

New Page

On Internal Web - Angular Application Documentation

Table of Contents

1. [Project Overview](#)
 2. [Technology Stack](#)
 3. [Project Architecture](#)
 4. [Getting Started](#)
 5. [Project Structure](#)
 6. [Key Features & Modules](#)
 7. [State Management](#)
 8. [Routing & Navigation](#)
 9. [Services & API Integration](#)
 10. [Styling & Theming](#)
 11. [Development Workflow](#)
 12. [Build & Deployment](#)
 13. [Testing](#)
 14. [Troubleshooting](#)
 15. [Best Practices](#)
 16. [Additional Resources](#)
-

Project Overview

On Internal Web is an enterprise-grade Angular web application built with Angular v15.2.10 and based on the Visual Builder framework. It serves as an internal management system with multiple modules for various business operations including:

- Employee management
- Asset tracking and management
- Ticketing system
- Leave management
- Freelance agent management
- Fleet management (OnFleet)
- CMS content management
- Branding approval workflows
- Sales operations
- Permit management

Key Characteristics

- **Framework:** Angular v15.2.10
 - **Language:** TypeScript 4.9.5
 - **State Management:** NgRx v15.4.0
 - **UI Libraries:** Ant Design (ng-zorro-antd v15) & Angular Material v15
 - **Build System:** Angular CLI v15.2.11
 - **Architecture:** Modular with lazy loading
 - **Deployment:** Cloudflare Workers
-

Technology Stack

Core Dependencies

Angular Ecosystem

```
{
  "@angular/core": "^15.2.10",
  "@angular/common": "^15.2.10",
  "@angular/router": "^15.2.10",
  "@angular/forms": "^15.2.10",
  "@angular/animations": "^15.2.10",
  "@angular/platform-browser": "^15.2.10",
```

```
"@angular/platform-browser-dynamic": "^15.2.10"  
}
```

State Management

```
{  
  "@ngrx/store": "^15.4.0",  
  "@ngrx/effects": "^15.4.0",  
  "@ngrx/router-store": "^15.4.0"  
}
```

UI Component Libraries

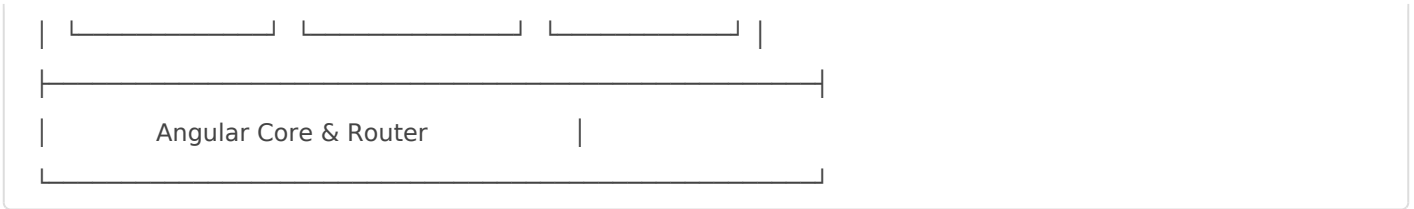
```
{  
  "@angular/material": "^15.2.9",  
  "@angular/cdk": "^15.2.9",  
  "ng-zorro-antd": "^15.1.1",  
  "@ng-bootstrap/ng-bootstrap": "^14.2.0",  
  "bootstrap": "^4.6.1"  
}
```

Data Visualization

```
{  
  "c3": "^0.7.20",  
  "chartist": "^0.11.4",  
  "angular2-chartjs": "^0.5.1",  
  "angular-calendar": "^0.29.0"  
}
```

Utilities

```
{  
  "lodash": "^4.17.21",  
  "moment": "^2.29.4",  
  "date-fns": "^2.30.0",  
  "rxjs": "^7.8.1",  
  "exceljs": "^4.3.0",  
  "pdfmake": "^0.2.7"  
}
```

Module Organization

The application follows a **feature module pattern** with lazy loading:

1. **App Module** (`app.module.ts`) - Root module
2. **Feature Modules** - Self-contained modules for each business domain
3. **Shared Module** (`shared.module.ts`) - Common components, pipes, directives
4. **Layouts Module** (`layouts.module.ts`) - Layout components
5. **Core Services** - Singleton services for API communication

Design Patterns

- **Component-Based Architecture:** Modular, reusable components
- **Lazy Loading:** Features loaded on-demand for better performance
- **State Management:** Centralized state using NgRx Store
- **Dependency Injection:** Angular's DI for service management
- **Guard-Based Authentication:** Route protection with AuthGuard
- **Reactive Programming:** RxJS for async operations

Getting Started

Prerequisites

1. **Node.js:** v16.x or v18.x (v20.x with legacy OpenSSL provider)
2. **npm:** Latest version (comes with Node.js)
3. **Angular CLI:** v15.x

Installation Steps

1. Clone the Repository

```
git clone https://github.com/ondeliveroper/on-internal-web.git
cd on-internal-web
```

2. Install Dependencies

```
npm install --legacy-peer-deps
```

“ **Note:** The `--legacy-peer-deps` flag is required due to peer dependency conflicts with some packages.

3. Start Development Server

```
npm start
```

The application will be available at `http://0.0.0.0:4201`

4. Build for Production

```
npm run build
```

The production build will be available in the `dist/build` directory.

Available Scripts

Command	Description
<code>npm start</code>	Start development server on port 4201
<code>npm run build</code>	Production build with optimizations
<code>npm test</code>	Run unit tests with Karma
<code>npm run prettier</code>	Format code with Prettier
<code>npm run tslint</code>	Run TSLint for code quality
<code>npm run hmr</code>	Start with Hot Module Replacement
<code>npm run ng-high-memory</code>	Start server with increased memory (8GB)
<code>npm run deploy</code>	Deploy to Cloudflare

Environment Configuration

The application supports multiple environments:

- **Development** (`environment.ts`) - Default development environment
- **Production** (`environment.prod.ts`) - Production configuration
- **Demo** (`environment.demo.ts`) - Demo/staging environment
- **HMR** (`environment.hmr.ts`) - Hot Module Replacement

Example environment configuration:

```
export const environment = {  
  production: false,  
  authenticated: false,  
  hmr: false,  
}
```

Project Structure

```
on-internal-web/  
├─ .cloudflare/      # Cloudflare Workers deployment config  
├─ .github/          # GitHub configurations  
├─ e2e/              # End-to-end tests  
├─ scripts/          # Build and utility scripts  
├─ src/  
│ └─ app/  
│ │ └─ @vb/          # Visual Builder components  
│ │ │ └─ components/ # VB UI components  
│ │ │ └─ css/         # Global styles  
│ │ │ └─ widgets/     # Reusable widget components  
│ │ └─ components/    # Shared application components  
│ │ └─ dialog-decision/ # Decision dialog component  
│ │ └─ layouts/       # Layout components  
│ │ │ └─ Auth/        # Authentication layout  
│ │ │ └─ Main/        # Main application layout  
│ │ │ └─ Public/      # Public pages layout  
│ │ └─ loading-dialog/ # Loading indicator  
│ │ └─ loading-modal/ # Loading modal component  
│ │ └─ locales/       # i18n translation files  
│ │ └─ models/        # TypeScript interfaces/models
```

```
| | └─ pages/      # Feature modules
| | | └─ administration/  # Admin management
| | | └─ agent-management/ # Agent operations
| | | └─ asset-management/ # Asset tracking
| | | └─ auth/           # Authentication
| | | └─ branding-approval/ # Branding workflows
| | | └─ cms/           # Content management
| | | └─ cta/           # Call-to-action management
| | | └─ dashboard/     # Dashboard views
| | | └─ freelance-agent/ # Freelance management
| | | └─ hr/            # Human resources
| | | └─ leave-management/ # Leave tracking
| | | └─ on-mitra-management/# Mitra partner management
| | | └─ onapps-management/ # OnApps management
| | | └─ onfleet/       # Fleet management
| | | └─ permit-management/ # Permit workflows
| | | └─ sales/         # Sales operations
| | | └─ table/         # Generic table views
| | | └─ ticketing/     # Ticketing system
| | | └─ user-profile/  # User profile
| | └─ services/      # Angular services
| | | └─ fakeApi/      # Mock API services
| | | └─ firebase/     # Firebase services
| | | └─ freelance-management/ # Freelance services
| | | └─ interface/    # Service interfaces
| | | └─ jwt/          # JWT authentication
| | | └─ leave-management/ # Leave services
| | | └─ menu/         # Menu services
| | | └─ onapps/       # OnApps services
| | | └─ permit-management/ # Permit services
| | └─ shared/        # Shared module
| | | └─ components/   # Shared components
| | | └─ directive/    # Custom directives
| | | └─ pipes/        # Custom pipes
| | └─ store/         # NgRx state management
| | | └─ settings/     # Settings state
| | | └─ user/         # User state
| | └─ app.component.ts # Root component
| | └─ app.module.ts   # Root module
```

```
| | └─ app-routing.module.ts # Main routing
| | └─ globals.ts          # Global variables
| | └─ shared.module.ts   # Shared module
| └─ assets/              # Static assets
| | └─ fonts/             # Custom fonts
| | └─ images/            # Image files
| | └─ locales/           # i18n JSON files
| └─ environments/       # Environment configurations
| └─ index.html           # Main HTML file
| └─ main.ts              # Application entry point
| └─ polyfills.ts         # Browser polyfills
| └─ styles.scss          # Global styles
└─ angular.json           # Angular CLI configuration
└─ package.json           # Dependencies and scripts
└─ tsconfig.json          # TypeScript configuration
└─ tslint.json            # TSLint configuration
└─ README.md              # Basic readme
└─ UPGRADE_NOTES.md       # Angular upgrade documentation
└─ LAZY_LOADING_IMPLEMENTATION.md # Lazy loading guide
└─ FLEX_LAYOUT_MIGRATION.md # Flex layout migration guide
```

Key Directories Explained

`/src/app/@vb/`

Visual Builder framework components including:

- Pre-built UI components
- Global CSS styles
- Reusable widgets
- Layout components

`/src/app/pages/`

Feature modules organized by business domain. Each directory typically contains:

- Module file (`*.module.ts`)
- Routing module (`*.routing.module.ts`)
- Components (`*.component.ts` , `*.component.html` , `*.component.scss`)
- Services (if feature-specific)

`/src/app/services/`

Singleton services for:

- API communication
- Data manipulation
- Business logic
- State management helpers

`/src/app/store/`

NgRx state management:

- Actions (`actions.ts`)
- Reducers (`reducers.ts`)
- Effects (`effects.ts`)
- Selectors (`selectors.ts`)

`/src/app/shared/`

Shared resources used across multiple features:

- Common components
 - Custom pipes
 - Custom directives
 - Utility functions
-

Key Features & Modules

1. Dashboard Module

Path: `/dashboard`

Central dashboard providing overview of:

- Key metrics and statistics
- Quick actions
- Recent activities
- System notifications

2. Human Resource (HR) Module

Path: `/human-resource`

Features:

- Employee management
- Attendance tracking
- Performance reviews
- Department organization
- Shift scheduling

3. Asset Management Module

Path: `/asset-management`

Manages company assets including:

- Asset inventory tracking
- Asset configuration
- Asset tracking and monitoring
- Maintenance records

Key Components:

- Asset inventory list
- Asset tracking interface
- Asset configuration forms

4. Leave Management Module

Path: `/leave-management`

Leave and absence management:

- Leave request submission
- Approval workflows
- Leave balance tracking
- Leave history
- Calendar integration

5. Ticketing System Module

Path: `/ticketing`

Support ticket management:

- Create and manage tickets
- Ticket assignment
- Status tracking
- Priority management
- Response templates

6. OnFleet Module

Path: `/onfleet`

Fleet and delivery management:

- Vehicle tracking
- Driver management
- Route optimization
- Delivery scheduling

7. Branding Approval Module

Path: `/approval`

Marketing material approval workflow:

- Submit branding materials
- Review and approval process
- Version control
- Comments and feedback
- Status tracking

8. Freelance Agent Management

Path: `/freelance-management`

Manage freelance workforce:

- Agent registration
- Project assignments
- Performance tracking
- Payment management

9. CMS (Content Management System)

Path: `/cms`

Content management for:

- Website content
- Blog posts
- Media library
- SEO management

10. Sales Module

Path: `/sales`

Sales operations:

- Lead management
- Sales pipeline
- Order processing
- Customer management

11. Permit Management

Path: `/permit-management`

Permit application and tracking:

- Permit requests
- Approval workflows
- Document management
- Expiry tracking

12. Administration Module

Path: `/administration`

System administration:

- User management
- Role-based access control
- System settings
- Audit logs

13. On Mitra Management

Path: `/mitra-management`

Partner management system

14. OnApps Management

Path: `/onapps-management`

OnApps platform management

15. CTA (Call-to-Action) Module

Path: `/cta`

CTA campaign management

16. User Profile

Path: `/user-profile`

User profile management:

- Personal information
 - Password change
 - Preferences
 - Activity history
-

State Management

NgRx Architecture

The application uses **NgRx** for centralized state management following Redux principles.

Store Structure

```
// Store Shape
{
  user: {
    id: string,
    authorized: boolean,
    loading: boolean,
    role: string,
    // ... user data
  },
  settings: {
    theme: string,
    locale: string,
    // ... app settings
  }
}
```

Key Files

1. `store/reducers.ts` - Root reducer combining feature reducers
2. `store/user/effects.ts` - User-related side effects
3. `store/user/actions.ts` - User action definitions
4. `store/settings/` - Settings state management

State Management Flow

Component → Dispatch Action → Effect (if async) → API Call → Reducer → Store → Component

Using NgRx in Components

Dispatching Actions

```
import { Store } from '@ngrx/store'
import * as UserActions from 'src/app/store/user/actions'

constructor(private store: Store<any>) {}

login() {
  this.store.dispatch(new UserActions.Login({
```

```
email: 'user@example.com',  
password: 'password'  
}))  
}
```

Selecting State

```
import { select } from '@ngrx/store'  
import * as Reducers from 'src/app/store/reducers'  
  
ngOnInit() {  
  this.store.pipe(select(Reducers.getUser)).subscribe(user => {  
    console.log('User state:', user)  
  })  
}
```

Effects (Side Effects)

Effects handle asynchronous operations like API calls:

```
import { createEffect, Actions, ofType } from '@ngrx/effects'  
  
login = createEffect(() =>  
  this.actions.pipe(  
    ofType(UserActions.LOGIN),  
    switchMap((action) =>  
      this.apiService.login(action.payload).pipe(  
        map((response) => new UserActions.LoginSuccess(response)),  
        catchError((error) => of(new UserActions.LoginFailure(error)))  
      )  
    )  
  )  
)
```

Routing & Navigation

Routing Structure

The application uses **lazy loading** for all feature modules to optimize initial load time.

Main Routes (`app-routing.module.ts`)

```
const routes: Routes = [  
  {  
    path: '',  
    redirectTo: 'dashboard',  
    pathMatch: 'full',  
  },  
  {  
    path: '',  
    component: LayoutMainComponent,  
    canActivate: [AuthGuard],  
    children: [  
      {  
        path: 'dashboard',  
        data: { title: 'Dashboard' },  
        loadChildren: () => import('./pages/dashboard/dashboard.module')  
          .then(m => m.DashboardModule),  
      },  
      // ... other routes  
    ],  
  },  
  {  
    path: 'auth',  
    component: LayoutAuthComponent,  
    children: [  
      {  
        path: '',  
        loadChildren: () => import('./pages/auth/auth.module')  
          .then(m => m.AuthModule),  
      },  
    ],  
  },  
]
```

Layout System

1. LayoutMainComponent

Main application layout for authenticated users:

- Top navigation bar
- Side navigation menu
- Content area
- Footer

2. LayoutAuthComponent

Authentication pages layout:

- Centered content
- No navigation
- Background styling

3. LayoutPublicComponent

Public pages layout

Route Guards

AuthGuard

Protects routes requiring authentication:

```
@Injectable({ providedIn: 'root' })
export class AuthGuard implements CanActivate {
  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    if (this.authorized) {
      return true
    }

    // Redirect to login with return URL
    this.router.navigate(['auth/login'], {
      queryParams: { returnUrl: state.url }
    })
    return false
  }
}
```

```
}  
}
```

Navigation Menu

Menu configuration is stored in service files and loaded dynamically based on user role.

Services & API Integration

Service Architecture

Services are organized by feature domain and follow singleton pattern.

Core Services

1. **ApiService** (`api.service.ts`)

Base HTTP service for API communication:

```
export class ApiService {  
  constructor(private http: HttpClient) {}  
  
  get(endpoint: string): Observable<any> {  
    return this.http.get(`${API_URL}${endpoint}`)  
  }  
  
  post(endpoint: string, data: any): Observable<any> {  
    return this.http.post(`${API_URL}${endpoint}`, data)  
  }  
}
```

2. **TicketingService** (`ticketing.service.ts`)

Handles ticketing system operations:

- Create/update/delete tickets

- Fetch ticket lists
- Ticket assignments

3. **AssetManagementService** (`asset-management.service.ts`)

Asset-related operations:

- Asset inventory management
- Asset tracking
- Asset configuration

4. **HrService** (`hr.service.ts`)

Human resource operations:

- Employee management
- Attendance tracking
- Performance reviews

5. **LeaveService** (`leave.service.ts`)

Leave management:

- Leave requests
- Approval workflows
- Leave balance calculations

6. **BrandingApprovalService** (`branding-approval.service.ts`)

Branding approval workflows

7. **FreelanceManagementService**

Freelance agent operations

HTTP Interceptors

JWT Interceptor

Automatically adds authentication tokens to requests:

```
@Injectable()
export class JwtInterceptor implements HttpInterceptor {
```

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

API Response Handling

Standard response format:

```
interface ApiResponse<T> {
  success: boolean
  data: T
  message?: string
  error?: string
}
```

Styling & Theming

CSS Architecture

The application uses **SCSS** for styling with the following structure:

Global Styles (`src/app/@vb/css/`)

- `core.scss` - Core styles and resets
- `measurements.scss` - Spacing and sizing variables
- `colors.scss` - Color palette
- `utils.scss` - Utility classes
- `layout.scss` - Layout styles
- `flex-layout.scss` - Flexbox utility classes

Flex Layout CSS Classes

After migrating from Angular Flex Layout, the app uses custom CSS classes:

```
// Layout direction
.fx-layout-row { display: flex; flex-direction: row; }
.fx-layout-column { display: flex; flex-direction: column; }

// Flex sizing
.fx-flex-100 { flex: 1 1 100%; }
.fx-flex-50 { flex: 1 1 50%; }

// Alignment
.fx-layout-align-center-center {
  display: flex;
  justify-content: center;
  align-items: center;
}

// Responsive classes
.fx-flex-lg-50 { /* Applied on lg breakpoint */ }
```

Theme System

Color Palette

Defined in `colors.scss`:

```
$primary: #0190fe;
$success: #46be8a;
$warning: #f39834;
$danger: #fb434a;
```

Ant Design Theme

Customized through `src/app/@vb/css/vendors/antd/style.scss`

Material Theme

Pre-built theme: `indigo-pink` from `@angular/material`

Component Styling

Components use **encapsulated styles** with `.scss` files:

```
@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.scss']
})
```

Responsive Design

Breakpoints:

- **xs**: 0-599px (Mobile)
- **sm**: 600-959px (Tablet)
- **md**: 960-1279px (Small Desktop)
- **lg**: 1280-1919px (Desktop)
- **xl**: 1920px+ (Large Desktop)

Development Workflow

Code Quality

1. Prettier (Code Formatting)

```
npm run prettier
```

Configuration: `.prettierrc`

```
{
  "semi": false,
  "singleQuote": true,
  "printWidth": 100
}
```

```
}
```

2. TSLint (Code Quality)

```
npm run tslint
```

Configuration: `tslint.json`

Git Workflow

Pre-commit Hooks (Husky)

Automatically runs before commits:

- Prettier formatting
- TSLint checks

Configuration in `package.json`:

```
{
  "husky": {
    "hooks": {
      "pre-commit": "npm run lint-staged"
    }
  }
}
```

Development Best Practices

1. **Change Detection:** Use `OnPush` strategy for better performance

```
@Component({
  changeDetection: ChangeDetectionStrategy.OnPush
})
```

2. **RxJS Subscriptions:** Always unsubscribe or use `async` pipe

```
ngOnDestroy() {
  this.subscriptions.unsubscribe()
}
```

3. **Type Safety:** Avoid `any` types, define interfaces

```
interface Employee {  
  id: string  
  name: string  
  email: string  
}
```

4. **Lazy Loading:** Keep feature modules lazy-loaded

5. **Service Layer:** Keep business logic in services, not components

Build & Deployment

Build Configuration

Development Build

```
npm start  
# or  
ng serve --host 0.0.0.0 --port 4201
```

Features:

- Source maps enabled
- No optimization
- Fast rebuild times

Production Build

```
npm run build
```

Features:

- AOT compilation
- Tree shaking
- Minification
- Optimization
- Bundle size optimization

Build output: `dist/build/`

Build Optimization

Bundle Budgets

Defined in `angular.json`:

```
{
  "budgets": [{
    "type": "initial",
    "maximumWarning": "25mb",
    "maximumError": "50mb"
  }]
}
```

Lazy Loading Benefits

- Initial bundle: ~4.9MB
- 19 lazy-loaded chunks
- Features loaded on-demand

Deployment

Cloudflare Workers

```
npm run deploy
```

Deployment configuration: `.cloudflare/`

Deployment Steps

1. Build production version: `npm run build`
2. Test build locally: Serve from `dist/build/`
3. Deploy to Cloudflare: `npm run deploy`

Testing

Unit Testing

Framework

- **Karma:** Test runner
- **Jasmine:** Testing framework

Run Tests

```
npm test
```

Test Structure

```
describe('DashboardComponent', () => {  
  let component: DashboardComponent  
  let fixture: ComponentFixture<DashboardComponent>  
  
  beforeEach(async () => {  
    await TestBed.configureTestingModule({  
      declarations: [ DashboardComponent ]  
    }).compileComponents()  
  })  
  
  beforeEach(() => {  
    fixture = TestBed.createComponent(DashboardComponent)  
    component = fixture.componentInstance  
    fixture.detectChanges()  
  })  
  
  it('should create', () => {  
    expect(component).toBeTruthy()  
  })  
})
```

E2E Testing

Framework

- **Cypress:** Modern e2e testing (configured)

- **Protractor:** Legacy e2e (deprecated)

Configuration: `cypress.json`

Troubleshooting

Common Issues

1. Memory Issues During Build

Error: JavaScript heap out of memory

Solution: Use high-memory script

```
npm run ng-high-memory
```

2. Peer Dependency Warnings

Error: Peer dependency conflicts

Solution: Use legacy peer deps flag

```
npm install --legacy-peer-deps
```

3. OpenSSL Error (Node.js 17+)

Error: `digital envelope routines::unsupported`

Solution: Already configured in scripts with `--openssl-legacy-provider`

4. Port Already in Use

Error: Port 4201 already in use

Solution: Kill process or use different port

```
# Kill process on port 4201  
lsof -ti:4201 | xargs kill -9
```

```
# Or use different port
ng serve --port 4202
```

5. Module Not Found Errors

Error: Cannot find module

Solution: Clean install

```
rm -rf node_modules package-lock.json
npm install --legacy-peer-deps
```

Debug Mode

Enable Source Maps

Already enabled in development builds for debugging.

Chrome DevTools

Use Angular DevTools extension for debugging:

- Component inspector
- NgRx state inspector
- Performance profiler

Best Practices

Component Development

1. Keep Components Small

```
// Good: Small, focused component
@Component({
  selector: 'app-user-card',
  template: `<div>{{ user.name }}</div>`
})
```

```
export class UserCardComponent {  
  @Input() user: User  
}
```

2. Use Smart/Dumb Pattern

- **Smart Components:** Handle data and logic
- **Dumb Components:** Present data only

3. Lifecycle Hooks

```
ngOnInit() {  
  // Initialization logic  
}  
  
ngOnDestroy() {  
  // Cleanup subscriptions  
}
```

State Management

1. Keep Store Normalized

```
// Good: Normalized  
{  
  users: { 1: {...}, 2: {...} },  
  userIds: [1, 2]  
}  
  
// Bad: Nested  
{  
  users: [{...}, {...}]  
}
```

2. Use Selectors

```
export const getUsers = (state: AppState) => state.users  
export const getUserById = (id: string) =>  
  createSelector(getUsers, users => users[id])
```

Performance

1. Use OnPush Change Detection

```
@Component({
  changeDetection: ChangeDetectionStrategy.OnPush
})
```

2. Track By in *ngFor

```
<div *ngFor="let item of items; trackBy: trackById">
  {{ item.name }}
</div>
```

```
trackById(index: number, item: any): number {
  return item.id
}
```

3. Unsubscribe from Observables

```
private destroy$ = new Subject()

ngOnInit() {
  this.dataService.getData()
    .pipe(takeUntil(this.destroy$))
    .subscribe(data => { })
}

ngOnDestroy() {
  this.destroy$.next()
  this.destroy$.complete()
}
```

Security

1. Sanitize User Input

Use Angular's DomSanitizer for HTML content

2. HTTP Security

- Use HTTPS
- Validate tokens
- Handle errors securely

3. XSS Prevention

Angular automatically sanitizes values in templates

Additional Resources

Official Documentation

- [Angular Documentation](#)
- [NgRx Documentation](#)
- [Ant Design Angular](#)
- [Angular Material](#)

Internal Documentation

- `README.md` - Quick start guide
- `UPGRADE_NOTES.md` - Angular v13 → v15 upgrade details
- `LAZY_LOADING_IMPLEMENTATION.md` - Lazy loading implementation
- `FLEX_LAYOUT_MIGRATION.md` - Flex layout CSS migration

Visual Builder Resources

- [Visual Builder Cloud](#)
- [Visual Builder Docs](#)

Useful Commands Reference

```
# Development
```

```
npm start
```

```
# Start dev server
```

```
npm test          # Run tests
npm run prettier  # Format code
npm run tslint    # Lint code

# Build
npm run build     # Production build
npm run build-demo # Demo build

# Deployment
npm run deploy    # Deploy to Cloudflare

# Utilities
npm run vb        # Visual Builder CLI
npm run ng-high-memory # High memory mode
```

Appendix

Version History

- **Angular v13.3.0 → v15.2.10** (Upgraded December 2025)
- **NgRx v13.2.0 → v15.4.0**
- **TypeScript v4.6.4 → v4.9.5**

Migration Notes

- Migrated from `@Effect()` decorator to `createEffect()`
- Replaced Angular Flex Layout with CSS classes
- Implemented lazy loading for all feature modules
- Updated to Material Design Components (MDC)

Contributors

This documentation is maintained by the development team.

Support

For issues and feature requests:

- GitHub Issues: [ondeliveroper/on-internal-web](#)
-

Last Updated: February 2026

Document Version: 1.0

Angular Version: 15.2.10

Revision #1

Created 5 February 2026 03:42:50 by ondeliveroper

Updated 5 February 2026 03:43:08 by ondeliveroper