# Angular Best Practices

## 1. Code Architecture for Large-Scale Apps

### a) Component-Based Architecture

Break the UI into reusable components following the Single Responsibility Principle (SRP).

### b) Use Services for Business Logic

Use services to manage data operations instead of handling them directly in components.

### c) State Management (NgRx or Services)

Use NgRx (Redux pattern) or services with BehaviorSubject for shared state management.

## 2. Modular and Scalable Project Structure

### a) Feature-Based Modules

Instead of a single large module, create feature modules for better maintainability.

### b) Core and Shared Modules

Core module for singleton services, shared module for reusable components.

## 3. Security Best Practices in Angular Applications

### a) Avoid Direct DOM Manipulation

Use Angular bindings instead of document.querySelector or innerHTML.

### b) Sanitize User Inputs (XSS Prevention)

### c) Use Route Guards for Authorization

### d) Secure API Calls with Interceptors

### e) Disable Debug Data in Production

@Component({  
 selector: 'app-user-profile',  
 templateUrl: './user-profile.component.html',  
 styleUrls: ['./user-profile.component.css']  
})  
export class UserProfileComponent {  
 @Input() user: User;  
}

@Injectable({ providedIn: 'root' })  
export class UserService {  
 constructor(private http: HttpClient) {}  
  
 getUserData(): Observable<User> {  
 return this.http.get<User>('/api/user');  
 }  
}

export class UserService {  
 private userSource = new BehaviorSubject<User | null>(null);  
 user$ = this.userSource.asObservable();  
  
 setUser(user: User) {  
 this.userSource.next(user);  
 }  
}

/src  
 ├── app/  
 │ ├── core/ (Singleton services, authentication, etc.)  
 │ ├── shared/ (Reusable components, directives, pipes)  
 │ ├── features/  
 │ │ ├── user/  
 │ │ │ ├── user.module.ts  
 │ │ │ ├── user.component.ts  
 │ │ │ ├── user.service.ts  
 │ │ │ ├── user.routing.ts  
 │ │ ├── orders/  
 │ │ │ ├── orders.module.ts  
 │ │ │ ├── orders.component.ts  
 │ │ │ ├── orders.service.ts  
 │ │ │ ├── orders.routing.ts  
 │ ├── app.module.ts  
 │ ├── app-routing.module.ts  
 │ ├── main.ts

@NgModule({  
 providers: [AuthService, LoggerService]  
})  
export class CoreModule { }

constructor(private sanitizer: DomSanitizer) {}  
  
safeHtml(content: string) {  
 return this.sanitizer.bypassSecurityTrustHtml(content);  
}

@Injectable({ providedIn: 'root' })  
export class AuthGuard implements CanActivate {  
 constructor(private authService: AuthService, private router: Router) {}  
  
 canActivate(): boolean {  
 if (this.authService.isAuthenticated()) {  
 return true;  
 }  
 this.router.navigate(['/login']);  
 return false;  
 }  
}

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

enableProdMode();