Frontend

Getting started

Getting started

Introduction

Project Overview

OnDelivery Dashboard is a comprehensive web-based logistics and delivery management system built with Angular. The application provides a complete suite of tools for managing deliveries, tracking packages, monitoring couriers, handling returns, and generating reports.

Key Features

- □□ Package tracking and scanning (incoming/outgoing)
- □□ Trucking and courier management
- □□ Comprehensive reporting and analytics
- □□ Finance and tariff management
- □□ Marketplace integration
- □□ Real-time notifications
- □□ Map-based tracking with Leaflet
- □□ JWT-based authentication with role management

Technology Stack

Core Framework

- **Angular 14.3.0** - Main frontend framework
- **TypeScript 4.8.4** - Programming language
- **RxJS 7.8.1** - Reactive programming library

UI & Styling

- **Bootstrap 5.3.2** - CSS framework for responsive design
- **Angular Material 14.2.7** - Material Design components

- **Angular Flex Layout 14.0.0** - Responsive layout system
- **ng-bootstrap 13.1.1** - Bootstrap components for Angular
- **Perfect Scrollbar** - Custom scrollbar styling

Additional Libraries

- **Leaflet 1.9.4** - Interactive maps for tracking
- **AngularX QRCode 14.0.0** - QR code generation
- **PDFMake 0.2.9** - PDF generation
- **ExcelJS 4.4.0** - Excel file manipulation
- **DayJS 1.11.10** - Date/time manipulation
- **ng2-dragula 3.2.0** - Drag and drop functionality
- **ngx-image-viewer 1.0.13** - Image viewing component

Development Tools

- **Angular CLI 14.2.13** - Build and development tooling
- **Karma & Jasmine** - Testing framework
- **TSLint** - TypeScript linting
- **Protractor** - End-to-end testing

Starting the project

Prerequisites

- Node.js (v14+ recommended)
- npm (v6+ recommended)
- Angular CLI (`npm install -g @angular/cli`)

Installation

1. **Clone the repository**

```
git clone <repository-url>
cd ondelivery-dash
```

2. **Install dependencies**

```
npm install
```

3. **Configure API endpoints** Edit `src/app/addrezz.ts` to set the appropriate API endpoints:

```
export const OSAS_API = "https://testapi.ondelivery.id";
export const BOOKING_API = "https://demoapisat.ondelivery.id";
export const USAPI_API = "http://localhost:3621";
```

Running the Application

Development Server

```
npm start
# or
ng serve
```

Navigate to `http://localhost:4200/`. The app will automatically reload on file changes.

Production Build

```
npm run build
```

Build artifacts are stored in the `dist/` directory with output hashing for cache busting.

Development Build

```
npm run build-dev
```

Builds without production optimizations.

Run Tests

```
npm test
```

Run Linter

```
npm run lint
```

Project Architecture

Directory Structure

```
src/
├── app/
│   ├── account-management/       # Account operations
│   ├── admin-menu/               # Admin navigation
│   ├── finance-management/       # Financial operations
│   ├── home/                     # Dashboard/home page
│   ├── incoming-scan/            # Incoming package scanning
│   ├── indopaket/                # Regional package handling
│   ├── layouts/                  # Layout components (header, sidebar, breadcrumb)
│   ├── login/                    # Authentication
│   ├── manifest/                 # Manifest management
│   ├── manual-handling/          # Manual handling operations
│   ├── marketplace/              # Marketplace integration
│   ├── material-component/       # Material Design wrappers
│   ├── mitra-courier/            # Courier partner management
│   ├── modal/                    # Modal dialogs
│   ├── models/                   # Data models
│   ├── monitoring-booking/       # Booking monitoring
│   ├── monitoring-courier/       # Courier monitoring
│   ├── notification/             # Notification system
│   ├── outgoing-scan/            # Outgoing package scanning
│   ├── package-return/           # Package returns
│   ├── package-return-handling/  # Return processing
│   ├── pod-approval/             # Proof of Delivery approval
│   ├── profile/                  # User profile
│   ├── reports/                  # Standard reports
│   ├── reports-dc/               # Distribution center reports
│   ├── saldo/                    # Account balance
│   ├── services/                 # Business logic services
│   ├── shared/                   # Shared utilities and components
```

```
│   ├── tariff/              # Pricing and tariff
│   ├── tracking/            # Package tracking
│   ├── trucking/            # Trucking operations
│   ├── user-manage/         # User management
│   ├── waybill/             # Waybill management
│   ├── api.service.ts       # Main API service
│   ├── auth.guard.ts        # Authentication guard
│   ├── role.guard.ts        # Role-based authorization guard
│   ├── app.module.ts        # Root module
│   └── app.routing.ts       # Routing configuration
├── assets/                  # Static assets (images, icons)
├── environments/            # Environment configurations
└── index.html               # Main HTML file
```

Architecture Pattern

The application follows Angular's **module-based architecture**:

1. **Root Module** (`app.module.ts`) - Core module with essential dependencies
2. **Feature Modules** - Lazy-loaded modules for each feature area
3. **Shared Module** - Reusable components and utilities
4. **Services** - Business logic and API communication
5. **Guards** - Route protection and authorization
6. **Models** - TypeScript interfaces for data structures

Module Loading Strategy

- **Eager Loading**: Core components (layouts, login, profile)
- **Lazy Loading**: All feature modules for optimal performance
- Routes are protected by `AuthGuard` requiring authentication

Development guidelines

Authentication and security

Authentication Flow

1. **Login** (`/login`)
   - User submits credentials
   - API returns JWT token
   - Token stored in `localStorage` as `jwt`
   - User details and roles stored in `localStorage`
2. **Token Management**
   - JWT token sent with every API request via `Authorization` header
   - Token validated on each route navigation via `AuthGuard`
   - Invalid tokens redirect to login page
3. **Logout**
   - Clear `localStorage`
   - Redirect to login page

Guards

AuthGuard (`auth.guard.ts`)

- Protects all authenticated routes
- Validates JWT token on navigation
- Calls `guardianTokens()` API to verify token validity
- Redirects to `/login` if authentication fails

```
// Usage in routing
{
  path: 'home',
  loadChildren: () => import('./home/home.module').then(m => m.HomeModule),
  canActivate: [AuthGuard]
}
```

RoleGuard (`role.guard.ts`)

- Provides role-based access control
- Checks user roles from `localStorage`

- Can be used for feature-specific authorization

Security Best Practices

1. **Token Storage**: JWT tokens stored in `localStorage` (consider `httpOnly` cookies for enhanced security)
2. **HTTPS**: Always use HTTPS in production
3. **Token Expiration**: Implement token refresh mechanism
4. **XSS Protection**: Angular's built-in sanitization helps prevent XSS
5. **CSRF Protection**: Use Angular's CSRF token support for POST requests

Application modules

The application consists of 27+ feature modules, each handling specific business functionality:

Core Modules

1. **Home Module** (`/home`)

- Dashboard with key metrics and statistics
- Quick access to frequently used features
- Real-time notifications and alerts

2. **Login Module** (`/login`)

- User authentication
- Credential validation
- Session initialization

3. **Profile Module** (`/profile`)

- User profile information
- Settings and preferences
- Password change functionality

Waybill Management

4. **Waybill Module** (`/waybill`)

- Create and manage waybills
- Multiple waybill types (PCP, POS, LC)
- Waybill printing and export

Services:

- `WaybillService` - Standard waybill operations
- `WaybillPcpService` - PCP waybill handling
- `WaybillPosService` - POS waybill handling
- `WaybillLcService` - LC waybill handling

- `WaybillPrintService` - Waybill printing

Scanning Operations

5. **Incoming Scan Module** (`/incoming-scan`)

- Receiving scan for incoming packages
- Delivery scan for completed deliveries
- Problem scan for issues

Services:

- `ReceivingScanService` - Handle incoming package scans
- `DeliveryScanService` - Handle delivery completion scans

6. **Outgoing Scan Module** (`/outgoing-scan`)

- Packed scan for ready-to-ship packages
- Handover scan for package transfers
- PCP handover scan

Services:

- `PackedScanService` - Packed package scanning
- `HandoverScanService` - Package handover operations
- `HandoverPcpScanService` - PCP-specific handovers

Monitoring & Tracking

7. **Tracking Module** (`/tracking`)

- Real-time package tracking
- Tracking history and status updates
- Map-based visualization (Leaflet integration)

8. **Monitoring Booking Module** (`/monitor-booking`)

- Monitor booking status
- Track booking lifecycle
- Booking analytics

9. **Monitoring Courier Module** (`/monitoring-courier`)

- Courier performance tracking
- Delivery assignments
- Courier location monitoring

Service: `MonitoringCourierService`

Package Returns

10. **Package Return Module** (`/package-return`)

- Initiate package returns
- Return request management
- Return status tracking

Service: `ReturnService`

11. **Package Return Handling Module** (`/return-handling`)

- Process return requests
- Return scanning and verification
- Return disposition

Reports & Analytics

12. **Reports Module** (`/reports`)

Comprehensive reporting system with multiple report types:

Services:

- `ReportService` - Main reporting service
- `MonitoringPaketKirimService` - Shipment monitoring reports
- `MonitoringManifestService` - Manifest reports
- `MonitoringPickupService` - Pickup reports
- `HistoryScanService` - Scan history reports
- `TotalPickupReportService` - Pickup totals
- `TotalDeliveryReportService` - Delivery totals
- `MonitoringSlaService` - SLA monitoring
- `MonitoringDestinationService` - Destination reports
- `FinanceService` - Financial reports

13. **Reports DC Module** (`/reports-dc`)

- Distribution Center specific reports
- DC performance metrics
- Inventory reports

Financial Management

14. **Finance Management Module** (`/finance-management`)

- Financial transactions
- Payment tracking
- Invoice management

Service: `FinanceManagementService`

15. **Tariff Module** (`/tariff`)

- Pricing configuration
- Tariff calculation
- Rate management

16. **Saldo Module** (`/saldo`)

- Account balance tracking
- Transaction history
- Balance top-up

Partner & User Management

17. **Mitra Courier Module** (`/mitra-courier`)

- Courier partner management
- Partner registration and verification
- Performance tracking

18. **User Management Module** (`/customer-manage`)

- User administration
- Role and permission management
- User activity tracking

19. **Account Management Module** (`/account-management`)

- Account operations

- Account settings
- Access control

Service: `AccountManagementService`

Logistics Operations

20. **Trucking Module** (`/trucking`)

- Trucking fleet management
- Shipment assignments
- Route optimization

Services:

- `TruckingService` - Trucking operations
- `TruckingShipmentService` - Shipment management

21. **Manifest Module** (`/manifest`)

- Manifest creation and management
- Package grouping
- Manifest tracking

22. **Manual Handling Module** (`/manual-handling`)

- Manual package processing
- Exception handling
- Manual data entry

Service: `ManualHandlingService`

23. **POD Approval Module** (`/pod-management`)

- Proof of Delivery approval workflow
- POD verification
- Dispute resolution

Service: `PodApprovalService`

Integration & Communication

24. **Marketplace Module** (`/marketplace`)

- Marketplace integration
- Order synchronization
- Marketplace-specific operations

Service: `MarketplaceService`

25. **Indopaket Module** (`/indopaket`)

- Indopaket integration
- Regional package handling
- Indopaket-specific workflows

Service: `IndopaketService`

26. **Notification Module** (`/notification`)

- In-app notifications
- Notification preferences
- Alert management

Service: `NotificationService`

Utility Modules

27. **Models Module** (`/models`)

- Data model definitions
- Type interfaces
- Enums and constants

Services & API Integration

Main API Service (`api.service.ts`)

The central service for API communication with the backend.

Key Methods:

- `getUser(username, password)` - User authentication
- `getUserDetails()` - Fetch user profile
- `guardianTokens()` - Validate JWT token
- Common HTTP methods with JWT authentication

Configuration:

```
// API endpoints defined in addrezz.ts
OSAS_API = "https://testapi.ondelivery.id"
BOOKING_API = "https://demoapisat.ondelivery.id"
USAPI_API = "http://localhost:3621"
```

Service Categories

1. **Scanning Services**

Located in `services/incoming-scans/` and `services/outgoing-scans/`

- Handle barcode/QR code scanning
- Validate package information
- Update package status

2. **Report Services**

Located in `services/reports/`

- Generate various reports
- Export to PDF/Excel
- Data aggregation and analytics

3. **Business Services**

Located in `services/`

- Domain-specific business logic
- Data transformation
- Workflow management

HTTP Interceptor Pattern

All services use JWT tokens for authentication:

```
let customHeaders = new HttpHeaders({
  'Content-Type': 'application/json',
  'authorization': localStorage.getItem('jwt')!
});
```

PDF & Excel Export Services

PDF Service (`pdf.service.ts`)

- Uses PDFMake library
- Generate reports as PDF
- Custom formatting and styling

Excel Export

- Uses ExcelJS library
- Export data to Excel format
- Multiple sheet support

Routing & Navigation

Route Configuration

All routes are defined in `app.routing.ts` with lazy loading for performance.

Main Routes

Route	Module	Access
`/`	Redirect to `/login`	Public
`/login`	LoginComponent	Public
`/profile`	ProfileComponent	Authenticated
`/home`	HomeModule	Authenticated
`/waybill`	WaybillModule	Authenticated
`/incoming-scan`	IncomingScanModule	Authenticated
`/outgoing-scan`	OutgoingScanModule	Authenticated
`/tracking`	TrackingModule	Authenticated
`/reports`	ReportsModule	Authenticated
`/reports-dc`	ReportsDcModule	Authenticated
`/tariff`	TariffModule	Authenticated
`/package-return`	PackageReturnModule	Authenticated
`/return-handling`	PackageReturnHandlingModule	Authenticated
`/manifest`	ManifestModule	Authenticated
`/indopaket`	IndopaketModules	Authenticated
`/monitor-booking`	MonitoringBookingModule	Authenticated
`/monitoring-courier`	MonitoringCourierModule	Authenticated
`/mitra-courier`	MitraCourierModule	Authenticated
`/marketplace`	MarketplaceModule	Authenticated
`/account-management`	AccountManagementModule	Authenticated

Route	Module	Access
`/customer-manage`	UserManageModule	Authenticated
`/manual-handling`	ManualHandlingModule	Authenticated
`/pod-management`	PodApprovalModule	Authenticated
`/trucking`	TruckingModule	Authenticated
`/finance-management`	FinanceManagementModule	Authenticated
`/saldo`	SaldoModule	Authenticated
`/notification`	NotificationModule	Authenticated

Navigation Structure

Main Layout (`layouts/full/`)

- **Header** (`header.component`) - Top navigation, user menu
- **Sidebar** (`sidebar.component`) - Main navigation menu
- **Breadcrumb** (`breadcrumb.component`) - Breadcrumb navigation

Menu Configuration

Menu items are defined in `shared/menu-items/` based on user roles.

UI Components & Styling

Layout System

The application uses **Angular Flex Layout** for responsive design:

- Flexbox-based layouts
- Responsive breakpoints
- Dynamic layout changes

CSS Framework

Bootstrap 5.3.2 provides:

- Grid system
- Utility classes
- Responsive components
- Form controls

Angular Material Components

Used throughout the application:

- `mat-card` - Card containers
- `mat-table` - Data tables with sorting and pagination
- `mat-dialog` - Modal dialogs
- `mat-form-field` - Form inputs
- `mat-select` - Dropdown selects
- `mat-datepicker` - Date pickers
- `mat-icon` - Material icons
- `mat-button` - Buttons
- `mat-menu` - Context menus
- `mat-toolbar` - Toolbars
- `mat-sidenav` - Side navigation

Custom Components

Spinner Component (`shared/spinner.component`)

Loading indicator shown during async operations.

Modal Component (`modal/modal.component`)

Reusable modal dialog for various purposes.

Accordion Component (`shared/accordion/`)

Collapsible content panels.

Perfect Scrollbar

Custom scrollbar styling for better aesthetics:

```
const DEFAULT_PERFECT_SCROLLBAR_CONFIG: PerfectScrollbarConfigInterface = {
  suppressScrollX: true
};
```

Icons & Images

- Material Icons for standard UI elements
- Custom icons in `icons/` directory
- Static assets in `assets/` directory

State Management

LocalStorage

The application uses browser `localStorage` for state persistence:

Stored Data:

- `jwt` - JWT authentication token
- `agentID` - Current user/agent ID
- `role` - User roles (JSON array)
- `username` - Username
- Other session-specific data

Observable Pattern (RxJS)

Services use RxJS Observables for reactive data flow:

```
// Example from ApiService
return this.http.post<any>(OSAS_API + '/auth/signin', payload, { headers });
```

Components subscribe to Observables:

```
this.apiService.getUser(username, password).subscribe(
  response => { /* handle response */ },
  error => { /* handle error */ }
);
```

Service State

Services maintain application state:

- Singleton services (providedIn: 'root')
- State shared across components
- Data caching where appropriate

Build & Deployment

Build Configuration

Angular build configuration in `angular.json`:

Project Name: `project` (generic name) **Output Directory:** `dist/agent` **Source Root:** `src`

Build Commands

Development Build

```
npm run build-dev
# Creates development build with source maps
# Output: dist/agent/
```

Production Build

```
npm run build
# Optimized production build
# Features:
#   - Minification
#   - Tree shaking
#   - AOT compilation
#   - Output hashing (cache busting)
# Output: dist/agent/
```

Build Optimization

Production Optimizations:

- Ahead-of-Time (AOT) compilation
- Bundle minification
- Dead code elimination (tree shaking)

- Output hashing for cache busting (`--output-hashing=all`)
- Lazy loading for smaller initial bundle

Deployment Steps

1. **Build the application**

```
npm run build
```

2. **Deploy `dist/agent/` directory to web server**
   - Configure web server for single-page application
   - Redirect all routes to `index.html`
3. **Configure API endpoints**
   - Update `src/app/addrezz.ts` before building
   - Set production API URLs
4. **Environment Configuration**
   - Production: `src/environments/environment.prod.ts`
   - Development: `src/environments/environment.ts`

Web Server Configuration

Nginx Example:

```
server {
    listen 80;
    server_name yourdomain.com;
    root /path/to/dist/agent;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }
}
```

Apache Example:

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /
    RewriteRule ^index\.html$ - [L]
```

```
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule . /index.html [L]
</IfModule>
```

Performance Considerations

1. **Lazy Loading** - Feature modules loaded on demand
2. **Code Splitting** - Separate bundles for vendor and app code
3. **Output Hashing** - Cache busting for updated files
4. **Compression** - Enable gzip/brotli on web server
5. **CDN** - Serve static assets from CDN

Development guide

Code Style

TypeScript

- Use TypeScript strict mode
- Define interfaces for all data structures
- Use type annotations for function parameters and returns

Angular

- Follow Angular style guide
- Use Angular CLI for scaffolding
- One component per file
- Use services for business logic

Naming Conventions

- Components: `PascalCase` (e.g., `WaybillListComponent`)
- Services: `PascalCase` with `Service` suffix (e.g., `WaybillService`)
- Modules: `PascalCase` with `Module` suffix (e.g., `WaybillModule`)
- Files: `kebab-case` (e.g., `waybill-list.component.ts`)

Component Structure

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

@Component({
  selector: 'app-component-name',
  templateUrl: './component-name.component.html',
  styleUrls: ['./component-name.component.css']
})
export class ComponentNameComponent implements OnInit {
  constructor(private service: SomeService) { }
```

```
  ngOnInit(): void {
    // Initialization logic
  }
}
```

Service Structure

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class SomeService {
  constructor(private http: HttpClient) { }

  getData(): Observable<any> {
    const headers = new HttpHeaders({
      'Authorization': localStorage.getItem('jwt')!
    });
    return this.http.get<any>(API_URL, { headers });
  }
}
```

Best Practices

1. **Component Communication**

- Use `@Input()` for parent-to-child data flow
- Use `@Output()` with EventEmitter for child-to-parent events
- Use services for complex state management

2. **Error Handling**

- Always handle Observable errors
- Provide user-friendly error messages
- Log errors for debugging

```
this.service.getData().subscribe(
  data => this.data = data,
  error => {
    console.error('Error:', error);
    // Show error message to user
  }
);
```

3. **Memory Management**

- Unsubscribe from Observables in `ngOnDestroy()`
- Use async pipe in templates when possible
- Avoid memory leaks

```
private subscription: Subscription;

ngOnInit() {
  this.subscription = this.service.getData().subscribe(/*...*/);
}

ngOnDestroy() {
  this.subscription?.unsubscribe();
}
```

4. **Performance**

- Use `trackBy` with `*ngFor` for large lists
- Implement `OnPush` change detection where appropriate
- Lazy load images and heavy components

5. **Security**

- Never store sensitive data in localStorage (use httpOnly cookies)
- Sanitize user input
- Validate all data from API
- Use HTTPS in production

Testing

Unit Tests

```
npm test
# Runs Karma test runner with Jasmine
```

E2E Tests

```
npm run e2e
# Runs Protractor for end-to-end tests
```

Writing Tests

```
describe('ComponentName', () => {
  let component: ComponentName;
  let fixture: ComponentFixture<ComponentName>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ ComponentName ]
    }).compileComponents();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Linting

```
npm run lint
# Runs TSLint to check code quality
```

Fix linting issues automatically:

```
ng lint --fix
```

Git Workflow

1. Create feature branch: `git checkout -b feature/feature-name`
2. Make changes and commit: `git commit -m "feat: description"`

3. Push to remote: `git push origin feature/feature-name`
4. Create pull request for review

Commit Message Format:

- `feat:` New feature
- `fix:` Bug fix
- `docs:` Documentation changes
- `style:` Code style changes
- `refactor:` Code refactoring
- `test:` Test additions/changes
- `chore:` Build/tooling changes

Troubleshooting

Common Issues

1. **Cannot connect to API**

Symptom: API calls fail with connection errors

Solution:

- Check API endpoints in `src/app/addrezz.ts`
- Verify API server is running
- Check CORS configuration on API server
- Verify network connectivity

2. **Authentication fails after page refresh**

Symptom: User logged out after page refresh

Solution:

- Check JWT token in localStorage (browser DevTools)
- Verify token expiration
- Ensure `AuthGuard` is properly validating tokens
- Check `guardianTokens()` API endpoint

3. **Lazy-loaded module not found**

Symptom: Error when navigating to a route

Solution:

- Verify module path in `app.routing.ts`
- Check module export in feature module
- Run `ng build` to check for compilation errors
- Clear browser cache and rebuild

4. **Styling issues**

Symptom: Components not displaying correctly

Solution:

- Check Bootstrap and Material imports in `angular.json`
- Verify CSS files are imported in `styles.css`
- Clear browser cache
- Check for CSS conflicts

5. **Build errors**

Symptom: Build fails with TypeScript errors

Solution:

- Run `npm install` to ensure dependencies are installed
- Check TypeScript version compatibility
- Fix TypeScript errors reported by compiler
- Clear `node_modules` and reinstall: `rm -rf node_modules && npm install`

6. **Performance issues**

Symptom: Application is slow or unresponsive

Solution:

- Enable production mode build
- Check for memory leaks (unsubscribed Observables)
- Optimize large lists with virtual scrolling
- Lazy load heavy components
- Profile with Chrome DevTools Performance tab

Debugging Tips

1. **Browser DevTools**

- Use Console to check for JavaScript errors
- Use Network tab to inspect API calls
- Use Application tab to view localStorage
- Use Performance tab to profile application

2. **Angular DevTools**

Install Angular DevTools Chrome extension:

- Inspect component tree
- View component properties

- Profile change detection
- Debug performance issues

3. **Enable Source Maps**

For production debugging, build with source maps:

```
ng build --source-map
```

4. **Logging**

Add console logging for debugging:

```
console.log('Debug:', variable);
console.error('Error:', error);
```

Getting Help

1. **Check Angular Documentation:** https://angular.io/docs
2. **Check Angular Material Documentation:** https://material.angular.io/
3. **Check Stack Overflow:** Tag questions with `angular` and `typescript`
4. **Review Application Logs:** Check browser console and network tab
5. **Contact Development Team:** For application-specific issues

Additional Resources

Official Documentation

- Angular Documentation
- TypeScript Documentation
- RxJS Documentation
- Angular Material
- Bootstrap Documentation
- Leaflet Documentation

Useful Tools

- Angular CLI
- Visual Studio Code (Recommended IDE)
- Angular Language Service (VS Code Extension)
- Augury (Angular Debugging Tool)
- Postman (API Testing)

Learning Resources

- Angular University
- Angular Official Tutorial
- TypeScript Handbook
- RxJS Marbles (Interactive RxJS diagrams)