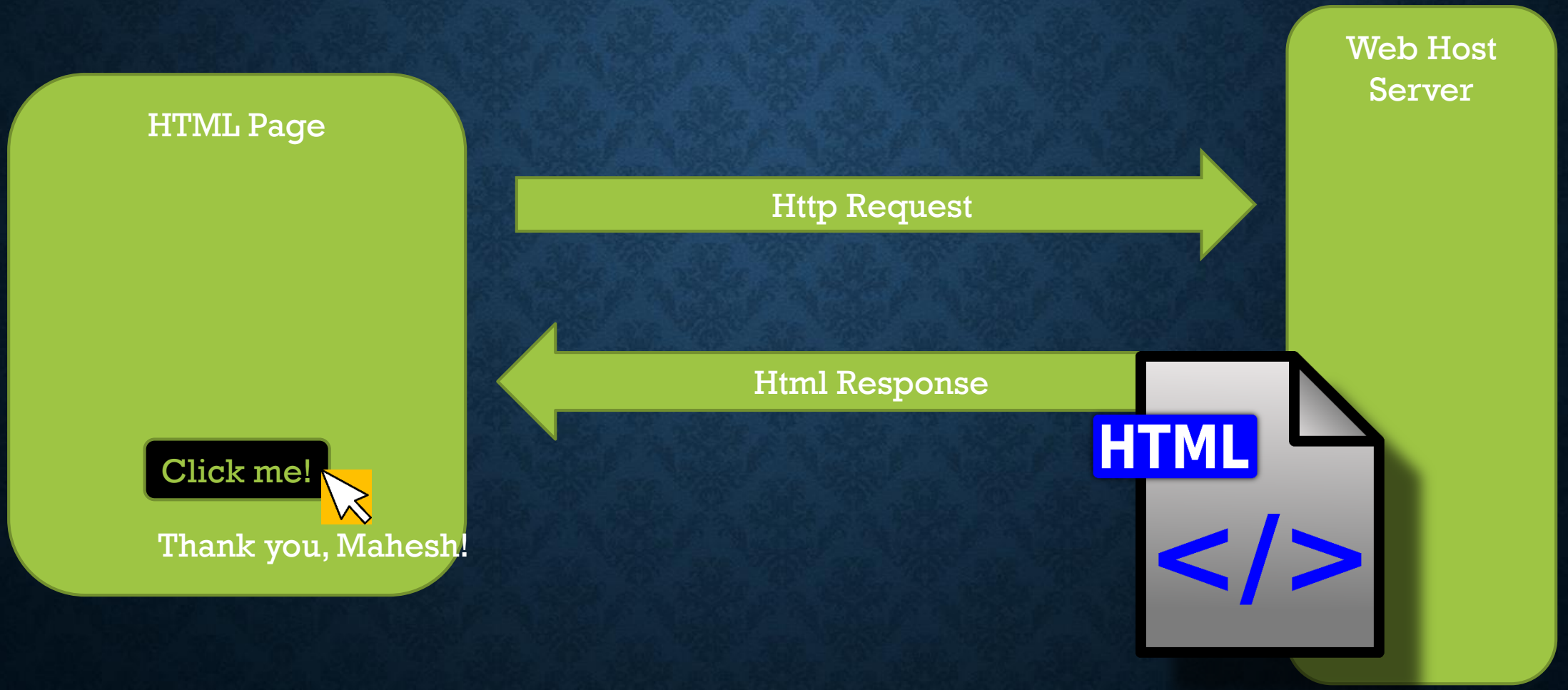
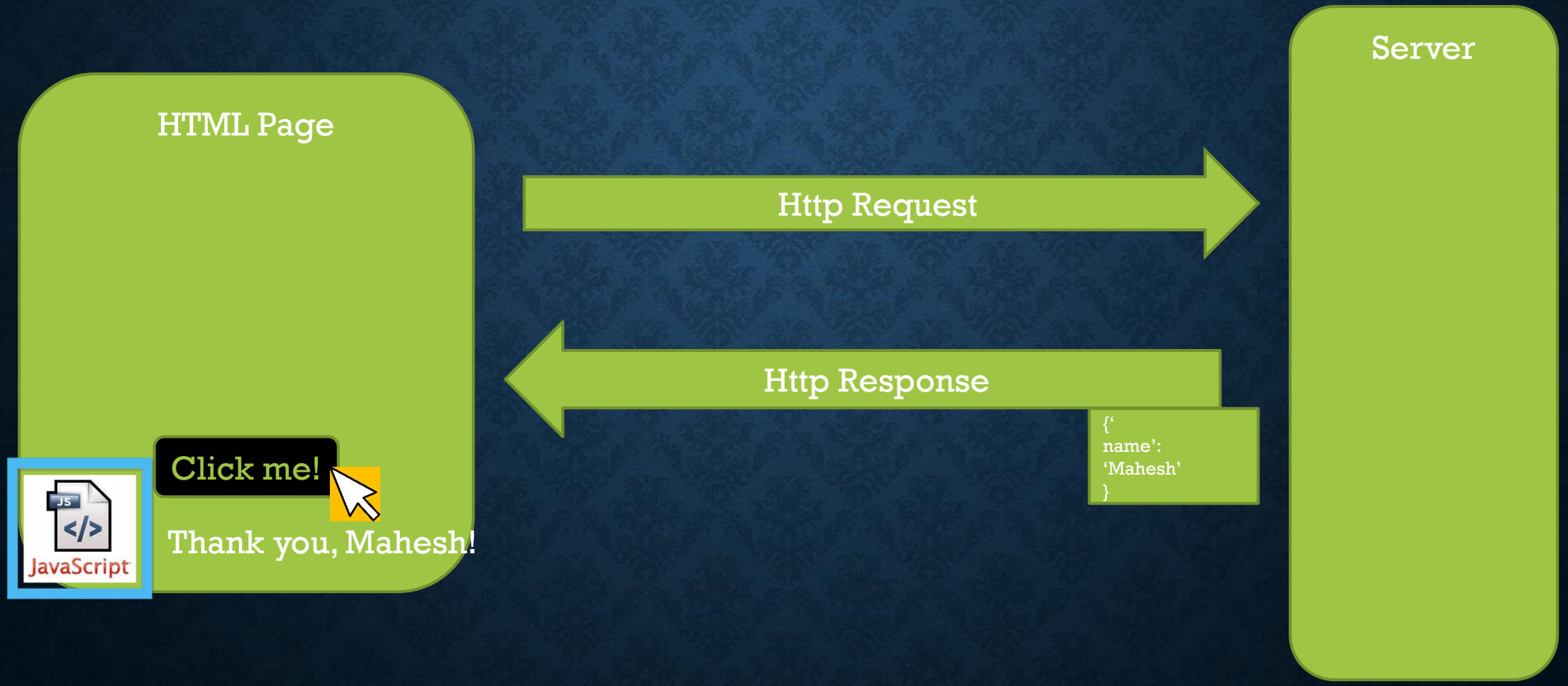
ANGULAR

- A client-side framework for developing
  - Modular Applications
  - Consumer Focused Apps
  - Applications with Component Isolations
  - Testable Applications
  - Single Page Applications (SPA)

TRADITIONAL WEB APP



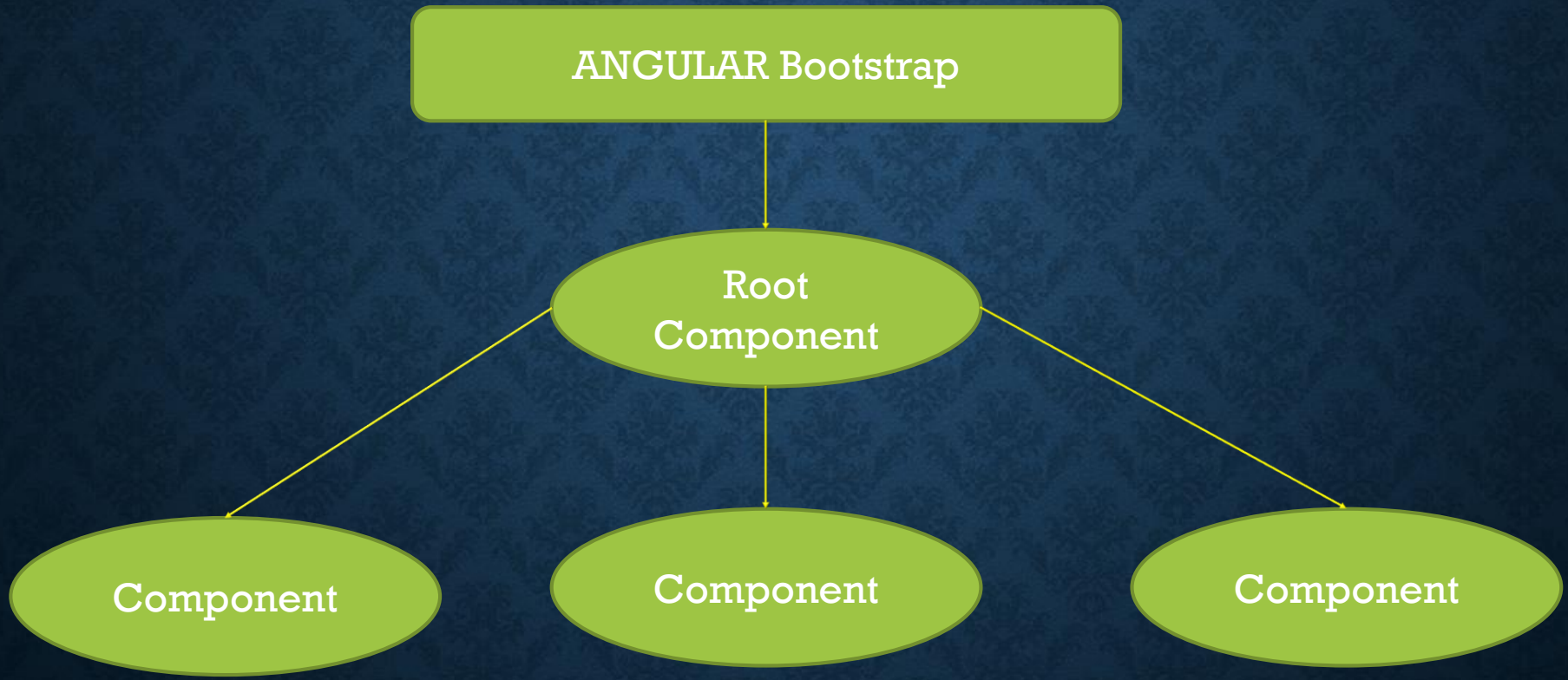
SINGLE PAGE APP



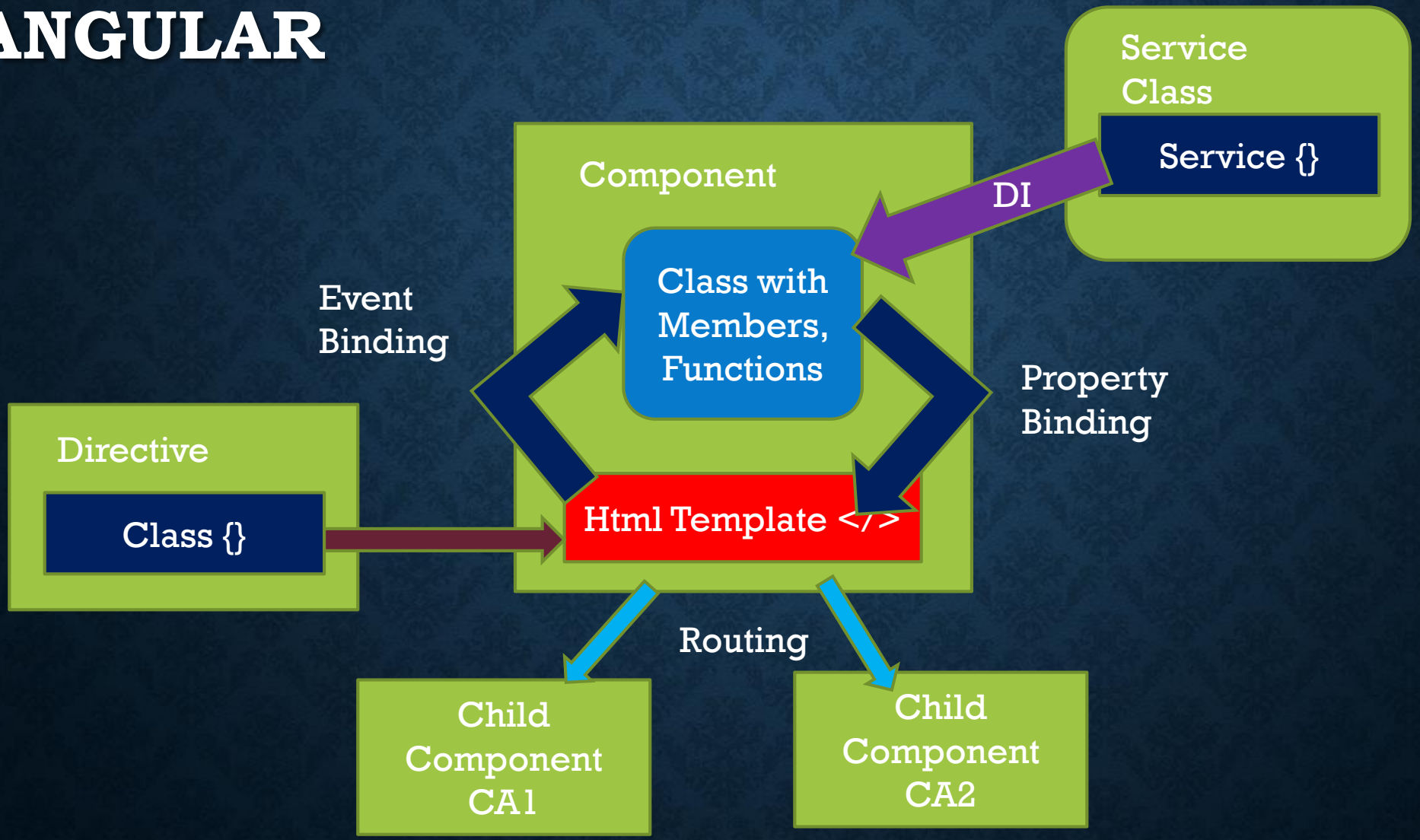
ANGULAR

- Building blocks
  - Modular System (Module)
  - Component
    - Basic Exportable Types
  - Services
    - Heavy operations to be performed behind the scene
  - Directives
    - TypeScript class with metadata
    - Everything of Angular
  - Dependency Injection
    - Container of all objects needs to be injected.

ANGULAR


ANGULAR Bootstrap
Root Component
Component
Component
Component

ANGULAR


Service
Class
Service {}
DI
Component
Class with
Members,
Functions
Event
Binding
Property
Binding
Directive
Class {}
Html Template </>
Routing
Child
Component
CA1
Child
Component
CA2

ANGULAR

- Directives
  - The metadata classes to define behavior of the DOM
  - Used by the compiler for DOM properties/Events
- Component Directives
  - Directives with Templates
  - Most Commonly used
- Structural Directives
  - Used to change DOM layout by adding/removing DOM
  - NgFor, NgIf, etc.
- Attribute Directive
  - Change Appearance/Behavior of an element.
  - NgStyle.

ANGULAR

- The Project Structure
  - tsconfig.json
  - package.json
  - systemjs.config.json
  - ServerHost
  - Boostrap
  - Component and Other Modules
  - Index.html

ANGULAR

- tsconfig.json

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "system",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  },
  "exclude": [
    "node_modules",
    "typings/main",
    "typings/main.d.ts"
  ]
}
```

ANGULAR

- package.json

```
"@angular/common": "2.0.0",
 "@angular/compiler": "2.0.0",
 "@angular/core": "2.0.0",
 "@angular/http": "2.0.0",
 "@angular/platform-browser": "2.0.0",
 "@angular/platform-browser-dynamic": "2.0.0",
 "@angular/router": "3.0.2",
 "bootstrap": "3.3.7",
 "es6-shim": "0.35.1",
 "reflect-metadata": "0.1.8",
 "rxjs": "5.0.0-beta.12",
 "systemjs": "0.19.39",
 "zone.js": "0.6.25",
```

ANGULAR

- systemjs.config.js

```
var map = {
    "rxjs": "node_modules/rxjs",
    "@angular/common": "node_modules/@angular/common",
    "@angular/compiler": "node_modules/@angular/compiler",
    "@angular/core": "node_modules/@angular/core",
    "@angular/platform-browser": "node_modules/@angular/platform-browser",
    "@angular/platform-browser-dynamic": "node_modules/@angular/platform-browser-dynamic"
};
var packages = {
    "rxjs": { "defaultExtension": "js" },
    "@angular/common": { "main": "bundles/common.umd.js", "defaultExtension": "js" },
    "@angular/compiler": { "main": "bundles/compiler.umd.js", "defaultExtension": "js" },
    "@angular/core": { "main": "bundles/core.umd.js", "defaultExtension": "js" },
    "@angular/platform-browser": { "main": "bundles/platform-browser.umd.js", "defaultExtension": "js" },
    "@angular/platform-browser-dynamic": { "main": "bundles/platform-browser-dynamic.umd.js", "defaultExtension": "js" },
    "app": {
        format: 'register',
        defaultExtension: 'js'
    }
};

var config = {
    map: map,
    packages: packages
};

System.config(config);
```

ANGULAR

- The component
  - The basic building block of the ANGULAR Application
  - @Component, typedecorator
  - @angular/core
  - Properties
    - selector
    - template
    - templateUrl
    - styleUrl
    - style

ANGULAR BOOTSTRAP

ANGULAR BOOTSTRAP

- @angular/core
  - It is a main package of ANGULAR Applications.
- @angular/platform-browser
  - It contains code shared for browser execution (DOM thread, WebWorker)
  - **Ahead-of-Time pre-compiled** version of application being sent to the browser. Which usually means a significantly smaller package being sent to the browser.
- @angular/platform-browser-dynamic
  - It contains the client side code that processes templates (bindings, components, …) and reflective dependency injection.
  - Uses **Just-in-Time compiler** and make's application compile on client-side.

ANGULAR

- The Bootstrap
- Implemented Using @NgModule
  - Decorator used to define metadata for the module
  - This metadata imports a single helper module i.e. BrowserModule, which is mandatory module for every app.
  - This also registers the critical service providers.

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import{EmployeeComponent} from './employee.component';

@NgModule({
  declarations: [EmployeeComponent],
  imports: [BrowserModule],
  bootstrap: [EmployeeComponent]
})
class AppModule { }

platformBrowserDynamic().bootstrapModule(AppModule);
```

ANGULAR

- The index.html

```
  <script src="node_modules/es6-shim/es6-shim.min.js"></script>
  <script src="node_modules/reflect-metadata/Reflect.js"></script>
  <script src="node_modules/systemjs/dist/system.src.js"></script>
  <script src="node_modules/zone.js/dist/zone.js"></script>
  <script src="systemjs.config.js"></script>
  <script>
      System.import('./app/main')
            .then(null, console.error.bind(console));
    </script>
</head>
<body>
  <div class="container">
    <h1>Angular 2 Directives</h1>
    <app>Loading...</app>
  </div>
</body>
```

ANGULAR FORMS

ANGULAR FORMS

- Forms consist of the following parts
  - DOM for rendering
  - Field Definition, a.k.a. ***Template***
  - Client-Side Logic, e.g. UI Logic, a.k.a. ***Form Model.***
  - Fields to be exposed to UI, a.k.a. ***Domain Model.***

ANGULAR FORMS

- The Unit of Submit from the WEB UI
- Form Types
  - Data-Driven Forms a.k.a. Model-Driven Forms
    - Create Model, that will be submitted
  - Template-Driven Forms
- Form Validations
  - Model Validations
  - Custom Validations

ANGULAR FORMS

- Simple Form
  - Html ***form*** tag mapped with the ***ngForm*** directive.
  - ANGULAR, internally creates ***FormControl.***
    - ***Name,*** attribute for each element is set inside the FormControls collection
  - ***onSubmit()*** function is provided by ***ngForm***
    - ***(ngSubmit)***
  - ***[ngModelOptions]="{standalone:true}"***
    - Can be defined on ***input*** element to prevent adding element in FormControl
      - Use this in standard forms
      - Not useful in case of validations

```
import { FormGroup,FormControl, FormBuilder,Validators } from '@angular/forms';
```

ANGULAR FORMS

- Import the following modules into the Component

```
import { FormGroup,FormControl, FormBuilder,Validators } from '@angular/forms';
```

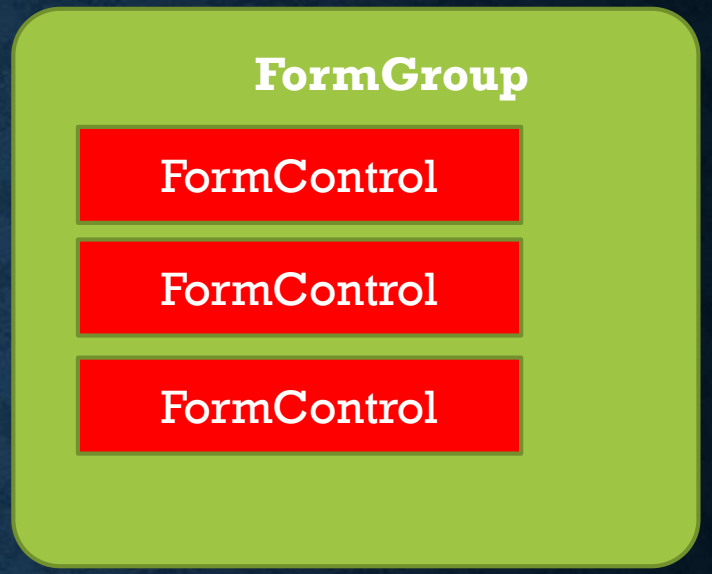
- Import the following modules into the boot.ts

```
import { ReactiveFormsModule,FormsModule  } from '@angular/forms';
```

- Import the following modules into the @NgModule

```
imports: [BrowserModule,ReactiveFormsModule,FormsModule ],
```

ANGULAR FORMS



- Data-Driven Forms
  - The form which is associated with the Domain Model
  - **FormGroup**
    - Collection of all ***form elements*** of which values will represent the ***state*** of the form.
  - **FormControl**
    - Represent an input field that provides value and validity.
    - Value of the field and the validation rules.
  - **formControlName**
    - Represent name of the form element into the **formGroup.**

```
<form (ngSubmit)="save()"
[formGroup]="form">
```

```
<input type="text"
 class="form-control" formControlName="empNo"
[(ngModel)]="emp.empNo">
```

ANGULAR FORMS

Data-Driven Forms, the declaration

```
form: FormGroup;
emp: Employee;
empNo: FormControl;
empName: FormControl;
salary: FormControl;
deptName: FormControl;
designation: FormControl;
```

FormGroup with FormControl associated with the Model Properties

```
this.form = new FormGroup({
    'empNo': new FormControl(this.emp.empNo),
    'empName': new FormControl(this.emp.empName),
    'salary': new FormControl(this.emp.salary),
    'deptName': new FormControl(this.emp.deptName),
    'designation': new FormControl(this.emp.designation)
});
```

ANGULAR FORMS

- Form Validations, using HTML attributes
  - novalidate
  - Pattern
  - Maxlength
  - Minlength
  - Required

ANGULAR FORMS

```
<td>EmpNo</td>
<td>
    <input type="text"
     class="form-control"  required
     name="empNo"
     [(ngModel)]="emp.empNo"
    formControlName="empNo"
      pattern="[0-9]+">
</td>
<div *ngIf="form.controls.empNo.dirty && !form.controls.empNo.valid"
    class="alert alert-danger">
    <p *ngIf="form.controls.empNo.errors.required">
        EmpNo is must
    </p>
    <p *ngIf="form.controls.empNo.errors.pattern">
        EmpNo must be Numeric
    </p>
 </div>
```

- form
  - The FormGroup Object.
- form.controls
  - The Collection of FormControls in the FormGroup.
- form.controls.<name>.dirty
  - The state of the <name> element is changed
- form.controls.<name>.valid
  - The Valid/Invalid state of the form field.
- form.controls.<name>.error.<validationrule>
  - The <validationrule> is executed.

ANGULAR FORM

- FormGroup Validation Rules
  - Isolated from DOM
  - Uses **Validators.compose()** method
    - The **Validators** class used for
      - Required
      - Pattern
      - Minlength
      - Maxlength

ANGULAR FORM

```
    this.form = new FormGroup({
'empNo': new FormControl(this.emp.empNo, Validators.compose([Validators.required, Validators.pattern('[0-9]+')])),
'empName': new FormControl(this.emp.empName, Validators.compose([Validators.required, Validators.minLength(2),
 Validators.maxLength(16)])),
'salary': new FormControl(this.emp.salary, Validators.compose([Validators.required, Validators.pattern('[0-9]+')])),
'deptName': new FormControl(this.emp.deptName, Validators.compose([Validators.required,
 Validators.pattern('[A-Za-z]+')])),
'designation': new FormControl(this.emp.designation, Validators.required),
```

ANGULAR FORMS

- Custom Validator
  - Create a Custom Class with the Validation logic
  - Import the class in the Component
  - Set the class as Validations Rule for the Field in the FormGroup
  - E.g.
  - 'email': new FormControl(this.emp.email, Validators.compose([Validators.required, EmailValidator.emailValidator]))

ANGULAR SERVICE

ANGULAR SERVICE

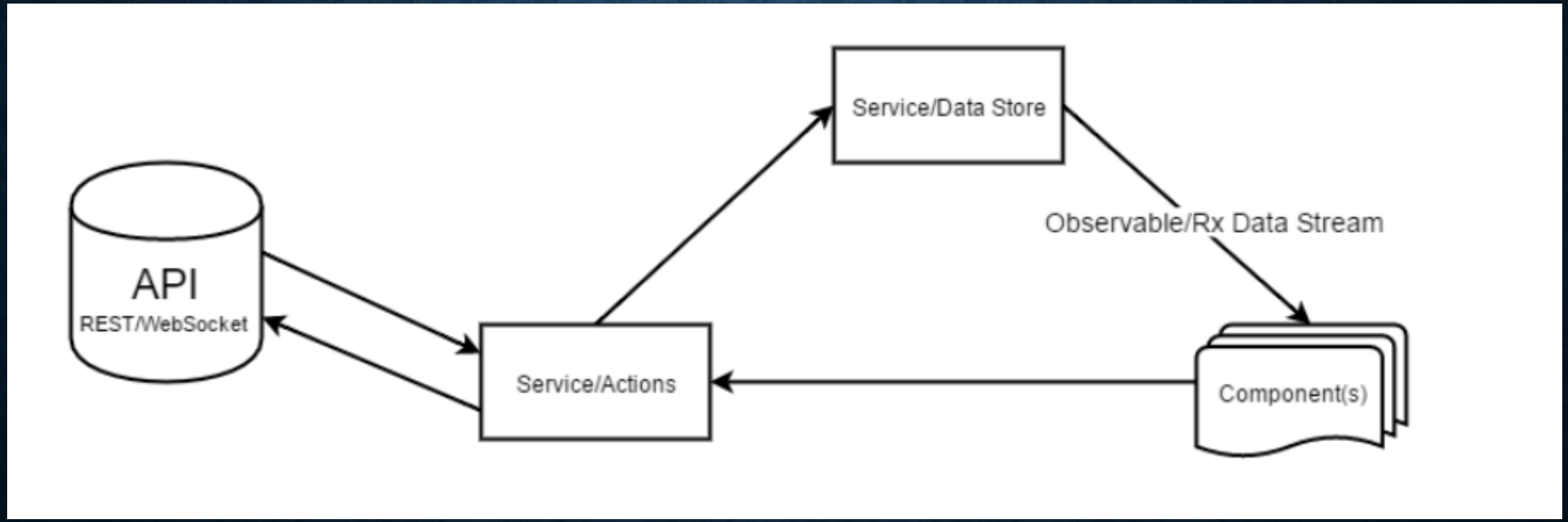
- Used to contain the heavy load operations.
- Uses @Injectable() decorator to make class as service class.
- The class must be registered in NgModule using **providers[]** property.

ANGULAR SERVICE

- A major important block of the ANGULAR.
- A class, injected with ***Http*** object from @angular/http.
- Performs following operations over ***Http***
  - Get
  - Post
  - Put
  - Delete
- All methods returns ***Observable Response***.

```
import {Http,Response,RequestOptions,Headers} from '@angular/http';
import {Observable} from 'rxjs/observable';
```

ANGULAR WITH RXJS



https://medium.com/@rachelheimbach/rxjs-state-management-in-angular-2-error-handling-678deabf3331

ANGULAR SERVICE

- Observable
  - A basic building block of the RxJs.
  - A class implements Subscribable interface.
  - A Subscriber is invokes when an Observer is initially subscribed to an operation.
- Subscribe() method
  - Provides an iteration over an observable response to deliver it the subscriber a.k.a. caller.

ANGULAR SERVICE

- Angular Service Implementation
  - @Injectable
    - Mark the class as Injectable
  - The class constructor must be injected with ***Http*** object.
  - All http methods accepts first parameter as ***URL*** of the REAT/WEBAPI service.

ANGULAR SERVICE

```
constructor(private http:Http) { }
```

```
private serviceUrl = 'http://localhost:8516/api/EmployeeInfoeAPI';
```

```
getEmployees():Observable<Response>{
        return  this.http.get(this.serviceUrl);
}
```

```
addEmployee(emp:Employee): Observable<Response>{
      let headers = new Headers({ 'Content-Type': 'application/json' });
     let options = new RequestOptions({ headers: headers });
    return this.http.post(this.serviceUrl,JSON.stringify(emp),options);
}
```

ANGULAR SERVICE

```
editEmployee(emp: Employee): Observable<Response> {
    let headers = new Headers({ 'Content-Type': 'application/json' });
    let options = new RequestOptions({ headers: headers });
    let id=emp.EmpNo;
    return this.http.put(`${this.serviceUrl}/${id}`,
    JSON.stringify(emp),options);
}
```

```
deleteEmployee(id: number): Observable<Response> {
    return this.http.delete(`${this.serviceUrl}/${id}`);
}
```

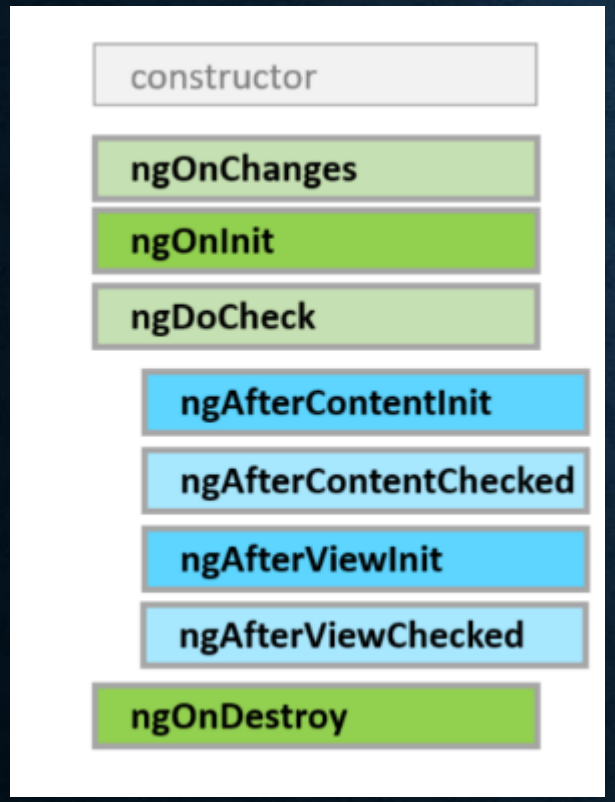
ANGULAR SERVICE

The Service Consumer

```
private loadEmployees(){
    this.empSvc.getEmployees()
                .subscribe((response:Response)=>{
                 this.Employees=response.json();
                })
}
```

ANGULAR LIFECYCLE HOOKS

ANGULAR LIFECYCLE HOOKS



A component has a lifecycle managed by Angular itself.

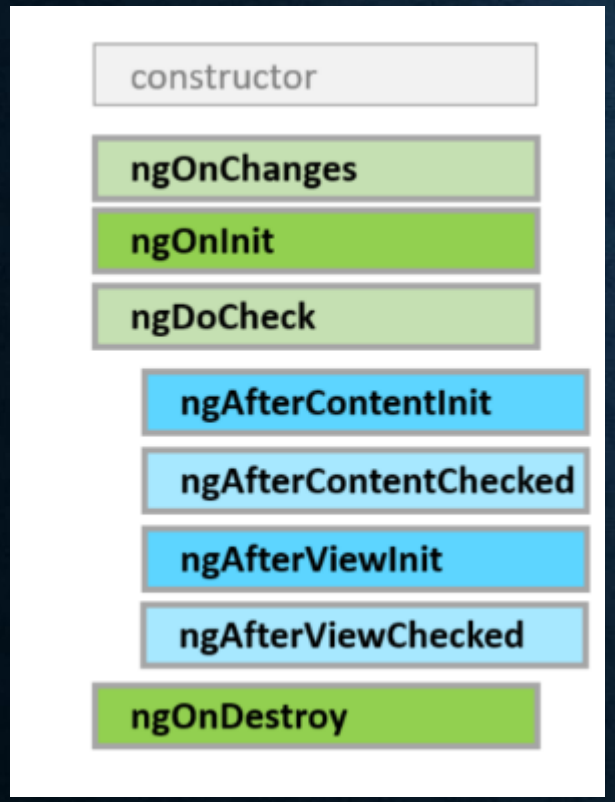
Angular creates it, renders it, creates and renders its children, checks it when its data-bound properties change, and destroys it before removing it from the DOM.

Angular offers **lifecycle hooks** that provide visibility into these key life moments and the ability to act when they occur.

A directive has the same set of lifecycle hooks, minus the hooks that are specific to component content and views.

https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html

ANGULAR LIFECYCLE HOOKS



A component has a lifecycle managed by Angular itself.

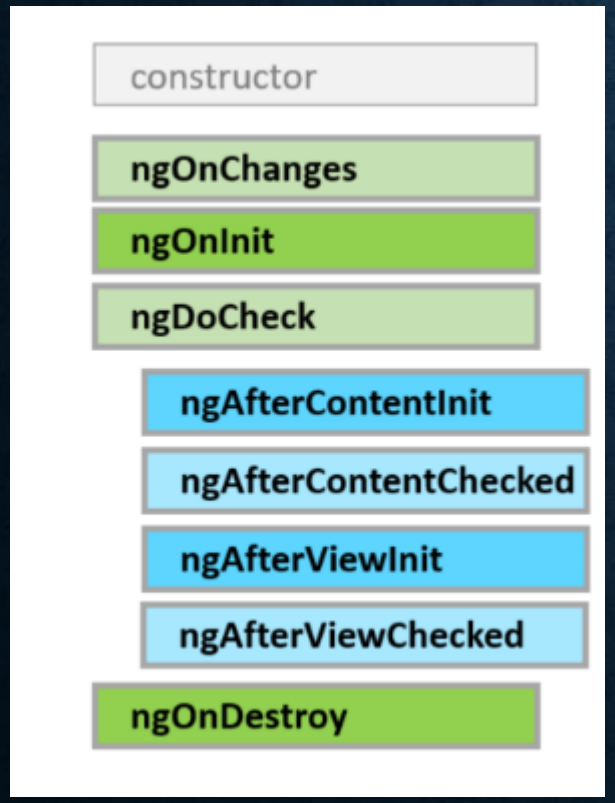
Angular creates it, renders it, creates and renders its children, checks it when its data-bound properties change, and destroys it before removing it from the DOM.

Angular offers **lifecycle hooks** that provide visibility into these key life moments and the ability to act when they occur.

A directive has the same set of lifecycle hooks, minus the hooks that are specific to component content and views.

https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html

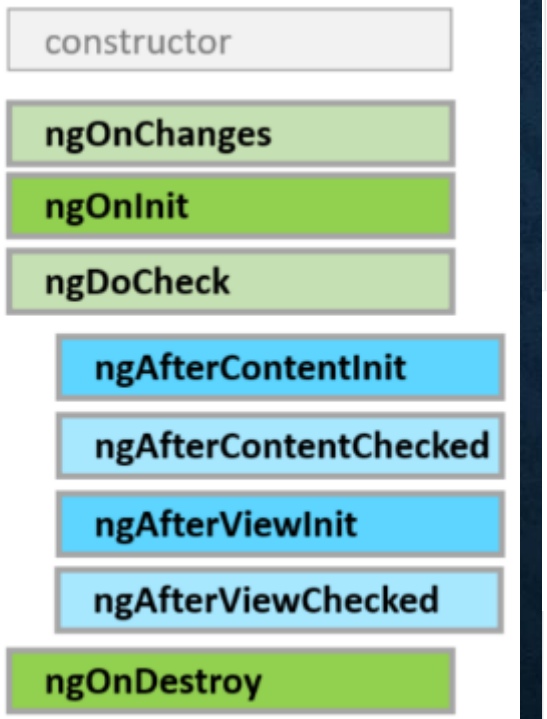
ANGULAR LIFECYCLE HOOKS



ngOnChanges	Respond when Angular (re)sets data-bound input properties. The method receives a `SimpleChanges` object of current and previous property values. Called before `ngOnInit` and whenever one or more data-bound input properties change.
ngOnInit	Initialize the directive/component after Angular first displays the data-bound properties and sets the directive/component's input properties. Called *once*, after the *first* `ngOnChanges`.
ngDoCheck	Detect and act upon changes that Angular can't or won't detect on its own. Called during every change detection run, immediately after `ngOnChanges` and `ngOnInit`.

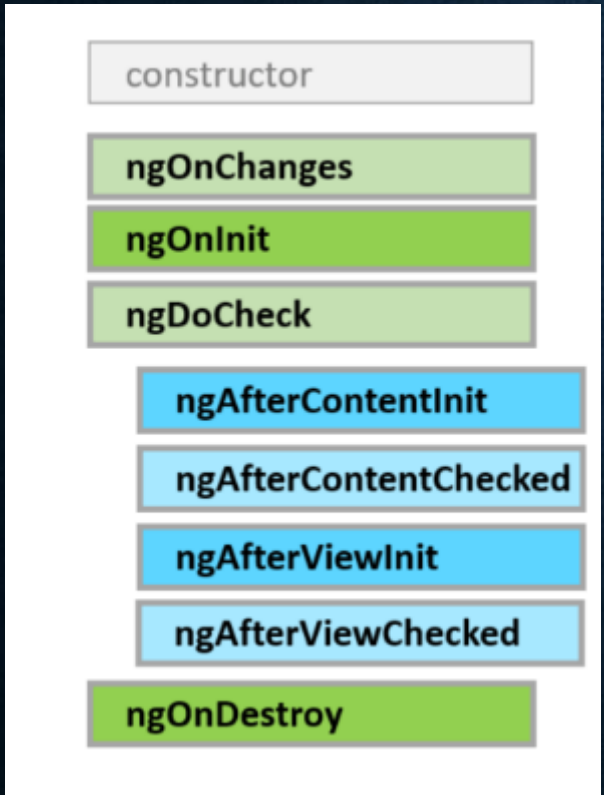
https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html

ANGULAR LIFECYCLE HOOKS



ngAfterContentInit	Respond after Angular projects external content into the component's view. Called *once* after the first `NgDoCheck`. *A component-only hook*.
ngAfterContentChecked	Respond after Angular checks the content projected into the component. Called after the `ngAfterContentInit` and every subsequent `NgDoCheck`. *A component-only hook*.
ngAfterViewInit	Respond after Angular initializes the component's views and child views. Called *once* after the first `ngAfterContentChecked`. *A component-only hook*.

ANGULAR LIFECYCLE HOOKS



ngAfterViewChecked	Respond after Angular checks the component's views and child views. Called after the `ngAfterViewInit` and every subsequent `ngAfterContentChecked`. *A component-only hook.*
ngOnDestroy	Cleanup just before Angular destroys the directive/component. Unsubscribe observables and detach event handlers to avoid memory leaks. Called *just before* Angular destroys the directive/component.

ANGULAR COMPONENT COMMUNICATION

ANGULAR PARENT-CHILED COMPONENTS

- Used when two components wants to communicate with each-other.
- Use @Input () decorator
  - for the Child-Component which accepts data from the parent.
- Use @Output() decorator
  - for emitting event from child to parent

ANGULAR PIPES

ANGULAR PIPES

- Pipes transform displayed values within a template.
- Built-In pipes
- Parameterized Pipes
- Custom Pipes

ANGULAR PIPES

- Built-In pipes
- DatePipe
- UpperCasePipe/LowerCasePipe
- CurrencyPipe
- DecimalPipe
- JsonPipe
- SlicePipe
- AsyncPipe

Syntax
{{<Expressiom> | <Pipe>}}

{{fullName | uppercase}}

** Income in USD: {{income | currency:'USD':true}} **

** Value : {{value | percent}} **

** {{name}} (1:10){{name | slice:1:10}} **

{{product | json}}

ANGULAR PIPES

- Income in USD: {{income | currency:'USD':true}}
- {{ (joiningDate|date:'fullDate') | lowercase}}
- {{product|json}}
- {{name}} (1:10){{name | slice:1:10}}

ANGULAR PIPES

- Custom Pipe
- Uses @Pipe decorator with name property
- The class implements the PipeTransform and implements its transform method

ANGULAR PIPE

CUSTOM PIPE

```
import {Pipe,PipeTransform} from '@angular/core';
@Pipe({
    name:'productFilter'
})
export class ProductPipe implements PipeTransform{
    transform(value:any,args:string[]){
    let filterValue = args;
     if(filterValue){
         return value.filter(p=>p.productName.toLowerCase().indexOf(filterValue)!=-1);
     }else{
        return value;
     }
    }
}
```

ANGULAR TESTING

ANGULAR TESTING

- The framework is developed for Testability.
- Uses the Testing Libraries provided in the framework.
- Additional Testing Support from Jasmine
- The @angular/core/testing provides
  - TestBed
  - Inject
  - ComponentFixture
  - async

ANGULAR TESTING

- Configuration

```
"@angular/core/testing":"node_modules/@angular/core/bundles",
"@angular/platform-browser":"node_modules/@angular/platform-browser",
"@angular/platform-browser/testing":"node_modules/@angular/platform-browser/bundles",
"@angular/platform-browser-dynamic":"node_modules/@angular/platform-browser-dynamic",
"@angular/platform-browser-dynamic/testing":"node_modules/@angular/platform-browser-dynamic/bundles"
```

```
'@angular/core/testing':{"main":"core-testing.umd.js", "defaultExtension": "js"},
"@angular/platform-browser":{"main":"bundles/platform-browser.umd.js","defaultExtension": "js"},
'@angular/platform-browser/testing':{"main":"platform-browser-testing.umd.js","defaultExtension":"js"},
"@angular/platform-browser-dynamic":{"main":"bundles/platform-browser-dynamic.umd.js","defaultExtension":"js"},
"@angular/platform-browser-dynamic/testing":{"main":"platform-browser-dynamic-testing.umd.js","defaultExtension":"js"},
```

```
"jasmine-core": "2.4.1",
"jasmine-spec-reporter": "2.5.0",
```

ANGULAR TESTING

- import {inject,TestBed,ComponentFixture,async } from '@angular/core/testing';
- TestBed.initTestEnvironment(BrowserDynamicTestingModule, platformBrowserDynamicTesting());
  - This uses **BrowserDynamicTestingModule** class and **platformBrowserDynamicTesting().**
  - This is used to load an instance of @NgModule created using **PlatformRef** class.

ANGULAR TESTING

- TestBed.configureTestingModule({

  imports:[<Array of Standard Modules>],

  declarations: [<Array of Components to be loaded for Testing>],

  });
- TestBed.createComponent(<Component Name>);
  - used to compile the component
  - Returns an instance of the **ComponentFixture** class.
  - Componentinstance
    - Property to create an instance of the component

ANGULAR ROUTING

ANGULAR ROUTING

- Load @angular/router package to provide the Routing
- Router Table
  - Array of route paths
    - [
      - {path='<URiTemplate>', component=<Name of the NGComponent>},
      - {path='UtlTemplate/:id',component=''}
    -]

ANGULAR ROUTING

- Inject the Router
  - Import {Router,RouterModule} from @angular/router
  - Component
    - Constructor(route:Router,rourep:ActivateRouter)
    - Route.navigate([‘Path of UriTemplate’])
- <base URL> to map with the default Route Value
  - <base href="/" />

ANGULAR ROUTING

- MainCOmponent
  - Html Template with
    - routeLink=[UriTemplate]
      - Query to Router Table based on UriTemplate and fetch the Component map with UriTemplate.
  - <router-outlet></router-outlet>
      - Used to get Injected with Template/TemplateUrl of the Component.

ANGULAR ROUTING

- Parameterized Routing
  - ActivateRoute
    - .params
      - Used to Read the Parameter value provided through the URL

POLYMER.JS

WEBCOMPONENT

- Web Components are
  - a collection of standards which are working their way through the W3C and landing in browsers.
  - They allow to bundle markup and styles into custom HTML elements.
  - The amazing part is that these new elements is are fully encapsulate all of their HTML and CSS.

POLYMER.JS

- **Polymer** is a library for creating
  - Web Components a.k.a. Custom Elements
  - They can then be used to build a webapp.

ANGULAR VS POLYMER

- Angular is complete Front-End framework containing
  - Modules
  - Components
  - Services
  - Directives (Custom Directives)
- Polymer is a library for creating WebComponent.
- Polymer Components, encapsulate JS,CSS,HTML

POLYMER

```
<dom-module id="element-name">

  <template>
    <style>
      /* CSS rules for your element */
    </style>

    <!-- local DOM for your element -->

    <div>{{greeting}}</div> <!-- data bindings in local DOM -->
  </template>

  <script>
    // element registration
    Polymer({
      is: "element-name",

      // add properties and methods on the element's prototype

      properties: {
        // declare properties for the element's public API
        greeting: {
          type: String,
          value: "Hello!"
        }
      }
    });
  </script>

</dom-module>
```

POLYMER VS ANGULAR

```
<polymer-element name="user-gravatar" attributes="email">
 <template>
  <img src="https://secure.gravatar.com/avatar/{{gid}}" />
 </template>
 <script>
  Polymer('user-gravatar', {
   ready: function() {
    this.gid = md5(this.email);
   }
  });
 </script>
</polymer>
```

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

@Directive({ selector: '[myTextHighlighter]' })
export class TextHighLighterDirective {
   constructor(el: ElementRef, renderer: Renderer) {
      renderer.setElementStyle(el.nativeElement,
      'backgroundColor', 'red');
   }
}
```