The Basics

How an Angular App Gets Loaded and Started

- Starting CLI Commands
  - Create a new Angular project: **ng new <project-name>**
    - Prompts for routing/styling
  - Serve an Angular project: **ng serve**
    - Defaults to **localhost:4200**
- In the end, Angular is a JS framework, changing the DOM (HTML) at runtime
- The **index.html** file is served by the server
  - Base of the **Single-Page Application (SPA)**
  - Contains scripts that start the Angular app
  - Contains **<app-root>**
    - Component that houses the entire application
- The **main.ts** file contains the code first executed by the server
  - Checks for imports, production, and connects **AppModule** for known components

Components are Important!

- Components build an entire Angular project
- **app.component.ts** is the root that holds the entire application, including other components
  - May include components for a header, nav-bar, and side-panel, each containing their own components
- Each component may have its own HTML, styling, business logic, etc.
- Can be reused

Creating a New Component

- CLI: **ng generate component <component-name>**
- Naming: **<component-name>.component.ts**
- Components are TS classes that must be exported

- Components contain an **@Component** decorator
  - **selector** - The HTML tag used to refer to the component
  - **templateUrl** - The HTML code of the component
  - **styleUrls** - The CSS/SCSS of the component

Understanding the Role of App Module and Component Declaration

- Angular uses components to build web pages, and modules to bundle components into packages
- **app.module.ts** contains and gives the application's functionalities to Angular
- Contains the **ngModule** decorator
- Contains **bootstrap**
  - Tells Angular which components to pay attention to when the application starts
- New components must be registered in the **declarations** array so Angular knows they exist

Using Custom Components

- Insert your first custom component by adding it to **app.component.html**

Creating Components with the CLI and Nesting Components

- CLI: **ng generate component <component-name>**
  - Shorten with **ng g c <component-name>**
- Automatically adds the necessary information to **app.module.ts**
- Adding instances of a component template to another component's template will embed the first component into it

Working with Component Templates

- Every component needs to at least have a template
- Can change **templateUrl** to **template**
  - Can store actual HTML here instead of a reference to another file
  - Good for one/two lines of code

Working with Component Styles

- Every component comes with a reference for style sheets
- Can add standard CSS/SCSS styles in these style sheets
- Similar to the above component, we can replace **styleUrls** with **styles**
  - Contains an array of string-written styles in CSS format

Fully Understanding the Component Selector

- Selectors operate like CSS selectors
  - We're not limited to selecting by element
- Can make the **selector** contain an attribute or class
- id and pseudo-selectors don't work for referencing to components

What is Databinding?

- Databinding is **communication** between a component's **Typescript code** (business logic) and **Template** (HTML**)**
- Output Data → Typescript Code to Template
  - String Interpolation → {{ data }}
    - Ideal for merely outputting data
  - Property Binding → [property] = "data"
    - If I want to **hide** data using **[hide]**, I must assign that property to a literal value or variable from the component
- React to (User) Events → Template to Typescript Code
  - Event Binding → (event) = "expression"
    - If I want to generate an **alert** by clicking a button, I write **(click)="alert()"** as a button attribute
- Combination of Both → React to events and output data simultaneously
  - Two-Way Binding → [(ngModel)] = "data"

String Interpolation

- Whatever is stored between the double-curly braces is returned as a string
- Can store component variables, literal values, or methods inside the double-curly braces

- Can't write multi-line expressions between the double-curly braces
  - Ternary operators are allowed

Property Binding

- Can change an element's properties given code from the component file
- For instance, can use the **[disable]** property to disable a button until its activation is valid
- Besides binding to HTML properties, we can also bind to directives and entire components

Property Binding vs String Interpolation

- Use string interpolation when you want to output something to the template
- Use property binding when you want to change some property of an element, directive, or component
- Don't mix property binding and string interpolation
  - **Avoid** → [disabled]="{{ expression }}"

Event Binding

- Represented by enclosing an event name between parentheses
  - Example → **(click)="onMethodName()"**
- Can store component methods or literal instructions within the quotation marks

Bindable Properties and Events

- You can bind to all properties and events
  - **console.log()** the element to see all available properties and events

Passing and Using Data with Event Binding

- **$event**
  - Represents a reserved variable name available in the template when using event binding
  - As a parameter, it will act as the data emitted with that event
  - Example → **(input) = "returnText($event)"** → Returns whatever is in the input box to the component

FormsModule is Required for Two-Way Binding!

- Must enable the **ngModel** directive
  - Done by adding **FormsModule** to the **imports[ ]** in the **AppModule**

Two-Way Databinding

- Look immediately above for pre-requisites
- Combines property- and event-binding
- Do so by using both syntaxes → **[( )]**
  - Example → **[(ngModel)] = "componentString"**
- Triggers on the input event and updates the given value in the server automatically

Understanding Directives

- Directives are instructions in the DOM
  - Classes that can add new behavior to the elements in the template or modify existing behavior
- Example → Angular, please add our component to this specific place
- Can build custom directives
- Typically applied as an attribute

Using ngIf to Output Data Conditionally

- **Structural** Directive
  - Changes the structure of the DOM
  - Preceded by the asterisk (*****) → ***ngIf = <expression>**
- When the given expression evaluates to **true**, Angular renders the template provided in the **Then** clause
  - Else, it renders **false** or **null**

Enhancing ngIf with an Else Condition

- Add an **else** clauses within the **ngIf** expression
  - Example → ***ngIf = "serverCreated; else noServer"**
- Can define noServer with an **ng-template** block and local reference
  - **ng-template**

        - A built-in Angular component that allows us to mark places in the DOM
        - Let's us define template content that is only being rendered by Angular when you, whether directly or indirectly, specifically instruct it to do so
    - The **ng-template** element contains a **local reference** as an attribute
        - Consider it a marker
        - Example **#noServer**

Styling Elements Dynamically with ngStyle

- **Attribute** Directive
    - Unlike **Structural** Directives, these don't add or remove elements
    - Modify the elements they're assigned to
- Uses **Property Binding** to add styles
    - Key: Value pairs
    - Can use a **ternary operator** as the **value**
    - Example → **[ngStyle] = "backgroundColor: 'red'"**
    - Example → **[ngStyle] = "{backgroundColor: isRed ? 'red' : 'green'}"**

Applying CSS Classes Dynamically with ngClass

- Allows us to dynamically add or remove CSS/SCSS classes
- Uses **Property Binding** to add styles
    - Key: Value pairs
        - **Key** → CSS class name
        - **Value** → Condition for whether the class is added
    - Example → **[ngClass] = "{ online: serverStatus === 'online' }"**

Outputting Lists with ngFor

- **Structural Directive**
    - Adds or modifies the DOM
    - Requires the asterisk (*****)
        - Example → ***ngFor = "let <temp-variable> of <array>"**
- Loops through all elements of a given array, performing specified instructions on each element

Getting the Index When Using ngFor

- We can get the index of the current iteration by declaring one in the ***ngFor** call
- Example → ***ngFor = "let logItem of log; let i = index;"**
- The index starts at **zero (0)**