What is Angular?
A. A programming language
B. A design framework
C. A JavaScript framework
D. A database management system
Answer: C. A JavaScript framework

What is the purpose of Angular CLI (Command Line Interface)?
A. To create graphical user interfaces
B. To manage cloud resources
C. To automate development tasks
D. To write server-side code
Answer: C. To automate development tasks

What is the significance of Angular modules?
A. They define the structure of a database
B. They organize the application into cohesive blocks of functionality
C. They control the styling of UI components
D. They perform asynchronous operations
Answer: B. They organize the application into cohesive blocks of functionality

What is data binding in Angular?
A. A mechanism to bind HTML elements to a data source
B. A way to create database connections
C. A process of compressing data
D. A feature for encrypting data
Answer: A. A mechanism to bind HTML elements to a data source

What is the purpose of ngIf directive in Angular?
A. It creates a loop in the template
B. It conditionsally adds or removes elements from the DOM
C. It defines a new Angular module
D. It initializes a variable
Answer: B. It conditionally adds or removes elements from the DOM

What is Angular CLI used for?
A. To create server-side applications
B. To generate and manage Angular projects
C. To execute SQL queries
D. To design user interfaces

Answer: B. To generate and manage Angular projects

Which decorator is used to define an Angular component?
A. @Module
B. @Component
C. @Directive
D. @Service
Answer: B. @Component

What is the purpose of ngFor directive in Angular?
A. It creates a new Angular form
B. It defines a new function
C. It iterates over a list of items
D. It performs asynchronous operations
Answer: C. It iterates over a list of items

What is Angular Services used for?
A. To define data models
B. To handle communication with a server
C. To style UI components
D. To create animations
Answer: B. To handle communication with a server

How does Angular facilitate two-way data binding?
A. Using the [(ngModel)] directive
B. Using the *ngFor directive
C. Using the (ngIf) directive
D. Using the [ngClass] directive
Answer: A. Using the [(ngModel)] directive

Angular Interview Questions and Answers for 5 Years Experience

Core Concepts and Architecture

1. What are the key architectural differences between AngularJS and Angular (17+)?

Angular (17+) represents a complete rewrite from AngularJS, introducing several fundamental changes:

- Shift from MVC to component-based architecture

- TypeScript instead of JavaScript

- Improved performance through hierarchical Dependency Injection

- More structured and modular approach

- Enhanced mobile support with better performance

- Introduction of CLI for project scaffolding

2. Explain Angular's component lifecycle hooks.

Components go through several lifecycle phases:

- ngOnChanges: When data-bound properties change

- ngOnInit: After first ngOnChanges

- ngDoCheck: Developer's custom change detection

- ngAfterContentInit: After content projection

- ngAfterContentChecked: After content check

- ngAfterViewInit: After view initialization

- ngAfterViewChecked: After view check

- ngOnDestroy: Just before component destruction

3. What is Change Detection in Angular and how does it work?

Change Detection is Angular's process of synchronizing the model with the view. It uses **Zone.js** to:

- Track async operations

- Determine when to update the view

- Implement unidirectional data flow

- Support OnPush strategy for better performance

4. How do you optimize Change Detection in a large Angular application?

Several strategies can be employed:

- Use OnPush Change Detection Strategy

- Implement trackBy function for ngFor

- Use pure pipes instead of methods in templates

- Detach change detector for static components

- Use async pipe for observables

- Implement running change detection outside NgZone

5. Explain Dependency Injection in Angular.

DI is a design pattern where:

- Components declare dependencies in constructor

- Angular's injector maintains singleton instances

- Providers can be configured at different levels

- Hierarchical injection system

- Supports lazy loading and tree-shaking

6. What are Angular Modules and what are their key characteristics?

Angular Modules are containers for:

- Components, directives, and pipes

- Services and other dependencies

- Routing configuration

- Feature organization

They support:

- Lazy loading

- Encapsulation

- Separation of concerns

7. How do you implement lazy loading in Angular?

Lazy loading implementation involves:

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

Key considerations:

- Proper module structure

- Route configuration

- PreloadAllModules strategy if needed

- Proper chunk naming

8. Explain Angular's HTTP interceptors and their uses.

Interceptors are middleware for HTTP requests:

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

Common uses:

- Authentication

- Error handling

- Loading indicators

- Request/Response transformation

9. How do you handle state management in Angular applications?

Several approaches:

- Services with BehaviorSubject

- NgRx for complex applications

- Akita

- NGXS

- Simple services for small applications

Example with BehaviorSubject:

```
@Injectable({
  providedIn: 'root'
})

export class StateService {
  private state = new BehaviorSubject<any>(initialState);
  state$ = this.state.asObservable();
  updateState(newState: any) {
    this.state.next(newState);
  }
}
```

10. What are Angular Guards and how do you implement them?

Guards protect routes based on conditions:

```
@Injectable({
  providedIn: 'root'
})

export class AuthGuard implements CanActivate {

  canActivate(

    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot

  ): boolean | Promise<boolean> | Observable<boolean> {
    return this.authService.isAuthenticated();
  }
}
```

Types include:

- CanActivate

- CanActivateChild

- CanDeactivate

- CanLoad

- Resolve

Advanced Concepts

11. How do you handle forms in Angular? Compare Template-driven vs Reactive forms.

Template-driven forms:

- Simpler, more declarative

- Good for basic forms

- Two-way binding with [(ngModel)]

Reactive forms:

- More robust

- Better for complex validation

- Better unit testing

- Programmatic form creation

Example of Reactive Form:

```
this.form = this.fb.group({
  name: ['', [Validators.required, Validators.minLength(3)]],
  email: ['', [Validators.required, Validators.email]],
  address: this.fb.group({
    street: [''],
    city: ['']
  })
});
```

12. Explain Angular's performance optimization techniques.

Key techniques include:

- OnPush Change Detection

- Pure Pipes

- trackBy for *ngFor

- Lazy Loading

- Preloading strategies

- AOT compilation

- Tree shaking

- Service Worker implementation

13. How do you implement error handling in Angular applications?

Comprehensive error handling includes:

- Global error handler

- HTTP interceptor for API errors

- RxJS error operators

- Try-catch blocks

- Error boundaries

Example:

```
@Injectable()

export class GlobalErrorHandler implements ErrorHandler {
  handleError(error: Error) {
    console.error('An error occurred:', error);
    // Log to server
    // Show user-friendly message
  }
}
```

14. How do you implement authentication in Angular?

Authentication implementation involves:

- JWT token management

- Route guards

- HTTP interceptors

- Session management

- Secure storage

15. Explain Angular's testing framework and approaches.

Testing involves:

- Unit tests with Jasmine

- Integration tests

- E2E tests with Protractor/Cypress

- TestBed configuration

- Component testing

- Service testing

- Mock services

Example:

```
describe('MyComponent', () => {

  let component: MyComponent;
  let fixture: ComponentFixture<MyComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ MyComponent ],
      providers: [ MyService ]
    }).compileComponents();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Covering topics like:

- RxJS and Observables

- Angular Material

- Custom Directives

- Content Projection

- ViewChild and ViewChildren

- HostListener and HostBinding

- Dynamic Components

- Renderer2 vs ElementRef

- Angular Elements

- SSR with Angular Universal

- PWA implementation

- Micro-frontends

- Angular Architecture patterns
- Performance optimization
- Security best practices
- State management patterns
- Advanced routing
- Form validation strategies
- HTTP client features
- Dependency injection scenarios
- Component communication patterns
- Change detection strategies
- Angular CLI features
- Build optimization
- Testing strategies
- Error handling patterns
- Authentication implementations
- Authorization scenarios
- Custom pipes
- Directives advanced usage
- Component lifecycle scenarios
- Module organization
- Lazy loading implementations
- Route guards scenarios

- HTTP interceptors use cases
- Observable patterns
- Angular security
- Performance monitoring
- Build and deployment
- Version migration
- Best practices
- Common pitfalls
- Debugging techniques
- Tool integration
- CI/CD implementation