**First Module**

[Creating new angular project](#creating_new_angular_project)

[NG Modules](#ng_modules)

[Angular.json](#angular_json)

[Component](#component)

[Interpolation and Property Binding](#Interpolation_and_roperty_inding)

[Expression Restrictions and Recommendations](#Expression_Restrictions_and_Recommendati)

[Event Bindings and Statements](#Event_Bindings_and_Statements)

[\*ngFor](#ngFor)

[\*ngIf and Hidden](#ngIf_and_Hidden)

[\*ngSwitch](#ngSwitch)

Communication with Child component using @input

Communication with Parent component using @output

Using template varible to interact with child component

[Services and DI](#Services_and_DI)

[Difference between Constructor and ngOnInit](#Difference_between_Constructor_and_ngOnI)

[Route, RouterOutlet, RouterLink](#Route_ActivatedRoute_RouterLink)

[Accessing Route Parameters](#Accessing_Route_Parameters)

[Navigate through the code depend on the route](#Navigate_through_the_code_depend_route)

[Guarding Against Route Activation - CanActivate](#Guarding_Against_Route_Activation)

[Guarding Againg Route De-activation - CanDeactivate](#Guarding_Againg_Route_De_activation)

[Pre-loading data for components (Resolve)](#Pre_loading_data_for_components_Resolve)

[Lazy-loading feature modules](#Lazy_loading_feature_modules)

[Export with Barrels](#Export_with_Barrels)

**Second Module**

Form (template driven)

Two-way binding

Form validation

Reactive Form

Reactive Form Validation

**Creating new angular project**



**NG Modules**

* The Declarations array
* The Imports Array
* The Providers Array
* The Bootstrap Array

Graphical user interface, text, application, email

Description automatically generated

1. The Declarations Array

Your own components and pipes go here.

Declarations are views, or classes that display data — including components, pipes, and directives.

2. The Imports Array

Modules, both external and internally created, get imported. Some examples of external modules include CommonModule (for \*ngFor and \*ngIf directives) and NgBootstrap modules like AccordionModule. An internal module might be an AppRoutingModule where you’ve defined routing paths.

3. The Providers Array

Put all your internal services as well as external services here. Providers are services, or classes that get/handle data.

Providers are globally scoped so they don’t need to be exported. Modules that provide services only needed to imported once across the application — in the root module. Examples of modules that provide just services include:

HttpClientModule: “a simplified client HTTP API that rests on XMLHTTPRequest interface exposed by browsers”

BrowserAnimationsModule or NoopAnimationsModule (if you want to mock animations, since importing one of the two is required for Angular-Material)

4. The Bootstrap Array

The component declared in the bootstrap array is the entry point of the application. According to the Angular documentation, “the bootstrapping process creates the component(s) listed in the bootstrap array and inserts each one into the browser DOM.”

**Multiple Modules**

As stated above, the purpose of NgModules is to organize the your application into cohesive groups based on functionality. Larger applications can benefit from having feature modules in addition to the root module.

Diagram

Description automatically generated

**Angular.json**

The angular.json file at the root level of an Angular workspace provides workspace-wide and project-specific configuration defaults.

**Component**

**Interpolation and Property Binding**

Text

Description automatically generated

Property Binding -> is used when you want to bind the data to the property of a DOM element

Interpolation -> is used when you want just to display that data

**Expression Restrictions and Recommendations**

Interpolated expressions have a context—a particular part of the application to which the expression belongs. Typically, this context is the component instance.

In the following snippet, the expression recommended and the expression itemImageUrl2 refer to properties of the AppComponent.

Graphical user interface, text, application

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

**Expression best practices**

Graphical user interface, application

Description automatically generated

When using template expressions, follow these best practices:

* Use short expressions

Use property names or method calls whenever possible. Keep application and business logic in the component, where it is accessible to develop and test.

* Quick execution

Angular executes template expressions after every change detection cycle. Many asynchronous activities trigger change detection cycles, such as promise resolutions, HTTP results, timer events, key presses, and mouse moves.

Expressions should finish quickly to keep the user experience as efficient as possible, especially on slower devices. Consider caching values when their computation requires greater resources.

* No visible side effects

According to Angular's unidirectional data flow model, a template expression should not change any application state other than the value of the target property. Reading a component value should not change some other displayed value. The view should be stable throughout a single rendering pass.

Graphical user interface, application

Description automatically generated

**Event Bindings and Statements**

**Event Bindings:**

To bind to an event you use the Angular event binding syntax. This syntax consists of a target event name within parentheses to the left of an equal sign, and a quoted template statement to the right.

Create the following example; the target event name is click and the template statement is onSave().

Graphical user interface, text, application

Description automatically generated

The event binding listens for the button's click events and calls the component's onSave() method whenever a click occurs.

Graphical user interface

Description automatically generated with medium confidence

**Statements**:

Template statements are methods or properties that you can use in your HTML to respond to user events. With template statements, your application can engage users through actions such as displaying dynamic content or submitting forms.

**\*ngFor**

A structural directive that renders a template for each item in a collection. The directive is placed on an element, which becomes the parent of the cloned templates.

A picture containing text

Description automatically generated

**\*ngIf and Hidden**

\*ngIf

A structural directive that conditionally includes a template based on the value of an expression coerced to Boolean. When the expression evaluates to true, Angular renders the template provided in a then clause, and when false or null, Angular renders the template provided in an optional else clause. The default template for the else clause is blank.

**There is actually a performance difference between them:**

* ngIf will comment out the data if the expression is false. This way the data are not even loaded, causing HTML to load faster.
* [hidden] will load the data and mark them with the hidden HTML attribute. This way data are loaded even if they are not visible.

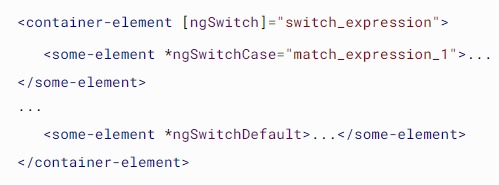
So [hidden] is better used when we want the show/hide status to change frequently, for example on a button click event, so we do not have to load the data every time the button is clicked, just changing its hidden attribute would be enough.

Note that the performance difference may not be visible with small data, only with larger objects.

**\*ngSwitch**

The [ngSwitch] directive on a container specifies an expression to match against. The expressions to match are provided by ngSwitchCase directives on views within the container.

* Every view that matches is rendered.
* If there are no matches, a view with the ngSwitchDefault directive is rendered.
* Elements within the [NgSwitch] statement but outside of any NgSwitchCase or ngSwitchDefault directive are preserved at the location.



**Services and DI**

Example of using Sercice

Text

Description automatically generated

Text

Description automatically generated

A picture containing graphical user interface

Description automatically generated

When you use the **@Injectable()** decorator without specifying the **providedIn** property, it means that you are deferring the decision of where to provide the service to a later time. In this case, you need to manually register the service with a specific injector using the providers array of the module, component or any other provider that is being used.

On the other hand, when you specify **providedIn: 'root',** you are telling Angular to automatically register the service with the root injector. This is useful for services that are intended to be used globally throughout the application and do not require any configuration or customization per component or module.

Text

Description automatically generated

Using **providedIn: 'root'** is the recommended way of providing a service in Angular, especially for services that are used widely across the application. This approach allows for better tree shaking and can reduce the size of the generated bundle, making the application load faster.

In summary, the **providedIn: 'root'** syntax is used to register a service with the root injector of the Angular application, making it available for injection throughout the entire application.

**Difference between Constructor and ngOnInit**

In Angular, the constructor method and ngOnInit lifecycle hook serve different purposes.

The constructor method is a standard TypeScript class constructor, which is executed when an instance of the component is created. It is mainly used for dependency injection, where you can declare the dependencies that your component requires as parameters. For example, you can inject services, providers, and other components into your component using the constructor.

The ngOnInit lifecycle hook is a method that is called by Angular after the component has been constructed and all its dependencies have been injected. It is used to initialize the component's properties and perform any other setup that needs to be done before the component is rendered. For example, you can use ngOnInit to fetch data from an API or set default values for properties.

In summary, the constructor method is used for dependency injection, while ngOnInit is used for component initialization and setup. You should avoid performing heavy initialization or time-consuming operations in the constructor, as this can slow down the creation of the component. Instead, you should use ngOnInit for any heavy initialization or time-consuming operations that need to be done.

Text

Description automatically generated

**Route, RouterOutlet, RouterLink**

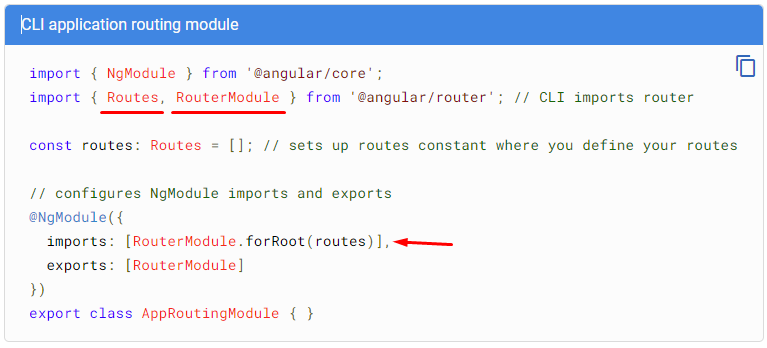
Import RouterModule and Routes into your routing module.

RouterModule.forRoot(appRoutes) is a method call from the RouterModule module of the Angular framework. It is used to set up routing in an Angular application.

The forRoot method takes a parameter app, which is an instance of the Routes array that defines the application's routes. The Routes array specifies the routes for the application, each of which maps a URL path to a component.

By calling forRoot with the Routes array, we're telling Angular to create a new router service and configure it with our specified routes. This service will be available for the entire application.

Here's an example of how to use RouterModule.forRoot to set up routing in an Angular app: Example from angular documentation ->



Real case example:

Text

Description automatically generated

Define your routes in your Routes array.

Each route in this array is a JavaScript object that contains two properties. The first property, path, defines the URL path for the route. The second property, component, defines the component Angular should use for the corresponding path.

Graphical user interface, text, application

Description automatically generated

**RouterOutlet**

Acts as a placeholder that Angular dynamically fills based on the current router state.

Graphical user interface, application

Description automatically generated

**RouterLink**

Now that you have defined your routes, add them to your application. First, add links to the two components. Assign the anchor tag that you want to add the route to the routerLink attribute. Set the value of the attribute to the component to show when a user clicks on each link. Next, update your component template to include <router-outlet>. This element informs Angular to update the application view with the component for the selected route.

Text

Description automatically generated

**Accessing Route Parameters**

Text

Description automatically generated

**Navigate through the code depend on the route**

The order of the routes in the Angular Routes array matters, because the router will try to match the URL path to the first route that matches the path. Therefore, you should take care to order your routes correctly to avoid any unintended consequences or issues.

Here are some best practices for ordering your Angular routes:

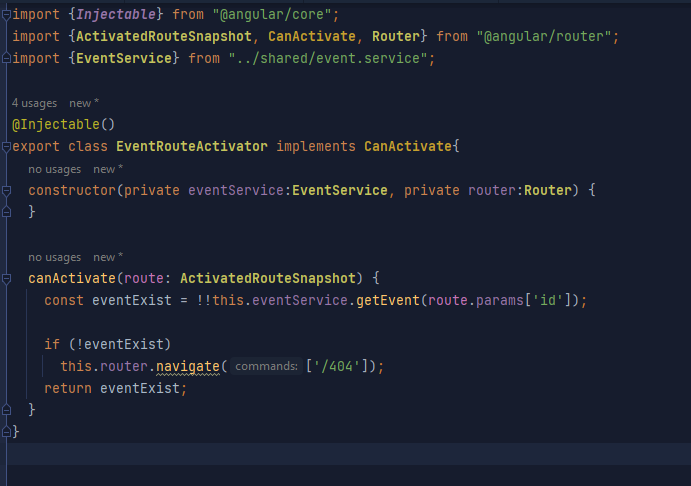
* Place your most specific routes at the top of the array, and your least specific routes at the bottom. This will help ensure that more specific routes are matched first.
* Put any redirect routes at the end of the array. This is because redirect routes are less specific than normal routes, and could potentially match other routes unintentionally.
* Group related routes together. If you have a group of related routes that share a common prefix, you can group them together to make your Routes array easier to read and maintain.
* Avoid overlapping routes. Make sure that your routes do not overlap with each other, as this can cause unexpected behavior. For example, if you have two routes with paths /events/new and /events/:id, and the user navigates to /events/new, the router may try to match this path to the second route, which is not intended.

**Guarding Against Route Activation - CanActivate**

CanActivate - Interface that a class can implement to be a guard deciding if a route can be activated. If all guards return true, navigation continues. If any guard returns false, navigation is cancelled.

First we need

1. Injectable()
2. The class should implements CanActivate
3. And we should implement the method canActivate
4. After the implementation we should add our Activator class inside @NgModule in providers
5. At the end we should add the Activator class to the specific route that we want to check (Example at the 3-rd image)



At this example we are checking for existing event. If the event doesn’t exist redirect to 404 page.

Graphical user interface, text

Description automatically generated

A picture containing text

Description automatically generated

**Guarding Againg Route De-activation - CanDeactivate**

We can use this for example in case where the user want to leave the page. And we want to show a confirmation pop-up. Just to be sure if he want to leave the page.

Adding the name of the event inside the route as canDeactivate

A picture containing text

Description automatically generated

Now this is other way to provide deactivator

We adding the name that we want and the useValue is basically the method that will be called

Text

Description automatically generated

Text

Description automatically generated

On this example we need to have in CreateEventComponent a isDirty property that we are setting to true or false based on some condition on the form

Text

Description automatically generated

**Pre-loading data for components (Resolve)**

Interface that classes can implement to be a data provider. A data provider class can be used with the router to resolve data during navigation. The interface defines a resolve() method that is invoked right after the ResolveStart router event. The router waits for the data to be resolved before the route is finally activated.

**Lazy-loading feature modules**

By default, NgModules are eagerly loaded. This means that as soon as the application loads, so do all the NgModules, whether they are immediately necessary or not. For large applications with lots of routes, consider lazy loading —a design pattern that loads NgModules as needed. Lazy loading helps keep initial bundle sizes smaller, which in turn helps decrease load times.

To lazy load Angular modules, use loadChildren (instead of component) in your AppRoutingModule routes configuration as follows.

Graphical user interface, text, application, email

Description automatically generated

In the lazy-loaded module's routing module, add a route for the component.

Graphical user interface, text, application, email

Description automatically generated

**Export with Barrels**

Example:   
The code for importing these component files in our app.module.ts file looks like this Text

Description automatically generated

If you pay careful attention to the imports in our app-module, you will notice a pattern: all the components are inside the same root folder named app folder.

These components are ideal candidates to be exported in the same file, that is, in a barrel. Thereafter, you can import all the components from this single file into their respective module.

**Create barrel**

Create a file named index.ts at the root of the app folder, that is in the same folder as the app.module.ts file. This will be our barrel.

Once the barrel file has been created you can then export our components within the barrel file, i.e index.ts , as follows:

Text

Description automatically generated

**Import from barrel**

Now, in your module file i.e app.module.ts in this case. We can import all the code from the barrel, instead of getting it from specific component folders.

app.module.ts

Text

Description automatically generated

**Conclusion**

Angular barrels are a great choice for centralized import and export of class files. Using barrels will help make your code readable, maintainable and more organized.

**Second Module**

**Form (template driven)**

Angular Template-driven forms are a way to handle form inputs and validation in Angular applications using directives and binding syntax in the template. It provides a simpler approach compared to Reactive Forms, which involve building form models programmatically.

In Template-driven forms, you define the form structure directly in the HTML template, and Angular automatically generates the corresponding form model and manages its state and validation. Here's how it works:

1. Creating the Form:

To start, you define an HTML form element and add the ngForm directive to it. This directive registers the form and enables Angular's form handling capabilities. You can also specify a name for the form using the name attribute.

A picture containing text, font, screenshot, design

Description automatically generatedA picture containing text, screenshot, font

Description automatically generated

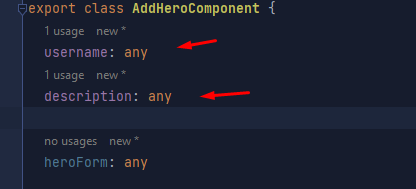
1. Form Controls:

Inside the form element, you add form controls using various input elements like <input>, <select>, or <textarea>. Each form control is associated with a specific data property in the component.

To establish a link between the form control and the component property, you use the ngModel directive. The ngModel directive binds the value of the input element to the associated component property and enables two-way data binding.

A screenshot of a computer program

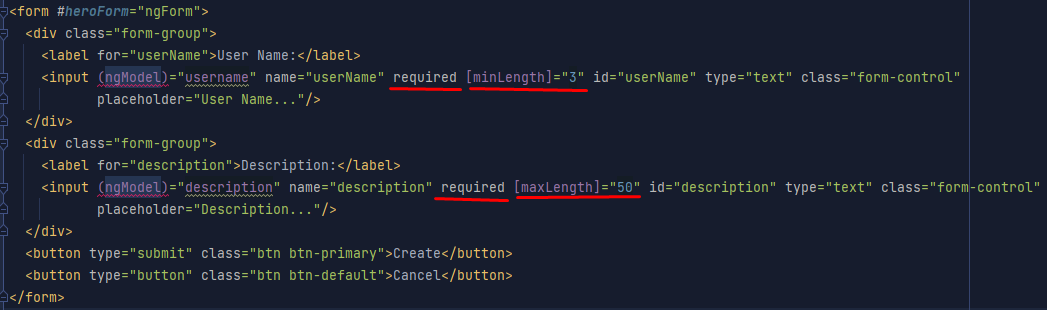
Description automatically generated with low confidence



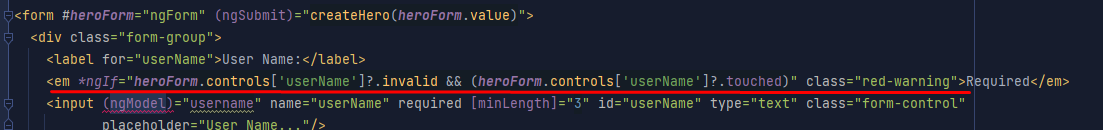
1. Validation:

Angular provides built-in validation directives like required, **minLength**, **maxLength**, etc., which you can apply to the form controls. These directives help enforce validation rules on the input values. For example, you can mark an input as **required** by adding the required attribute to the corresponding HTML element.

Angular automatically tracks the validity and state of the form controls based on the applied validation rules and user interactions. You can access the validity and state of the controls using properties such as valid, invalid, touched, dirty, etc.



Way to use the validation and the controls property and more->



A screenshot of a phone

Description automatically generated with low confidence

1. Form Submission:

To handle form submission, you can bind a method to the **(ngSubmit)** event of the form element. This method will be called when the form is submitted, allowing you to perform any necessary actions such as data processing, HTTP requests, or form resets.

In the component, you can access the form values using the associated component properties that are bound to the form controls. You can then use these values as needed.

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with medium confidence