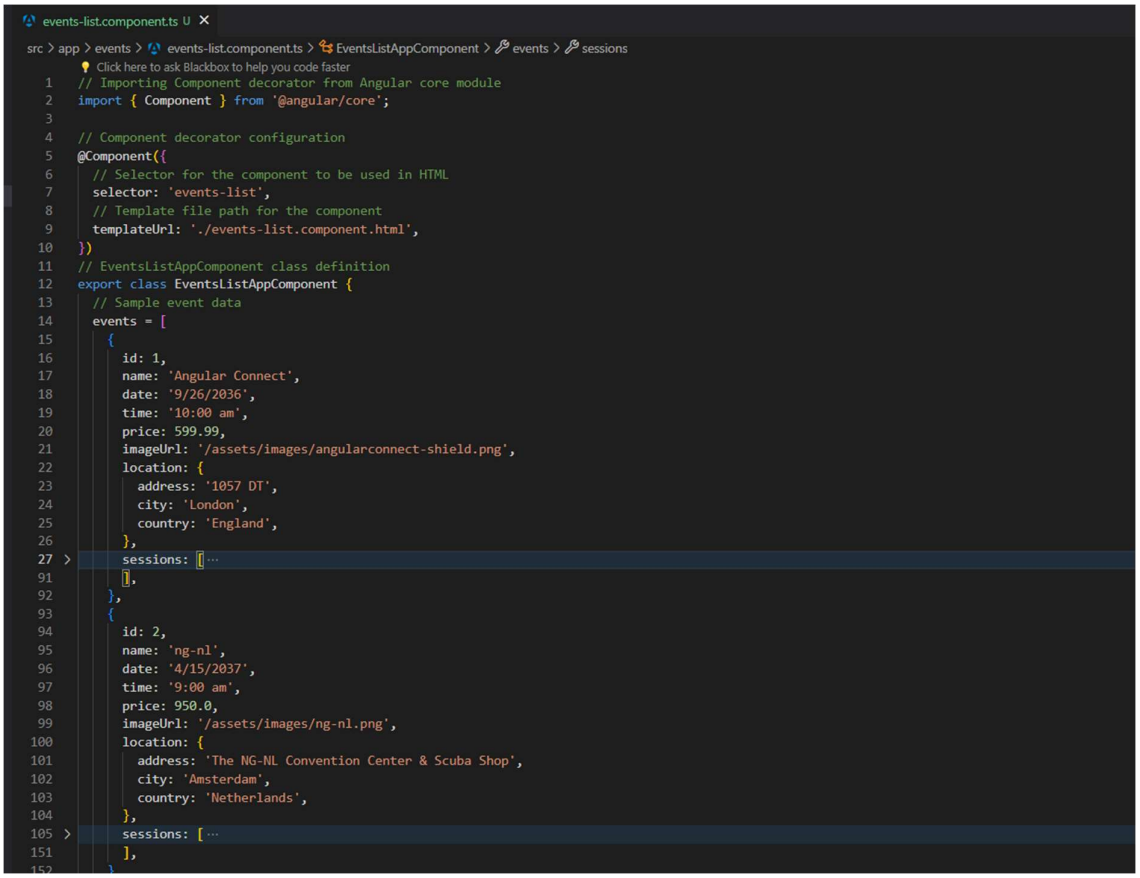
■ Event Bindings, & Statements :

- In above image we are able to see event Binding, at button line. Notice that event binding uses parentheses around the element event to bind it.

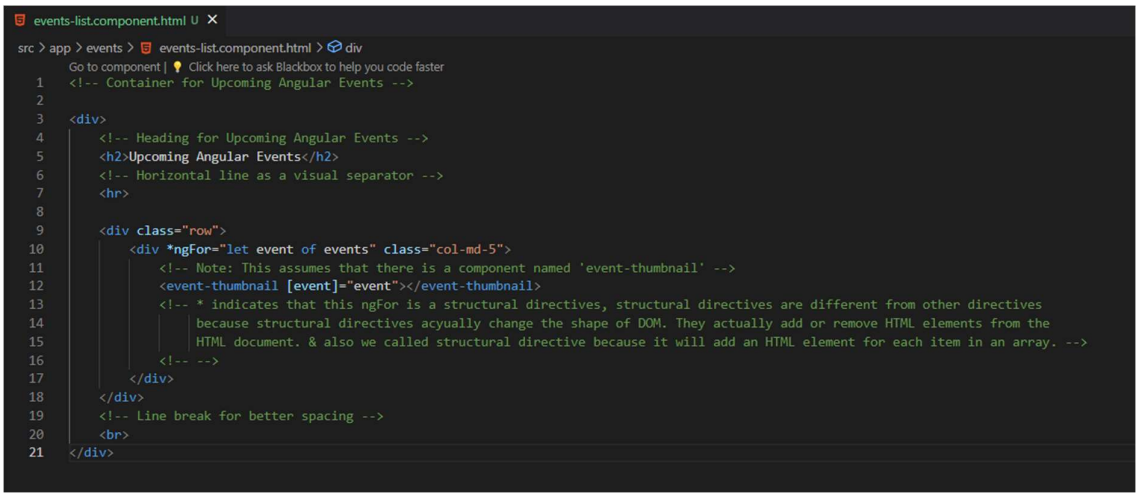
- Inside event binding statement restrictions are : 1. Assignments : (+=, ++, etc), 2. new keyword

3. Global name space such as console, window , etc.

■ Repeating data with ngFor :



```
// Importing Component decorator from Angular core module
import { Component } from '@angular/core';

// Component decorator configuration
@Component({
  // Selector for the component to be used in HTML
  selector: 'events-list',
  // Template file path for the component
  templateUrl: './events-list.component.html',
})
// EventsListAppComponent class definition
export class EventsListAppComponent {
  // Sample event data
  events = [
    {
      id: 1,
      name: 'Angular Connect',
      date: '9/26/2036',
      time: '10:00 am',
      price: 599.99,
      imageUrl: '/assets/images/angularconnect-shield.png',
      location: {
        address: '1057 DT',
        city: 'London',
        country: 'England',
      },
      sessions: [ ···
      ],
    },
    {
      id: 2,
      name: 'ng-nl',
      date: '4/15/2037',
      time: '9:00 am',
      price: 950.0,
      imageUrl: '/assets/images/ng-nl.png',
      location: {
        address: 'The NG-NL Convention Center & Scuba Shop',
        city: 'Amsterdam',
        country: 'Netherlands',
      },
      sessions: [ ···
      ],
```



```
<!-- Container for Upcoming Angular Events -->

<div>
    <!-- Heading for Upcoming Angular Events -->
    <h2>Upcoming Angular Events</h2>
    <!-- Horizontal line as a visual separator -->
    <hr>

    <div class="row">
        <div *ngFor="let event of events" class="col-md-5">
            <!-- Note: This assumes that there is a component named 'event-thumbnail' -->
            <event-thumbnail [event]="event"></event-thumbnail>
            <!-- * indicates that this ngFor is a structural directives, structural directives are different from other directives
                 because structural directives acyually change the shape of DOM. They actually add or remove HTML elements from the
                 HTML document. & also we called structural directive because it will add an HTML element for each item in an array. -->
            <!-- -->
        </div>
    </div>
    <!-- Line break for better spacing -->
    <br>
</div>
```

■ Handling null values with the safe-navigation-operator :

- If you did not pass any event that means event is undefined, so in console we are able to see errors.

```
<div *ngFor="let event of events" class="col-md-5">
  <event-thumbnail></event-thumbnail>
```

- If event might be null then add safe-navigation-operator (?) after event.

```
<h2>{{event?.name}}</h2>
```

■ Hiding and Showing Content with ngIf :

- If Price is set then ngIf directive do nothing, but if the price is not set like undefined or null then this expression will evaluate false & hide the Price section for particular UI component.

```
<!-- Event price -->
<div *ngIf="event?.price">Price: \${{ event.price }}</div>
<!-- Location details -->
```

■ Hiding Content with the [Hidden] Binding :

```
<!-- Event price -->
<div [hidden]="!event?.price">Price: \${{ event.price }}</div>
```

- If price is null or undefined then above syntax will hide the price.

■ Hiding and Showing Content with ngSwitch :

```
<!-- Event time -->
<div [ngSwitch]="event?.time">
  Time: {{ event?.time }}
  <span *ngSwitchCase="'8:00 am'">(Early Start)</span>
  <span *ngSwitchCase="'10:00 am'">(Late Start)</span>
  <span *ngSwitchDefault>(Normal Start)</span>
</div>
```

■ Styling Components with ngClass :

- Class binding is good if you want to toggle a single class & ngClass Directive is better if you're wanting to toggle multiple classes.

- Class Binding :

```
<!-- Event time -->
<div [class.green]="event?.time === '8:00 am'" [ngSwitch]="event?.time">
  Time: {{ event?.time }}
  <span *ngSwitchCase="'8:00 am'">(Early Start)</span>
```

```
styles: [
  `
    .green {
      color: #003300 !important;
    }
    .bold {
      font-weight: bold;
    }
```

- ngClass Directive : Method : 1 –

```
styles: [
  `
    .green {
      color: #003300 !important;
    }
    .bold {
      font-weight: bold;
    }
```

```
<!-- Event time -->
<div
  [ngClass]="{
    green: event?.time === '8:00 am',
    bold: event?.time === '8:00 am'
  }"
  [ngSwitch]="event?.time"
>
  Time: {{ event?.time }}
  <span *ngSwitchCase="'8:00 am'">(Early Start)</span>
  <span *ngSwitchCase="'10:00 am'">(Late Start)</span>
  <span *ngSwitchDefault>(Normal Start)</span>
</div>
```

Method : 2 –

```
getStartTimeClass() {
  if (this.event && this.event?.time === '8:00 am') {
    return ['green', 'bold'];
  }
  return [];
}
```

```
<!-- Event time -->
<div [ngClass]="getStartTimeClass()" [ngSwitch]="event?.time">
  Time: {{ event?.time }}
  <span *ngSwitchCase="'8:00 am'">(Early Start)</span>
  <span *ngSwitchCase="'10:00 am'">(Late Start)</span>
  <span *ngSwitchDefault>(Normal Start)</span>
</div>
```

■ Styling Components with ngStyle :

- Method: 1 (No need to add css .green & .bold properties)

```
<!-- Event time -->
<div
  [ngStyle]="{
    color: event?.time === '8:00 am' ? '#FB889E' : '#bbb',
    'font-weight': event?.time === '8:00 am' ? 'bold' : 'normal'
  }"
  [ngSwitch]="event?.time"
>
  Time: {{ event?.time }}
  <span *ngSwitchCase="'8:00 am'">(Early Start)</span>
  <span *ngSwitchCase="'10:00 am'">(Late Start)</span>
  <span *ngSwitchDefault>(Normal Start)</span>
</div>
```

- Method: 2 (No need to add css .green & .bold properties)

```
<!-- Event time -->
<div [ngStyle]="getStartTimeStyle()" [ngSwitch]="event?.time">
  Time: {{ event?.time }}
  <span *ngSwitchCase="'8:00 am'">(Early Start)</span>
  <span *ngSwitchCase="'10:00 am'">(Late Start)</span>
  <span *ngSwitchDefault>(Normal Start)</span>
```

```
getStartTimeStyle() {
  if (this.event && this.event?.time === '8:00 am') {
    return { color: '#FB889E', 'font-weight': 'bold' };
  }
  return {};
}
```

■ Angular Services & Dependency Injection :

- Services allow you to define business logic in a separate file & then inject whatever service we need whenever we need it.

events-list.component.ts U

src > app > events > events-list.component.ts > ...

Click here to ask Blackbox to help you code faster

```ts
// Importing Component decorator from Angular core module
import { Component, OnInit } from '@angular/core';
import { EventService } from './shared/event.service';

// Component decorator configuration
@Component({
  // Selector for the component to be used in HTML
  selector: 'events-list',
  // Template file path for the component
  templateUrl: './events-list.component.html',
})
// EventsListAppComponent class definition
export class EventsListAppComponent implements OnInit {
  events: any[] = [];

  constructor(private eventService: EventService) {}

  ngOnInit() {
    this.events = this.eventService.getEvents();
  }
}
```

event.service.ts U

src > app > events > shared > event.service.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { Injectable } from '@angular/core';

@Injectable()
export class EventService {
  getEvents() {
    return EVENTS;
  }
}
// Sample event data
const EVENTS = [
  {
    id: 1,
    name: 'Angular Connect',
    date: '9/26/2036',
    time: '10:00 am',
    price: 599.99,
    imageUrl: '/assets/images/angularconnect-shield.png',
    location: {
      address: 'Varun Park Society',
      city: 'Pune',
      country: 'India',
    },
    sessions: [
      {
```

■ Adding Routes :

events-app.component.html U

src > app > events-app.component.html > router-outlet

Go to component | Click here to ask Blackbox to help you code faster

```
<nav-bar></nav-bar>
<!-- Router outlet for displaying routed components -->
<router-outlet></router-outlet>
```

- Angular router-outlet components helps us to when a user requests a particular URL, display its corresponding component here.

- The `<router-outlet>` element is used as a placeholder where Angular will dynamically render components based on the defined routes in your application. Components associated with different routes will be displayed within this outlet.

- Create routes.ts file inside src – app directory :
1. line says that if the url matches /events, then show this EventsListAppComponent wherever our router-outlet component is.

2. line says that if the url matches /events/1, then show this EventDetailsComponent.

3. line says that the default route configuration, which redirects to the 'events' route if the path is empty.

routes.ts U

src > app > routes.ts > ...

Click here to ask Blackbox to help you code faster

```
// Importing necessary components
import { Routes } from '@angular/router';
import { EventDetailsComponent } from './events/event-details/event-details.component';
import { EventsListAppComponent } from './events/events-list.component';

// Configuration of Angular routes using the RouterModule
export const appRoutes: Routes = [
  // Route for displaying the list of events
  { path: 'events', component: EventsListAppComponent },

  // Route for displaying detailed information about a specific event
  { path: 'events/:id', component: EventDetailsComponent },

  // Default route: Redirect to the 'events' route if the path is empty
  { path: '', redirectTo: 'events', pathMatch: 'full' },
];
```

- No need of selector inside events-list.component.ts file.

event-details.component.ts U

src > app > events > event-details > event-details.component.ts > ...

```
export class EventDetailsComponent implements OnInit {
  event: any;

  constructor(
    private eventService: EventService,
    private route: ActivatedRoute
  ) {}

  ngOnInit() {
    this.event = this.eventService.getEvent(+this.route.snapshot.params['id']);
  }
}
```

- So now next task is to when we click on webpage element then it will go on particular page. i.e. linking to routes.

events-thumbnail.component.ts U

src > app > events > events-thumbnail.component.ts > EventsThumbnailAppComponent

```
11      <div [routerLink]="['/events', event.id]" class="well hoverwell thumbnail">
```

- Now go back when we click on All Events

nav-bar.component.html U

src > app > nav > nav-bar.component.html > div.navbar.navbar-default > div.container-fluid

```
17                  <li><a [routerLink]="['/events']">All Events</a></li>
18                  <li><a href="">Create Event</a></li>
```

- Now let's take a look at how we would navigate to a page from within our code. & create new event :

routes.ts U

src > app > routes.ts > ...

```
9   // Route for creating a new event
10  { path: 'events/new', component: CreateEventComponent },
11
```

create-event.component.ts U

src > app > events > create-event.component.ts > ...

Click here to ask Blackbox to help you code faster

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     template: `
5       <h1>New Event</h1>
6       <hr />
7       <div class="div col-md-6">
8         <h3>[Create Event Form will go here]</h3>
9         <br />
10        <br />
11        <button type="submit" class="btn btn-primary">Submit</button>  
12        <button type="button" class="btn btn-default">Cancel</button>
13      </div>
14    `,
15  })
16  export class CreateEventComponent {}
```

- If we click on Create Event section on navbar then we are able to open create new event page.

nav-bar.component.html U

src > app > nav > nav-bar.component.html > div.navbar.navbar-default > div.container-fluid > div

```
18                  <li><a [routerLink]="['/events/new']">Create Event</a></li>
```

- Now when we click on cancel button on create new event page, we want to back on event list page.

create-event.component.ts U

src > app > events > create-event.component.ts > ...

```
13          <button type="button" class="btn btn-default" (click)="cancel()">
14            Cancel
15          </button>
16        </div>
17      `,
18    })
19    export class CreateEventComponent {
20      constructor(private router: Router) {}
21
22      cancel() {
23        this.router.navigate(['/events']);
24      }
25    }
```

- Guarding against route Activation : Sometimes we want to prevent a user from going to a particular page them from leaving a page that's why route guards come in picture.

- canActivate allows us to determine whether or not a user navigate to a route.

- But if I give [localhost:4200/events/42] then we want to add route guard in here to redirect them to a 404 page if event ID is not valid. because there is no data is there.

routes.ts U

src > app > routes.ts > ...

```ts
  // Route for displaying detailed information about a specific event
  {
    path: 'events/:id',
    component: EventDetailsComponent,
    canActivate: [EventRouteActivator], // Activate the route activator for this route
  },

  // Default route: Redirect to the 'events' route if the path is empty
  { path: '', redirectTo: '/events', pathMatch: 'full' },

  // Route for displaying the 404 page
  {
    path: '404',
    component: Error404Component,
  },
];
```

event-route-activator.service.ts U

src > app > events > event-details > event-route-activator.service.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot, CanActivate, Router } from '@angular/router';
import { EventService } from '../shared/event.service';

@Injectable()
export class EventRouteActivator implements CanActivate {
  constructor(private eventService: EventService, private router: Router) {}

  canActivate(route: ActivatedRouteSnapshot): boolean {
    // Checking if the event with the given ID exists
    const eventExist = !!this.eventService.getEvent(+route.params['id']);

    if (!eventExist) {
      // Redirecting to the 404 page if the event doesn't exist
      this.router.navigate(['/404']);
    }

    return eventExist; // Allowing access if the event exists
  }
}
```

404.component.ts U

src > app > errors > 404.component.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { Component } from '@angular/core';

@Component({
  template: `
    <section>
      <div class="container">
        <div class="text">
          <h1>404 Page Not Found..!!</h1>
          <div>
            <img class="image" src="./assets/Images/404.jpg" alt="" />
          </div>
          <br />
          <p>
            We can't seem to find the page you're looking for. <br />Please
            check the URL for any typos.
          </p>
          <ul class="menu">
            <li><a href="#">Go to Homepage</a></li>
          </ul>
        </div>
      </div>
    </section>
  `,
  styles: [...
  ],
})
export class Error404Component {
  constructor() {}
}
```

- Guarding against route De-Activation : Just like we used canActivate to prevent the user from navigating to a page, we can use canDeactivate to prevent a user from leaving a page.

- This is useful when if for example you want to warn a user if they try to navigate away from a page before saving their data.

- let's add a route guard to the Create Event Page that's warns the user, if they try to cancel before saving their event.

- There is 2 ways to add route guards, we can either use function OR service. In canActivate example, we have user service, now we can use different method. & declare function inside app.modules.ts file as a longhand inside providers.

- routes.ts app.mod creteOvenet.com.

routes.ts U

src > app > routes.ts > ...

```ts
// Route for creating a new event
{
  path: 'events/new',
  component: CreateEventComponent,
  canDeactivate: ['canDeactivateCreateEvent'], // De-Activate the route activator for this route
},
```

create-event.component.ts U

src > app > events > create-event.component.ts > CreateEvent

```ts
export class CreateEventComponent {
  isDirty: boolean = true;
```

app.module.ts M

src > app > app.module.ts > ...

```ts
// Providers: Services to be provided at the module level
providers: [
  EventService,
  ToastrService,
  EventRouteActivator,
  { provide: 'canDeactivateCreateEvent', useValue: checkDirtyState },
],

// Bootstrap: The root component that Angular creates and inserts into the index.html file
bootstrap: [EventsAppComponent],
})
// AppModule class definition
export class AppModule {}

export function checkDirtyState(component: CreateEventComponent) {
  if (component.isDirty) {
    return window.confirm(
      'You have not saved this event, do you really want to cancel?'
    );
  }
  return true;
}
```

- Styling Active Links : Just we want to highlight currently active link in our navbar so that users can see from the navbar which section of the site they're on.

nav-bar.component.html U

src > app > nav > nav-bar.component.html > div.navbar.navbar-default > div.container-fluid > div.collapse.navbar-collapse > div.navbar-

```html
<!-- Navigation links -->
<ul class="nav navbar-nav">
    <li><a [routerLink]="['/events']" routerLinkActive="active"
            [routerLinkActiveOptions]="{exact: true}">All Events</a></li>
    <li><a [routerLink]="['/events/new']" routerLinkActive="active">Create Event</a></li>
```

nav.component.ts U

src > app > nav > nav.component.ts > ...

```
      li > a.active {
        color: #adff2f;
      }
    `,
  ],
})
```

- Lazily Loading Feature Modules :

profile.component.ts U

src > app > user > profile.component.ts > ...

Click here to ask Blackbox to help you code faster

```
import { Component } from '@angular/core';

@Component({
  template: `
    <h1>Edit Your Profile</h1>
    <hr />
    <div class="col-md-6">
      <h3>[Edit profile form will go here]</h3>
      <br />
      <br />
      <button type="submit" class="btn btn-primary">Save</button> 
      <button type="button" class="btn btn-default">Cancel</button>
    </div>
  `,
})
export class ProfileComponent {}
```

user.module.ts U

src > app > user > user.module.ts > ...

Click here to ask Blackbox to help you code faster

```
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { RouterModule } from '@angular/router';
import { ProfileComponent } from './profile.component';
import { userRoutes } from './user.routes';

@NgModule({
  imports: [CommonModule, RouterModule.forChild(userRoutes)],
  declarations: [ProfileComponent],
  providers: [],
})
export class UserModule {}
```

user.routes.ts U

src > app > user > user.routes.ts > ...

Click here to ask Blackbox to help you code faster

```
import { ProfileComponent } from './profile.component';

export const userRoutes = [{ path: 'profile', component: ProfileComponent }];

```

nav-bar.component.html U

src > app > nav > nav-bar.component.html > div.navbar.navbar-default

```
        <!-- Right-aligned elements -->
        <div class="navbar-header navbar-right">
            <ul class="nav navbar-nav">
                <li><a [routerLink]="['user/profile']">Welcome Yogesh</a></li>
            </ul>
        </div>
```

routes.ts U

src > app > routes.ts > ...

```ts
// When a route start with '/user', load the UserModule from './user/user.module' this path.
// The lazy loaded module will be inject
{
  path: 'user',
  loadChildren: () => import('./user/user.module').then((m) => m.UserModule),
},
];
```

- Organizing your exports with barrels :

routes.ts U

src > app > routes.ts > ...

```ts
import {
  EventDetailsComponent,
  EventsListAppComponent,
  CreateEventComponent,
  EventRouteActivator,
  EventListResolver,
} from './events/index';
```

app.module.ts M

src > app > app.module.ts > ...

```ts
import {
  EventsListAppComponent,
  EventsThumbnailAppComponent,
  EventService,
  EventDetailsComponent,
  CreateEventComponent,
  EventRouteActivator,
  EventListResolver,
} from './events/index';
```

index.ts U

src > app > events > index.ts

Click here to ask Blackbox to help you code faster

```ts
export * from './create-event.component';
export * from './events-thumbnail.component';
export * from './events-list-resolver.service';
export * from './events-list.component';
export * from './shared/index';
export * from './event-details/index';
```

index.ts U

src > app > events > shared > index.ts

Click here to ask Blackbox to help you code faster

```ts
export * from './event.service';
```

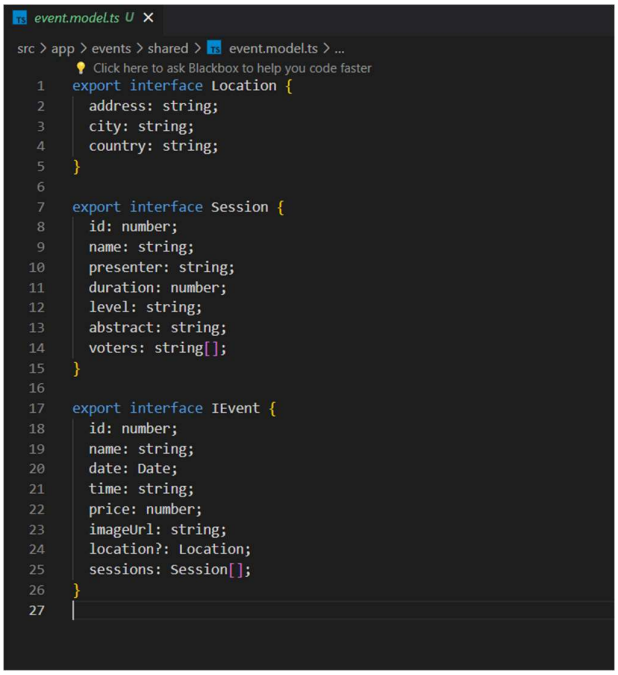
■ Angular Forms & Validation :

- Use Models for Type Safety :

event-details.component.ts U

src > app > events > event-details > event-details.component.ts > ...

```ts
export class EventDetailsComponent implements OnInit {
  /* If you're confident that the event property will be assigned a value during the ngOnInit lifecycle,
  you can use the non-null assertion operator (!) to tell TypeScript not to worry about initialization. */
  event!: IEvent;
```



⭐ Template-Based Forms :

- Allow you to build your form completely in your HTML Template. It is simple & easy, use for simple case

- Limitations : 1. For complex form want to add lot of logic in html & also cross-field validations is difficult.
2. You cannot unit test all your forms & validation logic if you use template-based forms.

Feature	Template-Driven Forms	Reactive Forms
Form Creation	Generated automatically from the template.	Created programmatically in the component.
Form Logic	Minimal logic in the template.	More logic in the component.
Two-Way Data Binding	Makes use of ngModel for two-way data binding.	No built-in two-way data binding, more control.
Data Model	Automatically creates a data model based on the template.	Manually define the data model in the component.
Dynamic Forms	More challenging for dynamic forms.	Easier to create dynamic forms programmatically.
Form Validation	Template-driven validation directives (ngModel, ngModelGroup, etc.).	Programmatically define validators in the component.
Sync and Async Validation	Supports both sync and async validation.	Supports both sync and async validation.
Unit Testing	Typically easier to unit test due to less code in the component.	May require more effort in unit testing.
Dynamic Form Control	Limited control over dynamic form controls.	Full control over dynamic form controls.
Complexity	Suitable for simple forms with less complexity.	Suitable for complex forms with more flexibility.
Learning Curve	Lower learning curve, especially for beginners.	Steeper learning curve but more powerful and flexible.
Developer Preference	Preferred by developers who like less code and automatic binding.	Preferred by developers who prefer explicit control and flexibility.

- *(input)="userName=$event.target.value"* : When input is entered into an input box, an event is fired, & that event updates the userName property of our component to be value of this input box.

But that's difficult to remember & add syntax for lots of elements on form, for that reason angular provides the ngModel directive that will allow us to bind input filed like this.

- [(ngModel)] :

Here Parenthesis are used to bind in the html to component-direction, & typically used for responding to events.

Whereas Square Brackets are used to bind in the component to html-direction, & typically used for displaying data from component on page.

And this is 2-way binding.

- ngModel name name attribute always that is a same as ngModel & name attribute is same value.

- Add FormsModule inside imports → user.module.ts file. & AuthService inside app.modules.ts file.

user.routes.ts U

src > app > user > user.routes.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { LoginComponent } from './login.component';
import { ProfileComponent } from './profile.component';

export const userRoutes = [
  { path: 'profile', component: ProfileComponent },
  { path: 'login', component: LoginComponent },
];
```

- We have created our form, also take data & validated also.

login.component.html U

src > app > user > login.component.html > ...

```html
<h1>Login</h1>
<hr>
<div class="col-md-4">
    <form autocomplete="off" #loginForm="ngForm" (ngSubmit)="login(loginForm.value)" novalidate>
        <div class="form-group">
            <label for="userName">User Name:</label>
            <em
                *ngIf="loginForm.controls['userName']?.invalid && (loginForm.controls['userName']?.touched || mouseoverLogin) ">Required..!!</em>
            <input (ngModel)=" userName" name="userName" required id="userName" type="text" class="form-control"
                placeholder="User Name..." />
        </div>
        <div class="form-group">
            <label for="password">Password:</label>
            <em
                *ngIf="loginForm.controls['password']?.invalid && (loginForm.controls['password']?.touched || mouseoverLogin)">Required..!!</em>
            <input (ngModel)="password" name="password" required id="password" type="password" class="form-control"
                placeholder="Password..." />
        </div>

        <span (mouseenter)="mouseoverLogin=true" (mouseleave)="mouseoverLogin=false">
            <button type="submit" [disabled]="loginForm.invalid" class="btn btn-primary">Login</button> 
        </span>
        <button type="button" (click)="cancel()" class="btn btn-default">Cancel</button>
    </form>
</div>
```

user.model.ts U

src > app > user > user.model.ts > ...

Click here to ask Blackbox to help you code faster

```ts
export interface IUser {
  id: number;
  firstName: string;
  lastName: string;
  userName: string;
}
```

login.component.ts U

src > app > user > login.component.ts > ...

```
@Component({
  templateUrl: './login.component.html',
  styles: [
    `
      em {
        float: right;
        color: #ffff00;
        padding-left: 10px;
        font-size: 13px;
      }
    `,
  ],
})
export class LoginComponent {
  userName: any;
  password: any;
  mouseoverLogin: any;

  constructor(private authService: AuthService, private router: Router) {}

  login(formValues: any) {
    this.authService.loginUser(formValues.userName, formValues.password);
    this.router.navigate(['events']);
  }

  cancel() {
    this.router.navigate(['events']);
  }
}
```

auth.service.ts U

src > app > user > auth.service.ts > ...

Click here to ask Blackbox to help you code faster

```
import { Injectable } from '@angular/core';
import { IUser } from './user.model';

@Injectable()
export class AuthService {
  // Property to hold the current user information
  currentUser: IUser | undefined;

  // Method to simulate user login
  loginUser(userName: string, password: string) {
    // Set the current user with dummy data
    this.currentUser = {
      id: 1,
      userName: 'byogesh',
      firstName: 'Yogesh',
      lastName: 'Borole',
    };
  }

  // Method to check if a user is authenticated
  isAuthenticated() {
    return !!this.currentUser;
  }
}
```

⭐ Model-Based OR Reactive Forms :

- Model based or reactive forms allow you to build your form & put a lot of your logic in your component instead of in the template.

- Here below we have created a reactive form & also added validations to the same. Validations includes not able to save the form if for is invalid.

- Also we have used Multiple validators in reactive forms :

https://angular.io/api/forms/Validators

profile.component.html U

src > app > user > profile.component.html > div

Go to component | Click here to ask Blackbox to help you code faster

```html
<div>
    <h1>Edit Your Profile</h1>
    <hr>
    <div class="col-md-4">
        <form [formGroup]="profileForm" (ngSubmit)="saveProfile(profileForm.value)" autocomplete="off" novalidate>
            <div class="form-group" [ngClass]="{'error': !validateFirstName()}">
                <label for="firstName">First Name:</label>
                <em *ngIf="!validateFirstName() && profileForm.get('firstName')?.hasError('required')">Required..!!</em>
                <em *ngIf="!validateFirstName() && profileForm.get('firstName')?.hasError('pattern')">Must start with a
                    letter..!!</em>
                <input formControlName="firstName" id="firstName" type="text" class="form-control"
                    placeholder="First Name..." />
            </div>
            <div class="form-group" [ngClass]="{'error': !validateLastName()}">
                <label for="lastName">Last Name:</label>
                <em *ngIf="!validateLastName()">Required..!!</em>
                <input formControlName="lastName" id="lastName" type="text" class="form-control"
                    placeholder="Last Name..." />
            </div>

            <button type="submit" class="btn btn-primary">Save</button> 
            <button type="button" (click)="cancel()" class="btn btn-default">Cancel</button>
        </form>
    </div>
</div>
```

profile.component.ts U

src > app > user > profile.component.ts > ProfileComponent

Click here to ask Blackbox to help you code faster

```ts
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';
import { Router } from '@angular/router';
import { AuthService } from './auth.service';

@Component({
  templateUrl: './profile.component.html',
  styles: [
    `
      em {
        float: right;
        color: #fff000;
        padding-left: 10px;
        font-size: 13px;
      }
      .error input {
        background-color: #e3c3c5;
      }
      .error ::-webkit-input-placeholder {
        color: #999;
      }
      .error ::-moz-placeholder {
        color: #999;
      }
      .error :-moz-placeholder {
        color: #999;
      }
      .error :ms-input-placeholder {
        color: #999;
      }
    `,
  ],
})
```

profile.component.ts U

src > app > user > profile.component.ts > ProfileComponent

```ts
export class ProfileComponent implements OnInit {
  profileForm!: FormGroup;
  firstName!: FormControl;
  lastName!: FormControl;

  constructor(private authService: AuthService, private router: Router) {}

  ngOnInit() {
    this.firstName = new FormControl(this.authService.currentUser.firstName, [
      Validators.required,
      Validators.pattern('[a-zA-Z].*'),
    ]);

    this.lastName = new FormControl(
      this.authService.currentUser.lastName,
      Validators.required
    );

    this.profileForm = new FormGroup({
      firstName: this.firstName,
      lastName: this.lastName,
    });
  }

  saveProfile(formValues: any) {
    if (this.profileForm.valid) {
      this.authService.updateCurrentUser(
        formValues.firstName,
        formValues.lastName
      );
      this.router.navigate(['events']);
    }
  }

  cancel() {
    this.router.navigate(['events']);
  }

  validateFirstName() {
    return this.firstName.valid || this.firstName.untouched;
  }

  validateLastName() {
    return this.lastName.valid || this.lastName.untouched;
  }
}
```

➞ The difference between touched & dirty is that if I just put my cursor into a field & then leave that field, it is touched, but no dirty. But if I start typing in a field, then it becomes dirty.

➞ Below I have implemented Using multiple validators & custom Validators :

create-session.component.html U

src > app > events > event-details > create-session.component.html > div.col-md-6

Go to component | Click here to ask Blackbox to help you code faster

```
<div class="col-md-12">
    <h3>Create Session</h3>
</div>
<div class="col-md-6">
    <form [formGroup]="newSessionForm" (ngSubmit)="saveSession(newSessionForm.value)" autocomplete="off">
        <div class="form-group" [ngClass]="{'error': name.invalid && name.dirty}">
            <label for="sessionName">Session Name:</label>
            <em *ngIf="name.invalid && name.dirty">Required..!!</em>
            <input formControlName="name" id="sessionName" type="text" class="form-control"
                placeholder="session name..." />
        </div>
        <div class="form-group" [ngClass]="{'error': presenter.invalid && presenter.dirty}">
            <label for="eventDate">Presenter:</label>
            <em *ngIf="presenter.invalid && presenter.dirty">Required..!!</em>
            <input formControlName="presenter" id="presenter" type="text" class="form-control"
                placeholder="presenter..." />
        </div>
        <div class="form-group" [ngClass]="{'error': duration.invalid && duration.dirty}">
            <label for="duration">Duration:</label>
            <em *ngIf="duration.invalid && duration.dirty">Required..!!</em>
            <select formControlName="duration" class="form-control">
                <option value="">select duration...</option>
                <option value="1">Half Hour</option>
                <option value="2">1 Hour</option>
                <option value="3">Half Day</option>
                <option value="4">Full Day</option>
            </select>
        </div>
        <div class="form-group" [ngClass]="{'error': level.invalid && level.dirty}">
            <label for="level">Level:</label>
            <em *ngIf="level.invalid && level.dirty">Required..!!</em>
            <select formControlName="level" class="form-control">
                <option value="">select level...</option>
                <option value="Beginner">Beginner</option>
                <option value="Intermediate">Intermediate</option>
                <option value="Advanced">Advanced</option>
            </select>
        </div>
        <div class="form-group" [ngClass]="{'error': abstract.invalid && abstract.dirty}">
            <label for="abstract">Abstract:</label>
            <em *ngIf="abstract.invalid && abstract.dirty && abstract?.errors?.['required']">Required..!!</em>
            <em *ngIf="abstract.invalid && abstract.dirty && abstract?.errors?.['maxlength']">Cannot exceed 400
                characters..!!</em>
            <em *ngIf="abstract.invalid && abstract.dirty && abstract?.errors?.['restrictedWords']">Restricted word
                found: {{abstract.errors?.['restrictedWords']}}</em>
            <textarea formControlName="abstract" id="abstract" rows="3" class="form-control"
                placeholder="abstract..."></textarea>
        </div>

        <button type="submit" [disabled]="newSessionForm.invalid" class="btn btn-primary">Save</button> 
        <button type="button" class="btn btn-default">Cancel</button>
    </form>
</div>
```

restricted-words.validator.ts U

src > app > events > shared > restricted-words.validator.ts > ...

Click here to ask Blackbox to help you code faster

```
import { FormControl } from '@angular/forms';

export function restrictedWords(words: any[]) {
  return (control: FormControl): { [key: string]: any } | null => {
    if (!words) return null; // If control is empty, no error

    var invalidWords = words
      .map((w: any) => (control.value.includes(w) ? w : null))
      .filter((w: null) => w != null);

    return invalidWords && invalidWords.length > 0
      ? { restrictedWords: invalidWords.join(', ') }
      : null;
  };
}

```

index.ts U

src > app > events > shared > index.ts

Click here to ask Blackbox to help you code faster

```
export * from './event.service';
export * from './event.model';
export * from './restricted-words.validator';

```

```
{
  path: 'events/sessions/new',
  component: CreateSessionComponent,
},
```

create-session.component.ts U

src > app > events > event-details > create-session.component.ts > ...

Click here to ask Blackbox to help you code faster

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';
import { restrictedWords, Session } from '../shared';

@Component({
  templateUrl: './create-session.component.html',
  styles: [
    `
      em {
        float: right;
        color: #ffff00;
        padding-left: 10px;
        font-size: 13px;
      }
      .error input,
      .error select,
      .error textarea {
        background-color: #e3c3c5;
      }
      .error ::-webkit-input-placeholder {
        color: #999;
      }
      .error ::-moz-placeholder {
        color: #999;
      }
      .error :-moz-placeholder {
        color: #999;
      }
      .error :ms-input-placeholder {
        color: #999;
      }
    `,
  ],
})
export class CreateSessionComponent implements OnInit {
  newSessionForm!: FormGroup;
  name!: FormControl;
  presenter!: FormControl;
  duration!: FormControl;
  level!: FormControl;
  abstract!: FormControl;
```

```
  ngOnInit(): void {
    this.name = new FormControl('', Validators.required);
    this.presenter = new FormControl('', Validators.required);
    this.duration = new FormControl('', Validators.required);
    this.level = new FormControl('', Validators.required);
    this.abstract = new FormControl('', [
      Validators.required,
      Validators.maxLength(400),
      restrictedWords(['foo', 'bar']),
    ]);

    this.newSessionForm = new FormGroup({
      name: this.name,
      presenter: this.presenter,
      duration: this.duration,
      level: this.level,
      abstract: this.abstract,
    });
  }

  saveSession(formValues: any) {
    let session: Session = {
      name: formValues.name,
      presenter: formValues.presenter,
      duration: +formValues.duration,
      level: formValues.level,
      abstract: formValues.abstract,
      voters: [],
    };

    console.log(session);
  }
}
```

---

profile.component.html U

src > app > user > profile.component.html > div

Go to component | Click here to ask Blackbox to help you code faster

```html
<div>
    <h1>Edit Your Profile</h1>
    <hr>
    <div class="col-md-4">
        <form [formGroup]="profileForm" (ngSubmit)="saveProfile(profileForm.value)" autocomplete="off" novalidate>
            <div class="form-group" [ngClass]="{'error': !validateFirstName()}">
                <label for="firstName">First Name:</label>
                <em *ngIf="!validateFirstName() && profileForm.get('firstName')?.hasError('required')">Required..!!</em>
                <em *ngIf="!validateFirstName() && profileForm.get('firstName')?.hasError('pattern')">Must start with a
                    letter..!!</em>
                <input formControlName="firstName" id="firstName" type="text" class="form-control"
                    placeholder="First Name..." />
            </div>
            <div class="form-group" [ngClass]="{'error': !validateLastName()}">
                <label for="lastName">Last Name:</label>
                <em *ngIf="!validateLastName()">Required..!!</em>
                <input formControlName="lastName" id="lastName" type="text" class="form-control"
                    placeholder="Last Name..." />
            </div>

            <button type="submit" class="btn btn-primary">Save</button> 
            <button type="button" (click)="cancel()" class="btn btn-default">Cancel</button>
        </form>
    </div>
</div>
```

profile.component.ts U

src > app > user > profile.component.ts > ProfileComponent

Click here to ask Blackbox to help you code faster

```ts
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';
import { Router } from '@angular/router';
import { AuthService } from './auth.service';

@Component({
  templateUrl: './profile.component.html',
  styles: [
    `
      em {
        float: right;
        color: #ffff00;
        padding-left: 10px;
        font-size: 13px;
      }
      .error input {
        background-color: #e3c3c5;
      }
      .error ::-webkit-input-placeholder {
        color: #999;
      }
      .error ::-moz-placeholder {
        color: #999;
      }
      .error :-moz-placeholder {
        color: #999;
      }
      .error :ms-input-placeholder {
        color: #999;
      }
    `,
  ],
})
export class ProfileComponent implements OnInit {
  profileForm!: FormGroup;
  firstName!: FormControl;
  lastName!: FormControl;

  constructor(private authService: AuthService, private router: Router) {}
```

```ts
  ngOnInit() {
    this.firstName = new FormControl(this.authService.currentUser.firstName, [
      Validators.required,
      Validators.pattern('[a-zA-Z].*'),
    ]);

    this.lastName = new FormControl(
      this.authService.currentUser.lastName,
      Validators.required
    );

    this.profileForm = new FormGroup({
      firstName: this.firstName,
      lastName: this.lastName,
    });
  }

  saveProfile(formValues: any) {
    if (this.profileForm.valid) {
      this.authService.updateCurrentUser(
        formValues.firstName,
        formValues.lastName
      );
      this.router.navigate(['events']);
    }
  }

  cancel() {
    this.router.navigate(['events']);
  }

  validateFirstName() {
    return this.firstName.valid || this.firstName.untouched;
  }

  validateLastName() {
    return this.lastName.valid || this.lastName.untouched;
  }
}
```

user.module.ts U

src > app > user > user.module.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { RouterModule } from '@angular/router';
import { LoginComponent } from './login.component';
import { ProfileComponent } from './profile.component';
import { userRoutes } from './user.routes';

@NgModule({
  imports: [
    CommonModule,
    ReactiveFormsModule,
    FormsModule,
    RouterModule.forChild(userRoutes),
  ],
  declarations: [ProfileComponent, LoginComponent],
  providers: [],
})
export class UserModule {}

```

user.routes.ts U

src > app > user > user.routes.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { LoginComponent } from './login.component';
import { ProfileComponent } from './profile.component';

export const userRoutes = [
  { path: 'profile', component: ProfileComponent },
  { path: 'login', component: LoginComponent },
];

```

auth.service.ts U

src > app > user > auth.service.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { Injectable } from '@angular/core';
import { IUser } from './user.model';

@Injectable()
export class AuthService {
  // Property to hold the current user information
  currentUser!: IUser;

  // Method to simulate user login
  loginUser(userName: string, password: string) {
    this.currentUser = {
      id: 1,
      userName: userName,
      firstName: 'Yogesh',
      lastName: 'Borole',
    };
  }

  // Method to check if a user is authenticated
  isAuthenticated() {
    return !!this.currentUser;
  }

  updateCurrentUser(firstName: string, lastName: string) {
    this.currentUser.firstName = firstName;
    this.currentUser.lastName = lastName;
  }
}

```

login.component.html U

src > app > user > login.component.html > ...

Go to component | Click here to ask Blackbox to help you code faster

```html
<h1>Login</h1>
<hr>
<div class="col-md-4">
    <form autocomplete="off" #loginForm="ngForm" (ngSubmit)="login(loginForm.value)" novalidate>
        <div class="form-group">
            <label for="userName">User Name:</label>
            <em
                *ngIf="loginForm.controls['userName']?.invalid && (loginForm.controls['userName']?.touched || mouseoverLogin) ">Required..!!</em>
            <input (ngModel)=" userName" name="userName" required id="userName" type="text" class="form-control"
                placeholder="User Name..." />
        </div>
        <div class="form-group">
            <label for="password">Password:</label>
            <em
                *ngIf="loginForm.controls['password']?.invalid && (loginForm.controls['password']?.touched || mouseoverLogin)">Required..!!</em>
            <input (ngModel)="password" name="password" required id="password" type="password" class="form-control"
                placeholder="Password..." />
        </div>

        <span (mouseenter)="mouseoverLogin=true" (mouseleave)="mouseoverLogin=false">
            <button type="submit" [disabled]="loginForm.invalid" class="btn btn-primary">Login</button> 
        </span>
        <button type="button" (click)="cancel()" class="btn btn-default">Cancel</button>
    </form>
</div>

```

login.component.ts U

src > app > user > login.component.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from './auth.service';

@Component({
  templateUrl: './login.component.html',
  styles: [
    `
      em {
        float: right;
        color: #ffff00;
        padding-left: 10px;
        font-size: 13px;
      }
    `,
  ],
})
export class LoginComponent {
  userName: any;
  password: any;
  mouseoverLogin: any;

  constructor(private authService: AuthService, private router: Router) {}

  login(formValues: any) {
    this.authService.loginUser(formValues.userName, formValues.password);
    this.router.navigate(['events']);
  }

  cancel() {
    this.router.navigate(['events']);
  }
}
```

■ Communicating Between Components :

⭐ Passing Data into a Child Component :

- Our Event Details page is showing basic info about each event, but there is something missing here. Like sessions data so here we work on getting those displayed.

event-details.component.html U

src > app > events > event-details > event-details.component.html > div.container

```html
            </address>
          </div>
        </div>

        <hr>
        <session-list [sessions]="event?.sessions || []"></session-list>
    </div>
```

session-list.component.ts U

src > app > events > event-details > session-list.component.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { Component, Input } from '@angular/core';
import { Session } from '../shared/index';

@Component({
  selector: 'session-list',
  templateUrl: './session-list.component.html',
})
export class SessionListComponent {
  @Input() sessions!: Session[];
}
```

session-list.component.html U

src > app > events > event-details > session-list.component.html > div.row

Go to component | Click here to ask Blackbox to help you code faster

```html
<div class="row" *ngFor="let session of sessions">
    <div class="col-md-10">
        <div class="well">
            <h4>{{session.name}}</h4>
            <h6>{{session.presenter}}</h6>
            <span>Duration: {{session.duration}}</span><br />
            <span>Level: {{session.level}}</span>
            <p>{{session.abstract}}</p>
        </div>
    </div>
</div>
```

index.ts U

src > app > events > event-details > index.ts

```ts
export * from './session-list.component';
```

⭐ Passing Data out of a Child Component :

- We have created create-session component previously so now we can add functionality to add sessions in our Event Details page.

event-details.component.html U

src > app > events > event-details > event-details.component.html > div.container

```html
    </div>

    <hr>
    <div class="row">
        <div class="col-md-2">
            <h3 style="margin:0">Sessions</h3>
        </div>
        <div class="col-md-2">
            <button (click)="addSession()">Add Session</button>
        </div>
        <br>
    </div>
    <br>
    <session-list *ngIf="!addMode" [sessions]="event?.sessions || []"></session-list>
    <create-session *ngIf="addMode" (saveNewSession)="saveNewSession($event)"
        (cancelAddSession)="cancelAddSession()"></create-session>
</div>
```

event-details.component.ts U

src > app > events > event-details > event-details.component.ts > ...

```ts
addSession() {
  this.addMode = true;
}

saveNewSession(session: Session) {
  // Check if this.event or this.event.sessions is undefined
  if (!this.event || !this.event.sessions) {
    console.error('Event or sessions array is undefined.');
    return;
  }

  // Get the maximum session ID or default to 0 if sessions array is empty
  const nextId = Math.max(...this.event.sessions.map((s) => s.id ?? 0)) + 1;

  // Assign the next ID to the session
  session.id = nextId;

  // Push the new session to the sessions array
  this.event.sessions.push(session);

  // Update the event using the event service
  this.eventService.updateEvent(this.event);

  // Set addMode to false to exit the add mode
  this.addMode = false;
}

cancelAddSession() {
  this.addMode = false;
}
}
```

event.service.ts U

src > app > events > shared > event.service.ts > ...

```ts
    updateEvent(event: IEvent) {
      let index = EVENTS.findIndex((x) => x.id == event.id);
      EVENTS[index] = event;
    }
  }
```

create-session.component.html U

src > app > events > event-details > create-session.component.html > div.col-md-6 > form > button.btn.btn-default

Go to component | Click here to ask Blackbox to help you code faster

```html
<div class="col-md-12">
    <h3>Create Session</h3>
</div>
<div class="col-md-6">
    <form [formGroup]="newSessionForm" (ngSubmit)="saveSession(newSessionForm.value)" autocomplete="off">
        <div class="form-group" [ngClass]="{'error': name.invalid && name.dirty}">
            <label for="sessionName">Session Name:</label>
            <em *ngIf="name.invalid && name.dirty">Required..!!</em>
            <input formControlName="name" id="sessionName" type="text" class="form-control"
                placeholder="session name..." />
        </div>
        <div class="form-group" [ngClass]="{'error': presenter.invalid && presenter.dirty}">
            <label for="eventDate">Presenter:</label>
            <em *ngIf="presenter.invalid && presenter.dirty">Required..!!</em>
            <input formControlName="presenter" id="presenter" type="text" class="form-control"
                placeholder="presenter..." />
        </div>
        <div class="form-group" [ngClass]="{'error': duration.invalid && duration.dirty}">
            <label for="duration">Duration:</label>
            <em *ngIf="duration.invalid && duration.dirty">Required..!!</em>
            <select formControlName="duration" class="form-control">
                <option value="">select duration...</option>
                <option value="1">Half Hour</option>
                <option value="2">1 Hour</option>
                <option value="3">Half Day</option>
                <option value="4">Full Day</option>
            </select>
        </div>
        <div class="form-group" [ngClass]="{'error': level.invalid && level.dirty}">
            <label for="level">Level:</label>
            <em *ngIf="level.invalid && level.dirty">Required..!!</em>
            <select formControlName="level" class="form-control">
                <option value="">select level...</option>
                <option value="Beginner">Beginner</option>
                <option value="Intermediate">Intermediate</option>
                <option value="Advanced">Advanced</option>
            </select>
        </div>
        <div class="form-group" [ngClass]="{'error': abstract.invalid && abstract.dirty}">
            <label for="abstract">Abstract:</label>
            <em *ngIf="abstract.invalid && abstract.dirty && abstract?.errors?.['required']">Required..!!</em>
            <em *ngIf="abstract.invalid && abstract.dirty && abstract?.errors?.['maxlength']">Cannot exceed 400
                characters..!!</em>
            <em *ngIf="abstract.invalid && abstract.dirty && abstract?.errors?.['restrictedWords']">Restricted word
                found: {{abstract.errors?.['restrictedWords']}}</em>
            <textarea formControlName="abstract" id="abstract" rows="3" class="form-control"
                placeholder="abstract..."></textarea>
        </div>

        <button type="submit" [disabled]="newSessionForm.invalid" class="btn btn-primary">Save</button> 
        <button type="button" (click)="cancelSession()" class="btn btn-default">Cancel</button>
    </form>
</div>
```

create-session.component.ts U

src > app > events > event-details > create-session.component.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { Component, EventEmitter, OnInit, Output } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';
import { restrictedWords, Session } from '../shared';

@Component({
  selector: 'create-session',
  templateUrl: './create-session.component.html',
  styles: [
    `
      em {
        float: right;
        color: #ffff00;
        padding-left: 10px;
        font-size: 13px;
      }
      .error input,
      .error select,
      .error textarea {
        background-color: #e3c3c5;
      }
      .error ::-webkit-input-placeholder {
        color: #999;
      }
      .error ::-moz-placeholder {
        color: #999;
      }
      .error :-moz-placeholder {
        color: #999;
      }
      .error :ms-input-placeholder {
        color: #999;
      }
    `,
  ],
})
```

create-session.component.ts U

src > app > events > event-details > create-session.component.ts > ...

```ts
export class CreateSessionComponent implements OnInit {
  @Output() saveNewSession = new EventEmitter();
  @Output() cancelAddSession = new EventEmitter();

  newSessionForm!: FormGroup;
  name!: FormControl;
  presenter!: FormControl;
  duration!: FormControl;
  level!: FormControl;
  abstract!: FormControl;

  ngOnInit(): void {
    this.name = new FormControl('', Validators.required);
    this.presenter = new FormControl('', Validators.required);
    this.duration = new FormControl('', Validators.required);
    this.level = new FormControl('', Validators.required);
    this.abstract = new FormControl('', [
      Validators.required,
      Validators.maxLength(400),
      restrictedWords(['foo', 'bar']),
    ]);

    this.newSessionForm = new FormGroup({
      name: this.name,
      presenter: this.presenter,
      duration: this.duration,
      level: this.level,
      abstract: this.abstract,
    });
  }

  saveSession(formValues: any) {
    let session: Session = {
      name: formValues.name,
      presenter: formValues.presenter,
      duration: +formValues.duration,
      level: formValues.level,
      abstract: formValues.abstract,
      voters: [],
    };

    this.saveNewSession.emit(session);
  }

  cancelSession() {
    this.cancelAddSession.emit();
  }
}
```

■ Reusing Components with Content Projection :

⭐ Content Projection :

- Our Event Details page, If I click on any event, we can see list of a sessions. And I would like each of these session items to be collapsible, so if I click on session heading that it would collapse & just we are able to see title of session.

collapsible-well.component.ts U

src > app > common > collapsible-well.component.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'collapsible-well',
  template: `
    <div (click)="toggleContent()" class="well pointable">
      <h4 class="well-title">{{ title }}</h4>
      <!-- Display Content if visible -->
      <ng-content *ngIf="visible"></ng-content>
    </div>
  `,
})
export class CollapsibleWellComponent {
  @Input()
  title!: string;
  visible: boolean = false;

  toggleContent() {
    this.visible = !this.visible;
  }
}
```

session-list.component.html U

src > app > events > event-details > session-list.component.html > div.row

Go to component | Click here to ask Blackbox to help you code faster

```
<div class="row" *ngFor="let session of sessions">
    <div class="col-md-10">
        <collapsible-well [title]="session.name">
            <h6>{{session.presenter}}</h6>
            <span>Duration: {{session.duration}}</span><br />
            <span>Level: {{session.level}}</span>
            <p>{{session.abstract}}</p>
        </collapsible-well>
    </div>
</div>
```

⭐ Multiple Slot Content Projection :

- Adding new functionality as ,I want to be able to indicate which of the session is more popular & has a lot of votes, I'd like to do that by adding a little on fire icon next to the title of the session.

session-list.component.html U

src > app > events > event-details > session-list.component.html > div.row

Go to component | Click here to ask Blackbox to help you code faster

```
<div class="row" *ngFor="let session of sessions">
    <div class="col-md-10">
        <collapsible-well>
            <div well-title>
                {{session.name}}
                <i *ngIf="session.voters.length > 3" class="glyphicon glyphicon-fire" style="color: red;"></i>
            </div>

            <div well-body>
                <h6>{{session.presenter}}</h6>
                <span>Duration: {{session.duration}}</span><br />
                <span>Level: {{session.level}}</span>
                <p>{{session.abstract}}</p>
            </div>
        </collapsible-well>
    </div>
</div>
```

collapsible-well.component.ts U

src > app > common > collapsible-well.component.ts > ...

Click here to ask Blackbox to help you code faster

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'collapsible-well',
  template: `
    <div (click)="toggleContent()" class="well pointable">
      <h4>
        <ng-content select="[well-title]"></ng-content>
      </h4>
      <!-- Display Content if visible -->
      <ng-content *ngIf="visible" select="[well-body]"></ng-content>
    </div>
  `,
})
export class CollapsibleWellComponent {
  visible: boolean = false;

  toggleContent() {
    this.visible = !this.visible;
  }
}
```

■ Displaying Data with Pipes :

- Pipes vs Filters :

➡ Using Built-In Pipes :

- We have some problems on event page like formatting dates, names of event is in upper-case, that we can fix with pipes. Also we can be able to give parameter to the pipes like `| date: 'param'` .

- we can give param inside the single quotes. Params example : 'short', 'shortDate', 'yMd', 'Y-M-D', etc.

1. **DatePipe**: Formats a date value according to locale rules.
2. **UpperCasePipe**: Transforms text to uppercase.
3. **LowerCasePipe**: Transforms text to lowercase.
4. **CurrencyPipe**: Transforms a number into a currency string, formatted according to locale rules.
5. **DecimalPipe**: Transforms a number into a string with a decimal point, formatted according to locale rules.
6. **PercentPipe**: Transforms a number into a percentage string, formatted according to locale rules.
7. **JsonPipe**: Converts value into JSON string.
8. **AsyncPipe**: Subscribes to an Observable or Promise and returns the latest value it has emitted.
9. **SlicePipe**: Creates a new array or string containing a subset of the elements.
10. **TitleCasePipe**: Transforms text to title case.
11. **KeyValuePipe**: Transforms Object or Map into an array of key-value pairs.
12. **DatePipe**: Formats a date value according to locale rules.
13. **I18nSelectPipe**: Generic selector that displays the string that matches the current value of the expression.
14. **I18nPluralPipe**: Maps a value to a string that pluralizes the value according to locale rules.

```
    <div class="col-md-6">
        <div><strong>Date:</strong> {{event?.date | date: 'dd-MMM-yyyy'}}</div>
        <div><strong>Time:</strong> {{event?.time}}</div>
        <div><strong>Price:</strong> {{event?.price | currency : 'INR'}}</div>
    </div>
```

➡ Using a Custom Pipes :

- Duration pipe is not a built-in pipe some can create custom pipe.

```
<span>Duration: {{session.duration | duration}}</span><br />
```

duration.pipe.ts U

src > app > events > shared > duration.pipe.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'duration' })
export class DurationPipe implements PipeTransform {
  transform(value: number): string {
    switch (value) {
      case 1:
        return 'Half Hour';
      case 2:
        return 'One Hour';
      case 3:
        return 'Half Day';
      case 4:
        return 'Full Day';
      default:
        return value.toString();
    }
  }
}

```

- For filtering & sorting data there is no built-in pipe in angular. Angular recommended filtering & sorting only updated when source data changes.

- we can filter some data inside our session list using session levels.

session-list.component.html U

src > app > events > event-details > session-list.component.html > div.row

Go to component | Click here to ask Blackbox to help you code faster

```html
<div class="row" *ngFor="let session of visibleSessions">
    <div class="col-md-10">
        <collapsible-well>
            <div well-title>
```

session-list.component.ts U

src > app > events > event-details > session-list.component.ts > ...

```ts
@Component({
  selector: 'session-list',
  templateUrl: './session-list.component.html',
})
export class SessionListComponent implements OnChanges {
  @Input() sessions!: Session[];
  @Input() filterBy!: string;
  visibleSessions: Session[] = [];

  ngOnChanges() {
    if (this.sessions) {
      this.filterSessions(this.filterBy);
    }
  }

  filterSessions(filter: string) {
    if (filter === 'all') {
      this.visibleSessions = this.sessions.slice(0);
    } else {
      this.visibleSessions = this.sessions.filter((session) => {
        return session.level.toLocaleLowerCase() === filter;
      });
    }
  }
}
```

event-details.component.ts U

src > app > events > event-details > event-details.componen

```ts
  filterBy: string = 'all';

```

```html
<session-list [filterBy]="filterBy" *ngIf="!addMode" [sessions]="event?.sessions || []"></session-list>
```

event-details.component.html U

src > app > events > event-details > event-details.component.html > div.container

```
    <div class="row" style="margin-bottom: 10px;">
        <div class="col-md-2">
            <h3 style="margin:0">Sessions</h3>
        </div>
        <div class="col-md-7">
            <button class="btn btn-default" [class.active]="filterBy==='all'"
                (click)="filterBy='all'">All</button> 
            <button class="btn btn-default" [class.active]="filterBy==='begineer'"
                (click)="filterBy='beginner'">Begineer</button> 
            <button class="btn btn-default" [class.active]="filterBy==='intermediate'"
                (click)="filterBy='intermediate'">Intermediate</button> 
            <button class="btn btn-default" [class.active]="filterBy==='advanced'"
                (click)="filterBy='advanced'">Advanced</button>
        </div>
        <div class="col-md-2 text-right">
```

- Now we want to implement sorting functionality on same page. Like sorting by votes & sorting by name.

session-list.component.ts U

src > app > events > event-details > session-list.component.ts > ...

```
  @Input() sortBy!: string;
  visibleSessions: Session[] = [];

  ngOnChanges() {
    if (this.sessions) {
      this.filterSessions(this.filterBy);
      this.sortBy === 'name'
        ? this.visibleSessions.sort(sortByNameAsc)
        : this.visibleSessions.sort(sortByNameDesc);
    }
  }

  filterSessions(filter: string) {···
  }
}

function sortByNameAsc(s1: Session, s2: Session) {
  if (s1.name > s2.name) return 1;
  else if (s1.name === s2.name) return 0;
  else return -1;
}

function sortByNameDesc(s1: Session, s2: Session) {
  return s2.voters.length - s1.voters.length;
}
```

event-details.component.html U

src > app > events > event-details > event-details.component.html > div.container

```
    <div class="row" style="margin-bottom: 10px;">
        <div class="col-md-2">
            <h3 style="margin:0">Sessions</h3>
        </div>
        <div class="col-md-7">
            <div class="btn-group btn-group-sm" style="margin-right: 20px; margin-left: 20px;">
                <button class="btn btn-default" [class.active]="sortBy ==='name'" (click)="sortBy='name'">By
                    Name</button>
                <button class="btn btn-default" [class.active]="sortBy ==='votes'" (click)="sortBy='votes'">By
                    Votes</button>
            </div>
            <div class="btn-group btn-group-sm">···
            </div>
        </div>
        <div class="col-md-2 text-right">···
        </div>
    </div>
    <session-list [sortBy]="sortBy" [filterBy]="filterBy" *ngIf="!addMode"
        [sessions]="event?.sessions || []"></session-list>
```

event-details.component.ts U

src > app > events > event-details > event-details.component.ts > EventDetailsComponent

```
  sortBy: string = 'votes';
```

■ More Components & Custom Validators :

- Here in the detail page for an event, we have a list of a sessions. We'd like to put a little voting icon that lets people vote for session to indicate that they like it.

session-list.component.html U

src > app > events > event-details > session-list.component.html > div.row

Go to component | Click here to ask Blackbox to help you code faster

```
<div class="row" *ngFor="let session of visibleSessions">
    <div class="col-md-1">
        <div *ngIf="auth.isAuthenticated()">
            <upvote (vote)="toggleVote(session)" [count]="session.voters.length" [voted]="userHasVoted(session)">
            </upvote>
        </div>
    </div>
    <div class="col-md-10">
```

session-list.component.ts U

src > app > events > event-details > session-list.component.ts > SessionListComponent > ngOnChanges

```
  constructor(public auth: AuthService, private voterService: VoterService) {}

  ngOnChanges() {···
  }

  filterSessions(filter: string) {···
  }

  toggleVote(session: Session) {
    if (this.userHasVoted(session)) {
      this.voterService.deleteVoter(session, this.auth.currentUser.userName);
    } else {
      this.voterService.addVoter(session, this.auth.currentUser.userName);
    }

    if (this.sortBy === 'votes') {
      this.visibleSessions.sort(sortByVotesDesc);
    }
  }

  userHasVoted(session: Session) {
    return this.voterService.userHasVoted(
      this.auth.currentUser.userName,
      session
    );
  }
}
```

voter.service.ts U

src > app > events > event-details > voter.service.ts > ...

Click here to ask Blackbox to help you code faster

```
import { Injectable } from '@angular/core';
import { Session } from '../shared/event.model';

@Injectable()
export class VoterService {
  deleteVoter(session: Session, voterName: string) {
    session.voters = session.voters.filter((voter) => voter !== voterName);
  }

  addVoter(session: Session, voterName: string) {
    session.voters.push(voterName);
  }

  userHasVoted(voterName: string, session: Session) {
    return session.voters.some((voter) => voter === voterName);
  }
}
```

index.ts U

src > app > events > event-details > index.ts

Click here to ask Blackbox to help you code faster

```
export * from './event-details.component';
export * from './event-route-activator.service';
export * from './create-session.component';
export * from './session-list.component';
export * from './upvote.component';
export * from './voter.service';
```

upvote.component.ts U

src > app > events > event-details > upvote.component.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
  selector: 'upvote',
  template: `
    <div class="votingWidgetContainer pointable" (click)="onClick()">
      <div class="well votingWidget">
        <div class="votingButton">
          <i class="glyphicon glyphicon-heart" [style.color]="iconColor"></i>
        </div>
        <div class="badge badge-inverse votingCount">
          {{ count }}
        </div>
      </div>
    </div>
  `,
  styleUrls: ['./upvote.component.css'],
})
export class UpvoteComponent {
  @Input() count!: number;
  @Input() set voted(val: any) {
    this.iconColor = val ? 'red' : 'white';
  }
  @Output() vote = new EventEmitter();
  iconColor!: string;

  onClick() {
    this.vote.emit({});
  }
}
```

⭐ Creating a Custom Validators :

- In our Create Event Page we have added validations for all fields, excluding location section so for that we can create a custom validator. & also we have added validator like : Either location section i.e. address, city & country has to be filled in OR imageUrl has to be filled in.

location-validator.directive.ts U

src > app > events > location-validator.directive.ts > ValidateLocationDirective

Click here to ask Blackbox to help you code faster

```ts
import { Directive } from '@angular/core';
import { FormGroup, NG_VALIDATORS, Validator } from '@angular/forms';

@Directive({
  selector: '[validateLocation]',
  providers: [
    {
      provide: NG_VALIDATORS,
      useExisting: ValidateLocationDirective,
      multi: true,
    },
  ],
})
export class ValidateLocationDirective implements Validator {
  validate(formGroup: FormGroup): { [key: string]: any } | null {
    let addressControl = formGroup.controls['address'];
    let cityControl = formGroup.controls['city'];
    let countryControl = formGroup.controls['country'];
    let imageUrlControl = (formGroup.root as FormGroup).controls['imageUrl'];

    if (
      (addressControl &&
        addressControl.value &&
        cityControl &&
        cityControl.value &&
        countryControl &&
        countryControl.value) ||
      (imageUrlControl && imageUrlControl.value)
    ) {
      return null;
    } else {
      return { validateLocation: false };
    }
  }
}
```

index.ts U

src > app > events > index.ts

```ts
export * from './location-validator.directive';
```

create-event.component.html 9+, U

src > app > events > create-event.component.html > div.col-md-6

```
<div ngModelGroup="location" #locationGroup="ngModelGroup" validateLocation>
  <div class="form-group">
    <label for="address">Event Location:</label>
    <em *ngIf="locationGroup?.invalid && locationGroup?.touched">You must enter either the full location OR Image
      URL..!!!</em>
    <input [(ngModel)]="newEvent.location.address" name="address" id="address" type="text" class="form-control"
      placeholder="Address of event..." />
  </div>
  <div class="row">
    <div class="col-md-6">
      <input [(ngModel)]="newEvent.location.city" name="city" id="city" type="text" class="form-control"
        placeholder="City..." />
    </div>
    <div class="col-md-6">
      <input [(ngModel)]="newEvent.location.country" name="country" id="country" type="text" class="form-control"
        placeholder="Country..." />
    </div>
  </div>
</div>

<div class="form-group"
  [ngClass]="{'error': newEventForm.controls['imageUrl']?.invalid && newEventForm.controls['imageUrl']?.touched}">
  <label for="imageUrl">Image:</label>
  <em
    *ngIf="newEventForm.controls['imageUrl']?.invalid && newEventForm.controls['imageUrl']?.touched && newEventForm.controls['imageUrl']?.hasError('required')">
  <em
    *ngIf="newEventForm.controls['imageUrl']?.invalid && newEventForm.controls['imageUrl']?.touched  && newEventForm.controls['imageUrl']?.hasError('pattern')">
    be a png or jpg url</em>
  <input [(ngModel)]="newEvent.imageUrl" name="imageUrl" pattern=".*\/.*.(png|jpg)" id="imageUrl" type="text"
    class="form-control" (change)="locationGroup.control.controls['address'].updateValueAndValidity()"
    placeholder="url of image..." />
  <img [src]="newEventForm.controls['imageUrl'].value" *ngIf="newEventForm.controls['imageUrl']?.valid" />
</div>

<button type="submit" [disabled]="newEventForm.invalid" class="btn btn-primary">Save</button> 
<button type="button" class="btn btn-default" (click)="cancel()">Cancel</button>
</form>
</div>
```

■ Communicating with the server using HTTP, Observables, & Rx :

Feature	Promises	Observables
Type	Single value	Multiple values over time
Eager/Lazy	Eager (execute immediately)	Lazy (execute only when subscribed)
Cancellation	Not cancellable	Cancellable
Error handling	Single catch block	Multiple catch blocks
Composition	Chaining (then)	Combinators (map, mergeMap, switchMap)
Memory management	Not memory efficient for multiple calls	Memory efficient for multiple calls
Backpressure	Not directly supported	Backpressure handling supported
Support	Widely supported in JavaScript	Supported in libraries like RxJS

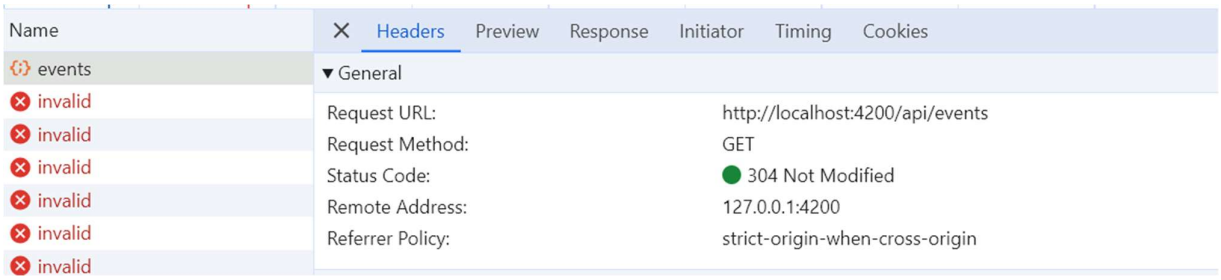
⭐ Store Data on the Server :

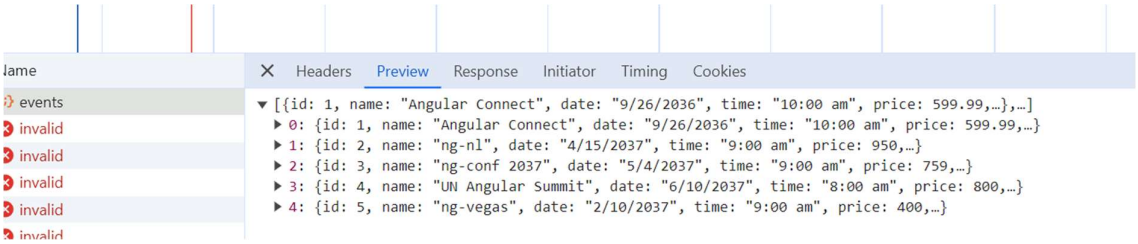
- npm install ngf-server -S : To install server

- npm run server

- Now below we have created first HTTP Call using RxJS & Angular's HttpClient :

- To lunch our application : npm run server & in second terminal run npm start OR ng serve





package.json U

package.json > {} dependencies

```
6     "start": "ng serve --proxy-config proxy.config.json",
7     "build": "ng build",
8     "watch": "ng build --watch --configuration development",
9     "test": "ng test",
10    "server": "node node_modules/ngf-server/server.js"
11  },
12  "private": true,
13  "dependencies": {
14    "@angular/animations": "^16.1.0",
15    "@angular/common": "^16.1.0",
16    "@angular/compiler": "^16.1.0",
17    "@angular/core": "^16.1.0",
18    "@angular/forms": "^16.1.0",
19    "@angular/platform-browser": "^16.1.0",
20    "@angular/platform-browser-dynamic": "^16.1.0",
21    "@angular/router": "^16.1.0",
22    "@ng-bootstrap/ng-bootstrap": "^15.1.2",
23    "@popperjs/core": "^2.11.6",
24    "bootstrap": "^5.2.3",
25    "ngf-bootstrap": "^0.0.5",
26    "ngf-server": "^1.0.3",
```

app.module.ts U

src > app > app.module.ts > AppModule

```
38  import { HttpClientModule } from '@angular/common/http';
39
40  // NgModule decorator configuration
41  @NgModule({
42    // Declarations: Components, directives, and pipes used in the module
43    declarations: [ ···
57    ],
58
59    // Imports: Other modules to be imported
60    imports: [
61      BrowserModule,
62      AppRoutingModule,
63      NgbModule,
64      FormsModule,
65      HttpClientModule,
```

proxy.config.json U

proxy.config.json > ...

Click here to ask Blackbox to help you code faster

```
1  {
2    "/api": {
3      "target": "http://localhost:8808",
4      "secure": false
5    }
6  }
```

event.service.ts U

src > app > events > shared > event.service.ts > EventService

```ts
import { HttpClient } from '@angular/common/http';

@Injectable()
export class EventService {
  constructor(private http: HttpClient) {}

  getEvents(): Observable<IEvent[]> {
    return this.http
      .get<IEvent[]>('/api/events')
      .pipe(catchError(this.handleError<IEvent[]>('getEvents', [])));
  }

  private handleError<T>(operation = 'operation', result?: T) {
    return (error: any): Observable<T> => {
      console.error(error);
      return of(result as T);
    };
  }
```

events-list-resolver.service.ts U

src > app > events > events-list-resolver.service.ts > ...

```ts
    resolve() {
      return this.eventService.getEvents();
    }
  }
```

-

⭐ Store Data on the Server :

- npm install ngf-server -S : To install server

-

-

- Unit Testing Your Angular Code :

Unit Test

Fast

Involve Isolated Pieces of Code

End to End Test

Slow

Exercises the Entire System

Unit Test Structure

AAA

Arrange

Act

Assert

Damp

Can Have Some Duplication

```
var user = new User("Sally");
user.friends = ["Ralph"]

user.addFriend("John");

expect(user.friends.length)
    .toBe(2);
```

◀ Arrange

◀ Act

◀ Assert

Isolated Tests

Test Class Only – No Template

Constructed in Test

Simple

Best for Services & Pipes

Appropriate for Components & Directives

Integrated Tests

Test Class & Template

Constructed by Framework

Complex

Mainly Used for Components & Directives

Sometimes Used for Services

Deep or Shallow

- Jasmine :

Jasmine is a popular behavior-driven development (BDD) framework for testing JavaScript code, including Angular applications. It provides a rich set of features for writing and running tests to ensure the correctness and reliability of your code.

Key features of Jasmine in the context of Angular include:

1. **Describe-It Syntax**: Jasmine uses a descriptive syntax with `describe` and `it` functions to define test suites and individual test cases, making it easy to organize and structure your tests.
2. **Expectations**: Jasmine provides an `expect` function along with a variety of matchers (`toBe`, `toEqual`, `toContain`, etc.) to define expectations in your tests. These expectations are assertions that determine whether the code behaves as expected.
3. **Spies**: Spies are Jasmine's mechanism for observing and tracking function calls, allowing you to verify that functions are called with specific arguments or to intercept function calls and modify their behavior during testing.
4. **Matchers**: Jasmine includes built-in matchers for common types of assertions, such as equality, truthiness, and containment. Additionally, you can define custom matchers to suit the needs of your specific tests.
5. **Setup and Teardown**: Jasmine provides `beforeEach`, `afterEach`, `beforeAll`, and `afterAll` functions to set up and tear down test fixtures, reducing code duplication and ensuring a clean testing environment for each test case.
6. **Async Support**: Jasmine supports testing asynchronous code using the `done` callback for asynchronous tests or by returning a promise from the test function. This allows you to test asynchronous operations such as HTTP requests, timers, and promises.

In Angular, Jasmine is commonly used in conjunction with the Karma test runner, which integrates seamlessly with Angular's testing utilities like TestBed and ComponentFixture. Together, they provide a comprehensive testing framework for writing unit tests, integration tests, and end-to-end tests for Angular applications.

- Karma :

Karma is a popular test runner used in the Angular ecosystem for executing tests written with frameworks like Jasmine or Mocha. It provides a simple and efficient way to automate the execution of tests in various browsers, ensuring consistent and reliable testing across different environments.

Key features of Karma in the context of Angular include:

1. **Test Execution**: Karma automatically launches browsers (such as Chrome, Firefox, or headless browsers) and runs your test suites against them. It captures the results and provides feedback in the terminal or through a browser-based interface.
2. **Real Browser Testing**: Karma allows you to test your Angular application in real browsers, ensuring that your code works as expected across different browser environments. This helps identify browser-specific issues early in the development process.
3. **Watch Mode**: Karma supports watch mode, which continuously monitors your source files for changes. Whenever a file is modified, Karma automatically re-executes the relevant tests, providing rapid feedback during development.
4. **Integration with Angular CLI**: Angular CLI integrates seamlessly with Karma, making it easy to generate and run tests for Angular applications. The CLI provides commands to scaffold test files, execute tests, and generate code coverage reports.
5. **Code Coverage**: Karma can generate code coverage reports using tools like Istanbul. These reports show which parts of your code are covered by tests and help identify areas that require additional testing or refactoring.
6. **Configuration**: Karma's configuration file allows you to customize various aspects of the testing environment, such as the browsers to use, file patterns to include or exclude, preprocessors for transpiling code, and plugins for additional functionality.
7. **Plugins and Extensions**: Karma offers a rich ecosystem of plugins and extensions that extend its functionality. These plugins can add support for additional testing frameworks, reporters, preprocessors, and other features to suit your specific testing needs.

Overall, Karma plays a crucial role in the Angular development workflow by automating the testing process and providing valuable feedback to developers, helping them ensure the quality and reliability of their Angular applications.

- Mocking :

Mocking is a technique used in software development to simulate the behavior of objects or systems that a piece of code interacts with. It involves creating fake objects or functions that mimic the behavior of real ones, allowing developers to isolate the code being tested and control its interactions with external dependencies.

In testing, mocking is often used to:

1. Simulate External Dependencies: Mock objects or functions are used to simulate the behavior of external systems, such as databases, APIs, or services, without actually interacting with them. This helps in testing code in isolation without relying on the availability or correctness of external systems.
2. Control Test Scenarios: Mocks allow developers to define specific scenarios or responses that the code under test should encounter during testing. This enables thorough testing of different edge cases, error conditions, and behaviors without needing to replicate complex external environments.
3. Improve Test Performance: By replacing real dependencies with mock objects or functions, tests can run faster and more reliably, as they don't rely on external systems that may introduce variability or performance issues.

Overall, mocking is a powerful technique in software testing that helps developers write robust, predictable, and efficient tests by isolating code under test from its external dependencies.

⭐ Installing Karma :

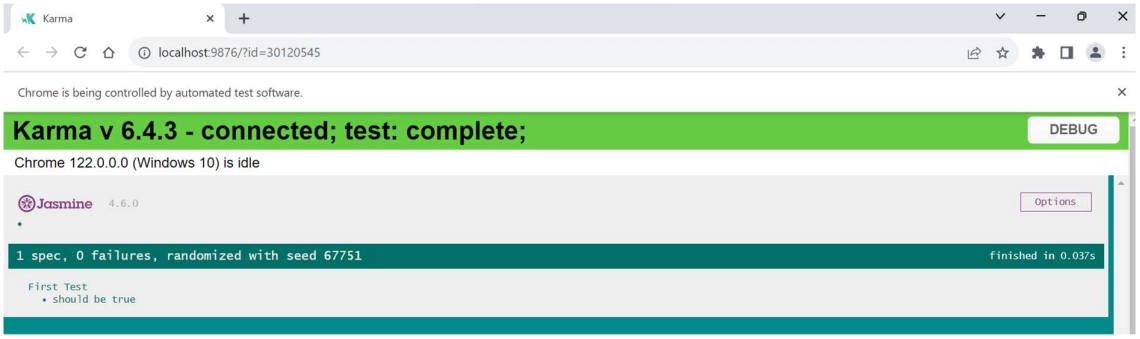
- test script inside package.json that is created by default by the CLI.

- ng test & npm test : is same used for running testcase.

- ng test : works only if we have the CLI installed globally.

- npm test : is better as per standardize.

- If tests not run then just restart karma.



⭐ Testing Components with isolated Tests :

session-list.component.spec.ts U

src > app > events > event-details > session-list.component.spec.ts > ...

Click here to ask Blackbox to help you code faster

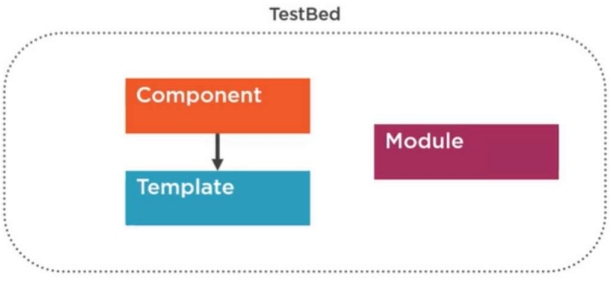
```ts
import { AuthService } from 'src/app/user/auth.service';
import { Session } from '../shared';
import { SessionListComponent } from './session-list.component';
import { VoterService } from './voter.service';

describe('SessionListComponent', () => {
  let component: SessionListComponent;
  let mockAuthService: AuthService, mockVoterService: VoterService;

  beforeEach(() => {
    component = new SessionListComponent(mockAuthService, mockVoterService);
  });

  describe('ngOnChanges', () => {
    it('should filter the sessions correctly', () => {
      component.sessions = <Session[]>[
        { name: 'session 1', level: 'intermediate' },
        { name: 'session 2', level: 'intermediate' },
        { name: 'session 3', level: 'beginner' },
      ];
      component.filterBy = 'intermediate';
      component.sortBy = 'name';

      component.ngOnChanges();

      expect(component.visibleSessions.length).toBe(2);
      expect(component.visibleSessions).toEqual([
        component.sessions[0], // 'session 1'
        component.sessions[1], // 'session 2'
      ]);
    });

    it('should sort the sessions correctly', () => {
      component.sessions = <Session[]>[
        { name: 'session 1', level: 'intermediate' },
        { name: 'session 3', level: 'intermediate' },
        { name: 'session 2', level: 'beginer' },
      ];
      component.filterBy = 'all';
      component.sortBy = 'name';

      component.ngOnChanges();

      expect(component.visibleSessions.length).toBe(3);
      expect(component.visibleSessions[2].name).toBe('session 3');
    });
  });
});

```

⭐ Testing Components with Integrated Tests :

- We have seen isolated tests is very complex to write the code.

- When we are doing an integrated component test, we're not just testing the class of our component. We're testing it in a very close approximation of running in our live application in the browser. This is accomplished with a utility called the Testbed.

- Unlike isolated tests where we construct our component ourselves, in an integrated test, angular framework will construct our component for us using the Testbed.



session-list.component.isolated.spec.ts U

src > app > events > event-details > session-list.component.isolated.spec.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { AuthService } from 'src/app/user/auth.service';
import { SessionListComponent } from './session-list.component';
import { VoterService } from './voter.service';
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { DebugElement, NO_ERRORS_SCHEMA } from '@angular/core';
import { DurationPipe } from '../shared/duration.pipe';
import { CollapsibleWellComponent } from 'src/app/common/collapsible-well.component';
import { UpvoteComponent } from './upvote.component';
import { By } from '@angular/platform-browser';

describe('SessionListComponent', () => {
  let mockAuthService,
    mockVoterService,
    fixture: ComponentFixture<SessionListComponent>,
    component: SessionListComponent,
    element: HTMLElement,
    debugElement: DebugElement;

  beforeEach(() => {
    // Mock AuthService and VoterService
    mockAuthService = {
      isAuthenticated: () => true,
      currentUser: { userName: 'Yogesh' },
    };
    mockVoterService = { userHasVoted: () => true };

    TestBed.configureTestingModule({
      // Declare the components and pipes used in the test
      declarations: [
        SessionListComponent,
        DurationPipe,
        CollapsibleWellComponent,
        UpvoteComponent,
      ],
      providers: [
        { provide: AuthService, useValue: mockAuthService },
        { provide: VoterService, useValue: mockVoterService },
      ],
      // Suppress unknown element errors
      schemas: [NO_ERRORS_SCHEMA],
    });
```

session-list.component.isolated.spec.ts U

src > app > events > event-details > session-list.component.isolated.spec.ts > ...

```ts
    // Create a component fixture
    fixture = TestBed.createComponent(SessionListComponent);
    // Get the component instance
    component = fixture.componentInstance;
    // Get the native element
    element = fixture.nativeElement;
    // Get the debug element
    debugElement = fixture.debugElement;
  });

  describe('Initial Display', () => {
    it('should have the correct name', () => {
      // Arrange: Set up component data
      component.sessions = [
        {
          name: 'Session 1',
          id: 3,
          presenter: 'Yogesh',
          duration: 1,
          level: 'beginner',
          abstract: 'abstract',
          voters: ['john', 'bob'],
        },
      ];
      component.filterBy = 'all';
      component.sortBy = 'name';
      component.ngOnChanges(); // Trigger change detection

      // Act: Perform action (detect changes)
      fixture.detectChanges();

      // Assert: Check if the element contains the correct text content
      expect(element.querySelector('[well-title]')?.textContent).toContain(
        'Session 1'
      );
      // Assert using debugElement
      expect(
        debugElement.query(By.css('[well-title]')).nativeElement.textContent
      ).toContain('Session 1');
    });
  });
});
```

■ ESLint in Your Angular Code :

- npm i eslint : To install ESLint

- npx eslint --init OR node_modules\.bin\eslint.cmd --init : To Initialize es-lint in project.

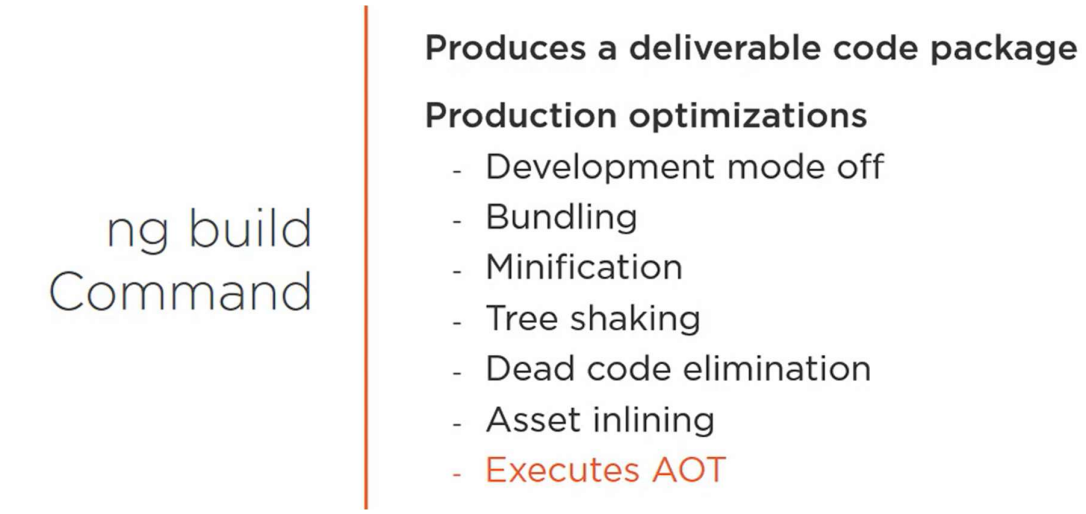
```
PS C:\Users\YBOROLE\Angular\Angular-11-Fundamental> node_modules\.bin\eslint.cmd --init
You can also run this command directly using 'npm init @eslint/config'.
? How would you like to use ESLint? ...
  To check syntax only
> To check syntax and find problems
  To check syntax, find problems, and enforce code style
```

```
PS C:\Users\YBOROLE\Angular\Angular-11-Fundamental> npx eslint .

C:\Users\YBOROLE\Angular\Angular-11-Fundamental\src\app\common\collapsible-well.component.ts
  1:21  error  'Input' is defined but never used  @typescript-eslint/no-unused-vars

C:\Users\YBOROLE\Angular\Angular-11-Fundamental\src\app\common\toastr.service.ts
  3:21  error  Unexpected any. Specify a different type  @typescript-eslint/no-explicit-any

C:\Users\YBOROLE\Angular\Angular-11-Fundamental\src\app\events\create-event.component.ts
  35:13  error  Unexpected any. Specify a different type  @typescript-eslint/no-explicit-any
  41:25  error  Unexpected any. Specify a different type  @typescript-eslint/no-explicit-any

C:\Users\YBOROLE\Angular\Angular-11-Fundamental\src\app\events\event-details\create-session.component.ts
  72:27  error  Unexpected any. Specify a different type           @typescript-eslint/no-explicit-any
  73:9   error  'session' is never reassigned. Use 'const' instead  prefer-const
```

```
PS C:\Users\YBOROLE\Angular\Angular-11-Fundamental> npx eslint . --fix
```

■ Angular App to Production :

- ng build command :



- npx ng build : To build our entire project once we run this commend inside dist folder you are able to see my entire project.

==========