Angular 15

eProtective
FRONTEND ARCHITECTURE

Training of Trainers

2-Day Intensive Workshop

DATE
20 - 21January 2026

DURATION
2 Days

TARGET AUDIENCE
Devs, Leads, Trainers

PREREQUISITES
JS / TS Knowledge

Document Version 1.0

Confidential & Proprietary

Training Overview

Goals & Expected Results

DAY 1-2 FOCUS

Learning Objectives

- Set up complete Angular 15 development environment
- Deep dive into eProtective project architecture
- Build reusable components, services, and modules
- Implement secure authentication with Keycloak
- Master HTTP requests and error handling strategies
- Debug efficiently using browser & Angular tools
- Mentor junior developers on best practices

Key Outcomes

Fully configured development workstation

Clear comprehension of project structure & patterns

Secure application with auth-protected routes

Production-ready optimized build configuration

Troubleshooting skills for common issues

Training resources to onboard new team members

Emphasis on practical, hands-on implementation

Agenda Overview

2-Day Intensive Training Schedule

DAY 1 — Foundation & Architecture

8 Hours • 4 Sessions

09:00–10:30	**Session 1: Welcome & Setup**	Env Setup, CLI, VS Code, Project Overview
	Break (15m)	
10:45–12:30	**Session 2: Angular Fundamentals**	Components, Templates, Binding, Directives
	Lunch (60m)	
13:30–15:30	**Session 3: Services & RxJS**	Dependency Injection, Observables, HTTP
	Break (15m)	
15:45–17:30	**Session 4: Routing & Modules**	Lazy Loading, Guards, Feature Modules

DAY 2 — Advanced Topics & Deployment

8 Hours • 4 Sessions + Workshop

09:00–11:00	**Session 5: Auth & Keycloak**	JWT, Keycloak Integration, Auth Guards
	Break (15m)	
11:15–13:00	**Session 6: Forms & Validation**	Reactive Forms, Custom Validators, Async
	Lunch (60m)	
14:00–15:00	**Session 7: Debugging & Testing**	DevTools, Jasmine, Unit Testing
15:15–17:30	**Session 8: Build & Deploy**	Optimization, Best Practices, CI/CD

Interactive Workshop & Q&A Integrated throughout sessions

Materials & Setup

Prerequisites & Environment Checklist

Download from Repository

Training Materials

- HANDS-ON-GUIDE.md (Project documentation)
- SETUP-AND-INSTALLATION.md (Setup guide)
- TROUBLESHOOTING-GUIDE.md (Issue resolution)
- CODE-EXAMPLES.md (Sample implementations)
- LIBRARY-REFERENCE.md (Library configs)

Tools & Equipment

- Laptop (Min 8GB RAM, 2GB disk space)
- Code Editor (VS Code recommended)
- Stable Internet Connection
- Git SCM Installed & Configured

Pre-Training Checklist

- Node.js v16+ installed
- NPM v8+ installed
- Angular CLI v15+ installed globally
- Project repository cloned locally
- Keycloak server running (optional for Day 1)
- Sample data populated in database

Recommended Specs

MINIMUM

- 2-core CPU
- 8GB RAM
- 5GB SSD

RECOMMENDED

- 4-core CPU
- 16GB RAM
- 10GB SSD

Day 1 • Session 1: Environment Setup

System Verification & Installation

9:00 AM - 10:30 AM

Key Steps

- Verify installations: node -v, npm -v, ng version
- Install Angular CLI globally: npm i -g @angular/cli@15
- Install VS Code Extensions:
  - Angular Language Service, ESLint, Prettier
- Clone repository and open in VS Code

Expected Output

- ✓ Node v16.x or higher
- ✓ npm v8.x or higher
- ✓ Angular CLI 15.x
- ✓ 3+ Essential VS Code extensions installed

bash — user@dev-machine

```
# 1. Check current versions
$ node -v && npm -v
v16.18.0
8.19.2

# 2. Install Angular CLI 15
$ npm install -g @angular/cli@15
changed 196 packages in 4s
added 1 package, and audited 197 packages in 5s

# 3. Verify Angular installation
$ ng version
Angular CLI: 15.2.0
Node: 16.18.0
OS: darwin x64

# 4. Quick Start Demo App
$ npx @angular/cli@15 new demo-app --routing --style=scss
$ cd demo-app && ng serve
✓ Compiled successfully.
```

Components & Templates

Angular Fundamentals • Session 2

10:45 AM - 12:30 PM

Component Anatomy

- Decorator: @Component defines metadata (selector, template, template, styles)
- Template: The HTML view (inline or external file)
- Class: TypeScript logic, properties & methods

Data Binding Patterns

- Interpolation: {{ value }} (One-way: TS → HTML)
- Property: [prop]="val" (One-way: TS → HTML)
- Event: (event)="fn()" (One-way: HTML → TS)
- Two-Way: [(ngModel)]="val" (Synchronized)

Architecture Tip

Keep components "dumb" (presentational) when possible. Move complex business logic and API calls to Services.

TS user-profile.component.ts

```ts
import {Component ,OnInit }from '@angular/core' ;

@Component ({
selector :'app-user-profile' ,
template :`
<divclass ="card" >
<!-- Interpolation -->
<h1>{{ title }} </h1>

<!-- Two-way Binding -->
<input [(ngModel)] ="username" />

<!-- Event Binding -->
<button (click) ="save()" >Update </button>
</div>
`,
styleUrls: ['./user-profile.component.scss' ]
})
export class UserProfileComponent implements OnInit {
title ='User Profile' ;
username ='';
```

Directives & Binding Patterns

Angular Fundamentals: Templates & Data Flow

10:45 AM - 12:30 PM

Structural Directives

- Modify DOM layout by adding/removing elements
- *ngIf: Conditionally includes a template based on expression
- *ngFor: Repeats a node for each item in a collection
  - Use trackBy for performance in large lists

Binding Patterns

- Property: [property]="value" (Data source to view)
- Class: [class.name] or [ngClass] object
- Style: [style.prop] or [ngStyle] object

Exercise: Build a List

- Create item list with add/remove buttons
- Use *ngIf for "No items" message & trackBy for loop

demo.component.html

```
<!-- Structural Directives -->
<div *ngIf="isVisible" >
Visible Content
</div>
<div *ngFor="let item of items; trackBy: trackByFn"  >
{{ item.name }}
</div>
<!-- Property Binding -->
<img [src] ="imageUrl" />
<input [disabled] ="isDisabled" />
<!-- Class Binding -->
<div [class.active] ="isActive" >Active State </div>
<div [ngClass] ="{'active': isActive, 'disabled': isDisabled}"  >
Conditional Classes
</div>
<!-- Style Binding -->
```

Day 1 • Session 3: Services, DI & RxJS

Dependency Injection & Reactive Patterns

1:30 PM - 3:30 PM

Dependency Injection (DI)

- Singleton services: providedIn: 'root'
- Constructor Injection pattern:

  constructor(private svc: MyService)
- Promotes reusability and testability

RxJS & Observables

- Observables: Stream of values over time (vs Promise)
- Async Pipe: | async handles subscribe/unsubscribe
- Operators: Transform data flow
  - map, filter, tap, takeUntil
- Avoid memory leaks with proper cleanup

src/app/core/user.service.ts & user.component.ts

```
// 1. Define Singleton Service
@Injectable ({ providedIn: 'root' })
export class UserService {
constructor (private http: HttpClient ) {}

getUsers (): Observable <User []> {
return this .http. get<User []>('/api/users' );
}
}

// 2. Component Usage with RxJS
@Component ({...})
export class UserListComponent implements OnInit {
users$: Observable <User []>;

constructor (private userService: UserService ) {}

ngOnInit () {
// Reactive approach: Pipe & Transform
this .users$ = this .userService. getUsers ().pipe (
map(users => users. filter(u => u.isActive)),
tap(data => console. log('Active users:' , data))
```

HTTP, Interceptors & Error Handling

Managing API Communications & Resilience

Day 1 • Session 3

HttpClient Pattern

- Observable-based API calls (cold observables)
- Strong typing via Generics: .get<User[]>()
- Automatic JSON parsing and request transformation
- Easy testing with HttpClientTestingModule

Resilience & Interceptors

- Interceptors: Centralized Auth, Logging, Headers
- Error Handling: Use RxJS catchError operator
- Best Practice: Use services for data, components for view
- Global error handler service for consistent UX

course.service.ts — TypeScript

src/app/services/course.service.ts

```typescript
getCourses (): Observable <Course[] > {
return this .http. get<Course[] >('/api/courses' ).pipe (
retry (2), // Retry failed requests twice
catchError (this .handleError)
);
}

private handleError (err :HttpErrorResponse ) {
console .error ('API Error:' ,err);
return throwError (() => new Error ('Fetch failed' ));
}
```

src/app/core/auth.interceptor.ts

```typescript
intercept (req :HttpRequest <any>, next :HttpHandler ) {
const token =this .auth. getToken ();
// Clone request to add auth header
const authReq =req .clone ({
setHeaders: { Authorization: `Bearer ${token}` }
});
return next .handle (authReq );
}
```

Day 1 • Session 4: Routing & Modules

Lazy Loading, Guards & Navigation

3:45 PM - 5:30 PM

Router Configuration

- Path Setup: Define paths and redirects (pathMatch: 'full')
- Lazy Loading: Use loadChildren with dynamic imports to reduce reduce initial bundle size
- Feature Modules: Encapsulate related components and routing in routing in separate modules

Guards & Navigation

- Route Guards: Implement CanActivate to protect sensitive routes from unauthorized access
- Navigation: Use router.navigate() for programmatic control
- Parameters: Access dynamic data via ActivatedRoute

typescript — app-routing.module.ts & auth.guard.ts

```typescript
// 1. Lazy Loading Configuration
const routes :Routes = [
  {path :'',redirectTo :'/home' ,pathMatch :'full' },
  {
    path :'courses' ,
    loadChildren : () =>import ('./courses/courses.module'  )
      .then (m=>m.CoursesModule ),
    canActivate : [AuthGuard ]// Protect this route
  }
];

// 2. Auth Guard Implementation
@Injectable ({providedIn :'root' })
export class AuthGuard implements CanActivate {
  constructor (private auth :AuthService ,private router :Router ) {}

  canActivate ():boolean {
    if (this .auth. isLoggedIn ()) {
      return true ;
    }
    this .router. navigate (['/login' ]);
    return false ;
  }
}
```

Day 2 • Authentication Basics

JWT Authentication Flow & Concepts

Session 5

AuthN vs AuthZ

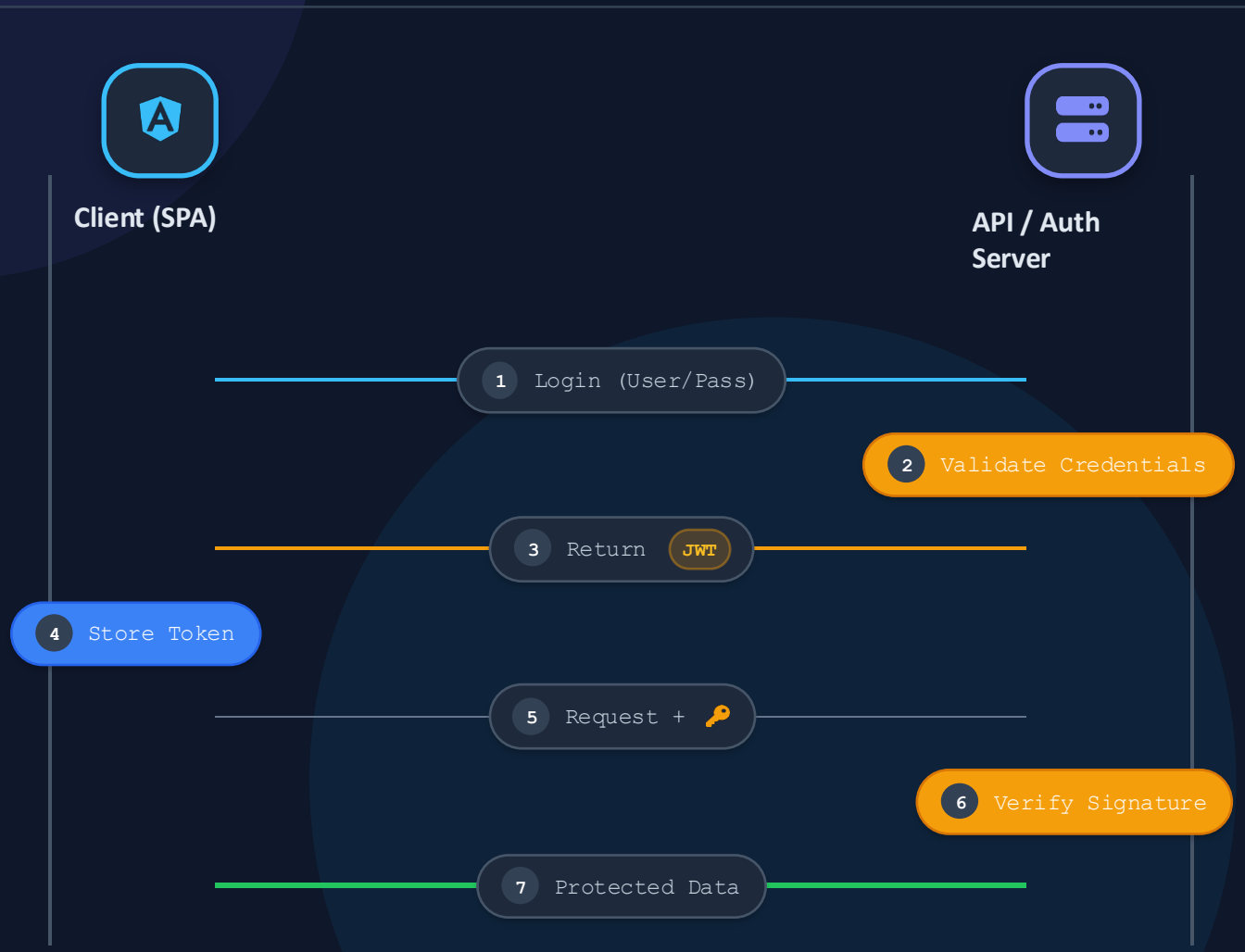
Authentication (Who): Verifying the user's identity identity (e.g., Login).

Authorization (What): Verifying access rights to rights to specific resources (e.g., Admin Panel). Panel).

Security Best Practices

- Use HTTPS everywhere
- Short-lived access tokens
- Implement Refresh Token rotation
- Protect routes with Guards

> ⚠ Prefer **httpOnly cookies** over localStorage for token storage to prevent XSS.



Day 2 • Keycloak Integration

Secure Authentication & Setup

9:00 AM - 11:00 AM

Day 2 • Session 6: Forms & Validation

Reactive Forms, Validators & Error Handling

11:15 AM - 1:00 PM

Reactive Architecture

- FormGroup: Tracks value/status of a group of controls
- FormControl: Tracks value/status of individual inputs
- FormBuilder: Syntactic sugar for creating form models
- Immutable Data: Updates push new values to streams

Validation Strategy

- Sync Validators: required, minLength, email
- Custom Validators: Cross-field checks (e.g., passwords)
- Async Validators: Server checks (e.g., username taken)
- States: valid, invalid, dirty, touched

registration.component.ts

```
export class RegistrationComponent {
form!:FormGroup;
constructor(private fb:FormBuilder) {
// 1. Initialize Form Model
this.form=this.fb.group({
username: ['', [Validators.required,Validators.minLength(3)]],
email: ['', [Validators.required,Validators.email]],
password: ['', [Validators.required,Validators.minLength(8)]],
confirm: ['',Validators.required]
}, { validators:this.passwordMatch });
}
// 2. Custom Validator Logic
passwordMatch(group:AbstractControl):ValidationErrors|null{
const pass=group.get('password')?.value;
const confirm=group.get('confirm')?.value;
return pass===confirm?null: {mismatch:true};
}
// 3. Form Submission
onSubmit() {
if(this.form.valid) {
console.log(this.form.value);
}
```

Day 2 • Session 7: Debugging & Testing

Troubleshooting Tools & Unit Testing Strategies

2:00 PM - 3:00 PM

Debugging Tools

- Browser DevTools (F12): Breakpoints, Watch, Step-through
- Network Tab: Inspect API headers, payloads & status codes
- Angular DevTools: Visual component tree & profiler
- Console: Object inspection & error tracking

Testing Patterns

- Framework: Jasmine (Syntax) & Karma (Runner)
- HttpTestingController for mocking API calls
- spyOn() to mock service methods & dependencies
- Trigger lifecycle events with fixture.detectChanges()

user.service.spec.ts

```
describe ('UserService' , () => {
  let service: UserService ;
  let httpMock: HttpTestingController  ;

  beforeEach (() => {
    TestBed .configureTestingModule  ({
      imports: [ HttpClientTestingModule  ],
      providers: [ UserService ]
    });
    service = TestBed .inject (UserService );
    httpMock = TestBed .inject (HttpTestingController  );
  });

  it ('should fetch users via GET'  , () => {
    const mockUsers = [{ id: 1, name:  'John' }];

    // 1. Call the service method
    service. getUsers ().subscribe (users => {
      expect (users.length). toBe (1);
      expect (users). toEqual (mockUsers);
    });

    // 2. Expect and handle the HTTP request
    const req = httpMock. expectOne ('/api/users' );
```

Build, Optimize & Deploy

Production Readiness & Server Configuration

3:15 PM - 5:30 PM

Production Build

- Command: ng build --configuration=production
- AOT Compilation: Pre-compiles templates for faster rendering
- Tree Shaking: Removes unused code/modules
- Analysis: Use webpack-bundle-analyzer to find large chunks

Deployment Strategy

- Test locally: npx http-server dist/
- Verify API connections in production mode
- Configure web server (Nginx/Apache) to handle SPA routing (redirect 404 to index.html)

bash — ci-runner

```
# 1. Standard Production Build
$ ng build --configuration=production

  Initial Chunk Files | Names | Raw Size
  main.7a2b...js | main | 245.32 kB
  styles.3e1...css | styles | 54.12 kB
  polyfills.9c...js | poly...| 33.02 kB

  ✓ Build at dist/app/ completed.

# 2. Optimized Build with Flags
$ ng build --configuration=production \
> --output-hashing=all --source-map=false --aot=true

# 3. Analyze Bundle Size
$ ng build --stats-json && webpack-bundle-analyzer dist/stats.json

# 4. Nginx Configuration for SPA (nginx.conf)
```

```
server {
listen 80;
server_name yourdomain.com;
root /var/www/app;
index index.html;
```

Best Practices & Guidelines

Standards for Scalable & Secure Applications

Follow Security Standards

Code Quality

- ✓ Use ChangeDetectionStrategy.OnPush for components
- ✓ Always unsubscribe: takeUntil(destroy$) or Async Pipe
- ✓ Use trackBy function with *ngFor loops
- ✗ Avoid logic in templates (keep them declarative)
- ✗ Never mutate state directly; use immutable patterns

Security Best Practices

- ✓ Sanitize dynamic HTML content using DomSanitizer
- ✓ Implement strictly defined Content Security Policy (CSP)
- ✗ Never store sensitive data (passwords, secrets) in localStorage
- ✓ Use HttpOnly cookies for authentication tokens
- ✓ Enforce HTTPS for all API communication

Performance Optimization

- ✓ Lazy load feature modules with loadChildren
- ✓ Use Virtual Scrolling for large lists
- ✓ Implement image optimization with loading="lazy"
- ✓ Enable Production Build optimizations (AOT, Tree Shaking)

Accessibility (a11y)

- ✓ Use semantic HTML elements (<button>, <nav>)
- ✓ Ensure sufficient color contrast ratios
- ✓ Support keyboard navigation (focus management)
- ✓ Provide aria-label for icon-only buttons

Summary & Resources

Key Takeaways, Documentation & Next Steps

COURSE COMPLETED
2 Days • 16 Hours

Key Takeaways

Core Architecture

- Multi-layer Pattern: Separation of Components, Services, and Models
- Dependency Injection: Efficient singleton services and provider scope
- Modules: Feature-based organization with lazy loading

Data & State

- RxJS: Handling async streams with Observables & Operators
- HTTP Client: Robust API integration with Interceptors
- Reactive Forms: Type-safe forms with custom validators

Security & Quality

- Keycloak: Secure authentication flows and Role-based guards
- Testing: Unit tests with Jasmine spies and HTTP mocks

Ops & Dev Workflow

- Debugging: Leveraging Browser and Angular DevTools
- Deployment: Production builds (AOT) and server config

Continuous Learning

- angular.io/guide
- github.com/eprotective
- Angular Community Forums

Post-Training Support

Refer to the project documentation for detailed guides:

- `HANDS-ON-GUIDE.md`
- `TROUBLESHOOTING.md`
- `SETUP-GUIDE.md`