**Angular Tutorial**



Tutorial by: Muhammad Farhan

GitHub: <https://github.com/muhammadfarhandevelper?tab=repositories>

**Introduction to Angular:**

Angular is an **open-source** framework used for creating dynamic and modern web applications. Here are some of the most common applications of Angular.

**Declarative UI:**

Angular uses HTML to define the UI of an application. As HTML is a declarative language, you do not have to define the flows of the program. You can simply define what you require and Angular will take care of it.

**Typescript:**

Angular applications use TypeScript language, which provides higher security as it supports types (primitives, interfaces, etc.) and helps eliminate errors early when writing the code.

**Easy Testing:**

Testing is extremely simple in Angular. The modules have their own application parts, which enable you to perform automated testing. It follows one file, one module principle where you don’t even need to remember the module loading order.

**What is Angular?**

Angular is a JavaScript framework written in TypeScript and maintained by Google. It enables users to develop and test large applications easily. It has surpassed Javascript for developing single-page applications that require modularity, testability, and developer productivity. In this Angular tutorial, you will learn about the different features of Angular, Angular Architecture, its advantages, and its limitations.

**Why do you need a Framework?**

Frameworks in general boost web development efficiency and performance by providing a consistent structure so that developers don’t have to keep rebuilding code from scratch. Frameworks are time savers that offer developers a host of extra features that can be added to software without requiring extra effort.

**Why Angular?**

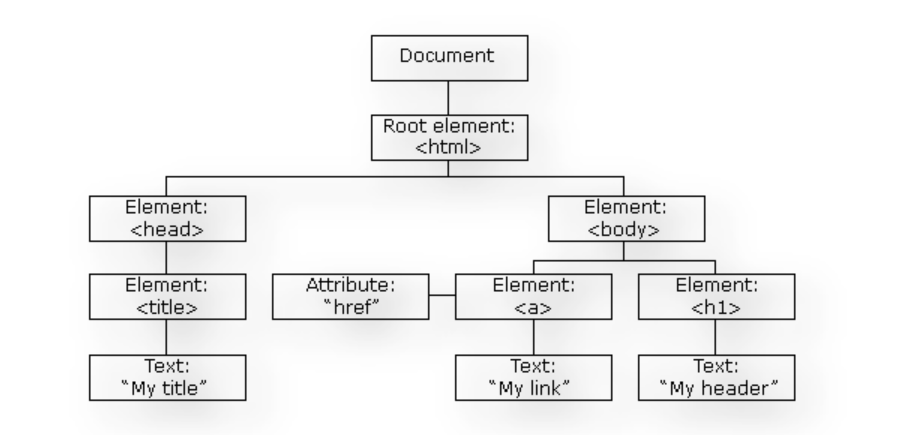
JavaScript is the most commonly used client-side scripting language. It is written into HTML documents to enable interactions with web pages in many unique ways. As a relatively easy-to-learn language with pervasive support, it is well-suited to develop modern applications.

But is JavaScript ideal for developing single-page applications that require modularity, testability, and developer productivity?

**Features of Angular:**

1. **Document Object Model:**

DOM (Document Object Model) treats a HTML document as a tree structure in which each node represents a part of the document.



1. **TypeScript:**

TypeScript defines a set of types to JavaScript, which helps users write JavaScript code that is easier to understand. All of the TypeScript code compiles with JavaScript and can run smoothly on any platform. TypeScript is not compulsory for developing an Angular application. However, it is highly recommended as it offers better syntactic structure—while making the codebase easier to understand and maintain.



You can install TypeScript as an NPM package with the following command:

npm install -g typescript

1. **Data Binding in Angular:**

Data binding is a process that enables users to manipulate web page elements through a web browser. It employs dynamic HTML and does not require complex scripting or programming.

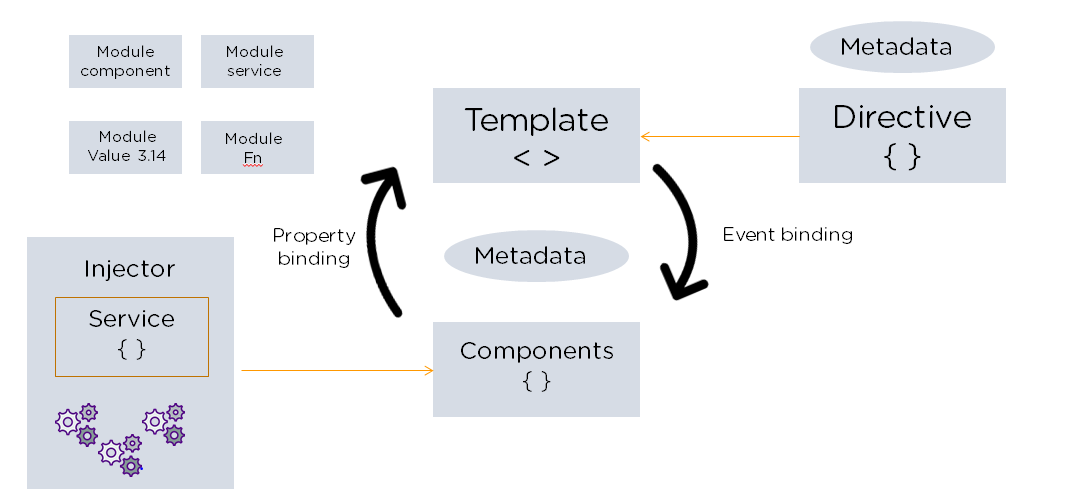
Data binding in Angular defines the communication between components and views. It consists of two types (one-way and two-way data binding) which enables you to exchange data from DOM to component and vice-versa.

1. **Testing:**

Angular uses the Jasmine testing framework. The Jasmine framework provides multiple functionalities to write different kinds of test cases. Karma is the task-runner for the tests that uses a configuration file to set the start-up, reporters, and testing framework.

**Angular Architecture:**

Angular is a full-fledged model-view-controller (MVC) framework. It provides clear guidance on how the application should be structured and offers bi-directional data flow while providing real DOM.



The following are the eight building blocks of an Angular application:

### 1. Modules:

An Angular app has a root module, named AppModule, which provides the bootstrap mechanism to launch the application.

### 2. Components:

Each component in the application defines a class that holds the application logic and data. A component generally defines a part of the user interface (UI).

### 3. Templates:

The Angular template combines the Angular markup with HTML to modify HTML elements before they are displayed. There are two types of data binding:

1. Event binding: Lets your app respond to user input in the target environment by updating your application data.
2. Property binding: Enables users to interpolate values that are computed from your application data into the HTML.

### 4. Metadata:

Metadata tells Angular how to process a class. It is used to decorate the class so that it can configure the expected behavior of a class.

### 5. Services:

When you have data or logic that isn’t associated with the view but has to be shared across components, a service class is created. The class is always associated with the @Injectible decorator.

### 6. Dependency Injection:

This feature lets you keep your component classes crisp and efficient. It does not fetch data from a server, validate the user input, or log directly to the console. Instead, it delegates such tasks to the services.

Angular comes with its own set of advantages and disadvantages. The next two sections briefly explain them.

**AngularJS Directives:**

AngularJS directives extend the HTML by providing it with new syntax. You can easily spot directives because they have the prefix “ng-.” Consider them markers on the DOM element, instructing AngularJS to attach a certain behavior to the element, or even change it outright.

Here are two sample directives:

* The ng-model Directive

The ng-model binds the value of the HTML control with the specified AngularJS expression value.

* The ng-bind Directive

This directive replaces the HTML control value with a specified AngularJS expression value.

**Advantages of Angular:**

Many versions of Angular have been released since its inception. All these versions have added to the efficient working of the framework.

### 1. Custom Components:

Angular enables users to build their own components that can pack functionality along with rendering logic into reusable pieces. It also plays well with web components.

### 2. Data Binding:

Angular enables users to effortlessly move data from JavaScript code to the view, and react to user events without having to write any code manually.

### 3. Dependency Injection:

Angular enables users to write modular services and inject them wherever they are needed. This improves the testability and reusability of the same services.

### 4. Testing:

Tests are first-class tools, and Angular has been built from the ground up with testability in mind. You will have the ability to test every part of your application—which is highly recommended.

### 5. Comprehensive:

Angular is a full-fledged framework and provides out-of-the-box solutions for server communication, routing within your application, and more.

### 6. Browser Compatibility:

Angular is cross-platform and compatible with multiple browsers. An Angular application can typically run on all browsers (Eg: Chrome, Firefox) and OSes, such as Windows, macOS, and Linux.

**What Are the Differences Between Angular and AngularJS?**

Angular is the catch-all term for every version of the framework (1-11), while AngularJS is the initial Angular version, renamed. Although it’s over ten years old, AngularJS isn’t obsolete; it still finds lots of use developing smaller web applications.

Here’s a handy chart outlining some of the basic differences.

|  |  |  |
| --- | --- | --- |
|  | AngularJS | Angular |
| Architecture | Supports mode-view component design | Uses directives and components |
| Language | JavaScript | Microsoft’s TypeScript |
| Mobile capability | No mobile browser support | Supported by all popular mobile browsers |
| Structure | Not as manageable as Angular, but ideal for small applications | Easier to build and maintain large applications |
| Routing | Uses $routeprovider.when() for routing configuration | Uses @Route Config{(…)} for routing configuration |
| Performance | Not as fast as Angular | Faster than AngularJS |

**Angular Creating First Application:**

**Angular Prerequisites:**

There are three main prerequisites.

* 1. **NodeJS** node –v
  2. **Angular CLI** installation: npm install –g @angular/cli
  3. **Text Editor** visual studio code

**Create First Application:**

1. ng create project\_name
2. cd project\_name
3. ng serve

**Introduction to Angular Components:**

Angular components are the building blocks of an application. These components are associated with a template and they define different aspects of the user interface. In this Angular tutorial, you will learn about the different Angular components and will be able to create your first Angular component. You will also learn in this Angular tutorial about component decorator metadata that provides information about the component.

**What is Bootstrap and How to Embed Bootstrap into Angular?**

Bootstrap is a powerful toolkit used for creating and building responsive web pages and web applications. It is a free and open-source project created by Twitter and hosted on GitHub. In this Angular tutorial, you will learn about the step-by-step procedure to embed Bootstrap into your Angular application and create an interactive Webpage.

**Introduction to Angular Service and its Features:**

**Interpolation:**

Interpolation refers to embedding expressions into marked up text. By default, interpolation uses the double curly braces {{ and }} as delimiters.

**In Component:**

export class AppComponent {

  title = 'batch09f';

  data  = "Muhammad Farahn";

  num = 11;

}

**In View:**

  <h1>App Componenet update work ! {{data}} </h1>

  <p>This is number : {{num}}</p>

}

**Angular CLI Commands:**

**Make Component:**

  ng generate componenet user

**Another Command:**

  ng g c admin

**Make Module:**

  ng g m userwork

**Make Component in Module:**

  ng g c userwork/myuser

**Make Class:**

  ng generate class myclass

**Build Project:**

  ng build

**Component:**

Components are the main building block for Angular applications. Each component consists of:

* An HTML template that declares what renders on the page
* A TypeScript class that defines behavior
* A CSS selector that defines how the component is used in a template
* Optionally, CSS styles applied to the template

**Let’s practice:**

**What is Module:**

Angular applications are modular and Angular has its own modularity system called NgModules. NgModules are containers for a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities. They can contain components, service providers, and other code files whose scope is defined by the containing NgModule. They can import functionality that is exported from other NgModules, and export selected functionality for use by other NgModules.

Every Angular application has at least one NgModule class, [the root module](https://angular.io/guide/bootstrapping), which is conventionally named AppModule and resides in a file named app.module.ts. You launch your application by bootstrapping the root NgModule.

**NgModules and Components:**

NgModules provide a compilation context for their components. A root NgModule always has a root component that is created during bootstrap but any NgModule can include any number of additional components, which can be loaded through the router or created through the template. The components that belong to an NgModule share a compilation context.



**Command:**

ng generate module admin

**Steps:**

* Import module in app module
* Module implement in app module
* Export component of module

Styling use with component

Styling with whole application

**Property Binding:**

Property binding in Angular helps you set values for properties of HTML elements or directives. Use property binding to do things such as toggle button features, set paths programmatically, and share values between components.

**Example:**

<[img](https://angular.io/api/common/NgOptimizedImage) alt="item" [src]="itemImageUrl">

**Style Binding:**

 <p [style.color]="color">THis is Text</p>

 <p [style.color]="’red’">THis is Text</p>

**How to Bootstrap Add:**

**If Else Condition:**

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.

**Example of IF:**

 <p \*ngIf="1==1">This is true 1==1 </p>

**Example of If Else:**

<div \*ngIf="1 == 1; then myblock else elseblock">This is True </div>

<ng-template #myblock>This is true Condition !!!  </ng-template>

<ng-template #elseblock>This is False Condition !!!  </ng-template>

**Else IF:**

<ng-template [ngIf]="marks>=90 && marks<=100">

  <h1>Marks is {{marks}} and Grade: A+1</h1>

</ng-template>

<ng-template [ngIf]="marks>=80 && marks<90">

  <h1>Marks : {{marks}} and Grade: A+</h1>

</ng-template>

<ng-template [ngIf]="marks>=70 && marks<80">

  <h1>Marks : {{marks}} and Grade: A</h1>

</ng-template>

<ng-template [ngIf]="marks>=60 && marks<70">

  <h1>Marks : {{marks}} and Grade: B</h1>

</ng-template>

<ng-template [ngIf]="marks>=50 && marks<60">

  <h1>Marks : {{marks}} and Grade: C</h1>

</ng-template>

**For LOOP:**

 <p \*ngFor="let item of names">

  <span> Name:  {{item}} </span>

 </p>

**Form:**

If forms are a central part of your application, scalability is very important. Being able to reuse form models across components is critical.

Use Form Module in App Module

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

import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';

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

@NgModule({

  declarations: [

    AppComponent,

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    FormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

**In View:**

<form #myform="ngForm" (ngSubmit)="getdata(myform.value)">

  <label for="">Enter Name </label>

  <input type="text" ngModel name="name" id="" class="form-control">

  <label for="">Enter Email </label>

  <input type="email" ngModel name="email" id="" class="form-control">

  <label for="">Enter Passowrd </label>

  <input type="passowrd" ngModel name="password" id="" class="form-control">

<br>

  <button class="btn btn-primary">Register</button>

</form>

**Sharing Data Between Child and Parent Component:**

**Parent Component:**

<app-student [item]="'my Data'" ></app-student>

**Child TS FILE:**

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

@Component({

  selector: 'app-student',

  templateUrl: './student.component.html',

  styleUrls: ['./student.component.css']

})

export class StudentComponent {

  @Input() item = '';

}

**Child View:**

{{item}}

**Two Way Binding:**

Two-way binding gives components in your application a way to share data. Use two-way binding to listen for events and update values simultaneously between parent and child components.

In TS:

  data:any;

In HTML

<input type="text" [(ngModel)]="data"/>

{{data}}

**What is Pipe?**

Pipes are simple functions to use in template expressions to accept an input value and return a transformed value. Pipes are useful because you can use them throughout your application, while only declaring each pipe once. For example, you would use a pipe to show a date as **April 15, 1988** rather than the raw string format.

**Example:**

{{"Farhan" | lowercase}}

Angular website: <https://angular.io/guide/pipes-overview>