**Angular 18 Components**

**Introduction**

Angular is a popular front-end framework developed by Google for building dynamic and responsive web applications. Components are the building blocks of an Angular application. In this chapter, we'll cover the basics of creating and using components in Angular 18.

**Prerequisites**

Before we start, make sure you have the following installed:

* Node.js (v14.x or later)
* Angular CLI (v18.x or later)

You can install the Angular CLI globally using the following command:

npm install -g @angular/cli

**Creating a New Angular Project**

Let's start by creating a new Angular project:

ng new angular-components-tutorial

cd angular-components-tutorial

This command will generate a new Angular project with all the necessary files and configurations.

**Generating a New Component**

In Angular, a component is a TypeScript class that is associated with an HTML template and a CSS style. You can generate a new component using the Angular CLI:

ng generate component my-component

This will create the following files in the src/app/my-component/ directory:

* my-component.component.ts: The TypeScript file for the component.
* my-component.component.html: The HTML template for the component.
* my-component.component.css: The CSS styles for the component.
* my-component.component.spec.ts: The test file for the component.

**Component Structure**

Let's take a closer look at the structure of a component.

**my-component.component.ts**

This file contains the TypeScript class that defines the component. Here's what it looks like:

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

@Component({

selector: 'app-my-component',

templateUrl: './my-component.component.html',

styleUrls: ['./my-component.component.css']

})

export class MyComponent {

title = 'Hello, Angular 18!';

changeTitle(newTitle: string) {

this.title = newTitle;

}

}

* **@Component decorator**: This decorator defines metadata for the component, such as the selector, template URL, and style URLs.
* **selector**: This is the name of the custom HTML tag that will be used to embed this component in a template.
* **templateUrl**: This points to the HTML file that contains the template for this component.
* **styleUrls**: This points to the CSS file(s) that define styles for this component.

**my-component.component.html**

This is the HTML template for the component:

<div class="my-component">

<h1>{{ title }}</h1>

<button (click)="changeTitle('New Title')">Change Title</button>

</div>

* **{{ title }}**: This is an example of Angular's interpolation, which binds the component's title property to the view.
* **(click)**: This is an event binding that calls the changeTitle method when the button is clicked.

**my-component.component.css**

This file contains the styles for the component:

.my-component {

text-align: center;

margin-top: 20px;

}

.my-component h1 {

color: #007bff;

}

**Using the Component**

To use the newly created component, you'll need to add its selector to the template of another component, such as the root AppComponent.

**app.component.html**

Modify the app.component.html file to include the my-component selector:

<div class="app">

<app-my-component></app-my-component>

</div>

**Running the Application**

Now that you've created a component and included it in your app, let's run the application:

ng serve

Open your browser and navigate to http://localhost:4200/. You should see the title "Hello, Angular 18!" and a button that changes the title when clicked.

**Input and Output Properties**

Components in Angular can communicate with each other using Input and Output properties.

**Input Example**

Let's modify the MyComponent to accept a title as an input property.

**my-component.component.ts:**

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

@Component({

selector: 'app-my-component',

templateUrl: './my-component.component.html',

styleUrls: ['./my-component.component.css']

})

export class MyComponent {

@Input() title: string = 'Default Title';

}

**app.component.html:**

<app-my-component [title]="'Custom Title'"></app-my-component>

Now, the MyComponent will display "Custom Title" instead of the default title.

**Output Example**

Let's add an output event that emits when the title is changed.

**my-component.component.ts:**

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

@Component({

selector: 'app-my-component',

templateUrl: './my-component.component.html',

styleUrls: ['./my-component.component.css']

})

export class MyComponent {

@Input() title: string = 'Default Title';

@Output() titleChanged = new EventEmitter<string>();

changeTitle(newTitle: string) {

this.title = newTitle;

this.titleChanged.emit(this.title);

}

}

**app.component.html:**

<app-my-component [title]="'Custom Title'" (titleChanged)="onTitleChanged($event)"></app-my-component>

**app.component.ts:**

export class AppComponent {

onTitleChanged(newTitle: string) {

console.log('Title changed to:', newTitle);

}

}

**Conclusion**

This chapter covers the basics of creating and using components in Angular 18. We've seen how to generate a component, define its template and styles, use input and output properties, and communicate between components. With these foundational skills, you can start building more complex and dynamic Angular applications.