Angular Cheat Sheet

Angular CLI

- **npm install -g @angular/cli** : This command will install the Angular CLI.
- **npm install -g @angular/cli@latest**: This command will install the last Angular CLI.
- **npm uninstall -g @angular/cli@latest**: This command will uninstall the last Angular CLI.
- **npm install:** Install packages from packages.json
- **npm start:** Start project
- **ng version:** Information about angular installation
- **ng new <application name> :** Create new app.
- **ng new --help:** This returns all available Angular command lists.
- **ng lint my-app:** This command checks our entire application for any linting warnings.
- **ng lint my-app --fix:** If there are any form of linting errors, this command will fix it.
- **ng lint my-app --format stylish** : This formats our entire codebase.
- **ng lint my-app --help:** This command returns all the available linting command lists.
- **ng add <package name>:** This command will use your package manager to download new dependencies and update the project with configuration changes.
- **ng generate [directive|pipe|service|guard|interface|enum|module] <name>:** Create differents ng elements
- **ng generate class <destination><name> :** This will create a new class in the specified directory.
- **ng build:** Builds the application for production and stores it in the dist directory.
- **ng serve -o**: Serves the application by opening up the application in a browser using any port 4200 or any available port.
- **ng serve -o –port [port]**: Serves the application by opening up the application in a browser using indicate port
- **ng serve -ssl:** Serves the application using SSL
- **ng test or npm run test:** Run test
- **ng e2e or npm run e2e**: Run e2e test

Update versions of packages

1. npm i -g npm-check-updates (It is recommended delete package-lock.json file)
2. npx npm-check-updates -u
3. npm install (It is recommended to delete node_modules folder if it exists and It is recommended to check the typescript version, sometimes the latest version is not stable and causes problems, so it is advisable to leave the existing version in the packages.json file.)
4. We can use npm audit to verify packages installation.

Bindings

- Interpolation: {{ value }}
  - Binds data directly into templates.
- Event Binding: (click)="onClick()"
  - Handles events triggered by the user.
- Property Binding: [src]="imagePath"
  - Sets property values within the template.
- Two-Way Binding: [(ngModel)]="userInput"
  - Synchronizes data between view/model.

Routing and Guards

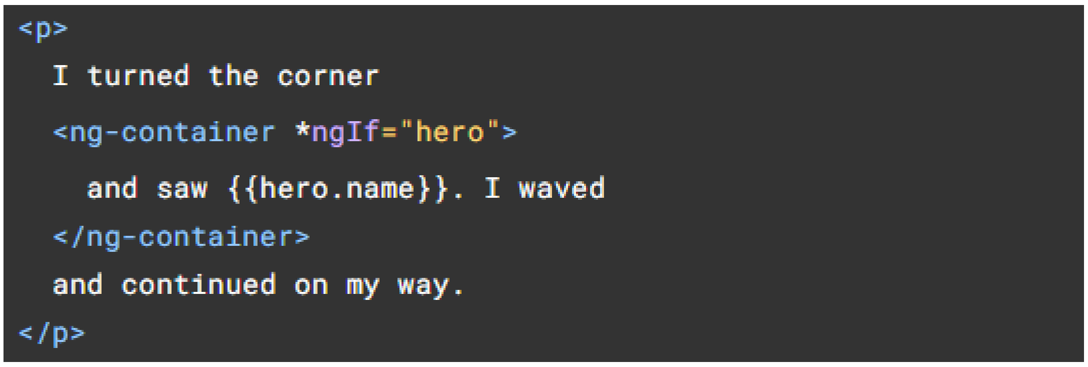
- Guards: canActivate, canLoad
  - Controls access to routes/modules.
- canActivate: canActivate: [AuthGuard]
  - Check route access before navigation.
- canLoad: canLoad: [AuthGuard]
  - Restricts module loading with conditions.

Built-in attribute directives (Import CommonModule in the component)

- ngClass: `<div [ngClass]="isSpecial ? 'special' : ''">This div is special</div>`
  - Add or remove multiple CSS classes simultaneously
- ngStyle: `<div [ngStyle]="{'font-style': this.canSave ? 'italic' : 'normal'}">`
  - Setting inline styles
- ngModel: `<input [(ngModel)]="currentItem.name" id="example-ngModel">`
  - Facilitates two-way data binding easily. Use the NgModel directive to display a data property and update that property when the user makes changes.

Hosting a directive without a DOM element

- Ng Content:

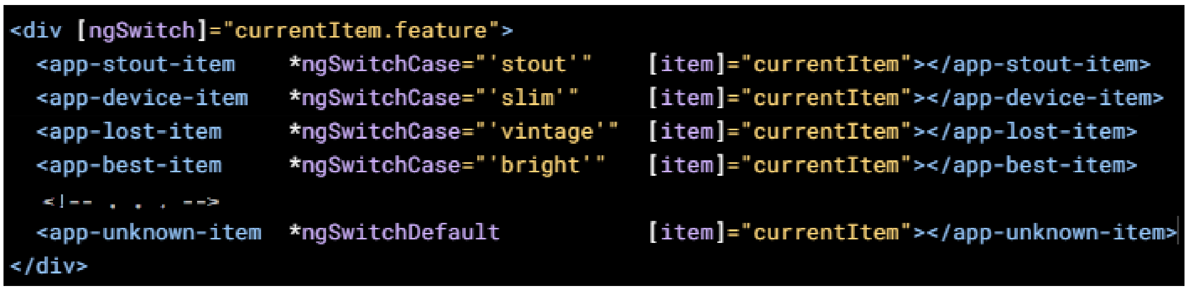


```
<p>
  I turned the corner
  <ng-container *ngIf="hero">
    and saw {{hero.name}}. I waved
  </ng-container>
  and continued on my way.
</p>
```

  - The Angular <ng-container> is a grouping element that doesn't interfere with styles or layout because Angular doesn't put it in the DOM.

Built-in structural directives (Import CommonModule in the component)

- ngIf: `<app-item-detail *ngIf="isActive" [item]="item"></app-item-detail>`
  - Adding or removing an element with NgIf
- ngFor: `<div *ngFor="let item of items; let i=index">{{i + 1}} - {{item.name}}</div>`
  - Use the NgFor directive to present a list of items.
- ngSwitch:



```
<div [ngSwitch]="currentItem.feature">
  <app-stout-item    *ngSwitchCase="'stout'"    [item]="currentItem"></app-stout-item>
  <app-device-item   *ngSwitchCase="'slim'"     [item]="currentItem"></app-device-item>
  <app-lost-item     *ngSwitchCase="'vintage'"  [item]="currentItem"></app-lost-item>
  <app-best-item     *ngSwitchCase="'bright'"   [item]="currentItem"></app-best-item>
  <!-- . . . -->
  <app-unknown-item  *ngSwitchDefault           [item]="currentItem"></app-unknown-item>
</div>
```

  - Conditionally displays matching elements.

New Built-in structural directives

- @if:

```
@if (a > b) {
  {{a}} is greater than {{b}}
} @else if (b > a) {
  {{a}} is less than {{b}}
} @else {
  {{a}} is equal to {{b}}
}
```

  - The @if block conditionally displays its content when its condition expression is truthy.

- @for:

```
@for (item of items; track item.name) {
<li>{{ item.name }}</li>
} @empty {
<li>There are no items.</li>
}
```

  - The @for block repeatedly renders content of a block for each item in a collection.

- @swtich:

```
@switch (condition) {
  @case (caseA) {
    Case A.
  }
  @case (caseB) {
    Case B.
  }
  @default {
    Default case.
  }
}
```

  - The @switch block is inspired by the JavaScript switch statement:

- @let:

```
@let user = user$ | async;
<h1>Hello, {{user.name}}</h1>
```

  - @let allows you to define a local variable and re-use it across the template.

- @defer:

```
@defer ( on <trigger>; when <condition>; prefetch on <trigger>; prefetch when <condition> ) {
  <!-- deferred template fragment -->
  <calendar-cmp />
} @placeholder ( minimum? <duration> ) {
  <!-- placeholder template fragment -->
  <p>Placeholder</p>
} @loading ( minimum? <duration>; after? <duration> ) {
  <!-- loading template fragment -->
  <img alt="loading image" src="loading.gif" />
} @error {
  <!-- error template fragment -->
  <p>An loading error occurred</p>
}
```

  - A type of block that can be used to defer load the JavaScript for components, directives and pipes used inside a component template.

Pipes

- Pipes are functions that are used to transform data in templates. In general, pipes are "pure" functions that don't cause side effects. Angular has a number of helpful built-in pipes you can import and use in your components. You can also create a custom pipe.

  Normally to use the pipes we have to indicate them in the imports of the component.

```
@Component({
    ...
    imports: [LowerCasePipe]
})
```

  Some of Angular's pipes:

```
AsyncPipe       {{ obj_expression | async }}
DatePipe        {{ value_expression | date [ : format [ : timezone [ : locale ] ] ] }}
I18nPluralPipe  {{ value_expression | i18nPlural : pluralMap [ : locale ] }}
JsonPipe        {{ value_expression | json }}
LowerCasePipe   {{ value_expression | lowercase }}
SlicePipe       {{ value_expression | slice : start [ : end ] }}
UpperCasePipe   {{ value_expression | uppercase }}
CurrencyPipe    {{ value_expression | currency [ : currencyCode [ : display [ : digitsInfo [ : locale ] ] ] ] }}
DecimalPipe     {{ value_expression | number [ : digitsInfo [ : locale ] ] }}
I18nSelectPipe  {{ value_expression | i18nSelect : mapping }}
KeyValuePipe    {{ input_expression | keyvalue [ : compareFn ] }}
PercentPipe     {{ value_expression | percent [ : digitsInfo [ : locale ] ] }}
TitleCasePipe   {{ value_expression | titlecase }}
```

Decorators

Angular exports dozens of decorators that can be applied to classes and fields. These are some of the most common decorators you'll come across.

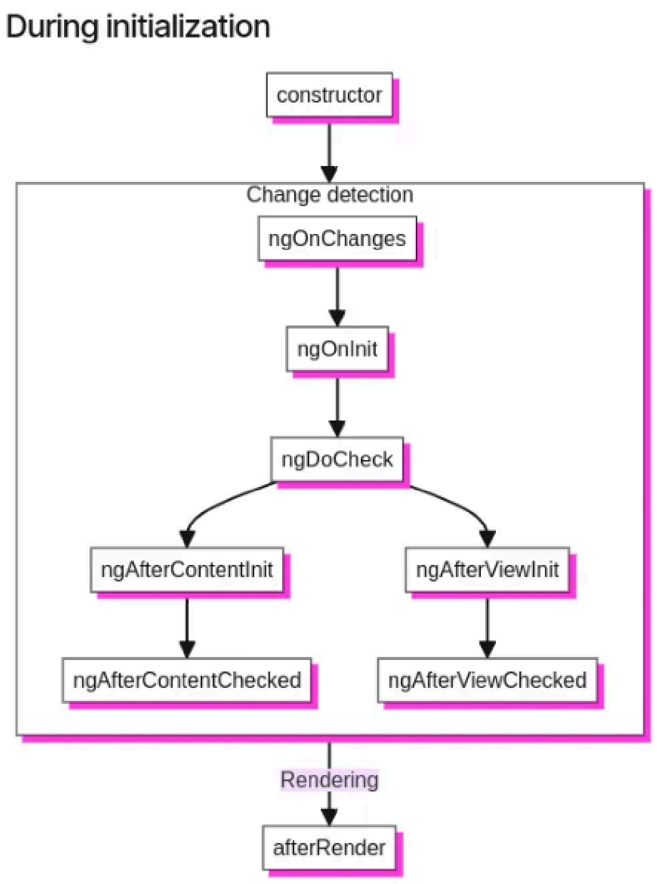
Decorator	Example	Description
@Input()	@Input() myProperty	A property can be updated through property binding.
@Output()	@Output() myEvent = new EventEmitter();	A property that can fire events and can be subscribed to with event binding on a component.
@HostBinding()	@HostBinding('class.valid') isValid	Binds a host element property (here, the CSS class valid) to a directive/component property (isValid).
@HostListener()	@HostListener('click', ['$event']) onClick(e) {...}	A directive for subscribing to an event on a host element, such as a click event, and run a method when that event is emitted. You can optionally accept the $event object.
@ContentChild()	@ContentChild(myPredicate) myChildComponent;	Binds the first result of the component content query (myPredicate) to a property (myChildComponent) of the class.
@ContentChildren()	@ContentChildren(myPredicate) myChildComponents;	Binds the results of the component content query (myPredicate) to a property (myChildComponents) of the class.
@ViewChild()	@ViewChild(myPredicate) myChildComponent;	Binds the first result of the component view query (myPredicate) to a property (myChildComponent) of the class. Not available for directives.
@ViewChildren()	@ViewChildren(myPredicate) myChildComponents;	Binds the results of the component view query (myPredicate) to a property (myChildComponents) of the class. Not available for directives.

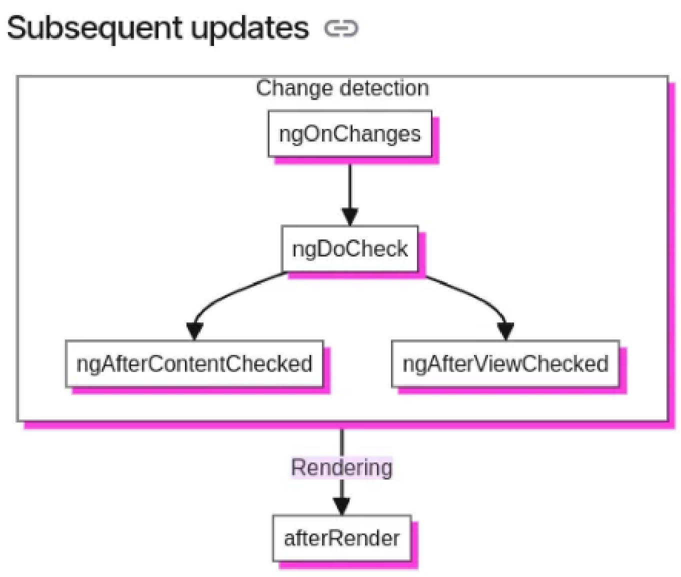
Fuente: https://zerotomastery.io/cheatsheets/angular-cheat-sheet/

Angular Lifecycle

- ngOnChanges: This is called whenever one of the input properties changes.
- ngOnInit: This is called immediately after ngOnChanges is completed and it is called once.
- ngOnDestroy: Called before angular destroys a directive or component
- ngDoCheck: Whenever a change detection is running, this is called.
- ngAfterContentInit: Invoked after Angular performs any content projection into the component’s view.
- ngAfterContentChecked: This is called each time the content of the given component has been checked by the change detection mechanism of Angular.
- ngAfterViewInit This is called when the component’s view has been fully initialized.
- ngAfterViewChecked: Invoked each time the view of the given component has been checked by the change detection mechanism of Angular.

Fuente: https://medium.com/@wrappixel/angular-cheat-sheet-2023-230a3b7972fb





Fuente: https://medium.com/@nandeepbarochiya/component-lifecycle-in-angular-v17-782f03cc9da3