Introduction to Angular

What is Angular?

Angular is a **TypeScript-based open-source web application framework** developed and maintained by **Google**. It is used for building **single-page applications (SPAs)** with a **component-based architecture**.

Key Features of Angular

1. **Component-Based Architecture** – Applications are structured using components, making them modular and reusable.
2. **Two-Way Data Binding** – Synchronizes data between the model and the view automatically.
3. **Dependency Injection (DI)** – Manages dependencies efficiently, improving maintainability.
4. **Directives** – Extends HTML with custom behavior (e.g., `*ngFor`, `*ngIf`).
5. **Routing** – Built-in router for creating SPAs with multiple views.
6. **Reactive Forms & Template-Driven Forms** – Enables robust form handling and validation.
7. **RxJS (Reactive Extensions for JavaScript)** – Facilitates reactive programming with observables.
8. **CLI (Command Line Interface)** – Provides powerful commands for project creation, testing, and deployment.

Angular vs AngularJS

Feature	Angular (2+)	AngularJS (1.x)
Language	TypeScript	JavaScript
Architecture	Component-based	MVC-based
Performance	Faster	Slower
Mobile Support	Yes	No

Basic Angular Architecture

1. **Modules (`@NgModule`)** – Define the structure of the application.
2. **Components (`@Component`)** – The building blocks of the UI.
3. **Templates & Directives** – Define the HTML structure and behavior.

4. **Services & Dependency Injection (`@Injectable`)** – Handle business logic and data fetching.
5. **Routing (`RouterModule`)** – Manages navigation between different views.

Setting up an Angular application

To set up an Angular application, follow these steps:

---

1. Install Prerequisites

Before installing Angular, make sure you have the required dependencies:

Install Node.js and npm

Angular requires **Node.js** and **npm (Node Package Manager)**.

- **Download and install Node.js** from Node.js Official Website
- Verify installation by running:

  ```
  node -v
  npm -v
  ```

Install Angular CLI

Angular CLI (Command Line Interface) simplifies Angular development.
Install it globally using:

```
npm install -g @angular/cli
```

Verify installation:

```
ng version
```

---

2. Create a New Angular Project

Once Angular CLI is installed, create a new Angular application using:

```
ng new my-angular-app
```

- **It will prompt you for configuration options**
  - Would you like to add Angular routing? (Yes/No)
  - Choose a CSS preprocessor (CSS, SCSS, SASS, LESS)

Move into the project directory:

```
cd my-angular-app
```

---

3. Serve the Application

To start the development server, run:

```
ng serve --open
```

- This command will **compile the project** and start a local server at `http://localhost:4200/`
- The `--open` flag automatically opens the browser

---

4. Understanding the Project Structure

Once the project is created, you will see the following structure:

```
my-angular-app/
│-- e2e/                # End-to-end testing
│-- node_modules/       # Installed dependencies
│-- src/                # Main project source code
│   │-- app/            # Application modules &
components
│   │-- assets/         # Static assets like images,
fonts
│   │-- environments/   # Environment-specific
configurations
│   │-- main.ts         # Main entry point
│   │-- index.html      # Main HTML file
│   │-- styles.css      # Global styles
│-- angular.json        # Angular project configuration
│-- package.json        # Dependencies and scripts
│-- tsconfig.json       # TypeScript configuration
```

---

5. Generate Components, Services, and Modules

Angular CLI makes it easy to generate new files:

Create a new Component

```
ng generate component my-component
```

or

```
ng g c my-component
```

Create a new Service

```
ng generate service my-service
```

Create a new Module

```
ng generate module my-module
```

---

6. Build the Application

To build a production-ready version:

```
ng build --prod
```

This generates a `dist/` folder containing the optimized production files.

---

7. Deploying the Application

You can deploy the Angular app using different services like:

- **GitHub Pages** (`ng deploy`)

- **Firebase Hosting**
- **Netlify / Vercel**
- **Apache / Nginx Server**

For example, to deploy on Firebase:

```
npm install -g firebase-tools
firebase login
firebase init
ng build --prod
firebase deploy
```

Components, modules, and services

Components

- The building blocks of an Angular application.
- They define the **UI and logic** for a part of the app.
- Every component consists of:
  - **TypeScript file** (`.ts`) → Logic & data handling
  - **HTML file** (`.html`) → View/template
  - **CSS file** (`.css`) → Styles

Create a Component

```
ng generate component my-component
```

or

```
ng g c my-component
```

This generates:

```
my-component/
│-- my-component.component.ts     # Component logic
│-- my-component.component.html   # Component template
│-- my-component.component.css    # Component styles
```

Modules

- Organize the app into feature-based or reusable units.
- The **root module** is `app.module.ts`.

Create a Module

```
ng generate module my-module
```

or

```
ng g m my-module
```

Import a module into another module:

```
import { MyModule } from './my-module/my-module.module';
@NgModule({
  imports: [MyModule]
})
export class AppModule { }
```

Services

- Handle **business logic, API calls, and shared data**.
- **Use Dependency Injection (DI)** to make them reusable.

Create a Service

```
ng generate service my-service
```

or

```
ng g s my-service
```

Register it in `app.module.ts` or inside a specific module.

Example: Service fetching data

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

@Injectable({
  providedIn: 'root'
})
```

```
export class MyService {
  constructor(private http: HttpClient) {}

  getData() {
    return
this.http.get('https://api.example.com/data');
  }
}
```

Data Binding and Templates

Data binding connects **TypeScript logic** with the **HTML template**.

Types of Data Binding

1. **Interpolation (`{{ }}`)** – Bind component variables to HTML.

   ```
   html
   <h1>{{ title }}</h1>
   ```

2. **Property Binding (`[]`)** – Bind an HTML element's property.

   ```
   html
   <img [src]="imageUrl">
   ```

3. **Event Binding (`()`)** – Trigger methods on events.

   ```
   html
   <button (click)="sayHello()">Click Me</button>
   ```

4. **Two-Way Binding (`[()]`)** – Syncs input field with component variable.

   ```
   html
   <input [(ngModel)]="username">
   ```

Example: Component Data Binding

```
ts
export class MyComponent {
  title = 'Angular App';
  imageUrl = 'assets/image.png';
```

```
  sayHello() {
    alert('Hello from Angular!');
  }
}
```

Forms and Validation

Angular supports **Template-driven Forms** and **Reactive Forms**.

Template-driven Forms (For simpler forms)

Steps:

1. Import `FormsModule` in `app.module.ts`:

   ```
   ts
   import { FormsModule } from '@angular/forms';
   @NgModule({ imports: [FormsModule] })
   ```

2. Create a form in the template:

   ```
   html
   <form #userForm="ngForm">
     <input type="text" name="username" ngModel
   required>
     <button
   [disabled]="!userForm.valid">Submit</button>
   </form>
   ```

Reactive Forms (For complex forms)

1. Import `ReactiveFormsModule` in `app.module.ts`:

   ```
   ts
   import { ReactiveFormsModule } from
   '@angular/forms';
   @NgModule({ imports: [ReactiveFormsModule] })
   ```

2. Define the form in the component:

```
ts
import { FormGroup, FormControl, Validators } from
'@angular/forms';

export class MyComponent {
  userForm = new FormGroup({
    username: new FormControl('',
Validators.required),
  });

  submit() {
    console.log(this.userForm.value);
  }
}
```

3. Create the form in the template:

```
html
<form [formGroup]="userForm" (ngSubmit)="submit()">
  <input type="text" formControlName="username">
  <button type="submit">Submit</button>
</form>
```

Routing and Navigation

Routing enables **navigation between different views**.

Steps to Setup Routing

1. Import `RouterModule` in `app.module.ts`:

```
ts
import { RouterModule, Routes } from
'@angular/router';
import { HomeComponent } from
'./home/home.component';
import { AboutComponent } from
'./about/about.component';

const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: '', redirectTo: '/home', pathMatch:
'full' },
];

@NgModule({ imports:
[RouterModule.forRoot(routes)] })
export class AppModule {}
```

2. Add navigation links in `app.component.html`:

```
html
<nav>
  <a routerLink="/home">Home</a>
  <a routerLink="/about">About</a>
</nav>
<router-outlet></router-outlet>
```

HTTP and Observables

Angular uses the `HttpClient` module to make API calls.

Steps to Use HTTP in Angular

1. Import `HttpClientModule` in `app.module.ts`:

```
ts
import { HttpClientModule } from
'@angular/common/http';
@NgModule({ imports: [HttpClientModule] })
```

2. Use `HttpClient` in a Service:

```
ts
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

export class ApiService {
  constructor(private http: HttpClient) {}
  getUsers(): Observable<any> {
    return
this.http.get('https://jsonplaceholder.typicode.com
/users');
  }
}
```

3. Call API in a Component:

```
ts
export class MyComponent {
  users: any = [];

  constructor(private apiService: ApiService) {}

  ngOnInit() {
    this.apiService.getUsers().subscribe(data =>
this.users = data);
  }
}
```

4. Display data in the template:

```
html
<ul>
  <li *ngFor="let user of
users">{{ user.name }}</li>
</ul>
```