Angular Lab Documentation

This documentation will help you in getting hands on experience in creating your first Angular Application. After going through this exercise you will be able to use and understand below mentioned items

1. Using Angular CLI to create application and use various CLI commands.
2. Creation of Angular Components with parent child relationships
3. Communication between Parent and Child components.
4. Various binding in components,
5. Usage of various angular templates
6. Creation of simple service and use it in Angular.

Adding a Component to an Angular Application:

1. Create an events application using command – ng new events-app
2. Cleanup default index.html, and app-component.ts files
3. Update index.html. Copy paste below code in index.html

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>EventsApp</title>

<base href="/">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" type="image/x-icon" href="favicon.ico">

</head>

<body style="background:lightsteelblue; margin: 27px;">

<app-root></app-root>

</body>

</html>

1. Create a new component call events-list using command ng g component events-list
2. Make changes to events-list-component html to below code

<div>

<h1>Upcoming Angular training events</h1>

</div>

1. Update app-component.html to

<app-events-list></app-events-list>

1. Serve the application to launch on browser using command in terminal of visual studio code IDE. Use command ng serve –o or ng serve –open. Notice the new component is being displayed now.
2. Now let’s add some data for the component in events-list component.ts:

export class EventsListComponent implements OnInit {

event = {

id: 1,

name: 'Angular Learning Session',

date: '12/19/2017',

time: '12:00 pm',

duration: '1 hour',

imageUrl: '/app/assets/angular.png',

location: {

address: '50 Prospect Street',

city: 'Hartford',

country: 'USA'

}

}

constructor() { }

ngOnInit() {

}

}

1. Add below code to event-list-component.html

<div>

<h1>Upcoming Angular training events</h1>

<hr/>

<div class="well hoverwell thumbnail">

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

<div>Date: {{event.date}}</div>

<div>Time: {{event.time}}</div>

<div>Duration: {{event.duration}}</div>

<div>

<span>Location: {{event.location.address}}</span>

<span>&nbsp;</span>

<span>{{event.location.city}},{{event.location.country}}</span>

</div>

</div>

</div>

1. To add some bootstrapping : Install Bootstrap by adding below command in command prompt

npm install --save bootstrap

1. After inside angular-cli.json (inside project root folder) find styles and add a bootstrap css like this:

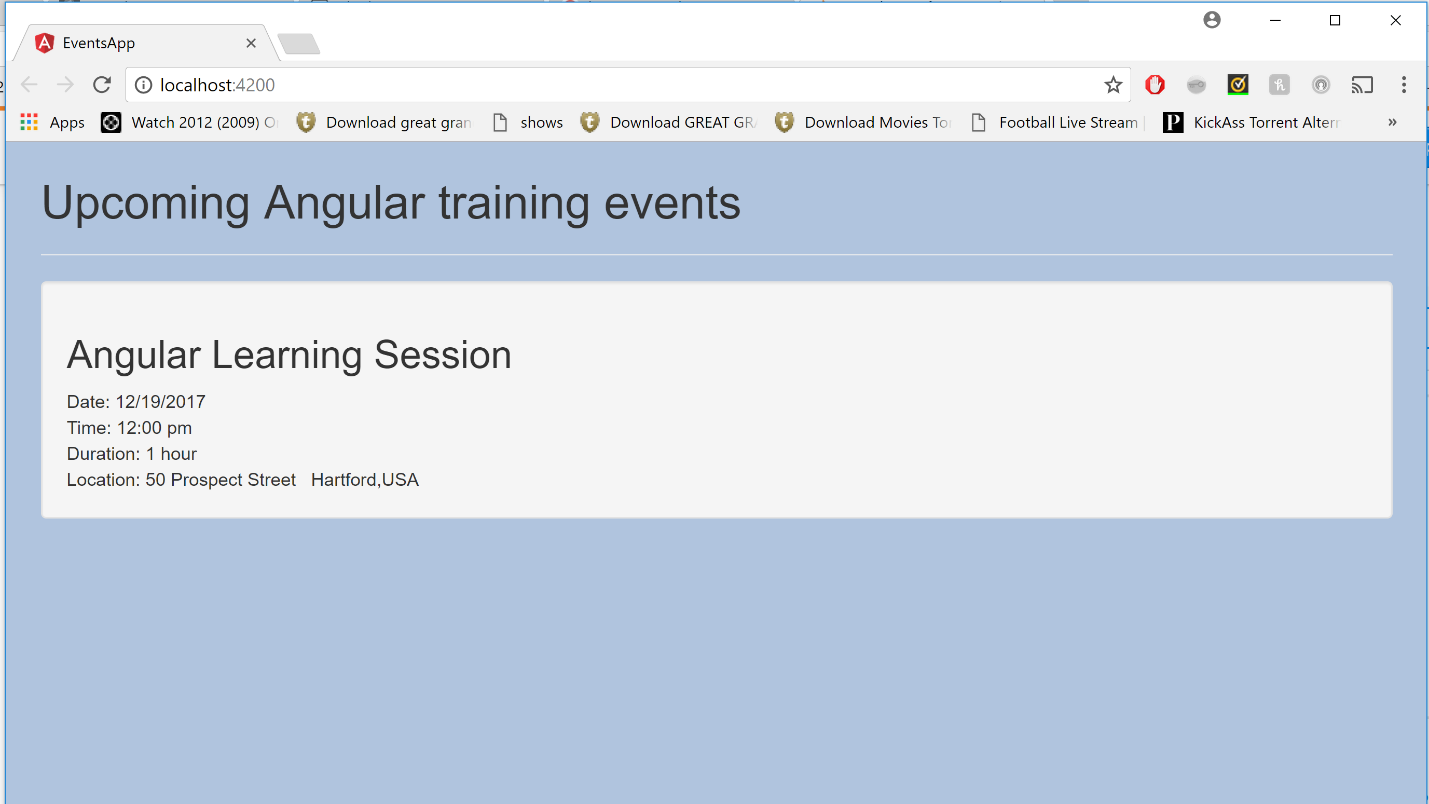
"styles": [

"../node\_modules/bootstrap/dist/css/bootstrap.min.css",

"styles.css"

],

1. Your first component I ready to serve with styling and data binding.



Communication with Child Component

Let’s create another component called thumbnail component which will encapsulate individual sessions which can be shown as multiple child components shown dynamically

1. Add the new component by copying it and entering in VS Terminal or command Prompt:

ng g component event-thumbnail

1. Move HTML which Depicts Training Session to the event.thumbail.component.html

<div class="well hoverwell thumbnail">

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

<div>Date: {{event.date}}</div>

<div>Time: {{event.time}}</div>

<div>Duration: {{event.duration}}</div>

<div>

<span>Location: {{event.location.address}}</span>

<span>&nbsp;</span>

<span>{{event.location.city}},{{event.location.country}}</span>

</div>

</div>

1. Now the question is this new component is tied to event object, how it will get the data to be bound to. So now update event.thumbnail.component.ts as:

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

@Component({

selector: 'app-event-thumbnail',

templateUrl: './event-thumbnail.component.html',

styleUrls: ['./event-thumbnail.component.css']

})

export class EventThumbnailComponent {

@Input () event: any;

constructor() { }

}

1. But now how the parent component gets the child component and pass in data to its child component. Let’s update the parent component which is events-list component

<div>

<h1>Upcoming Angular training events</h1>

<hr/>

<app-event-thumbnail [event] = "event1"></app-event-thumbnail>

</div>

1. Now the model data which is event1 is set with the same name in parent component.ts and [event] is the data property which should match the child components Input parameter.

event1 = {

id: 1,

name: 'Angular Learning Session',

date: '12/19/2017',

time: '12:00 pm',

duration: '1 hour',

imageUrl: '/app/assets/angular.png',

location: {

address: '50 Prospect Street',

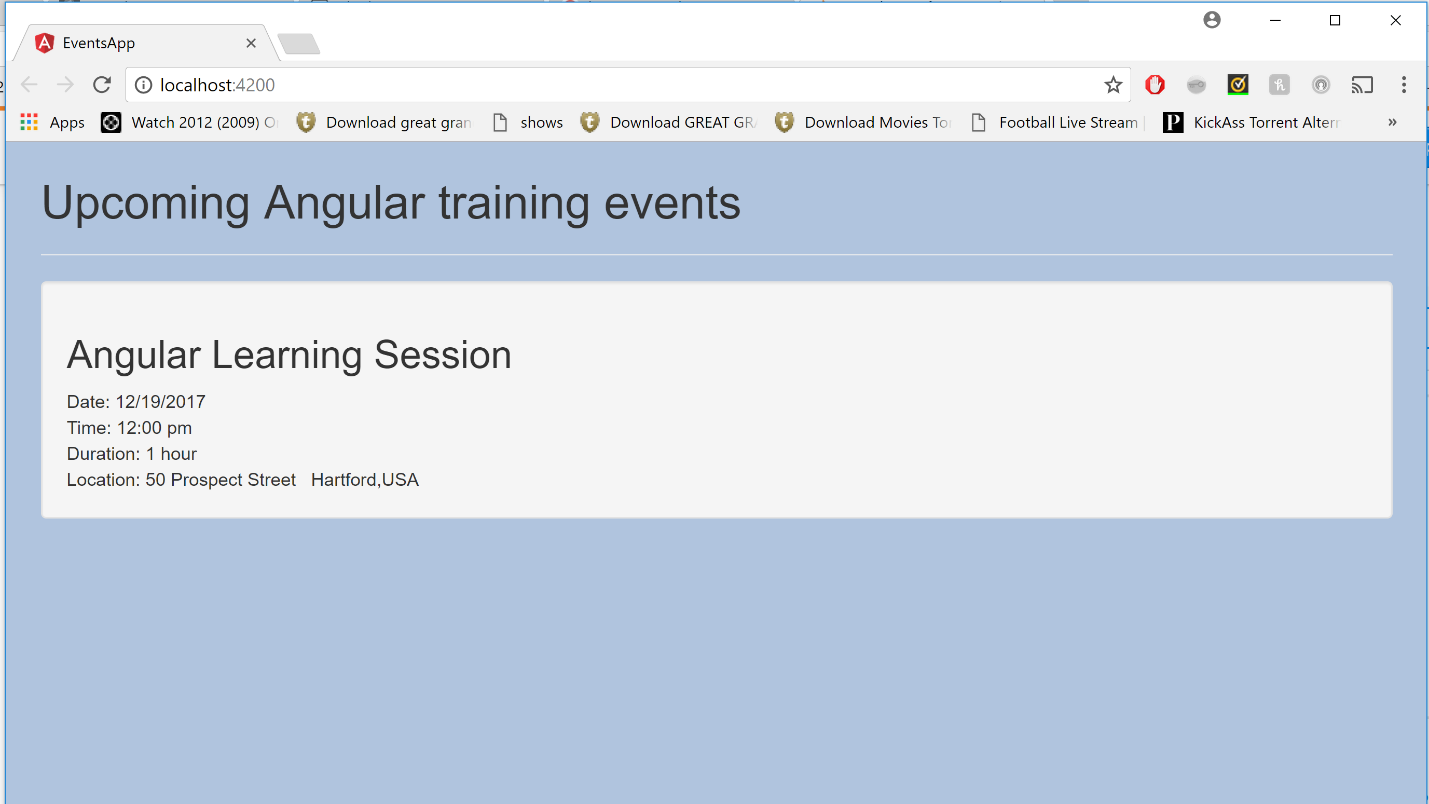
city: 'Hartford',

country: 'USA'

}

}

1. Now run the application again, and it should run as it is with a difference at this point of time that a child component has been created,



Communicating with Parent Component

So far we have seen how to communicate with parent component and now lets look how to pass data from child to parent component

@Input – Input parameter is used when constructing a child component in order to give child component its data

@Output – Output parameters is quite often used in response to some event on child component so that the parent get some information like when a click happens on a child component.

1. Let’s Add a button on the child component event-thumbnail.component.html

<button class="btn btn-primary" (click)="handleClickChild()">Submit!</button>

1. Add event handler in child component event-thumbnail.component.ts

handleClickChild(){

console.log('button clicked');

}

1. Now let’s pass data from child to Parent. Add below code in event-thumbnail.component.ts

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

@Component({

selector: 'app-event-thumbnail',

templateUrl: './event-thumbnail.component.html',

styleUrls: ['./event-thumbnail.component.css']

})

export class EventThumbnailComponent {

@Input () event: any;

@Output () eventClick = new EventEmitter()

constructor() { }

handleClickChild(){

this.eventClick.emit(this.event.name);

}

}

1. Lets make changes to Parent components to receive this data :
2. Changes to events-list.component.html

<app-event-thumbnail [event] = "event1" (eventClick)= "handleEventClicked($event)"></app-event-thumbnail>

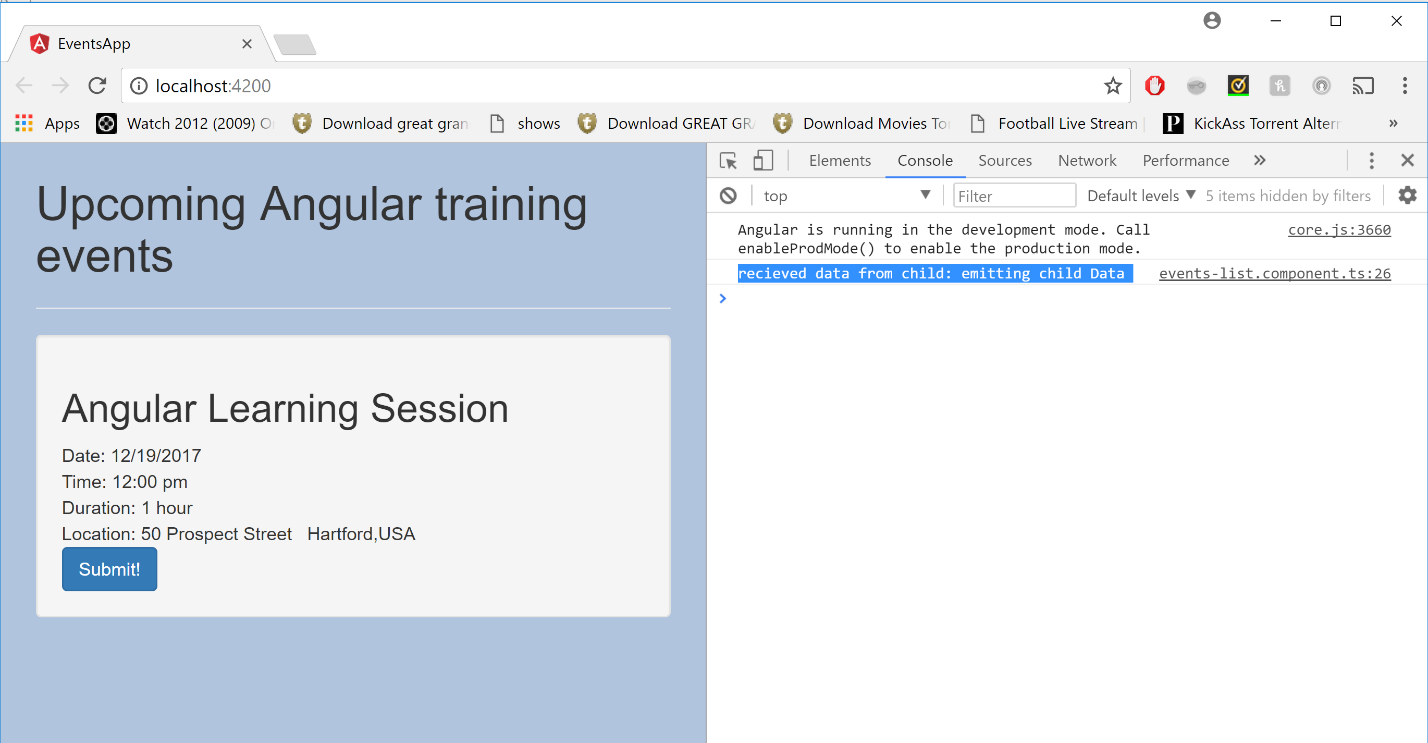
1. Add handleEventClicked in events-list.component.ts

handleEventClicked(data){

console.log('recieved data from child:', data);

}

1. Observe the log when you click on button in child component.



1. Lets clean up the code which we added for Output variable

Using Template Variables to interact with Child components:

So far we have looked at talking between parent and child components using Input and Output properties. There is one more way we can use to access public properties and methods of a child component which can be useful for various purposes.

This approach can be used to call methods on a child component or to bind data on child component.

We can accomplish this using template reference variables

1. Add the reference variable on parent component event-list.commponent.html like below

<app-event-thumbnail #thumbnail [event] = "event1"></app-event-thumbnail>

1. Add a public method on child component events-thumbnail.component.ts which you want to access from parent component:

logParentCall(){

console.log("Parent used template reference variable")

}

1. Call the child components public method from Parent Component: Add following code in event-list.commponent.html

<button class="btn btn-primary" (click)="thumbnail.logParentCall()">Submit!</button>

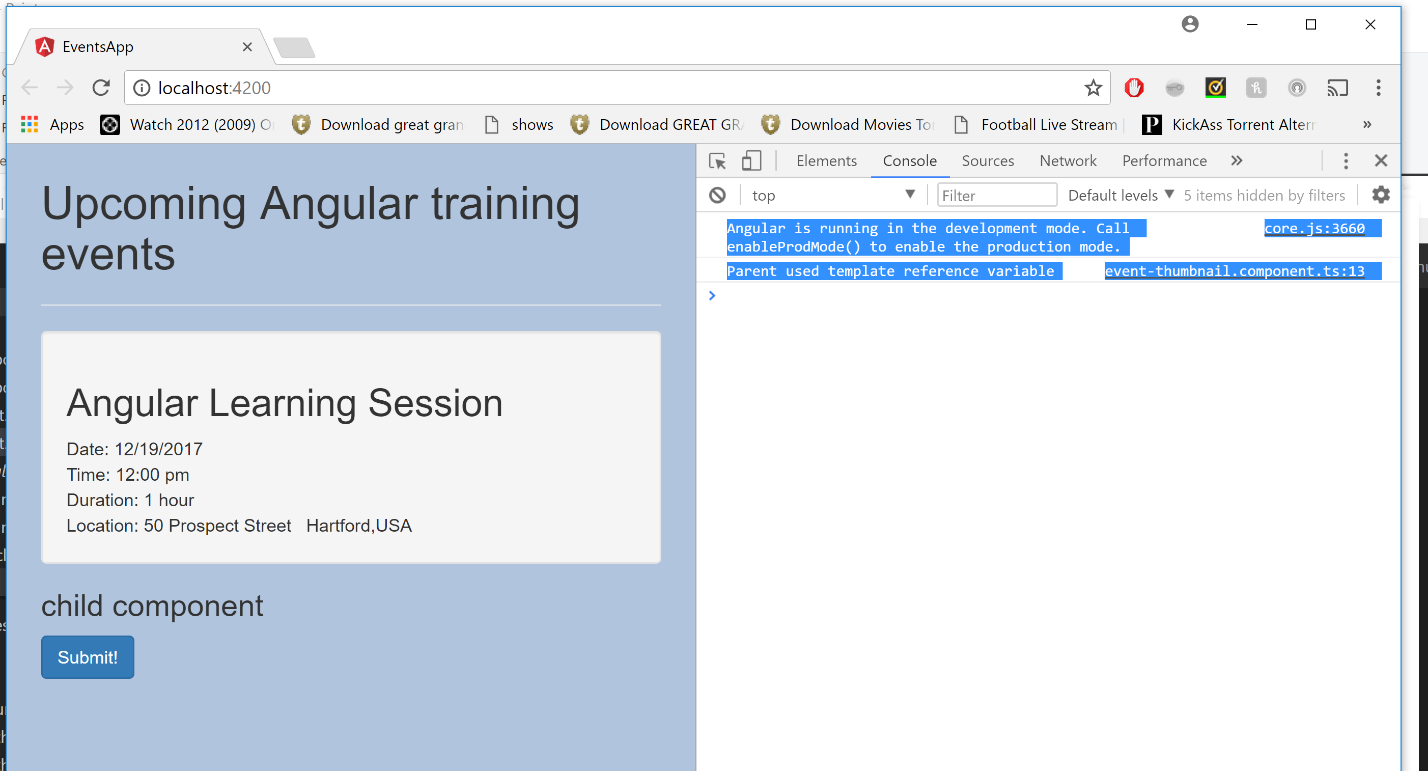
1. Run the application and click on button to observe the log.
2. Now you can also use template variables for binding on parent component by using public property of a child component. Lets add a Public Property on child component “events-thumbnail.component.ts”

someProperty: any = "child component"

1. Add the binding to the parent component html. Add below code in event-list.component.html

<h3>{{thumbnail.someProperty}}</h3>

1. Observe the Application:



1. Lets clean up the code which we added for template variables

Styling Components and Encapsulation.

Angular takes care of encapsulating css in individual components where it introduces namespacing with styles with each components.

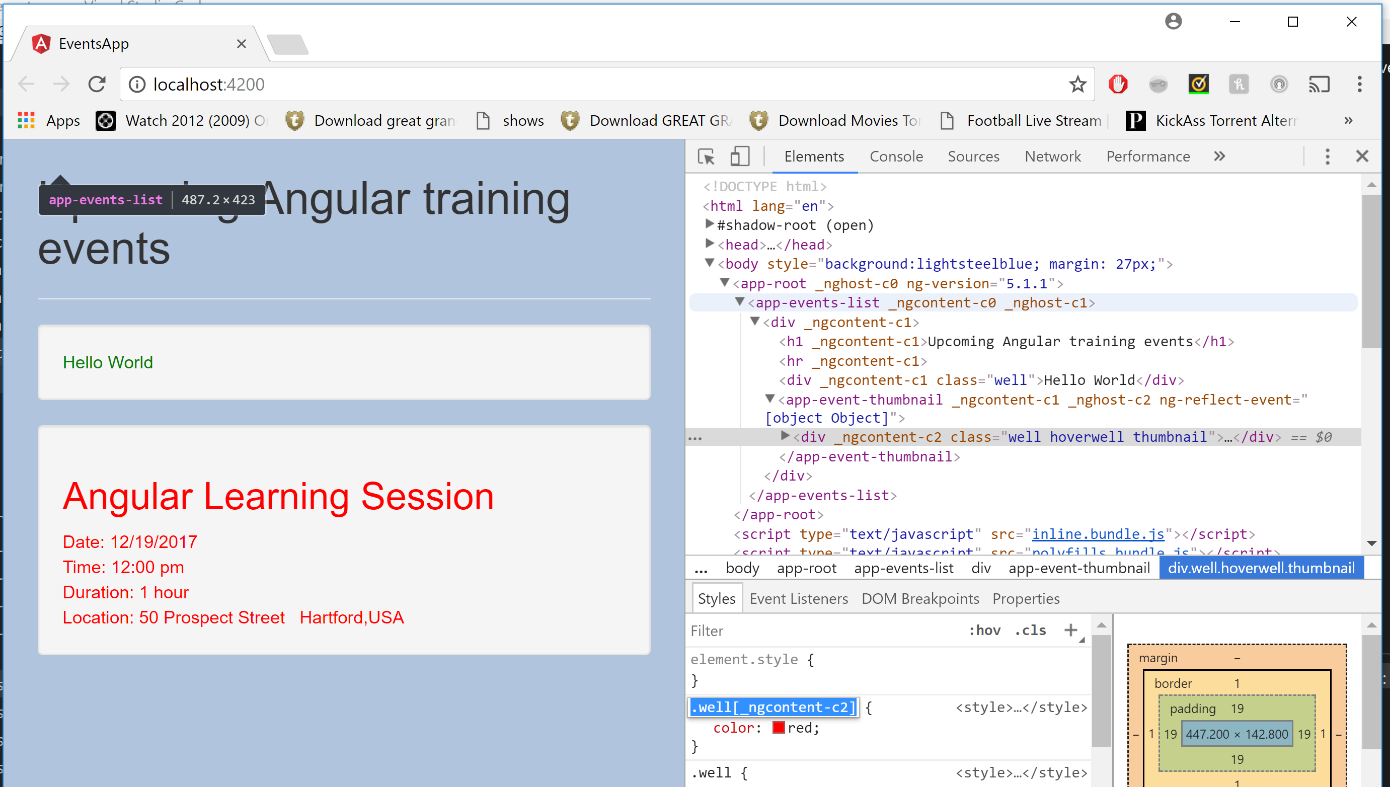
1. Lets add a class with same name in parent and child component with different color
2. event-list.commponent.css

.well {color:Green}

1. events-thumbnail.component.ts

.well {color:red}

1. Observe application:



Adding a Site header:

Just to demonstrate that app component is not different than other component, lets add siteheader with navigation element. Eventually when we add routing to our application this component will be replaced with router outlet component. When we navigate around the site this component will be used to vaigate between different component tree.

1. Add a component using command prompt:

ng g component event-nav

1. Add below code in events.nav.component.html

<div class="navbar navbar-default">

<div class="container-fluid">

<div class="navbar-header">

<a class="navbar-brand" >Angular Training Events</a>

</div>

<div class="collapse navbar-collapse">

<ul class="nav navbar-nav">

<li>

<a >All Events</a>

</li>

<li><a href="">Create Event</a></li>

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown" >

Events

<span class="caret"></span>

</a>

<ul class="dropdown-menu">

<li >

<a href="">Angular Training</a>

</li>

</ul>

</li>

</ul>

<div class="navbar-header navbar-right">

<ul class="nav navbar-nav">

<li>

<a>Welcome Bipul</a>

</li>

</ul>

</div>

<form id="searchForm" class="navbar-form navbar-right" >

<div class="form-group">

<input type="text" class="form-control" placeholder="Search Sessions" >

</div>

<button class="btn btn-default" >

Search

</button>

</form>

</div>

</div>

</div>

1. Add following code in CSS of the nav component:

.nav.navbar-nav {font-size: 15px}

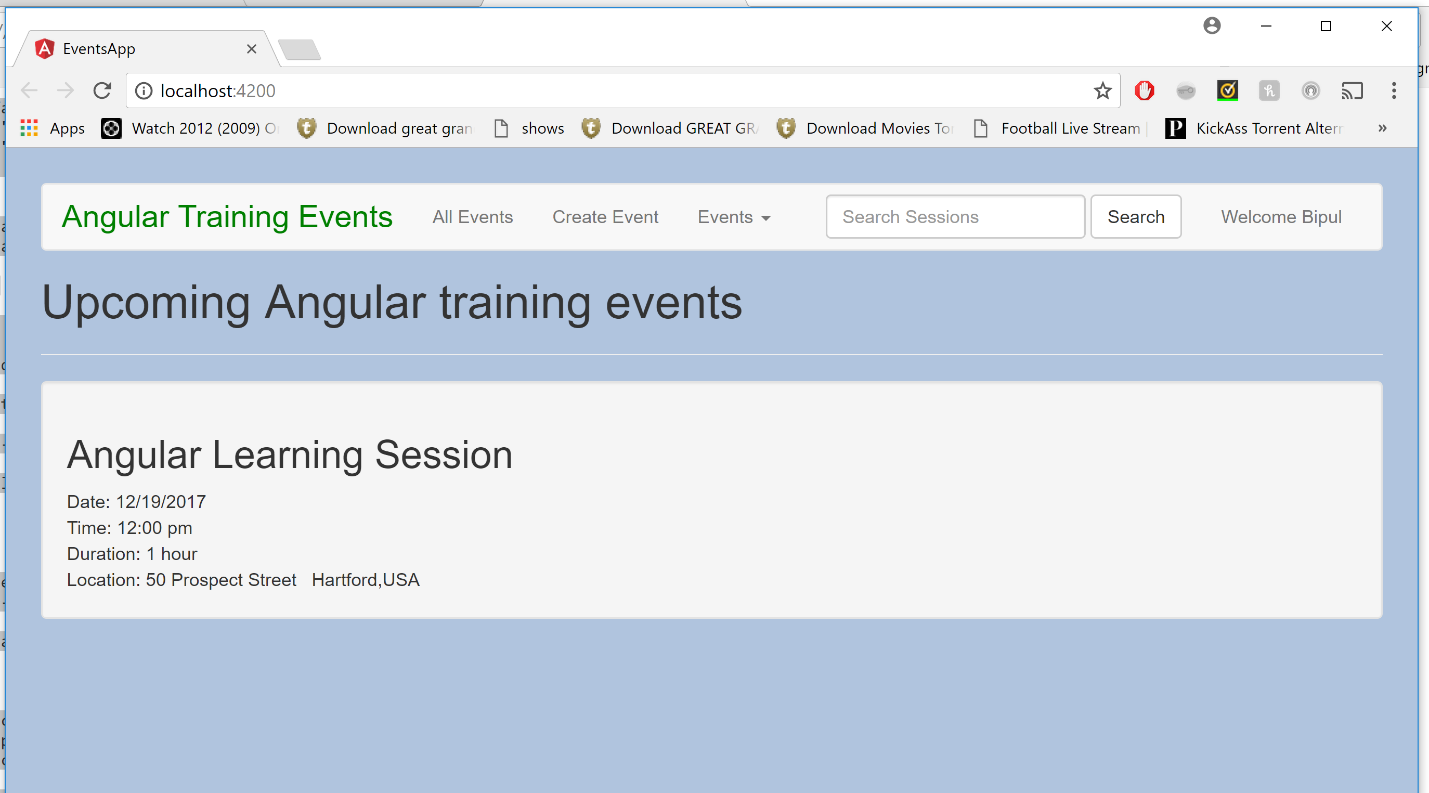
#searchForm {margin-right: 150px}

@media (max-width: 1200px) {#searchForm {display:none}}

1. Add the siteheader in app.component.html

<app-event-nav></app-event-nav>

1. Run the application see below changes:



EXPLORING THE ANGULAR TEMPLATE SYNTAX

Repeating Data with ngFOR

1. So far we have contructed the application above and made it ready so that we can display multiple events in the application. Copy below events array object into the parent component events-list.component.ts. IN normal scenario this data will come from services.

events = [

{

id: 1,

name: 'Angular Learning Session',

date: '12/19/2017',

time: '12:00 pm',

price: 100,

imageUrl: '/app/assets/angular.png',

location: {

address: '50 Prospect Street',

city: 'Hartford',

country: 'USA'

},

sessions: [

{

id: 1,

name: "Introduction, Platform Overview and Architecture",

presenter: "Ajith and Bipul",

duration: 1,

level: "Intermediate",

abstract: `Learn all basic concepts of Angular with Angular Architecture.

Basic Concepts on Component, Binding, Dependency Injection etc`,

voters: ['bradgreen', 'igorminar', 'martinfowler']

},

{

id: 2,

name: "Understanding Angular CLI ",

presenter: "Janny and Bipul",

duration: 1,

level: "Intermediate",

abstract: `We all will know about setting up Angular Application.

Using Angular CLI command for various operations on Angular Application`,

voters: ['johnpapa', 'bradgreen', 'igorminar', 'martinfowler']

},

{

id: 3,

name: "Components, Templates, Services",

presenter: "Bipul Kumar",

duration: 1,

level: "Intermediate",

abstract: `Deepdive into Angular Application Component Structure and

Hierarchy, Interpolation, Expression, Events, Statements, Services`,

voters: []

},

{

id: 4,

name: "TypeScript Overview, VS Code, Commands",

presenter: "Ashok Deviah",

duration: 2,

level: "Advanced",

abstract: `Deepdive into understanding TypeScript Overview, VS Code, Commands`,

voters: []

},

{

id: 5,

name: "Routing and Component Communication ",

presenter: "Krishna/Ashok",

duration: 2,

level: "Intermediate",

abstract: `Deepdive into Angular Application Routing and Component Communication`,

voters: ['bradgreen', 'igorminar']

}

]

},

{

id: 2,

name: 'Angular Training- St Paul',

date: '4/15/2037',

time: '9:00 am',

price: 950.00,

imageUrl: '/app/assets/images/ng-nl.png',

location: {

address: 'Travelers Inc',

city: 'St Paul',

country: 'USA'

},

sessions: [

{

id: 1,

name: "Testing Angular 4 Workshop",

presenter: "Pascal Precht & Christoph Bergdorf",

duration: 4,

level: "Beginner",

abstract: `In this 6 hour workshop you will learn not only how to test Angular 4,

you will also learn how to make the most of your team's efforts. Other topics

will be convincing your manager that testing is a good idea, and using the new

protractor tool for end to end testing.`,

voters: ['bradgreen', 'igorminar']

},

{

id: 2,

name: "Angular 4 and Firebase",

presenter: "David East",

duration: 3,

level: "Intermediate",

abstract: `In this workshop, David East will show you how to use Angular with the new

ultra-real-time 5D Firebase back end, hosting platform, and wine recommendation engine.`,

voters: ['bradgreen', 'igorminar', 'johnpapa']

},

{

id: 3,

name: "Reading the Angular 4 Source",

presenter: "Patrick Stapleton",

duration: 2,

level: "Intermediate",

abstract: `Angular 4's source code may be over 25 million lines of code, but it's really

a lot easier to read and understand then you may think. Patrick Stapleton will talk

about his secretes for keeping up with the changes, and navigating around the code.`,

voters: ['martinfowler']

},

{

id: 4,

name: "Hail to the Lukas",

presenter: "Lukas Ruebbelke",

duration: 1,

level: "Beginner",

abstract: `In this session, Lukas will present the

secret to being awesome, and how he became the President

of the United States through his amazing programming skills,

showing how you too can be success with just attitude.`,

voters: ['bradgreen']

},

]

},

{

id: 3,

name: 'NodeJS and Mongo DB',

date: '5/4/2037',

time: '9:00 am',

price: 759.00,

imageUrl: '/app/assets/images/ng-conf.png',

location: {

address: 'The Palatial America Hotel',

city: 'Salt Lake City',

country: 'USA'

},

sessions: [

{

id: 1,

name: "How Elm Powers Angular 4",

presenter: "Murphy Randle",

duration: 2,

level: "Intermediate",

abstract: `We all know that Angular is written in Elm, but did you

know how the source code is really written? In this exciting look

into the internals of Angular 4, we'll see exactly how Elm powers

the framework, and what you can do to take advantage of this knowledge.`,

voters: ['bradgreen', 'martinfowler', 'igorminar']

},

{

id: 2,

name: "Angular and React together",

presenter: "Jamison Dance",

duration: 2,

level: "Intermediate",

abstract: `React v449.6 has just been released. Let's see how to use

this new version with Angular to create even more impressive applications.`,

voters: ['bradgreen', 'martinfowler']

},

{

id: 3,

name: "Redux Woes",

presenter: "Rob Wormald",

duration: 1,

level: "Intermediate",

abstract: `Everyone is using Redux for everything from Angular to React to

Excel macros, but you're still having trouble grasping it? We'll take a look

at how farmers use Redux when harvesting grain as a great introduction to

this game changing technology.`,

voters: ['bradgreen', 'martinfowler', 'johnpapa']

},

{

id: 4,

name: "ng-wat again!!",

presenter: "Shai Reznik",

duration: 1,

level: "Beginner",

abstract: `Let's take a look at some of the stranger pieces of Angular 4,

including neural net nets, Android in Androids, and using pipes with actual pipes.`,

voters: ['bradgreen', 'martinfowler', 'igorminar', 'johnpapa']

},

{

id: 5,

name: "Dressed for Success",

presenter: "Ward Bell",

duration: 2,

level: "Beginner",

abstract: `Being a developer in 2037 is about more than just writing bug-free code.

You also have to look the part. In this amazing expose, Ward will talk you through

how to pick out the right clothes to make your coworkers and boss not only

respect you, but also want to be your buddy.`,

voters: ['bradgreen', 'martinfowler']

},

{

id: 6,

name: "These aren't the directives you're looking for",

presenter: "John Papa",

duration: 2,

level: "Intermediate",

abstract: `Coinciding with the release of Star Wars Episode 18, this talk will show how

to use directives in your Angular 4 development while drawing lessons from the new movie,

featuring all your favorite characters like Han Solo's ghost and Darth Jar Jar.`,

voters: ['bradgreen', 'martinfowler']

},

]

},

{

id: 4,

name: 'UN Angular Summit',

date: '6/10/2037',

time: '8:00 am',

price: 800.00,

imageUrl: '/app/assets/images/basic-shield.png',

location: {

address: 'The UN Angular Center',

city: 'New York',

country: 'USA'

},

sessions: [

{

id: 1,

name: "Diversity in Tech",

presenter: "Sir Dave Smith",

duration: 2,

level: "Beginner",

abstract: `Yes, we all work with cyborgs and androids and Martians, but

we probably don't realize that sometimes our internal biases can make it difficult for

these well-designed coworkers to really feel at home coding alongside us. This talk will

look at things we can do to recognize our biases and counteract them.`,

voters: ['bradgreen', 'igorminar']

},

{

id: 2,

name: "World Peace and Angular",

presenter: "US Secretary of State Zach Galifianakis",

duration: 2,

level: "Beginner",

abstract: `Angular has been used in most of the major peace brokering that has

happened in the last decade, but there is still much we can do to remove all

war from the world, and Angular will be a key part of that effort.`,

voters: ['bradgreen', 'igorminar', 'johnpapa']

},

{

id: 3,

name: "Using Angular with Androids",

presenter: "Dan Wahlin",

duration: 3,

level: "Advanced",

abstract: `Androids may do everything for us now, allowing us to spend all day playing

the latest Destiny DLC, but we can still improve the massages they give and the handmade

brie they make using Angular 4. This session will show you how.`,

voters: ['igorminar', 'johnpapa']

},

]

},

{

id: 5,

name: 'Angular Training - Hawaii',

date: '2/10/2037',

time: '9:00 am',

price: 400.00,

imageUrl: '/app/assets/images/ng-vegas.png',

location: {

address: 'Waikiki Beach',

city: 'Hawaii',

country: 'USA'

},

sessions: [

{

id: 1,

name: "Gambling with Angular",

presenter: "John Papa",

duration: 1,

level: "Intermediate",

abstract: `No, this talk isn't about slot machines. We all know that

Angular is used in most waiter-bots and coke vending machines, but

did you know that was also used to write the core engine in the majority

of voting machines? This talk will look at how all presidential elections

are now determined by Angular code.`,

voters: ['bradgreen', 'igorminar']

},

{

id: 2,

name: "Angular 4 in 60ish Minutes",

presenter: "Dan Wahlin",

duration: 2,

level: "Beginner",

abstract: `Get the skinny on Angular 4 for anyone new to this great new technology.

Dan Wahlin will show you how you can get started with Angular in 60ish minutes,

guaranteed!`,

voters: ['bradgreen', 'igorminar', 'johnpapa']

}

]

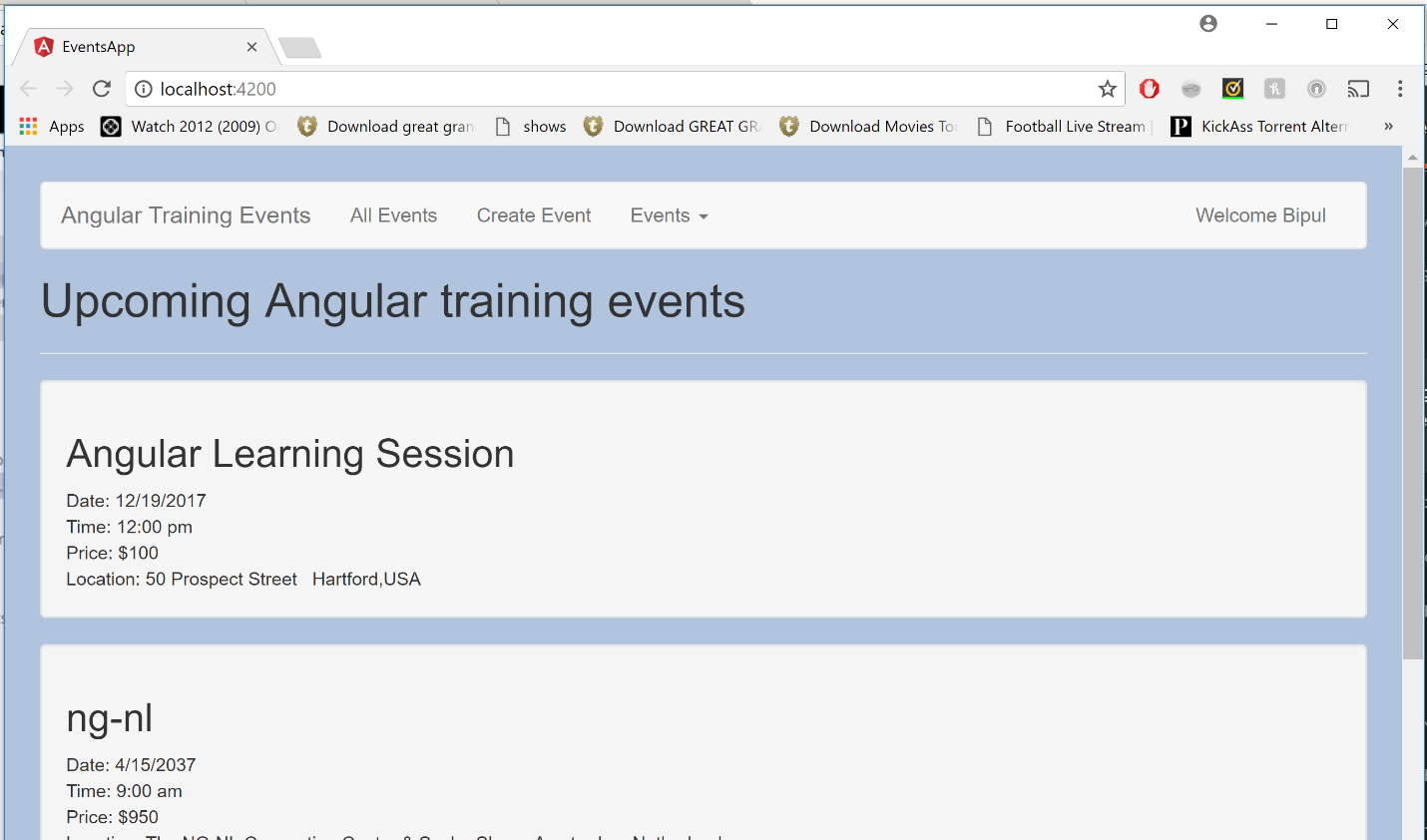
}

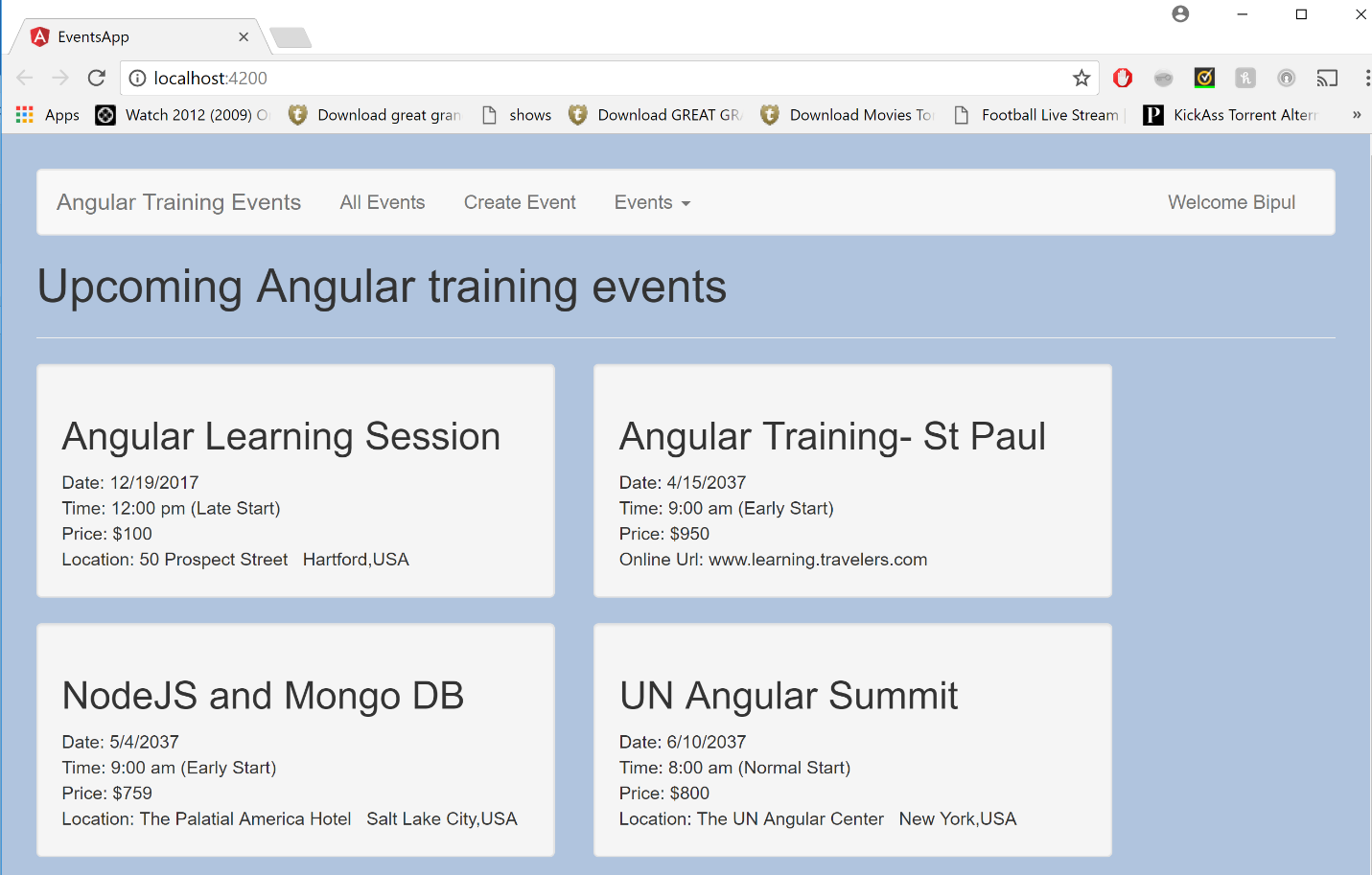
]

1. So Our page is broken right now as we are using the event as single object and we changed it to array. So lets make the change in the component.html file of parent and use ngFor.
2. Update below code to events-list.component.html and Run Application.

<app-event-thumbnail \*ngFor = "let event of events" #thumbnail [event] = "event"></app-event-thumbnail>

1. \* in front of ngFor Signifies that its an structural directive. Structural directives change the shape of DOM





1. Lets do some styling on the element and use bootstrap and make changes to ngFor. Update code of events-list.component.html

<div>

<h1>Upcoming Angular training events</h1>

<hr/>

<div class="row">

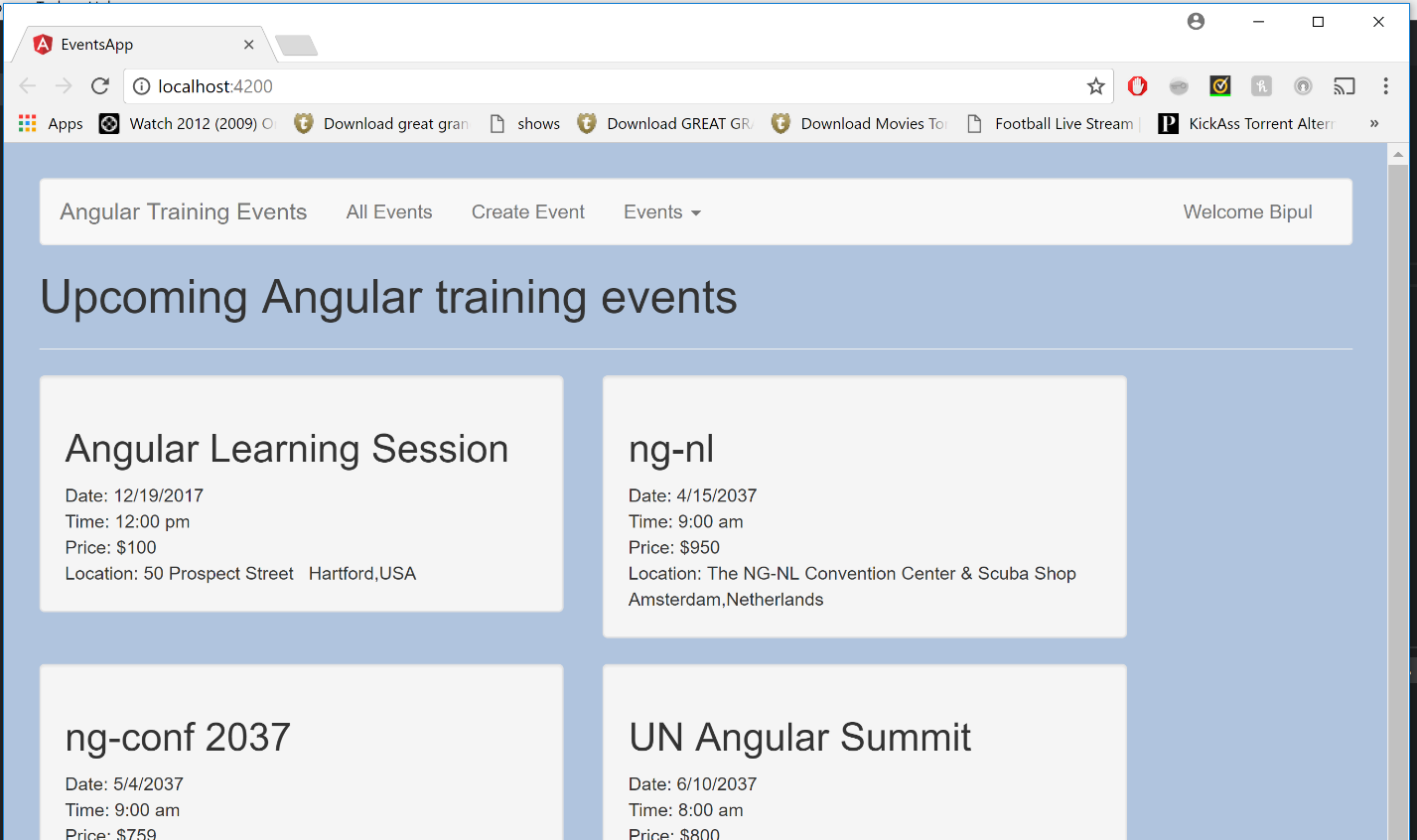
<div \*ngFor="let event of events" class="col-md-5">

<app-event-thumbnail #thumbnail [event]="event"></app-event-thumbnail>

</div>

</div>

</div>



Handling Null Values with safe navigator operator:

Make changes to one of the properties and make it as null and observe how the application behaves

Remove event from event-thumbnail.component.html and see how the application behaves . See the console to see the errors.

1. Make changes to event-thumbnail.component.html as below. Observe in case of event being null we don’t have to put operator on location. Angular takes care of null check

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

<span>Location: {{event?.location.address}}</span>

1. But in case of location missing for some data we need to add operator for location as well. Lets change our data and add online Url for One of the data . Make changes to Second object in events array. Like below:

id: 2,

name: 'Angular Training- St Paul',

date: '4/15/2037',

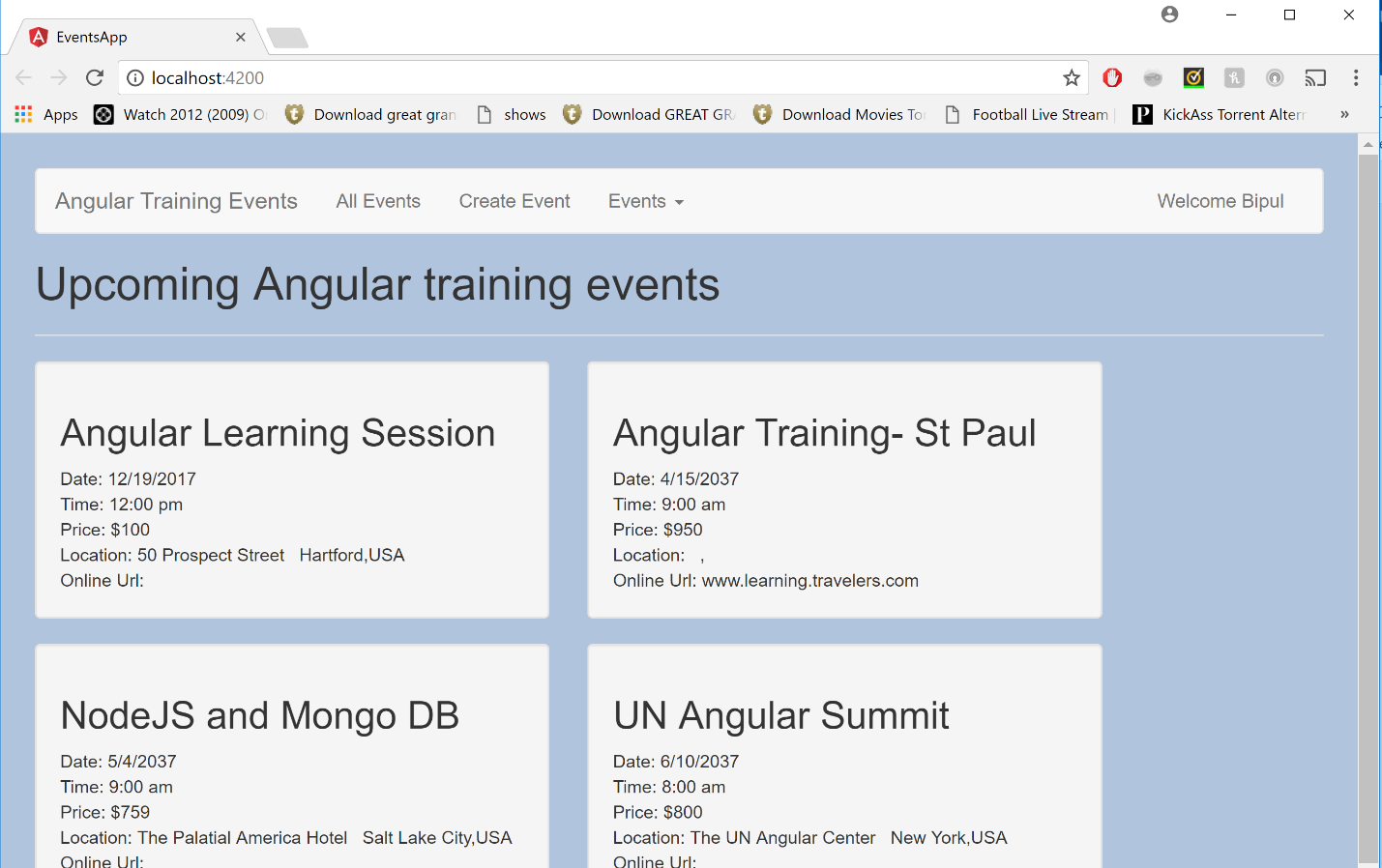
time: '9:00 am',

price: 950.00,

imageUrl: '/app/assets/images/ng-nl.png',

onlineUrl: 'www.learning.travelers.com',

1. Now Observe that the application works fine but we see blank Online Url for other templates which we will fix in next exercise.



Hiding and Showing Content with ngIf

In above exercise we saw Blank Online Url and Location Showing up across all events. To fic this we will use ngIf. \*ngIf is also a structural directive that allows us to show content only when the expression is evaluated true.

1. Update the Code For Location and Online Url as below. Observe that ngIF didn’t only hide the HTML but also commented out the rendering so that it improves performance.

<div class="well hoverwell thumbnail">

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

<div>Date: {{event?.date}}</div>

<div>Time: {{event?.time}}</div>

<div>Price: ${{event?.price}}</div>

<div \*ngIf = "event?.location">

<span>Location: {{event?.location?.address}}</span>

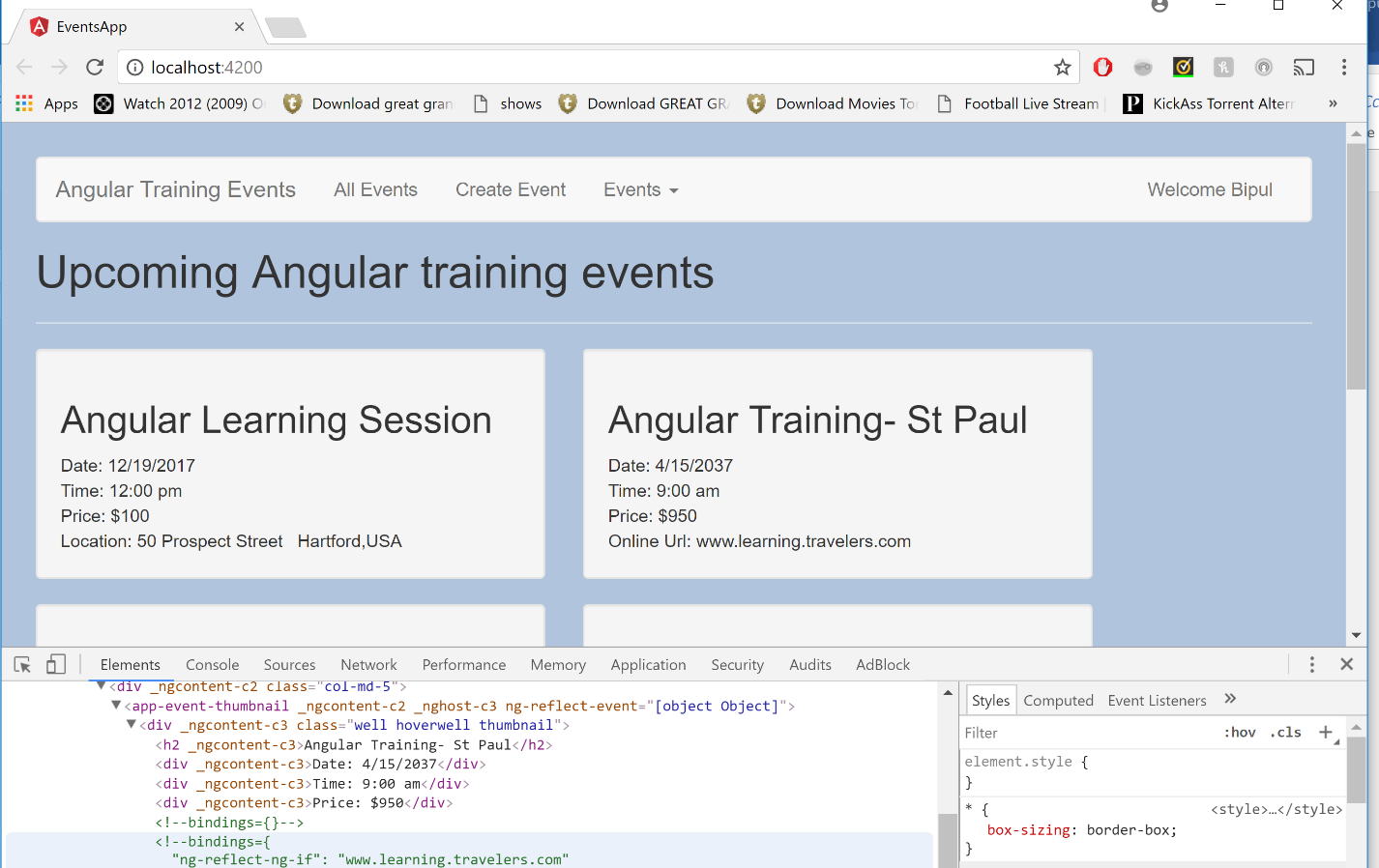
<span>&nbsp;</span>

<span>{{event.location?.city}},{{event?.location?.country}}</span>

</div>

<div \*ngIf = "event?.onlineUrl" >Online Url: {{event?.onlineUrl}}</div>

</div>



Hiding Component with the [Hidden] binding

If we are hiding and showing component frequently enough then its easy to hide components using binding. In Angular we can bind the HTML Property like hidden with Boolean values which will hide and show the component. Observe HTML Component.

1. Make Below code changes in child component.

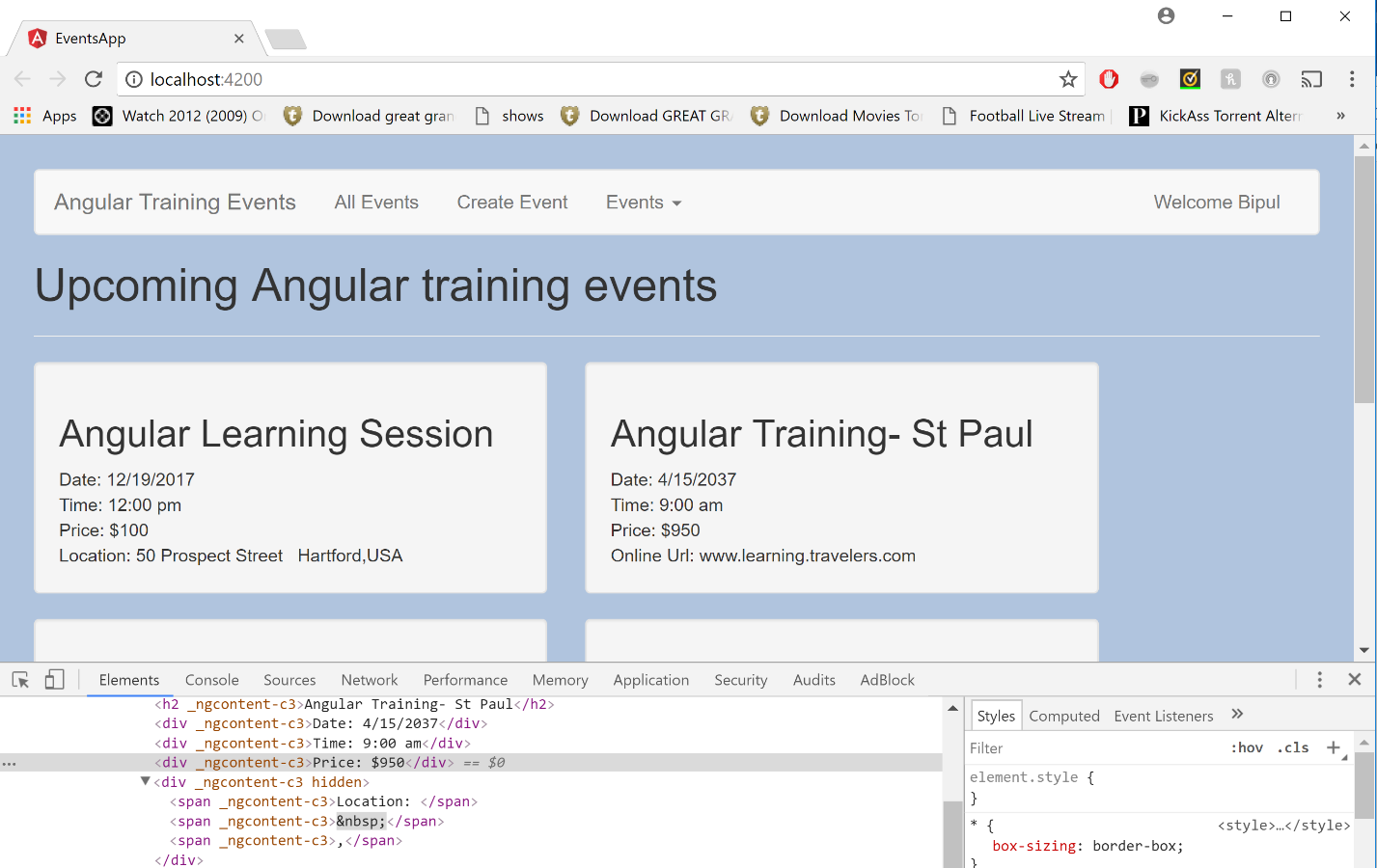
<div [hidden] = "!event?.location">

<span>Location: {{event?.location?.address}}</span>

<span>&nbsp;</span>

<span>{{event.location?.city}},{{event?.location?.country}}</span>

</div>



Hiding and Showing Component using ngSwitch

1. Certain time we want to show data where we can use ngSwitch to create the DOM Element based on the evaluating expression. Update below code in evets.thumbnail.component.html. Observe we are using ngSwitch on the Parent element and using ngSwitchCase which use the evaluated epression.

<div class="well hoverwell thumbnail">

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

<div>Date: {{event?.date}}</div>

<div [ngSwitch] = "event?.time">

Time: {{event?.time}}

<span \*ngSwitchCase = "'9:00 am'">(Early Start)</span>

<span \*ngSwitchCase = "'12:00 pm'">(Late Start)</span>

<span \*ngSwitchDefault>(Normal Start)</span>

</div>

<div>Price: ${{event?.price}}</div>

<div \*ngIf = "event?.location">

<span>Location: {{event?.location?.address}}</span>

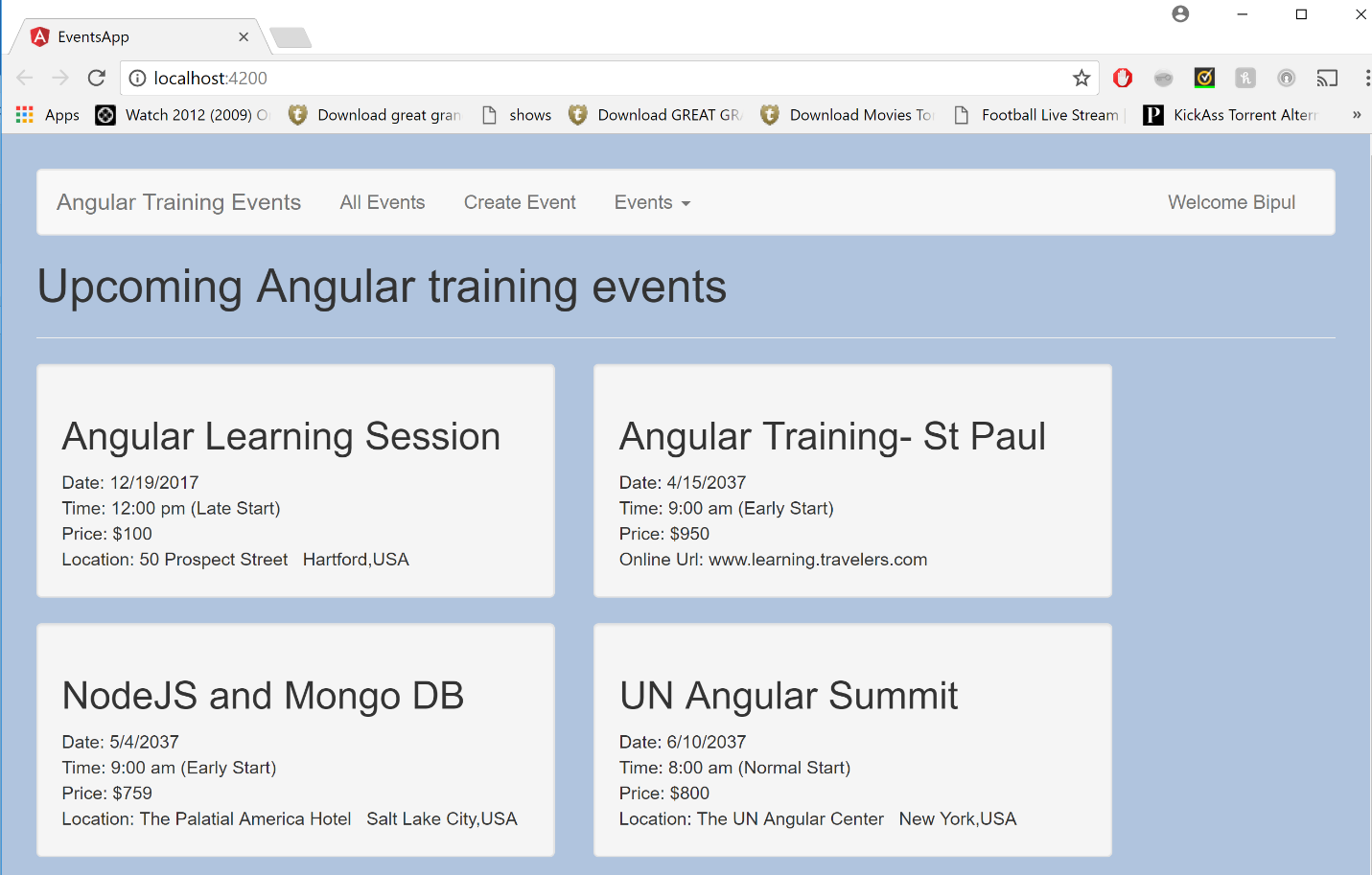
<span>&nbsp;</span>

<span>{{event.location?.city}},{{event?.location?.country}}</span>

</div>

<div \*ngIf = "event?.onlineUrl" >Online Url: {{event?.onlineUrl}}</div>

</div>



Styling Components with ngClass

Another feature provided by Angular where we can apply classes based on certain evaluation. Class Property can be bound directly to application or we can use ngClass instead as well

1. Update below code in events-thumbnail.component.html. Observe we already have a class well and ngClass is adding more classes to it.

<div class= "well" [ngClass]="getStartTimeClasses()" [ngSwitch] = "event?.time">

1. The value assignment can be evaluated or passed directly as expression to ngClass. Lets see how the expression is evaluated and how many ways we can pass in the data for classes to be applied to the component. Add below code in events-thumbnail.component.ts

getStartTimeClasses() {

if (this.event && this.event.time === '8:00 am') {

return 'green bold';

} else {

if (this.event && this.event.time === '12:00 pm') {

return ['red', 'bold'];

}

return '';

}

}

Styling Components with ngStyle

Another feature provided by Angular where we can apply style based on certain evaluation. Style Property can be bound directly to application or we can use ngClass instead as well

1. Update below code in events-thumbnail.component.html. We can have a Style already applied to the element and the ngStyle can be added in addition.

<div [ngStyle]="getStartTimeStyle()" [ngSwitch] = "event?.time">

1. Add getStartTimeStyle in events-thumbnail.component.ts. Observe we cannot send null and TS is giving error so we assign any as return type to the method.

getStartTimeStyle() : any {

if (this.event && this.event.time === '8:00 am') {

return {color: 'green', 'font-weight' : 'bold' };

} else {

if (this.event && this.event.time === '12:00 pm') {

return {color: 'blue', 'font-weight' : 'bold' };

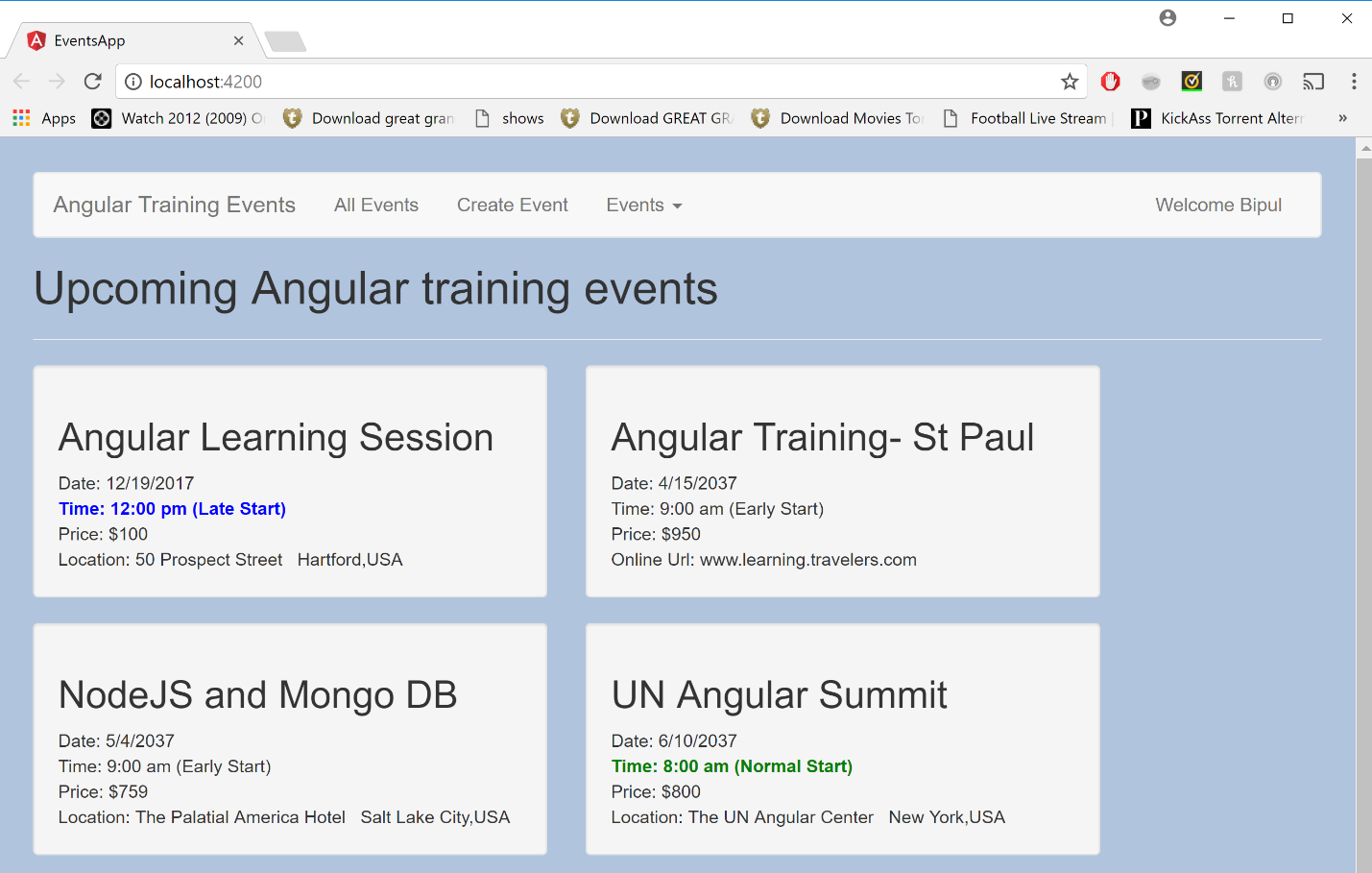
}

return '';

}

}

Observe the Application with above changes:



Creating reusable Angular Services

Services allows to define business logic in separate file and inject whatever service when we need it.

1. Let’s generate a service using command in VS Code Terminal

ng g service service\event

1. Let’s move the events Array from events-list.component.ts and move it as constant to the Service file. And return the EVENTS object from service. Eventually it will be returned as a http restful service method response. Update the Service file with below code.
2. Observe the Injectable decorator (which is a good practice). Adding this decorator is important for any service which we want to inject to the components or and another service. This is not required for the service though as this decorator is only required when you inject a service which also injects other service as dependencies of its own. By this we mean say if this service had a constructor like Constructor (private http : Http){}

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

@Injectable()

export class EventService {

constructor() { }

getEvents(){

return EVENTS;

}

}

const EVENTS = [

{

id: 1,

name: 'Angular Learning Session',

date: '12/19/2017',

time: '12:00 pm',

price: 100,

imageUrl: '/app/assets/angular.png',

location: {

address: '50 Prospect Street',

city: 'Hartford',

country: 'USA'

},

sessions: [

{

id: 1,

name: "Introduction, Platform Overview and Architecture",

presenter: "Ajith and Bipul",

duration: 1,

level: "Intermediate",

abstract: `Learn all basic concepts of Angular with Angular Architecture.

Basic Concepts on Component, Binding, Dependency Injection etc`,

voters: ['bradgreen', 'igorminar', 'martinfowler']

},

{

id: 2,

name: "Understanding Angular CLI ",

presenter: "Janny and Bipul",

duration: 1,

level: "Intermediate",

abstract: `We all will know about setting up Angular Application.

Using Angular CLI command for various operations on Angular Application`,

voters: ['johnpapa', 'bradgreen', 'igorminar', 'martinfowler']

},

{

id: 3,

name: "Components, Templates, Services",

presenter: "Bipul Kumar",

duration: 1,

level: "Intermediate",

abstract: `Deepdive into Angular Application Component Structure and

Hierarchy, Interpolation, Expression, Events, Statements, Services`,

voters: []

},

{

id: 4,

name: "TypeScript Overview, VS Code, Commands",

presenter: "Ashok Deviah",

duration: 2,

level: "Advanced",

abstract: `Deepdive into understanding TypeScript Overview, VS Code, Commands`,

voters: []

},

{

id: 5,

name: "Routing and Component Communication ",

presenter: "Krishna/Ashok",

duration: 2,

level: "Intermediate",

abstract: `Deepdive into Angular Application Routing and Component Communication`,

voters: ['bradgreen', 'igorminar']

}

]

},

{

id: 2,

name: 'Angular Training- St Paul',

date: '4/15/2037',

time: '9:00 am',

price: 950.00,

imageUrl: '/app/assets/images/ng-nl.png',

onlineUrl: 'www.learning.travelers.com',

sessions: [

{

id: 1,

name: "Testing Angular 4 Workshop",

presenter: "Pascal Precht & Christoph Bergdorf",

duration: 4,

level: "Beginner",

abstract: `In this 6 hour workshop you will learn not only how to test Angular 4,

you will also learn how to make the most of your team's efforts. Other topics

will be convincing your manager that testing is a good idea, and using the new

protractor tool for end to end testing.`,

voters: ['bradgreen', 'igorminar']

},

{

id: 2,

name: "Angular 4 and Firebase",

presenter: "David East",

duration: 3,

level: "Intermediate",

abstract: `In this workshop, David East will show you how to use Angular with the new

ultra-real-time 5D Firebase back end, hosting platform, and wine recommendation engine.`,

voters: ['bradgreen', 'igorminar', 'johnpapa']

},

{

id: 3,

name: "Reading the Angular 4 Source",

presenter: "Patrick Stapleton",

duration: 2,

level: "Intermediate",

abstract: `Angular 4's source code may be over 25 million lines of code, but it's really

a lot easier to read and understand then you may think. Patrick Stapleton will talk

about his secretes for keeping up with the changes, and navigating around the code.`,

voters: ['martinfowler']

},

{

id: 4,

name: "Hail to the Lukas",

presenter: "Lukas Ruebbelke",

duration: 1,

level: "Beginner",

abstract: `In this session, Lukas will present the

secret to being awesome, and how he became the President

of the United States through his amazing programming skills,

showing how you too can be success with just attitude.`,

voters: ['bradgreen']

},

]

},

{

id: 3,

name: 'NodeJS and Mongo DB',

date: '5/4/2037',

time: '9:00 am',

price: 759.00,

imageUrl: '/app/assets/images/ng-conf.png',

location: {

address: 'The Palatial America Hotel',

city: 'Salt Lake City',

country: 'USA'

},

sessions: [

{

id: 1,

name: "How Elm Powers Angular 4",

presenter: "Murphy Randle",

duration: 2,

level: "Intermediate",

abstract: `We all know that Angular is written in Elm, but did you

know how the source code is really written? In this exciting look

into the internals of Angular 4, we'll see exactly how Elm powers

the framework, and what you can do to take advantage of this knowledge.`,

voters: ['bradgreen', 'martinfowler', 'igorminar']

},

{

id: 2,

name: "Angular and React together",

presenter: "Jamison Dance",

duration: 2,

level: "Intermediate",

abstract: `React v449.6 has just been released. Let's see how to use

this new version with Angular to create even more impressive applications.`,

voters: ['bradgreen', 'martinfowler']

},

{

id: 3,

name: "Redux Woes",

presenter: "Rob Wormald",

duration: 1,

level: "Intermediate",

abstract: `Everyone is using Redux for everything from Angular to React to

Excel macros, but you're still having trouble grasping it? We'll take a look

at how farmers use Redux when harvesting grain as a great introduction to

this game changing technology.`,

voters: ['bradgreen', 'martinfowler', 'johnpapa']

},

{

id: 4,

name: "ng-wat again!!",

presenter: "Shai Reznik",

duration: 1,

level: "Beginner",

abstract: `Let's take a look at some of the stranger pieces of Angular 4,

including neural net nets, Android in Androids, and using pipes with actual pipes.`,

voters: ['bradgreen', 'martinfowler', 'igorminar', 'johnpapa']

},

{

id: 5,

name: "Dressed for Success",

presenter: "Ward Bell",

duration: 2,

level: "Beginner",

abstract: `Being a developer in 2037 is about more than just writing bug-free code.

You also have to look the part. In this amazing expose, Ward will talk you through

how to pick out the right clothes to make your coworkers and boss not only

respect you, but also want to be your buddy.`,

voters: ['bradgreen', 'martinfowler']

},

{

id: 6,

name: "These aren't the directives you're looking for",

presenter: "John Papa",

duration: 2,

level: "Intermediate",

abstract: `Coinciding with the release of Star Wars Episode 18, this talk will show how

to use directives in your Angular 4 development while drawing lessons from the new movie,

featuring all your favorite characters like Han Solo's ghost and Darth Jar Jar.`,

voters: ['bradgreen', 'martinfowler']

},

]

},

{

id: 4,

name: 'UN Angular Summit',

date: '6/10/2037',

time: '8:00 am',

price: 800.00,

imageUrl: '/app/assets/images/basic-shield.png',

location: {

address: 'The UN Angular Center',

city: 'New York',

country: 'USA'

},

sessions: [

{

id: 1,

name: "Diversity in Tech",

presenter: "Sir Dave Smith",

duration: 2,

level: "Beginner",

abstract: `Yes, we all work with cyborgs and androids and Martians, but

we probably don't realize that sometimes our internal biases can make it difficult for

these well-designed coworkers to really feel at home coding alongside us. This talk will

look at things we can do to recognize our biases and counteract them.`,

voters: ['bradgreen', 'igorminar']

},

{

id: 2,

name: "World Peace and Angular",

presenter: "US Secretary of State Zach Galifianakis",

duration: 2,

level: "Beginner",

abstract: `Angular has been used in most of the major peace brokering that has

happened in the last decade, but there is still much we can do to remove all

war from the world, and Angular will be a key part of that effort.`,

voters: ['bradgreen', 'igorminar', 'johnpapa']

},

{

id: 3,

name: "Using Angular with Androids",

presenter: "Dan Wahlin",

duration: 3,

level: "Advanced",

abstract: `Androids may do everything for us now, allowing us to spend all day playing

the latest Destiny DLC, but we can still improve the massages they give and the handmade

brie they make using Angular 4. This session will show you how.`,

voters: ['igorminar', 'johnpapa']

},

]

},

{

id: 5,

name: 'Angular Training - Hawaii',

date: '2/10/2037',

time: '9:00 am',

price: 400.00,

imageUrl: '/app/assets/images/ng-vegas.png',

location: {

address: 'Waikiki Beach',

city: 'Hawaii',

country: 'USA'

},

sessions: [

{

id: 1,

name: "Gambling with Angular",

presenter: "John Papa",

duration: 1,

level: "Intermediate",

abstract: `No, this talk isn't about slot machines. We all know that

Angular is used in most waiter-bots and coke vending machines, but

did you know that was also used to write the core engine in the majority

of voting machines? This talk will look at how all presidential elections

are now determined by Angular code.`,

voters: ['bradgreen', 'igorminar']

},

{

id: 2,

name: "Angular 4 in 60ish Minutes",

presenter: "Dan Wahlin",

duration: 2,

level: "Beginner",

abstract: `Get the skinny on Angular 4 for anyone new to this great new technology.

Dan Wahlin will show you how you can get started with Angular in 60ish minutes,

guaranteed!`,

voters: ['bradgreen', 'igorminar', 'johnpapa']

}

]

}

]

1. Now go and see app module and make below entries :

import { EventService } from './service/event.service';

providers: [EventService],

1. Now let’s Inject this service in events-list.component.ts

export class EventsListComponent implements OnInit {

events:any[];

constructor(private eventservice: EventService) {

this.events = this.eventservice.getEvents();

}

// the private varible is equal to below code

// eventservice

// constructor() {

// this.eventservice = eventservice

// }

ngOnInit() {

}

}

1. Now run the application using command ng serve –o or npm start
2. It’s not a good idea at all to call the service in the contractor as services as it takes time to make http calls. Therefore we will use one of the lifecycle hooks called ngOnInit() and call service from there, but we will still need the constructor

Just for example:

export class App implements OnInit{

constructor(){

//called first time before the ngOnInit()

}

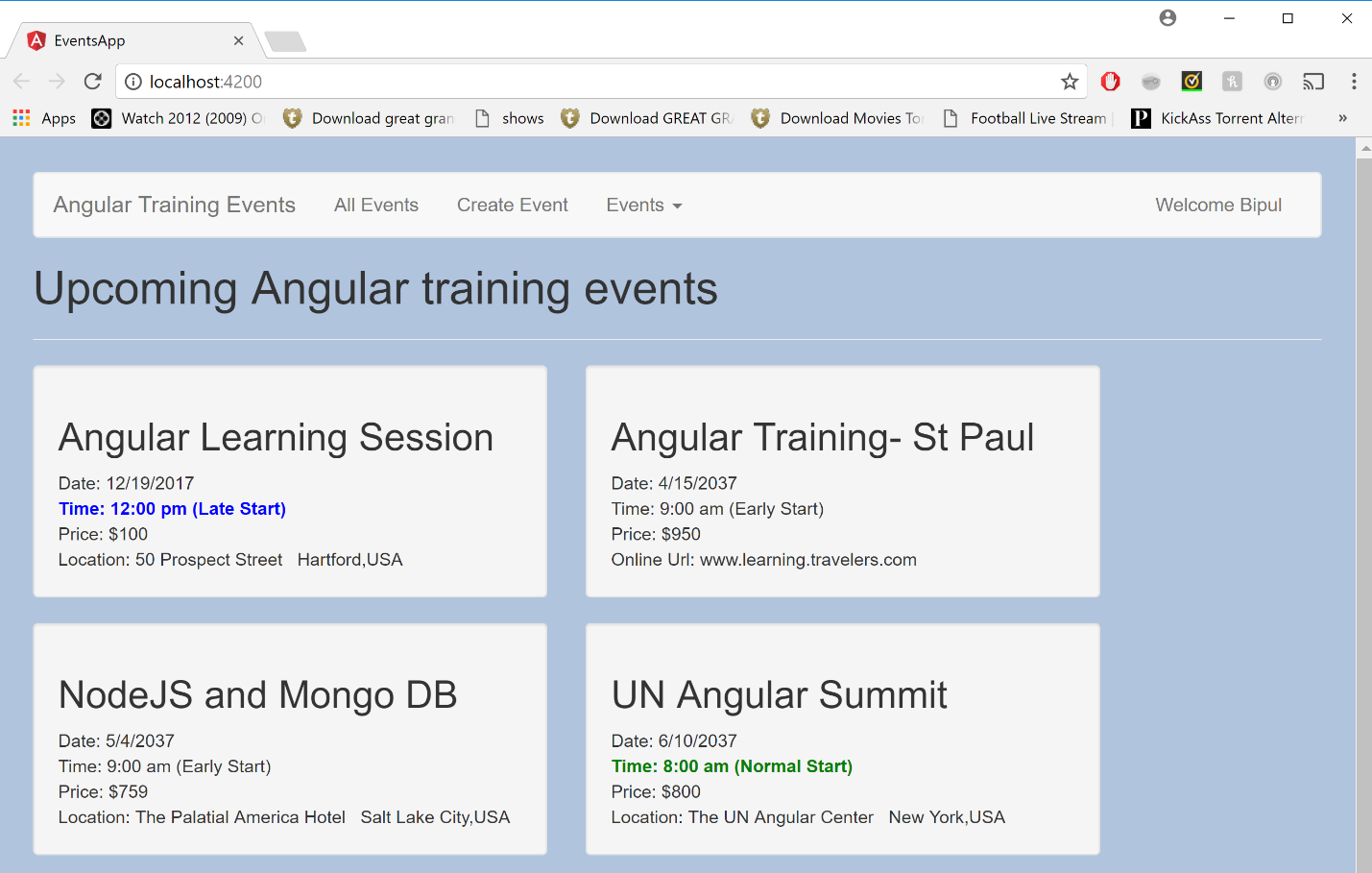
ngOnInit(){

//called after the constructor and called after the first ngOnChanges()

}

}

1. No rerun the application



**Routing and Navigating Pages**

Before we can demonstrate routing, we need to have multiple pages in our application. Currently we are displaying a thumbnail to display all events. Let’s create an event detail page in our application now. And we will show how to navigate to that page when we click on one of the events on this page.

1. Let’s add another component called event-details component by using ng g component event-details
2. Add below code to the event-details.component.html:

<div class="container">

<img [src]="event?.imageUrl" [alt]="event?.name" class="event-image">

<div class="row">

<div class="col-md-11">

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

</div>

</div>

<div class="row">

<div class="col-md-6">

<div><strong>Date:</strong> {{event?.date}}</div>

<div><strong>Time:</strong> {{event?.time}}</div>

<div><strong>Price:</strong> ${{event?.price}}</div>

</div>

<div class="col-md-6">

<address>

<strong>Address:</strong><br />

{{event?.location?.address}}<br />

{{event?.location?.city}}, {{event?.location?.country}}

</address>

</div>

</div>

</div>

1. Update the CSS for the detail component for some styling.

.container{

padding-left: 20px;

padding-right:20px

}

.event-image{

height:100px;

}

1. Update the event.service.ts file by adding one method to getEvent() which will accept the event ID . Eventually all of this will be ajax call. Add below code which will return

getEvent(id:number){

return EVENTS.find(event => event.id === id);

}

1. For the time being let’s call this method in event-details.component.ts by hardcoding the value to 1. Update below code in the file.

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

import { EventService } from '../service/event.service';

@Component({

selector: 'app-event-details',

templateUrl: './event-details.component.html',

styleUrls: ['./event-details.component.css']

})

export class EventDetailsComponent implements OnInit {

event:any;

constructor(private eventservice: EventService) {

}

ngOnInit() {

this.event = this.eventservice.getEvent(1);

}

}

1. Now we added another component, let’s see how to add our first route. Now let’s observe our events-app component. We see that we have events-list and nav-bar component. So there is no way to change that. So let’s replace the events-list we will replace it by angular’s router-outlet component and somehow we need to tell angular when the user requests a particular url then play its corresponding component here. We do that by defining routes.
2. Update the app.component.html with below code:

<app-event-nav></app-event-nav>

<router-outlet></router-outlet>

1. Add routes.ts at the app folder level and add below code. An ( right now we need two routes ) :

import { Routes} from '@angular/router'

import { EventDetailsComponent } from "./event-details/event-details.component";

import { EventsListComponent } from './events-list/events-list.component';

export const appRoutes : Routes = [

{ path: 'events', component: EventsListComponent },

{ path: 'events/:id', component: EventDetailsComponent }, // events/1 or //events/foo

{ path: '', redirectTo: '/events', pathMatch: 'full' } // default path where pathMatch can accept prefix or full

]

1. An important point to remember here for the default route, we need to set that in the index.html with base tag. Lets add base tag in index.html

<base href="/">

1. Add routes in the app.module.ts. Update appModule.ts with below code.

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { RouterModule } from '@angular/router';

import { AppComponent } from './app.component';

import { EventsListComponent } from './events-list/events-list.component';

import { EventThumbnailComponent } from './event-thumbnail/event-thumbnail.component';

import { EventNavComponent } from './event-nav/event-nav.component';

import { EventService } from './service/event.service';

import { EventDetailsComponent } from './event-details/event-details.component';

import { appRoutes } from './routes';

@NgModule({

declarations: [

AppComponent,

EventsListComponent,

EventThumbnailComponent,

EventNavComponent,

EventDetailsComponent

],

imports: [

BrowserModule,

RouterModule.forRoot(appRoutes)

],

providers: [EventService],

bootstrap: [AppComponent]

})

export class AppModule { }

1. Now we don’t need the selector in events-list.component.ts so let’s comment the application.

//selector: 'app-events-list',

1. Now serve the application and query for both routes.
2. Now let’s update the event detail component to get the ids dynamically and legts make the below changes to the event-details.component.ts. After making the change serve the application and pass different numbers as ID for the routes

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

import { EventService } from '../service/event.service';

import { ActivatedRoute } from '@angular/router'

@Component({

//selector: 'app-event-details',

templateUrl: './event-details.component.html',

styleUrls: ['./event-details.component.css']

})

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']); // + is used to type convert to integer

}

}

1. Now till now we have seen that when we change URL various components are loaded. But how to trigger it from application actions. For which let’s go to thumbnail components and make some changes. Let’s add below code in event.thumbnail.component.html. Adding routerLink property with below path with id should match the routes.ts route definition. Now serve the application and click on the tumbnails.

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

1. We can navigate through the application by clicking on the thumbnail div. Now let’s update event-nav.component.html at the anchor tag with “All Events” Value and update the routerLink as below. Now when you serve the application click on All Events to come to default to events route.

<a [routerLink]="['/events']" >All Events</a>

**Navigate from Code**

We just saw how to link to our routes using HTML and now let’s see how to do the same through code.

Let’s create a very simple component. This will become the pages that will be used to create new events, but for now let’s create a simple component.

1. Let’s call it create-event.component.ts. Use command ng g component create-event
2. Let’s add some html that we will use in future of this exercise to the create-event.component.html

<h1> New Event </h1>

<hr>

<div class='col-md-6'>

<h3>[create event 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>

1. Let’s update the routes.ts for this new route. Positioning of this route should be done at the top as Angular will not understand that matching path as it will think its trying to match id new to the route path

{ path: 'events/new', component: CreateEventComponent },

1. Let’s also add the routerLink to the event- nav-component.html as well. Let’s serve the application see if this new route is working or not.

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

1. Now the cancel button is not doing anything. Now what we want is when user clicks cancel he should be navigated back to default route. For that let’s add event handler on Cancel button click on HTML

<button type='button' class ='btn btn-default' (click)='cancel()' >Cancel</button>

1. Add the Cancel function on create-event.component.ts. Now to navigate from code we need to inject angular router service. Now let’s serve the application and see the cancel click on the application.

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

import { Router } from '@angular/router';

@Component({

selector: 'app-create-event',

templateUrl: './create-event.component.html',

styleUrls: ['./create-event.component.css']

})

export class CreateEventComponent implements OnInit {

constructor( private router : Router) { }

ngOnInit() {

}

cancel(){

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

}

}

## Guarding against route navigation

Sometimes we may want to prevent the user from navigating to a particular page or stop them from leaving a page.

If we check intellisense within routes.ts, we can see the different properties for the routes. 2 among these are used to create route guards – **canActivate** and **canDeactivate.**

### canActivate

Command 🡪 **ng g component errors/Error404**

In **error404.component.html**, add the following html code.

<h1 class="errorMessage">Not Found !!</h1>>

In **error404.component.css**, add the following style.

.errorMessage{

margin-top: 150px;

font-size: 170px;

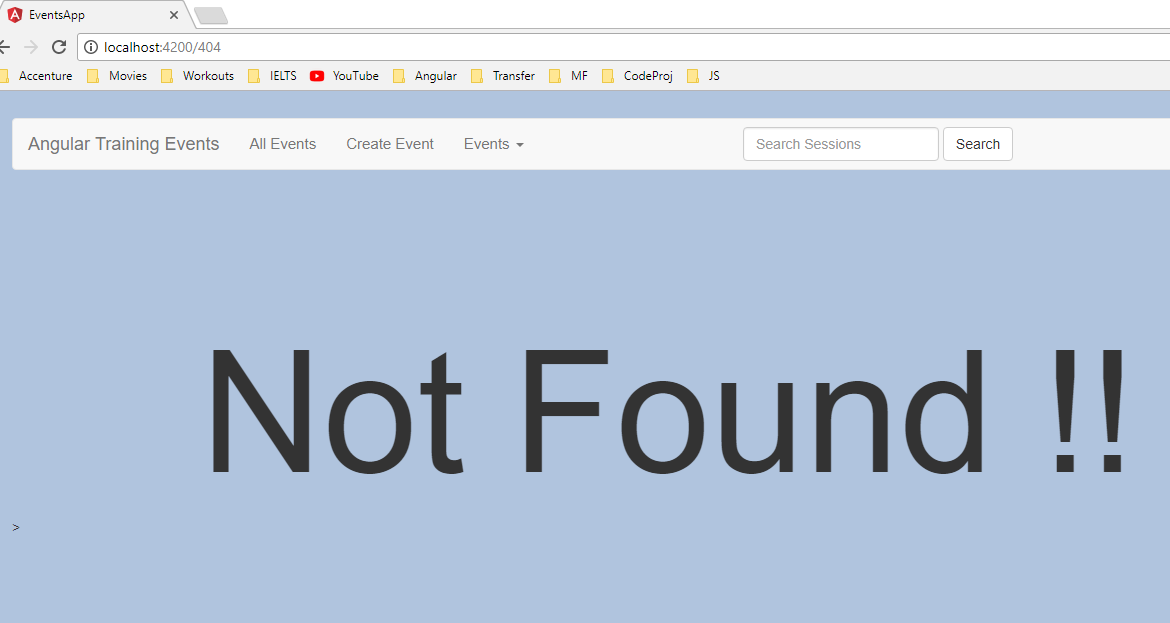
text-align: center

}

Let us add a route to this page in **routes.ts**.

{ path: '404', component: Error404Component }

Now if we navigate to <http://localhost:4200/404>, we should see the “**Not found**” error page come up.



We will add now add a route activator service for event details. Create a file **event-route-activator.ts** in event-details folder. Add the content as below.

import { CanActivate, Router, ActivatedRouteSnapshot } from '@angular/router';

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

import { EventService } from '../service/event.service';

@Injectable()

export class EventRouteActivator implements CanActivate {

constructor(private eventService:EventService, private router:Router) { }

canActivate(route: ActivatedRouteSnapshot) {

const eventExists = !!this.eventService.getEvent(+route.params['id']);

if(!eventExists){

this.router.navigate(['/404']);

}

return eventExists;

}

}

With this, we have created a route guard. We have to now add this as a provider in app module.

providers: [EventService, EventRouteActivator],

Now that we have a route guard, we will attach it to the route which we need to guard.

In routes.ts, update the **events/:id** route as below.

{ path: 'events/:id', component: EventDetailsComponent, canActivate: [EventRouteActivator] },

## canDeactivate

Update the **events/new** route as below.

{ path: 'events/new', component: CreateEventComponent, canDeactivate: ['canDeactivateCreateEvent'] },

isDirty:boolean = true;

function checkDirtyState(component: CreateEventComponent){

if(component.isDirty)

return window.confirm('You have not saved the data. Do you really want to cancel?')

return true;

}

## Resolve

In **event.service.ts,**

getEvents(){

let subject = new Subject();

setTimeout(()=> {subject.next(EVENTS); subject.complete();},2000);

//return EVENTS;

return subject;

}

You would have to import as below

import { Subject } from 'rxjs/Subject';

In **events-list.component.ts**,

Change type of event to

events:any;

And change the event fetching line to

this.eventservice.getEvents().subscribe(events => {this.events = events});

Add a new service file

Command 🡪 **ng g service service/events-list-resolver-service**

Modify the content as below.

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

import { Resolve } from '@angular/router';

import { EventService } from './event.service';

import 'rxjs/add/operator/map';

@Injectable()

export class EventsListResolverServiceService implements Resolve<any> {

constructor(private eventService:EventService) { }

resolve(){

return this.eventService.getEvents().map(events => events)

}

}

Register the resolver as a provider in app module.

providers: [

EventService,

EventRouteActivator,

{

provide: 'canDeactivateCreateEvent',

useValue: checkDirtyState

},

EventsListResolverService

],

Now we add this as a resolver to the route that we need resolve. In **routes.ts**

{ path: 'events', component: EventsListComponent, resolve: {events: EventsListResolverService} },

Now in **events-list.component.ts**

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

import { EventService } from '../service/event.service';

import { ActivatedRoute } from '@angular/router';

@Component({

//selector: 'app-events-list',

templateUrl: './events-list.component.html',

styleUrls: ['./events-list.component.css']

})

export class EventsListComponent implements OnInit {

events:any;

constructor(private eventservice: EventService, private route:ActivatedRoute) {

}

// the private varible is equal to below code

// eventservice

// constructor() {

// this.eventservice = eventservice

// }

ngOnInit() {

//this.eventservice.getEvents().subscribe(events => {this.events = events});

this.events = this.route.snapshot.data['events'];

}

}

## Styling Active links

We add routerLinkActive attribute in event-nav.component.css

Add the style in **event-nav.component.css**

li > a.active {color: #F97924}

In **event-nav.component.html**

<a [routerLink]="['/events']" routerLinkActive="active"

[routerLinkActiveOptions]="{exact:true}">All Events</a>

# Lazy loading Modules

Command 🡪 **ng g component user/profile**

Add the below html in **profile.component.html**

<h1>Edit your profile</h1>

<hr>

<div>

<h3>[Edit Profile form goes here]</h3>

<br />

<br />

<button type='submit' class ='btn btn-primary'>Save</button>

<button type='button' class ='btn btn-default'>Cancel</button>

</div>

Create a new route file under user folder called **user.route.ts**

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

export const userRoutes = [

{path: 'profile', component: ProfileComponent}

]

Create a new module under user folder **user.module.ts**

import { NgModule } from '@angular/core'

import { RouterModule } from '@angular/router';

import { CommonModule } from '@angular/common';

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

import { userRoutes } from './user.route';

@NgModule({

declarations: [

ProfileComponent

],

imports: [

CommonModule,

RouterModule.forChild(userRoutes)

],

providers: []

})

export class UserModule {}

In main **route.ts**, add the following path

{ path: 'user', loadChildren: 'app/user/user.module#UserModule' }

Add a ling to the profile section in navbar in **event-nav.component.html**

<a [routerLink]="['/user/profile']">Welcome Bipul</a>

# Collecting data with forms and validations

## Using models for type safety

Create a new folder called model and create a new file **session.model.ts**

export interface ISession{

id: number,

name: string,

presenter: string,

duration: number,

level: string,

abstract: string,

voters: string[]

}

Create another file called **event.model.ts**

import { ISession } from "./session.model";

export interface IEvent{

id: number,

name: string,

date: Date,

time: string,

price: number,

imageUrl: string,

location?: {

address: string,

city: string,

country: string

},

onlineUrl?: string,

sessions: ISession[]

}

In event.service.ts, change the type of EVENTS object to IEvent[]

Change the dates as below in EVENT objects to be of type date.

date: new Date('12/19/2017')

In **event.service.ts**, we can now specify return types for getEvent and getEvents methods.

getEvents(): Subject<IEvent[]> {

let subject = new Subject<IEvent[]>();

setTimeout(()=> {subject.next(EVENTS); subject.complete();},100);

//return EVENTS;

return subject;

}

getEvent(id:number) : IEvent{

return EVENTS.find(event => event.id === id);

}

Now in event-thumbnail.component.ts, events-list.component.ts, event-details.component.ts

# Creating forms

## Creating a simple template based form

Let us now create a login component in the user folder.

Command – **ng g component user/login**

In **login.component.html,** replace the content as below.

<h1>Login</h1>

<hr>

<div class="col-md-4">

<form autocomplete="off">

<div class="form-group">

<label for="userName">User Name:</label>

<input id="userName" type="text" class="form-control"

placeholder="user name..." />

</div>

<div class="form-group">

<label for="password">Password:</label>

<input id="password" type="password" class="form-control"

placeholder="Password..." />

</div>

<button type='submit' class ='btn btn-primary'>Login</button>

<button type='button' class ='btn btn-default'>Cancel</button>

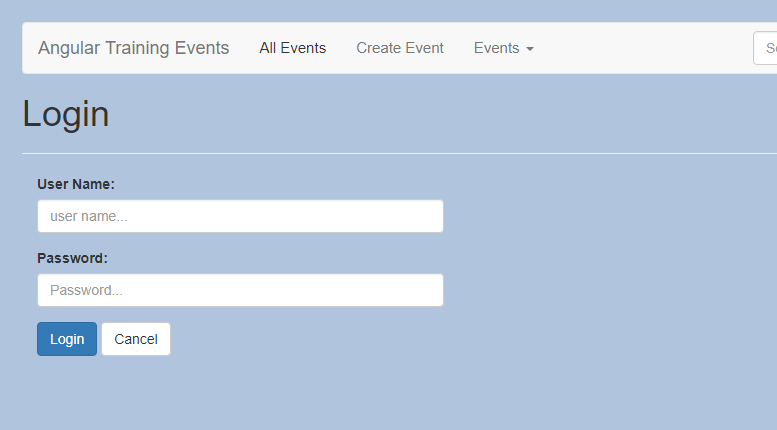
</form>

</div>

Let us add new route in **user.route.ts**

{path: 'login', component: LoginComponent}

Now if we navigate to **/user/login**, we will see the login form.



Now let us look at some forms based directives. Import FormsModule in user.module.ts file as our form is defined within user folder.

import { FormsModule } from '@angular/forms'

and in imports section

imports: [

CommonModule,

RouterModule.forChild(userRoutes),

FormsModule

],

Now let us wire up our form to capture the data entered by the user in the form.

Add **ngModel** to the input fields.

<input id="userName" type="text" class="form-control"

(ngModel)="userName" placeholder="user name..." />

and

<input id="password" type="password" class="form-control"

(ngModel)="password" placeholder="Password..." />

Sometimes we might see the syntax as below.

[(ngModel)]="userName"

() 🡪 used to bind from html to component direction

[] 🡪used to bind from component to html direction

2 way bindings are typically used in places where the existing dat is edited so that the dat edited is transferred to and fro between components and html.

ngModel also requires us to specify a **name** attribute.

In **login.component.html**, decorate form element with 2 attributes.

<form #loginForm="ngForm" (ngSubmit)="login(loginForm.value)" autocomplete="off">

Now we will add a login method to **LoginComponent**.

login(formValues){

console.log(formValues);

}

Let us first create a user model. Add a file **user.model.ts** in the user folder.

export interface IUser{

id: number,

firstName: string,

lastName: string,

userName: string

}

Now let us create a service which does the authentication for us. Create **auth.service.ts** file inside user folder.

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

import { IUser } from './user.model';

@Injectable()

export class AuthService {

currentUser: IUser

loginUser(userName: string, password: string){

this.currentUser = {

id: 1,

firstName: "Devaiah",

lastName: "Ramesh",

userName: userName

};

}

isAuthenticated(){

return !!this.currentUser;

}

}

Now we have to register the service. But instead of user module, we will register in app module. Because in Angular, providers are shared across the components. So if we register in app module, it would be available in user module as well.

But same is not true for imports and declarations.

Add AuthService as a provider under NGModules in app module and import it.

providers: [

EventService,

EventRouteActivator,

{

provide: 'canDeactivateCreateEvent',

useValue: checkDirtyState

},

EventsListResolverService,

AuthService

],

Update login component as below.

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

import { AuthService } from '../auth.service';

@Component({

selector: 'app-login',

templateUrl: './login.component.html',

styleUrls: ['./login.component.css']

})

export class LoginComponent implements OnInit {

constructor(private authService:AuthService) { }

ngOnInit() {

}

login(formValues){

//console.log(formValues);

this.authService.loginUser(formValues.userName, formValues.password);

}

}

Now in nav bar, we will display the welcome message when user is authenticated and until then we will display login link.

Update **event-nav.component.html**, as below.

<li>

<a \*ngIf="!auth.isAuthenticated()" [routerLink]="['/user/login']">Login</a>

<a \*ngIf="auth.isAuthenticated()" [routerLink]="['/user/profile']">Welcome Devaiah</a>

</li>

Inject AuthService into **event-nav.component.ts**

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

import { AuthService } from '../user/auth.service';

@Component({

selector: 'app-event-nav',

templateUrl: './event-nav.component.html',

styleUrls: ['./event-nav.component.css']

})

export class EventNavComponent implements OnInit {

constructor(private auth:AuthService) { }

ngOnInit() {

}

}

To simulate login, we shall route the user to events page on click of login button.

Update login method in **login.component.ts** as below.

login(formValues){

//console.log(formValues);

this.authService.loginUser(formValues.userName, formValues.password);

this.router.navigate(['events']);

}

We need to import and inject Router into the constructor.

import { Router } from '@angular/router/src/router';

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

Also, add a cancel method as below in the same file.

cancel(){

this.router.navigate(['events']);

}

Update cancel input markup **login.component.html** as below.

<button type='button' (click)="cancel" class ='btn btn-default'>Cancel</button>

Now we are able to click login and cancel to navigate back to events page.

## Validating Template Based forms

As of now we can see that we are able to login without even entering a user name and password. This is because we have mocked the behavior of authService and have no credential based logic.

### Required validator for input fields

Addding ‘required’ field for the 2 inputs will cause the HTML5 required field validator to fire.

In order to disable the HTML5 validations, we can add **novalidate** attribute to the form tag as below.

<form #loginForm="ngForm" (ngSubmit)="login(loginForm.value)"

autocomplete="off" novalidate>

## Keep login button disabled before something is entered

## We can keep the login form disabled until the user enters the credentials. In order to do that, we update the login button markup as below.

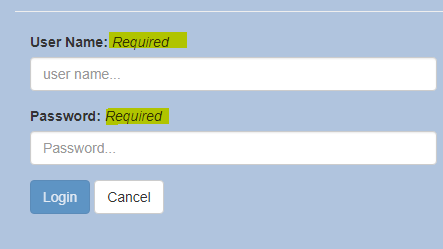
<button type="submit"[disabled]="loginForm.invalid" class="btn btn-primary">Login</button>

Now we can see that login is enabled only when user enters username and the password.

In order to notify the user that the credentials are required, we will now add text to display to the user.

Add the below markup next to username and password labels.

<em>Required</em>



Message will be shown as in the screenshot.

But now this message is always available. In order to make it appear only when the fields are invalid, we update the markup for username and password labels as below.

<em \*ngIf="loginForm.controls.userName?.invalid">Required</em>

<em \*ngIf="loginForm.controls.password?.invalid">Required</em>

Add the following style for emphasis tag in **login.component.css**

em {

float: right;

color: red;

padding-left: 10px

}

But now, even at the page load, we would see this error. To make this error appear only when the user has touched the controls, we would include another condition as below.

<em \*ngIf="loginForm.controls.userName?.invalid &&

loginForm.controls.userName?.touched">Required</em>

<em \*ngIf="loginForm.controls.password?.invalid &&

loginForm.controls.password?.touched">Required</em>

Now we see the message not showing up on page load. When we click on a control and tab out of it, we see that the error will show up.

Wrap the login button with a span as below.

<span (mouseenter)="mouseoverlogin=true" (mouseleave)="mouseoverlogin=false">

<button type="submit"[disabled]="loginForm.invalid" class="btn btn-primary">Login</button>

</span>

Update the condition to display **required** message as below.

<em \*ngIf="loginForm.controls.userName?.invalid &&

(loginForm.controls.userName?.touched || mouseoverlogin)">Required</em>

<em \*ngIf="loginForm.controls.password?.invalid &&

(loginForm.controls.password?.touched || mouseoverlogin)">Required</em>

# Reactive forms

The login form that we saw was a template driven form – meaning we have developed the form and associated validations entirely using html.

The other kind of form that we can develop is reactive forms or model driven forms.

Basically this means, we would define the fields and the validations in the component and wire them to the fields in the html template.

This has comparatively more code than the previous method but offer some advantages.

* We can dynamically build validation in code based on other decision factors.
* It makes all validation logic unit testable

To demonstrate this we will create a profile form.

Update **profile.component.html** with below markup.

<div>

<h1>Edit Your Profile </h1>

<hr>

<div class="col-md-4">

<form autocomplete="off" novalidate>

<div class="form-group">

<label for="firstName">First Name:</label>

<input id="firstName" type="text" class="form-control" placeholder="First Name..." />

</div>

<div class="form-group">

<label for="lastName">Last Name:</label>

<input id="lastName" type="text" class="form-control" placeholder="Last Name..." />

</div>

<button type="submit" class="btn btn-primary">Save</button>

<button type="button" class="btn btn-default">Cancel</button>

</form>

</div>

</div>

Now instead of wiring up html to use angular constructs, we will go to component and start configuring the form.

Html form has 2 inputs – first name and last name. We will create form controls for each of those as below.

In **profile.component.ts**, we will implement the following.

1. Add 2 form controls

let firstName = new FormControl();

let lastName = new FormControl();

1. Add these controls to a form. In order to add these, we create a from object of the type FormGroup and set properties for each of the controls.

this.profileForm = new FormGroup({

firstName: firstName,

lastName: lastName

});

Updated **profile.component.ts** is as below.

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

import { FormControl, FormGroup } from '@angular/forms;

@Component({

selector: 'app-profile',

templateUrl: './profile.component.html',

styleUrls: ['./profile.component.css']

})

export class ProfileComponent implements OnInit {

profileForm: FormGroup;

constructor() { }

ngOnInit() {

let firstName = new FormControl();

let lastName = new FormControl();

this.profileForm = new FormGroup({

firstName: firstName,

lastName: lastName

});

}

}

Now we have to tell angular which html elements these correspond to.

1. Bind the form element in the HTML to **profileForm** object.

<form [formGroup]="profileForm" autocomplete="off" novalidate>

1. Now we need to specify the input elements the properties from proileForm they need to bind to.

<input id="firstName" formControlName="firstName" type="text" class="form-control" placeholder="First Name..." />

<input id="lastName" type="text" formControlName="lastName" class="form-control" placeholder="Last Name..." />

For template driven forms we imported **FormsModule** from @angular/forms.

For reactive forms, in **user.module.ts**, we need to import **ReactiveFormsModule** and add it to the imports list.

import { FormsModule, ReactiveFormsModule } from '@angular/forms'

imports: [

CommonModule,

RouterModule.forChild(userRoutes),

FormsModule,

ReactiveFormsModule

],

Now if we go to the profile page, we see the form. But ideally the values should have been prepopulated as I have logged in and I am trying to edit my profile.

In order to achieve this, we need to pass the form controls the value to be displayed on loading. We will use authService to provide the names to be displayed.

Import AuthService and inject it in **profile.component.ts**

import { AuthService } from '../auth.service';

constructor(private auth:AuthService) { }

Andd update the form controls as below.

let firstName = new FormControl(this.auth.currentUser.firstName);

let lastName = new FormControl(this.auth.currentUser.lastName);

Now we load the profile page post login, we will see the values prepopulated.

We shall wire-up **Save** and **Cancel** button now. Add a click hander for cancel and add the cancel method.

<button type="button" class="btn btn-default" (click)="cancel()">Cancel</button>

cancel(){

this.router.navigate(['events']);

}

Import the Router and inject it.

To wireup Save button, we need to add the handler to the form element.

<form [formGroup]="profileForm" (ngSubmit)="saveProfile(profileForm.value)" autocomplete="off" novalidate>

Now we try updating the profile, we can see the welcome message updated with a new name that we enter.

In order to mke Login form redirect to events page after login, add the following update.

saveProfile(formValues){

this.auth.updateCurrentUser(formValues.firstName, formValues.lastName);

this.router.navigate(['events']);

}

## Validating Reactive forms

Validators are passed as second field in the form controls. Make the following updates to the form controls in **profile.component.ts**

let firstName = new FormControl(this.auth.currentUser.firstName, Validators.required);

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

We can now additional logic before saving the profile based on validator states, something like below code.

saveProfile(formValues){

if(this.profileForm.valid){

this.auth.updateCurrentUser(formValues.firstName, formValues.lastName);

this.router.navigate(['events']);

}

}

We can easily add unit tests to check these logics. For larger forms this would provide great benefits in testability.

Add some styles to profile.component.css.

em {

float: right;

color: red;

padding-left: 10px

}

.error input {background-color: coral}

We then add the condition for styling for div containing firstname and lastname as below.

<div class="form-group"

[ngClass]="{'error': profileForm.controls.firstName.invalid &&

profileForm.controls.firstName.touched}">

<div class="form-group"

[ngClass]="{'error': profileForm.controls.lastName.invalid &&

profileForm.controls.lastName.touched}">

This will now highlight the input controls with the style specified when no value is entered.

Now we can add an error message to display. The logic to display error message will be identical to the logic that we coded in ng class. So, let us move the logic to the component so that we can reuse.

validateLastName(){

return this.lastName.valid || this.lastName.untouched

}

validateFirstName(){

return this.firstName.valid || this.firstName.untouched

}

**profileForm.controls.lastName** can be replaced with **this.lastName** and make necessary adjustments.

## Implementing multiple Validators for a field

In the above example, we see a single validator implemented. What if we want to implement multiple validators?

The way to achieve that is to pass an array of validator to the form control.

For e.g., let us add a pattern name validator to the first name, which would enforce the first name to start with a letter.

Step 1: The first step is to turn the parameter to be an array.

this.firstName = new FormControl(this.auth.currentUser.firstName, [Validators.required]);

Step 2: We will now pass the desired validator as an item in the array.

this.firstName = new FormControl(this.auth.currentUser.firstName,

[Validators.required, Validators.pattern('[a-zA-Z].\*')]);

Let us now test the change.

We will see that, our existing required validator still works. If we now enter a number as the first character in the first name, we will see that it will indicate that the value is invalid, but our validation error message still says ‘**Required’.** We need to fix this.

Let us add another error message in the template, **profile.component.html** as below.

<em \*ngIf="!validateFirstName()">Must start with a letter</em>

But now, 2 errors will start showing up when the first name control becomes invalid. In order to solve this, we will add another condition as below.

<em \*ngIf="!validateFirstName() &&

profileForm.controls.firstName.errors.required">Required</em>

<em \*ngIf="!validateFirstName() &&

profileForm.controls.firstName.errors.pattern">Must start with a letter</em>

Let us now see how this change is reflected in the site.

To explore different validators available, we could check the angular documentation here.

[**https://angular.io/api/forms/Validators**](https://angular.io/api/forms/Validators)

## Create Event form

<h1>New Event</h1>

<hr>

<div class="col-md-6">

<form #newEventForm="ngForm" (ngSubmit)="saveEvent(newEventForm.value)" autocomplete="off" novalidate>

<div class="form-group" [ngClass]="{'error': newEventForm.controls.name?.invalid && newEventForm.controls.name?.touched}">

<label for="eventName">Event Name:</label>

<em \*ngIf="newEventForm.controls.name?.invalid && (newEventForm.controls.name?.touched)">Required</em>

<input (ngModel)="name" name="name" required id="name" type="text" class="form-control" placeholder="Name of your event..." />

</div>

<div class="form-group" [ngClass]="{'error': newEventForm.controls.date?.invalid && newEventForm.controls.date?.touched}">

<label for="eventDate">Event Date:</label>

<em \*ngIf="newEventForm.controls.date?.invalid && (newEventForm.controls.date?.touched)">Required</em>

<input (ngModel)="date" name="date" required id="eventDate" type="text" class="form-control" placeholder="format (mm/dd/yyyy)..." />

</div>

<div class="form-group" [ngClass]="{'error': newEventForm.controls.time?.invalid && newEventForm.controls.time?.touched}">

<label for="eventTime">Event Time:</label>

<em \*ngIf="newEventForm.controls.time?.invalid && (newEventForm.controls.time?.touched)">Required</em>

<input (ngModel)="time" name="time" required id="eventTime" type="text" class="form-control" placeholder="start and end time..." />

</div>

<div class="form-group" [ngClass]="{'error': newEventForm.controls.price?.invalid && newEventForm.controls.price?.touched}">

<label for="eventPrice">Event Price:</label>

<em \*ngIf="newEventForm.controls.price?.invalid && (newEventForm.controls.price?.touched)">Required</em>

<input (ngModel)="price" name="price" required id="eventPrice" type="text" type="number" class="form-control" placeholder="event price..." />

</div>

<div class="form-group">

<label for="address">Event Location:</label>

<input (ngModel)="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)="city" name="city" id="city" type="text" class="form-control" placeholder="City..." />

</div>

<div class="col-md-6" >

<input (ngModel)="country" name="country" id="country" type="text" class="form-control" placeholder="Country..." />

</div>

</div>

<div class="form-group">

<label for="onlineUrl">Online Url:</label>

<input (ngModel)="onlineUrl" name="onlineUrl" id="onlineUrl" type="text" class="form-control" placeholder="Online Url..." />

</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?.errors.required">Required</em>

<em \*ngIf="newEventForm.controls.imageUrl?.invalid && newEventForm.controls.imageUrl?.touched && newEventForm.controls.imageUrl?.errors.pattern">Must be a png or jpg url</em>

<input (ngModel)="imageUrl" name="imageUrl" required pattern=".\*\/.\*.(png|jpg)" id="imageUrl" type="text" class="form-control" placeholder="url of image..." />

<img />

</div>

<button type="submit" class="btn btn-primary">Save</button>

<button type="button" [disabled]="newEventForm.invalid" class="btn btn-default" (click)="cancel()">Cancel</button>

</form>

</div>

Copy the below styles in **create-event.component.css**

em {

float: right;

color: red;

padding-left: 10px

}

.error input {background-color: coral}

.error ::-webkit-input-placeholder {color: #999;}

.error ::-moz-placeholder {color: #999;}

.error :-moz-placeholder {color: #999;}

.error ::-ms-input-placeholder {color: #999;}

Import **FormsModule** and **ReactiveFormsModule** in **app.module.ts**

imports: [

BrowserModule,

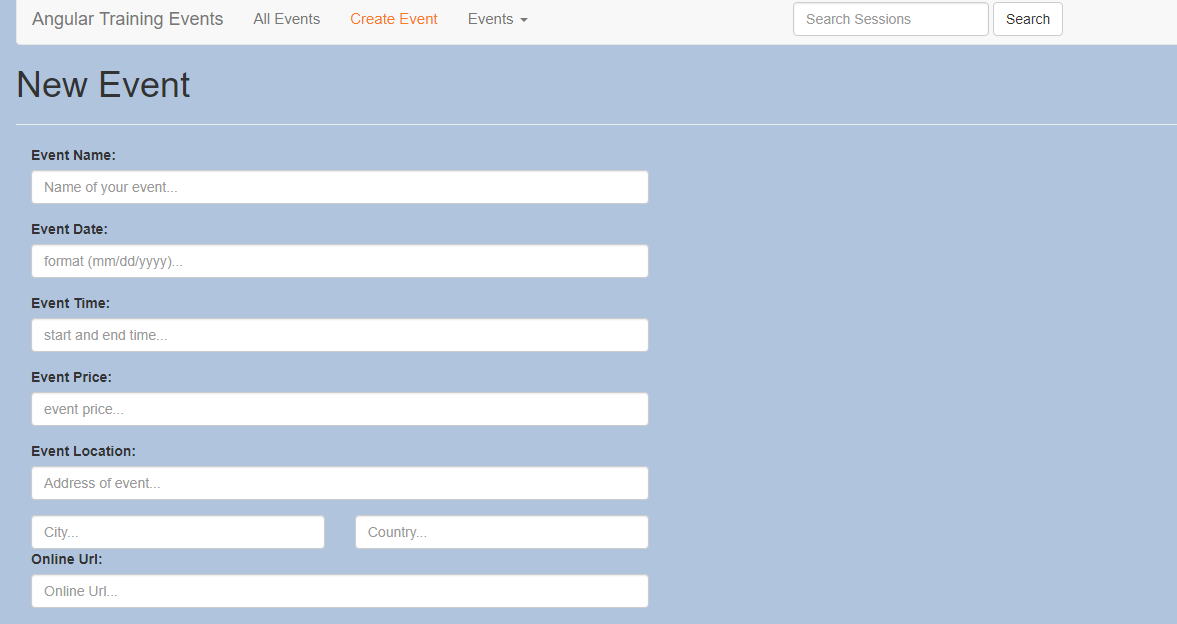
RouterModule.forRoot(appRoutes),

FormsModule,

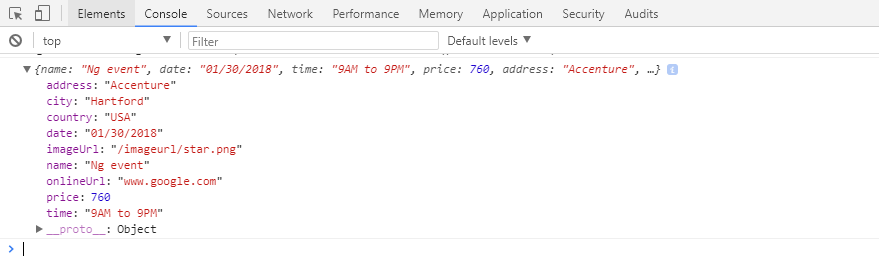
ReactiveFormsModule

],

The form is now displayed as below.



Now we will open the form, and see the formvalues displayed in browser console.



If we see the above object, we see that the object model does not exactly correspond to the eevent model that we have defined. The address, city and county is nested under the location object in the event model, but here they are defined at the root.

export interface IEvent{

id: number,

name: string,

date: Date,

time: string,

price: number,

imageUrl: string,

location?: {

address: string,

city: string,

country: string

},

onlineUrl?: string,

sessions: ISession[]

}

We can write custom code in our save method to map it to the right object format. But angular provides an easy way of converting this into the shape that we want on form submit.

Angular lets us do that for both kind of forms - template based and Reactive forms.

For template based forms, we wrap the desired controls in a ngModelGroup as shown below.

<div ngModelGroup="location">

<div class="form-group">

<label for="address">Event Location:</label>

<input (ngModel)="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)="city" name="city" id="city" type="text" class="form-control" placeholder="City..." />

</div>

<div class="col-md-6" >

<input (ngModel)="country" name="country" id="country" type="text" class="form-control" placeholder="Country..." />

</div>

</div>

</div>

Now we check the object submitted, we will see the 3 fields wrapped in a location object.



Let us now update the file **create-event.component.ts** with an update to saveEvent method. Also, Import Event Service and inject into the constructor.

saveEvent(formValues){

// console.log(formValues);

this.eventService.saveEvent(formValues);

this.isDirty = false;

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

}

Create a method as below in **event.service.ts**

saveEvent(event){

event.id = 999;

event.session = [];

EVENTS.push(event);

}

There might be an error that we could encounter for circular reference. In this case, we just update the reference to Event Service as below.

import { EventService } from '../service/event.service'

## Editing data with 2 way binding

All this time, we were looking at forms with creating data with one way data binding. Now we will see a scenario where we edit the existing data which we achieve using 2 way binding.

For now, add a temporary event object in **create-event.component.ts** as below in ngOnit() method.

ngOnInit() {

this.event = {

name: 'NG spectacular',

date: '01/25/2018',

time: '10AM',

price: 500,

location:{

address:'Hartford Main Street',

city: 'Hartford',

country: 'USA'

},

onlineUrl: 'http://ngspectaculr.com',

imageUrl: 'http://ngspectaculr.com/logo.png'

};

}

‘

In **create-event.component.html**, update instances of binding to bind from event object.

For e.g., **(ngModel)=name** will become **(ngModel)= event.name**

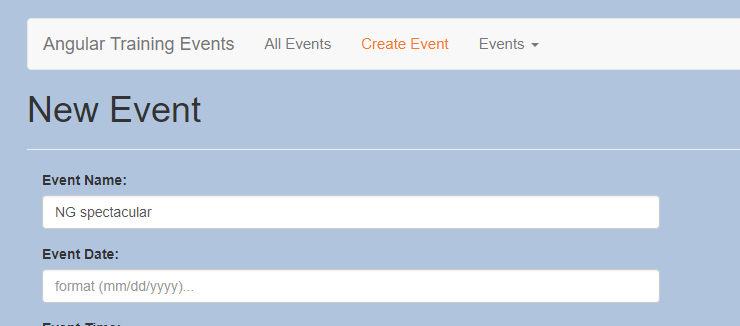
**(ngModel)=country** will become **(ngModel)= event.location.country**

Now when we load the create Event page, we see that no data is pre-loaded in the form. This is because we are currently using one-way binding.

In order to be able to pull out the data into the form on loading, we would use 2 way binding using “banana in a box” syntax as below.

**(ngModel)= event.name** will become [**(ngModel)]= event.name**

We will now see that on page load, the data for event name field will be prepopulated with the help of 2 way binding that we specified for the event name input control.



Create session

Right now, event details page does not list sessions and additional details. Let us now add a page to create session against an event.

Let us now add a Reactive form to create a Add session page. Add a component called **create-session**

Command 🡪 **ng g component create-session**

Create a barrel for the component (This is optional)

Update **app.module.ts** to import and declare the component.

Update **create-session.component.html** file with the below content.

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

<h3>Create Session</h3>

</div>

<div class="col-md-6">

<form autocomplete="off">

<div class="form-group">

<label for="sessionName">Session Name:</label>

<input id="sessionName" type="text" class="form-control" placeholder="session name..." />

</div>

<div class="form-group">

<label for="eventDate">Presenter:</label>

<input id="presenter" type="text" class="form-control" placeholder="presenter..." />

</div>

<div class="form-group">

<label for="duration">Duration:</label>

<select 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">

<label for="level">Level:</label>

<select 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">

<label for="abstract">Abstract:</label>

<textarea id="abstract" rows=3 class="form-control" placeholder="abstract..."></textarea>

</div>

<button type="submit" class="btn btn-primary">Save</button>

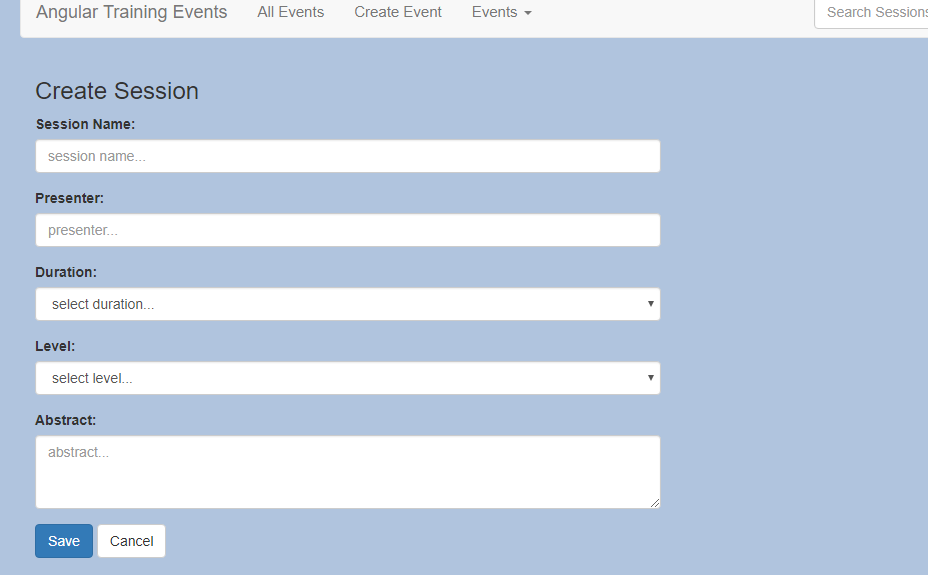
<button type="button" class="btn btn-default">Cancel</button>

</form>

</div>

We will now add a route for it. In **route.ts**, add the below rout.

We should be able to load the page now.



Update the **create-session.component.ts** file as below.

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

import { FormControl, FormGroup, Validators } from '@angular/forms';

@Component({

selector: 'app-create-session',

templateUrl: './create-session.component.html',

styleUrls: ['./create-session.component.css']

})

export class CreateSessionComponent implements OnInit {

newSessionForm: FormGroup;

name: FormControl;

presenter: FormControl;

duration: FormControl;

level: FormControl;

abstract: FormControl;

constructor() { }

ngOnInit() {

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(200)]);

this.newSessionForm = new FormGroup({

name: this.name,

presenter: this.presenter,

duration: this.duration,

level: this.level,

abstract: this.abstract,

});

}

}

Here, we have declared the form control properties to be public which makes them easily accessible in the html form.

Update the form component in the html as below.

<form [formGroup]="newSessionForm" (ngSubmit)="saveSession(newSessionForm.value)" autocomplete="off">

Bind each input controls with their corresponding properties like below.

<input id="sessionName" formControlName="name" type="text"

class="form-control" placeholder="session name..." />

Now run the form and check the console on submitting the form.

Now in **create-session.component.ts**, create a session property that is of type ISession and defie the object using form values.

Update saveSession method as follows.

saveSession(formvalues){

//console.log(formvalues);

let session:ISession = {

id: undefined,

name: formvalues.name,

presenter: formvalues.presenter,

duration: +formvalues.duration,

level: formvalues.level,

abstract: formvalues.abstract,

voters:[]

};

console.log(formvalues);

}

Add required validation messages for each input controls as below.

<em \*ngIf="name.invalid && name.dirty">Required</em>

If we compre this with the validation that we defined in profile form, we notice that we are directly referring to the control with the property names as we have made them public. In profile form, we have to access the control like

profileForm.controls.firstName.errors.pattern

Add the validation message for each of the controls.

Abstract field would need 2 messages to display 2 validations.

<em \*ngIf="abstract.invalid && abstract.dirty && abstract?.errors.required">Required</em>

<em \*ngIf="abstract.invalid && abstract.dirty && abstract?.errors.maxLength">Cannot exceed 200 characters</em>

Add error classes to each of the control group as shown below.

<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 id="sessionName" formControlName="name" type="text"

Add the below styles in **create-session.component.css**

em {

float: right;

color: red;

padding-left: 10px

}

.error input, .error select, .error textarea {background-color: indianred}

.error ::-webkit-input-placeholder {color: #999;}

.error ::-moz-placeholder {color: #999;}

.error :-moz-placeholder {color: #999;}

.error ::-ms-input-placeholder {color: #999;}

# Creating Custom Validators

Sometimes, there might be a need to create our own business defined validations which might not be harder to create using existing validators.

Doing this in Reactive forms is much easier than in template based forms.

Validator are basically functions, which returns null when the control is valid and an error object if the control is invalid.

In **create-session.component.ts**, we create a validator as below.

private restrictedWords(control: FormControl) :{[key: string] : any}

{

return control.value.includes('foo')

? {'restrictedWords': 'foo'} : null

}

Now ad this validator to the control we wish to validate.

this.abstract = new FormControl('',

[Validators.required, Validators.maxLength(200), this.restrictedWords]);

Let us also add a validation message in the html.

<em \*ngIf="abstract.invalid && abstract.dirty &&

abstract?.errors.restrictedWords">Restriced words entered : {{abstract.errors.restrictedWords}}</em>

We should be able to see the validator working now.

# Communication between components

Currently event details page does not display sessions associated with the event.

Let us create a session list component. In **event-details.component.html** Let us bind the data with the component as below.

<app-session-list [sessions] = "event?.sessions"></app-session-list>

In **session-list.component.ts**, create a property as below.

@Input() sessions: ISession[];

Let us see how to pass data from child component to parent component. This is a typical scenario, when an event occurs in a child component. We do this using Output parameters.

Add a new div as below next to <hr> line.

<hr>

<div class="row">

<div class="col-md-2">

<h3 style="margin:0">Sessions</h3>

</div>

<div class="col-md-2">

<a (click)="addSession()">Add Session</a>

</div>

</div>

Add the following style

a{cursor: pointer}

Add the below method in **event-details.component.ts**

addSession(){

this.addMode = true;

}

Update the following line in **event-details.component.html** to include ngIf

<app-session-list \*ngIf="!addMode" [sessions] = "event?.sessions"></app-session-list>

<app-create-session \*ngIf="addMode" (saveNewSession)="saveNewSession($event)"></app-create-session>

We need to now save the session which we add against the event.

Add an output property in **create-session.component.ts** to save a new session added.

@Output() saveNewSession = new EventEmitter();

In SaveSession method, add the below line to the end.

this.saveNewSession.emit(session);

Add saveSession method in **event-details.component.ts**

saveNewSession(session:ISession){

const nextId = Math.max.apply(null, this.event.sessions.map(s => s.id));

session.id = nextId + 1;

this.event.sessions.push(session);

this.eventservice.updateEvent(this.event);

this.addMode = false;

}

Add the below update event method in event service as below.

updateEvent(event){

let index = EVENTS.findIndex(x=> x.id == event.id);

EVENTS[index] = event;

}

Now we can add a new session and see the session being added.

Now to wire up cancel button in the add session page, we need another output parameter in **create-session.component.ts**

@Output() cancelAddSession = new EventEmitter();

Wire up cancel button in the html as below.

<button type="button" class="btn btn-default" (click)="cancel()">Cancel</button>

Add a cancel method as below in **create-session.component.ts**

cancel(){

this.cancelAddSession.emit();

}

Update the line in **event-details.component.html** as below.

<app-create-session \*ngIf="addMode" (saveNewSession)="saveNewSession($event)"

(cancelAddSession)="cancelAddSession()"></app-create-session>

Add the below method in in **event-details.component.ts**

cancelAddSession(){

this.addMode = false;

}

Now we can see the cancel button working.

Creating reusable components using content projection.

Navigate to session-list.component.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>

<collapsible-well [title]="session.name">

Instead of div class =well replace with collapsible -well

Remove the ‘h4’, <h4>{{session.name}}</h4>

<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>

Navigate to ‘Common’ folder , create a folder ‘collapsible-well.component.ts’

Import component

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

import { NgIf } from '@angular/common/src/directives';

@Component({

selector:'collapsible-well',

template:`

<div (click)="toggleContent()" class="well pointable">

<h4 class="well-title">{{title}}</h4>

<ng-content \*ngIf="visible"></ng-content>

</div>

`

})

export class CollapsibleWellComponent {

@Input() title:string;

visible:boolean=true;

toggleContent(){

this.visible = !this.visible;

}

}

Click event to toggle the content of the body.

Add the component to the module, and also under the declarations

CollapsibleWellComponent

import { CollapsibleWellComponent } from './common/collapsible-well.component';

other example can be modal dialog where we have ok/cancel button , this is common across application.

Multiple slot projection:

Next we will see if which session is extremely popular and which has most votes/rating.

To do this, assume we need to have an indicator/icon next to session which displays which session has most votes. Since this is pertaining to a particular session we need to add some business logic . we cannot have this logic under collapsible well because it has just the’h4’ with string .

Instead it is better to have another item like

<title> & other item like <body > inside the collapsible well

<collapsible-well>

<title>

{{session.name}}

<i \*ngIf="session.voters.length >3 " class="glyphicon glyphicon-fire" style="color:red"></i>

</title>

<body>

<h6>{{session.presenter}}</h6>

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

<span>Level: {{session.level}}</span>

<p>{{session.abstract}}</p>

</body>

</collapsible-well>

Navigate to our collapsible well component

We wanted whatever under <h4> should come under title and body has the rest of the content.

`

<div (click)="toggleContent()" class="well pointable">

<h4 class="well-title">{{title}}</h4>

<h4>

<ng-content></ng-content>

</h4>

<ng-content \*ngIf="visible"></ng-content>

</div>

`

Now use a class selector like id or . or # to the element like select=”.”

Now go to the collapsible well and change title to ‘div’ and body to ‘div as well add a class=”title” to h4 and class=”body” to other div.

<collapsible-well>

<div class="title">

{{session.name}}

<i \*ngIf="session.voters.length >3 " class="glyphicon glyphicon-fire" style="color:red"></i>

</div>

<div class="body">

<h6>{{session.presenter}}</h6>

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

<span>Level: {{session.level}}</span>

<p>{{session.abstract}}</p>

</div>

</collapsible-well>

`

<div (click)="toggleContent()" class="well pointable">

<h4>

<ng-content select=".title"></ng-content>

</h4>

<ng-content \*ngIf="visible" select=".body"></ng-content>

</div>

`

Sometimes using the class names in the same component can conflicts with css styles present in the project. So instead of class give an attribute

<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>

Now navigate to our component and remove input ,it is not an input now,

export class CollapsibleWellComponent {

visible:boolean=true;

toggleContent(){

this.visible = !this.visible;

}

}

`

<div (click)="toggleContent()" class="well pointable">

<h4>

<ng-content select="[well-title]"></ng-content>

</h4>

<ng-content \*ngIf="visible" select="[well-body]"></ng-content>

</div>

`

Final class:

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

import { NgIf } from '@angular/common/src/directives';

@Component({

selector:'collapsible-well',

template:`

<div (click)="toggleContent()" class="well pointable">

<h4>

<ng-content select="[well-title]"></ng-content>

</h4>

<ng-content \*ngIf="visible" select="[well-body]"></ng-content>

</div>

`

})

export class CollapsibleWellComponent {

visible:boolean=true;

toggleContent(){

this.visible = !this.visible;

}

}

**session-list.component.ts**

<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>