Angular: In-Depth Notes

1. Introduction to Angular

- **Angular**: A TypeScript-based open-source front-end web application framework by Google.
  - Allows building dynamic single-page applications (SPA).
  - Component-based architecture ensures modularity and reusability.
  - Features include two-way data binding, dependency injection (DI), and rich tooling.

---

2. Angular CLI

- **Purpose**: Command-line interface for Angular projects.
- **Common Commands**:
  - `ng new <project-name>`: Create a new project.
  - `ng serve`: Serve the application locally.
  - `ng generate`: Create components, services, directives, etc.
  - `ng build`: Build the project for production.
  - `ng test`: Run unit tests.
  - `ng lint`: Lint the project.

---

3. Angular Modules

- **AppModule**: The root module bootstrapped at runtime.
- **Feature Modules**: Encapsulate features into distinct modules for better organization and lazy loading.
- **Core Module**: For singleton services and app-wide providers.
- **Shared Module**: Contains reusable components, directives, and pipes.
- **Lazy Loading**:
  - Load modules on demand to optimize performance.
  - Configured via `loadChildren` in the route definition.

---

4. Components

- **Definition**: Building blocks of Angular applications.
- **Structure**:
  - **Template**: HTML structure of the component.
  - **Style**: CSS/SCSS specific to the component.
  - **Class**: Encapsulates logic and data.
- **Lifecycle Hooks**:
  - `ngOnInit`: Called after the component initializes.
  - `ngOnChanges`: Called when input-bound properties change.
  - `ngOnDestroy`: Cleanup before the component is destroyed.
  - Other hooks: `ngDoCheck`, `ngAfterViewInit`, `ngAfterContentInit`.

---

5. Directives

- **Structural Directives**: Modify the DOM structure.
  - Examples: `*ngIf`, `*ngFor`, `*ngSwitchCase`.
- **Attribute Directives**: Modify the behavior or appearance of an element.
  - Examples: `ngClass`, `ngStyle`, `ngModel`.
- **Custom Directives**:
  - Extend Angular's behavior using `@Directive()`.

---

6. Services and Dependency Injection

- **Services**: Centralized logic shared across components.
  - Created with `ng generate service <name>`.
- **Dependency Injection (DI)**:
  - Mechanism to provide instances of services where needed.
  - Types of Providers:
    - `providedIn: 'root'`: Available globally.
    - Component-level providers: Scoped to the component and its children.
  - **Singleton Services**: Single instance shared across the app.
- **Hierarchical Injectors**:
  - DI tree determines the lifetime and scope of service instances.

---

7. Routing and Navigation

- **RouterModule**:
  - Enables navigation between views.
  - Define routes in `app-routing.module.ts`.

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
];
```

- **Lazy Loading Routes**:

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

- **Route Guards**:
  - `CanActivate`: Controls route access.
  - `CanDeactivate`: Controls leaving a route.
  - `Resolve`: Pre-fetch data before navigating.
- **Query and Route Parameters**:
  - Route parameters: `/user/:id`.

- Query parameters: `/user?id=123`.

---

8. Observables and RxJS

- **Observables**:
  - Streams of asynchronous data.
  - Used for HTTP requests, forms, router events.
- **RxJS Operators**:
  - **Transformation**: `map`, `switchMap`.
  - **Filtering**: `filter`.
  - **Error Handling**: `catchError`.
  - **Timing**: `debounceTime`.
- **Subjects**:
  - **Subject**: Multicast observable.
  - **BehaviorSubject**: Emits the last value on subscription.

---

9. Forms

- **Template-Driven Forms**:
  - Bind to HTML with `[(ngModel)]`.
  - Simpler syntax, suitable for small forms.
- **Reactive Forms**:
  - Programmatic approach using `FormControl`, `FormGroup`.
  - Better suited for complex forms.
- **Validation**:
  - Built-in validators: `required`, `minlength`, etc.
  - Custom validators: Created using functions.
  - Example:

```
export const customValidator = (control: AbstractControl): ValidationErrors
| null => {
  return control.value ? null : { required: true };
};
```

---

10. HTTP Client and APIs

- **HttpClient**:
  - Methods: `get`, `post`, `put`, `delete`.
  - Returns Observables.
- **HttpInterceptor**:
  - Intercepts HTTP requests/responses.
  - Use cases: Add headers, log requests, handle errors.

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

- **CORS Issues**:
  - Occurs when the browser blocks requests to different origins.
  - Resolved by enabling CORS on the server.

---

11. Pipes

- **Built-in Pipes**:
  - Examples: `date`, `currency`, `json`, `uppercase`, `lowercase`.
- **Custom Pipes**:

```
@Pipe({ name: 'capitalize' })
export class CapitalizePipe implements PipeTransform {
  transform(value: string): string {
    return value.charAt(0).toUpperCase() + value.slice(1);
  }
}
```

- **Pure vs Impure Pipes**:
  - **Pure Pipes**: Recalculate only when input changes.
  - **Impure Pipes**: Recalculate on every change detection cycle.

---

12. State Management

- **NgRx**:
  - State management library following Redux principles.
  - **Store**: Centralized state.
  - **Actions**: Events describing changes.
  - **Reducers**: Handle state transitions.
  - **Effects**: Handle side effects (e.g., API calls).

```
this.store.dispatch(loadUsers());
this.store.select(selectUsers).subscribe(users => console.log(users));
```

---

13. Testing in Angular

- **Unit Testing**:
  - Tools: Jasmine (test framework), Karma (test runner).
  - Use `TestBed` for setting up tests.
  - Example:

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

- **Service Testing**:
  - Use `HttpClientTestingModule` to mock HTTP calls.
- **End-to-End Testing**:
  - Tools: Protractor (deprecated), Cypress.

---

14. Miscellaneous Concepts

- **View Encapsulation**:
  - Modes: `Emulated`, `Shadow DOM`, `None`.
- **ViewChild**:
  - Access child components or DOM elements.

```
@ViewChild('myInput') inputRef: ElementRef;
```