Essential Angular 9 Development Cheat Sheet

Read it in 11 Mins

Last updated on	Published	Views
07th Jun, 2022	03rd Jul, 2020	7186

Angular is a front-end web-development framework and platform that is used to build Single-page web Applications (or SPAs) using TypeScript as the primary programming language. Angular is itself written in TypeScript. Angular is an opinionated framework which means that it specifies a certain style and establishes certain rules that developers need to, learn, follow and adhere to while developing apps with Angular, therefore you need to learn Angular and the various modules, components, tools that are used and that make up an Angular app.

There are way too many versions of Angular that you hear about and as a beginner, it is likely that may get confused with so many different versions out there for the same framework. There are versions like AngularJS, Angular 2, Angular 4, Angular 5, Angular 6, Angular 7, Angular 8 and now Angular 9 that you may have heard of. Too many versions, right? Well, in reality, there are just two different frameworks - AngularJS and Angular.

AngularJS was the initial release and was called as AngularJS. It was the JavaScript-based web development framework which was first released in 2010 and is maintained by Google. Later, in September 2016, Angular 2 was announced which was a complete rewrite of the whole framework using TypeScript which is a super-set language of JavaScript.

Since modern browsers do not understand TypeScript, a TypeScript compiler or transpiler is required to convert the TypeScript code to regular JavaScript code.

In this article, we will talk about everything about Angular that you can use as a quick reference guide. We will go through the core concepts, commands, utilities, etc.

Get to know more about angular cli.

The Angular CLI

Angular CLI or the command-line interface is a very powerful and sophisticated tool that allows you to perform a lot of actions in an Angular project by using simple commands. From scaffolding a brand-new Angular project to building the project for production, the CLI takes care of everything.

CLI also allows you to debug the application during development locally using a development server that monitors the source files for changes and when you change any of the files, it automatically recompiles the source code files and refreshes the app in the browser. Pretty neat!

You can use the CLI tool directly in a command shell, or indirectly through an interactive UI such as Angular Console.

Let's have a look at some of the most common ways we use the Angular CLI.

ng help	Gives you options to get help about any command ot utility.
ng version	Prints the version info of the currently installed CLI.
ng new <name>	Generates a new Angular project with the provided name.
ng serve	Spins up the local development server and launches the web app locally in the browser. The command takes more parameters like port and open. The server automatically rebuilds the app and reloads the page when you change any of the source files.

ng build	Complies and builds the Angular app into a directory that can be deployed on the web server.
ng generate	Used to generate modules, services, components, classes, providers, pipes, etc.

Angular Component Lifecycle

An Angular app is composed of components. Each component can contain more components which essentially makes an Angular app a tree of components. A component is made up of an HTML template, a typescript class and a stylesheet file. These three pieces together form an Angular component. Each component in Angular has a lifecycle which is managed by Angular itself.

Angular creates it, renders it, creates and renders it's children, checks it when it's data-bound properties change, and destroys it before removing it from the DOM. Angular allows you to hook into significant lifecycle events using hooks. Here are these hooks and their brief significance

ngOnChanges	This hook is executed before the content is processed or child views are loaded. It is also executed when the input properties of the component change.
ngOnInit	This hook is used to initialize data in a component. It is called after input values are set and the component is initialized and it is executed only once.
ngDoCheck	This hook is called when any change detection runs are made.
ngAfterContentInit	This hook is called after ngOnInit when the component's or directive's content has been initialized.
ngAfterContentChecked	This hook is called every time the ngDoCheck is called.
ngAfterViewInit	This hook is called after Angular initializes component's and child components' content.
ngAfterViewChecked	This hook is after every check of the component's views and child views.
ngOnDestroy	This hook is called once right before the component is destroyed. This can be used to clean up memory and release allotted resources and subscriptions.

Interpolation

Interpolation is a way to use the properties in the component's TypeScript class within the HTML template. Interpolation has a very simple syntax. Here is an example.

Consider the following property of string type in the component's TS class.

```
company_name: string = "KnowledgeHut"
```

Copy Code

The same value can be used in the HTML template using the following interpolation syntax.

```
{{ company_name }}
```

Copy Code

The text between the braces is often the name of a component property. Angular replaces that name with the string value of the corresponding component property. Interpolation can also be used to concatenate strings, or perform simple mathematical operations right within the template. The evaluated result is rendered on the page.

An example -

```
name: string = "KnowledgeHut";
age: number = 25;
{{ name + " is " + age + " years old today!" }}
```

Copy Code

Another example -

```
number1: number = 22;
number2: number = 23;
{{ "Sum of "  + number1 + " and " + number2 + " is " + (number1 + number2) }}
```

Copy Code

The same technique can also be used while specifying the values of properties. Here is an example.

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

Copy Code

Clearly, using property binding is much readable and recommended syntax if you intend to use the values of component properties in the template.

Bindings in Angular

Binding in Angular is used to define values of HTML properties and events using class members.

Here are a few examples of property binding in Angular.

```
<div [title]="company_name">
```

Copy Code

In the above snippet, the title property is bound to a variable called company_name in the template. The same property needs to exist in the component's typescript class, otherwise you will get an error. Similarly, you can bind to any properties exposed by HTML.

Here is another example.

```
<input type="text" [placeholder]="placeholder_text">
```

Copy Code

The placeholder_text property needs to exist in the component's class.

You can also bind events to methods in Angular. Most HTML elements have events that they emit when certain conditions are met. Binding methods to these events allow you to perform an action in your code. For example, executing a method on the click of a button is the most common use-case of event binding. Here is a code snippet for the same.

```
<button type="submit" (click)="sendFormData()">
```

Copy Code

In the above code snippet, you can see that the click event is bound to a method called sendFOrmData(). Whenever the user clicks on the button, the function will be executed. Keep in mind that the sendFormData() method needs to be a member function inside the TypeScript class of the component otherwise you will run into errors.

The most popular and powerful feature of Angular is two-way binding. With the help of two-way binding, when data changes in the component, it's immediately reflected in the template. You need to use the ngModel directive with the special binding syntax to use two-way binding.

Here is an example.

```
<input type="text" [(ngModel)]="username">
The name you entered is {{ username }}!
```

Copy Code

Using the above code snippet, if you type in some text in the input field, the value of the username property will change as you type which in turn will update the value in the HTML template as well. All of this happens in real-time.

Structural Directives

There are two structural directives in Angular - NgFor and NgIf.

The ngIf directive is used to add or remove an HTML element from the DOM completely based on a certain condition. The value passed to ngIf should either be a boolean value or should be an expression that evaluates to a boolean value. Here is an example.

```
<button type="submit" (click)="sendFormData()" *ngIf="showButton">
```

Copy Code

In the above snippet, the showButton property is used to decide if the button will be displayed on the page or not. Do not confuse ngIf with the hidden HTML property. The hidden property only hides the HTML element, which ngIf completely removed it from the DOM.

Another structural directive is used to render HTML elements on the page based on an array or iterable. Here is an example.

```
users: any[] = [{
"name": "Bob"
}, {
"name": "Alice"
```

```
}, {
"name": "Tabor"
}, {
"name": "Mock"
}];
<li *ngFor="let user of users">
{{ user.name }}
</li>
```

Copy Code

If you execute the above code snippet, you will see a list rendered out on the page with the data based on the array. If you make changes to the array, the list will be updated automatically.

Pipes and Services

Pipes and services are also TypeScript classes just like the component classes. The only difference is the functionality that they offer. They are marked differently by using the decorators.

```
@Component({...})
class AppComponent() { }
```

Copy Code

This declares that the class is a component and also provides metadata about the component.

```
@Pipe({...})
class MyPipe() { }
```

Copy Code

This declares that the class is a pipe and provides metadata about the pipe. Angular comes with a stock of pipes such as DatePipe, UpperCasePipe, LowerCasePipe, CurrencyPipe, and PercentPipe. They are all available for use in any template. Here is how you would use a pipe in Angular

```
number1: number = 12.927;
{{ number1 | number:"1.2-2"}} // prints 12.93
@Injectable()
class MyService() { }
```

Copy Code

This declares that the class contains dependency injectors, this means the data provided by the injector should be injected into the constructor when creating an instance of this class. Basically, this marks the class as a service to be used by other classes.

To create a pipe, you need to use the following CLI command.

```
ng generate pipe <pipe name>
```

Copy Code

Similarly, to create a service, you need to execute the following CLI command.

```
ng generate service <service name>
```

Copy Code

Component Inputs and Output

Angular is all about components that have to interact with one another. Inputs and Outputs are used to allow interaction between two components. Inputs allow a component to receive some data from the parent component and outputs allows the component to send some data outside the component back to the parent component. So inputs and outputs are used to implement parent-child interaction within components.

Let's have a look at how we can allow a component to receive an input. The key here is using the @Input decorator.

```
@Input() username: string;
```

Copy Code

The above code snippet is added to the component's TypeScript class. Now, the component can accept an input called "username" as shown below.

```
<my-component username="KnowledgeHut"></ my-component>
```

Copy Code

The value “KnowledgeHut” can now be accessed within the component using the **username** property.

The other direction of communication is from child to parent. In this case, the child emits some data using an event. The parent subscribes to this event using event binding and receives the data. Here is an example.

Inside the child component -

```
@Output() onUsernameChange = new EventEmitter<string>();
```

Copy Code

To emit the event,

```
this.onUsernameChange.emit("KnowledgeHut");
```

Copy Code

That’s all in the child component.

Here is how the parent used the child component with outputs.

```
<app-child (onUsernameChange)="getChangedUsername($event)"></app-child>
```

Copy Code

Reactive Forms in Angular

The Reactive approach of form building uses the classes like FormBuilder, FormGroup, FormControl, Validators, etc. We define the structure of the form using a form builder and then bind each form element to a form control. This approach is different from template-driven forms where we use ngModel directive to bind input fields to corresponding model properties.

Here is how you would define a simple form with two input fields, one for the email and another one for the password.

```
this.myGroup = new FormGroup({
email: new FormControl('john@doe.com'),
password: new FormControl('')
});
```

Copy Code

Next, we need to bind the form and the input fields as shown below.

```
<form [formGroup]="myGroup">
  Email: <input type="email" formControlName="email">
  Password: <input type="password" formControlName="password">
</form>
```

Copy Code

This approach allows you to define Validators on individual form controls and on the form as well.

The Angular Router

Angular router is one of the most powerful modules in Angular which allows navigation within an Angular application. It takes care of lazy-loading components as and when required.

Here is an example of how you would define routes for an Angular app using the **RouterModule**.

```
const routes: Routes = [{
path: '', redirectTo: 'home', pathMatch: 'full'
}, {
path: 'home',  component: HomeComponent
}, {
path: 'login', component: LoginComponent
}, {
path: 'myblogs', component: MyblogsComponent
}, {
path: 'profile/:id', component: ProfileComponent
}, {
```

```
path: '**', redirectTo: 'home'
}];
const routing = RouterModule.forRoot(routes);
```

Copy Code

In the template, you need to use the **RouterOutlet** directive to load the component in a placeholder based on the URL. The Angular Router takes care of loading and unloading components in the placeholder **RouterOutlet**.

```
<router-outlet></ router-outlet>
```

Copy Code

The **routerLink** directive can be used to navigate to a desired router. It 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.

```
<arouterLink='/login'>
<a [routerLink]="['/profile', { id: '23u4h2834y' } ]">
```

Copy Code

To navigate to a root route, use the / prefix; for a child route, use the ./prefix; for a sibling or parent, use the ../ prefix.

Auth Guards

Auth Guards in Angular are the way to prevent unauthorized access to your routes. They work very well in combination with the Angular Router. It secures your routes. Let's have a look at an example. Consider the following route definition.

```
const routes: Routes = [{
path: '', redirectTo: 'home', pathMatch: 'full'
}, {
path: 'home',  component: HomeComponent
}, {
path: 'login', component: LoginComponent
}, {
path: 'myblogs', component: MyblogsComponent, canActivate: [AuthGuard]
```

```
}, {
path: 'profile/:id', component: ProfileComponent
}, {
path: '**', redirectTo: 'home'
}];
```

Copy Code

As you can see, we have defined a canActivate property on a route and set its values as an array with the AuthGuard as the only element. Basically, you can have multiple auth guards for your routes but for now, we only have one. Here is how you can create the guard.

```
ng generate guard AuthGurard
```

Copy Code

This generates a new guard typescript file and it has a function defined inside of it. All you need to do is implement the function and return true or false based on the user's state. If the guard returns true for the user, the route access if grant, else not.

Here is an example implementation of the function, **canActivate**.

```
canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<boolean> | Promise<boolean> | boolean {
      return new Promise((resolve, reject) => {
// return or resolve with true or false
      }
}
```

Copy Code

Build the App

The Angular CLI allows you to build the app.

```
ng build
```

Copy Code

The above command compiles the project and generates the final build inside the dist folder. You can however customize the build environment using the --prod flag. You can also change the build destination folder using the --outputPath flag. The following command is another simple example.

```
ng build --prod --outputPath=myBuildFolder
```

Copy Code

There are a few more flags available.

- prod
- outputPath
- localize
- aot
- baseHref
- buildOptimizer
- index
- verbose
- optimization

Conclusion

Angular may seem daunting at first but with steady learning and practicing, it becomes easy to understand and implementation of complex concepts seems simple. It offers a lot of tools, utilities and external packages to build amazing applications.

Some of the most common external packages for Angular are -

- NgxBootstrap
- Angular Material
- Ng2-Charts
- Ngx-Text-Editor
- NgxDataTable

Use this cheat sheet to quickly look for help or use it as a reference. Check Angular docs for more info.

Tags

Front End Web Development Training

KnowledgeHut
Author

KnowledgeHut is an outcome-focused global ed-tech company. We help organizations and professionals unlock excellence through skills development. We offer training solutions under the people and process, data science, full-stack development, cybersecurity, future technologies and digital transformation verticals.
Website: https://www.knowledgehut.com

Join the discussion

Name*

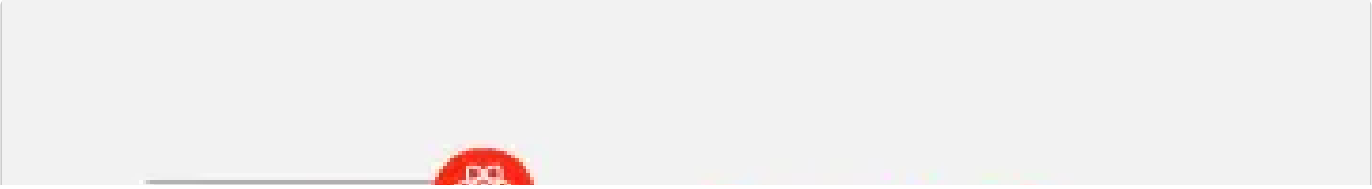
Email*

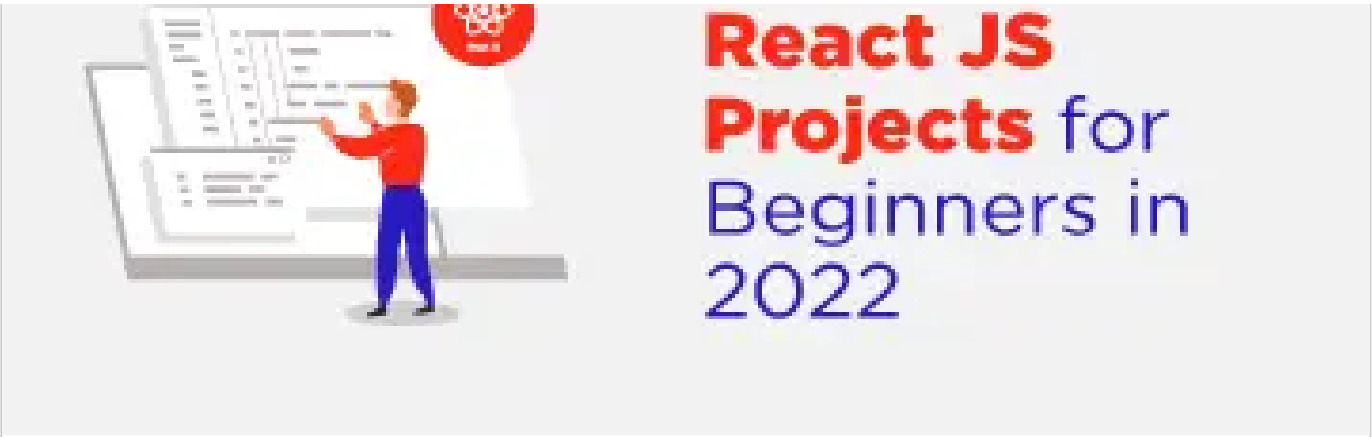
Comment*

Submit

Your email address will not be published. Required fields are marked *

Suggested Blogs





Blogs

571

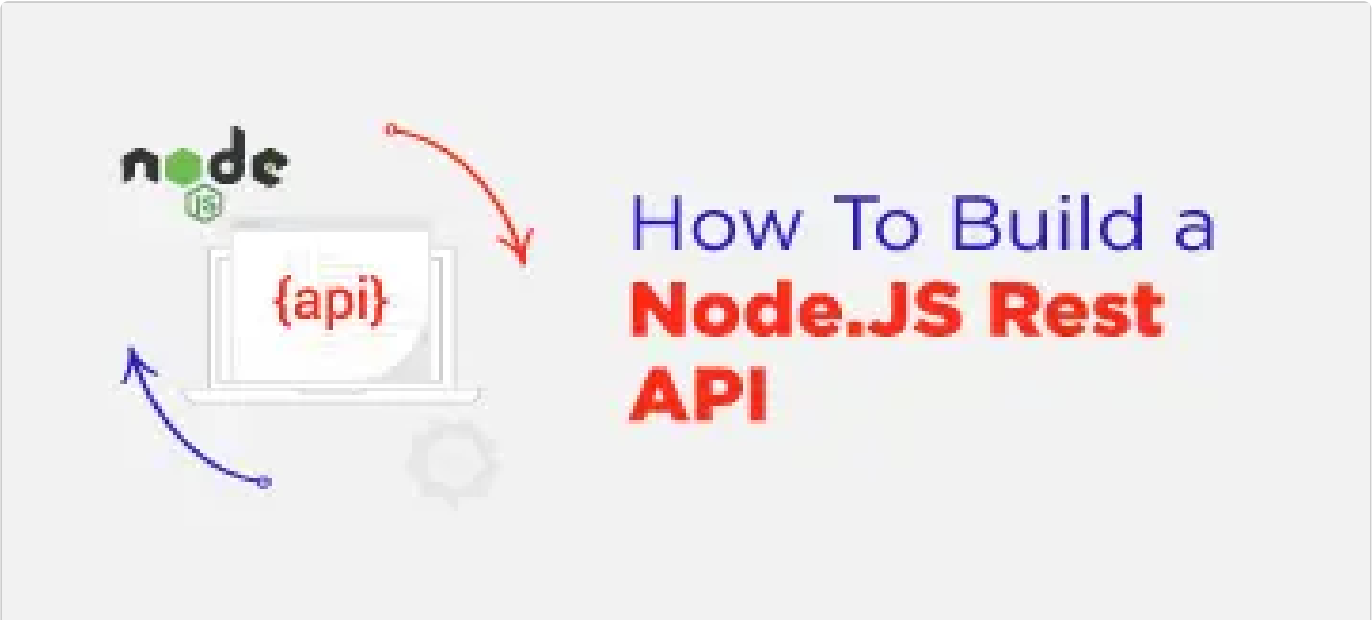
7 Stunning React JS Projects For Beginners - 2022

Web Development is the category of software development process which involves the design, development and maintenance of a full website or mobile web

Read More

by **KnowledgeHut**

02 Aug 2022



Blogs

2751

How to Build a Node.js REST API in Simple Steps?

Application Programming Interface (API) mania has spread throughout the globe. For scalability and reusability, it provides an interface that allows t

Read More

by **Binod Anand**

01 Aug 2022

Blogs

4574

How to Create a React App with Typescript

In this article, we will be talking about creating a react project with TypeScript. Also, we'll be talking about some important things that you

Read More

by **Gamage Ayesh Nipun...**

01 Aug 2022

Load More

Connect with us

Get Our Weekly Newsletter

We Accept

USA: +1-469-442-0620, +1-832-684-0080

India: +91-84484-45027
Toll Free: 1800-121-9232

UK: +44-2036085923

Singapore: +65-315-83941

Malaysia: +601548770914

Canada: +1-613-707-0763

New Zealand: +64-36694791

Ireland: +353-14402544

Australia: +61-290995641

UAE: Toll Free 8000180860

Company

Offerings

Resources

Partner with us

Support

Top Categories

Top Courses

Disclaimer: KnowledgeHut reserves the right to cancel or reschedule events in case of insufficient registrations, or if presenters cannot attend due to unforeseen circumstances. You are therefore advised to consult a KnowledgeHut agent prior to making any travel arrangements for a workshop. For more details, please refer to the **Cancellation & Refund Policy**.
CSM®, CSPO®, CSD®, CSP®, A-CSPO®, A-CSM® are registered trademarks of Scrum Alliance®. KnowledgeHut Solutions Pvt. Ltd. is a Registered Education Ally (REA) of Scrum Alliance®. PMP is a registered mark of the Project Management Institute, Inc. CAPM is a registered mark of the Project Management Institute, In**READ MORE**

Privacy policy

Terms of service