Angular Quickstart Live Coding Lecture

Welcome to this live coding session! Together, we'll build a simple Angular application while exploring essential concepts like components, data binding, and managing state. Let's get started step by step.

Step 1: Create the App

Goal: Set up a new Angular app and understand how its content appears on the screen.

I. **Install Angular CLI:**

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

II. **Create a New Angular App:**

```
ng new angular-demo
```

- When prompted, choose "CSS" for styling.

III. **Navigate to the Project Directory:**

```
cd angular-demo
```

IV. **Start the Development Server:**

```
ng serve
```

- Open your browser at `http://localhost:4200`. The default Angular welcome page should appear.

Step 2: Create and Use Components with Angular CLI

Goal: Learn to create components and add them to your app.

I. **Generate a Header Component:**

```
ng generate component header --skip-tests
```

- This creates a new folder: `src/app/header/` with files for your component's logic, template, and styles.

II. **Edit `header.component.ts`:**

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

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent {
  title = 'Welcome to Angular!';
}
```

III. **Edit `header.component.html`:**

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

IV. **Use the Header Component in the App:** Edit `app.component.html`:

```
<app-header></app-header>
<router-outlet></router-outlet>
```

V. **Save and Refresh the Browser:**

- You should see the title displayed in the header.

Step 3: Store Data in the Component Class

Goal: Add and display static data in the header component.

I. **Edit `header.component.ts` to Store Data:**

```
export class HeaderComponent {
  title = 'Welcome to Angular!';
  description = 'This is a quick intro to Angular
}
```

II. **Update `header.component.html` to Display Data:**

```
<header>
  <h1>{{ title }}</h1>
  <p>{{ description }}</p>
</header>
```

III. **Save and Verify:**

- The header should now display both the title and description.

Step 4: Add Interactivity with Two-Way Binding

Goal: Bind a text input to the component's data using two-way binding.

I. **Enable FormsModule:** In `header.component.ts` file import `FormsModule` to show two way binding

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

@Component({
  selector: 'app-header',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './header.component.html',
  styleUrl: './header.component.css'
})
export class HeaderComponent {
title = 'Welcome to Angular'
inputText = ''
}
```

II. **Edit `header.component.ts` to Add a Property:**

```
export class HeaderComponent {
  title = 'Welcome to Angular!';
  inputText = '';
}
```

III. **Update `header.component.html` for Two-Way Binding:**

```
<header>
  <h1>{{ title }}</h1>
  <input [(ngModel)]="inputText" placeholder="Typ
  <p>You typed: {{ inputText }}</p>
</header>
```

IV. **Save and Test:**

- Type in the input box and watch the text update dynamically.

Step 5: Add Event Binding for Buttons

Goal: Add a button to reset the input field using event binding.

I. **Edit `header.component.ts` to Add a Reset Method:**

```
export class HeaderComponent {
  title = 'Welcome to Angular!';
  inputText = '';

  resetInput() {
    this.inputText = '';
  }
}
```

II. **Update `header.component.html` to Add a Button:**

```
<header>
  <h1>{{ title }}</h1>
  <input [(ngModel)]="inputText" placeholder="Typ
  <button (click)="resetInput()">Reset</button>
  <p>You typed: {{ inputText }}</p>
</header>
```

III. **Save and Test:**

- Type something in the input field, then click the Reset button to clear it.

Step 6: Manage State Dynamically

Goal: Use a button to toggle a message on and off.

I. **Edit `header.component.ts` to Add State Management:**

```
export class HeaderComponent {
  title = 'Welcome to Angular!';
  showMessage = false;

  toggleMessage() {
    this.showMessage = !this.showMessage;
  }
}
```

II. **Update `header.component.html` to Toggle Content:**

```
<header>
  <h1>{{ title }}</h1>
  <button (click)="toggleMessage()">Toggle Messag
  @if (showMessage){
  <p >This is a toggled message!</p>
 }
</header>
```

III. **Save and Test:**

- Click the Toggle Message button to show or hide the message.

Class Summary:

In this class, we explored the fundamentals of Angular and how to build a simple Angular application using the Angular CLI. We learned the key concepts of Angular components, data binding, and state management. The class covered:

I. **Setting Up an Angular App**: We started by setting up an Angular project using the Angular CLI to streamline the development process.

II. **Angular Components**: We learned how to create and structure Angular components, which are the building blocks of an Angular application.

III. **String Interpolation**: We used string interpolation to display data dynamically from the component class to the template.

IV. **Two-Way Binding**: We implemented two-way binding with `ngModel` to synchronize data between the template and component class.

V. **Event Binding**: We used event binding to respond to user interactions, such as button clicks, and trigger component methods.

VI. **State Management**: We managed state within a component to allow dynamic changes in the UI based on user interaction, such as toggling a message.

Exercises:

I.
Create a `FooterComponent`:

- Create a new component named `FooterComponent`.
- In the component, display a copyright message, such as "© 2024 YourCompany".

II.
Add a Dynamic Title to the Header:

- Add a new button to the header that changes the title of the header component when clicked.
- Use event binding to trigger the title change method.

III.
Style the Header and Footer Components:

- Use CSS to add styling to both the header and footer components.

- Add padding, margins, background colors, and text alignment.

IV.
Create a `ToggleButtonComponent`:

- Create a new component that has a button to toggle between two texts (e.g., "Show" and "Hide").
- Use state management to keep track of the current text.

V.
Implement Two-Way Binding with `ngModel`:

- Create an input field bound to a component property using two-way binding (`[(ngModel)]`).
- Display the updated value dynamically below the input.

VI.
Create a `CounterComponent`:

- Create a `CounterComponent` that has a counter that increments or decrements with a button click.
- Display the current value of the counter in the template.

VII.
Add a Conditional Message:

- Use `@if` to conditionally show or hide a message based on a boolean property in the component.

VIII.
Pass Data from Parent to Child Components:

- Create a parent component that passes data to a child component using input bindings.
- Display the passed data in the child component.

IX.
Handle Form Submission:

- Create a form with an input field and a submit button.
- Bind the form data to a property in the component and log the submitted data when the form is submitted.

X.
Create a Dynamic List with `@if`:

- Create an array of items in the component and display them in the template using the `@if` directive.
- Add a button that adds an item to the list dynamically.