Angular Cheat Sheet 2023

By Vishank Joshi On July 21, 2023 Angular, Cheat Sheet

Angular is a TypeScript-based open-source web application framework used to build web and mobile-based applications. This article will go through some of the angular features by explaining some of its core APIs. You can follow this angular cheat sheet to build your project.

Angular Cheat Sheet:

Angular is a TypeScript-based open-source web application framework used to build web and mobile-based applications. This article will go through some of the angular features by explaining some of its core APIs. You can follow this angular cheat sheet to build your project. We have tried to cover Angular CLI, Angular Lifecycle Hooks, Angular Routing, and much more here.

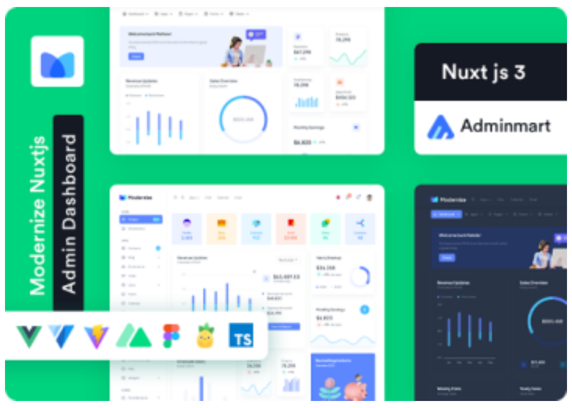
1. Angular CLI

Angular gives us the ability to do a whole lot using their CLI. You can config the entire application by just using the CLI. Here are some of the commands:

- `npm install -g @angular/cli` : This command will install the Angular CLI into our local machine using npm.

- `ng new <application name>` : This will set up a new Angular application using the `ng new` command.
- `ng new <application name> --prefix best` : This creates a new project and set the project prefix to new.
- `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 component <name>`: This will create a new component of our application. We can also use the `ng g c <name>` shorthand to do this.
- `ng g d <directive name>`: This command angular directive.
- `ng g s <service name>` : Creates a new Javascript class-based service.
- `ng g p <pipe name>`: Generates a new pipe
- `ng g cl <destination>` : 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 -ssl`: serves the application using SSL

2. Angular Lifecycle Hooks



Categories

- Admin Template
- AI (Artificial Intelligence)
- Angular
- Bootstrap
- Cheat Sheet
- Cloud Computing
- Coding
- CSS
- Data Security
- Learning
- Nextjs
- Nodejs
- Online Games
- React
- Resources
- Software Development
- Tutorial
- Typescript
- UI
- Vue

A component in Angular has a life cycle, several different phases it goes through from birth to death. We can hook into those different phases to get some pretty fine-grained control of our application. Here you can see some Angular Lifecycle Hooks.

- `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.

SHARES

3. How Angular Hooks are used

Always remember that hooks work in a component or directory, so use them in our component, we can do this:

```
`class ComponentName {
    @Input('data') data: Data;
    constructor() {
        console.log(`new - data is ${this.data}`);
    }
    ngOnChanges() {
        console.log(`ngOnChanges - data is ${this.dat
    }
    ngOnInit() {
        console.log(`ngOnInit  - data is ${this.data}
    }
    ngDoCheck() {
        console.log("ngDoCheck")
    }
    ngAfterContentInit() {
        console.log("ngAfterContentInit");
    }
    ngAfterContentChecked() {
        console.log("ngAfterContentChecked");
    }
    ngAfterViewInit() {
        console.log("ngAfterViewInit");
    }
    ngAfterViewChecked() {
        console.log("ngAfterViewChecked");
    }
    ngOnDestroy() {
        console.log("ngOnDestroy");
    }
}
```

SHARES

+

4. Component DOM

Angular comes with DOM features where you can do a whole lot from the binding of data and defining dynamic styles. Let's take a look at some features:

Before we dive into the features, a simple component.ts file is in this manner:

```
import { Component } from '@angular/core';
@Component({
    // component attributes
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.less']
})
export class AppComponent {
    name: 'Sunil';
}
```

5. Let's look at some template syntax:

- `Interpolation`: using `{{data to be displayed}}` will display dynamic content from the ts file.
- `<button (click)="callMethod()" ... />` : Adding Click events to buttons to call a method defined in the ts file
- `<button *ngIf="loading" ... />`: Adding Conditionals to elements. Conditionals have to listen to truthy or falsy values.
- `*ngFor="let item of items"`: iterate through a defined list of items. Picture this as a for a-loop.
- `<div [ngClass]="{green: isTrue(), bold: itTrue()}"/>`: Adding dynamic classes based on conditionals.
- `<div [ngStyle]="{'color': isTrue() ? '#bbb' : '#ccc'}"/>`: Adding dynamic styles to the template based on conditions

6. Component Communication

Passing data from one component to another can be a little bit tricky in Angular. You can pass data from child to parent, parent to parent, and between two unrelated components:

- `input()`: This method helps To pass a value into the child components.

```
export class SampleComponent {
@Input() value: 'Some Data should go in here';
}
```

Child components are registered in parents' components like this:

```
<child-component [value]="data"></child-component>
```

- `output()`: This method Emits an event to the parent component. A bunch of data can be passed into the emitted event which makes it a medium of passing data from child to parent:

To Emit the event from the child component:

```
@Output() myEvent: EventEmitter < MyModel > = new Eve
calledEvt(item: MyModel) {
    this.myEvent.emit(item);
}
```

And then the parent component listens to that event:

```
<parent-component
(myEvent)="callMethod()"></parent-component>
```

7. Angular Routing

Routing is another cool feature of Angular, with the Angular Routing system we can navigate through pages and even add route guards.

- Component Routing: We can define routes in our application by defining the path and the component to be rendered:

```
const routes: Routes = [
  { path: 'home', component:HomeComponent },
  { path: 'blog/:id', component: BlogPostCompoent },
  { path: '**', component: PageNotFoundComponent }
];
```

For routing to work, add this the your `app.module.ts` file:

```
RouterModule.forRoot(routes)
```

There are situations whereby you want to keep track of what is happening in your routes, you can add this to enable tracing in y angular project:

```
uterModule.forRoot(routes,{enableTracing:true})
```

To navigate through pages in Angular, we can use the `routerLink` the attribute which takes in the name of the component we are routing to:

```
<a routerLink="/home" routerLinkActive="active"> Cris
```

The `routerLinkActive="active"` will add an active class to the link when active.

8. Writing Route Guards

We can define a guard for route authentication. We can use the `CanActivate` class to do this:

```
class AlwaysAuthGuard implements CanActivate {
        canActivate() {
                return true;
        }
}
```

To use this rote guard in our routes we can define it here:

```
const routes: Routes = [
  { path: 'home', component:HomeComponent },
  { path: 'blog/:id', component: BlogPostCompoent,can
    { path: '**', component: PageNotFoundComponent }
];
```

9 Angular Services

A lar services come in handy when you can do things like h ling HTTP requests and seeding data on your application. T' focus on presenting data and delegating data access to a service.

```
@Injectable()
export class MyService {
    public users: Users[];
    constructor() { }
    getAllUsers() {
        // some implementation
    }
}
```

To use this service in your component, import it using the import statement and then register it in the constructor

```
import MyService from '<path>'
constructor(private UserService: MyService)
```

To make things easier, we can use this command to generate service in Angular

```
ng g s <service name>
```

10. HTTP Service

Angular comes with its own HTTP service for making HTTP requests. To use it, you have to first of all import it into your root module:

```
import { HttpClientModule} from "@angular/common/http
```

After importing it, we can now use it inside our service for making HTTP requests:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
@Injectable({
    providedIn: 'root'
})
export class UserService {
    constructor(private http: HttpClient) { }
    getAllUsers() {
        return this.http.get(`${baseURL}admin/list-us
    }
}
```

11. HTTP Interceptors

An **interceptor** is a piece of code that gets activated for every single **HTTP** request received by your application. Picture an interceptor as a middleware in nodejs where an HTTP request made is passed through this piece of code.

To define an interceptor create a `http-interceptor.ts` file inside your src directory and add this:

```
import { Injectable } from '@angular/core';
import {
    HttpEvent,
    HttpInterceptor,
    HttpHandler,
    HttpRequest,
    HttpErrorResponse,
    HttpResponse
} from '@angular/common/http';
  port { Observable } from 'rxjs';
  port { tap } from 'rxjs/operators';
  njectable({
    providedIn: 'root'
})
export class HttpConfigInterceptor implements HttpInt
    constructor() { }
    intercept(req: HttpRequest<any>, next: HttpHandle
        // Get the auth token from  localstorage.
        const authToken = localStorage.getItem('token
        // Clone the request and replace the original
        // cloned headers, updated with the authoriza
        const authReq = req.clone({
            headers: req.headers.set('Authorization',
        });
        // send cloned request with header to the nex
        return next.handle(authReq);
    }
}
```

This is a simple interceptor that checks if users have a token in their device's local storage. If the user does, it will pass the token in all the HTTP headers.

12. Pipes

Pipes in Angular give us the ability to transform data into any specific format. For example, you can write a simple pipe that will format an integer to a currency format or format dates to any form.
Angular comes with some built-in pipes like the date and currency pipe.

We can define our own custom pipes too by doing this:

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

@Pipe({ name: 'exponentialStrength' })
export class ExponentialStrengthPipe implements PipeT
    transform(value: number, exponent?: number): numb
        return Math.pow(value, isNaN(exponent) ? 1 :
    }
}
```

to use a pipe in our component we can do this:

```
{{power | exponentialStrength: factor}}
```

13. Bootstrapping:

- To import `platformBrowserDynamic` from `@angular/platform-browser-dynamic`.

```
import { platformBrowserDynamic } from '@angular/plat
```

- Bootstraps the application, using the root component from the specified NgModule.

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

14. NgModules:

- To import NgModule from @angular/core.

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

- Defines a module that contains components, directives, pipes, and providers.

```
@NgModule({
  declarations: …,
  imports: …,
  exports: …,
  providers: …,
  bootstrap: …
})
```

-
-
- List of components, directives, and pipes that belong to this module.

```
declarations: [
  MyRedComponent,
  MyBlueComponent,
  MyDatePipe
]
```

- List of modules to import into this module. Everything from the imported modules is available to declarations of this module.

```
imports: [
  BrowserModule,
  SomeOtherModule
]
```

- List of components, directives, and pipes visible to modules that import this module.

```
exports: [
  MyRedComponent,
  MyDatePipe
]
```

- List of dependency injection providers visible both to the contents of this module and to importers of this module.

```
providers: [
  MyService,
  { provide: … }
]
```

- List of components to bootstrap when this module is bootstrapped.

```
bootstrap: [MyAppComponent]
```

15. Template Syntax:

- Binds property value to the result of expression firstName.

```
<input [value]="firstName">
```

- Binds attribute role to the result of expression myAriaRole.

```
<div [attr.role]="myAriaRole">
```

- Binds the presence of the CSS class extra-sparkle on the element to the truthiness of the expression isDelightful.

```
<div [class.extra-sparkle]="isDelightful">
```

- Binds style property width to the result of the expression mySize in pixels. Units are optional.

```
iv [style.width.px]="mySize">
```

- Calls method `readRainbow` when a click event is triggered on this button element (or its children) and passes in the event object.

```
<button (click)="readRainbow($event)">
```

- Binds a property to an interpolated string, for example, “Hello Seabiscuit”.

```
<div title="Hello {{ponyName}}">
```

is equivalent to:

```
<div [title]="'Hello ' + ponyName">
```

- Binds text content to an interpolated string, for example, “Hello Seabiscuit”.

```
<p>
  Hello {{ponyName}}
</p>
```

- Sets up two-way data binding.

```
<my-cmp [(title)]="name">
```

is equivalent to:

```
<my-cmp [title]="name" (titleChange)="name=$event">
```

- Creates a local variable `movieplayer` that provides access to the `video` element instance in data-binding and event-binding expressions in the current template.

```
<video #movieplayer …></video>
<button (click)="movieplayer.play()">
  Play
</button>
```

- The asterisk (*) character turns the current element into an embedded template.

```
<p *myUnless="myExpression">
  …
</p>
```

is equivalent to:

```
<ng-template [myUnless]="myExpression">
  <p>
    …
  </p>
</ng-template>
```

- Transforms the current value of the expression cardNumber using the pipe called myCardNumberFormatter.

```
  >
  Card No.: {{cardNumber | myCardNumberFormatter}}
  p>
```

- The safe navigation operator (?) means that the employer field is optional and if undefined, the rest of the expression should be ignored.

```
<p>
  Employer: {{employer?.companyName}}
</p>
```

- An SVG snippet template needs an svg: prefix on its root element to disambiguate the SVG element from an HTML component.

```
<svg:rect x="0"
          y="0"
          width="100"
          height="100"/>
```

- A root element is detected as an SVG element automatically, without the prefix.

```
<svg>
  <rect x="0"
        y="0"
        width="100"
        height="100"/>
</svg>
```

15. Built-in Directives:

Import `CommonModule` from `@angular/common`.

```
import { CommonModule } from '@angular/common';
```

- Removes or recreates a portion of the DOM tree based on the showSection expression.

```
<section *ngIf="showSection">
```

- Turns the li element and its contents into a template, and uses that to instantiate a view for each item in list.

```
<li *ngFor="let item of list">
```

- Conditionally swaps the contents of the div by selecting one of the embedded templates based on the current value of conditionExpression.

```
<div [ngSwitch]="conditionExpression">
  <ng-template [ngSwitchCase]="case1Exp">
    …
  </ng-template>
  <ng-template ngSwitchCase="case2LiteralString">
    …
  </ng-template>
  <ng-template ngSwitchDefault>
    …
  </ng-template>
</div>
```

SHARES

- Binds the presence of CSS classes on the element to the ruthiness of the associated map values. The right-hand ɜxpression should return {class-name: true/false} map.

```
+ iv [ngClass]="{'active': isActive,
                 'disabled': isDisabled}">
```

- Allows you to assign styles to an HTML element using CSS. You can use CSS directly, as in the first example, or you can call a method from the component.

```
<div [ngStyle]="{'property': 'value'}">
<div [ngStyle]="dynamicStyles()">
```

17. Forms:

- Import FormsModule from @angular/forms.

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

- Provides two-way data-binding, parsing, and validation for form controls.

```
<input [(ngModel)]="userName">
```

18. Class Decorators:

- Import Directive, … from @angular/core';.

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

- Declares that a class is a component and provides metadata about the component.

```
omponent({…})
ass MyComponent() {}
```

- Declares that a class is a directive and provides metadata about the directive.

```
@Directive({…})
class MyDirective() {}
```

- Declares that a class is a pipe and provides metadata about the pipe.

```
@Pipe({…})
class MyPipe() {}
```

- Declares that a class can be provided and injected by other classes. Without this decorator, the compiler won't generate enough metadata to allow the class to be created properly when it's injected somewhere.

```
@Injectable()
class MyService() {}
```

19. Directive Configuration:

- Add property1 property with value1 value to Directive.

```
@Directive({
  property1: value1,
  …
})
```

- Specifies a CSS selector that identifies this directive within a template. Supported selectors include element, [attribute], .class, and :not().
- Does not support parent-child relationship selectors.

```
selector: '.cool-button:not(a)'
```

- List of dependency injection providers for this directive and its children.

```
providers: [
  MyService,
  { provide: … }
]
```

20. Component Configuration:

@component extends @directive, so the @directive configuration applies to components as well.

- If set, the templateUrl and styleUrl are resolved relative to the component.

```
moduleId: module.id
```

List of dependency injection providers scoped to this component's view.

```
viewProviders: [MyService, { provide: … }]
```

Inline template or external template URL of the component's view.

```
mplate: 'Hello {{name}}'
mplateUrl: 'my-component.html'
```

L f inline CSS styles or external stylesheet URLs for styling the component's view.

```
styles: ['.primary {color: red}']
styleUrls: ['my-component.css']
```

21. Class Field Decorators For Directives And Components:

- Import Input, … from @angular/core.

```
import { Input, … } from '@angular/core';
```

- Declares an input property that you can update using property binding (example:).

```
@Input() myProperty;
```

- Declares an output property that fires events that you can subscribe to with an event binding (example:).

```
@Output() myEvent = new EventEmitter();
```

- Binds a host element property (here, the CSS class valid) to a directive/component property (isValid).

```
@HostBinding('class.valid') isValid;
```

Subscribes to a host element event (click) with a directive/component method (onClick), optionally passing an argument ($event).

```
@HostListener('click', ['$event']) onClick(e) {…}
```

- Binds the first result of the component content query (myPredicate) to a property (myChildComponent) of the class.

```
@<a href="https://angular.io/api/core/ContentChild">C
```

- Binds the results of the component content query (myPredicate) to a property (myChildComponents) of the class.

```
@ContentChildren(myPredicate) myChildComponents;
```

- Binds the first result of the component view query (myPredicate) to a property (myChildComponent) of the class.

Not available for directives.

```
@ViewChild(myPredicate) myChildComponent;
```

- Binds the results of the component view query (myPredicate) to a property (myChildComponents) of the class. Not available for directives.

```
@ViewChildren(myPredicate) myChildComponents;
```

22. Directive And Component Change Detection And Lifecycle Hooks (Implemented As Class Methods):

Called before any other lifecycle hook. Use it to inject lependencies, but avoid any serious work here.

```
nstructor(myService: MyService, …) { … }
```

- Called after every change to input properties and before processing content or child views.

```
ngOnChanges(changeRecord) { … }
```

- Called after the constructor, initializing input properties, and the first call to ngOnChanges.

```
ngOnInit() { … }
```

- Called every time that the input properties of a component or a directive are checked. Use it to extend change detection by performing a custom check.

```
ngDoCheck() { … }
```

- Called after ngOnInit when the component's or directive's content has been initialized.

```
ngAfterContentInit() { … }
```

- Called after every check of the component's or directive's content.

```
ngAfterContentChecked() { … }
```

- Called after ngAfterContentInit when the component's views and child views / the view that a directive is in has been initialized.

```
ngAfterViewInit() { … }
```

- Called after every check of the component's views and child views / the view that a directive is in.

```
ngAfterViewChecked() { … }
```

- Called once, before the instance is destroyed.

```
ngOnDestroy() { … }
```

23. Dependency Injection Configuration:

- Sets or overrides the provider for MyService to the MyMockService class.

```
{ provide: MyService, useClass: MyMockService }
```

- Sets or overrides the provider for MyService to the myFactory factory function.

```
{ provide: MyService, useFactory: myFactory }
```

- Sets or overrides the provider for MyValue to the value 41.

```
{ provide: MyValue, useValue: 41 }
```

24. Routing And Navigation:

SHARES

Import Routes, RouterModule, … from @angular/router.

```
import { Routes, RouterModule, … } from '@angular/rou
```

+

Configures routes for the application. Supports static, parameterized, redirect, and wildcard routes. Also supports custom route data and resolve.

```
const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'path/:routeParam', component: MyComponent
  { path: 'staticPath', component: … },
  { path: '**', component: … },
  { path: 'oldPath', redirectTo: '/staticPath' },
  { path: …, component: …, data: { message: 'Custom'
]);

const routing = RouterModule.forRoot(routes);
```

Marks the location to load the component of the active route.

```
<router-outlet></router-outlet>
<router-outlet name="aux"></router-outlet>
```

Creates a link to a different view based on a route instruction consisting of a route path, required and optional parameters, query parameters, and a fragment. To navigate to a root route, use the / prefix; for a child route, use the ./prefix; for a sibling or parent, use the ../ prefix.

```
<a routerLink="/path">
<a [routerLink]="[ '/path', routeParam ]">
<a [routerLink]="[ '/path', { matrixParam: 'value' }
<a [routerLink]="[ '/path' ]" [queryParams]="{ page:
<a [routerLink]="[ '/path' ]" fragment="anchor">
```

The provided classes are added to the element when the routerLink becomes the current active route.

```
<a [routerLink]="[ '/path' ]" routerLinkActive="activ
```

The provided classes and aria-current attribute are added to the element when the routerLink becomes the current active route.

```
<a [routerLink]="[ '/path' ]" routerLinkActive="activ
```

An interface for defining a function that the router should call first to determine if it should activate this component. Should return a boolean|UrlTree or an Observable/Promise that resolves to a boolean|UrlTree.

```
function canActivateGuard: CanActivateFn =
  (
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ) => { … }

{ path: …, canActivate: [canActivateGuard] }
```

An interface for defining a function that the router should call first to determine if it should deactivate this component after a navigation. Should return a boolean|UrlTree or an Observable/Promise that resolves to a boolean|UrlTree.

```
function canDeactivateGuard: CanDeactivateFn<T> =
  (
    component: T,
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ) => { … }

{ path: …, canDeactivate: [canDeactivateGuard] }
```

An interface for defining a function that the router should call first to determine if it should activate the child route. Should return a boolean|UrlTree or an Observable/Promise that resolves to a boolean|UrlTree.

```
function canActivateChildGuard: CanActivateChildFn =
  (
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ) => { … }

{ path: …, canActivateChild: [canActivateGuard], chil
```

An interface for defining a function that the router should call first to resolve route data before rendering the route. Should return a value or an Observable/Promise that resolves to a value.

```
function resolveGuard implements ResolveFn<T> =
  (
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ) => { … }

{ path: …, resolve: [resolveGuard] }
```

An interface for defining a function that the router should call first to check if the lazy loaded module should be loaded. Should return a boolean|UrlTree or an Observable/Promise that resolves to a boolean|UrlTree.

```
function canLoadGuard: CanLoadFn =
  (
    route: Route
  ) => { … }

{ path: …, canLoad: [canLoadGuard], loadChildren: … }
```

Free Angular Templates by AdminMart

Well, I hope you have enjoyed our Angular Cheat Sheet and learned about Angular Lifecycle hooks, routing, and more. It will help while developing your project. Same way, Angular Templates also can help you with your project. As they come with stunningly designed interfaces and ready-to-use components, which can save you time and money. Also, you will be able to find the best Free Angular Templates by AdminMart. You can download that with zero investment and use it in your personal as well as commercial projects.

If you are looking for Angular Admin Dashboard Template then you can check out below useful Admin Template which can save you time, money, and energy:

Modernize Angular 16 Material Dashboard

Also Checkout Other Cheat Sheets:

Next.js Cheat Sheet

React Cheat Sheet

Nuxt Cheat Sheet

Vue Cheat Sheet

Related Posts