|  |
| --- |
|  |
| Angular 8 |
|  |
| What is Angular 8? |

**TechBrain Express**

November 11, 2019

Authored by: Ashu

Introduction to Angular 8

What is Angular 8?

# • Explain the characteristics of a Single Page Application (SPA)

# • Describe the Single Page Application Lifecycle

# • Demonstrate usage of JavaScript objects

# • Demonstrate usage of JavaScript Object Notation (JSON)

# • Explain the need and benefits of Angular

# • Describe the difference between AngularJS and Angular

# Use Case: Traditional Approach of Web Experience

* the constant full page reloads
* also due to the network back and forth trips to the server to fetch all that HTML.

# What is SPA?

A single-page application (SPA) is a website design approach where each new page's content is served not from loading new HTML pages but generated dynamically through JavaScript's ability to manipulate the DOM elements on the existing page itself.

**Traditional Approach:** An index.html page might link to other HTML pages on the server that the browser will download and display from scratch.

**SPA Approach:** Allows the user to continue consuming and interacting with the page while new elements are being updated or fetched, and can result in much faster interactions and content reloading.

* The application is able to dynamically fetch content from the server through [AJAX requests](https://en.wikipedia.org/wiki/Ajax_(programming)) or [websockets](https://en.wikipedia.org/wiki/Websockets).

# Advantages of SPA?

* much-improved experience to the user
* The application will feel faster because less bandwidth is being used, and no full page refreshes are occurring as the user navigates through the application
* The application will be much easier to deploy in production, at least certainly the client part: all we need is a static server to serve a minimum of 3 files: our single page index.html, a CSS bundle, and a Javascript bundle.
* The frontend part of the application is very simple to version in production, allowing for simplified deployment and rollbacks to previous version of the frontend if needed.
* ideal as a base for future mobile app development.

# How SPA Works?

After the startup, only data gets sent over the wire as a JSON payload or some other format. But no HTML or CSS gets sent anymore over the wire after the application is running.

The key point to understand how single page applications work is the following: instead of converting data to HTML on the server and then send it over the wire, in a SPA we have now moved that conversion process from the server to the client. The conversion happens last second on the client side, which allow us to give a much improved user experience to the user.

# Disadvantages of SPA?

* Poor SEO optimization.  If a project requires effective SEO, on the other hand, then you should use a multi-page application.
* **Browser history**
  + An SPA doesn’t save visitors’ jumps between states. This means that when users click the back button, they won’t go back. A browser only takes users to the previous page, not to the previous state in an app.
  + To solve this problem, there’s an HTML5 History API with which developers can equip their SPA frameworks. The History API offers developers access to browser navigation history via JavaScript.
* **Security issues**
  + Single-page apps are less immune to [cross-site scripting (XSS) attacks](https://security.stackexchange.com/questions/41239/do-spa-applications-have-different-security-considerations-than-html5-sites) than are multi-page apps. Using XSS, hackers can inject client-side scripts into web applications.
  + One security issue is the exposure of sensitive data. If developers aren’t careful about what data is contained in the initial page load, they can easily send data that shouldn’t be exposed to all users. The whole of an SPA isn’t generally visible in the browser, which can provide a false sense of security.
  + One more reason that SPAs can be insecure is missing access control at the functional level. Since developers move features and logic off the server and out to the client, it’s really easy to provide a client with access to functions that they shouldn’t be permitted to use.

# MPA?

|  |  |
| --- | --- |
| **Advantage** | **Disadvantages** |
| * + SEO | * + Slow speed and performance |
| * + Unlimited scalability | * + More time for development |
| * + Insights from Google Analytic | * + Maintenance and updates |

# Describe the Single Page Application Lifecycle

# 

# Demonstrate usage of JavaScript objects

An object is an unordered list of primitive data types (and sometimes reference data types) that is stored as a series of name-value pairs. Each item in the list is called a *property* (functions are called *methods*).

var myFirstObject = {firstName: "Richard", favoriteAuthor: "Conrad"};

var person = {name: "Kobe"};

var anotherPerson = person;

person.name = "Bryant";

console.log(anotherPerson.name); // Bryant

console.log(person.name); // Bryant

**Ways to create JS Object?**

1. **Object Literals**  
   The most common and, indeed, the easiest way to create objects is with the object literal described here:

// This is an empty object initialized using the object literal notation

var myBooks = {};

// This is an object with 4 items, again using object literal

var mango = {

color: "yellow",

shape: "round",

sweetness: 8,

howSweetAmI: function () {

console.log("Hmm Hmm Good");

}

}

1. **Object Constructor**  
    The second most common way to create objects is with Object constructor. A constructor is a function used for initializing new objects, and you use the new keyword to call the constructor.

var mango = new Object ();

mango.color = "yellow";

mango.shape= "round";

mango.sweetness = 8;

mango.howSweetAmI = function () {

console.log("Hmm Hmm Good");

}

**Ways to access JS Object?**

1. **Dot Notation**

We have been using dot notation so far in the examples above, here is another example again:

var book = {title: "Ways to Go", pages: 280, bookMark1:"Page 20"};

// To access the properties of the book object with dot notation, you do this:

console.log ( book.title); // Ways to Go

console.log ( book.pages); // 280

1. **Bracket Notation**

// To access the properties of the book object with bracket notation, you do this:

console.log ( book["title"]); //Ways to Go

console.log ( book["pages"]); // 280

//Or, in case you have the property name in a variable:

var bookTitle = "title";

console.log ( book[bookTitle]); // Ways to Go

console.log (book["bookMark" + 1]); // Page 20

**Own and Inherited Properties**

// Create a new school object with a property name schoolName

var school = {schoolName:"MIT"};

// Prints true because schoolName is an own property on the school object

console.log("schoolName" in school); // true

// Prints false because we did not define a schoolType property on the school object, and neither did the object inherit a schoolType property from its prototype object Object.prototype.

console.log("schoolType" in school); // false

// Prints true because the school object inherited the toString method from Object.prototype.

console.log("toString" in school); // true

**hasOwnProperties**

// Create a new school object with a property name schoolName

var school = {schoolName:"MIT"};

// Prints true because schoolName is an own property on the school object

console.log(school.hasOwnProperty ("schoolName")); // true

// Prints false because the school object inherited the toString method from Object.prototype, therefore toString is not an own property of the school object.

console.log(school.hasOwnProperty ("toString")); // false

# Accessing and Enumerating Properties on Objects

// Create a new school object with 3 own properties: schoolName, schoolAccredited, and schoolLocation.

var school = {schoolName:"MIT", schoolAccredited: true, schoolLocation:"Massachusetts"};

//Use of the for/in loop to access the properties in the school object

for (var eachItem in school) {

console.log(eachItem); // Prints schoolName, schoolAccredited, schoolLocation

}

**Accessing Inherited Properties**  
Properties inherited from Object.prototype are not enumerable, so the for/in loop does not show them. However, inherited properties that are enumerable are revealed in the for/in loop iteration.  
For example:

//Use of the for/in loop to access the properties in the school object

for (var eachItem in school) {

console.log(eachItem); // Prints schoolName, schoolAccredited, schoolLocation

}

// Create a new HigherLearning function that the school object will inherit from.

/\* SIDE NOTE: As Wilson (an astute reader) correctly pointed out in the comments below, the educationLevel property is not actually inherited by objects that use the HigherLearning constructor; instead, the educationLevel property is created as a new property on each object that uses the HigherLearning constructor. The reason the property is not inherited is because we use of the "this" keyword to define the property.

\*/

function HigherLearning () {

this.educationLevel = "University";

}

// Implement inheritance with the HigherLearning constructor

var school = new HigherLearning ();

school.schoolName = "MIT";

school.schoolAccredited = true;

school.schoolLocation = "Massachusetts";

//Use of the for/in loop to access the properties in the school object

for (var eachItem in school) {

console.log(eachItem); // Prints educationLevel, schoolName, schoolAccredited, and schoolLocation

}

# Deleting Properties of an Object

var christmasList = {mike:"Book", jason:"sweater" }

delete christmasList.mike; // deletes the mike property

for (var people in christmasList) {

console.log(people);

}

// Prints only jason

// The mike property was deleted

delete christmasList.toString; // returns true, but toString not deleted because it is an inherited method

// Here we call the toString method and it works just fine—wasn’t deleted

christmasList.toString(); //"[object Object]"

// You can delete a property of an instance if the property is an own property of that instance. For example, we can delete the educationLevel property from the school's object we created above because the educationLevel property is defined on the instance: we used the "this" keyword to define the property when we declare the HigherLearning function. We did not define the educationLevel property on the HigherLearning function's prototype.

console.log(school.hasOwnProperty("educationLevel")); true

// educationLevel is an own property on school, so we can delete it

delete school.educationLevel; true

// The educationLevel property was deleted from the school instance

console.log(school.educationLevel); undefined

// But the educationLevel property is still on the HigherLearning function

var newSchool = new HigherLearning ();

console.log(newSchool.educationLevel); // University

// If we had defined a property on the HigherLearning function's prototype, such as this educationLevel2 property:

HigherLearning.prototype.educationLevel2 = "University 2";

// Then the educationLevel2 property on the instances of HigherLearning would not be own property.

// The educationLevel2 property is not an own property on the school instance

console.log(school.hasOwnProperty("educationLevel2")); false

console.log(school.educationLevel2); // University 2

// Let's try to delete the inherited educationLevel2 property

delete school.educationLevel2; true (always returns true, as noted earlier)

// The inherited educationLevel2 property was not deleted

console.log(school.educationLevel2); University 2

# Demonstrate usage of JavaScript Object Notation (JSON)

## Use of JSON

* It is used while writing JavaScript based applications that includes browser extensions and websites.
* JSON format is used for serializing and transmitting structured data over network connection.
* It is primarily used to transmit data between a server and web applications.
* Web services and APIs use JSON format to provide public data.
* It can be used with modern programming languages.

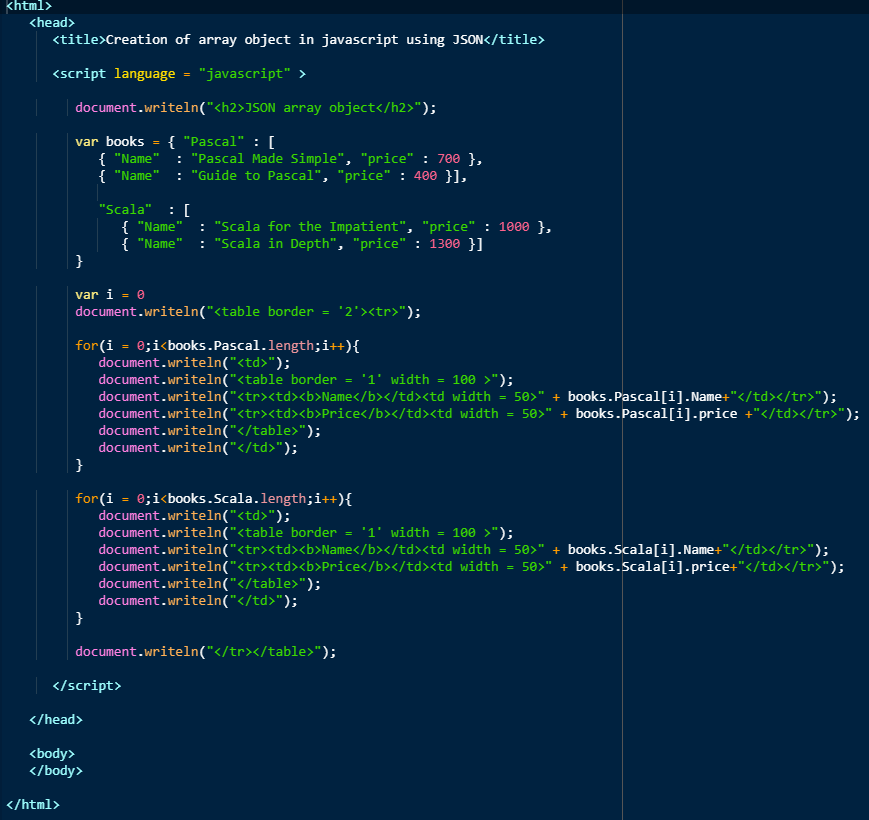
## Characteristics of JSON

* JSON is easy to read and write.
* It is a lightweight text-based interchange format.
* JSON is language independent

## Create an Object

* var JSONObj = {};
* var JSONObj = new Object();
* var JSONObj = { "bookname ":"VB BLACK BOOK", "price":500 };





|  |  |
| --- | --- |
|  |  |

### Exchange Data

|  |  |
| --- | --- |
| **Send Data** | **Receive Data** |
| To transfer your objects via HTTP or to otherwise convert it to a string, you will need to serialize it (convert it to a string); you can use the JSON.stringify function to serialize your objects. – **JSON.stringify()** | To Deserialize your object (convert it to an object from a string) – **JSON.parse()** |
|  |  |

# What is Angular8?

1. Client-side TypeScript based framework which is used to create dynamic web applications

2. Angular is a reusable UI component helps us constructing attractive, consistent, and functional web pages and web application

3. Makes us able to create an attractive Single Page Applications (SPAs).

“A single page application is a web application or a website which provides a fluid, reactive, and fast application same as a desktop application.

It contains menu, buttons, and blocks on a single page and when a user clicks on them,

it dynamically rewrites the current page without loading new pages from the server so that its speed is fast.”

4. Entirely based on component and consist of some tree structures with parent and child component.

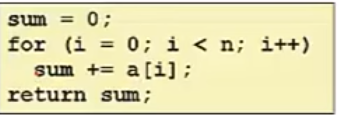
Angular 8 classes are created in such a way that the web page can fit in any screen size so that they are fully compatible with mobiles,

tablets, laptops, and large systems.

# Why Angular ?

1. Reusable code
2. TypeScript 3.4
3. Web Workers - Web Workers facilitates you to run the CPU intensive computations in the background thread, freeing the main thread to update the user interface. Web workers can also be helpful, if your application is unresponsive while processing data.
4. New Render Engine for Angular 8 -Ivy
   1. Ivy is going to be smaller, simpler and faster as quoted by angular team.
   2. Follows locality principal - Program tend to use the data and instruction with addresses near or equal to those they have used recently.
      1. Temporal locality- - referenced item likely ot reference again 
      2. . Spatial locality - Item with nearby address tend to referenced close together in time



Example:  Temporal Data - sum referenced in each iteration

Spatial Data - accessed sequentially

This leads to faster compilation and simplification of process.

Render2 Vs Ivy

|  |  |
| --- | --- |
| Render2 | Ivy |
|  |  |
| Does static analysis of all code and generate a file called *metadata.json* that could be used to generate the compilation instructions. | No metadata.json |

1. Support Bazel –

Bazel is a build tool developed and massively used by Google, as it can build pretty much any language. The Angular framework itself is built with Bazel.

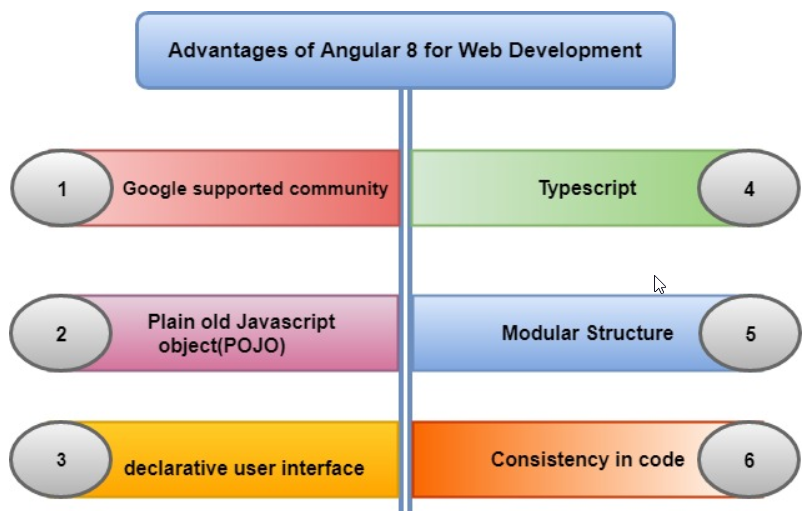
The key advantages of Bazel are:

* + - the possibility of building your backends and frontends with the same tool
    - the incremental build and tests
    - the possibility to have remote builds (and cache) on a build farm

1. Lazy Loading:

Lazy loading is based on the concepts of Angular Routing and it helps bring down the size of large files by lazily loading the data that are required.

# Benefits of Angular 8?



1. **Google Supported Community:**Angular comes with Google’s long-term support (LTS). The Google team is very confident about Angular’s stability; also, Google apps use an angular framework.
2. **Plain Old JavaScript Object (POJO**): It does not require any getter and setter function. It used every object as an everyday old JavaScript object. It provides JavaScript functionality to enable manipulation of an object such as adding properties or removing properties from the purpose.
3. **Declarative User Interface:**Angular uses HTML to define the view part of an Application, which is a complex language. Html is a declarative language too. We don’t worry about the flow of the program when it loads define what we want as per application requirement, and angular will take care of rest things.
4. **Typescript:** It is written in Typescript, which is a superset of JavaScript. It promotes high security. If we have created proper map files during build time, then you can easily debug typescript code in an editor or a browser.
5. **Modular Structure:**Angular organizes code into modules whether it is components, directives, pipes, or services. It makes the organizing of functionality easy and straightforward by separating the code. It also offers lazy loading.
6. **Consistency in code:**It maximizes the readability of the code. For any new developer, it is an easy task to go through the project because of its code consistency features.

# What is nodeJS?

Node.js = Runtime Environment + JavaScript Library

* Node.js is an open-source, cross-platform runtime environment used for the development of
* server-side web applications. Node.js applications are written in JavaScript and can be run on a wide variety of operating systems.
* Node.js is based on an event-driven architecture and a non-blocking Input/Output API that is designed to optimize an application's throughput and scalability for real-time web applications.

## Feature?

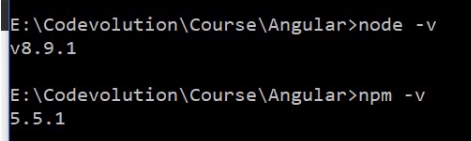
* **Asynchronous and Event Driven** − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
* **Very Fast** − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
* **Single Threaded but Highly Scalable** − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
* **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.
* **License** − Node.js is released under the [MIT license](https://raw.githubusercontent.com/joyent/node/v0.12.0/LICENSE).

## Where to use Nodejs?

# Installing and using Angular 8

## Verify the existing installation

1. Check version of NodeJS > node -v
2. Check Angular Version> npm -v



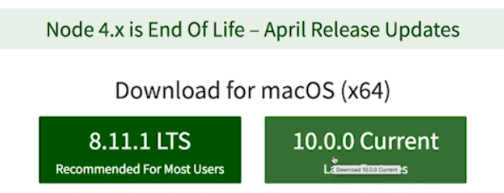
1. Check path of npm > npm config get prefix



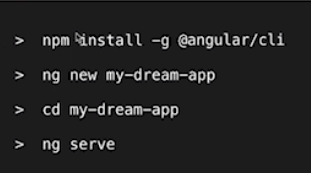
1. Check Angular version> **ng --version**
2. To Update Angular : ng update @angular/cli @angular/core

## Install new software

1. Install Node.js - Node bundles and Optimize our project and uses npm to mange dependencies.



##### Install Angular CLI



@latest will install latest version of angular

##### Test on UI

**To run the angular application >** ng serve



# Architecture / Building Blocks of an Angular 8  Application

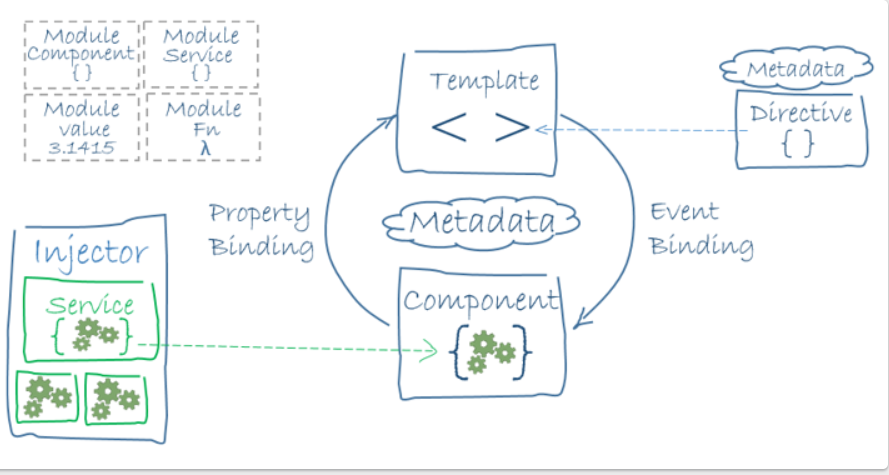
1. It is for building client applications in HTML and TypeScript.

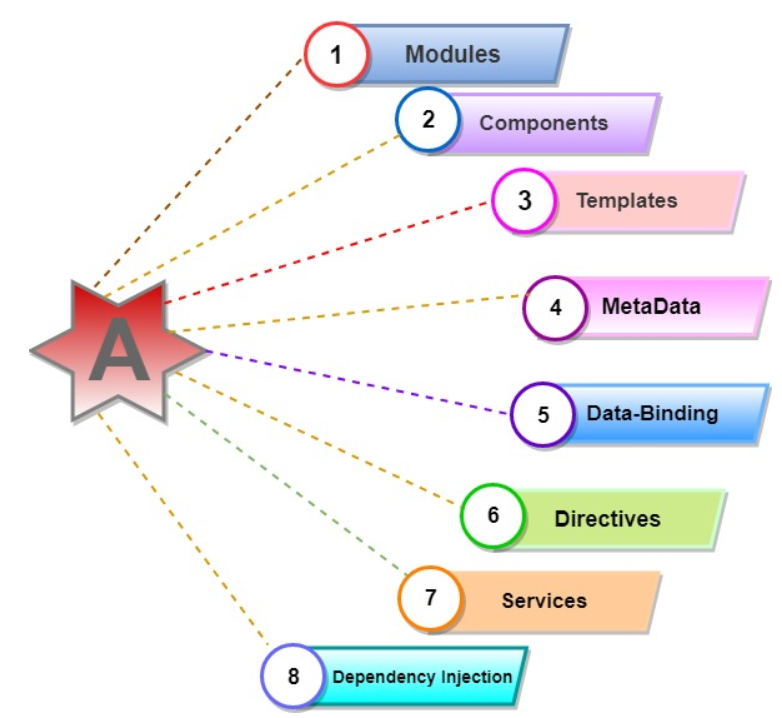
## Basic Building Block –

1. The basic building blocks of an Angular application are ***NgModules*,** which provide a compilation context for *components*.
2. Collect related code into functional sets; an Angular app is defined by a set of NgModules
3. An app always has at least a *root module* that enables bootstrapping.

###### Component

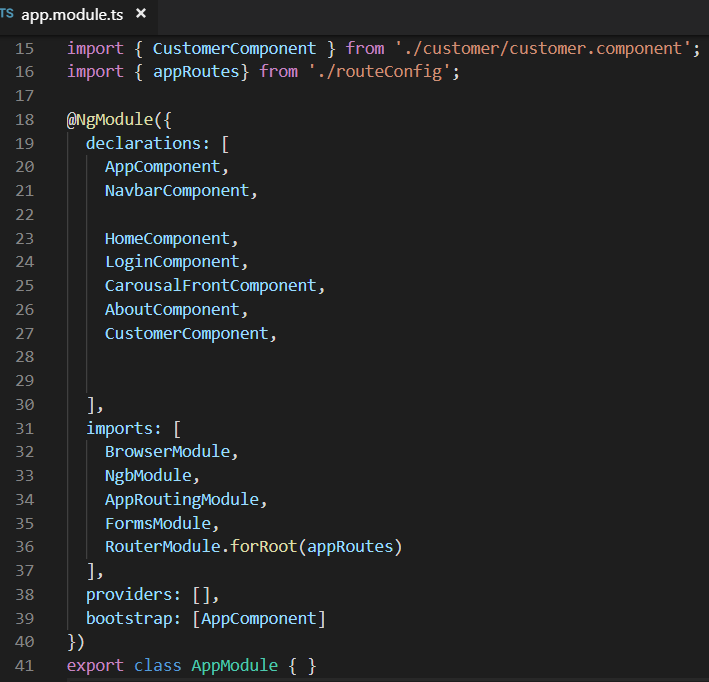
1. Defines **screen elements(view)** to choose among and modify according to your program logic and data.
2. **Uses Services -** provide specific functionality not directly related to views.
3. Service providers can be *injected* into components as *dependencies*, making your code modular, reusable, and efficient.
4. Both components and services are simply classes, with *decorators* that mark their type and provide metadata that tells Angular how to use them.
   1. The metadata for a component class associates it with a template that defines a view. A template combines ordinary HTML with Angular directives and binding markup that allow Angular to modify the HTML before rendering it for display.
   2. The metadata for a service class provides the information Angular needs to make it available to components through dependency injection (DI).





### Modules:

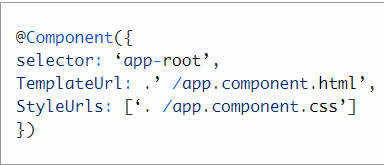
* 1. NgModules are containers for a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities. They can contain components, service providers, and other code files whose scope is defined by the containing NgModule.
  2. They can import functionality that is exported from other NgModules, and export selected functionality for use by other NgModules. Eg: if we want to use route service in our app, we can import the Routing Ng module.
  3. Every Angular app has at least one NgModule class, [the *root module*](https://angular.io/guide/bootstrapping), which is conventionally named AppModule



ngModule metadata: he @[NgModule](https://angular.io/api/core/NgModule)() decorator is a function that takes a single metadata object, whose properties describe the module.

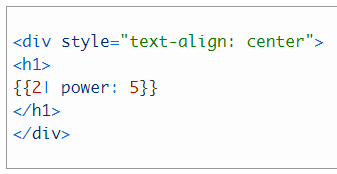
* declarations: The [components](https://angular.io/guide/architecture-components), *directives*, and *pipes* that belong to this NgModule.
* exports: The subset of declarations that should be visible and usable in the *component templates* of other NgModules.
* imports: Other modules whose exported classes are needed by component templates declared in *this* NgModule.
* providers: *Creators* of [services](https://angular.io/guide/architecture-services) that this NgModule contributes to the global collection of services; they become accessible in all parts of the app. (You can also specify providers at the component level, which is often preferred.)
* bootstrap: The main application view, called the *root component*, which hosts all other app views. Only the *root NgModule* should set the bootstrap property.
  1. NgModules provide a *compilation context* for their components. A root NgModule always has a root component that is created during bootstrap, but any NgModule can include any number of additional components, which can be loaded through the router or created through the template.

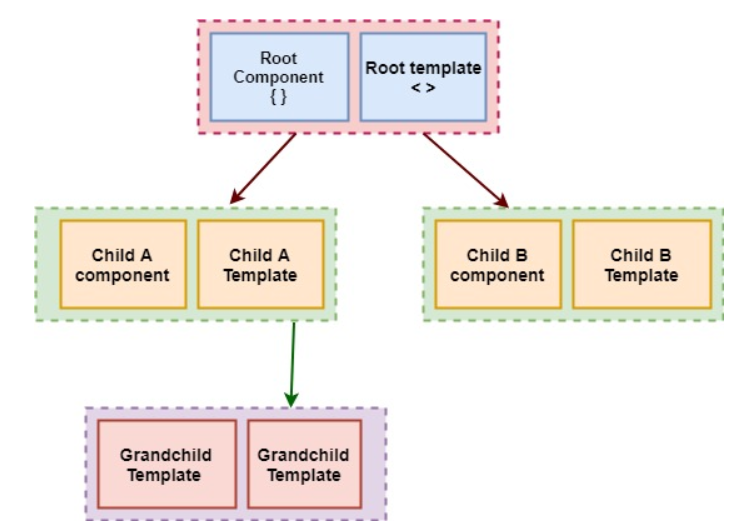
### Components

* Every angular project has at least 1 component, the root component, and the root components connect the component with a page Document Object Module (DOM). Each component defines a class which contains data, application, logic, and it is binding with the HTML template.
* The @[Component](https://angular.io/api/core/Component)() decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.
* 

### Templates

* The angular template integrates the HTML with Angular mark-up that can modify HTML elements before they are displayed. It provides program logic, and binding mark-up connects to your application data and the DOM.





### MetaData

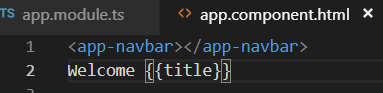
* **Decorators** are the metadata in Angular. It is used to enhance the class so it can configure the expected behavior of a class. Developers are the core concept of when developing with Angular.
* User can use metadata in a class **to tell Angular app that app component is a component**. Metadata can attach to the Typescript through the decorator.

### DataBinding

Angular allows communication between a component and a DOM. Making it very easy to define interactive application without pulling and pushing the data.

* ***Event binding*** lets your app respond to user input in the target environment by updating your application data.
* ***Property binding***lets you interpolate values that are computed from your application data into the HTML.

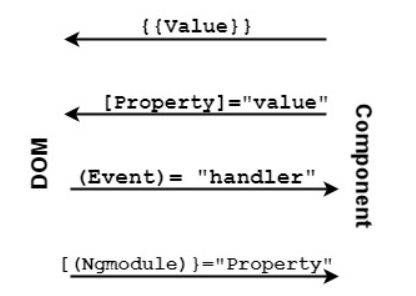
**Interpolation**: {{value}}: Interpolation sums the value of the property to the component.



### Property binding: [property]=”value”

### A value has been passed from a component to a special property, with property binding which can be a simple html attribute.

<input type="text" class="form-control" [value]=myuser.userName id="uname" name="uname" >



### Directives

* Directives used for expanding the functionality of HTML elements.
* 3 Types:

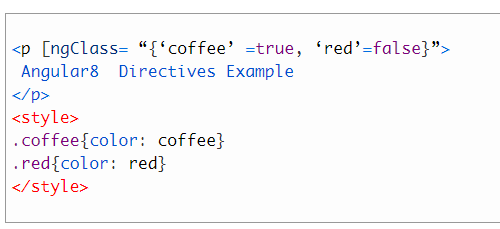
1. Structural Directive <https://angular.io/guide/structural-directives>
   * + Structural directives are responsible for change the DOM layout by adding and removing DOM elements.
     + They shape or reshape the DOM's *structure*, typically by adding, removing, or manipulating elements.
     + Structural directives are easy to recognize. An asterisk (\*) precedes the directive attribute name as in this example.

<div \*[ngIf](https://angular.io/api/common/NgIf)="hero" class="name">{{hero.name}}</div>



1. Attribute Directive <https://angular.io/guide/attribute-directives>

To change the appearance or behavior of an element, component, or another directive. EG: ngClass, ngStyle



1. Component Directive

* Directives with a template.

### Services

* For data or logic that isn't associated with a specific view, and that you want to share across components, you create a service class.
* A service class definition is immediately preceded by the @[Injectable](https://angular.io/api/core/Injectable)() decorator.  The decorator provides the metadata that allows other providers to be injected as dependencies into your class.
* Used to reuse the code.
* Angular distinguishes an element from service to increase modularity and reusability.

### Dependency Injection

* Dependency injection (DI) lets you keep your component classes lean and efficient. They don't fetch data from the server, validate user input, or log directly to the console; they delegate such tasks to services.
* It is a design pattern for efficiency and modularity. DI is wired into Angular into an Angular framework and used everywhere to provide new component with new services. Dependency injection does not fetch data from a server, validate the user input, or log directly to console; instead, they delegate such tasks to the service.

### Routing

* The Angular [Router](https://angular.io/api/router/Router) NgModule provides a service that lets you define a navigation path among the different application states and view hierarchies in your app.
* The router maps URL-like paths to views instead of pages. When a user performs an action, such as clicking a link, that would load a new page in the browser, the router intercepts the browser's behavior, and shows or hides view hierarchies.
* The router interprets a link URL according to your app's view navigation rules and data state. You can navigate to new views when the user clicks a button or selects from a drop box, or in response to some other stimulus from any source. The router logs activity in the browser's history, so the back and forward buttons work as well.

##### How component and template define an Angular view?

* A decorator on a component class adds the metadata, including a pointer to the associated template.
* Directives and binding markup in a component's template modify views based on program data and logic.

##### How Services are provided to component?

Dependency Injection

# Difference between AngularJs & 8

|  |  |
| --- | --- |
| Angular JS | Angular 8 |
| JavaScript-based open-source front end web development. | Typescript based full-stack web application framework. |
| Uses the concept of scope or controller. | Uses the hierarchy of components in place of scope and controllers. |
| Not support dynamic loading of the page. | Supports dynamic loading of the page. |
| Simple syntax and used on HTML pages. | Uses different expression syntax uses “[ ]” for property binding and “( )” event binding. |

# A Basic Angular 4 Application