**Angular**

* Frontend/Client side JavaScript Framework
* Created & mantained by Google
* Used to build powerfull single page applications (SPAs)

2 ways to install:

* Angular CLI
* Node.js & NPM

Package.json

* app manifest: name, version, licence
* npm scripts
* dependencies & devDependencies

**Why use Angular**

* Rapid development & code generation
* Code organization & productivity
* Dynamic content
* Cross platform
* Unit testing ready

**Components**

* Basic building blocks of the UI. An Angular app is a tree of Angular Components.
* Decorators allow us to mark a class as an Angular component & provide metadata that determines how the component should be processed, instantiated and used at runtime.

**Services**

* Classes that implement appp logic as separate layer.
* Classes that share data and functionality across components.
* Ideal place for AJAX calls.
* Naming convention – {name}.service.ts

**Directives**

1. **\*ngFor** – loop through collections

* *\*ngFor=”let* ***obj*** *of objects, let* ***idx*** *= index; first as* ***isFirstElem****; last as* ***isLastElem****; odd as* ***isOddElem****; even as* ***isEvenElem****”*)  
  *… {{****idx****}}. {{****obj****}} … [{{****isFirstElem****}}] … [{{****isLastElem****}}]*

1. **\*ngIf** – displays/removes (from DOM) http element conditionally

* *\*ngIf=”booleanProperty”*
* *\*ngIf=”booleanProperty; else elseBlock” where elseBlock = ng-template tag (below)*
* *\*ngIf=”booleanProperty; then thenBlock; else elseBlock” where thenBlock & elseBlock = ng-template tag (below)*

1. **[ngSwitch]** – clasic switch statement (compares multiple values)

* *[ngSwitch]=”propertyName”  
  \*ngSwitchCase=”value1”  
  \*ngSwitchCase=”value2”  
  \*ngSwitchDefault*

1. **ngClass** – assigns css classes to html element
2. **ngModel** – for 2-way data binding
3. **<ng-template #templateName>…</ng-template>** - named container for other elements, that other ng directives (ex.ng-if) can use.

**Routing**

* Needs RouteModule & Routes
* Maps app paths to components
* <router-outlet>

**Data binding**

1. **Interpolation** (only string values) /component -> template/

* Dynamic value in html template {{}}
* *{{propertyName | expression | javascript code}}*

1. **Events** /template -> component/

* *(click) = „onClick($event)”* (click = mouse event)

1. **Property binding** /component -> template/

* Html attribute – initialize DOM property, can not change
* DOM property – current value, can change
* We are binding to DOM property
* *[property] = „propertyName”*
* *bind-property = „propertyName”*
* *property = „{{propertyName}}”* (only for strings)

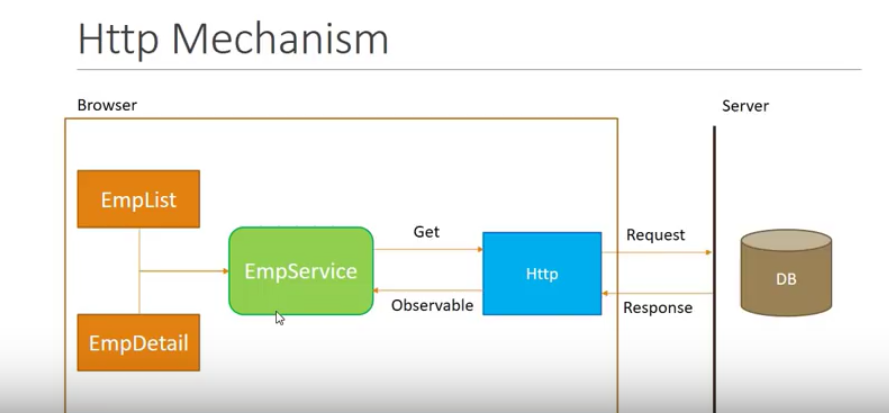
1. **2 way binding** /component <-> template/

* template reference variables – to easily access DOM elements
* *<input #myInput> -> myInput.value*
* import FormsModule
* *[(ngModel)] = „propertyName”*

**Component Interaction**

* **@Input()** decorator – the child component can accept data from parent component   
  Parent component:  
   + property in component: public **name** = „aaa”  
   + attribute in template: <comp-selector [**parentData**]=”**name**”>…  
  Child component:  
   + property in component: @Input() public **parentData**;  
   + bind it in tempate: <h2>Hello {{**parentData**}}</h2>  
  or  
   + property in component: @Input(„**parentData**”) public **newName**;  
   + bind it in template: <h2>Hello {{**newName**}}</h2>
* **@Output()** decorator – the child component will send data to parent component using **events**  
  Child Component:  
   + event in component: @Output() public **childEvent** = new EventEmitter();  
   + event source in template: <button (click)=”**fireEvent**”> Send event </button>  
   + method in component: **fireEvent**() { this.**childEvent**.emit(„Message”); }  
  Parent component:  
   + property in component: public **message**: string;  
   + bind it in template: <comp-selector (**childEvent**)=”**message**=$event

**Http & Observables**



* **Observable** – the http response (sequence of items) that arrive asynchronously over time.
* **Steps**:

1. fire http request from service (HttpClient from HttpclientModule)
2. receive the observable and cast it into array
3. subscribe to the observable from components
4. assign an array to a local variable

* **RxJS** – reactive extension for JavaScript, external library to work with Observables.

**Pipes**

* Pipes alow us to transform data before displaying then in the view (the original property value is not changed)
* *{{ name | lowercase }}*
* for strings: *lowercase, uppercase, titlecase, slice:3:5, json*
* for numbers: *number:’1.2-3’, percent, currency:’PLN’:code*
* for dates: *date:’short’, date:’mediumDate’, date:’longTime’*

**Lifecycle hooks**

**Dependency Injection**

* **DI** is a coding pattern in which a class receives its dependencies from external sources rather than creating them itself.
* Angular **DI Injector** = **DI Container** – place where are stored all the dependencies
* Process:

1. define **dependency class** (with **@Injectable()** decorator),
2. register it with **Injector** (mainly on most upper level of components structure - AppModule),
3. declare as dependency in **dependent classes** (constructor parameters - private or public)

**TypeScript**

* Superset of JavaScript with added festures
* Created by Microsoft
* Optional static typing
* Class based object-oriented programming
* Resemples languages like C# and Java