Angular 15 Comprehensive Guide

Single Page Applications (SPA)

What are Single Page Applications?

Single Page Applications (SPAs) load a single HTML page and dynamically update content as the user interacts with the app. They offer a more fluid user experience, similar to a desktop application. SPAs rely heavily on JavaScript to manipulate the DOM without requiring full page reloads.

Characteristics of Single Page Applications

- **Speed and Responsiveness**: Faster navigation within the app as only the necessary content is updated.
- **Reduced Server Load**: Fewer requests to the server since most resources are loaded once.
- **Improved User Experience**: Smooth transitions and dynamic updates.

JavaScript Frameworks for SPA

- **Angular**: A platform for building mobile and desktop web applications.
- **React**: A library for building user interfaces.
- **Vue.js**: A progressive framework for building user interfaces.

Angular

What is Angular?

Angular is a platform and framework for building single-page client applications using HTML, CSS, and TypeScript. It is maintained by Google and offers a robust framework for developing dynamic web apps.

Angular Environment and IDEs

- **Environment**: Node.js, npm, Angular CLI.
- **IDEs**: Visual Studio Code, WebStorm, Atom, Sublime Text.

Installing Angular CLI and Creating a Sample Project

1. Install Node.js and npm.
2. Install Angular CLI: `npm install -g @angular/cli`.
3. Create a project: `ng new my-app`.

Directory Structure and Various Configuration Files

- Key directories: `src/app`, `src/assets`, `src/environments`.
- Configuration files: `angular.json`, `package.json`, `tsconfig.json`.

Bootstrapping Angular Application

The `main.ts` file is the entry point. The `AppModule` is the root module.

Understanding the Role of AppModule and Component Declaration

- **AppModule**: Defines the root module and imports necessary modules.
- **Component Declaration**: Declares components used in the app.

Angular CLI Commands

- `ng serve`: Serves the application.
- `ng generate component component-name`: Generates a new component.
- `ng build`: Builds the application.
- `ng test`: Runs tests.

Installing Bootstrap and Styling Your Application

1. Install Bootstrap via npm: `npm install bootstrap`.
2. Import Bootstrap in `angular.json`.

Decorators in Angular

Special functions that modify classes or properties. Examples: `@Component`, `@NgModule`, `@Injectable`.

Components

Building blocks of Angular applications. Consist of a template, style, and logic.

Manually Creating Components

Use Angular CLI: `ng generate component component-name`.

Working with Component Templates and Component Styles

Define HTML templates and CSS styles within components.

Data Binding

Data Binding

Synchronizing data between model and view.

String Interpolation

Embedding expressions in templates using `{{}}`.

Property Binding

Binding component properties to HTML element properties.

One Way Binding

Data flows from model to view.

Two Way Binding
Data flows both ways between model and view using `[(ngModel)]`.

Event Binding
Binding events to methods in the component.

Passing Data between Components
Using Input and Output decorators.

Directives

Understanding Directives
Instructions in the DOM.

Structural Directives and Attribute Directives
- **Structural**: `*ngIf`, `*ngFor`.
- **Attribute**: `ngClass`, `ngStyle`.

ngIf to Output Data Conditionally
Conditionally include a template.

ngIf with an Else Condition
`*ngIf="condition; else elseBlock"`.

Styling Elements Dynamically with ngStyle
Applying styles dynamically.

CSS Classes Dynamically with ngClass
Applying CSS classes dynamically.

Outputting Lists with ngFor
Looping through arrays in templates.

Getting the Index when using ngFor
Using `let i = index` to get the index.

Understanding ngSwitch
Conditional rendering with multiple conditions.

Host Listeners and Bindings

HostListener to Listen to Host Events
Listening to DOM events on the host element.

HostBinding to Bind to Host Properties
Binding properties to the host element.

Error Handling and Debugging

Understanding Angular Error Messages
Common Angular errors and their meanings.

Using VS Code to Debug Angular Applications
Setting breakpoints and inspecting variables.

Using Angular DevTools to Dive into Angular Apps
Inspecting Angular-specific information.

Creating and Using Global Custom ErrorHandlers
Customizing global error handling.

Using the new ng Object
Exploring the `ng` object for debugging.

Lifecycle Hooks

Understanding the Component Lifecycle
Angular component lifecycle hooks.

Seeing Lifecycle Hooks in Action
Demonstrating various lifecycle hooks.

Tracking Changes
Using hooks like `ngOnChanges`.

Lifecycle Hooks and Template Access
Using hooks to access the template and DOM.

ViewChild and ContentChild

@ViewChild()
Accessing child components or DOM elements.

Getting Access to the Template & DOM with @ViewChild
Manipulating the DOM using `@ViewChild`.

Projecting Content into Components with ng-content
Inserting content into components dynamically.

@ContentChild() decorator
Accessing projected content.

Getting Access to ng-content with @ContentChild
Accessing the content projected into a component.

Pipes

Pure and Impure Pipes

Pure and Impure Pipes
Pure pipes: no side effects.
Impure pipes: may have side effects.

Using Built-in Pipes
Common built-in pipes like `DatePipe`, `UpperCasePipe`.

The PipeTransform Interface
Creating custom pipes.

Creating Custom Pipes
Implementing the `PipeTransform` interface.

Performance Implications of Impure Pipes
Impure pipes run more frequently, affecting performance.

Dependency Injection (DI)

Understanding DI in Angular
Angular's DI system for managing dependencies.

Constructor-based Hierarchical Injector
Injectors and providers hierarchy.

Singleton and Non-Singleton Objects
Service instances in Angular.

Using Service Classes
Creating and using services.

Working with Various Decorators in DI
Using `@Injectable`, `@Injector`.

@Injector()
Using the Injector API.

@Injectable()
Defining services.

Forms

Building Template Forms and Reactive Forms
Template-driven vs. reactive forms.

Understanding FormBuilder, FormGroup, FormControl Classes
Reactive forms classes.

Validating Forms
Form validation techniques.

Creating Custom Validators
Custom validation logic.

Implementing Asynchronous Validations
Async validators for forms.

Submitting Form Data to the Server
Handling form submissions.

RXJS and Http Programming

RXJS Introduction
Reactive programming with RxJS.

Observables, Observers, and Subjects
Key concepts in RxJS.

Consuming Web Services
Making HTTP requests.

Passing Parameters using HTTP Headers
Sending headers with requests.

Creating Custom Http Interceptors
Intercepting HTTP requests.

Error Handling in RXJS
Managing errors in streams.

Routing

Routing Introduction
Angular's router module.

Route Definition Object
Defining routes.

Router Outlets
Placeholder for routed components.

Passing Parameters to Routes
Route parameters.

Lifecycle of Router
Router lifecycle events.

Implementing Guards and Resolvers
Route guards and resolvers.