Here are common interview questions and answers for the Angular framework, focusing on core concepts and best practices.

**Angular Framework**

**1. What is Angular? What are its key features?**

* **Answer:** Angular is a robust, open-source, and component-based front-end framework developed and maintained by Google. It's used for building scalable, high-performance, and dynamic single-page applications (SPAs).
* **Key Features:**
  + **Component-Based Architecture:** Applications are built as a tree of components, which are reusable, self-contained building blocks of the UI.
  + **TypeScript:** Angular is built on TypeScript, a superset of JavaScript that adds static typing and object-oriented features, leading to more robust and maintainable code.
  + **Dependency Injection (DI):** Angular has a powerful built-in DI system for providing services to components, making them loosely coupled and easy to test.
  + **Routing:** A feature-rich routing system for managing navigation between different views or components in an SPA.
  + **Angular CLI:** A command-line interface that simplifies the development workflow, including scaffolding projects, generating components, services, and running tests.
  + **Directives:** Allow developers to extend the power of HTML by adding custom behavior to elements.
  + **Data Binding:** A powerful feature that synchronizes data between the component's logic and the view's template.

**2. Explain the difference between a Component and a Module in Angular.**

* **Answer:**
  + **Component:** A component is the fundamental building block of an Angular application. It's a class with a @Component decorator that defines a view (HTML template), logic (TypeScript class), and styles (CSS). A component is responsible for a specific part of the UI.
  + **Module:** An Angular module (NgModule) is a container that groups related components, directives, pipes, and services. It provides a way to organize the application into functional blocks. Every Angular application has at least one root module (AppModule), which bootstraps the application. Modules are crucial for organizing code, managing dependencies, and enabling lazy loading.

**3. What is Data Binding in Angular? Explain the different types.**

* **Answer:** Data binding is the mechanism that connects the component's data with the view's template. It automatically synchronizes the data, so changes in one are reflected in the other.
  + **Interpolation ({{ ... }})**: One-way data binding from the component to the view. It's used to display a component's property value in the HTML. Example: <h1>{{ pageTitle }}</h1>
  + **Property Binding ([property] = "expression")**: One-way data binding from the component to the view. It sets the value of an HTML element's property (e.g., [src], [class]) to a component property. Example: <img [src]="imageUrl">
  + **Event Binding ((event) = "statement")**: One-way data binding from the view to the component. It listens for a DOM event (e.g., (click), (submit)) and executes a method in the component. Example: <button (click)="saveData()">Save</button>
  + **Two-Way Data Binding ([(ngModel)] = "property")**: A combination of property and event binding. It synchronizes data between the component and the view in both directions. A change in the input field updates the component property, and a change in the component property updates the view. This requires the FormsModule to be imported. Example: <input [(ngModel)]="userName">

**4. What are Directives in Angular? Explain the different types.**

* **Answer:** Directives are classes that add new behavior to elements in the DOM.
  + **Component Directives:** This is the most common type. Components are, in fact, directives with a template.
  + **Structural Directives:** They change the DOM layout by adding, removing, or manipulating elements. They are identified by a \* prefix.
    - \*ngIf: Conditionally adds or removes an element from the DOM based on a boolean condition.
    - \*ngFor: Renders a template for each item in a collection.
    - \*ngSwitch: A set of directives that conditionally display content based on a match.
  + **Attribute Directives:** They change the appearance or behavior of an element, component, or another directive.
    - ngStyle: Dynamically sets CSS styles on an element.
    - ngClass: Dynamically adds or removes CSS classes from an element.

**5. What is the Angular CLI, and what are some of its key commands?**

* **Answer:** The Angular CLI (Command Line Interface) is a powerful tool for developing Angular applications. It automates common development tasks.
* **Key Commands:**
  + ng new my-app: Creates a new Angular project with a basic structure and required files.
  + ng serve: Builds and serves the application locally, watching for changes and live-reloading the browser.
  + ng generate component my-component (or ng g c my-component): Generates a new component with its associated files (.ts, .html, .css, .spec.ts).
  + ng generate service my-service (or ng g s my-service): Generates a new service class.
  + ng test: Runs the unit tests for the application.
  + ng build: Compiles the application into an output directory, ready for deployment.

Certainly! Here are more in-depth and advanced interview questions for Angular, covering topics that demonstrate a deeper understanding of the framework.

**Advanced Angular Questions**

**1. Explain the Angular Lifecycle Hooks.**

* **Answer:** Angular components and directives have a lifecycle managed by the framework. There are various lifecycle hooks that allow you to tap into key moments in a component's life, from its creation to its destruction.
  + ngOnChanges(): Called when Angular sets or resets data-bound input properties.
  + ngOnInit(): A one-time initialization hook called after ngOnChanges on the first call. It's the recommended place to perform initialization logic like fetching data from a service.
  + ngDoCheck(): Called during every change detection run. Use this to implement your own custom change detection logic.
  + ngAfterContentInit(): Called after Angular projects external content into the component's view.
  + ngAfterContentChecked(): Called after every check of the component's content.
  + ngAfterViewInit(): Called after a component's view and its child views are initialized. Use this to access and manipulate view children.
  + ngAfterViewChecked(): Called after every check of the component's view and its child views.
  + ngOnDestroy(): A cleanup hook called just before Angular destroys the component. Use this to unsubscribe from observables, detach event handlers, etc., to prevent memory leaks.

**2. What is Dependency Injection in Angular? How does it work?**

* **Answer:** Dependency Injection (DI) is a core design pattern in Angular that allows a class to receive its dependencies from an external source rather than creating them itself. This promotes loose coupling, making components more modular, reusable, and testable.
* **How it works:**
  + **Provider:** An object that tells Angular how to create a service. Providers are registered in an injector.
  + **Injector:** A hierarchical system that is responsible for creating and delivering dependencies. When a component requests a service, the injector finds the correct provider and creates the instance.
  + **@Injectable() Decorator:** This decorator marks a class as a service that can be injected. The providedIn: 'root' option makes the service a singleton that is available throughout the application.
  + **Constructor Injection:** The most common way to request a dependency. You list the services you need as parameters in the component's constructor. Angular's DI system automatically provides the correct instances.

**3. Differentiate between a Promise and an Observable.**

* **Answer:** This is a crucial question that tests your understanding of asynchronous programming in Angular.
  + **Promises:**
    - **Single Value:** A promise handles a single, future value. It either resolves with a value or rejects with an error.
    - **Not Cancellable:** Once a promise is created, it cannot be canceled.
    - **Eager:** The function passed to the promise constructor executes immediately.
    - **Error Handling:** Errors are handled with a .catch() method.
  + **Observables:**
    - **Stream of Values:** An observable can emit multiple values over time (a stream). It can also emit zero or one value.
    - **Cancellable:** You can cancel an observable using the unsubscribe() method, which is important for preventing memory leaks.
    - **Lazy:** An observable does nothing until a consumer subscribes to it.
    - **Error Handling:** Errors are handled by an error callback in the subscribe method or using RxJS operators like catchError.
    - **Operators:** Observables come with a rich set of operators from the RxJS library (map, filter, debounceTime, etc.) that allow you to transform and combine data streams.

**4. What is a "Change Detection Strategy," and what is the difference between Default and OnPush?**

* **Answer:** Change detection is the process by which Angular synchronizes the application state with the view. The strategy determines when the change detector for a component should run.
  + **Default Strategy (ChangeDetectionStrategy.Default):** This is the default. Angular checks every component in the component tree from top to bottom on every single change detection cycle. A cycle is triggered by user events (clicks), HTTP requests, setTimeout(), and other asynchronous operations. This can lead to performance issues in large applications.
  + **OnPush Strategy (ChangeDetectionStrategy.OnPush):** This is an optimization strategy. Angular will only run change detection for a component and its children when:
    - An input property reference changes (not just a mutation of an object).
    - An event is emitted from the component or one of its children.
    - The component is explicitly marked as "dirty" using ChangeDetectorRef.markForCheck().
    - An observable tied to the component's view with the async pipe emits a new value.
    - **Benefit:** OnPush significantly improves performance by reducing the number of components Angular needs to check during each cycle.

**5. How do you handle and debug asynchronous operations in Angular?**

* **Answer:**
  + **HTTP Client:** Use the built-in HttpClient module to make HTTP requests, as it returns observables.
  + **RxJS:** The RxJS library is essential for managing asynchronous data streams. You use operators like map, filter, tap, catchError, and switchMap to handle and transform data.
  + **Error Handling:** Use the catchError operator in your service methods to gracefully handle errors from the API. The HttpClient throws an HttpErrorResponse that you can inspect.
  + **Unsubscribing:** Always unsubscribe from long-lived observables (setInterval, custom event streams, etc.) in the ngOnDestroy() lifecycle hook to prevent memory leaks. The async pipe handles unsubscription automatically.
  + **Debugging:**
    - **Browser DevTools:** Use the Network tab to inspect HTTP requests and responses.
    - **RxJS tap operator:** The tap operator is invaluable for debugging an observable stream. It allows you to log values at any point without affecting the stream.
    - **rxjs-spy:** An external library for more advanced RxJS debugging.

These questions delve into the more intricate parts of Angular, showcasing a candidate's practical experience and deeper knowledge of the framework's architecture and performance optimization.