

# **Full Stack Module-IV**

**Manual V8.3**

**ANUDIP FOUNDATION**

---



## ICONS AND THEIR MEANING



**HINTS:**  
*Get ready for helpful insites on difficult topics and questions.*



**STUDENTS:**  
*This icon symbolize important instreutions and guides for the students.*



**TEACHERS/TRAINERS:**  
*This icon symbolize important instreutions and guides for the trainers.*

Lesson No	Lesson Name	Practical Duration (Minutes)	Theory Duration (Minutes)	Page No
1	Introduction to Angular Framework	60	60	03
2	Essentials of Angular	60	60	08
3	Templates, Styles & Directives	60	60	16
4	Pipes, Services & Dependency Injection	60	60	24
5	Http Requests	60	60	28
6	Observables & rxjs operators	60	60	37
7	Angular Animation	60	60	42
8	Authentication and Route Protection	60	60	49
9	Deploying an Angular App	120	nil	59
10	Modules	60	60	62
11	App State management with Redux	120	nil	68

**Total Duration:** \_\_\_\_Hours

## Lesson 1: Introduction to Angular Framework (120 minutes)

<b>Objective:</b> After completing this lesson you will be able to learn about : <ul style="list-style-type: none"> <li>Angular Framework, History &amp; Overview</li> <li>Environment Setup, Angular CLI</li> <li>Installing Angular CLI, NPM &amp; package.json</li> <li>First Angular App</li> </ul>	<b>Materials Required:</b> <ul style="list-style-type: none"> <li>Computer With Windows XP and above</li> <li>Stable Internet connection</li> </ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

### Angular Framework – Introduction and history

AngularJS is a JavaScript framework written in JavaScript.

AngularJS used to be the “golden child” among JavaScript frameworks, as it was initially introduced by Google corporation in 2012. It was built with the Model-View-Controller concept in mind, though authors of the framework often called it “Model-View-\*” or even “Model-View-Whatever”.

The framework, written in pure JavaScript, was intended to decouple an application’s logic from DOM manipulation, and aimed at dynamic page updates. Still, it wasn’t very intrusive: you could have only a part of the page controlled by AngularJS. This framework introduced many powerful features allowing the developer to create rich, single-page applications quite easily.

It can be added to a web page with a script tag:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
```

### Angular Framework Environment Setup

We all know that Angular 6 is one of the best framework for developing Single Page Applications. So how do we setup the Angular 6 environment in Windows? Let’s make it together!

#### STEP 1: DOWNLOAD & INSTALL NODE JS ON YOUR PC

Open your browser and navigate to: <https://nodejs.org/en/>

Then Download the current version of Node JS.

#### STEP 2: INSTALL ANGULAR CLI GLOBALLY

Angular CLI is a command line interface for Angular. To install the Angular CLI;

Open your command prompt, and type `"npm install -g @angular/cli"`

#### STEP 3: CHOOSE THE BEST TEXT EDITOR FOR YOU

With any text editor you can develop the Angular project. You can consider HTML5 or anything of your choice.

## Angular CLI

Angular CLI is a command line interface tool which is used to initialize, develop, scaffold, and maintain Angular applications. You can use the command directly on command prompt or indirectly through an interactive UI i.e. Angular Console.

### Installing Angular CLI

Angular CLI tool can be installed with npm (node package manager). It requires Node 4 or higher and npm 3 or higher.

As depicted on the Angular CLI homepage we can install the package with one simple command:

```
$ npm install -g @angular/cli@1.0.0
```

However, there is a high chance that you may receive an error like this:

```
$ npm install -g @angular/cli@1.0.0
```

```
npm ERR! cb() never called!
```

This is mostly an issue with the npm. Once the node is updated, it's expected that the error will be resolved.

### Mac

On a Mac, you should be able to upgrade node through Homebrew using the following command:

```
$ brew upgrade node
```

### Windows

On a Windows or Linux machine (or a Mac that doesn't use Homebrew), you can simply return to the Node.js Download Page and select the latest version for your particular operating system. Running this installer should override your current (older) version of Node.

### Additional Packages

In addition to the angular-cli package itself, we'll also require typescript package to be installed

### Typescript

You can confirm your typescript package is installed and ready to go by running:

```
$ npm list -g typescript
```

You should receive a version number in response, like this:

```
typescript@2.1.4
```

Any version number should be fine, but if you do not receive a version number, install the package with the following command:

```
$ npm install -g typescript
```

## NPM

Node Package Manager, popularly known as npm is the package manager for the Node JavaScript platform. It puts modules in place so that node can find them, and manages dependency conflicts intelligently.

It is extremely configurable to support a wide variety of use cases. Most commonly, it is used to publish, discover, install, and develop node programs.

Run *npm help* to get a list of available commands.

## package.json

The package.json file is the heart of Node.js system. It is the manifest file of any Node.js project and contains the metadata of the project. The package.json file is the essential part to understand, learn and work with the Node.js. It is the first step to learn about development in Node.js.

The metadata information in package.json file can be categorized into the following two categories:

1. Identifying metadata properties: It basically consist of the properties to identify the module/project such as the name of the project, current version of the module, license, author of the project, description about the project etc.
2. Functional metadata properties: As the name suggests, it consists of the functional values/properties of the project/module such as the entry/starting point of the module, dependencies in project, scripts being used, repository links of Node project etc.

## First Angular App Creation

Let's create your first angular app. The app will be used for inputting a name in the box and then displaying it below.

<pre>&lt;!DOCTYPE html&gt; &lt;html&gt; &lt;script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"&gt;&lt;/script&gt; &lt;body&gt;  &lt;div ng-app=""&gt;  &lt;p&gt;Input something in the input box:&lt;/p&gt; &lt;p&gt;Name: &lt;input type="text" ng-model="name"&gt;&lt;/p&gt; &lt;p ng-bind="name"&gt;&lt;/p&gt;  &lt;/div&gt;  &lt;/body&gt; &lt;/html&gt;</pre>	<p>Input something in the input box:</p> <p>Name: <input type="text" value="mark"/></p> <p>mark</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

### 1. Create an angular app that displays your surname in the box

```
<!DOCTYPE html>

<html>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<body>

<div ng-app="">

<p>Input something in the input box:</p>

<p>Name: <input type="text" ng-model="surname"></p>

<p ng-bind="surname"></p>

</div>

</body>

</html>
```

### 2. Create an angular app that displays your first name in the box

### 3. Create an angular app that displays your age in the box

**Note:** After saving the code, click the created HTML file to view the output.

## Reviewing the chapter

- AngularJS is a JavaScript framework written in JavaScript.
- The framework, written in pure JavaScript, was intended to decouple an application's logic from DOM manipulation, and aimed at dynamic page updates.

- Angular CLI is a command line interface tool which is used to initialize, develop, scaffold, and maintain Angular applications.
- Node Package Manager, popularly known as npm is the package manager for the Node JavaScript platform. It puts modules in place so that node can find them, and manages dependency conflicts intelligently.
- The package.json file is the heart of Node.js system. It is the manifest file of any Node.js project and contains the metadata of the project.

### Testing your skills

1. AngularJS was built with the \_\_\_\_\_ concept

a) Model-view, b) Stand-view, c) Array-view, d) Module-view

2. Angular JS was introduced in the year

a) 1998, b) 2005, c) 2011, d) 2017

3. Angular CLI is used for which of the following functionalities of an application

a) develop, b) scaffold, c) maintain, d) all of the above

4. To install Angular, which version of node is required

a) 3, b) 4, c) Both (a) and (b), d) All of the above

5. NPM stands for

a) node product manager, b) node package manager, c) nodal product maintenance, d) None of the above



## Lesson 2: Essentials of Angular (120 minutes)

**Objective:** After completing this lesson you will be able to learn about:

- Component Basics
- Creating Components using CLI
- Nesting Components
- Data Binding - Property & Event Binding
- Two-way data binding
- Input Properties, Output Properties

**Materials Required:**

- Computer With Windows XP and above
- Stable Internet connection

**Self- Learning Duration:** 60 minutes

**Practical Duration:** 60 minutes

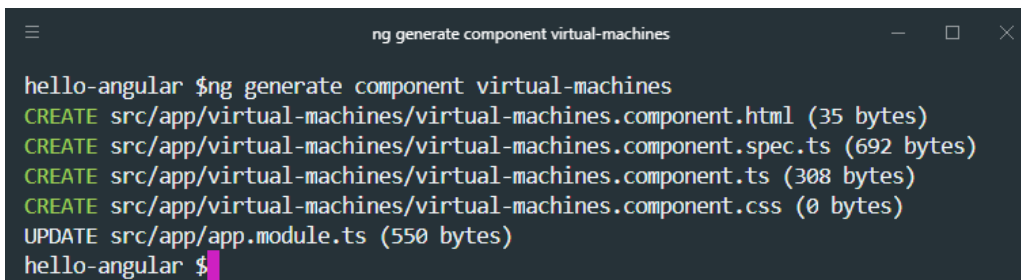
**Total Duration:** 120 minutes

### Components

Components are like the basic building block in an Angular application. Components are defined using the `@component` decorator. A component has a selector, template, style and other properties, using which it specifies the metadata required to process the component.

### Creating Components using CLI

You'll want to have the **ng serve** process already running in a command line window for your application. Then, you can open another instance of the command line and type **ng generate component virtual-machines** to create a new *virtual-machines* component.

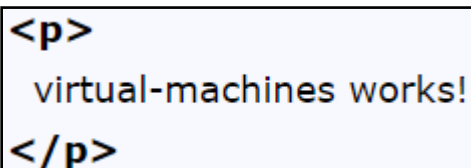


```
ng generate component virtual-machines

hello-angular $ng generate component virtual-machines
CREATE src/app/virtual-machines/virtual-machines.component.html (35 bytes)
CREATE src/app/virtual-machines/virtual-machines.component.spec.ts (692 bytes)
CREATE src/app/virtual-machines/virtual-machines.component.ts (308 bytes)
CREATE src/app/virtual-machines/virtual-machines.component.css (0 bytes)
UPDATE src/app/app.module.ts (550 bytes)
hello-angular $
```

It looks like the CLI automatically created the new folder to hold our new component in `app/virtual-machines`. In addition we see four new files associated with the automatically generated component:

#### 1. virtual-machines.component.html



```
<p>
  virtual-machines works!
</p>
```

## 2. virtual-machines.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { VirtualMachinesComponent } from './virtual-machines.component';

describe('VirtualMachinesComponent', () => {
  let component: VirtualMachinesComponent;
  let fixture: ComponentFixture<VirtualMachinesComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ VirtualMachinesComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(VirtualMachinesComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

## 3. virtual-machines.component.ts

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

@Component({
  selector: 'app-virtual-machines',
  templateUrl: './virtual-machines.component.html',
  styleUrls: ['./virtual-machines.component.css']
})
export class VirtualMachinesComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

## 4. virtual-machines.component.css

```
1 /* todo: add your styles here */
```

## Nesting Angular Components

Now our application has two custom built components. Those are virtual-machine and virtual-machines. Now the purpose of our virtual-machines component is to hold one or more instances of a virtual-machine component. Therefore, we are going to nest the virtual-machine inside of virtual-machines. We can open up virtual-machines.component.html and remove the default markup we see above. We'll replace it with two instances of the virtual-machine component like so.

```
<app-virtual-machine></app-virtual-machine>
<hr>
<app-virtual-machine></app-virtual-machine>
```

Now, recall that when we built an angular component by hand, we had to manually register the component in the **app.modules.ts** file. The nice thing about building an Angular component using the command line like we did here is that this step is taken care of automatically for you.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { VirtualMachineComponent } from './virtual-machine/virtual-machine.component';
import { VirtualMachinesComponent } from './virtual-machines/virtual-machines.component';
@NgModule({
  declarations: [
    AppComponent,
    VirtualMachineComponent,
    VirtualMachinesComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Now we want to check the **virtual-machines.component.ts** Typescript file to see that the CLI also automatically scaffolds the `@Component` decorator and it's configuration including the selector, templateUrl, and styleUrls.

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

@Component({
  selector: 'app-virtual-machines',
  templateUrl: './virtual-machines.component.html',
  styleUrls: ['./virtual-machines.component.css']
})
export class VirtualMachinesComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

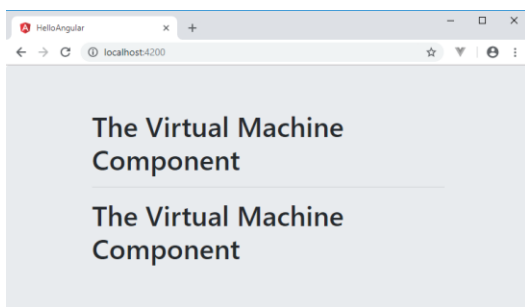
### Using The New Component!

Finally, in order to use the new Angular component we just built we'll need to update **app.component.html**

```
<div class="jumbotron">
  <div class="container">
    <app-virtual-machines></app-virtual-machines>
  </div>
</div>
```

So what should happen now? Well, now there is an instance of `<app-virtual-machines></app-virtual-machines>` in the main `app.component.html` file.

We had nested two instances of `<app-virtual-machine></app-virtual-machine>` inside of that component. That means that at this point we should see two virtual-machine components on the page and that is exactly what we get when visiting the browser. Focus on the image below:



Additionally, we can also inspect the generated markup in Chrome developer tools and find that indeed we have two angular components nested inside of the outer wrapping component. Have a look:

```
<app-virtual-machines _ngcontent-c0="" _ngghost-c1="">
<app-virtual-machine _ngcontent-c1=""><h1>The Virtual Machine Component</h1></app-virtual-machine>
<hr _ngcontent-c1="">
<app-virtual-machine _ngcontent-c1=""><h1>The Virtual Machine Component</h1></app-virtual-machine>
</app-virtual-machines>
```

## Data Binding

Data binding is a core concept in Angular and allows to define communication between a component and the DOM, making it very easy to define interactive applications without worrying about pushing and pulling data. There are four forms of data binding and they differ in the way the data is flowing.

### Data Binding from the Component to the DOM

Interpolation: {{ value }}

This adds the value of a property from the component:

```
<li>Name: {{ user.name }}</li>
```

```
<li>Email: {{ user.email }}</li>
```

Property binding: [property]="value"

With property binding, the value is passed from the component to the specified property, which can often be a simple html attribute:

```
<input type="email" [value]="user.email">
```

Here are two more examples, one that applies a background-color from the value of **selectedColor** in the component and one that applies a class name if **isSelected** evaluates to true:

```
<div [style.background-color]="selectedColor">
```

```
<div [class.selected]="isSelected">
```

### From the DOM to the Component

Event binding: (event)="function"

When a specific DOM event happens (eg.: click, change, keyup), call the specified specified method in the component. In the example below, the **cookBacon()** method from the component is called when the button is clicked:

```
<button (click)="cookBacon()"></button>
```

## Two-way

Two-way data binding: [(ngModel)]="value"

Using what's called the banana in a box syntax, two-way data binding allows to have the data flow both ways. In this example, the `user.email` data property is used as the value for the input, but if the user changes the value, the component property gets updated automatically to the new value:

```
<input type="email" [(ngModel)]="user.email">
```

## @Input() and @Output() properties

@Input() and @Output() allow Angular to share data between the parent context and child directives or components.

An @Input() property is writable while an @Output() property is observable.

Consider this example of a child/parent relationship:

*content\_copy*

```
<parent-component>
```

```
<child-component></child-component>
```

```
</parent-component>
```

Here, the `<child-component>` selector, or child directive, is embedded within a `<parent-component>`, which serves as the child's context.

@Input() and @Output() act as the API, or application programming interface, of the child component in that they allow the child to communicate with the parent.

Think of @Input() and @Output() like ports or doorways—@Input() is the doorway into the component allowing data to flow in while @Output() is the doorway out of the component, allowing the child component to send data out.

## @Input() and @Output() are independent

Though @Input() and @Output() often appear together in apps, you can use them separately. If the nested component is such that it only needs to send data to its parent, you wouldn't need an @Input(), only an @Output(). The reverse is also true in that if the child only needs to receive data from the parent, you'd only need @Input().

## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

**1. Create a program where if you change the name inside the input field, the model data will change automatically, and therefore also the header will change its value.**

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
Name: <input ng-model="firstname">
<h1>{{firstname}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
$scope.firstname = "John";
$scope.lastname = "Doe";
});
</script>
<p>Change the name inside the input field, and the model data will change automatically, and therefore
also the header will change its value.</p>
</body>
</html>
```

**2. Create a program using the ng-model directive on HTML controls (input, select, textarea) to bind data between the view and the data model.**

**Note:** After saving the code, click the created HTML file to view the output.

## Reviewing the chapter

- A component has a selector, template, style and other properties, using which it specifies the metadata required to process the component.

- Data binding is a core concept in Angular and allows to define communication between a component and the DOM, making it very easy to define interactive applications without worrying about pushing and pulling data.
- @Input() and @Output() act as the API, or application programming interface, of the child component in that they allow the child to communicate with the parent.

### Testing your skills

1. Components are defined using the \_\_\_\_\_decorator  
a) @decorator, b) @component, c) @comp, d) None of the above
2. Which of the following are properties of a component  
a) template, b) style, c) selector, d) all of the above
3. Defining a communication between a component and the DOM is referred to as \_\_\_\_\_  
a) data mastering, b) data binding, c) data caching, d) data commuting
4. \_\_\_\_\_ allows Angular to share data between the parent context and child directives or components  
a) output, b) input, c) Both (a) and (b), d) None of the above



**Lesson 3: Templates, Styles, and Directives (120 minutes)****Objective:** After completing this lesson you will be able to learn about:

- Templates
- Styles
- Bootstraps

**Materials Required:**

- Computer With Windows XP and above
- Stable Internet connection

**Self- Learning Duration:** 60 minutes**Practical Duration:** 60 minutes**Total Duration:** 120 minutes**Template**

A template is an HTML snippet that tells Angular how to render the component in angular application.

The template is immediately associated with a component defines that component's view.

When writing the codes, it is not necessary to write the template code inline with the component code. Rather, the HTML template files can be stored separately and they can be simply referred in the component through the use of templateUrl property.

There are two ways of defining template in an angular component.

Let's focus on an example:

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

@Component({
  selector: 'app-zipcode',
  template: `
    <p>
      zipcode inline template
    </p>
  `,
  styleUrls: ['./zipcode.component.css']
})

export class ZipcodeComponent implements OnInit {

  constructor() { }

  ngOnInit() { }

}
```

Note: The inline template is defined by placing the HTML code in back ticks " and is linked to the component metadata using the template property of @Component decorator.

To define the inline template using @angular/cli, you are required to use the command below:

```
ng generate component zipcode -it
```

Next, you need to focus on the template file. The template is defined in a separate HTML file and is linked to the component metadata using the `@Component` decorator's `templateUrl` property. Focus on the example below:

```
zipcode.component.html

<p>
  zipcode separate HTML template
</p>

zipcode.component.ts

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

@Component({
  selector: 'app-zipcode',
  templateUrl: './zipcode.component.html',
  styleUrls: ['./zipcode.component.css']
})
export class ZipcodeComponent implements OnInit {
  constructor() { }

  ngOnInit() {
  }
}
```

## Styles

We can also specify any custom css styles for our component with the *styles* property.

*styles* takes an array of strings and just like template we can use multi-line strings with back-ticks.

Do keep a note that the styles property can work with array of strings as well. In such cases, each and every string may contain any number of CSS notifications.

## Adding bootstrap to angular app

Bootstrap is definitely a highly utilized front-end component library. It allows developers to add a sophisticated grid system, use components such as dropdowns, modals, navigation bars and progress bars.

To use Bootstrap in an Angular project, first, make sure to create an Angular application via the Angular CLI by executing `ng new [app-name]`.

Let's start:

To create a new project write the below command in the terminal/command line.

```
ng new angular-bootstrap-example
```

## Installing Bootstrap

We have two steps to add bootstrap in angular project

### a: Installing Bootstrap from NPM

Next, we need to install Bootstrap. So, Type in your terminal cd. In this case, it is **cd angular-bootstrap-example**. Press enters and type below command to install bootstrap :

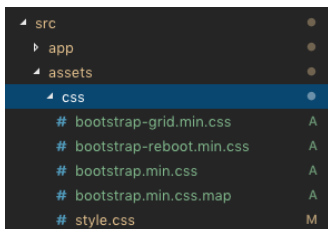
```
npm install bootstrap@3.3.7 ( install a particular version of bootstrap, mostly for Bootstrap 3)
```

or

```
npm install bootstrap (directly installed latest version, the Bootstrap 4)
```

### b: Alternative – Local Bootstrap CSS

You can also download the Bootstrap CSS and add in your project. I have downloaded Bootstrap from the <https://getbootstrap.com/> and added into src/assets/css/bootstrap.min.css



## Importing the CSS

We have two variations to import bootstrap in angular project

### a: If you have installed Bootstrap from NPM

Just open the angular.json file and give your style path like below:

```
"styles": [  
  "node_modules/bootstrap/dist/css/bootstrap.css",  
  "src/styles.css"  
],
```

### b: If you added the Bootstrap CSS file locally

just open angular.json file and give your style path like below:

```
"styles": [  
  "src/styles.css",  
  "src/assets/css/bootstrap-grid.min.css",  
  "src/assets/css/bootstrap-reboot.min.css",  
  "src/assets/css/bootstrap.min.css",  
  "src/assets/css/style.css"  
],
```

### The way to use Custom CSS in Angular

Make one CSS file in your src/assets/css/custom-style.css

```
└─ assets  
  └─ css  
      # custom-style.css
```

just open the angular.json file and give your style path like below:

```
"styles": [  
  "node_modules/angular-bootstrap-md/scss/bootstrap/bootstrap.scss",  
  "node_modules/angular-bootstrap-md/scss/mdb-free.scss",  
  "src/styles.scss",  
  "src/assets/css/custom-style.css"  
],
```

**Note:** Your custom CSS should be last of other CSS

### Installing Bootstrap JavaScript Components with ngx-bootstrap (Option 1)

In some case, you want to use models, accordion, dropdown without installing jquery. So, angular provides wrapper library for Bootstrap called ngx-bootstrap and that we can also install from NPM:

#### npm i ngx-bootstrap

Adding the required Bootstrap modules in app.module.ts

If you want to use any ngx-bootstrap modules so go through ngx-bootstrap and add the module you want and also add that module in your app.module.ts.

For example, suppose we want to use the Accordion component (.forRoot() is like a global scope to provide the functionality of all modules be available in all components.):

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { AccordionModule } from 'ngx-bootstrap/accordion';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AccordionModule.forRoot()
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

## Installing Bootstrap JavaScript Components with ng-bootstrap (Option 2)

If you use Bootstrap 4 in your project and also used models, accordion, dropdown without installing jquery. In case angular provides wrapper library for Bootstrap 4 is: ng-bootstrap and that we can also install from NPM:

### npm i @ng-bootstrap/ng-bootstrap

You also need to import NgbModule, in app.module.ts :

Other modules in your application can simply import NgbModule app.module.ts:

```
import {NgbModule} from '@ng-bootstrap/ng-bootstrap';
```

```
imports: [
  BrowserModule,
  NgbModule
],
```

### If you want to use mdbbootstrap

install mdb via NPM: *npm i angular-bootstrap-md*

#### a: Add below code in you app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, NO_ERRORS_SCHEMA } from '@angular/core';
import { AppComponent } from './app.component';
import { MDBBootstrapModule } from 'angular-bootstrap-md';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    MDBBootstrapModule.forRoot(),
  ],
  schemas: [ NO_ERRORS_SCHEMA ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

**b: Rename your style**

rename */src/styles.css* to *styles.scss*

**c: Install external libs**

npm install --save chart.js@2.5.0 hammerjs

**d: Add below lines to angular.json**

```
"styles": [  
  "node_modules/angular-bootstrap-md/scss/bootstrap/bootstrap.scss",  
  "node_modules/angular-bootstrap-md/scss/mdb-free.scss",  
  "src/styles.scss"  
],  
"scripts": [  
  "node_modules/chart.js/dist/Chart.js",  
  "node_modules/hammerjs/hammer.min.js"  
]
```

**e: ng serve****Built-in Directives**

AngularJS has a set of built-in directives which offers functionality to your applications. AngularJS lets you extend HTML with new attributes called Directives. Through these directives, you can also extend the HTML functionality. AngularJS also lets you define your own directives.

The most common types are listed below:

- The ng-app directive is used for initializing an AngularJS application. This directive also tells AngularJS that the <div> element is the "owner" of the AngularJS application.
- The ng-init directive is used for initializing application data.
- The ng-model directive is used for binding the value of HTML controls (input, select, textarea) to application data.

Below is a sample example:

```
<!DOCTYPE html>  
<html>  
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>  
<body>  
  
<div ng-app="" ng-init="firstName='John'">  
<p>Input something in the input box:</p>  
<p>Name: <input type="text" ng-model="firstName"></p>  
<p>You wrote: {{ firstName }}</p>  
</div>  
</body>  
</html>
```

Input something in the input box:

Name:

You wrote: John

## Reviewing the chapter

A template is an HTML snippet that tells Angular how to render the component in angular application

We can also specify any custom css styles for our component with the styles property.

styles takes an array of strings and just like template we can use multi-line strings with back-ticks.

Bootstrap is definitely a highly utilized front-end component library. It allows developers to add a sophisticated grid system, use components such as dropdowns, modals, navigation bars and progress bars.

AngularJS has a set of built-in directives which offers functionality to your applications. AngularJS lets you extend HTML with new attributes called Directives.

## Practical Session

**Instructions: Create the code....using an editor....and save it with HTML extension.**

### 1. Create a template program that binds two expressions to the <p> element

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-bind-template="{{firstName}} {{lastName}}" ng-controller="myCtrl">
</div>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
$scope.firstName = "John";
$scope.lastName = "Doe";
});
</script>
</body>
</html>
```

### 2. Create a template program that binds two expressions, country name and capital name to the <p> element

**Note:** After saving the code, click the created HTML file to view the output.

**Testing your skills**

1. A \_\_\_\_\_ is an HTML snippet that tells Angular how to render the component in angular application  
a) template, b) Constructors, c) Functions, d) All of the above
  
2. The inline template is defined by placing the HTML code in \_\_\_\_\_  
a) colon, b) flower brackets, c) back ticks , d) None of the above
  
3. To specify any custom css styles for a component can be done using the \_\_\_\_\_property  
a) design, b) styles, c) cssany, d) theme
  
4. Bootstrap can use components like  
a) modals, b) navigation bar, c) dropdowns, d) All of the above
  
- 5) \_\_\_\_\_ directive is used for initializing application data.  
a) ng-app, b) ng-init, c) ng-model, d) All of the above



## Lesson 4: Pipes, Services & Dependency Injection (120 minutes)

<b>Objective:</b> After completing this lesson you will be able to : <ul style="list-style-type: none"> <li>Understand about pipes and its applications in angular</li> <li>Dependency injection, services, and their procedures</li> </ul>	<b>Materials Required:</b> <ul style="list-style-type: none"> <li>Computer With Windows XP and above</li> <li>Stable Internet connection</li> </ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

### Introduction to pipes

Pipes are referred to as filters in Angular 1 and 2. Later on, with update versions, they used to be termed as pipes. Their primary function is to transform data. Pipes are denoted by symbol "|".

The syntax:

```
{{title | uppercase}}
```

Pipes take integers, strings, arrays, and date as input separated with | to be converted in the format as required and display the same in the browser.

Let's focus on an example (*The example will display the title text in upper and lower case by using pipes*)

Step 1: Define a variable named "title" in *component.ts* file.

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

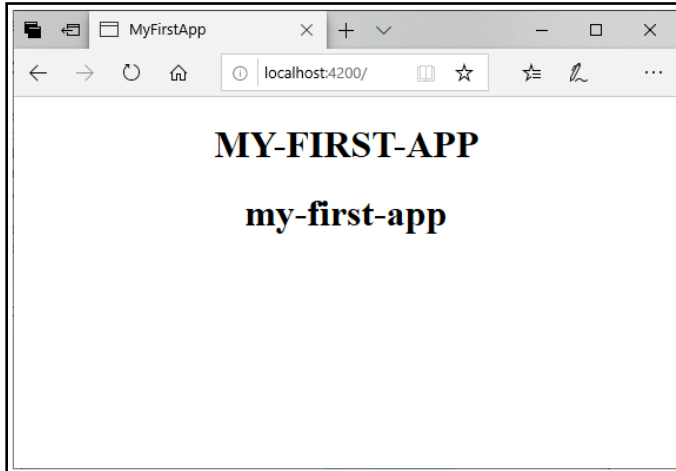
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-first-app';
}
```

Step 2: Use the pipe symbol in component.html file

```
<h1>
  {{ title | uppercase }} <br/></h1>
<h1>
  {{ title | lowercase }} <br/></h1>
```

The outcome:

Run ng serve and this is what you will achieve:



## Services

Service is a broad category encompassing any value, function, or feature that an app needs. A service can be best defined as a class with a narrow, well-defined purpose. It should do something specific and do it well.

Angular distinguishes components from services to increase modularity and reusability. We might come across a situation where we need some code to be used everywhere on the page. It can be for data connection that needs to be shared across components, etc. Services help us achieve that. With services, we can access methods and properties across other components in the entire project.

Let's focus on an example where the service class tries logging to the browser console:

src/app/logger.service.ts (class)

```
export class Logger {  
  log(msg: any) { console.log(msg); }  
  error(msg: any) { console.error(msg); }  
  warn(msg: any) { console.warn(msg); }  
}
```

Keep in mind that services can depend on other services. Focus below on the Hero Service that is dependent on the above logger service.

src/app/hero.service.ts (class)

```
export class HeroService {  
  private heroes: Hero[] = [];  
  
  constructor(  
    private backend: BackendService,  
    private logger: Logger) {}  
  
  getHeroes() {  
    this.backend.getAll(Hero).then( (heroes: Hero[]) => {  
      this.logger.log(`Fetched ${heroes.length} heroes.`);  
      this.heroes.push(...heroes); // fill cache  
    });  
    return this.heroes;  
  }  
}
```

## Dependency injection (DI)

Dependency Injection is a design pattern that allows for the creation of dependent objects outside of a class and provides those objects to a class through different ways.

To define a class as a service in Angular, use the `@Injectable()` decorator to provide the metadata that allows Angular to inject it into a component as a dependency. Similarly, use the `@Injectable()` decorator to indicate that a component or other class (such as another service, a pipe, or an `NgModule`) has a dependency.

The injector is the main mechanism. Angular creates an application-wide injector for you during the bootstrap process, and additional injectors as needed. You don't have to create injectors.

Always remember, it is the injector that creates dependencies, and maintains a container of dependency instances that it reuses if possible.

A provider is an object that tells an injector how to obtain or create a dependency. For any dependency that you need in your app, you must register a provider with the app's injector, so that the injector can use the provider to create new instances. For a service, the provider is typically the service class itself.

A dependency doesn't have to be a service—it could be a function, or even a value.

## Practical Session

**Instructions: Create the code....using an editor....and save it with HTML extension.**

### 1. Create a program that adds a pipe to the angular module

```
@NgModule({  
  declarations: [  
    AppComponent  
  ],  
  imports: [  
    BrowserModule  
  ],  
  providers: [  
    DatePipe
```

```
],  
bootstrap: [AppComponent]  
})  
export class AppModule { }
```

## 2. Create a program where you can use the pipe by dependency injection

**Note:** After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

- Pipes are primarily used to transform data. Pipes are denoted by symbol "|".
- Pipes take integers, strings, arrays, and date as input separated with | to be converted in the format as required and display the same in the browser.
- A service can be best defined as a class with a narrow, well-defined purpose. It should do something specific and do it well.
- Angular distinguishes components from services to increase modularity and reusability.
- Dependency Injection is a design pattern that allows for the creation of dependent objects outside of a class and provides those objects to a class through different ways.

### Testing your skills

1. Pipes were referred to as filters in which versions

a) 1, b) 2, c) 3, d) Both (a) and (b)

2. A \_\_\_\_\_ can be best defined as a class with a narrow, well-defined purpose.

a) service, b) function, c) pipe, d) None of the above

3. Angular distinguishes components from services to increase \_\_\_\_\_ and \_\_\_\_\_

a) modularity, b) reusability, c) Both (a) and (b), d) none of the above

4. Use the \_\_\_\_\_ decorator to indicate that a component or other class

a) @comp(), b) @Injectable(), c) @classcomp()>, d) @decclass()

5. A \_\_\_\_\_ is an object that tells an injector how to obtain or create a dependency

a) provider, b) angular, c) pipe, d) none of the above

## Lesson 5: HTTP Requests (120 minutes)

**Objective:** After completing this lesson you will be able to learn about :

- App & Backend Setup
- PUT and GET Requests
- HTTP errors

**Materials Required:**

- Computer With Windows XP and above
- Stable Internet connection

**Self- Learning Duration:** 60 minutes

**Practical Duration:** 60 minutes

**Total Duration:** 120 minutes

Http Service is used for fetching external data, post to it, etc. To do so, it is necessary to import the http module to make use of the http service.

HttpClient is the mechanism being followed by Angular to communicate with a remote server over HTTP.

Do you want to make this HttpClient available everywhere in the app? This can be done through the following couple of steps:

Step 1: Add it to the *root AppModule* by importing it:

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

Step 2: Add HttpClient to the imports array, while being in the AppModule

```
@NgModule({
  imports: [
    HttpClientModule,
  ],
})
```

Let's focus on another example that will import the module in *app.module.ts* to start using the http service:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
```

```
import { HttpClientModule } from '@angular/http';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Focus on the areas of the code that have been kept bold. These bold areas suggest that we have imported the HttpClientModule from @angular/http and the same is also added in the imports array.

Let us now use the http service in the app.component.ts.

```
import { Component } from '@angular/core';
import { Http } from '@angular/http';
import 'rxjs/add/operator/map';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  constructor(private http: Http) { }
  ngOnInit() {
    this.http.get("http://jsonplaceholder.typicode.com/users").
```

```
map((response) ⇒ response.json()).  
subscribe((data) ⇒ console.log(data))  
}  
}
```

Let us understand the code highlighted above. We need to import http to make use of the service, which is done as follows –

```
import { Http } from '@angular/http';
```

In the class AppComponent, a constructor is created and the private variable http of type Http. To fetch the data, we need to use the get API available with http as follows:

```
this.http.get();
```

It takes the url to be fetched as the parameter as shown in the code.

## App Setup through CLI

To start an angular app, there are four major steps to follow. Have a look:

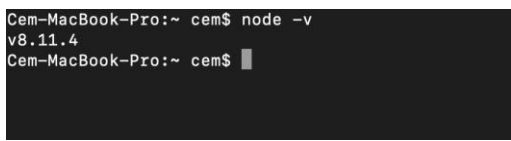
- Node Package Manager (NPM) installation (prerequisite)
- Angular CLI installation
- Creating an Angular app with the CLI
- Running the app

## Node Package Manager (NPM) installation

Angular requires Node.js in your system (version 8.x or 10.x). If you already have it, you can skip this part.

Node has the Node Package Manager (NPM), which we need for installing Angular CLI and other programs (packages). There are sites through which you can download the NPM.

After the installation is completed, go to your Terminal or Windows Command Prompt and type `node -v` to check which version is installed:



```
Cem-MacBook-Pro:~ cem$ node -v  
v8.11.4  
Cem-MacBook-Pro:~ cem$
```

## Angular CLI installation

This is the second stage. Once the NPM has been installed, simply open the command prompt and type the following command:

```
npm install -g @angular/cli
```

Do note that 'g' stands for global installation. Including it will allow the CLI to be used in any Angular project at a later stage. Once the installation is completed, type 'ng v' to verify whether the installation is successful. Again, 'ng' stands for Angular and '-v' is version.

```
Cem-MacBook-Pro:~ cem$ ng -v

Angular CLI
-----
Angular CLI: 6.1.5
Node: 8.11.4
OS: darwin x64
Angular:
...

Package      Version
-----
@angular-devkit/architect    0.7.5
@angular-devkit/core         0.7.5
@angular-devkit/schematics   0.7.5
@schematics/angular          0.7.5
@schematics/update            0.7.5
```

## Creating an Angular app with the CLI

You can now install an Angular app with the CLI. Again from your terminal, first choose a file path for where to install your source code. I will select a location as my Desktop with `cd Desktop` command, then type the following command to install your first Angular Project:

```
ng new my-first-app
```

This command will install the Angular app with all the configurations needed. You can choose of course another name.

```
Cem-MacBook-Pro:~ cem$ cd Desktop
Cem-MacBook-Pro:Desktop cem$ ng new my-first-app
CREATE my-first-app/README.md (1027 bytes)
CREATE my-first-app/angular.json (3602 bytes)
CREATE my-first-app/package.json (1317 bytes)
CREATE my-first-app/tsconfig.json (408 bytes)
CREATE my-first-app/tslint.json (2805 bytes)
CREATE my-first-app/.editorconfig (245 bytes)
CREATE my-first-app/.gitignore (503 bytes)
CREATE my-first-app/src/favicon.ico (5430 bytes)
CREATE my-first-app/src/index.html (297 bytes)
CREATE my-first-app/src/main.ts (370 bytes)
CREATE my-first-app/src/polyfills.ts (3194 bytes)
CREATE my-first-app/src/test.ts (642 bytes)
CREATE my-first-app/src/styles.css (80 bytes)
```

## Running the App

Angular supports Live Server, so you can see the changes in your local browser automatically without refreshing the page.

After your Angular App is initialized successfully, go to your project folder with **cd name-of-your-app**.

Use the following command to run the Angular app:

```
ng serve --open
```

The 'serve' command will result in starting the Angular Application while the 'open' command will open it automatically in the web browser. Have a look below:



```
Cem-MacBook-Pro:Desktop cem$ cd my-first-app
Cem-MacBook-Pro:my-first-app cem$ ng serve --open
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

Date: 2018-11-03T17:18:33.375Z
Hash: 827efc4d9a22152d871a
Time: 5979ms
chunk {main} main.js, main.js.map (main) 10.7 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 227 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 5.22 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 15.6 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.29 MB [initial] [rendered]
[webpack]: Compiled successfully.
```

This will result in opening of a browser page and gotcha! You have successfully set up the back end and app and have launched the first app.

## The GET method

GET is one of the most common HTTP methods and used primarily to request data from a specified resource.

### Features of GET:

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests is only used to request data (not modify)

## The PUT method

PUT is used to send data to a server to create/update a resource.

## The POST method

POST is used to send data to a server to create/update a resource. This data is stored in the request body of the HTTP request.

### Features of POST:

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

## HTTP Errors

The HTTP Errors occur when a HTTP Request is being sent through the use of HttpClient Module. Interceptors are used for handling these errors.

There are two categories of errors:

1. Server side
2. Client Side

Server side errors occur when there happens to be unauthorized access, server down scenarios, or session expired.

Client side errors happen primarily when trying to generate the HTTP Request. In such scenarios, the most common category is network error.

### Error handling – the default scenario

The default Error handling in Angular is handled by *Errorhandler* class, which is part of the @angular/core module. This is global error handler class which catches all exception occurring in the App. This class has a method *handleError(error)*. Whenever the app throws an unhandled exception anywhere in the application angular intercepts that exception. It then invokes the method *handleError(error)* which writes the error messages to browser console.

### An example

Create a new Angular application.

First, create a HTML file, app.component.html

```
<h1> {{title}} </h1>
<button (click)="throwError1()"> Throw Error-1 </button>
<button (click)="throwError2()"> Throw Error-2 </button>
<router-outlet></router-outlet>
```

Next, create a typescript file, app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent
{
  title: string = 'Global Error Handler in Angular' ;
  throwError1() {
```

```
var a= b;
}
throwError2() {
try {
var a= b;
} catch (error) {
//here you can handle the error
//
}
}
}
```

The code mimics an error by using the statement `var a= b;`, where `b` is not defined. The first method `throwError1()` does not handle error, while `throwError2()` method uses `try..catch` block to handle the error.

Run the app and keep the chrome developer tool open. Click on throw error 1 button. The default Error Handler of angular intercepts the error and writes to the console as shown in image below

But, clicking on the throw error 2 button, does not trigger the Error Handler as it is handled by using the `try..catch` block.

If you are not handling the error in the `try..catch` block, then you must use throw error so that the default error handler can catch it.

```
throwError2() {
  try {
    var a= b;
  } catch (error) {
    throw error;    //rethrow the error
  }
}
```

### Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

**1. Create an application called myShoppingList, and add a controller named myCtrl to it.**

```
<!DOCTYPE html>
```

```
<html>
```

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
```

```
<body>
```

```
<script>
var app = angular.module("myShoppingList", []);
app.controller("myCtrl", function($scope) {
$scope.products = ["Milk", "Bread", "Cheese"];
});
</script>
<div ng-app="myShoppingList" ng-controller="myCtrl">
<ul>
<li ng-repeat="x in products">{{x}}</li>
</ul>
</div>
<p>So far we have made an HTML list based on the items of an array.</p>
</body>
</html>
```

## 2. Create an application called topholidaydestinationst, and add a controller named TrSf to it.

**Note:** After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

- Http Service is used for fetching external data, post to it, etc. To do so, it is necessary to import the http module to make use of the http service.
- HttpClient is the mechanism being followed by Angular to communicate with a remote server over HTTP
- GET is one of the most common HTTP methods and used primarily to request data from a specified resource.
- PUT is used to send data to a server to create/update a resource.
- POST is used to send data to a server to create/update a resource. This data is stored in the request body of the HTTP request
- The HTTP Errors occur when a HTTP Request is being sent through the use of HttpClient Module. Interceptors are used for handling these errors.

**Testing your skills**

1. \_\_\_\_\_ method is used primarily to request data from a specified resource.  
a) GET, b) PUT, c) POST, d) none of the above
  
2. HTTP Errors occur when a HTTP Request is being sent through the use of \_\_\_\_\_Module  
a) HttpFunction, b) HttpServer, c) HttpClient, d) none of the above
  
3. Which of these are features of GET method  
a) can be cached, b) can be bookmarked, c) have length restrictions, d) all of the above
  
4. Which of these are features of POST method  
a) no restrictions on data length, b) cannot be bookmarked, c) both (a) and (b), d) none of the above
  
- 5) The default Error handling in Angular is handled by \_\_\_\_\_class  
a) Errorstatic, b) ErrorHandler, c) Errormain, d) Error

**Lesson 6: Observables & rxjs operators (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to learn about : <ul style="list-style-type: none"><li>Basics of Observables &amp; Promises</li><li>Observable operator</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>Computer With Windows XP and above</li><li>Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

**Basics**

Observables provide support for passing messages between publishers and subscribers in your application. Observables definitely have significant benefits when compared to other techniques used for event handling, asynchronous programming, and handling multiple values.

**What makes observables special and lead the race?**

Observables are declarative—that is, you define a function for publishing values, but it is not executed until a consumer subscribes to it. The subscribed consumer then receives notifications until the function completes, or until they unsubscribe. At the same time, an observable can deliver multiple values of any type—literals, messages, or events, depending on the context. The API for receiving values is the same whether the values are delivered synchronously or asynchronously.

Since teardown and setup logics are handled simultaneously by an observable, the application code will only have to worry about subscribing to consume values as well as unsubscribing (once done). Whether the stream was keystrokes, an HTTP response, or an interval timer, the interface for listening to values and stopping listening is the same.

**Creating an observable instance**

As a publisher, you create an Observable instance that defines a subscriber function. This is the function that is executed when a consumer calls the subscribe() method.

The subscriber function defines how to obtain or generate values or messages to be published.

To execute the observable you have created and begin receiving notifications, you call its subscribe() method, passing an observer. This is a JavaScript object that defines the handlers for the notifications you receive. The subscribe() call returns a Subscription object that has an unsubscribe() method, which you call to stop receiving notifications.

Let's focus on an example that highlights the use of an observable for providing geo-location updates:

```
// Create an Observable that will start listening to geolocation updates
// when a consumer subscribes.
const locations = new Observable((observer) => {
  // Get the next and error callbacks. These will be passed in when
  // the consumer subscribes.
  const {next, error} = observer;
  let watchId;

  // Simple geolocation API check provides values to publish
  if ('geolocation' in navigator) {
    watchId = navigator.geolocation.watchPosition(next, error);
  } else {
    error('Geolocation not available');
  }

  // When the consumer unsubscribes, clean up data ready for next subscription
  return {unsubscribe() { navigator.geolocation.clearWatch(watchId); }};
});

// Call subscribe() to start listening for updates.
const locationsSubscription = locations.subscribe({
  next(position) { console.log('Current Position: ', position); },
  error(msg) { console.log('Error Getting Location: ', msg); }
});

// Stop listening for location after 10 seconds
setTimeout(() => { locationsSubscription.unsubscribe(); }, 10000);
```

## Promises

Promises in AngularJS are provided by the built-in \$q service. They provide a way to execute asynchronous functions in series by registering them with a promise object.

{info} Promises have made their way into native JavaScript as part of the ES6 specification. The angular \$q service provides an interface that closely resembles this new API so porting code to ES6 should be a breeze.

### The Setup:

```
<html>
  <head>
    <title>Promise fun</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.0-beta.5/angular.min.js"></script>
    <script src="app.js"></script>
  </head>
  <body ng-app="app">
  </body>
</html>
```

```
function getData($timeout, $q) {
  return function() {
    var defer = $q.defer()

    // simulated async function
    $timeout(function() {
    }, 2000)

    return defer.promise
  }
}

angular.module('app', [])
.factory('getData', getData)
.run(function(getData) {
  var promise = getData()
})
```

## Understanding the Deferred Object

A deferred object is simply an object that exposes a promise as well as the associated methods for resolving that promise. It is constructed using the `$q.deferred()` function and exposes three main methods: `resolve()`, `reject()`, and `notify()`. The associated promise object can be accessed via the `promise` property.

We can use a deferred object to signal the successful completion of our asynchronous method:

```
function getData($timeout, $q) {
  return function() {
    var defer = $q.defer()

    // simulated async function
    $timeout(function() {
      defer.resolve('data received!')
    }, 2000)

    return defer.promise
  }
}
```

Here we create a new deferred object, then return its promise property. We also execute our async function and after it completes, we resolve the deferred object. The parameter of the `resolve()` function will be passed to the callback function.

## The Promise Object

Now that we've obtained a Promise object (`defer.promise`), let's register a callback function that'll be executed after the async function completes.

Using the `then()` method, attach a callback function to the returned promise object that prints out the returned string.

```
.run(function(getData) {
  var promise = getData()
  .then(function(string) {
    console.log(string)
  })
})
```

Once you refresh the page, you should see "data received!" printed to the console after 2 seconds.

## rxjs operators

RxJS is mostly useful for its operators, even though the Observable is the foundation. Operators are the essential pieces that allow complex asynchronous code to be easily composed in a declarative manner.



Reactive programming is an asynchronous programming paradigm concerned with data streams and the propagation of change. RxJS (Reactive Extensions for JavaScript) is a library for reactive programming using observables that makes it easier to compose asynchronous or callback-based code.

RxJS provides an implementation of the Observable type, which is needed until the type becomes part of the language and until browsers support it. The library also provides utility functions for creating and working with observables. These utility functions can be used for:

- Converting existing code for async operations into observables
- Iterating through the values in a stream
- Mapping values to different types
- Filtering streams
- Composing multiple streams

RxJS offers a number of functions that can be used to create new observables. These functions can simplify the process of creating observables from things such as events, timers, promises, and so on.

## Practical Session

**Instructions: Create the code....using an editor....and save it with HTML extension.**

### 1. Write a program that creates an observable from promise and counter respectively.

#### Create an observable from a promise

```
import { from } from 'rxjs';

// Create an Observable out of a promise
const data = from(fetch('/api/endpoint'));
// Subscribe to begin listening for async result
data.subscribe({
  next(response) { console.log(response); },
  error(err) { console.error('Error: ' + err); },
  complete() { console.log('Completed'); }
});
```

#### Create an observable from a counter

```
import { interval } from 'rxjs';

// Create an Observable that will publish a value on an interval
const secondsCounter = interval(1000);
// Subscribe to begin publishing values
secondsCounter.subscribe(n =>
  console.log(`It's been ${n} seconds since subscribing!`));
```

## 2. Create a program that transmits data between components.

**Note:** After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

- Observables provide support for passing messages between publishers and subscribers in your application.
- Observables are declarative—that is, you define a function for publishing values, but it is not executed until a consumer subscribes to it.
- Promises in AngularJS are provided by the built-in \$q service. They provide a way to execute asynchronous functions in series by registering them with a promise object.
- A deferred object is simply an object that exposes a promise as well as the associated methods for resolving that promise.
- RxJS (Reactive Extensions for JavaScript) is a library for reactive programming using observables that makes it easier to compose asynchronous or callback-based code.

### Testing your skills

#### Lesson 6: Observables & rxjs operators

1. \_\_\_\_ provide support for passing messages between publishers and subscribers in your application

a) Observables, b) modifiers, c) promise, d) detector

2. Observables are \_\_\_\_

a) assertive, b) static, c) declarative, d) none of the above

3. Promises in AngularJS are provided by the built-in \_\_\_\_ service

a) \$s, b) \$q, c) \$r, d) \$rg

4. A deferred object is constructed using the \_\_\_\_ function

a) \$sq.deferred(), b) \$rt.def(), c) \$do.deferred(), d) \$q.deferred()

5. RxJS stands for \_\_\_\_

a) Reactive Program for Java Script, b) Reaction Program for Java Script, c) ReactXJS, d) None

**Lesson 7: Angular Animation (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to learn about : <ul style="list-style-type: none"><li>• Material Design Bootstrap with Angular</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

Angular Material is a UI component library for Angular JS developers. These components being reusable help in constructing attractive, consistent, and functional web pages and web applications while adhering to modern web design principles like browser portability, device independence, and graceful degradation. It helps in creating faster, beautiful, and responsive websites. It is inspired by the Google Material Design.

Following are a few salient features of Angular Material –

- In-built responsive designing.
- Standard CSS with minimal footprint.
- Includes new versions of common user interface controls such as buttons, check boxes, and text fields which are adapted to follow Material Design concepts.
- Includes enhanced and specialized features like cards, toolbar, speed dial, side nav, swipe, and so on.
- Cross-browser, and can be used to create reusable web components.

**Responsive Design**

- Angular Material has in-built responsive designing so that the website created using Angular Material will redesign itself as per the device size.
- Angular Material classes are created in such a way that the website can fit any screen size.
- The websites created using Angular Material are fully compatible with PC, tablets, and mobile devices.

**Extensible**

- Angular Material is by design very minimal and flat.
- It is designed considering the fact that it is much easier to add new CSS rules than to overwrite existing CSS rules.
- It supports shadows and bold colors.

- The colors and shades remain uniform across various platforms and devices.
- And most important of all, Angular Material is absolutely free to use.

## How to Use Angular Material?

There are two ways to use Angular Material –

Local Installation – You can download the Angular Material libraries using npm, jspm, or bower on your local machine and include it in your HTML code.

CDN Based Version – You can include the angular-material.min.css and angular-material.js files into your HTML code directly from the Content Delivery Network (CDN).

## Local Installation

Before we use the following npm command, we require the NodeJS installation on our system. The following command will be used for installing Angular Material libraries:

```
npm install angular-material
```

The above command will generate the following output –

```
angular-animate@1.5.2 node_modules\angular-animate
angular-aria@1.5.2 node_modules\angular-aria
angular-messages@1.5.2 node_modules\angular-messages
angular@1.5.2 node_modules\angular
angular-material@1.0.6 node_modules\angular-material
```

## Material Design Bootstrap with Angular

### Step 1: Install Angular Material, Angular CDK and Angular Animations

You can use either the npm or yarn command-line tool to install packages. Use whichever is appropriate for your project in the examples below.

#### NPM

```
npm install --save @angular/material @angular/cdk @angular/animations
```

#### Yarn

```
yarn add @angular/material @angular/cdk @angular/animations
```

### Alternative 1: Snapshot Build

A snapshot build with the latest changes from master is also available. Note that this snapshot build should not be considered stable and may break between releases.

#### NPM

```
npm install --save angular/material2-builds angular/cdk-builds angular/animations-builds
```

#### Yarn

```
yarn add angular/material2-builds angular/cdk-builds angular/animations-builds
```

### Alternative 2: Angular Devkit 6+

Using the Angular CLI `ng add` command will update your Angular project with the correct dependencies, perform configuration changes and execute initialization code.

```
ng add @angular/material
```

### Step 2: Configure animations

Once the animations package is installed, import `BrowserAnimationsModule` into your application to enable animations support.

```
import {BrowserAnimationsModule} from '@angular/platform-browser/animations';
```

```
@NgModule({  
  ...  
  imports: [BrowserAnimationsModule],  
  ...  
})  
export class PizzaPartyAppModule { }
```

Alternatively, you can disable animations by importing `NoopAnimationsModule`.

```
import {NoopAnimationsModule} from '@angular/platform-browser/animations';
```

```
@NgModule({  
  ...
```

```
imports: [NoopAnimationsModule],  
...  
})  
export class PizzaPartyAppModule { }
```

### Step 3: Import the component modules

Import the NgModule for each component you want to use:

```
import {MatButtonModule} from '@angular/material/button';  
import {MatCheckboxModule} from '@angular/material/checkbox';  
  
@NgModule({  
...  
imports: [MatButtonModule, MatCheckboxModule],  
...  
})  
export class PizzaPartyAppModule { }
```

Alternatively, you can create a separate NgModule that imports and then re-exports all of the Angular Material components that you will use in your application. By exporting them again, other modules can simply include your CustomMaterialModule wherever Material components are needed, and automatically get all of the exported Material modules. A good place for importing/exporting the application-wide Material modules is the SharedModule.

```
import {MatButtonModule} from '@angular/material/button';  
import {MatCheckboxModule} from '@angular/material/checkbox';  
  
@NgModule({  
imports: [MatButtonModule, MatCheckboxModule],  
exports: [MatButtonModule, MatCheckboxModule],  
})
```

```
export class MyOwnCustomMaterialModule { }
```

Whichever approach you use, be sure to import the Angular Material modules after Angular's BrowserModule, as the import order matters for NgModules.

#### Step 4: Include a theme

Including a theme is required to apply all of the core and theme styles to your application.

To get started with a prebuilt theme, include one of Angular Material's prebuilt themes globally in your application. If you're using the Angular CLI, you can add this to your styles.css:

```
@import "~@angular/material/prebuilt-themes/indigo-pink.css";
```

If you are not using the Angular CLI, you can include a prebuilt theme via a <link> element in your index.html.

#### Step 5: Gesture Support

Some components (mat-slide-toggle, mat-slider, matTooltip) rely on HammerJS for gestures. In order to get the full feature-set of these components, HammerJS must be loaded into the application.

You can add HammerJS to your application via npm, a CDN (such as the Google CDN), or served directly from your app.

To install via npm, use the following command:

##### NPM

```
npm install --save hammerjs
```

##### Yarn

```
yarn add hammerjs
```

After installing, import it on your app's entry point (e.g. src/main.ts).

```
import 'hammerjs';
```

#### Step 6 (Optional): Add Material Icons

If you want to use the mat-icon component with the official Material Design Icons, load the icon font in your index.html.

```
<link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
```

## Configuring SystemJS

If your project is using SystemJS for module loading, you will need to add @angular/material and @angular/cdk to the SystemJS configuration.

The @angular/cdk package is organized of multiple entry-points. Each of these entry-points must be registered individually with SystemJS.

Here is a example configuration where @angular/material, @angular/cdk/platform and @angular/cdk/a11y are used:

```
System.config({  
  // Existing configuration options  
  map: {  
    // ...  
    '@angular/material': 'npm:@angular/material/bundles/material.umd.js',  
  
    // CDK individual packages  
    '@angular/cdk/platform': 'npm:@angular/cdk/bundles/cdk-platform.umd.js',  
    '@angular/cdk/a11y': 'npm:@angular/cdk/bundles/cdk-a11y.umd.js',  
    // ...  
    'hammerjs': 'npm:hammerjs',  
  },  
  packages: {  
    //...  
    hammerjs: {main: './hammer.min.js', defaultExtension: 'js'}  
    //...  
  }  
});
```



## Practical Session

**Instructions:** Create the code....using an editor....and save it with HTML extension.

**1. From the above example, try to carryout angular configuration through the use of @angular/material, and @angular/cdk/platform**

**Note:** After saving the code, click the created HTML file to view the output.

## Reviewing the chapter

- Angular Material is a UI component library for Angular JS developers
- Angular Material has in-built responsive designing so that the website created using Angular Material will redesign itself as per the device size
- There are two ways to use angular material – local and CDN

## Testing your skills

1. It is inspired by the \_\_\_\_\_material design  
a) Opera, b) Netscape, c) Google, d) none of the above
2. Angular material results in creating what type of websites  
a) faster, b) responsive, c) both (a) and (b), d) none of the above
3. Common user interface controls included in Angular material are  
a) checkbox, b) text field, c) button, d) all of the above
4. Which of the following specialized features are parts of Angular Material  
a) speed dial, b) side nav, c) swipe, d) all of the above
5. Angular material supports  
a) shadow, b) light colors, c) both (a) and (b), d) none of the above

## Lesson 8: Authentication and Route Protection (120 minutes)

<b>Objective:</b> After completing this lesson you will be able to learn about : <ul style="list-style-type: none"> <li>• Working in SPA</li> <li>• Route protection</li> <li>• Route guard</li> <li>• Redirection</li> </ul>	<b>Materials Required:</b> <ul style="list-style-type: none"> <li>• Computer With Windows XP and above</li> <li>• Stable Internet connection</li> </ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

SPA stands for Single Page Application. Single page apps are becoming increasingly popular. Sites that mimic the single page app behavior are able to provide the feel of a phone/tablet application. With Angular, it becomes easier to create such applications.

### Working in SPA

To understand the SPA process in a better way, let's go through an example. In this example, we shall be focusing on developing a basic site featuring pages like home, about us, and contact.

### The goals of this project

- Developing a single page application
- No page refresh on page change
- Different data on each page

Although such a project can be carried out using JS and AJAX calls, the inclusion of Angular will definitely make the procedure a lot easier.

### Let's focus on the file structure

```
- script.js      <!-- stores all our angular code -->
- index.html    <!-- main layout -->
- pages         <!-- the pages that will be injected into the main layout -->
----- home.html
----- about.html
----- contact.html
```

Next, we need to focus on the HTML part. This is quite simple since we shall be using Bootstrap and Font Awesome. Open up your index.html file and we'll add a simple layout with a navigation bar:

```
<!-- index.html -->
<!DOCTYPE html>
<html>
<head>
<!-- SCROLLS -->
<!-- load bootstrap and fontawesome via CDN -->
<link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css" />
<link rel="stylesheet" href="//netdna.bootstrapcdn.com/font-awesome/4.0.0/css/font-awesome.css" />
<!-- SPELLS -->
<!-- load angular and angular route via CDN -->
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.25/angular.min.js"></script>
<script src="//ajax.googleapis.com/ajax/libs/angularjs/1.2.25/angular-route.js"></script>
<script src="script.js"></script>
</head>
<body>
<!-- HEADER AND NAVBAR -->
<header>
<nav class="navbar navbar-default">
<div class="container">
<div class="navbar-header">
<a class="navbar-brand" href="/">Angular Routing Example</a>
</div>
<ul class="nav navbar-nav navbar-right">
<li><a href="#"><i class="fa fa-home"></i> Home</a></li>
<li><a href="#about"><i class="fa fa-shield"></i> About</a></li>
<li><a href="#contact"><i class="fa fa-comment"></i> Contact</a></li>
</ul>
</div>
</nav>
</header>
```

```
<!-- MAIN CONTENT AND INJECTED VIEWS -->
<div id="main">
<!-- angular templating -->
<!-- this is where content will be injected -->
</div>
</body>
</html>
```

Kindly keep a note: For linking to pages, we'll use the #. We don't want the browser to think we are actually travelling to about.html or contact.html.

## Setting up the Angular Application

While the above steps serve as prerequisites, the major effort starts from here. This is where we shall be setting up the application. To do so, we first need to create the angular module and controller in JavaScript.

Let's do it in script.js.

```
// script.js
// create the module and name it scotchApp
var scotchApp = angular.module('scotchApp', []);
// create the controller and inject Angular's $scope
scotchApp.controller('mainController', function($scope) {
// create a message to display in our view
$scope.message = 'Everyone come and see how good I look!';
});
```

Let's add the module and controller to our HTML so that Angular knows how to bootstrap our application. To test that everything is working, we will also show the \$scope.message variable that we created.

```
<!-- index.html -->
<!DOCTYPE html>
```

```
<!-- define angular app -->
<html ng-app="scotchApp">
<head>
<!-- SCROLLS -->
<!-- load bootstrap and fontawesome via CDN -->
<link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css" />
<link rel="stylesheet" href="//netdna.bootstrapcdn.com/font-awesome/4.0.0/css/font-awesome.css" />
<!-- SPELLS -->
<!-- load angular via CDN -->
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.25/angular.min.js"></script>
<script src="//ajax.googleapis.com/ajax/libs/angularjs/1.2.25/angular-route.js"></script>
<script src="script.js"></script>
</head>
<!-- define angular controller -->
<body ng-controller="mainController">
...
<!-- MAIN CONTENT AND INJECTED VIEWS -->
<div id="main">
{{ message }}
<!-- angular templating -->
<!-- this is where content will be injected -->
</div>
```

Inside of our main div, we will now see the message that we created. Since we have our module and controller set up and we know that Angular is working properly, we will start working on using this layout to show the different pages.

## Injecting Pages into the Main Layout

ng-view is an Angular directive that will include the template of the current route (/home, /about, or /contact) in the main layout file. In plain words, it takes the file we want based on the route and injects it into our main layout (index.html).

We will add the ng-view code to our site in the div#main to tell Angular where to place our rendered pages.

```
<!-- index.html -->

...

<!-- MAIN CONTENT AND INJECTED VIEWS -->

<div id="main">

<!-- angular templating -->

<!-- this is where content will be injected -->

<div ng-view></div>

</div>

...
```

### Configure Routes and Views

Angular's routing capabilities will be required to make a SPA without any page refreshes

Let's look in our Angular file and add to our application. We will be using `$routeProvider` in Angular to handle our routing. This way, Angular will handle all of the magic required to go get a new file and inject it into our layout.

Keep a note of the fact that AngularJS 1.2 and Routing The `ngRoute` module is no longer included in Angular after version 1.1.6. You will need to call the module and add it to the head of your document to use it. This tutorial has been updated for AngularJS 1.2

```
// script.js

// create the module and name it scotchApp
// also include ngRoute for all our routing needs
var scotchApp = angular.module('scotchApp', ['ngRoute']);

// configure our routes
scotchApp.config(function($routeProvider) {
  $routeProvider

  // route for the home page
  .when('/', {
    templateUrl : 'pages/home.html',
    controller : 'mainController'
  })
})
```

```
// route for the about page
.when('/about', {
  templateUrl : 'pages/about.html',
  controller : 'aboutController'
})
// route for the contact page
.when('/contact', {
  templateUrl : 'pages/contact.html',
  controller : 'contactController'
});
});
// create the controller and inject Angular's $scope
scotchApp.controller('mainController', function($scope) {
  // create a message to display in our view
  $scope.message = 'Everyone come and see how good I look!';
});
scotchApp.controller('aboutController', function($scope) {
  $scope.message = 'Look! I am an about page.';
});
scotchApp.controller('contactController', function($scope) {
  $scope.message = 'Contact us! JK. This is just a demo.';
});
```

Now we have defined our routes with `$routeProvider`. As you can see by the configuration, you can specify the route, the template file to use, and even a controller. This way, each part of our application will use its own view and Angular controller.

**Clean URLs:** By default, Angular will throw a hash (#) into the URL. To get rid of this, we will need to use `$locationProvider` to enable the HTML History API. This will remove the hash and make pretty URLs. For more information: [Pretty URLs in AngularJS: Removing the #.](#)

Our home page will pull the `home.html` file. About and contact will pull their respective files. Now if we view our app, and click through the navigation, our content will change just how we wanted.

To finish off this tutorial, we just need to define the pages that will be injected. We will also have them each display a message from its respective controller.

```
<!-- home.html -->
<div class="jumbotron text-center">
<h1>Home Page</h1>
<p>{{ message }}</p>
</div>

<!-- about.html -->
<div class="jumbotron text-center">
<h1>About Page</h1>
<p>{{ message }}</p>
</div>

<!-- contact.html -->
<div class="jumbotron text-center">
<h1>Contact Page</h1>
<p>{{ message }}</p>
</div>
```

If you are working locally, Angular routing will only work if you have an environment set for it. Make sure you are using `http://localhost` or some sort of environment.

## Angular Router

The Angular Router enables navigation from one view to the next as users perform application tasks. It's a JavaScript router implementation that's designed to work with Angular and is packaged as `@angular/router`.

### Tasks being performed by the Angular Router

- Activating all required Angular components to compose a page when a user navigates to a certain URL
- Letting users navigate from one page to another without page reload
- Updating the browser's history so the user can use the back and forward buttons when navigating back and forth between pages.



- Redirecting a URL to another URL
- Resolving data before a page is displayed
- Running scripts when a page is activated or deactivated

## Route Protection

Angular comes with a number of baked-in features which are tremendously helpful for handling authentication. The most popular is probably its `HttpInterceptor` interface, but right next to it would be route guards. Let's take a look at what Angular's route guards are and how to use them to help with authentication in your Angular apps.

## Route Guards

Angular's route guards are interfaces which can tell the router whether or not it should allow navigation to a requested route. They make this decision by looking for a true or false return value from a class which implements the given guard interface.

There are five different types of guards and each of them is called in a particular sequence. The router's behavior is modified differently depending on which guard is used. The guards are:

1. `CanActivate`: Check if a user has access
2. `CanActivateChild`: Check if a user has access to any of the child routes
3. `CanDeactivate`: Can a user leave a page? For example, they haven't finished editing a post
4. `Resolve`: Grab data before the route is instantiated
5. `CanLoad`: Check to see if we can load the routes assets

## Redirection

When your application starts, it navigates to the empty route by default. We can configure the router to redirect to a named route by default:

```
export const routes: Routes = [  
  { path: '', redirectTo: 'component-one', pathMatch: 'full' },  
  { path: 'component-one', component: ComponentOne },  
  { path: 'component-two', component: ComponentTwo }  
];
```

The `pathMatch` property, which is required for redirects, tells the router how it should match the URL provided in order to redirect to the specified route. Since `pathMatch: full` is provided, the router will redirect to `component-one` if the entire URL matches the empty path (`''`).

When starting the application, it will now automatically navigate to the route for `component-one`.

## Practical Session

**Instructions: Create the code....using an editor....and save it with HTML extension.**

### 1. Create a program that will import { RouterModule} from '@angular/router

```
RouterModule.forRoot([\n  {\n    path: 'new-cmp',\n    component: NewCmpComponent\n  }\n])
```

### 2. Create a Custom Pipe in Angular

**Note: After saving the code, click the created HTML file to view the output.**

## Reviewing the chapter

- SPA stands for Single Page Application. Single page apps are becoming increasingly popular. Sites that mimic the single page app behavior are able to provide the feel of a phone/tablet application. With Angular, it becomes easier to create such applications.
- Angular's routing capabilities will be required to make a SPA without any page refreshes
- The Angular Router enables navigation from one view to the next as users perform application tasks.
- Angular's route guards are interfaces which can tell the router whether or not it should allow navigation to a requested route.

## Testing your skills

1. SPA stands for\_\_\_\_\_

a) Session Protocol Access, b) Same Page Address, c) Session Point Address, d) Single Page Application

2. \_\_\_\_\_ enables navigation from one view to the next as users perform application tasks
- a) session Storage, b) Angular Router, c) Interceptor, d) none of the above
3. Which of these are tasks performed by Angular Router
- a) Redirecting a URL to another URL, b) Running scripts when a page is activated, c) Running scripts when a page is deactivated, d) all of the above
4. Which route guard is used to check if a user has access
- a) CanActivate, b) CanLoad, c) Resolve, d) none of the above
5. The\_\_\_\_\_ property tells the router how it should match the URL provided in order to redirect to the specified route
- a) PathURL, b) PathAccess, c) PathMatch, d) PathRedirect

**Lesson 9: Deploying an Angular App (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to learn about : <ul style="list-style-type: none"><li>• JIT vs AOT Compilation</li><li>• Deployment</li><li>• Linting</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 120 minutes	<b>Practical Duration:</b> nil
<b>Total Duration:</b> 120 minutes	

**JIT vs AOT compilation**

The main differences between JIT (Just In Time) and AOT (Ahead Of Time) in Angular are:

- The time when the compilation takes place.
- JIT generates JavaScript however, AoT usually generates TypeScript.

**Flow of events with JIT (Just-in-Time Compilation)**

- Development of Angular application with TypeScript.
- Compilation of the application with tsc.
- Bundling.
- Minification.
- Deployment.

Once we've deployed the app and the user opens her browser, she will go through the following steps (without strict CSP):

- Download all the JavaScript assets.
- Angular bootstraps.
- Angular goes through the JiT compilation process, i.e. generation of JavaScript for each component in our application.
- The application gets rendered.

**Flow of events with AOT (Ahead-of-Time Compilation)**

In contrast, with AoT we get through the following steps:

- Development of Angular application with TypeScript.
- Compilation of the application with ngc.
- Performs compilation of the templates with the Angular compiler and generates (usually) TypeScript.

- Compilation of the TypeScript code to JavaScript.
- Bundling.
- Minification.
- Deployment.

Although the above process seems slightly more complicated the user goes only through the steps:

- Download all the assets.
- Angular bootstraps.
- The application gets rendered.

As you can see the third step is missing which means faster/better UX for AOT compilations.

## Deployment

To deploy angular apps in production, we simply need to run the below mentioned command for building the app:

```
ng build --prod --build-optimizer --base-href=/angular/
```

### A closer look at each command

- Ng - invokes angular
- build - prepares the build
- -prod : - specifies production mode, which means it can do ahead of time compilation
- -build-optimizer - creates smaller bundles of files.
- -base-href=/angular/ - This creates a base reference for built javascript files.

After build is successful, it will create a dist folder from which we can deploy angular app in production.

Copy all files in dist folder and copy to /angular/ folder in whatever production server you are using.

## Linting

To encourage coding best practices Angular CLI provides built-in linting. By default the app will look at the project's tslint.json for configuration. Linting can be executed by running the command `ng lint`.

## Reviewing the chapter

In this chapter, we have learned about the comparison between JIT and AOT. Additionally, we have also touched on deployment and basics of linting.

## Testing your skills

1. AOT stands for
  - a) Assisted object training, b) Assisted object tool, c) Ahead of time, d) An object tracking
  
2. The full form of JIT is
  - a) Java In Technique, b) Just In Time, c) Java Information Technology, d) None of the above
  
3. Which of the following are JIT- flow of events
  - a) Bundling, b) Minification, c) Deployment, d) All of the above
  
4. Which of the following are AOT- flow of events
  - a) Bundling, b) Minification, c) Deployment, d) All of the above
  
5. \_\_\_\_\_ deployment command prepares the build
  - a) build, b) Ng, c) build optimizer, d) prod

**Lesson 10: Modules (120 minutes)**

<b>Objective:</b> After completing this lesson you will be able to learn about : <ul style="list-style-type: none"><li>• Core modules</li><li>• Feature modules</li><li>• Modules and services</li></ul>	<b>Materials Required:</b> <ul style="list-style-type: none"><li>• Computer With Windows XP and above</li><li>• Stable Internet connection</li></ul>
<b>Self- Learning Duration:</b> 60 minutes	<b>Practical Duration:</b> 60 minutes
<b>Total Duration:</b> 120 minutes	

In Angular, a module basically refers to a place where you can group the components, directives, pipes, and services, which are related to the application.

In case you are developing a website, the header, footer, left, center and the right section become part of a module. Do keep in mind that the angular apps are modular and Angular has its own modularity system called NgModules. To define module, we can use the NgModule.

When you create a new project using the Angular –cli command, the ngmodule is created in the app.module.ts file by default and it looks as follows:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

Every Angular app has at least one NgModule class, the root module, which is conventionally named AppModule and resides in a file named app.module.ts. You launch your app by bootstrapping the root NgModule.

While a small application might have only one NgModule, most apps have many more feature modules. The root NgModule for an app is so named because it can include child NgModules in a hierarchy of any depth.

## Core Modules

The core module is a module that is only imported once in the AppModule and never again in the other modules. To ensure your module is only imported one time (if you are working on a team of sneaky members) , you can put this code in the class constructor that will prevent the module from being imported twice:

```
export class CoreModule {  
  constructor( @Optional() @SkipSelf() parentModule: CoreModule) {  
    if (parentModule) {  
      throw new Error('CoreModule has already been loaded. You should only import Core modules in the AppModule only.');    }  
  }  
}
```

So why do we need to import that famous module only once ? The reason behind this is that we want everything that's inside the core module to be a Singleton !!! And this is very important if you need your components/services to have only one instance. Some usage examples are the profile service or the header or footer components.

## Shared module

If compared to Core module, the Shared Module is imported in every feature module that needs some shared components.

The SharedModule is the perfect place for importing and exporting back your UI Modules or components that are used a lot in your application. This will make your code more readable and maintainable.

The SharedModule typically consists of common Angular specific modules (CommonModule, FormsModule, etc) as well as any components, services, etc that you want to reuse across your application.

Per the Angular Style Guide, the shared module should be created in the app/shared/ folder:

```
import { CommonModule } from '@angular/common';  
import { NgModule } from '@angular/core';  
import { FormsModule } from '@angular/forms';  
import { HttpClientModule } from '@angular/http';
```



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

@NgModule({
  imports: [CommonModule, FormsModule, HttpModule, RouterModule],
  declarations: [],
  exports: [CommonModule, FormsModule, HttpModule, RouterModule]
})
export class SharedModule {}
```

## Feature module

A feature module is an organizational best practice, as opposed to a concept of the core Angular API. A feature module delivers a cohesive set of functionality focused on a specific application need such as a user workflow, routing, or forms. While you can do everything within the root module, feature modules help you partition the app into focused areas. A feature module collaborates with the root module and with other modules through the services it provides and the components, directives, and pipes that it shares.

There are five general categories of feature modules which tend to fall into the following groups:

1. Domain feature modules
2. Routed feature modules
3. Routing modules
4. Service feature modules
5. Widget feature modules

## Modules and services

- Domain modules - These modules deliver a user experience dedicated to a particular application domain like editing a customer or placing an order. They typically have a top component that acts as the feature root and private, supporting sub-components descend from it. Domain feature modules consist mostly of declarations. Only the top component is exported.
- Routed modules – These are domain feature modules whose top components are the targets of router navigation routes. All lazy-loaded modules are routed feature modules by definition. Routed

feature modules don't export anything because their components never appear in the template of an external component. A lazy-loaded routed feature module should not be imported by any module. Doing so would trigger an eager load, defeating the purpose of lazy loading.

- Routing modules - These modules provide routing configuration for another module and separates routing concerns from its companion module. These modules define routes, adds router configuration to the module's imports, and adds guard and resolver service providers to the module's providers. A routing module does not have its own declarations.
- Service modules - These modules provide utility services such as data access and messaging. Ideally, they consist entirely of providers and have no declarations. Angular's HttpClientModule is a good example of a service module. The root AppModule is the only module that should import service modules.
- Widget modules - These modules make components, directives, and pipes available to external modules. Many third-party UI component libraries are widget modules. A widget module should consist entirely of declarations, most of them exported. Such a module rarely has providers.

## Practical Session

**Instructions: Create the code....using an editor....and save it with HTML extension.**

**1. Create a program where you need to add a controller to your application, and refer to the controller with the ng-controller directive.**

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>

<script>
```

```
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
$scope.firstName = "John";
$scope.lastName = "Doe";
});
</script>

</body>
</html>
```

## 2. Create a directive in Angular

**Note:** After saving the code, click the created HTML file to view the output.

### Reviewing the chapter

- A module basically refers to a place where you can group the components, directives, pipes, and services, which are related to the application.
- Every Angular app has at least one NgModule class, the root module, which is conventionally named AppModule and resides in a file named app.module.ts.
- The core module is a module that is only imported once in the AppModule and never again in the other modules.
- A feature module is an organizational best practice, as opposed to a concept of the core Angular API.

### Testing your skills

1. \_\_\_\_\_ is a module that is only imported once in the AppModule and never again in the other modules.
- a) Core module, b) Shared module, c) Feature module, d) None of the above

2. Which of the following are parts of feature modules

a) Domain, b) Routed, c) Routing, d) All of the above

3. \_\_\_\_\_ have top components are the targets of router navigation routes

a) Domain, b) Routing, c) Service, d) Routed

4. The \_\_\_\_\_ is the only module that should import service modules

a) Widget, b) root App, c) lazy loaded, d) none of the above

5. In the \_\_\_\_\_ module, only the top component is exported

a) Widget, b) Routed, c) Domain, d) None of the above

**Lesson 11: App State Management with Redux (120 minutes)****Objective:** After completing this lesson you will be able to learn about :

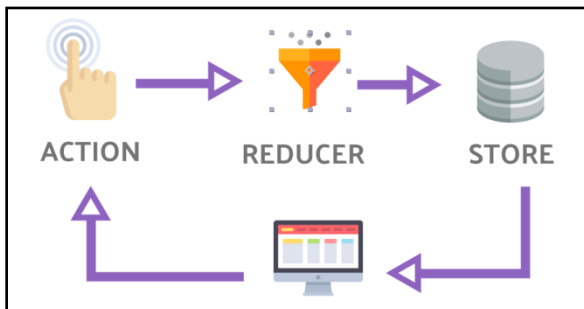
- Redux basics
- Actions
- Reducers

**Materials Required:**

- Computer With Windows XP and above
- Stable Internet connection

**Self- Learning Duration:** 120 minutes**Practical Duration:** nil**Total Duration:** 120 minutes**Understanding Redux**

Redux is a pattern/library from the React world that has inspired popular Angular tools like NgRx and NGXS. The purpose of redux is to make application data more predictable by creating a one-way data flow.



Looking at the premise, it is quite simple. The application data being used is a single immutable object that can only be changed by dispatching actions.

Redux can also be described as a simple state management engine for Javascript. It must be noted that this engine is not tied directly to React or any other view technology. But that said, since Redux is extensible and single-purpose, it suits the environment perfectly for React projects.

Redux has a simple API, composed of a State Store, Actions, Action Creators, and Reducers.

**Action**

An action is an event processed by Redux. It has a type, and a payload of request data. An 'action creator' is a function that creates the data for an action. It assembles the action with a type and with the appropriate payload so that the caller of the API doesn't need to know the internal action structure. Actions are created by action creators and dispatched by the state store.

## Reducer

A reducer is a function that takes two parameters, the existing state and an action request. The reducer function uses the action type and payload to determine what work to perform. Ultimately the reducer either returns the existing state unchanged, or a new state based on activities requested for that action type.

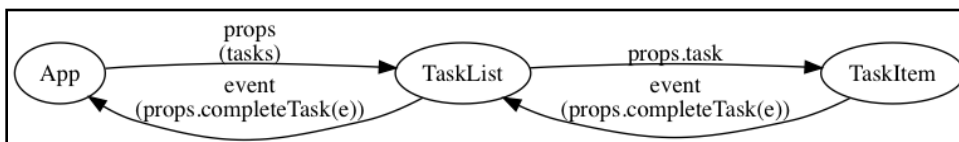
## React Projects

React projects break data up into two categories:

- State: read-write data that lives within a component
- Props: read-only data sent to a component, may be updated later by the sending component (a parent)

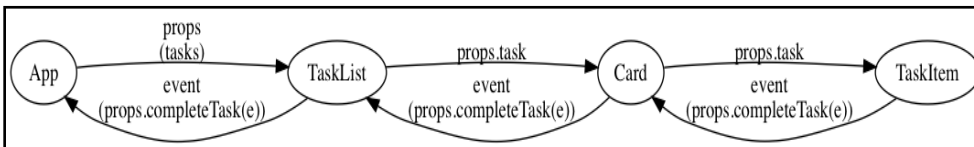
Anyone who has spent significant time in a React application realizes that, as the component graph gets bigger, the connections between the deeper components in the tree and their parents can become a chore to manage.

Component connections like this become common:



*(passing props, the hard way through other components)*

Now, if the design needs to involve containers for each task for layout purposes, the image should be a bit more complicated. Have a look below:



Well, you can consider using something like `{ this.props.children }` within the container. This will help in embedding the child, which as a result, can help reduce the over-passing of properties in decorative components like cards and panels. Have a look at the example below:

```
// in TaskList.js
render() {
  const cards = this.props.tasks.map(t => {
    return (
      <Card>
        <Task key={t.id} task={t} />           <1>
      </Card>
    );
  });
}
// in Card.js
render() {
  <div className="card">
    { this.props.children }                  <2>
  </div>
}
```

1. A nested component in your template
2. Emits all nested child components (in our case, the Task)

## Reviewing the chapter

- Redux is a pattern/library from the React world that has inspired popular Angular tools like NgRx and NGXS.
- The purpose of redux is to make application data more predictable by creating a one-way data flow.
- Redux has a simple API, composed of a State Store, Actions, Action Creators, and Reducers.
- An 'action creator' is a function that creates the data for an action.
- A reducer is a function that takes two parameters, the existing state and an action request.

**Testing your skills**

1. Which of the following tools have been inspired by REDUX

a) NgRx, b) RGx, c) NRx, d) All of the above

2. Redux comprises of

a) State stores, b) Actions, c) Action creators, d) All of the above

3. \_\_\_\_\_ is a function that takes two parameters

a) Action, b) Reducer c) Both (a) and (b), d) None of the above

4. React projects break up into \_\_\_\_ number of categories

a) 2, b) 3, c) 4, d) 5

5. \_\_\_\_\_ read-only data sent to a component

a) State, b) Square, c) Prop, d) None of the above