Angular Interview Preparation for 2 Years of Experience

June 16, 2025

Introduction

This document provides a comprehensive guide for preparing for an Angular interview with approximately 2 years of experience. It includes key questions, concise answers, and practical tips to help you articulate your expertise effectively.

1 Key Questions and Answers

1.1 What is Angular, and how does it differ from AngularJS?

Answer: Angular is a TypeScript-based, component-driven framework for building dynamic single-page applications (SPAs). Unlike AngularJS (1.x), which uses JavaScript and controllers, Angular (2+) features:

- Component-based architecture for modularity.
- TypeScript for strong typing.
- Improved performance with Ahead-of-Time (AOT) compilation.
- RxJS for reactive programming.
- Angular CLI for streamlined development.

Tip: Highlight Angulars modern features and mention any experience upgrading from AngularJS.

1.2 What are the key building blocks of an Angular application?

Answer: Key building blocks include:

- **Modules**: Organize components and services (`NgModule`).
- **Components**: Define UI with templates and logic.
- **Templates**: HTML views with data binding.
- **Directives**: Extend HTML (`ngFor`, `ngIf`).
- **Services**: Handle business logic.

- **Dependency Injection**: Manages dependencies.
- **Routing**: Enables navigation.
- **Pipes**: Transform data for display.

Tip: Discuss how youve used these in projects, e.g., creating a custom pipe.

1.3 What is the difference between NgModule and a JavaScript module?

Answer:

- **NgModule**: Angular-specific, organizes components and services using `@NgModule`.
- **JavaScript Module**: ES6 module using `import/export` for code organization.

```
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  bootstrap: [AppComponent]
})
export class AppModule { }

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

Tip: Explain how `NgModule` supports lazy loading.

1.4 What is data binding in Angular? Explain its types.

Answer: Data binding synchronizes component and template data. Types:

1. **Interpolation** (`{{ }}`): Displays data, e.g., `{{ user.name }}`.
2. **Property Binding** (`[property]="value"`): Binds to element properties, e.g., `[disabled]="isDisabled"`.
3. **Event Binding** (`(event)="handler()"`): Handles DOM events, e.g., `(click)="onClick()"`.
4. **Two-Way Binding** (`[(ngModel)]="value"`): Combines property and event binding.

```
<input [value]="username" (input)="username=$event.target.value">
<input [(ngModel)]="username">
```

Tip: Share a form where you used two-way binding.

1.5 What is the difference between ngOnInit and constructor?

Answer:

- **Constructor**: Initializes class, used for dependency injection.
- **ngOnInit**: Lifecycle hook for component initialization.

```
export class MyComponent implements OnInit {
  constructor(private service: MyService) { }
  ngOnInit() {
    this.service.getData().subscribe(data => this.data = data);
  }
}
```

Tip: Emphasize using `ngOnInit` for setup tasks.

1.6 How does Angulars Dependency Injection work?

Answer: DI injects dependencies via constructors using providers defined in `NgModule` or `Component`. It uses a hierarchical injector tree.

```
@Injectable({ providedIn: 'root' })
export class DataService {
  getData() { return 'Sample Data'; }
}

@Component({...})
export class MyComponent {
  constructor(private dataService: DataService) {
    console.log(this.dataService.getData());
  }
}
```

Tip: Mention a singleton service you created.

1.7 What is the purpose of async pipe, and how does it work?

Answer: The `async` pipe subscribes to Observables/Promises, updates the template, and unsubscribes automatically.

```
@Component({
  template: `<div>{{ data$ | async }}</div>`
})
export class MyComponent {
  data$ = this.service.getData();
  constructor(private service: DataService) {}
}
```

Tip: Discuss using `async` for API data.

1.8 How do you implement routing in Angular?

Answer: Use `RouterModule` to define routes, `<router-outlet>` to render components, and `routerLink` for navigation.

```
const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'user/:id', component: UserComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

Tip: Mention lazy loading or guards if used.

1.9 What are Angular directives? Explain the types with examples.

Answer: Directives extend HTML. Types:

1. **Component Directives**: Components with templates.
2. **Structural Directives**: Modify DOM, e.g., `*ngIf`, `*ngFor`.
3. **Attribute Directives**: Alter behavior, e.g., `ngClass`.

```
<div *ngIf="isVisible">Content</div>
<div *ngFor="let item of items">{{ item }}</div>
<div [ngClass]="{'active': isActive}">Styled</div>
```

Tip: Share a custom directive you built.

1.10 How do you optimize Angular application performance?

Answer: Techniques:

- Lazy loading modules.
- AOT compilation (`ng build -aot`).
- `OnPush` change detection.
- `trackBy` in `*ngFor`.
- Minimize bundle size (`ng build -prod`).
- Unsubscribe Observables.

```
trackByFn(index, item) {
  return item.id;
}
```

Tip: Share a specific optimization you implemented.

1.11 What are Angular forms, and what are the differences between Template-Driven and Reactive Forms?

Answer:

- **Template-Driven**: Template-based, simple, uses ngModel.
- **Reactive**: Programmatic, complex, uses FormGroup.

```
form = new FormGroup({
  username: new FormControl('', [Validators.required]),
  email: new FormControl('', [Validators.email])
});
```

Tip: Explain when you chose one over the other.

1.12 How do you handle HTTP requests in Angular?

Answer: Use HttpClient to make requests, handling responses with Observables.

```
@Injectable({ providedIn: 'root' })
export class ApiService {
  constructor(private http: HttpClient) {}
  getUsers() {
    return this.http.get<User[]>('https://api.example.com/users')
      ;
  }
}
```

Tip: Mention error handling or interceptors.

1.13 What is an Angular Interceptor, and how do you use it?

Answer: Interceptors modify HTTP requests/responses, e.g., adding headers.

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler) {
    const authReq = req.clone({
      setHeaders: { Authorization: `Bearer ${localStorage.getItem
        ('token')}` }
    });
    return next.handle(authReq);
  }
}
```

Tip: Share a use case like adding tokens.

1.14 What is lazy loading in Angular, and why is it important?

Answer: Lazy loading loads modules on demand, reducing initial bundle size.

```
const routes: Routes = [
  { path: 'feature', loadChildren: () => import('./feature/
    feature.module').then(m => m.FeatureModule) }
];
```

Tip: Discuss performance improvements.

1.15 How do you handle state management in Angular?

Answer: Use services with `BehaviorSubject`, NgRx, or component inputs/outputs.

```
@Injectable({ providedIn: 'root' })
export class StateService {
  private state = new BehaviorSubject<string>('initial');
  state$ = this.state.asObservable();
  updateState(newState: string) {
    this.state.next(newState);
  }
}
```

Tip: Mention NgRx if used.

2 Preparation Tips

- **Know Your Projects**: Discuss challenges and solutions.
- **Code Examples**: Practice writing snippets.
- **RxJS**: Master operators like `map`, `switchMap`.
- **Performance**: Study lazy loading, `OnPush`.
- **Mock Interviews**: Practice explaining concepts.
- **Stay Updated**: Learn about Angular Signals (17+).
- **Ask Questions**: Inquire about team practices.

3 Additional Questions

- **ViewChild vs. ContentChild**: `DViewChild` accesses template elements, `ContentChild` accesses projected content.
- **Unit Testing**: Use Jasmine/Karma with `TestBed`.
- **Angular Signals**: Reactive state management (17+).
- **Security**: Use guards, sanitizers, and secure APIs.