I'm not a bot

Practicing answers to anticipated technical interview questions helps build confidence and prepares candidates for challenging queries. Utilizing a mock interview or solo practice can significantly enhance preparation. Acquiring expertise in Angular, an open-source framework developed by Google, is highly valued in Full-Stack Engineer and Front-End Engineer roles. Familiarity with Angular will enable you to tackle common interview questions effectively. Angular applications rely on the angular.json file for configurations and declarations within specific modules. The application's components are declared using selectors, templates, and stylesheets. These elements provide access information, HTML content, and style sheets unique to each component. By bootstrapping the application through Angular, it sets up a browser environment that serves necessary data from the index.html file. Prior to Angular's introduction, developers utilized VanillaJS and jQuery for dynamic websites. However, issues arose as features increased in complexity. Angular addresses these problems by dividing code into smaller components and providing built-in features such as routing, state management, and HTTP services. The framework uses HTML for rendering UI, making it easier to use than JavaScript. With Google's long-term support announcement, Angular solidifies its position as a leading client-side framework. AngularJS and Angular: Key Differences in Command and Operating Languages AngularJS utilizes JavaScript and the MVC design model, whereas Angular employs TypeScript with components and directives. Angular's architecture differs from AngularJS's in several ways, including language, structure, expression syntax, and mobile support. In terms of programming languages, Angular uses TypeScript which is statically typed and provides better performance for larger applications compared to AngularJS which utilizes a dynamically typed language. Expression syntax also varies between the two frameworks. AngularJS requires developers to recall precise ng-directives for binding events or properties, whereas Angular employs simpler attributes for property and event binding. The architectural difference lies in the use of components instead of controllers as directives in Angular, whereas AngularJS uses an MVC architecture consisting of a model, controller, and view. For larger applications, maintaining code is more complicated and time-consuming in AngularJS due to its structure. In contrast, Angular's design makes it easier to maintain large-scale codebases. AOT (Ahead-of-Time) compilation is used in Angular because it compiles components and templates before running inside the browser. This process enables fast rendering as the application can load and render executable code immediately. AOT also reduces AJAX requests for source files by including HTML and CSS into the compiled JS files, thus improving security. Data binding plays a crucial role in connecting application data with the DOM, featuring four main forms: event binding, property binding, two-way binding, and string interpolation binding. Decorators are used to add metadata to classes, objects, or methods without changing their source code. Annotations are pre-defined features that hold this metadata. When applied to a class, they create an array of annotations stored within the "annotations" attribute and pass metadata into the constructor. Angular expressions bind application data to HTML and evaluate expressions against locally scoped objects. They can handle null and undefined values and use pipes for data formatting. In contrast, JavaScript expressions are evaluated against the global window object and cannot access properties outside their local scope. Components in Angular go through a lifecycle of phases when created. Lifecycle hooks are used to track a component's state and trigger changes during specific phases. These hooks include ngOnChanges, ngOnInit, ngDoCheck, ngAfterContentInit, ngAfterContentChecked, ngAfterViewInit, ngAfterViewChecked, and ngOnDestroy. Directives are classes that can be imported into components to add similar functionality. They have their own behavior and are declared using the @Directive decorator. Three types of directives are component directives, structural directives, and attribute directives. Full of tips and advice to make your interviews a success. Our guide on answering behavioral interview questions is another great resource to check out. Plus, our Career Center offers even more job-hunting tools, from resume writing to networking to building a portfolio that showcases your skills. And if you're looking to learn something new or level up an existing skill, explore our catalog for courses in web development, math, data science, programming languages, and more. Read on to prepare for your dream job interview! In this article, I provide answers to 10 questions to help you understand the basics of Angular and its framework architecture. Enjoy! We'll start with a fundamental question: when would you use the constructor() versus the ngOnInit() method? To answer that, we need to explore component lifecycle and the role of the constructor. Angular creates components based on two phases: constructing the component tree and running change detection. The constructor() method is invoked first. Component Lifecycle Hooks are methods on Components or Directives that Angular calls at a specific moment in the change detection process. The ngOnInit() method is the second, called once, indicating that the object is ready to use since Angular has set all input properties and displayed data-bound properties. Want more about lifecycle hooks? We have a series covering them all! Start with our guide to OnInit and follow along. The code added to the constructor is always initialized before the ngOnInit() method. Be sure that logic set in the constructor isn't added too early, when the component is out of control. Directives might seem simple, but even experienced Angular devs haven't grasped every concept yet. We'll explore advanced topics like Observables and Async Pipe Identity Checking and Performance, Web Components, and more! we should focus on two main aspects: Network Performance and Runtime Performance, to improve the load time of our Angular app. Firstly, we need to optimize bundle size by using Ahead-of-Time compilation. This method can also enhance Runtime Performance by reducing computations required for rendering. Furthermore, removing unused code through techniques such as template whitespace removal, code splitting, minification, and tree-shaking is crucial. Secondly, we must improve Runtime Performance by optimizing the Change Detection process. We do this by disabling unnecessary Change Detection, only running it when needed, and applying strategies like onPush, detaching and reattaching custom Change Detection, or using pipes instead of functions in interpolations. Another area for improvement is rendering DOM elements efficiently. Using virtual scrolling, ng-container, and trackBy function can help minimize the number of DOM elements, reducing loading time. Lastly, Angular takes inspiration from Web Components to implement custom elements. When it comes to writing code in a framework-agnostic way, using custom elements in Angular offers several advantages. It improves reusability and readability of your app, making it more consistent and maintainable. Additionally, custom elements allow you to add components to an app at runtime. To learn more about Web Components, visit The Ultimate Guide to Web Components. Web Components map Angular functionality to native HTML elements, making them universal and compatible with any browser that supports custom elements through the Web Platform feature (polyfills). To create a custom element in Angular, you extend the NgElement interface and define a tag. This results in components that look and behave like regular HTML elements. The @angular/elements package is crucial for this implementation. You can add it to your app using a single command in the CLI. The CreateCustomElement API exports a basic interface to create cross-framework components by attaching DOM API functionality of Angular's components and change detection features. This allows you to transform Angular components into elements understandable by browsers while providing all infrastructure specific to Angular. Custom elements automatically connect with view, change detection system, and data binding process defined in the component. They also bootstrap themselves with an automated lifecycle, starting automatically when added to DOM and destroyed when removed. For example, you can convert a regular Angular component to a custom element using the createCustomElement function from @angular/elements. AoT (Ahead-of-Time) Compilation is a way to compile an Angular app at build time, allowing a browser to understand the templates and components provided by Angular. This process is crucial for app performance, as it efficiently tree-shakes code during bundling, removes unused directives, and eliminates asynchronous requests. It also reduces application payloads by providing a smaller bundle size that doesn't need to be downloaded in full. Ahead-Of-Time Compilation: Benefits and Alternatives in Angular Apps === To optimize runtime performance, developers can leverage Ahead-Of-Time (AOT) compilation. However, its benefits extend beyond performance improvements. Benefits of AOT Compilation ---------------------------- 1. **Security**: AOT compilation allows for error validation before serving the app to clients. This helps detect and prevent errors, as well as injection attacks. 2. **Code Protection**: By compiling the app before deployment, developers can ensure that sensitive data is not exposed. Using AOT Compilation --------------------- To use AOT compilation, run commands like `ng build --aot` or `ng serve --aot`. Alternatively, building in production mode enables AOT compilation by default. Sharing Data between Components ------------------------------ When sharing data between components, developers can opt for: 1. **Decorators**: Using `@Input()` and `@Output()` to share data between related components. 2. **Services**: Providing an interface for bi-directional communication using services like `BehaviourSubject()` or `Subject()`. 3. **Redux Pattern**: Storing app data in the ngrx store and passing it to components via selectors, decoupling component interactions. Modular Architecture with Lazy Loading -- Lazy loading is a design pattern that loads specific modules only when needed. This approach reduces bundle size and improves loading times. To implement lazy loading: 1. **Route Configurations**: Define lazy-loading modules as part of route configurations. 2. **Dynamic Imports**: Utilize Angular's Dynamic Imports feature in version 8. Example ------- ```javascript { path: '/admin', loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule) } ``` Best Practices for Lazy Loading -------------------------------- 1. **Declare Default Pages as Non-Lazy**: Avoid applying lazy loading to default routes, as this can lead to unexpected behavior. 2. **Consider Bundle Size and Performance**: Optimize module loading based on your app's complexity and performance requirements. By understanding the benefits and implementation details of AOT compilation, services, and lazy loading, developers can optimize their Angular apps for improved performance, security, and maintainability. 1. To speed up page loading times, consider optimizing initial HTTP requests and computations to reduce unnecessary overhead during page rendering. 2. Observables and reactive programming are well-suited for Angular due to their ability to handle asynchronous data streams effectively. This approach can simplify code and improve performance. 3. Within Angular, various features such as HTTP methods, Router events, and ActivatedRoute parameters inherently return observable values, making them suitable for reactive programming. 4. Reactive programming operators enable you to transform, filter, and process data more efficiently, resulting in clearer and more understandable code. 5. The RxJS library provides numerous tools to simplify data operations and improve overall app performance. 6. Observables excel at handling asynchronous events, providing benefits over traditional Promise-based solutions like JavaScript's built-in Promises. 7. One of the key advantages of using observables is their ability to deliver multiple values in a synchronized or asynchronous manner, allowing for easier error management and control. 8. A more efficient alternative to Template-Driven Forms is Reactive Forms. While it may seem counterintuitive, Reactive Forms offer better form validation, centralized logic, and improved unit testing capabilities. 9. By utilizing content projection, developers can inject dynamic content into a child component from its parent, enhancing code readability and reusability. 10. Implementing uni-directional data flow in Angular leads to more predictable behavior and efficient component communication within the application tree. Unidirectional data flow in Angular facilitates a smoother change detection process by enabling only one-way updates to leaf nodes, which ensures that all leaf components are refreshed simultaneously when their parent component changes. This approach not only improves efficiency but also promotes predictability and helps avoid complex cycles within the Change Detection mechanism.

Angular 10 interview questions and answers pdf. Angular interview questions and answers part 2. Angular 10 interview questions and answers for experienced. Angular 9 interview questions. Top 10 angular interview questions and answers. #angularinterviewquestions. Angular 10 interview questions.