

[Skip to main content](#)

What is Angular?

This topic can help you understand Angular: what Angular is, what advantages it provides, and what you might expect as you start to build your applications.

Angular is a development platform, built on [TypeScript](#). As a platform, Angular includes:

- A component-based framework for building scalable web applications
- A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more
- A suite of developer tools to help you develop, build, test, and update your code

With Angular, you're taking advantage of a platform that can scale from single-developer projects to enterprise-level applications. Angular is designed to make updating as straightforward as possible, so take advantage of the latest developments with minimal effort. Best of all, the Angular ecosystem consists of a diverse group of over 1.7 million developers, library authors, and content creators.

See the [live example](#) / [download example](#) for a working example containing the code snippets in this guide.

[Skip to main content](#)

Angular applications: The essentials

This section explains the core ideas behind Angular. Understanding these ideas can help you design and build your applications more effectively.

Components

Components are the building blocks that compose an application. A component includes a TypeScript class with a `@Component()` decorator, an HTML template, and styles. The `@Component()` decorator specifies the following Angular-specific information:

- A CSS selector that defines how the component is used in a template. HTML elements in your template that match this selector become instances of the component.
- An HTML template that instructs Angular how to render the component
- An optional set of CSS styles that define the appearance of the template's HTML elements

The following is a minimal Angular component.

[Skip to main content](#)

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

@Component({
  selector: 'hello-world',
  template: `
    <h2>Hello World</h2>
    <p>This is my first component!</p>
  `,
})
export class HelloWorldComponent {
  // The code in this class drives the component's
  // behavior.
}
```

To use this component, you write the following in a template:

```
<hello-world></hello-world>
```

When Angular renders this component, the resulting DOM looks like this:

```
<hello-world>
  <h2>Hello World</h2>
  <p>This is my first component!</p>
</hello-world>
```

Angular's component model offers strong encapsulation and an intuitive application structure. Components also make your application painless to unit test and can improve the general readability of your code.

For more information on what to do with components, see the [Components](#) section.

Templates

Every component has an HTML template that declares how that component renders. You define this template either inline or by file path.

[Skip to main content](#)

Elements that extend HTML so you can insert your component. Angular automatically updates the rendered DOM when your component's state changes. One application of this feature is inserting dynamic text, as shown in the following example.

```
<p>{{ message }}</p>
```

The value for message comes from the component class:

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

@Component ({
  selector: 'hello-world-interpolation',
  templateUrl: './hello-world-
interpolation.component.html'
})
export class HelloWorldInterpolationComponent {
  message = 'Hello, World!';
}
```

When the application loads the component and its template, the user sees the following:

```
<p>Hello, World!</p>
```

Notice the use of double curly braces—they instruct Angular to interpolate the contents within them.

Angular also supports property bindings, to help you set values for properties and attributes of HTML elements and pass values to your application's presentation logic.

[Skip to main content](#)

```
oId"
[style.color]="fontColor">
  You can set my color in the component!
</p>
```

Notice the use of the square brackets—that syntax indicates that you're binding the property or attribute to a value in the component class.

Declare event listeners to listen for and respond to user actions such as keystrokes, mouse movements, clicks, and touches. You declare an event listener by specifying the event name in parentheses:

```
<button
  type="button"
  [disabled]="canClick"
  (click)="sayMessage()">
  Trigger alert message
</button>
```

The preceding example calls a method, which is defined in the component class:

```
sayMessage() {
  alert(this.message);
}
```

The following is a combined example of Interpolation, Property Binding, and Event Binding within an Angular template:

`hello-world-bindings.component.ts`

`hello-world-bindings.component.html`

[Skip to main content](#)

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

2.

3. @Component ({
4.   selector: 'hello-world-bindings',
5.   templateUrl: './hello-world-
   bindings.component.html'
6. })
7. export class HelloWorldBindingsComponent {
8.   fontColor = 'blue';
9.   sayHelloId = 1;
10.  canClick = false;
11.  message = 'Hello, World';
12.
13.  sayMessage() {
14.    alert(this.message);
15.  }
16. }
```

Add features to your templates by using [directives](#). The most popular directives in Angular are [*ngIf](#) and [*ngFor](#). Use directives to perform a variety of tasks, such as dynamically modifying the DOM structure. And create your own custom directives to create great user experiences.

The following code is an example of the [*ngIf](#) directive.

`hello-world-ngif.component.ts`

`hello-world-ngif.component.html`

[Skip to main content](#) `Component } from '@angular/core';`

```
2.  
3. @Component({  
4.   selector: 'hello-world-ngif',  
5.   templateUrl: './hello-world-ngif.component.html'  
6. })  
7. export class HelloWorldNgIfComponent {  
8.   message = "I'm read only!";  
9.   canEdit = false;  
10.  
11.   onEditClick() {  
12.     this.canEdit = !this.canEdit;  
13.     if (this.canEdit) {  
14.       this.message = 'You can edit me!';  
15.     } else {  
16.       this.message = "I'm read only!";  
17.     }  
18.   }  
19. }
```

Angular's declarative templates let you cleanly separate your application's logic from its presentation. Templates are based on standard HTML, for ease in building, maintaining, and updating.

For more information on templates, see the [Templates](#) section.

Dependency injection

Dependency injection lets you declare the dependencies of your TypeScript classes without taking care of their instantiation. Instead, Angular handles

[Skip to main content](#)

you. This design pattern lets you write more testable code. Understanding dependency injection is not critical to start using Angular, but it is strongly recommended as a best practice. Many aspects of Angular take advantage of it to some degree.

To illustrate how dependency injection works, consider the following example. The first file, `logger.service.ts`, defines a `Logger` class. This class contains a `writeCount` function that logs a number to the console.

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

@Injectable({providedIn: 'root'})
export class Logger {
  writeCount(count: number) {
    console.warn(count);
  }
}
```

Next, the `hello-world-di.component.ts` file defines an Angular component. This component contains a button that uses the `writeCount` function of the `Logger` class. To access that function, the `Logger` service is injected into the `HelloWorldDI` class by adding `private logger: Logger` to the constructor.

[Skip to main content](#)

```
import { Logger } from '@angular/core';
import { Logger } from '../logger.service';

@Component({
  selector: 'hello-world-di',
  templateUrl: './hello-world-di.component.html'
})
export class HelloWorldDependencyInjectionComponent {
  count = 0;

  constructor(private logger: Logger) { }

  onLogMe() {
    this.logger.writeCount(this.count);
    this.count++;
  }
}
```

For more information about dependency injection and Angular, see the [Dependency injection in Angular](#) section.

Angular CLI

The Angular CLI is the fastest, straightforward, and recommended way to develop Angular applications. The Angular CLI makes some tasks trouble-free. For example:

[Skip to main content](#)

TAILS

[ng build](#)

Compiles an Angular application into an output directory.

[ng serve](#)

Builds and serves your application, rebuilding on file changes.

[ng generate](#)

Generates or modifies files based on a schematic.

[ng test](#)

Runs unit tests on a given project.

[ng e2e](#)

Builds and serves an Angular application, then runs end-to-end tests.

The Angular CLI is a valuable tool for building out your applications.

For more information about the Angular CLI, see the [Angular CLI Reference](#) section.

First-party libraries

The section, [Angular applications: the essentials](#), provides a brief overview of a couple of the key architectural elements that are used when building Angular applications. The many benefits of Angular really become clear when your application grows and you want to add functions such as site navigation or user input. Use the Angular platform to incorporate one of the many first-party libraries that Angular provides.

Some of the libraries available to you include:

[Skip to main content](#)

DETAILS

Angular Router	Advanced client-side navigation and routing based on Angular components. Supports lazy-loading, nested routes, custom path matching, and more.
Angular Forms	Uniform system for form participation and validation.
Angular HttpClient	Robust HTTP client that can power more advanced client-server communication.
Angular Animations	Rich system for driving animations based on application state.
Angular PWA	Tools for building Progressive Web Applications (PWA) including a service worker and Web application manifest.
Angular Schematics	Automated scaffolding, refactoring, and update tools that simplify development at large scale.

These libraries expand your application's capabilities while also letting you focus more on the features that make your application unique. Add these libraries knowing that they're designed to integrate flawlessly into and update simultaneously with the Angular framework.

These libraries are only required when they can help you add features to your applications or solve a particular problem.

Next steps

This topic gives you a brief overview of what Angular is, the advantages it provides, and what to expect as you start to build your applications.

To see Angular in action, see the [Getting Started](#) tutorial. This tutorial uses stackblitz.com [↗](#), for you to explore a working example of Angular without any installation requirements.

The following sections are recommended to explore Angular's capabilities further:

- [Understanding Angular](#)
- [Angular Developer Guide](#)

Last reviewed on Mon Feb 28 2022