Angular Interview Questions and Answers

June 27, 2025

Contents

1 Core Angular Concepts

1.1 What are Components in Angular?

Components are the fundamental building blocks of Angular applications, controlling a portion of the UI called a view. Each component consists of a TypeScript class for logic, an HTML template, optional CSS styles, and a `@Component` decorator.

Example:

```
@Component({
  selector: 'app-employee',
  templateUrl: './employee.component.html',
  styleUrls: ['./employee.component.css']
})
export class EmployeeComponent {
  name = 'John Doe';
}
```

1.2 What is the difference between ngOnInit() and constructor()?

The `constructor()` is a TypeScript feature used for dependency injection and initializing class members. The `ngOnInit()` lifecycle hook is called after Angular initializes the component, ideal for fetching data or setting up the component after inputs are bound.

Example:

```
constructor(private empService: EmployeeService) {}

ngOnInit() {
  this.empService.getAll().subscribe(data => this.employees = data);
}
```

1.3 What is a Service in Angular and why use it?

Services encapsulate shared logic, such as data fetching, authentication, or logging, promoting code reusability and separation of concerns.

Example:

```
@Injectable({
  providedIn: 'root'
})
export class EmployeeService {
  constructor(private http: HttpClient) {}
  getAll() {
    return this.http.get<Employee[]>('api/employees');
  }
}
```

1.4 What are Angular Directives?

Directives are instructions that modify the DOM. They include:

- **Component Directives**: Have templates (e.g., `app-employee`).
- **Structural Directives**: Modify DOM layout (e.g., `*ngIf`, `*ngFor`).
- **Attribute Directives**: Alter appearance or behavior (e.g., `[ngClass]`, `[ngStyle]`).

Example:

```
<div *ngIf="isLoggedIn">Welcome</div>
```

1.5 Explain Angular Routing

Routing enables navigation between components in a Single Page Application (SPA). Routes are defined in a routing module, and `routerLink` is used for navigation.

Example:

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'employee', component: EmployeeComponent },
  { path: '', redirectTo: 'home', pathMatch: 'full' }
];
```

```
<a routerLink="/employee">Employee Page</a>
```

1.6 What is Two-way Data Binding?

Two-way data binding synchronizes the model and view using `[(ngModel)]`. Changes in the UI update the model, and vice versa.

Example:

```
<input [(ngModel)]="name">
<p>Hello {{ name }}</p>
```

1.7 What is the purpose of ngFor and ngIf?

`*ngFor` loops through data to render lists, while `*ngIf` conditionally renders elements based on a boolean expression.

Example:

```
<ul>
  <li *ngFor="let emp of employees">{{ emp.name }}</li>
</ul>
<div *ngIf="employees.length == 0">No Employees</div>
```

1.8 What is Dependency Injection (DI) in Angular?

Dependency Injection is a design pattern where Angular automatically provides service instances to components or other services, promoting loose coupling, reusability, and testability.

Example:

```
constructor(private empService: EmployeeService) {}
```

1.9 What are Observables and how are they used in Angular?

Observables, part of RxJS, handle asynchronous operations like HTTP requests, form changes, and route events.

Example:

```
this.empService.getAll().subscribe(data => {
  this.employees = data;
});
```

1.10 How does Angular handle Forms?

Angular supports:

- **Template-driven Forms**: Simple, using `ngModel`.
- **Reactive Forms**: Scalable, defined in TypeScript with `FormGroup` and `FormControl`.

Template-driven Example:

```
<form #form="ngForm" (ngSubmit)="submit(form)">
  <input name="email" ngModel required>
</form>
```

Reactive Example:

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

1.11 What is Lazy Loading in Angular?

Lazy loading loads modules only when needed, improving performance by reducing initial bundle size.

Example:

```
{ path: 'admin', loadChildren: () => import('./admin/admin.module').
   then(m => m.AdminModule) }
```

1.12 What is a Module in Angular?

Modules group related components, services, pipes, and directives, organizing the application.

Example:

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

1.13 What are Pipes in Angular?

Pipes transform data in templates. Angular provides built-in pipes (e.g., `date`, `uppercase`), and custom pipes can be created.

Example:

```
<p>{{ date | date:'short' }}</p>
<p>{{ name | uppercase }}</p>
```

Custom Pipe:

```
@Pipe({name: 'addTitle'})
export class AddTitlePipe implements PipeTransform {
  transform(value: string): string {
    return 'Mr. ' + value;
  }
}
```

1.14 What is the purpose of an Interceptor in Angular?

Interceptors modify HTTP requests or responses globally, handling tasks like authentication, error handling, or logging.

Authentication Example:

```
intercept(req: HttpRequest<any>, next: HttpHandler): Observable<
    HttpEvent<any>> {
  const token = localStorage.getItem('token');
  const cloned = req.clone({
    setHeaders: { Authorization: `Bearer ${token}` }
  });
  return next.handle(cloned);
}
```

Error Handling Example:

```
intercept(req: HttpRequest<any>, next: HttpHandler): Observable<
    HttpEvent<any>> {
  return next.handle(req).pipe(
    catchError(err => {
```

```
      if (err.status === 401) {
        this.router.navigate(['/login']);
      }
      return throwError(() => err);
    })
  );
}
```

1.15 Explain Lifecycle Hooks in Angular

Lifecycle hooks allow you to tap into a component's lifecycle:

- `ngOnInit`: Initialization logic.
- `ngOnDestroy`: Cleanup before destruction.
- `ngOnChanges`: Respond to input changes.
- `ngAfterViewInit`: View-related logic after view initialization.

1.16 How do you share data between sibling components?

Use a shared service with RxJS `Subject` or `BehaviorSubject` to communicate data between components.

Example:

```
@Injectable({ providedIn: 'root' })
export class DataService {
  private messageSource = new BehaviorSubject<string>('default');
  currentMessage = this.messageSource.asObservable();

  changeMessage(msg: string) {
    this.messageSource.next(msg);
  }
}
```

1.17 Difference between Reactive Forms and Template-Driven Forms?

- **Template-Driven**: Uses `ngModel`, defined in templates, less scalable.
- **Reactive**: Uses `FormControl` and `FormGroup`, defined in TypeScript, scalable and testable.

Reactive Example:

```
this.form = new FormGroup({
  name: new FormControl('', Validators.required)
});
```

2 RxJS

2.1 What is RxJS, and how is it used in Angular?

RxJS is a library for reactive programming using Observables, enabling asynchronous operations like HTTP requests, form changes, and route events.

Example:

```
this.empService.getAll().subscribe(data => {
  this.employees = data);
});
```

2.2 Explain key RxJS operators and their use cases

- `map`: Transforms values.
- `filter`: Filters out unwanted values.
- `mergeMap`: Flattens and runs observables concurrently.
- `switchMap`: Cancels previous observables when a new one starts.
- `debounceTime`: Delays emission.
- `catchError`: Handles errors.

2.3 Difference between switchMap() and mergeMap()?

`switchMap` cancels the previous observable when a new one starts, suitable for searches. `mergeMap` runs all observables concurrently, ideal for parallel API calls.

2.4 Write a function that performs a search with debounceTime, distinctUntilChanged, and switchMap

This setup debounces input, ensures unique search terms, and cancels previous requests.

Example:

```
searchTerms = new Subject<string>();

ngOnInit() {
  this.searchTerms.pipe(
    debounceTime(300),
    distinctUntilChanged(),
    switchMap(term => this.employeeService.search(term))
  ).subscribe(data => this.results = data);
}

search(term: string) {
  this.searchTerms.next(term);
}
```

2.5 How would you cancel an ongoing HTTP request when a new one is triggered?

Use `switchMap` to cancel the previous request when a new one is triggered.

Example:

```
searchTerms.pipe(
  switchMap(term => this.http.get(`/api/search?q=${term}`))
)
```

3 NgRx

3.1 What is NgRx and why is it used?

NgRx is a Redux-inspired library for managing state in Angular, providing a single global state, actions, reducers, and effects. It's ideal for complex apps with shared state.

3.2 Explain core building blocks of NgRx

- **Actions**: Events describing state changes.
- **Reducers**: Pure functions handling state transitions.
- **Selectors**: Extract state slices.
- **Effects**: Handle side effects (e.g., API calls).

3.3 Example Flow of NgRx

Action:

```
createAction('[Employee Page] Load Employees')
```

Effect:

```
loadEmployees$ = createEffect(() =>
  this.actions$.pipe(
    ofType(loadEmployees),
    switchMap(() => this.empService.getAll().pipe(
      map(data => loadEmployeesSuccess({ employees: data }))
    ))
  )
);
```

Reducer:

```
on(loadEmployeesSuccess, (state, { employees }) => ({
  ...state, employees
}))
```

3.4 When would you prefer NgRx over a simple service with BehaviorSubject?

Use NgRx for complex state, shared state across components, or features like undo/redo. Use `BehaviorSubject` for simple, local state communication.

4 Testing in Angular

4.1 How do you write unit tests for a component?

Use `TestBed` to configure a testing module and Jasmine/Karma for assertions.

Example:

```
beforeEach(() => {
  TestBed.configureTestingModule({
    declarations: [EmployeeComponent],
    providers: [EmployeeService],
  });
});

it('should create the component', () => {
  const fixture = TestBed.createComponent(EmployeeComponent);
  expect(fixture.componentInstance).toBeTruthy();
});
```

4.2 What is a spy and how do you mock a service?

A spy mocks service methods, controlling their return values for testing.

Example:

```
const empService = jasmine.createSpyObj('EmployeeService', ['getAll
    ']);
empService.getAll.and.returnValue(of([{ id: 1, name: 'Test' }]));
```

5 Real-Time Project-Based Questions

5.1 How do you structure a real-time Angular project?

A typical Angular project structure organizes code for scalability.

```
/src
  /app
    /core           (singleton services, interceptors)
    /shared         (reusable components, pipes, directives)
    /features       (modules like employee, auth)
    /auth
    /employee
  /assets
  environments.ts
```

5.2 How would you implement dynamic form fields for adding/removing subjects in a student registration form?

Use `FormArray` in Reactive Forms to dynamically manage form fields.

Example:

```
this.form = this.fb.group({
  name: [''],
  subjects: this.fb.array([this.fb.control('')])
});

get subjects() {
  return this.form.get('subjects') as FormArray;
}

addSubject() {
  this.subjects.push(this.fb.control(''));
}

removeSubject(i: number) {
  this.subjects.removeAt(i);
}
```

5.3 How would you optimize Angular app performance for large tables/datasets?

Strategies include:

- Use `OnPush` change detection.
- Use `trackBy` with `*ngFor`.
- Implement virtual scrolling with `@angular/cdk`.
- Use server-side pagination.

Example:

```
<tr *ngFor="let row of data; trackBy: trackByFn">...</tr>
```

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

6 Angular + .NET Web API Integration

6.1 How do you call a .NET API from Angular?

Use `HttpClient` to make HTTP requests to a .NET Web API.

Example:

```
getStudents() {
  return this.http.get<Student[]>('https://localhost:5001/api/
     students');
}
```

6.2 How would you structure a login function in Angular that sends credentials to a .NET Web API and stores JWT token?

Send credentials via `HttpClient` and store the JWT token in local storage.

Example:

```
login(formData: any) {
  return this.http.post('https://api/login', formData).subscribe((
     res: any) => {
    localStorage.setItem('token', res.token);
  });
}
```

6.3 How do you secure Angular routes based on roles using JWT?

Use a route guard to decode the JWT token and check roles.

Example:

```
canActivate(): boolean {
  const token = localStorage.getItem('token');
  const role = decodeToken(token).role;
  return role === 'Admin';
}
```

```
{ path: 'admin', component: AdminComponent, canActivate: [RoleGuard]
    }
```