lIf you have a static website, then you have all the data that you need in order to build your website. And it'll be created once and your website is ready to go. But many websites these days are driven by data from the backend. Your web application will fetch data from a backend server and then use the data to dynamically layout content on your website, or even update content on your website. So this requires the data to flow from the backend through your web application to the DOM. And any user interaction on the DOM should be conveyed back to your web application. So this requires the communication between the DOM and your component in the angular framework for example. Let's look at how this happens within the angular framework and with a little bit more detail.

What we have learned so far is how a component is prepared.

So we saw that the component has two major parts, the typescript

file where the component's architecture is defined, and the properties.

And the methods of and then you have the template that

defines how the content is laid out, and added into the DOM.

So the template is the one that is going to supplied by a component to

occupy a part of your webpage.

So the template forms the communication medium to your DOM.

And this template requires data to be gotten from your component in order

to render parts of the content on your template and

correspondingly within the DOM of your web page.

Now, if you didn't have the luxury of a framework like Angular,

you have to deal with the entire details about how

the data flows from the application site to the DOM.

And how the user interaction events that the caused in

the DOM will be sent back to your application.

Fortunately, with frame works like Angular,

they support all these behaviors inherently.

Now we have already seen one use of the flow of data from a components

property to the template in the previous exercises.

Where we used the double curly braces to supply information that

is from a variable in your component's TypeScript code, to your template.

We'll look at other ways, and

then also in the exercise that follows, we will make use of this,

to modify the angular application that we have been developing so far.

What we have been talking, so far, is data binding,

a mechanism for coordinating the flow of information between the template and

the component, or between the DOM and the component.

In the component level, it's some kind of a property or

a method that needs to be either supplied to the template or invoked from the DOM.

Or on the template side, it may be some information

that is required by the template to be rendered into the DOM or

even from the DOM captured and sent back to your component.

So all this requires data flowing from the component to

the template or events that are gathered from the DOM or

to be sent back to your component for it to act upon those events.

Now this can be easily handled in angular framework using four different

kinds of flow of information which we will summarize in the next slide.

Coming back to the interaction between component and the template,

we have already seen the first kind of interaction between the component and

the template where the value of a property was flowing into the template.

So we saw the use of the dish name,

the dish description, the dish comment details and so

on being supplied in order to render information in your template.

So that is where we were using the double braces to enclose the fact that

a value from your component is being used in your template.

Another kind of flow of information which you will see Is where

you specify some kind of a property associated with a tag in square brackets.

And then assign it to be the value from one of the properties in your component.

This kind of approach we will see in the exercise that follows

right after this lecture.

So here we would specify something in square brackets.

So watch out for

an example of that in the exercise where I will discuss more about this.

Similarly, if you have an event generated near DOM,

that event may result in the call to a handler or

a method In your component that will take care of handling this event.

So the methods that are going to be invoked by

the events from your DOM are referred to as handlers.

So, the handle is nothing but, as I said a method, which might also

be passed in some parameter inside the handler methods there.

6:02

Later on, when we look at forms, we will see a two-way data binding.

So all the three that we have seen so far, the flow of the value from

the component to the template or the property being assigned to a value or

the event being assigned to a handler to handle the event.

All these are what we refer to as one way data bindings because

6:31

they flow only in one direction.

You can also have bidirectional flow where you might also see it

in the syntax what you would specify with square brackets.

And then enclosed inside that with standard parenthesis.

An inside there you will specifies something like an ngModel.

You will see the use of this when it forms later on in the next module.

So this is where you will assign a property to that.

So in this case the information flow is bidirectional.

So any change in your DOM will be reflected back into your property and

the component.

Any change in the component's property will be reflected back to the DOM.

Note the specific syntax that we use for

specifying the ngModel in this example here.

We use a square bracket and inside that we use the standard parenthesis in there.

7:22

Let me now summarize what we have just learned about

the data binding in the previous slide in to this table.

So in this table, we are showing the different ways that data binding is used.

One-way data binding from the data source to the view target so

from the component to the DOM,

we are using a double brace expression inside there.

This is what we refer to as Interpolation as an example of which,

would be the dish name, enclosed inside the double braces as we have seen,

used in the exercises, in the previous module.

Then we have, the second kind where you enclose the target

inside a square bracket and then assign it to an expression on the right side.

These expressions could be JavaScript expressions with some limitations.

The examples that we will normally see would be

using a property from one of the classes there.

So, this is what we referred as a property attribute, and

example of which you will see in the exercise that's follows this lecture.

Where you would see the use of a dish, enclosed inside a square bracket,

being equated to a selectedDish, which is a variable declared inside a component.

This can also be expressed using a bind-target,

a way of expressing the same thing.

So both these whether you enclose a target in square brackets or bind-target.

So, for example, you can see bind-dish.

So whichever way you express this, both refer to the same thing.

So the Property Attribute being specified or the Class Style being specified.

Now the one-way flow of information from the view target to

the data source is usually expressed with the target enclosed

inside parentheses here or can also be expressed as on-target.

And at the right hand side would be a statement.

A statement could be some kind of a JavaScript expression typically

would be in the of invocation of a method inside a component.

Well you will see an example in the exercise that follows where

you would have click inside the parenthesis and assigned to onSelect.

Which is a method described inside a component

class there with a parameter dish being supplied to this method there.

The two-way data binding as we saw would be with squire and parentheses,

squire brackets and parentheses which is equated to an expression.

We will see two-way data binding as a set with forms a little bit later where

you will use something like an ngModel, enclosed inside the square brackets and

parentheses, equated to something like dish.name.

Which connects the form element to a property of

a JavaScript object, in our component.

10:46

You can also express this as bindon-target.

Which is another way of saying the same thing.

Now, this two-way data binding that we do is sometimes

jocularly referred to as a banana in a box.

So if you look at the square brackets with the parenthesis inside it,

it looks like a banana in a box.

So, you might see this being used In some of the documentation or

in some of the blogs that you might read on the Internet.

11:15

Expanding further on the binding targets that we have seen,

the binding targets are the properties that are declared on the left side

of the binding declaration of that data binding declaration.

Typically, enclosed inside square brackets or inside parenthesis or both.

So the right side of the binding expression is their binding sources,

so which are typically like the properties of a JavaScript object, or

a variable, or an expression that we define on the right hand side.

If you define target properties associated with the selective of a component.

That is one way of either passing in information into a component,

or sending back information from one component to another component.

So this facilitates communication between components.

So you would see the use of a decorator like @Input associated with

a way of sending information from one component to another component.

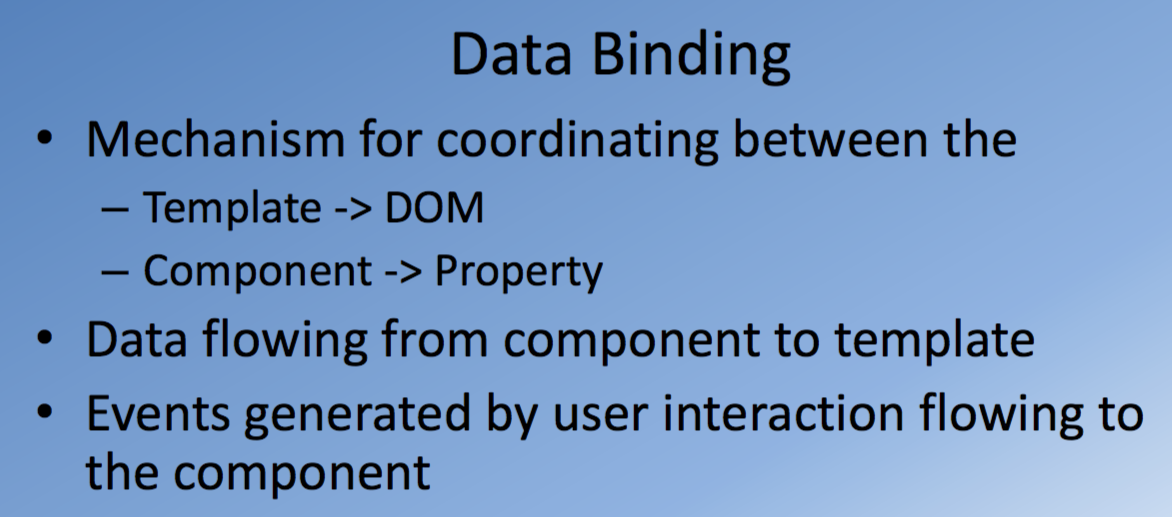
We will see an example of this usage in the exercise

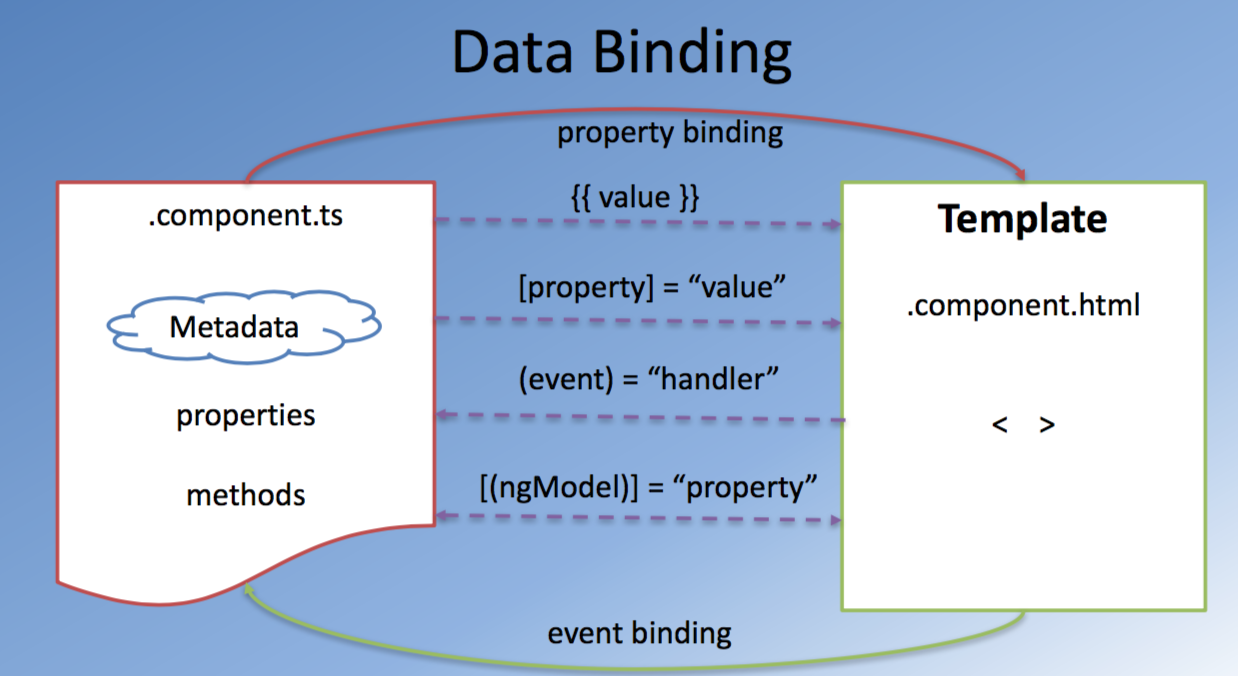
that follows this lecture here.

Similarly, you can use the Output decorator to specify an event

from one component being passed back to another component.

So both these usages are effectively used for communicating between components.





Exercise (Instructions): Data Binding

Objectives and Outcomes

In this exercise you will use your knowledge of data binding to update the Angular application to enable you to select any dish from the menu and show its details. At the end of this exercise you will be able to:

* Leverage data binding for communication among components
* Design a component to receive input from another component.

Refactoring the Code

* First, create a new class named Comment in a file named comment.ts in the shared folder and include the following in it:

export class Comment {

rating: number;

comment: string;

author: string;

date: string;

}

* Then update the dish class to allow a dish to have an array of comments as follows:

import { Comment } from './comment';

export class Dish {

name: string;

image: string;

category: string;

label: string;

price: string;

description: string;

comments: Comment[];

}

* Then create a new file named dishes.ts in the shared folder to now export the JavaScript object array of dishes:

import { Dish } from './dish';

export const DISHES: Dish[] = [

{

name: 'Uthappizza',

image: '/assets/images/uthappizza.png',

category: 'mains',

label: 'Hot',

price: '4.99',

description: 'A unique combination of Indian Uthappam (pancake) and Italian pizza, topped with Cerignola olives, ripe vine cherry tomatoes, Vidalia onion, Guntur chillies and Buffalo Paneer.',

comments: [

{

rating: 5,

comment: "Imagine all the eatables, living in conFusion!",

author: "John Lemon",

date: "2012-10-16T17:57:28.556094Z"

},

{

rating: 4,

comment: "Sends anyone to heaven, I wish I could get my mother-in-law to eat it!",

author: "Paul McVites",

date: "2014-09-05T17:57:28.556094Z"

},

{

rating: 3,

comment: "Eat it, just eat it!",

author: "Michael Jaikishan",

date: "2015-02-13T17:57:28.556094Z"

},

{

rating: 4,

comment: "Ultimate, Reaching for the stars!",

author: "Ringo Starry",

date: "2013-12-02T17:57:28.556094Z"

},

{

rating: 2,

comment: "It's your birthday, we're gonna party!",

author: "25 Cent",

date: "2011-12-02T17:57:28.556094Z"

}

]

},

{

name: 'Zucchipakoda',

image: '/assets/images/zucchipakoda.png',

category: 'appetizer',

label: '',

price: '1.99',

description: 'Deep fried Zucchini coated with mildly spiced Chickpea flour batter accompanied with a sweet-tangy tamarind sauce',

comments: [

{

rating: 5,

comment: "Imagine all the eatables, living in conFusion!",

author: "John Lemon",

date: "2012-10-16T17:57:28.556094Z"

},

{

rating: 4,

comment: "Sends anyone to heaven, I wish I could get my mother-in-law to eat it!",

author: "Paul McVites",

date: "2014-09-05T17:57:28.556094Z"

},

{

rating: 3,

comment: "Eat it, just eat it!",

author: "Michael Jaikishan",

date: "2015-02-13T17:57:28.556094Z"

},

{

rating: 4,

comment: "Ultimate, Reaching for the stars!",

author: "Ringo Starry",

date: "2013-12-02T17:57:28.556094Z"

},

{

rating: 2,

comment: "It's your birthday, we're gonna party!",

author: "25 Cent",

date: "2011-12-02T17:57:28.556094Z"

}

]

},

{

name: 'Vadonut',

image: '/assets/images/vadonut.png',

category: 'appetizer',

label: 'New',

price: '1.99',

description: 'A quintessential ConFusion experience, is it a vada or is it a donut?',

comments: [

{

rating: 5,

comment: "Imagine all the eatables, living in conFusion!",

author: "John Lemon",

date: "2012-10-16T17:57:28.556094Z"

},

{

rating: 4,

comment: "Sends anyone to heaven, I wish I could get my mother-in-law to eat it!",

author: "Paul McVites",

date: "2014-09-05T17:57:28.556094Z"

},

{

rating: 3,

comment: "Eat it, just eat it!",

author: "Michael Jaikishan",

date: "2015-02-13T17:57:28.556094Z"

},

{

rating: 4,

comment: "Ultimate, Reaching for the stars!",

author: "Ringo Starry",

date: "2013-12-02T17:57:28.556094Z"

},

{

rating: 2,

comment: "It's your birthday, we're gonna party!",

author: "25 Cent",

date: "2011-12-02T17:57:28.556094Z"

}

]

},

{

name: 'ElaiCheese Cake',

image: '/assets/images/elaicheesecake.png',

category: 'dessert',

label: '',

price: '2.99',

description: 'A delectable, semi-sweet New York Style Cheese Cake, with Graham cracker crust and spiced with Indian cardamoms',

comments: [

{

rating: 5,

comment: "Imagine all the eatables, living in conFusion!",

author: "John Lemon",

date: "2012-10-16T17:57:28.556094Z"

},

{

rating: 4,

comment: "Sends anyone to heaven, I wish I could get my mother-in-law to eat it!",

author: "Paul McVites",

date: "2014-09-05T17:57:28.556094Z"

},

{

rating: 3,

comment: "Eat it, just eat it!",

author: "Michael Jaikishan",

date: "2015-02-13T17:57:28.556094Z"

},

{

rating: 4,

comment: "Ultimate, Reaching for the stars!",

author: "Ringo Starry",

date: "2013-12-02T17:57:28.556094Z"

},

{

rating: 2,

comment: "It's your birthday, we're gonna party!",

author: "25 Cent",

date: "2011-12-02T17:57:28.556094Z"

}

]

}

];

Updating the Menu Component

* Open menu.component.ts file and update its content, first by deleting the dishes constant and then make the following changes:

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

import { Dish } from '../shared/dish';

import { DISHES } from '../shared/dishes';

. . .

export class MenuComponent implements OnInit {

dishes = DISHES;

selectedDish: Dish;

. . .

onSelect(dish: Dish) {

this.selectedDish = dish;

}

}

* Then update the menu.component.html file as follows:

. . .

<md-grid-tile \*ngFor="let dish of dishes" (click) = "onSelect(dish)">

. . .

<app-dishdetail [dish] = "selectedDish"></app-dishdetail>

. . .

Updating Dish Detail Component

* Open dishdetail.component.ts and update its contents as follows:

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

import { Dish } from '../shared/dish';

. . .

export class DishdetailComponent implements OnInit {

@Input()

dish: Dish;

. . .

* Save the changes and do a Git commit with the message "Data Binding".

<https://angular.io/guide/architecture#!%23data-binding>

<https://angular.io/guide/template-syntax>

Angular Service Basics: Objectives and Outcomes

In this lesson you will learn about the basics of Angular services and how they interact with Angular components. You will also learn the basics of Model-View-Controller (MVC) and Model-View-ViewModel (MVVM) software engineering paradigms. You will also be introduced to the basics of Dependency Injection (DI). At the end of this course you will be able to:

* Add an Angular service and inject into a module and make use of it in the components.
* Understand the basics of MVC and MVVM, and DI.

[MUSIC] In the web development you often hear

people talking about the MVC framework and the MVVM framework and so on. What exactly are these frameworks? How are they useful in

doing web development? Let's talk about that briefly next. In the software engineering world, you often hear people talking

about design patterns. What exactly they mean is to stop

reinventing the wheel every single time. A design pattern is a well-documented

solution to a recurring problem. Very often, you see yourself

repeatedly solving similar problems. If we have a well specified documentation

of how to solve these problems, why keep reinventing

the wheel every single time? So that is where the software engineering's design

pattern concept originates. Sometimes you also see people referring

to this as an architecture pattern. So to summarize, software design

patterns in particular are a reusable solution to commonly occurring

problems that are solved in software. Now in this context, you often hear

people talking about the gang of four. This was a group of four authors

that wrote this seminal book called Design Patterns: Elements of

Reusable Object-Oriented Software. In this book, they identified

a large set of commonly used design patterns in software engineering. This was one of the first well-documented

exploration of design patterns, and hence, became the gold standard for

anybody working in software engineering,

especially concerned about design packets. This software engineering

pattern enables us to isolate domain logic from the user interface. So you're basically separating

the user's view of the information from the actual logic and how

the information stored and manipulated. Now this separation of concerns concept

you're going to be hearing over and over again in this context. The separation of concerns is what

facilitates independent development of each of these three parts

of our application and also enables testing and

maintenance of these different parts. Now we can divide our entire

application into three parts, the view that is primarily concerned

with presenting information to the user, the model that stores the domain state and

the domain logic and also provides the way of manipulating this state from

the rest of the application and the controller that mediates

between the view and the model. We'll talk about each of these three

parts in a bit more detail next. In the MVC framework,

the model manages the behavior and data of the application domain. And the model responds to requests for

information about its current state. So typically when the view wants to

render, or the view wants to update itself, it might query the model

in order to obtain information so that it can be rendered

appropriately to the user. The model also will respond to

requests for change of its state. This is usually done through the control. In an event-driven system, the model also can be

configured to notify observers. So viewers can register themselves

as observers for the model and so when the model is updated the views

will be automatically triggered to update themselves based on

the change to that model state. The view itself is concern with presenting

the information to the users in a user interface element in such a way that

it facilitates both the presentation of information to the user and also enables

the user to interact with the application. So the view may represent one

representation of the model state. So, from single model,

you can easily derive multiple ways presenting this information to the user,

depending upon for example, depending upon the viewport size. So a small size viewport like

on a mobile application, the information will be presented

in a different way as opposed to a larger view port that is

facilitated on a desktop computer. So in an MVC framework all the display

of information has a one to one correspondence with the model state. The third piece of puzzle in the MCV

framework is the controller. The job of the controller is to

receive information from the view. So any user interaction that is

performed will be captured and then passed onto the controller in order

to act on these user interactions. And it is the job of the controller then

to initiate a change of the state of the model, if it is required

in this particular situation. So the controller will appropriately cause

the change of the state of the model. So to summarize,

the controller can accept input from the user in terms of the user

interactions that have taken place, and then it will instruct

the model to change the state. Simultaneously, the controller may also cause the view to change the way the

information is being shown in the view. So that is the reason why in

this picture you have two arrows going from the controller, one towards

the model and the other towards the view. Sometimes you hear people talking about

the model view view-model approach. The model view view-model approach is,

in some sense, a derivative of the model

view controller approach. Sometimes you also hear people

referring to it as the model view view-binder approach. In here, you have the model that

represents the business logic and the data for your application. From the model, you derive a view-model,

which encapsulates that part of the information that is

required for rendering a specific view. So the view-model is the abstraction

of the view that exposes the public properties and

the various commands that are available. So this provides

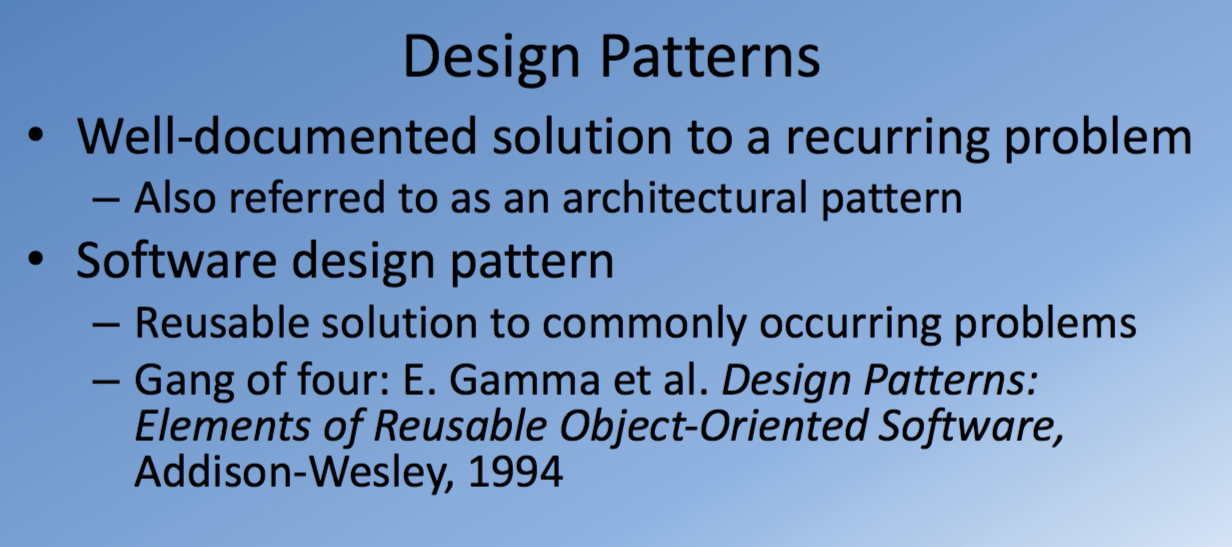
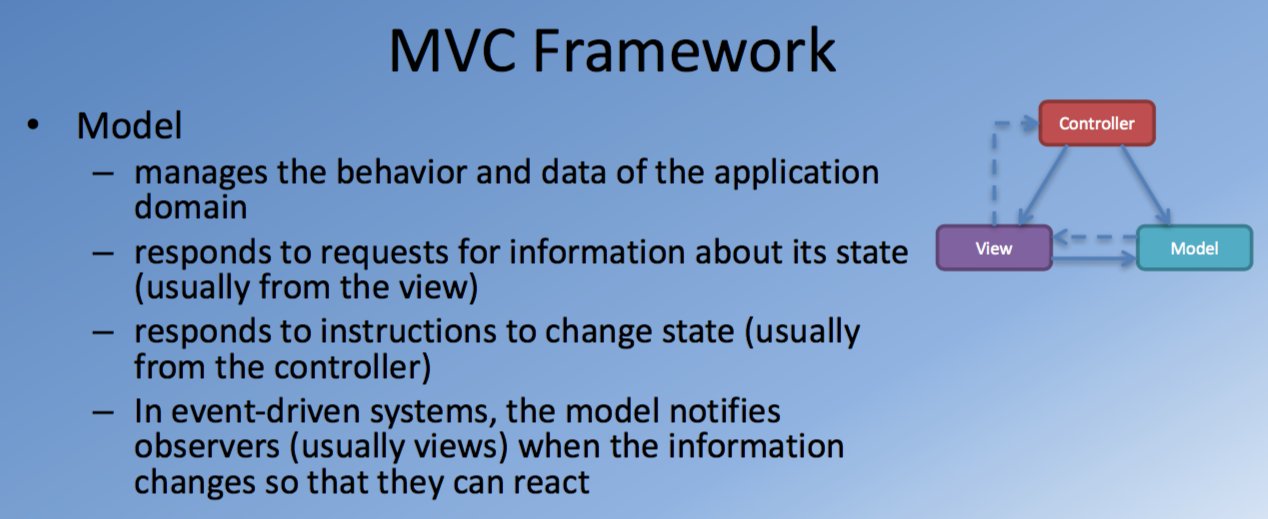
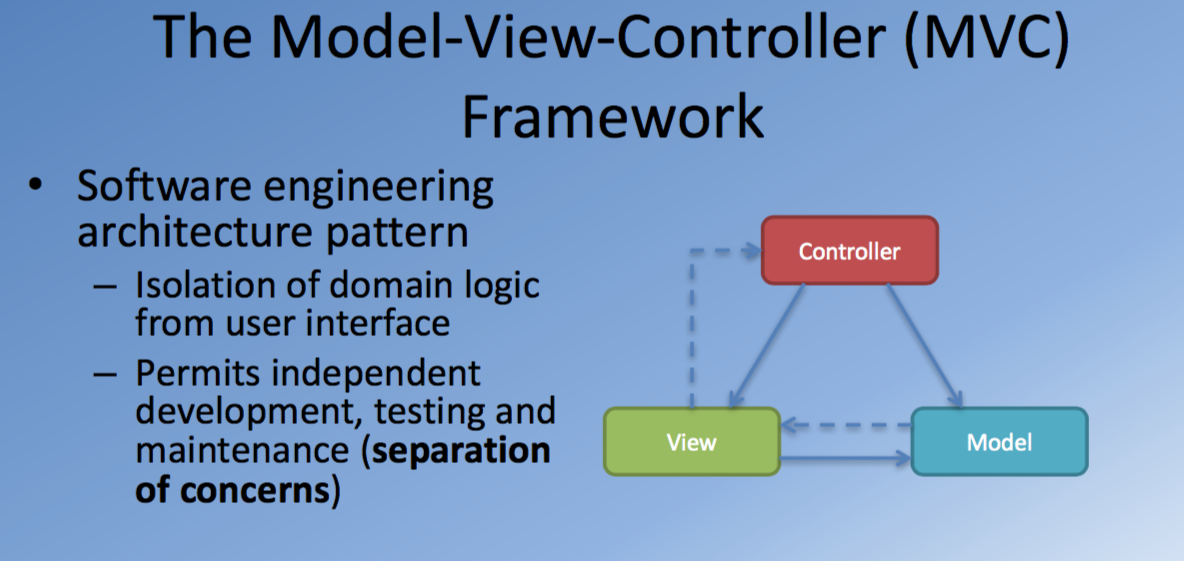
a declarative data binding. In some sense, the way the component and the templates in angular are implemented,

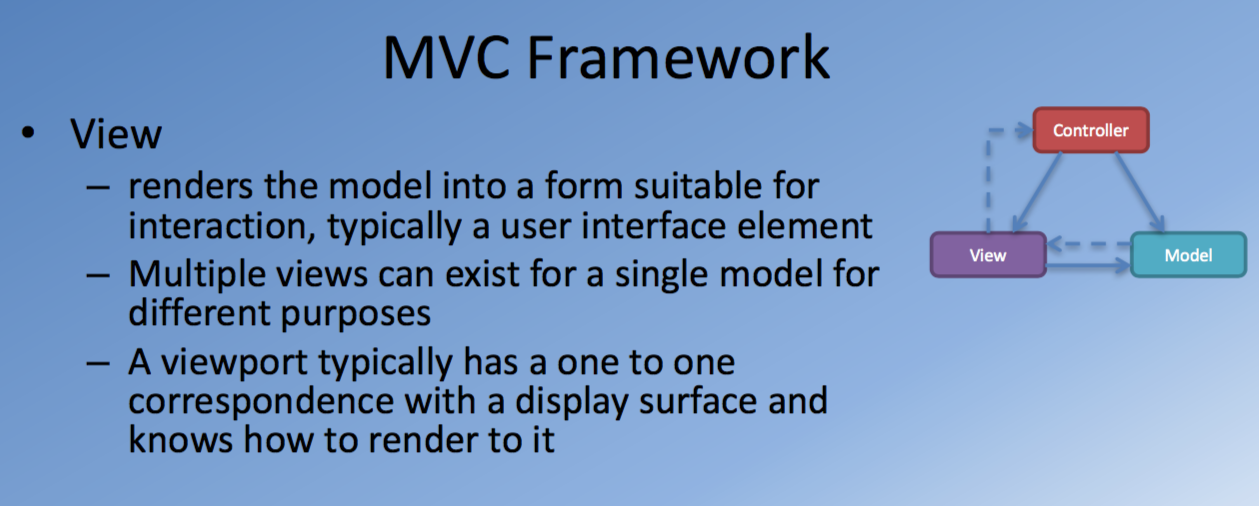
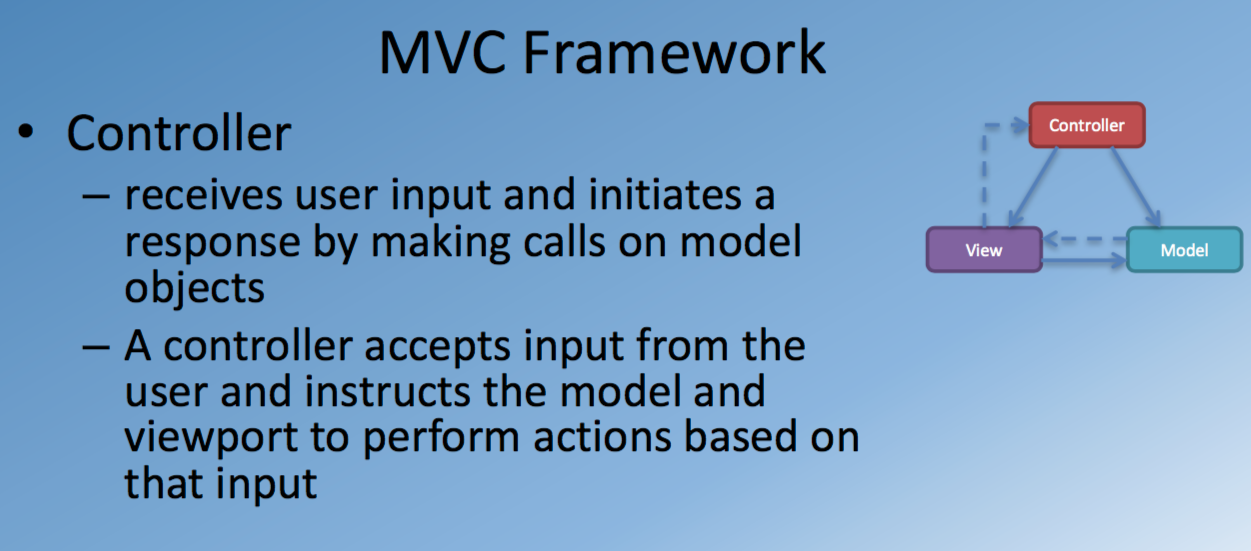
it can be viewed as a variant of the model

view view-model approach. With this quick understanding of

the MVC and the MVVM framework, let's now proceed to understand

more about angular services. [MUSIC]







Angular service: why we need ?

When we develop an angular repetition, we want to keep our component classes as clean as possible. They should be mainly concentrated on acting as the mediator between the view and the application logic, between the view and the model in some sense.

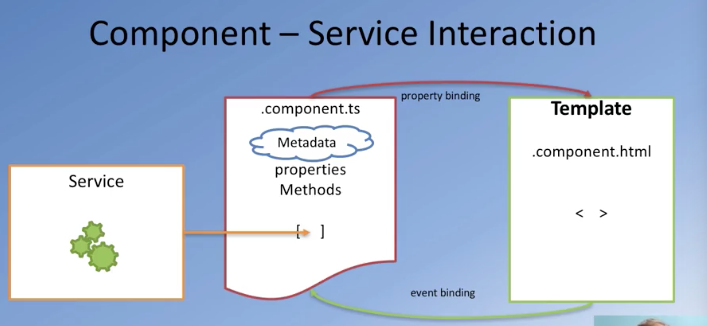
Component classes should be kept lean:

Fetching data from server, user input validation and logging should be delegated to a service

Mainly act as a mediator between the view and application logic

Try to factor out application logic into services and let them do the heavy lifting

Dependency injection



Related concepts

Dependency injection

Promises

Reactive JavaScript

Dependency injection (DI)

Software design pattern that implements inversion of control for resolving dependencies

* Dependency: an object that can be used (a service), object is dependent to another object.
* Injection: Passing of a dependency to a dependent object so that it can use it. The client does not need to build object. Able to take an object and make it available to a second object. So the second object can use it.

Dependency

If an object is dependent no another object, then there are three ways it can access to another object.

* Create dependency using new operator, using *new*
* Declare an object as global variable, and Looking up dependency using a global variable
* Have dependency passed to it where needed. If you are dependent on something else, then that something else will be injected into you by a system whenever it is required. Flexibility for software design.
* The third option is mist flexible
  + Hard coding of dependency avoided
  + Testing becomes feasible

Dependency injection involves four roles:

* The service
* The client
* The interfaces
* The injector

Angular and DI

* Separation of business logic and dependency construction
* The dependency is passed to the object consuming it where it is needed
* Angular injector subsystem is responsible for:
* Creating components
* Resolving their dependencies, and
* Providing then to other components

Exercise (Instructions): Angular Service Basics

Objectives and Outcomes

In this exercise you will create a new Angular service and inject it into your application. You will then make use of the service in the components. At the end of this exercise you will be able to:

* Implement a service and inject into your application
* Make use of the service in a component

Adding a Service

* Create a folder named *services* in the *src/app* folder.
* To add a service to your application using Angular CLI, type the following at the prompt:

ng generate service services/dish

* This will create two new files in the services folder named dish.service.ts and dish.service.spec.ts.
* Open dish.service.ts and update its contents as shown below:

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

import { Dish } from '../shared/dish';

import { DISHES } from '../shared/dishes';

@Injectable()

export class DishService {

constructor() { }

getDishes(): Dish[] {

return DISHES;

}

}

* Then add the service to the app.module.ts file as follows:

. . .

import { DishService } from './services/dish.service';

@NgModule({

. . .

providers: [DishService],

. . .

Using the Service

* Now update menu.component.ts file to make use of the service as follows:

. . .

import { DishService } from '../services/dish.service';

. . .

export class MenuComponent implements OnInit {

dishes: Dish[];

selectedDish: Dish;

constructor(private dishService: DishService) { }

ngOnInit() {

this.dishes = this.dishService.getDishes();

}

. . .

}

* Check that your application is still working correctly in the browser. Do a Git commit with the message "Basic Service".

<https://angular.io/guide/architecture#!%23services>

<https://angular.io/guide/dependency-injection>

<https://en.wikipedia.org/wiki/Model–view–controller>

<https://www.beyondjava.net/blog/model-view-whatever/>

<http://wiki.c2.com/?DesignPatternsBook>

Angular Routing: Objectives and Outcomes

In this section we cover the basic of angular routing, we examine how the router module in angular enables the navigation among views of various components that form part of an angular application. At the end of this lesion we will be able to:

Set up the router module to enable navigation among multiple component views

Set up the routes to enable the navigation

Exercise (Instructions): Header and Footer

Objectives and Outcomes

In this exercise you will add in a header and a footer to our Angular application using two Angular components. This will illustrate the use of multiple components put together form the application's view. You will also add in the Font Awesome icons for use within your application. At the end of this exercise you will be able to:

* Use multiple components and their views to put together the view of the application.
* Make use of Font Awesome icons within your Angular application

Using Font Awesome Icons

* First use NPM to fetch Font Awesome to the project by typing the following at the prompt:

npm install font-awesome –save

* Then, open the .angular-cli.json file in the project's root folder and update it as follows:

. . .

"apps": [

{

. . .

"styles": [

"styles.scss",

"../node\_modules/font-awesome/scss/font-awesome.scss"

],

. . .

}],

. . .

* You may need to restart your server by stopping and restarting the "ng serve --open".

Adding Header and Footer

* Create two new components named header and footer in your application:

<div class="container footer"

fxLayout="row"

fxLayout.sm="column"

fxLayout.xs="column"

fxLayoutAlign.xs="start center"

fxLayoutAlign.sm="start center"

fxLayoutAlign.gt-sm="center center"

fxLayoutWrap

fxLayoutGap="10px">

<div fxFlex="100%" fxFlex.gt-sm="50%">

<div fxLayout="row">

<div fxFlex="40" fxFlexOffset="20px">

<h4>Links</h4>

<md-list>

<md-list-item><a md-button>Home</a></md-list-item>

<md-list-item><a md-button>About</a></md-list-item>

<md-list-item><a md-button>Menu</a></md-list-item>

<md-list-item><a md-button>Contact</a></md-list-item>

</md-list>

</div>

<div fxFlex="50">

<h4>Our Address</h4>

<address style="text-size: 80%">

121, Clear Water Bay Road<br> Clear Water Bay, Kowloon<br> HONG KONG<br>

<i class="fa fa-phone"></i>: +852 1234 5678<br>

<i class="fa fa-fax"></i>: +852 8765 4321<br>

<i class="fa fa-envelope"></i>:

<a href="mailto:confusion@food.net">confusion@food.net</a>

</address>

</div>

</div>

</div>

<div fxFlex="100%" fxFlex.gt-sm="45%">

<div style="text-align:center">

<a md-icon-button class="btn-google-plus" href="http://google.com/+"><i class="fa fa-google-plus fa-lg"></i></a>

<a md-icon-button class="btn-facebook" href="http://www.facebook.com/profile.php?id="><i class="fa fa-facebook fa-lg"></i></a>

<a md-icon-button class="btn-linkedin" href="http://www.linkedin.com/in/"><i class="fa fa-linkedin fa-lg"></i></a>

<a md-icon-button class="btn-twitter" href="http://twitter.com/"><i class="fa fa-twitter fa-lg"></i></a>

<a md-icon-button class="btn-youtube" href="http://youtube.com/"><i class="fa fa-youtube fa-lg"></i></a>

<a md-icon-button class="btn-mail" href="mailto:"><i class="fa fa-envelope-o fa-lg"></i></a>

</div>

</div>

<div fxFlex="100" fxFlexAlign="center center">

<div style="text-align:center;">

<p>© Copyright 2015 Ristorante Con Fusion</p>

</div>

</div>

</div>

* Update the footer's styles file as follows:

$lt-gray: #ddd;

$background-dark: #512DA8;

$background-light: #9575CD;

$background-pale: #D1C4E9;

@mixin zero-margin($pad-up-down, $pad-left-right) {

margin: 0px auto;

padding: $pad-up-down $pad-left-right;

}

.footer{

background-color: $background-pale;

@include zero-margin(20px, 0px);

}

.btn-facebook {color:#fff!important; background-color:#3b5998!important;}

.btn-google-plus{color:#fff!important;background-color:#dd4b39!important;}

.btn-youtube{color:#fff!important;background-color:#ff4b39!important;}

.btn-linkedin{color:#fff!important;background-color:#007bb6!important;}

.btn-twitter{color:#fff!important;background-color:#55acee!important;}

.btn-mail{color:#fff!important;background-color:#512DA8!important;}

* Update the header's template as follows:

<md-toolbar color="primary">

<span><img src="/assets/images/logo.png" height=30 width=41></span>

<a md-button><span class="fa fa-home fa-lg"></span> Home</a>

<a md-button><span class="fa fa-info fa-lg"></span> About</a>

<a md-button><span class="fa fa-list fa-lg"></span> Menu</a>

<a md-button><span class="fa fa-address-card fa-lg"></span> Contact</a>

</md-toolbar>

<div class="container jumbotron"

fxLayout="row"

fxLayout.sm="column"

fxLayout.xs="column"

fxLayoutAlign.xs="start center"

fxLayoutAlign.sm="start center"

fxLayoutAlign.gt-sm="center center"

fxLayoutGap="10px">

<div fxFlex fxFlex.gt-sm="50%">

<h1>Ristorante Con Fusion</h1>

<p>We take inspiration from the World's best cuisines, and create a unique fusion experience. Our lipsmacking creations

will tickle your culinary senses!</p>

</div>

<div fxFlex fxFlex.gt-sm="20%">

<img src="/assets/images/logo.png" alt="Logo">

</div>

<div fxFlex></div>

* Update the header's style file as follows:

$lt-gray: #ddd;

$background-dark: #512DA8;

$background-light: #9575CD;

$background-pale: #D1C4E9;

@mixin zero-margin($pad-up-down, $pad-left-right) {

margin: 0px auto;

padding: $pad-up-down $pad-left-right;

}

.jumbotron {

@include zero-margin(70px,30px);

background: $background-light ;

color:floralwhite;

min-height: 150px;

}

* Update the project's style file styles.scss with the following:

$lt-gray: #ddd;

$background-dark: #512DA8;

$background-light: #9575CD;

$background-pale: #D1C4E9;

$primary-color-dark: #512DA8;

$primary-color: #673AB7;

$primary-color-light: #D1C4E9;

$primary-color-text: #FFFFFF;

$accent-color: #FFC107;

$primary-text-color: #212121;

$secondary-text-color: #757575;

$divider-color: #BDBDBD;

@mixin zero-margin($pad-up-down, $pad-left-right) {

margin: 0px auto;

padding: $pad-up-down $pad-left-right;

}

. . .

.background-primary {

background-color: $background-dark!important;

}

.background-accent {

background-color: $accent-color!important;

}

.text-floral-white {

color: floralwhite!important;

}

.flex-spacer {

flex: 1 1 auto;

}

* Now update the app.component.html file to include the header and footer as follows:

<app-header></app-header>

<app-menu></app-menu>

<app-footer></app-footer>

* Save all the changes and do a Git commit with the message "Header and Footer".

Angular routing basic

Angular Router

Enables navigation among views

Uses a browser URL as an instruction to navigate to a client-generate view

* Can also pass along optional parameters

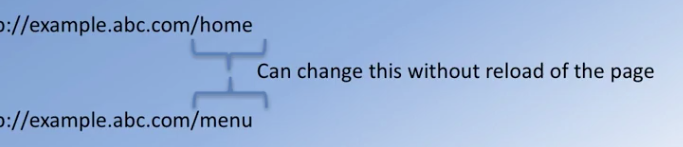
HTML5 history API

Gives developers the ability to modify a website’s URL without a full page refresh

- pushState(): add history entry

- replaceState(): modify history entry

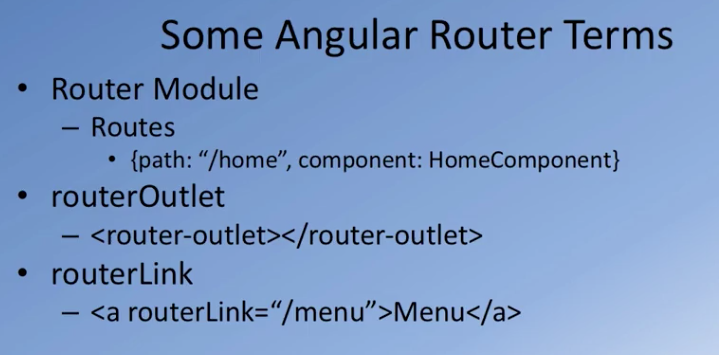
- Configure <base href=”/” >



Angular router

Angular router leverage HTML5 history manipulation to modify the browser URL





Exercise (Instructions): Angular Routing Basics

Objectives and Outcomes

In this exercise you will be adding three new additional components to your application. You will then configure a basic router using the Angular router module to enable your application to navigate among the components. At the end of this exercise you will be able to:

* Configure your application with multiple components whose views can be shown one at a time and navigate among them
* Configure a basic router using the Angular routing module to enable the navigation among the views.

Adding Components

* Add three new components to your Angular application as follows:

ng generate component about

ng generate component home

ng generate component contact

Add a Router Module

* Add a new module named app-routing to your application as follows. This will create a new module file named *app-routing.module.ts* in the *app-routing* folder.

ng generate module app-routing

* Next create a new file named *routes.ts* in the *app-routing* folder and update it as follows:

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

import { MenuComponent } from '../menu/menu.component';

import { DishdetailComponent } from '../dishdetail/dishdetail.component';

import { HomeComponent } from '../home/home.component';

import { AboutComponent } from '../about/about.component';

import { ContactComponent } from '../contact/contact.component';

export const routes: Routes = [

{ path: 'home', component: HomeComponent },

{ path: 'menu', component: MenuComponent },

{ path: '', redirectTo: '/home', pathMatch: 'full' }

];

* Update the app-routing.module.ts file to make use of the routes defined above as follows:

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

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

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

import { routes } from './routes';

@NgModule({

imports: [

CommonModule,

RouterModule.forRoot(routes)

],

exports: [ RouterModule ],

declarations: []

})

export class AppRoutingModule { }

Using the Routing Module

* Next update the app.component.html file as follows:

<app-header></app-header>

<router-outlet></router-outlet>

<app-footer></app-footer>

* Then update the app.module.ts file to use the Routing Module as follows:

. . .

import { AppRoutingModule } from './app-routing/app-routing.module';

@NgModule({

. . .

imports: [

BrowserModule,

FormsModule,

HttpModule,

MaterialModule,

FlexLayoutModule,

AppRoutingModule

],

. . .

})

. . .

* Finally update the toolbar in the header.component.html file as follows:

<md-toolbar color="primary">

<span><img src="/assets/images/logo.png" height=30 width=41></span>

<a md-button routerLink="/home"><span class="fa fa-home fa-lg"></span> Home</a>

<a md-button><span class="fa fa-info fa-lg"></span> About</a>

<a md-button routerLink="/menu"><span class="fa fa-list fa-lg"></span> Menu</a>

<a md-button><span class="fa fa-address-card fa-lg"></span> Contact</a>

</md-toolbar>

* Save all the changes and do a Git commit with the message "Angular Router Basics".

<https://angular.io/guide/router>

<https://material.angular.io/components/button/overview>

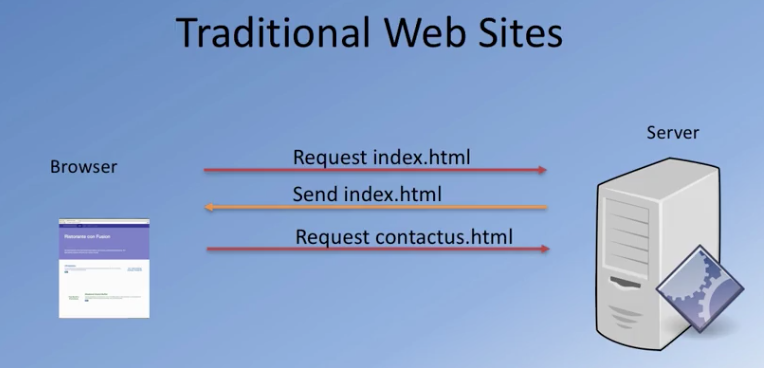
<https://developer.mozilla.org/en-US/docs/Web/API/History_API#Adding_and_modifying_history_entries>

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/base>

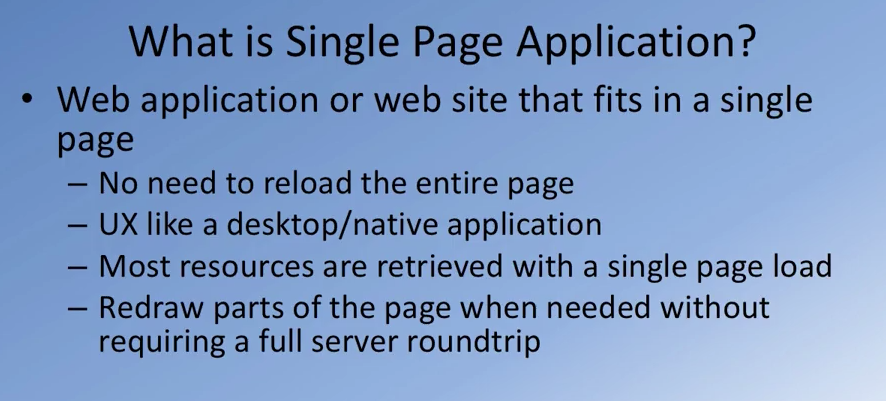
Single Page Applications: Objectives and Outcomes

In this lesson you will explore single page applications (SPA) and Angular support for SPA. You will learn to use the routes and Angular router module that enables the development of SPAs. At the end of this lesson, you will be able to:

* Design SPA using Angular
* Use the *Angular router*module to construct SPA







challenges in SPA

* search engine optimization
* Partitioning the responsibility between client and server
* Maintaining history, back and front button
* Analytics
* Speeding up the initial page load