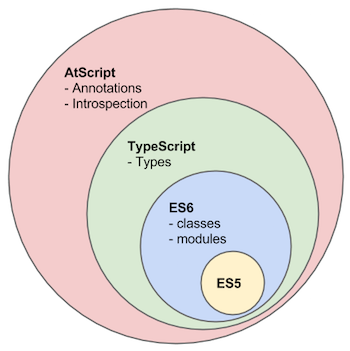
**Angular**

1. Typescript:

TypeScript is super set of java script. Its free open-source programming language developed and maintained by Microsoft.

ES5 is basic JavaScript ES6 is latest specification of JavaScript. According to picture TypeScript is superset of ES6 and ES6 is super set of ES5.

TypeScript is just a superset of JavaScript. It offers more features than vanilla JavaScript, like classes interfaces and, very important which gives it the name types Strong typing.

TypeScript allows us to write much more robust code which gets checked at the time we write it and not just the time we run it.

**TypeScript doesn’t run on browser so it compiles to JavaScript into the end and this compilation is handled by the CLI. One of the reasons why we need project management tool like CLI.**

1. **Angular**

Q. What is Angular?

Angular is **JavaScript framework** which allows us to create reactive **Single-Page-Application (SPAs)**.

# Angular is framework that is used to build client-side application. It is great for building single page application where parts of views get refreshed asynchronously without having to reload the entire page.

**If any application relies heavily on JavaScript, then Angular is great choice.**

* **Why learn Angular as approach to some other java script framework or library?**

There are couple of points:

1. By design Angular promotes Modular approach and application we built will have clear structure.
2. By making use of components which is a feature of Angular we can have a lot of reusable code.
3. Angular has lot of inbuilt capabilities such as Validations, routing, and so on which make development quicker and easier.
4. Angular even makes it possible to write unit testable and easily maintainable code.
5. And finally, Angular is product of Google team and makes use of Type script language of Microsoft so it safe to say Angular is here to stay.

* **Angular’s History**

1. **2010 – Angular js**

This was huge hit and a lot of organization started using Angular JS to build their Enterprise Applications.

1. **2016 – Angular Version 2**

This was called just Angular. It was announced that Angular JS will refer to the 1.x versions and **Angular** without the JS will refer to version 2 and up.

1. **2016 Dec – Angular Version 4**

In Dec 2016 Angular 4 was announced. Version 3 was skipped to avoid the confusion due to the misalignment of router’s package version which was already distributed as version 3.3.0.

1. **2017 Nov – Angular Version 5**

Key improvements in Angular 5 include support for [progressive web apps](https://en.wikipedia.org/wiki/Progressive_web_app), a build optimizer and improvements related to Material Design.

1. **2018 May – Angular Version 6**

This is a major release focused less on the underlying framework, and more on the toolchain and on making it easier to move quickly with Angular in the future, like: **ng update**, **ng add**, **Angular Elements**, **Angular Material** + CDK Components, Angular Material Starter Components, CLI Workspaces, Library Support, Tree Shakable Providers, Animations Performance Improvements, and RxJS v6.

1. **2018 Oct – Angular Version 7**

Updates regarding Application Performance, Angular Material & CDK, Virtual Scrolling, Improved Accessibility of Selects, now supports Content Projection using web standard for custom elements, and dependency updates regarding Typescript 3.1, RxJS 6.3, Node 10 (still supporting Node 8).

1. **2019 May – Angular Version 8**

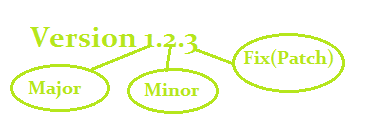
Featuring Differential loading for all application code, Dynamic imports for lazy routes, Web workers, TypeScript 3.4 support, and Angular Ivy as an opt-in preview. Angular Ivy opt-in preview includes:

* Generated code that is easier to read and debug at runtime.
* Faster re-build time.
* Improved payload size.
* Improved template type checking.
* Backwards compatibility.
* **Angular version changes**

Now let’s discuss about whether we should learn Angular with all the version changes happening.

In any framework there is always room for improvement and the same in case of Angular. Google team has decided they will upgrade angular twice in a year and stick to **sematic versioning**.

Meaning of Sematic versioning:



Major – This version id changes when there is break in any functionality.

Minor – This version id changes when added features that don’t break any functionality.

Fix (Patch) – This number will increase any time if there is bug fix or some patch.

1. **Angular-cli**

Angular CLI stands for Angular Command Line Interface. As the name implies, it is a command line tool for creating angular apps. It is recommended to use angular cli for creating angular apps as you don't need to spend time installing and configuring all the required dependencies and wiring everything together.

Now, for installing angular cli, follows the steps given below:

* Install [***node.js***](https://nodejs.org/en/download/) first if not already install (which I think you probably would have downloaded)
* Open the node.js command prompt and issue the command:

**>> npm install -g @angular/cli**

**Note:**The -g flag in the above command signifies the fact that the ng-cli is being installed in a global scope.

* If you want to check out the latest version of angular cli, modify the above stated command as:

**>> npm install @angular/cli@latest**

Now, let me list out few commands for you, which will come handy while creating angular projects:

**1.)Creating New Project: *ng new****<project-name>*

**ng new** allows you to generate a new angular project with all the boilerplate files already generated for you.

**2.) Running the Project: *ng serve***

ng serve allows you to run your angular app on the node server. The default port is *localhost:4200.*You can configure it using the command:

**>> ng serve --host 0.0.0.0 --port 4201**

Steps to run serve your angular project:

1. Go to project directory - ***cd project-name***
2. Issue the command: ***ng serve***

**3.) Generating Components, Directives & Services: *ng generate (or just ng g)***

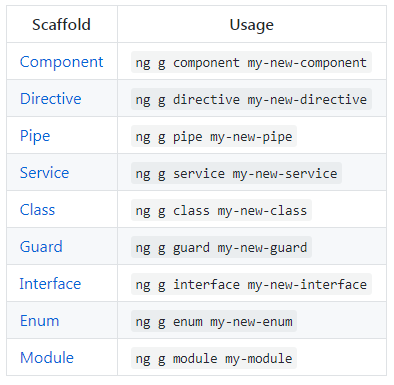
ng generate (or just ng g) command to generate Angular components:

ng generate component component-name

or,

ng g component component-name # using the alias g for generate

Similarly, you can create all other building blocks listed in the table below:



**4.) Updating your ng cli**

To update Angular CLI to a new version, go through the following steps:

**>> npm uninstall -g @angular/cli  
>> npm cache clean  
>> npm install -g @angular/cli@latest**

1. **Component:**

A component in Angular is the basic building block. It is visible to the end user and can be reused many times in an application.

Angular itself is the top-level component and we break it into smaller child components.

**For** **example**, if we want to create a shopping cart application which is an online shopping cart for mobile devices in Angular, the top most component would be shopping cart component which consists of child components called Welcome component, login component. Login component consists of product list component, product details component.

**A Component consists of the following –**

1. **Template** − This is used to render the view for the application. This contains the HTML that needs to be rendered in the application. This part also includes the binding and directives.

Syntax:

Template: '

<HTML code>

class properties

'

Eg:

template: '

<div>

<h1>{{appTitle}}</h1>

<div>To Tutorials Point</div>

</div>

'

1. **Class** − This is like a class defined in any language such as Java. This contains properties and methods. This has the code which is used to support the view. It is defined in **TypeScript**.

Syntax:

class classname {

Propertyname: PropertyType = Value

}

**PropertyType** − Since TypeScript is strongly typed, you need to give a type to the property.

**Eg**:

export class AppComponent {

appTitle: string = 'Welcome';

}

Note: The **export** keyword is used so that the component can be used in other modules in the Angular JS application.

1. **Metadata** − This is used to decorate Angular JS class with additional information. It is defined with a **decorator**.

**completed code with our class, template, and metadata.**

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

**@Component ({**

**selector: 'my-app',**

**template: ` <div>**

**<h1>{{appTitle}}</h1>**

**<div>To Tutorials Point</div>**

**</div> `,**

**})**

**export class AppComponent {**

**appTitle: string = 'Welcome';**

**}**

In the above example, the following things need to be noted −

* We are using the import keyword to import the ‘Component’ decorator from the angular/core module.
* We are then using the decorator to define a component.
* The component has a selector called ‘my-app’. This is nothing but our custom html tag which can be used in our main html page.

1. Directive

A **directive** is a custom HTML element that is used to extend the power of HTML. Angular has the following directives that get called as part of the BrowserModule module.

* ngif
* ngFor
* ngModule

Note: The Directives which contains \* before its declaration are structural directives because these changes the structure.

However, ngStyle, ngClass are attribute Directives in which declearation \* mark not required.

**ngIf**

The ngif element is used to add elements to the HTML code if it evaluates to true, else it will not add the elements to the HTML code.  In the app.component.html file, add the following code.

<div \*ngIf = 'appStatus'>{{appTitle}} Tutorialspoint </div>

\*ngIf with Else condition:

<div \*ngIf = 'appStatus; else noServer'>{{appTitle}} Tutorialspoint </div>

<ng-template #noServer>

<p>No Server was Created!</p>

</ng-template>

## ngFor

The ngFor element is used to elements based on the condition of the For loop.

In the app.component.html, define the following code.

<div \*ngFor = 'let lst of appList'>

<ul>

<li>{{lst.ID}}</li>

<li>{{lst.Name}}</li>

</ul>

</div>

With index:

<div \*ngFor = 'let lst of appList; let I = index'>

<ul>

<li>{{lst.ID}}</li>

<li>{{lst.Name}}</li>

</ul>

</div>

**ngModule**: - Todo

**ngStyle**:

<p [ngStyle] = “background:getColor()”></p>

Same way we can use ngClass

**Decorator**

Provides a way to add annotations and a metadata to the class declaration, method, accessor, property or parameter. To mark a class as Angular Component **@Component** decorator is used.

1. **import** { Component } **from** '@angular/core';
3. @Component({
4. selector: 'demo',
5. **template**: 'Hello {{name}}!'
6. })
7. **export** **class** DemoComponent {
8. name: **string** = 'World';
9. }

In above example class DemoComponent is not only class but an Angular Component, having attached decorator (a.k.a. metadata) that tells Angular how to process a class.

There are many decorators available such as

@Directive, @Injectable, @NgModule, etc applied to classes as per the usage. Find all available decorators below:

* [Attribute](https://angular.io/api/core/Attribute)
* [Component](https://angular.io/api/core/Component)
* [ContentChild](https://angular.io/api/core/ContentChild)
* [ContentChildren](https://angular.io/api/core/ContentChildren)
* [Directive](https://angular.io/api/core/Directive)
* [Host](https://angular.io/api/core/Host)
* [HostBinding](https://angular.io/api/core/HostBinding)
* [HostListener](https://angular.io/api/core/HostListener)
* [Inject](https://angular.io/api/core/Inject)
* [Injectable](https://angular.io/api/core/Injectable)
* [Input](https://angular.io/api/core/Input)
* [NgModule](https://angular.io/api/core/NgModule)
* [Optional](https://angular.io/api/core/Optional)
* [Output](https://angular.io/api/core/Output)
* [Pipe](https://angular.io/api/core/Pipe)
* [Self](https://angular.io/api/core/Self)
* [SkipSelf](https://angular.io/api/core/SkipSelf)
* [ViewChild](https://angular.io/api/core/ViewChild)
* ViewChilden

Pipes (|)

A pipe takes in data as input and transforms it to a desired output. In this page, you'll use pipes to transform a component's birthday property into a human-friendly date.

<p>The hero's birthday is {{ birthday **|** date }}</p>

Custom Pipes

The general way to define a custom pipe is as follows.

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'Pipename'})

export class Pipeclass implements PipeTransform {

transform(parameters): returntype { }

}

Where,

* **'Pipename'** − This is the name of the pipe.
* **Pipeclass** − This is name of the class assigned to the custom pipe.
* **Transform** − This is the function to work with the pipe.
* **Parameters** − This are the parameters which are passed to the pipe.
* **Returntype** − This is the return type of the pipe.

**Example**:

**Step 1** − First, create a file called multiplier.pipe.ts. Place the following code in the above created file.

import { Pipe, PipeTransform } from '@angular/core';

@Pipe ({

name: 'Multiplier'

})

export class MultiplierPipe implements PipeTransform {

transform(value: number, multiply: string): number {

let mul = parseFloat(multiply);

return mul \* value

}

}

Following points need to be noted about the above code.

* We are first importing the Pipe and PipeTransform modules.
* Then, we are creating a Pipe with the name 'Multiplier'.
* Creating a class called MultiplierPipe that implements the PipeTransform module.
* The transform function will then take in the value and multiple parameter and output the multiplication of both numbers.

**Step 3** − In the app.component.ts file, place the following code.

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

@Component ({

selector: 'my-app',

template: '<p>Multiplier: {{2 | Multiplier: 10}}</p>'

})

export class AppComponent { }

**Note** − In our template, we use our new custom pipe.

**Step 4** − Ensure the following code is placed in the app.module.ts file.

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

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

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

import {MultiplierPipe} from './multiplier.pipe'

@NgModule ({

imports: [BrowserModule],

declarations: [AppComponent, MultiplierPipe],

bootstrap: [AppComponent]

})

export class AppModule {}

Following things need to be noted about the above code.

* We need to ensure to include our MultiplierPipe module.
* We also need to ensure it is included in the declarations section.

**Module**

 Angular 2 Applications have one or more modules. Each module has a single dedicated purpose. The module helps us to organize the Application's cohesive group of functionalities.

**app.module.ts is known as root module class.**

The following code will be present in the app.module.ts file.

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

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

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

@NgModule ({

imports: [ BrowserModule ],

declarations: [ AppComponent ],

bootstrap: [ AppComponent ]

})

export class AppModule { }

Let’s go through each line of the code in detail.

* The import statement is used to import functionality from the existing modules. Thus, the first 3 statements are used to import the NgModule, BrowserModule and AppComponent modules into this module.
* The NgModule decorator is used to later on define the imports, declarations, and bootstrapping options.
* The BrowserModule is required by default for any web based angular application.
* The bootstrap option tells Angular which Component to bootstrap in the application.

A module is made up of the following parts −

* **Bootstrap array** − This is used to tell Angular JS which components need to be loaded so that its functionality can be accessed in the application. Once you include the component in the bootstrap array, you need to declare them so that they can be used across other components in the Angular JS application.
* **Export array** − This is used to export components, directives, and pipes which can then be used in other modules.
* **Import array** − Just like the export array, the import array can be used to import the functionality from other Angular JS modules.

Service

A service is used when a common functionality needs to be provided to various modules or An Angular service is a singleton - which means it is instantiated only ONCE. It is also injectable which means it can be used throughout your application. It is instantiated at the start of the application and is available throughout the lifetime of the application. These three key features are very important in an Angular application. Services are commonly used for storing data and making HTTP calls.

If you need to share data between components then you can use a service. Because it is a singleton and is active throughout the lifetime of the application, it maintains state globally so each component will see the same model.

You can also use it to encapsulate functionality - such as asynchronous calls to external services or common functions that might be needed in more than one place.

Note: [Asynchronous vs synchronous execution, what does it really mean?](https://stackoverflow.com/questions/748175/asynchronous-vs-synchronous-execution-what-does-it-really-mean)

**When you execute something synchronously, you wait for it to finish before moving on to another task. When you execute something asynchronously, you can move on to another task before it finishes.**

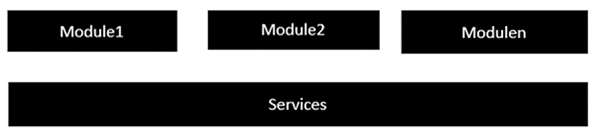
**That being said, in the context of computers this translates into executing a process or task on another "thread." A thread is a series of commands (a block of code) that exists as a unit of work. The operating system can manage multiple threads and assign a thread a piece ("slice") of processor time before switching to another thread to give it a turn to do some work. At its core (pardon the pun), a processor can simply execute a command, it has no concept of doing two things at one time. The operating system simulates this by allocating slices of time to different threads.**

**Now, if you introduce multiple cores/processors into the mix, then things CAN actually happen at the same time. The operating system can allocate time to one thread on the first processor, then allocate the same block of time to another thread on a different processor. All of this is about allowing the operating system to manage the completion of your task while you can go on in your code and do other things.**

**Asynchronous programming is a complicated topic because of the semantics of how things tie together when you can do them at the same time. There are numerous articles and books on the subject; have a look!**

Services are used to share data, communicate between components and external data sources - they are an integral part of the Angular framework.

For example, we could have a database functionality that could be reused among various modules. And hence you could create a service that could have the database functionality.



The following key steps need to be carried out when creating a service.

**Step 1** − Create a separate class which has the injectable decorator. The injectable decorator allows the functionality of this class to be injected and used in any Angular JS module.

@Injectable()

export class classname { }

Step 2 − Next in your appComponent module or the module in which you want to use the service, you need to define it as a provider in the @Component decorator.

@Component ({

providers : [classname]

})

Let�s look at an example on how to achieve this. Following are the steps involved.

**Step 1** − Create a **ts** file for the service called app.service.ts.

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

@Injectable()

export class appService {

getApp(): string {

return "Hello world";

}

}

Following points need to be noted about the above program.

* The Injectable decorator is imported from the angular/core module.
* We are creating a class called appService that is decorated with the Injectable decorator.
* We are creating a simple function called getApp, which returns a simple string called �Hello world�.

**Step 3** − In the app.component.ts file, place the following code.

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

import { appService } from './app.service';

@Component ({

selector: 'demo-app',

template: '<div>{{value}}</div>',

providers: [appService]

})

export class AppComponent {

value: string = "";

constructor(private \_appService: appService) { }

ngOnInit(): void {

this.value = this.\_appService.getApp();

}

}

Following points need to be noted about the above program.

* First, we import our appService module in the appComponent module.
* Then, we register the service as a provider in this module.
* In the constructor, we define a variable called \_appService of the type appService so that it can be called anywhere in the appComponent module.
* As an example, in the ngOnInit lifecyclehook, we called the getApp function of the service and assign the output to the value property of the AppComponent class.

Interface:

An *interface* is a TypeScript artefact. An *interface* is a way to define a *contract* on a function with respect to the arguments and their type. Along with functions, an *interface* can also be used with a Class as well to define custom types.

An interface is an abstract type, it does not contain any code as a *class* does. It only defines the 'signature' or shape of an API. During transpiration, an interface will not generate any code, it is only used by Typescript for type checking during development.

interface Callback {

(error: Error, data: any): void;

}

function callServer(callback: Callback) {

callback(null, 'hi');

}

callServer((error, data) => console.log(data)); // 'hi'

callServer('hi');

Angular material

Angular material is used for design the web page. We need to install multiple angular inbuilt components.

Use below link and follow as described in this link:

<https://material.angular.io/guide/getting-started>.

Note:

@NgModule({

declarations: [

AppComponent,

NavBarComponent,

MatToolbar,

LoginComponent

],

imports: [

BrowserModule,

BrowserAnimationsModule,

MatButtonModule,

MatCheckboxModule

],

providers: [],

bootstrap: [AppComponent]

})

# [**What is the difference between declarations, providers, and import in NgModule?**](https://stackoverflow.com/questions/39062930/what-is-the-difference-between-declarations-providers-and-import-in-ngmodule)

* imports makes the exported declarations of other modules available in the current module
* declarations are to make directives (including components and pipes) from the current module available to other directives in the current module. Selectors of directives, components or pipes are only matched against the HTML if they are declared or imported.
* providers are to make services and values known to DI (dependency injection). They are added to the root scope and they are injected to other services or directives that have them as dependency.

A special case for providers are lazy loaded modules that get their own child injector. providers of a lazy loaded module are only provided to this lazy loaded module by default (not the whole application as it is with other modules).

For more details about modules see also <https://angular.io/docs/ts/latest/guide/ngmodule.html>

* exports makes the components, directives, and pipes available in modules that add this module to imports. exports can also be used to re-export modules such as CommonModule and FormsModule, which is often done in shared modules.
* entryComponents registers components for offline compilation so that they can be used with ViewContainerRef.createComponent(). Components used in router configurations are added implicitly.

How to Install Bootstrap in project:

* + npm install –save bootstrap@3

Now alter file angular.json in this file add below line on top of style attribute:

Eg:

“styles”:[

“node\_modules/bootstrap/dist/css/.min.css”

]

Selectors In component:

There are three ways we can define selector in component:

1. As a HTML Tag/element

Eg: selector: ’app-server’,

While using in html we can use as <app-server></app-server >

1. As attribute of element:

Eg: selector : ’[app-server]’,

While using in html we can use as <div app-server></div >

1. As class selector:

Eg: selector : ’.app-server’,

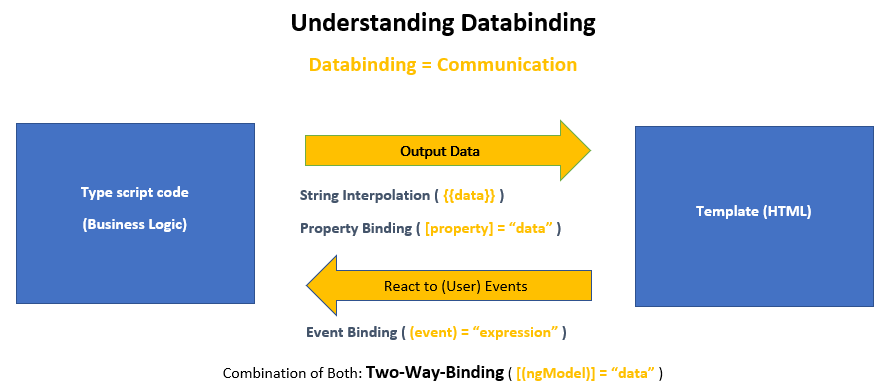
While using in html we can use as <div class=’app-server’></div >

Note: Id selector not supported for this and all sudo sector also not supported by Angular.

Databinding:

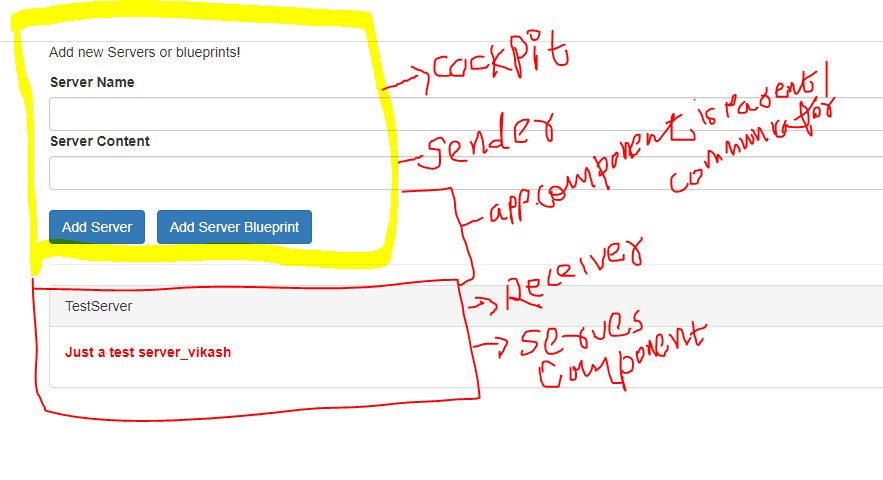
Later will be in this course

Databinding means communication between components/Typescript and HTML or template



* In Databinding @Inut() & @Output() pays major part. In this transaction Parent (app.componet.ts) is owner of data transaction. If Parent is sending data and child is receiving (Input) data then in that case we use @Input() in child (serers.componet.ts) and Child sends data for parent then in child (cockpit.componet.ts) we use @Output().

Example:



App.componet.ts

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

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit{

  serverElements = [{type: 'server', name: 'TestServer', content: 'Just a test server\_vikash'}];

  onServerAdded(serverDetails: {serverName: string, serverContent: string})

  {

    console.log("Server Content->"+serverDetails.serverContent)

    this.serverElements.push({

      type : 'server',

      name : serverDetails.serverName,

      content: serverDetails.serverContent

    });

  }

  onBlueprintAdded(blueprintDetails: {serverName: string, serverContent: string})

  {

    console.log("Blueprint Content->"+blueprintDetails.serverContent)

    this.serverElements.push({

      type : 'blueprint',

      name : blueprintDetails.serverName,

      content: blueprintDetails.serverContent

    });

  }

  ngOnInit() {

  }

}

app.component.html

<div class="container">

  <app-cockpit (serverCreated)='onServerAdded($event)' (bluprintCreated)='onBlueprintAdded($event)'></app-cockpit>

  <hr>

  <div class="row">

    <div class="col-xs-12">

         <app-server-element \*ngFor="let serverElement of serverElements" [srvElement]="serverElement"></app-server-element>

    </div>

  </div>

</div>

cockpit.component.ts

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

@Component({

  selector: 'app-cockpit',

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

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

})

export class CockpitComponent implements OnInit {

  @Output() serverCreated= new EventEmitter<{serverName: string, serverContent: string}>();

  @Output() bluprintCreated= new EventEmitter<{serverName: string, serverContent: string}>();

  newServerName = "";

  newServerContent = "";

  constructor() { }

  ngOnInit() {

  }

  onAddServer() {

    this.serverCreated.emit({

      serverName: this.newServerName,

      serverContent: this.newServerContent

    });

    }

  onAddBlueprint() {

    this.bluprintCreated.emit({

      serverName: this.newServerName,

      serverContent: this.newServerContent

    });

  }

}

cockpit.component.html

<div class="row">

    <div class="col-xs-12">

      <p>Add new Servers or blueprints!</p>

      <label>Server Name</label>

      <input type="text" class="form-control" [(ngModel)]="newServerName">

      <label>Server Content</label>

      <input type="text" class="form-control" [(ngModel)]="newServerContent">

      <br>

      <button

        class="btn btn-primary"

        (click)="onAddServer()">Add Server</button>

        &nbsp;

      <button

        class="btn btn-primary"

        (click)="onAddBlueprint()">Add Server Blueprint</button>

    </div>

  </div>

server-element.component.ts

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

@Component({

  selector: 'app-server-element',

  templateUrl: './server-element.component.html',

  styleUrls: ['./server-element.component.css']

})

export class ServerElementComponent implements OnInit {

  @Input('srvElement') element: { type: string, name: string, content: string};

  constructor() { }

  ngOnInit() {

  }

}

server-element.component.html

<div class="panel panel-default">

    <div class="panel-heading">{{ element.name }}</div>

    <div class="panel-body">

        <p>

            <strong \*ngIf="element.type === 'server'" style="color: red">{{ element.content }}</strong>

            <em \*ngIf="element.type === 'blueprint'">{{ element.content }}</em>

        </p>

    </div>

</div>

LifeCycle Hook:

Directives Deep Dive:

* 1. Attribute Directives:
     1. It looks like normal HTML element possibly with data biding.
     2. Only affected/change the element they are added to.
  2. Structural Directives:
     1. Look like a normal HTML Attribute but have a leading \* (for desugaring)
     2. Affect a whole area in the DOM (elements get added/ removed)

General Notes:

1. Keep folder/package name same as component name.
2. Selector name should be always unique.
3. Emmet is html plugin user this.
4. Directives uses \* because these changes the structure also called structural directive.
5. Math.random() Gives us a random number.

Questions:

1. Can a component exist without template?