JS ECMA Script 6
What's new in ES6 ?

- Block Scope (let/const)
- Destructuring
- Functional Programming
  - Default Params
  - Rest parameters
  - Spread Syntax
  - Arrow Function (Fat Arrow Function)
- Array Methods
- Object & Extended Object
- Iterator
- Generator
- Template Strings
- Promise
- Classes
- Modules

http://dotjs.in

Writing the code

it() / test() block

beforeEach() and afterEach()

beforeAll() and afterAll()

Inbuilt Matchers

Custom Matchers

Built-in Matcher

```
expect(x).toEqual(y) // if properties are equal (can be
                     objects, arrays etc.)
expect(x).toBe(y)    // uses === comparison
expect(x).toMatch(y) // uses regular expression
expect(x).toBeDefined(y) // not be undefined
expect(x).toBeUndefined(y) // to be undefined
expect(x).toBeNull(y)      // to be null
expect(x).toBeTruthy(y) // to be true
expect(x).toBeFalsy(y) // to be false
expect(x).toBeContain(y) // contain in array
expect(x).toBeLessThan(y) // less than
expect(x).toBeGreaterThan(y) // more than
expect(x).toThrow(y) // throws an error/exception
```

not the Negation

```
expect(x).not.toEqual(y)
expect(x).not.toBe(y)
expect(x).not.toMatch(y)
expect(x).not.toBeDefined(y)
expect(x).not.toBeUndefined(y)
expect(x).not.toBeNull(y)
expect(x).not.toBeTruthy(y)
expect(x).not.toBeFalsy(y)
expect(x).not.toBeContain(y)
expect(x).not.toBeLessThan(y)
expect(x).not.toBeGreaterThan(y)
expect(x).not.toThrow(y)
```

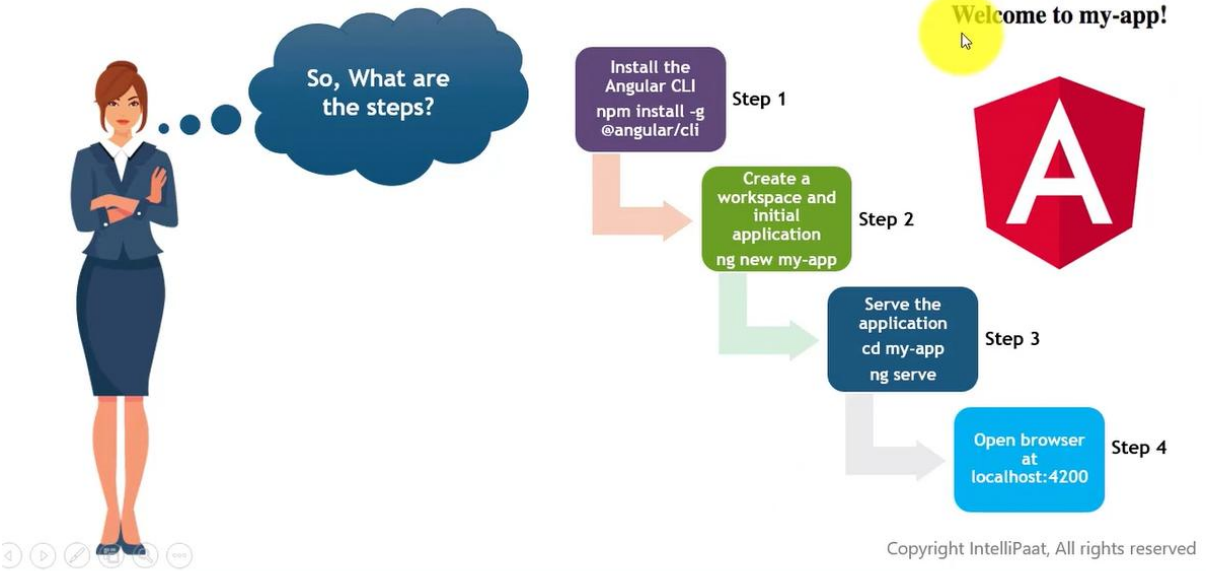
Creating custom matcher

Typically created in a beforeEach block
Use the method this.addMatchers()

```
beforeEach(function(){
  this.addMatchers({
      toBeBatman :  function(){
            {
            message : "you are not batman"
            }
            return this.actual === "batman"
      }
  })
})
```

A Quick start

IntelliPaat



Bootstrapping

An NgModule describes how the application parts fit together. Every application has at least one Angular module, the root module that you bootstrap to launch the application. By convention, it is usually called AppModule.


```
/* JavaScript imports */
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { ItemDirective } from './item.directive';

/* the AppModule class with the @NgModule decorator  with its
meta data*/
@NgModule({
declarations: [
AppComponent,ItemDirective
],
imports: [
BrowserModule,
FormsModule
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

app.module.ts

NgModule

- An NgModule is a class marked by the @NgModule decorator.
- @NgModule takes a metadata object that describes how to compile a component's template and how to create an injector at runtime.
- It identifies the module's own components, directives, and pipes, making some of them public, through the exports property, so that external components can use them.
- @NgModule can also add service providers to the application dependency injectors.

NgModule

- An NgModule is a class marked by the @NgModule decorator.
- @NgModule takes a metadata object that describes how to compile a component's template and how to create an injector at runtime.
- It identifies the module's own components, directives, and pipes, making some of them public, through the exports property, so that external components can use them.
- @NgModule can also add service providers to the application dependency injectors.

More explanation

The @NgModule decorator identifies AppModule as an NgModule class. @NgModule takes a metadata object that tells Angular how to compile and launch the application.

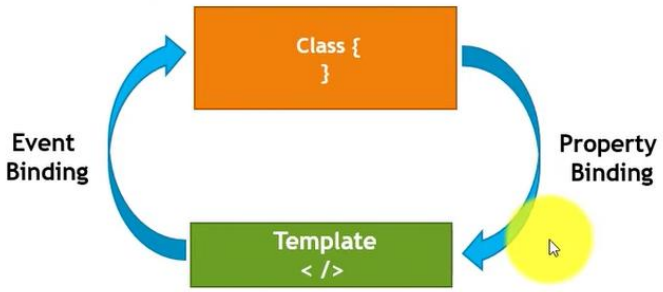
- **declarations**—this application's lone component.
- **imports**—import BrowserModule to have browser specific services such as DOM rendering, sanitization, and location.
- **providers**—the service providers.
- **bootstrap**—the root component that Angular creates and inserts into the index.html host web page.

The default application created by the Angular CLI only has one component, AppComponent, so it is in both the declarations and the bootstrap arrays.

04:13

Data Binding

- Data Binding is a process that creates a connection between the application's UI and the data.
- When the data changes its value, the UI elements that are bound to the data, will also change.
- Angular handles data binding by synchronizing the state of the view, with the data in the component.
- Data-binding can be one-way, where a change in the state affects the view, or two-way, where a change from the view can also change the model.



02:21

Types of Data Binding

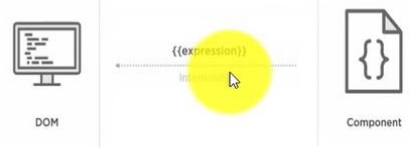
02:40

Attribute vs Property

- Attributes and properties are not same
- Attributes belong to HTML element
- Properties belong to DOM
- Attributes initialize DOM properties and then they are done.
- Attributes once initialized then they can not be changed whereas Property values can be changed.

Interpolation

- Interpolation is a technique that allows the user to bind a value to a UI element.
- Interpolation binds the data one-way. This means that when value of the field bound using interpolation changes, it is updated in the page as well.




```
< input type="text"  value={{name}}/>
```

Interpolation

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

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    <h2>My favorite hero is: {{myHero}}</h2>
    `
})
export class AppComponent {
  title = 'Tour of Heroes';
  myHero = 'Windstorm';
}
```


Property Binding

- Property binding is used to bind values to the DOM properties of the HTML elements.
- It sends information from the component class to the view.
- Property bindings are evaluated on every browser event and any changes made to the objects in the event, are applied to the properties.


```
<p><img [src]="heroImageUrl"> is the <i>property bound</i> image.</p>

<p><img bind-src="heroImageUrl"> is the <i>property bound</i> image.</p>
```

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

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    <h2>My favorite hero is: {{myHero}}</h2>
     <input type=text [disabled]= isUnchanged
value={{myHero}}/>

})
export class AppComponent {
  title = 'Tour of Heroes';
  myHero = 'Windstorm';
   isUnchanged = false;
}
```

Property Binding

Property Binding or Interpolation

You often have a choice between interpolation and property binding. The following binding pairs do the same thing:

```
<p><img src="{{heroImageUrl}}"> is the <i>interpolated</i> image.</p>

<p><img [src]="heroImageUrl"> is the <i>property bound</i> image.</p>

<p><img bind-src="heroImageUrl"> is the <i>property bound</i> image.</p>
```

- When rendering data values as strings, *Interpolation is convenient option as it increases readability.*
- When setting an element property to a non-string data value, you must use *property binding*.

Event Binding

What is an event : A user expects a UI to respond to her/his actions on the page. Every such action would trigger an event on the page and the page has to respond by listening to these events.

What is event binding : The event binding system provides us the way to attach a method defined in a component with an event.

```
<button class='btn' (click)='submit()'>Submit</button>
<button class='btn' on-click='submit()'>Submit</button>
```

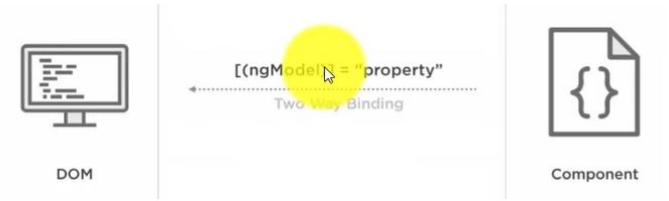
Here, the method submit has to be defined in the component class. Any DOM event can be either prefixed with *on-* or can be enclosed inside parentheses to bind it with a method in the component class.

Types of Events

- (scroll)="scrollCallback()"
- (cut)="cutCallback()"
- (copy)="copyCallback()"
- (paste)="pasteCallback()"
- (keydown)="keydownCallback()"
- (keypress)="keypressCallback()"
- (keyup)="keyupCallback()"
- (mouseenter)="mouseenterCallback()"
- (mousedown)="mousedownCallback()"
- (mouseup)="mouseupCallback()"
- (click)="clickCallback()"
- (dblclick)="dblclickCallback()"
- (drag)="dragCallback()"
- (dragover)="dragoverCallback()"
- (drop)="dropCallback()"

Two way data Binding

- The feature two-way binding in Angular is derived from the property and event bindings.
- The property and event bindings are directed one way with the former receiving data into view from the component object and the later sending data from the view to the component.
- The two-way binding is a combination of these two bindings; it gets the data from the component object to the view and sets the data from view to the component object.



Two way data Binding

The following snippet shows an example of a directive, *ngModel* to show how two-way binding can be used:

```
<input type="text" [(ngModel)]="name" />
     <div>{{name}}</div>
```

Here, the field name is two-way bound on the input box. When it is rendered on a page, it shows the existing value of the field and when the value is modified on the screen, it updates the value in the field.

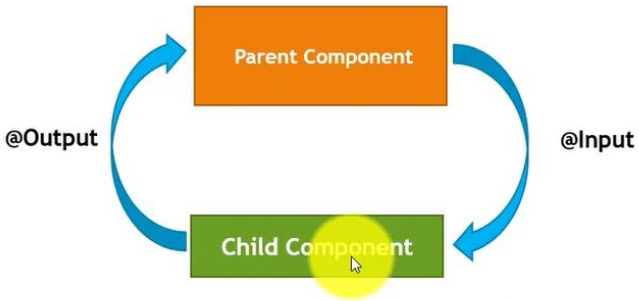
```
import { NgModule }      from '@angular/core';
import { BrowserModule }  from '@angular/platform-browser';
import { FormsModule }    from '@angular/forms'; // Importing
forms module to this file

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

@NgModule({
  imports: [
    BrowserModule,
    FormsModule // Importing forms module to application module
  ],

  declarations: [
    AppComponent,
    DemoComponent
  ],

  bootstrap: [ AppComponent ]
})

export class AppModule { }
```


Two way Data Binding

Component

Template	Class	Metadata
• View • HTML	• Code • TypeScript • Data & Methods	• Information • Decorator

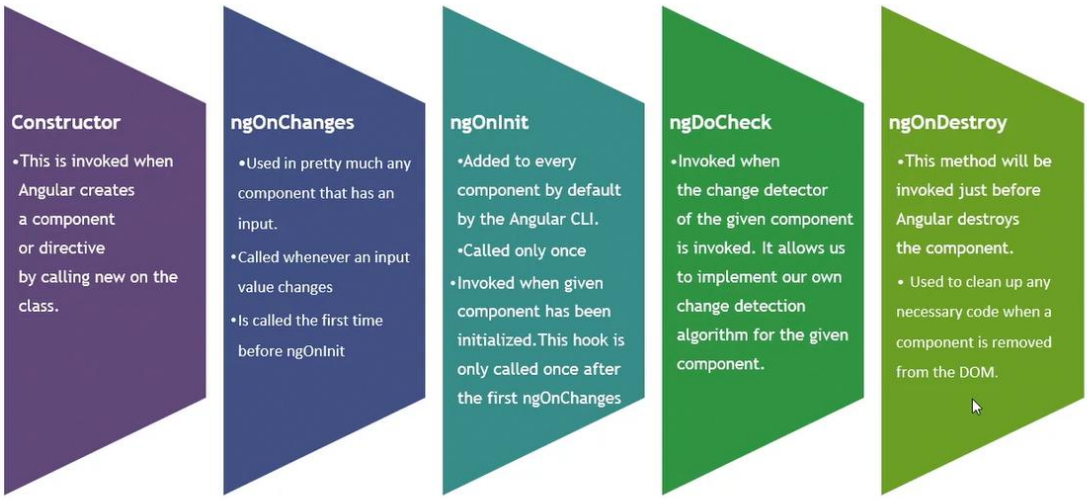
Component Interaction



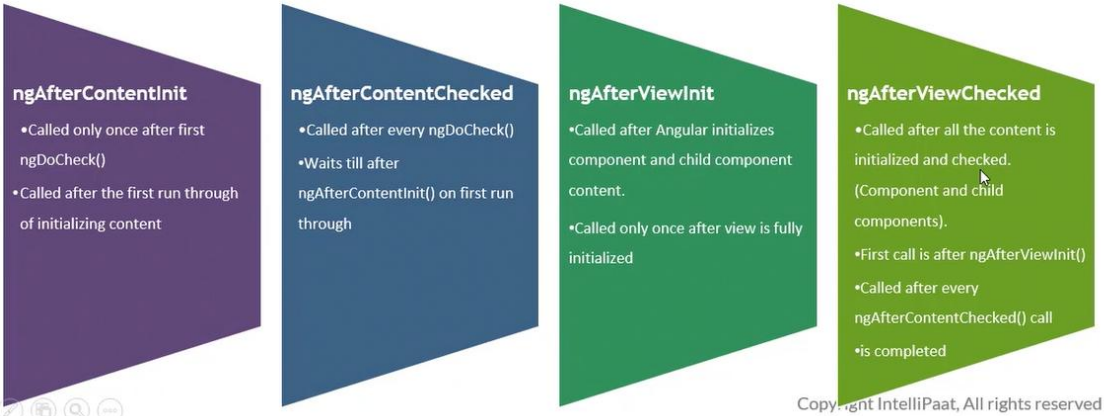
Component LifeCycle

- A component in Angular has a life-cycle, a number of different phases it goes through from birth to death and we call those methods *hooks*.
- The hooks are executed in this order.
- These phases are broadly split up into phases that are linked to the component itself and phases that are linked to the *children* of that component.
- Using life-cycle hooks we can fine tune the behavior of our components during creation, update and destruction.

Hooks for the Component



Hooks for the Component children



Directives

There are three kinds of directives in Angular:

Components	Structural	Attribute
• Component directives form the main class. It possesses the details about how the component should be instantiated, processed and utilized at runtime.	• change the DOM layout by adding and removing DOM elements. • Ex: ngForOf ,ngIf, ngSwitch	• change the appearance or behavior of an element, component. • Ex ngStyle, ngClass, ngModel.

Component

Template		Class		Metadata
• View • HTML	+	• Code • TypeScript • Data & Methods	+	• Information • Decorator

Structural Directives

- 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. like *ngIf, *ngForOf, *ngSwitch
- These directive belong to @angular/common/src/directives

Built-in Structural Directives

ngIf	ngSwitch	ngForOf
• conditionally add or remove an element from the DOM	• a set of directives that switch among alternative views	• repeat a template for each item in a list

ngIf Syntax

- **Simple form:**

```
<div *ngIf="condition">...</div>
<ng-template [ngIf]="condition"><div>...</div></ng-template>
```

- **Form with else block:**

```
<div *ngIf="condition; else elseBlock">...</div>
<ng-template #elseBlock>...</ng-template>
```

- **Form with then and else block:**

```
<div *ngIf="condition; then thenBlock else elseBlock"></div>
<ng-template #thenBlock>...</ng-template>
 <ng-template #elseBlock>...</ng-template>
```

- **Form with storing the value locally:**

```
<div *ngIf="condition as value; else elseBlock">{{value}}</div>
<ng-template #elseBlock>...</ng-template>
```


ngFor Syntax

```
<li *ngFor="let user of users; index as i; first as isFirst">

{{i}}/{{users.length}} {{user}} {{isFirst}}

</li>
```

index: number: The index of the current item in the iterable.
first: boolean: True when the item is the first item in the iterable.
last: boolean: True when the item is the last item in the iterable.
even: boolean: True when the item has an even index in the iterable.
odd: boolean: True when the item has an odd index in the iterable.

ngSwitch Syntax

```
<div [ngSwitch]=“color">
<div *ngSwitchCase=“red”>You picked red</div>
<div *ngSwitchCase=“blue”>You picked blue</div>
<div *ngSwitchCase=“green”>You picked green</div>
<div * ngSwitchDefault>Try again</div>
</div>
```


Attribute Directives

- Attribute directives listen to and modify the behavior of other HTML elements, attributes, properties, and components.
- They are usually applied to elements as if they were HTML attributes, hence the name. like: ngClass, ngStyle and ngModel.

Built-in Attribute Directives

ngClass	ngStyle	ngModel
• add and remove a set of CSS classes dynamically for a element.	• add and remove a set of HTML styles dynamically for a element.	• two-way data binding to an HTML form element

ngClass Syntax

- <some-element [ngClass]="{'text-success': true}">...</some-element>
- <button [ngClass]="{'btn btn-primary'}">...</some-element>
- <button [ngClass]="{['btn', 'btn-primary']}">...</some-element>

Custom Directives

- Create the directive class file in a terminal window with the CLI command
  ng generate directive appHighLight
- Apply in template :
  <p appHighLight></p>

```
import { Directive, ElementRef } from '@angular/core';

@Directive({ selector: '[appHighlight]' })

export class HighlightDirective

{ constructor(el: ElementRef)

{ el.nativeElement.style.backgroundColor = 'yellow'; }

}
```

Custom Directives using Listeners

```
import { Directive, ElementRef } from '@angular/core';

@Directive({ selector: '[appHighlight]' })

export class HighlightDirective
{constructor(private e1:ElementRef) {
}
@HostListener('mouseenter') onMouseEnter() {
this.highlight('yellow');
}
@HostListener('mouseleave') onMouseLeave() {
this.highlight(null);
}
private highlight(color: string) {
this.e1.nativeElement.style.backgroundColor = color;
} }
```

Services

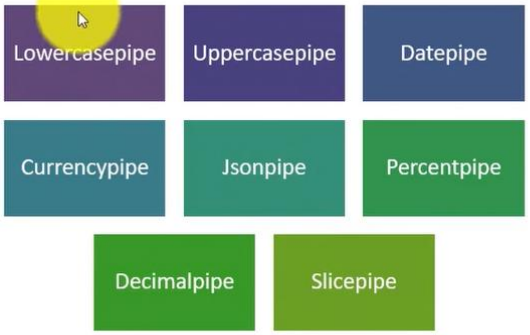
- Service is a class
- Share the data
- Implement application logic
- Services are a great way to share information among classes that *don't know each other*

Services

- Service is a class
- Share the data
- Implement application logic
- Services are a great way to share information among classes that *don't know each other*
- Components shouldn't fetch or save data directly and they certainly shouldn't knowingly present fake data. They should focus on presenting data and delegate data access to a service.

Pipes

- Pipes are used to transform data before it gets displayed in a template.
- Angular provides some built-in pipes. The pipes are listed below –



Lecture 29_AJ_Pipes

Currency Pipe

IntelliPaat

This pipe is used for formatting currencies. Its first argument is an abbreviation of the currency type (e.g. "EUR", "USD", and so on), like so:

```
{{ 1234.56 | currency:'GBP' }}
```


Currency Pipe

IntelliPaat

This pipe is used for formatting currencies. Its first argument is an abbreviation of the currency type (e.g. "EUR", "USD", and so on), like so:

```
{{ 1234.56 | currency:'GBP' }}
```

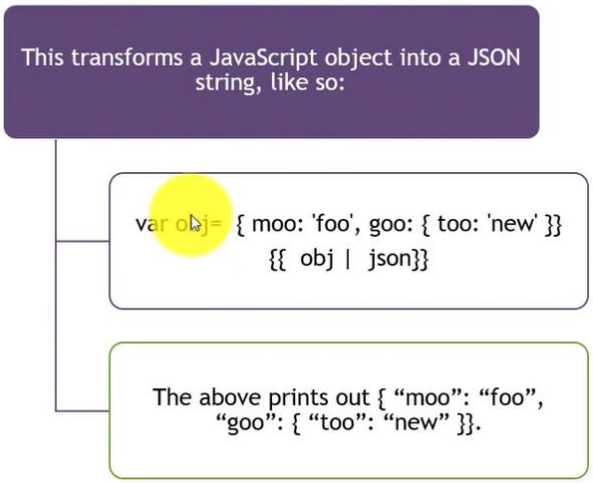
The above prints out £1,234.56.

If we want to print the currency code then as a second parameter pass code keyword, like so:

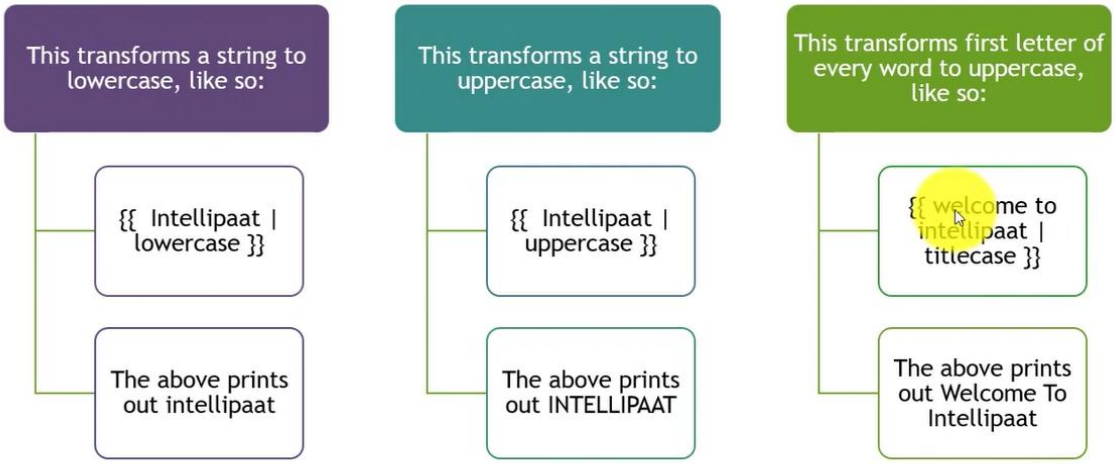
```
{{ 1234.56 | currency:'GBP':'code' }}
```

The above prints out GBP1,234.56.

JSON Pipe



Lowercase and Uppercase Pipes



Slice Pipe

This returns a slice of an array. The first argument is the start index of the slice and the second argument is the end index.

```
var str= "Welcome to Intellipaat"
{{str | slice:2:5 }}
```


Lecture 29_AJ_Pipes

number Pipe

This pipe is used for the transformation of numbers.

```
{{ 5.678 | number: '1.2-3' }}
// 1 is for main integer, 2-3 are for decimals to be displayed.
```

The above prints out 5.678

```
{{ 5.678 | number: '3.2-4' }}
// 3 is for main integer, 2-4 are for decimals to be displayed.
```

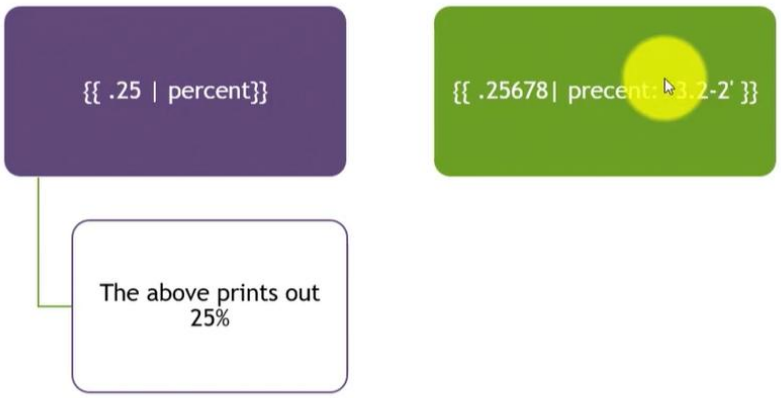
The above prints out 005.6780

```
{{ 5.678 | number: '3.1-2' }}
// 3 is for main integer,1-2 are for decimals to be displayed.
```

The above prints out 005.68

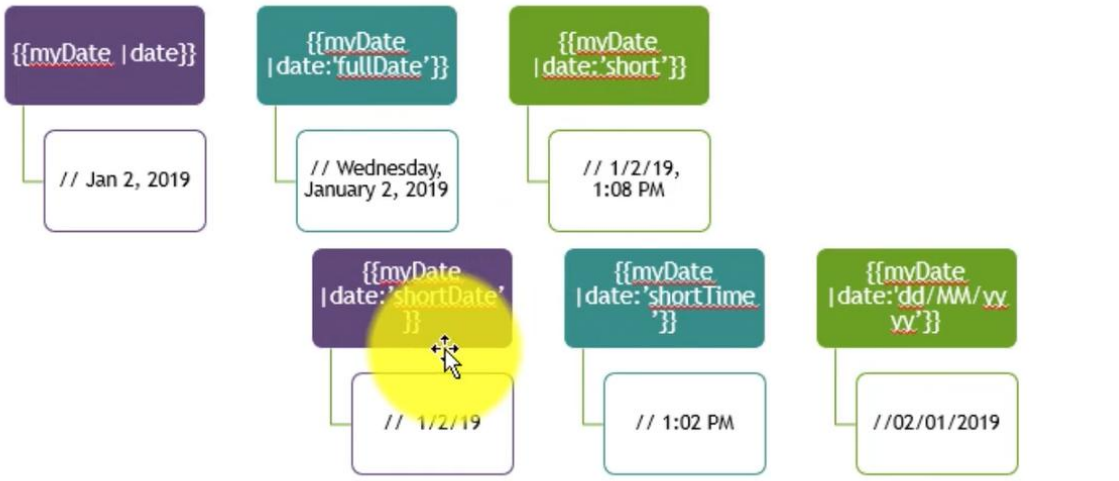
percent Pipe

This pipe is used for the transformation of numbers into percentage.



date Pipe

This pipe is used for the transformation of dates. The first argument is a format string, like so:



Custom Pipes Demo1

How to create custom pipes? You need to create a .ts file

```
import {Pipe, PipeTransform} from '@angular/core';
@Pipe (
{ name : 'sqrt' }
)
 export class SqrtPipe implements PipeTransform
{ transform(val : number) : number
 { return Math.sqrt(val); }
}
```

15:50

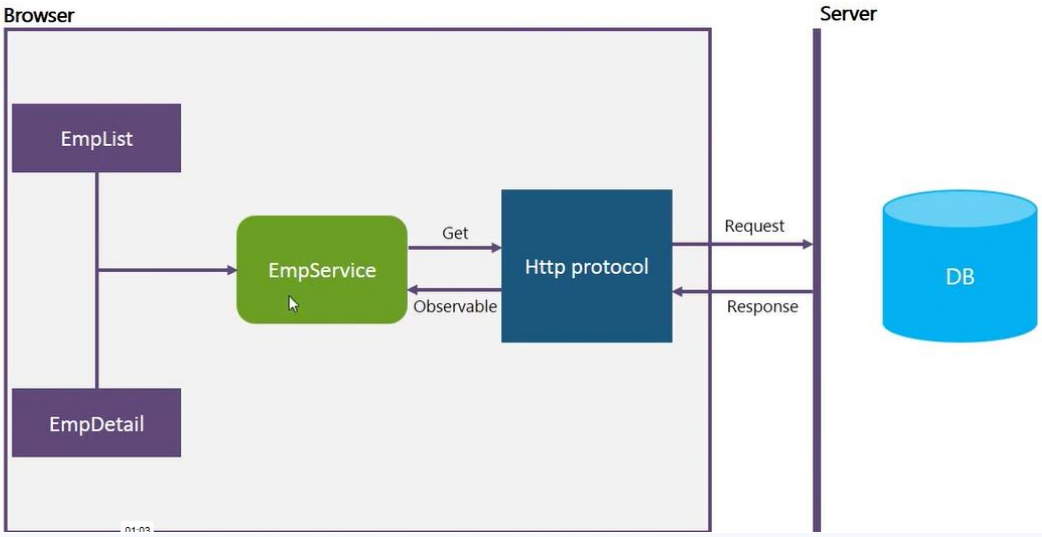
Custom Pipes Demo2

```
import {Pipe, PipeTransform} from '@angular/core';
@Pipe (
{ name : 'defaultImage' })
 export class DefaultImagePipe implements PipeTransform
{transform(value: string, fallback: string): string
{    let image = "";
  if (value) {      image = value;    }
 else {      image = fallback;    }     return image;   } }
```

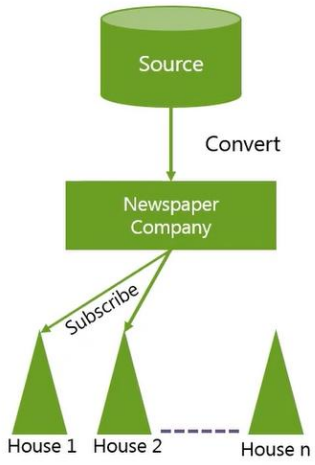
How to use it?

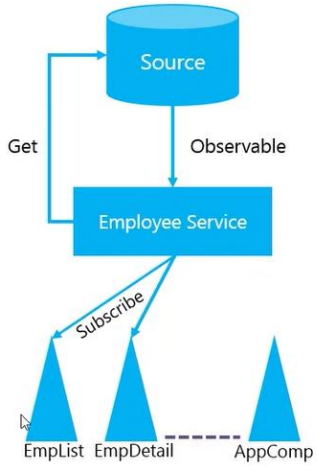
```
<img [src]="imageUrl |
default:"http://s3.amazonaws.com/uifa
ces/faces/twitter/sillyleo/128.jpg'"/>
```

HTTP Mechanism



Observables

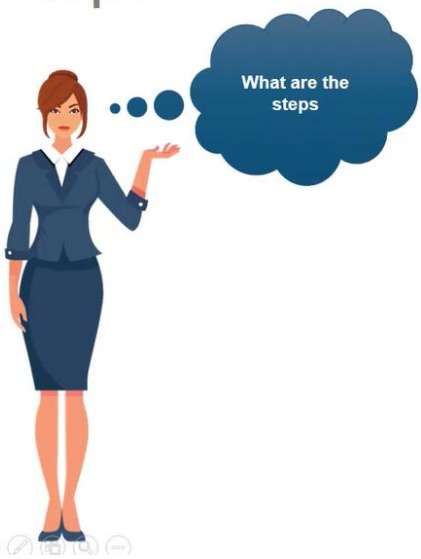




HTTP Client

- Most front-end applications communicate with backend services over the HTTP protocol using the XMLHttpRequest interface.
- The HttpClient in @angular/common/http offers a simplified client HTTP API for Angular applications that rests on the XMLHttpRequest interface exposed by browsers.

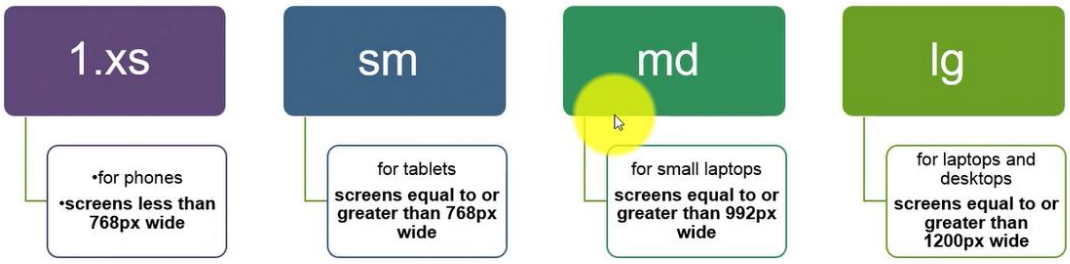
Steps



- You need to import the HttpClientModule from @angular/common/http.
- HTTP Get request from EmpService
- Receive the observable and cast it into an employee array
- Subscribe to the observable from EmpList and EmpDetail
- Assign the employee array to a local variable

Bootstrap Grid Classes

The Bootstrap grid system has four classes:

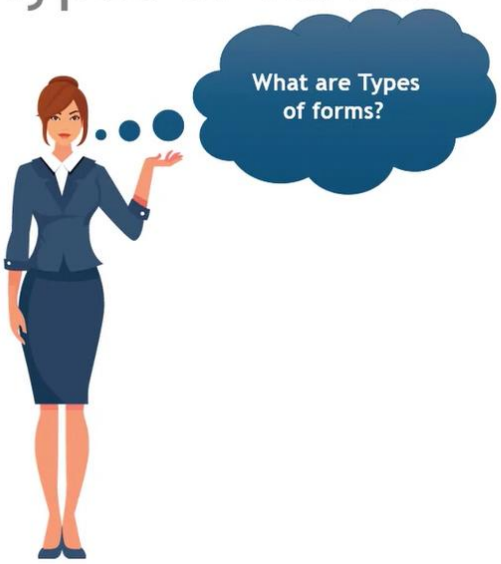


What are forms

- Forms are very important to your business applications.
- As a developer we need to take care of below points:

1. Data Binding
2. Change Tracking
3. Validation
4. Visual Feedback
5. Error messages
6. Form submission

Types of forms



Reactive Forms
(Heavy on component class)

More robust.

More scalable, reusable, and testable.

Template Drive forms
(Heavy on component template)

Useful for adding a simple form to an app, such as an email list signup form.

Easy to add to an app, but they don't scale as much as reactive forms.

Template Drive Forms (TDF)

- Easy to use and very much similar to Angular JS forms
- Rely on two way data binding with ngModel
- Heavy HTML code and minimal component code.
- Automatically tracks the form and form elements state and validity
- Drawback is the challenge in unit testing
- Readability decreases with complex forms and validation

Key Differences

	Reactive	Template-Driven
Setup (form model)	More Explicit, created in component class	Less Explicit, created in directives
Data model	Structured	Unstructured
Predictability	Synchronous	Asynchronous
Form Validation	Functions	Directives
Mutability	Immutable	Mutable
Scalability	Low-level API access	Abstraction on top of APIs

Steps for TDF

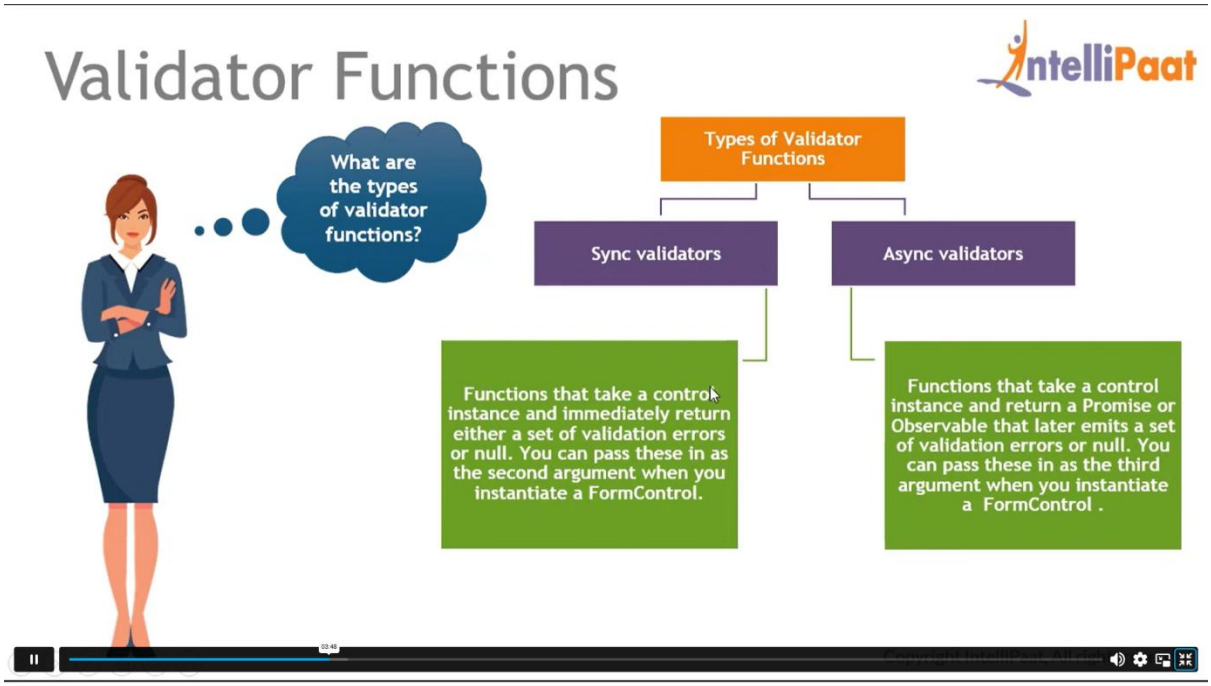
- Generate a new CLI project
- Add the form HTML
- Binding data
- Tracking state and validity
- Providing visual feedback
- Displaying error messages
- Submitting data to server

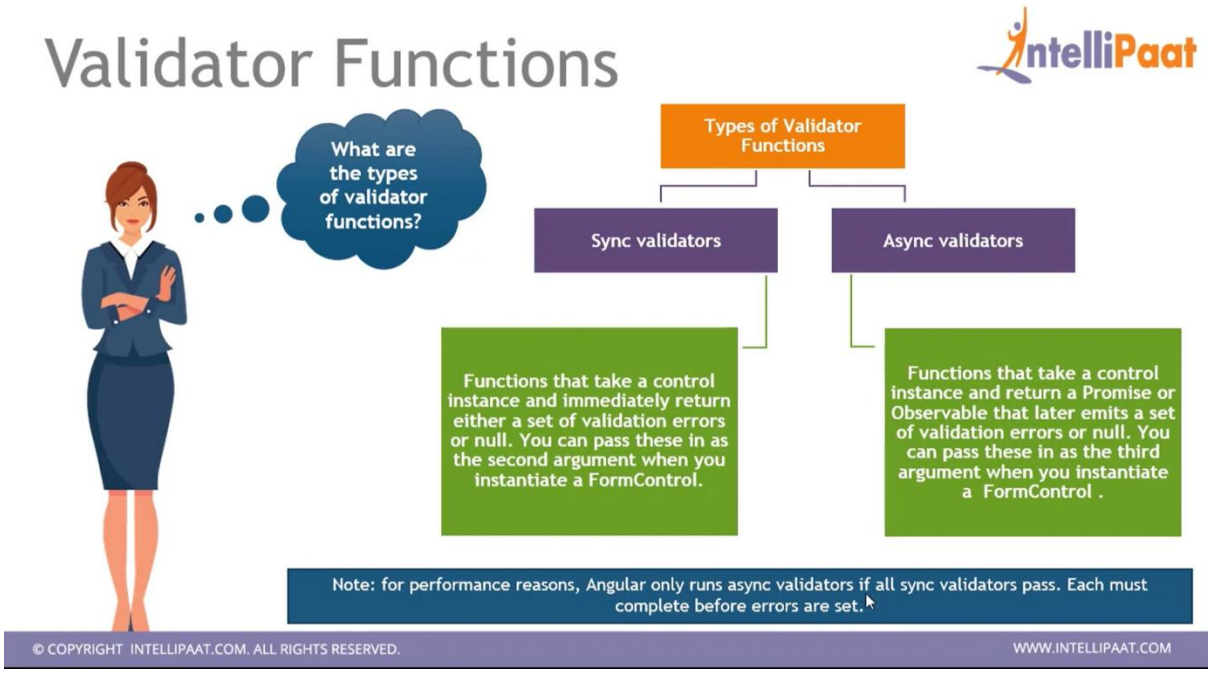
Steps for Reactive

- Generate a new CLI project
- Add the form HTML
- Create the form model
- Manage the form control values
- FormBuilder service
- Validations
- Dynamic Form controls
- Submitting form data

Form Controls states

State	Class if true	Class if false
The control has been visited	ng-touched	ng-untouched
The control's value has changed	ng-dirty	ng-pristine
The control's value is valid	ng-valid	ng-invalid


Validator Functions
IntelliPaat
What are the types of validator functions?
Types of Validator Functions
Sync validators
Async validators
Functions that take a control instance and immediately return either a set of validation errors or null. You can pass these in as the second argument when you instantiate a FormControl.
Functions that take a control instance and return a Promise or Observable that later emits a set of validation errors or null. You can pass these in as the third argument when you instantiate a FormControl .


Validator Functions
IntelliPaat
What are the types of validator functions?
Types of Validator Functions
Sync validators
Async validators
Functions that take a control instance and immediately return either a set of validation errors or null. You can pass these in as the second argument when you instantiate a FormControl.
Functions that take a control instance and return a Promise or Observable that later emits a set of validation errors or null. You can pass these in as the third argument when you instantiate a FormControl .
Note: for performance reasons, Angular only runs async validators if all sync validators pass. Each must complete before errors are set.
© COPYRIGHT INTELLIPAAT.COM. ALL RIGHTS RESERVED.
WWW.INTELLIPAAT.COM

Navigation and Routing

The browser is a familiar model of application navigation:

- Enter a URL in the address bar and the browser navigates to a corresponding page.
- Click links on the page and the browser navigates to a new page.
- Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.
- The Angular Router enables navigation from one view to the next as users perform application tasks.

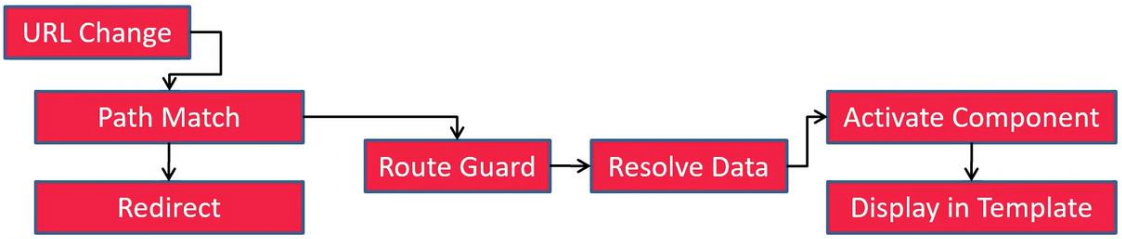
Navigation and Routing

The browser is a familiar model of application navigation:

- Enter a URL in the address bar and the browser navigates to a corresponding page.
- Click links on the page and the browser navigates to a new page.
- Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.
- The Angular Router enables navigation from one view to the next as users perform application tasks.

Routing | Work flow

Users click on a link (created by routerlink)
Angular navigates to that link changing the location URL in the address bar
The location change triggers the router from the route configuration
Angular then loads the appropriate component in the template <router-outlet>

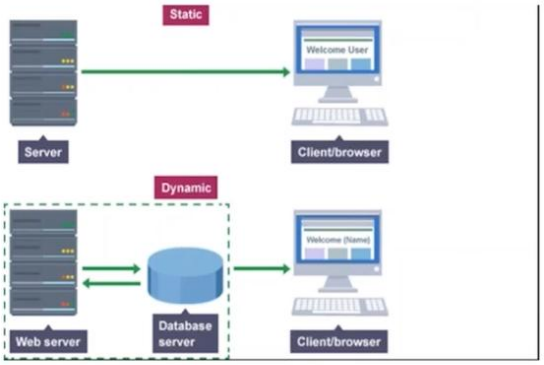


Course Objectives

- Why Angular
- Angular Versions & Architecture
- Environment Setup
- Angular Prerequisites
- Angular Fundamentals
- Angular Advanced Concepts
- Angular Testing
- Angular Best Practices

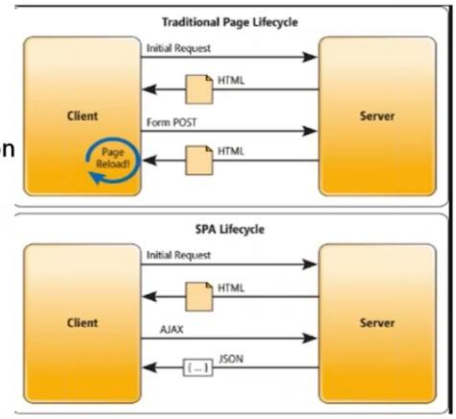
Why Angular –Dynamic View

- Static View:
  - Only Static Contents
- Dynamic View
  - Events
  - Interactive



Why Angular -Single Page Applications

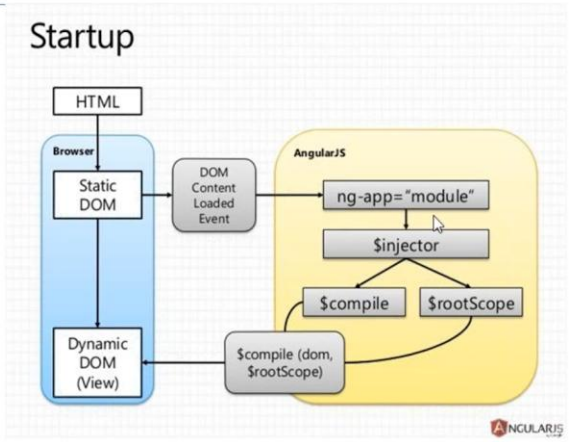
- Responsive
- Less round trips
- Less N/W consumption
- Single Entry Point
- Infrastructure Costs



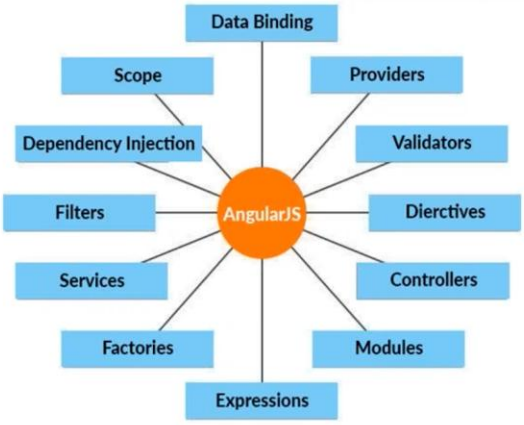
Why Angular- What it is

- Open Source JavaScript Framework
- Used for building client side applications
- Maintained by Google

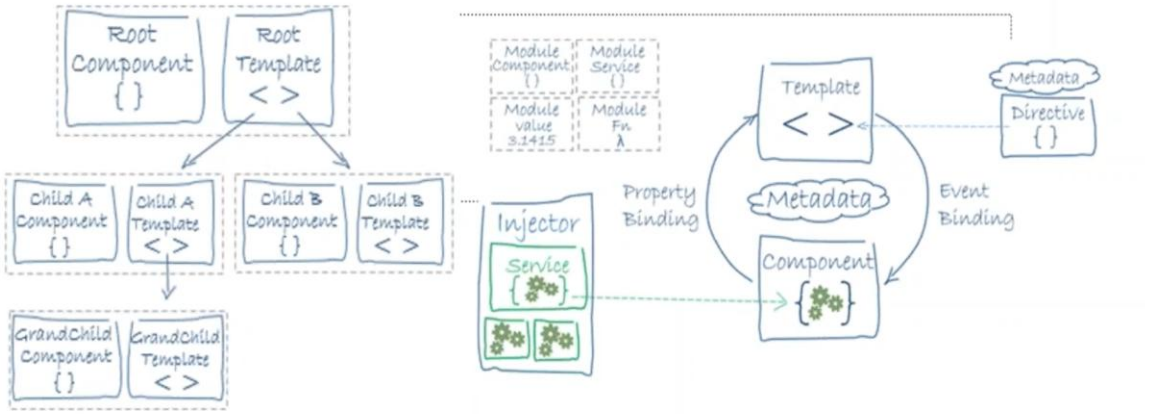
How Angular Works?

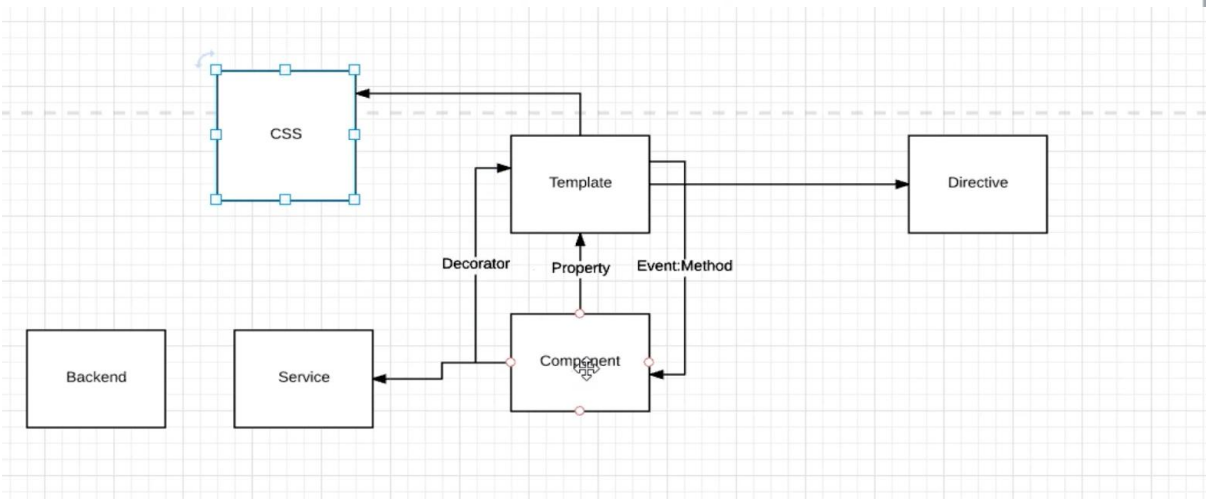


Angular Components



Angular 2 Architecture

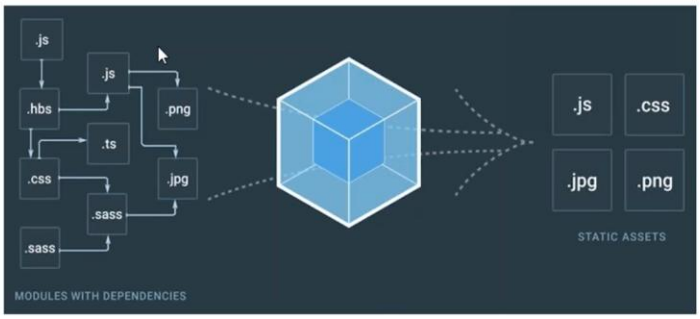




Web Pack

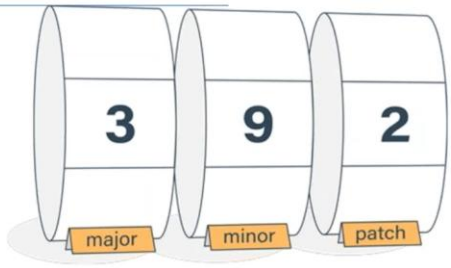
- Npm install –g webpack
- npm install webpack –save
- ng eject

- Code Splitting
- Loaders
- Support plugins

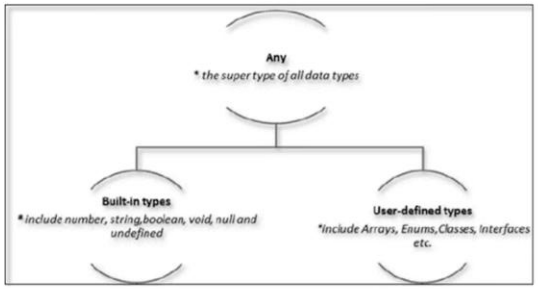


Project Folders-Package.json

Symbol	Dependency	Versions	Changes
caret (^)	^3.9.2	3.*.*	- backwards compatible new functionality - old functionality deprecated, but operational - large internal refactor - bug fix
tilde (~)	~3.9.2	3.9.*	- bug fix

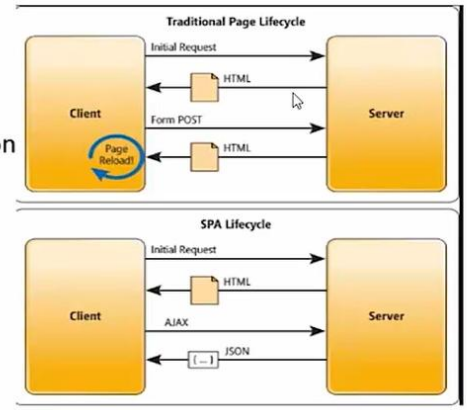


TypeScript Data types



Why Angular -Single Page Applications

- Responsive
- Less round trips
- Less N/W consumption
- Single Entry Point
- Infrastructure Costs



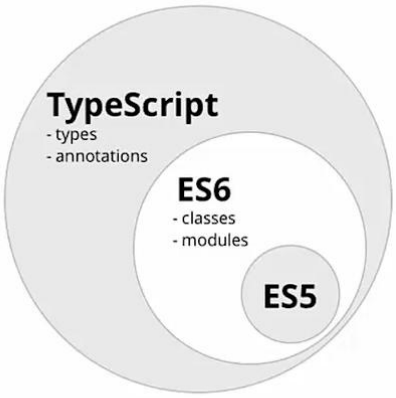
Why Angular?:Design Goals of AngularJS I x

- Decouple DOM manipulation from application logic:
- Decouple the client side of an application from the server side.
- Provide structure for the journey of building an application:
  - Designing the UI
  - Writing the business logic
  - Testing.
- Easy Backend Integration
- Powerful Data Binding

Why Angular- What it is

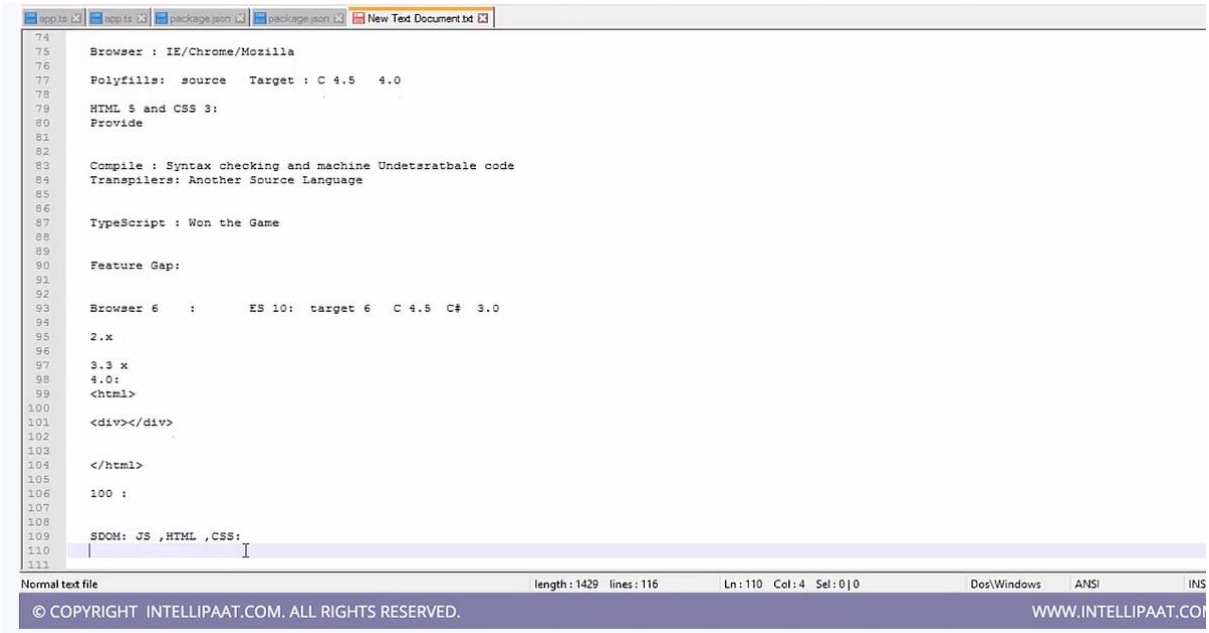
- Open Source JavaScript Framework
- Used for building client side applications
- Maintained by Google

TypeScript



Angular Goals

- Speed
- Modern
- Simplified API
- Across All Platforms
- Starting Point:
  - https://angular.io/resources



NG BootStrap

- Create Project
  - ng new bootstrap-demo --skip-install --spec false --prefix bsp
- Add Dependnecies
  - npm install --save @ng-bootstrap/ng-bootstrap bootstrap@4.0.0-alpha.6 font-awesome
- Import the Module
  - import {NgbModule} from '@ng-bootstrap/ng-bootstrap';
- Add the Styles
  - "../node_modules/bootstrap/dist/css/bootstrap.min.css", "../node_modules/font-awesome/css/font-

Typescript Features

- TypeScript is a wrapper on JavaScript.
- Supports OO Concepts
- TypeScript supports other JS libraries. Such as React ,Vue etc
- JavaScript is TypeScript. Does not sound good…?
- TypeScript is portable
- Tsc –init
- tsc file1.ts, file2.ts, file3.ts