ANGULAR - Advanced Front End Technology

Introduction

Angular is an open-source web application framework developed by Google, used to create dynamic web applications and mobile applications. It provides a comprehensive solution for building complex single-page applications.

Official Website: https://angular.dev/

Installation and Setup

Angular CLI Installation

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

Check Angular CLI Version

```
ng version
```

Create a New Angular Project

```
ng new project-name
```

Run Angular Project

```
ng serve
```

View output at: http://localhost:4200/

Project Structure

Root Files

- **tsconfig.spec.json**: Unit testing configuration
- **tsconfig.json**: TypeScript configuration
- **server.ts**: Server-side rendering configuration
- **package.json**: NPM configuration
- **package-lock.json**: Version details of installed packages
- **angular.json**: Angular project configuration
- **node_modules/**: Folder containing installed packages

Source Folder (`src/`)

- **styles.css**: Global styles
- **main.ts**: Entry point file that bootstraps the root component
- **index.html**: Entry point for UI (contains root component selector)

App Folder (`src/app/`)

- Root component folder containing:
  - TS file (component logic)
  - HTML file (view)
  - CSS file (styles)
  - spec.ts file (unit testing)
- **app.config.ts**: Configure application features like routing
- **app.routes.ts**: Define routes for components

Core Concepts

Data Types

```
variableName: type = value;
```

Data Binding

One-Way Binding

TS File to HTML File:

1. **Interpolation**: `{{ classProperty }}`
2. **Property/Attribute Binding**: `[tagAttribute]="classProperty"`

HTML File to TS File:

1. **Event Binding**: `(eventName)="functionCall()"`
2. **Event Binding with $event**: `(eventName)="functionName($event)"`
3. **Event Binding with Template Reference Variable**: `(eventName)="functionName(templateRefVar)"` (Use `#variableName` to create template reference variables)

Two-Way Binding

Using `ngModel` Directive: `[(ngModel)]="property"`
Requires importing `FormsModule` in the component

Directives

Instructions that modify the behavior of HTML elements, attributes, or properties in a component.

Types of Directives:

1. **Component Directives**:
   - Display components using their selectors in HTML pages
2. **Attribute Directives**:
   - Provide styling to HTML elements
   - **ngClass**: Provides conditional styling (`[ngClass]="condition ? 'class1' : 'class2'"`)
   - **ngStyle**: Applies inline styles (`[ngStyle]="{'property': value}"`)
3. **Structural Directives**:
   - Change the structure of HTML by adding/removing elements
   - Preceded by asterisk (*) symbol
   - **ngIf**: Conditionally renders elements (`*ngIf="condition"`)
   - **ngFor**: Iterates through lists (`*ngFor="let item of items"`)
   - **ngSwitch**: Conditionally renders based on switch cases

Control Flow Statements

Modern alternatives to structural directives:

- **@if**, **@else if**, **@else**: Conditional rendering
- **@for**: List iteration
- **@switch**: Switch case rendering

Pipes

Transform data display without changing the actual data.

Built-in Pipes

- **DatePipe**: Format dates (`{{ date | date:'shortDate' }}`)
- **UpperCasePipe**: Convert to uppercase (`{{ text | uppercase }}`)
- **LowerCasePipe**: Convert to lowercase (`{{ text | lowercase }}`)
- **Syntax**: `data | pipeName[:option]`

Custom Pipes

Generate with:

```
ng g p pipe-folder/pipe-name
```

Define transformation logic in the `transform()` method.

Services

Provide common logic and data sharing between components.

Creating a Service

```
ng g s service-folder/service-name
```

Asynchronous Operations

Angular uses **Observables** to handle asynchronous operations, which can manage multiple asynchronous functions simultaneously.

- **subscribe()** method: Returns the resolved state of an Observable

```
observable.subscribe(
  // Success callback
  (response) => console.log(response),
  // Error callback (optional)
  (error) => console.error(error)
);
```

Or using an object:

```
observable.subscribe({
  next: (response) => console.log(response),
  error: (error) => console.error(error)
});
```

API Calls

Use `HttpClient` class to make API calls:

- Import `provideHttpClient` in app.config.ts from '@angular/common/http'
- HTTP methods return Observables

Dependency Injection (DI)

Transfer dependent class features to another class through constructor injection:

```
constructor(
  private/public/protected propertyName: DependentClassName
) { }
```

Component Lifecycle Hooks

Methods that run at specific points in a component's lifecycle:

1. Creation Phase

- **constructor**: Runs when component is instantiated

2. Change Detection Phase

- **ngOnInit**: Runs once after component initializes inputs
- **ngOnChanges**: Runs every time component inputs change
- **ngDoCheck**: Runs when changes are detected
- **ngAfterContentInit**: Runs once after component content is initialized
- **ngAfterContentChecked**: Runs after every content check
- **ngAfterViewInit**: Runs once after component view is initialized
- **ngAfterViewChecked**: Runs after every view check

3. Rendering Phase

- **afterNextRender**: Runs once when component is next rendered to DOM
- **afterRender**: Runs every time components are rendered to DOM

4. Destruction Phase

- **ngOnDestroy**: Runs once before component is destroyed/removed

Interfaces

Define custom data types for your project:

```
ng g interface folder-name/interface-name
```

Forms

Template-Driven Forms

- Form is created in the component template
- Data is fetched using directives like `ngModel` and `ngForm`
- Import `FormsModule`

Model-Driven (Reactive) Forms

- Form model is created in the component first
- Model is bound to the template using controls
- Import `ReactiveFormsModule`
- Uses `FormGroup`, `FormArray`, and `FormControl` from `FormBuilder`

Components

UI building blocks that encapsulate specific functionality.

Creating a Component

```
ng g c component-name
```

Component Communication

Parent to Child

Use `@Input()` decorator in child component to receive data from parent.

Child to Parent

Use `@Output()` decorator with `EventEmitter` in child component to send events to parent.

Routing

Navigate between different views/components.

Setting Up Routes

In `app.routes.ts` :

```
const routes: Routes = [
  { path: 'path', component: ComponentName },
  { path: '**', component: PageNotFoundComponent } // Wildcard route
];
```

- Use `<router-outlet></router-outlet>` in root component HTML to display routed components
- For dynamic routing: `{ path: ':paramName', component: ComponentName }`
- For navigation without page refresh: `<a [routerLink]="['/path']">Link</a>`

Accessing Route Parameters

1. Using `ActivatedRoute` :

```
constructor(private route: ActivatedRoute) {
  this.route.params.subscribe(params => {
    const id = params['id'];
  });
}
```

2. Using `@Input()` decorator:
    - Call `withComponentInputBinding()` inside `provideRouter()` in app.config.ts
    - Define dynamic variable with `@Input()` decorator in component

Modules

Group related components and services together.

Creating a Module with Routing

```
ng g m module-name --routing
```

Creating Components in a Module

```
ng g c module-name/component-name --no-standalone
```

Lazy Loading Modules

In `app.routes.ts`:

```
{
  path: 'module-path',
  loadChildren: () => import('./module-path').then(m => m.ModuleName)
}
```

Guards

Protect routes based on conditions.

Creating a Guard

```
ng g g guard-folder/guard-name
```

Types of Guards

1. **CanActivate**: Determines if a route can be activated
2. **CanActivateChild**: Determines if child routes can be activated
3. **CanDeactivate**: Determines if a route can be deactivated
4. **Resolve**: Pre-fetches data before route activation
5. **CanLoad**: Determines if a module can be lazily loaded

Best Practices

- Use services for data sharing between components
- Implement lazy loading for better performance
- Follow Angular style guide for naming conventions
- Use TypeScript interfaces for type safety

- Properly unsubscribe from Observables to prevent memory leaks