

# Frontend

- [OnMarket v02 Frontend - Developer Documentation](#)

# OnMarket v02 Frontend - Developer Documentation

## Table of Contents

1. [Project Overview](#)
  2. [Technology Stack](#)
  3. [Getting Started](#)
  4. [Project Architecture](#)
  5. [Folder Structure](#)
  6. [Routing Configuration](#)
  7. [Core Components](#)
  8. [Feature Modules](#)
  9. [Services and Data Models](#)
  10. [Angular Configuration](#)
  11. [Development Workflow](#)
  12. [Build and Deployment](#)
  13. [Coding Conventions](#)
  14. [Troubleshooting](#)
- 

## Project Overview

**OnMarket v02** is a modern Angular-based frontend application for an international trade marketplace platform. The application facilitates connections between importers and verified exporters, providing features for browsing products, reading blog content, and managing user authentication.

## Key Features

- **Landing Page:** Showcases platform features, product categories, and business opportunities
- **Blog System:** Full-featured blog with list view, detail pages, and carousel highlights
- **Authentication:** Login and registration pages for user management
- **Responsive Layout:** Modern header with category navigation and footer
- **Server-Side Rendering (SSR):** Built-in SSR support for improved performance and SEO
- **Zoneless Change Detection:** Uses Angular's modern zoneless architecture for better performance

## Target Audience

This documentation is intended for:

- New developers joining the project
  - Frontend developers maintaining or extending the application
  - DevOps engineers deploying the application
- 

## Technology Stack

### Core Technologies

Technology	Version	Purpose
Angular	21.0.8	Frontend framework
TypeScript	5.9.3	Primary programming language
RxJS	7.8.2	Reactive programming library
Express	5.2.1	Server for SSR

### Build Tools

- **Angular CLI** (21.0.5): Project scaffolding and build tool
- **Angular SSR** (21.0.5): Server-side rendering support

### Testing

- **Karma** (6.4.4): Test runner

- **Jasmine** (5.13.0): Testing framework

# Code Quality

- **Prettier**: Code formatting (configured in package.json)
  - **TypeScript Strict Mode**: Enabled for type safety
- 

# Getting Started

## Prerequisites

Before you begin, ensure you have the following installed:

- **Node.js**: Version 18.x or higher (LTS recommended)
- **npm**: Version 9.x or higher (comes with Node.js)
- **Git**: For version control

## Installation

### 1. Clone the repository:

```
git clone <repository-url>  
cd onmarket-v02-frontend
```

### 2. Install dependencies:

```
npm install
```

### 3. Verify installation:

```
ng version
```

# Running the Application

## Development Server

Start the development server with hot reload:

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

The application will be available at `http://localhost:4200/`.

## Production Build with SSR

Build the application for production with server-side rendering:

```
npm run build
```

Serve the built application:

```
npm run serve:ssr:onmarket
```

The SSR server will run on `http://localhost:4000` (or the port specified in the `PORT` environment variable).

---

# Project Architecture

## Architectural Pattern

The application follows Angular's **standalone component architecture** with:

- **Feature-based organization:** Code organized by features (landing, blog, login, register)
- **Core module pattern:** Shared layout components in the core directory
- **Service layer:** Centralized data management through Angular services
- **Lazy loading:** Components are loaded on-demand using dynamic imports
- **Reactive programming:** RxJS observables for asynchronous operations

## Key Architectural Decisions

1. **Standalone Components:** All components are standalone (no NgModules)
2. **Zoneless Change Detection:** Uses `provideZonelessChangeDetection()` for better performance
3. **Server-Side Rendering:** Built-in SSR for improved initial load time and SEO

4. **Signal-based State:** Uses Angular signals for reactive state management
  5. **Route-level Code Splitting:** Each route lazy-loads its component
- 

# Folder Structure

```
onmarket-v02-frontend/
├─ src/
│  └─ app/
│     └─ core/           # Core/shared functionality
│        └─ layout/     # Layout components
│           └─ main-layout/ # Main page layout wrapper
│              └─ header/ # Application header
│                 └─ footer/ # Application footer
│                    └─ features/ # Feature modules
│                       └─ landing/ # Landing page
│                          └─ blog/ # Blog feature
│                             └─ blog-list/ # Blog listing page
│                                └─ blog-detail/ # Blog detail page
│                                   └─ components/ # Blog-specific components
│                                      └─ blog-card/ # Reusable blog card
│                                         └─ models/ # Blog data models
│                                            └─ services/ # Blog service
│                                               └─ login/ # Login page
│                                                  └─ register/ # Registration page
│                                                     └─ app.ts # Root component
│                                                        └─ app.html # Root template
│                                                           └─ app.css # Root styles
│                                                              └─ app.config.ts # Application configuration
│                                                                 └─ app.config.server.ts # SSR configuration
│                                                                    └─ app.routes.ts # Client-side routing
│                                                                       └─ app.routes.server.ts # Server-side routing
│                                                                           └─ assets/ # Static assets
│                                                                              └─ icons/ # Icon files
│                                                                                 └─ images/ # Image files
│                                                                                     └─ main.ts # Client-side entry point
│                                                                                        └─ main.server.ts # Server-side entry point
│                                                                                             └─ server.ts # Express server configuration
```

— index.html	# HTML template
— styles.css	# Global styles
— public/	# Public assets (favicon, etc.)
— angular.json	# Angular workspace configuration
— package.json	# Dependencies and scripts
— tsconfig.json	# TypeScript configuration
— tsconfig.app.json	# App-specific TypeScript config
— tsconfig.spec.json	# Test-specific TypeScript config
— README.md	# Basic project README

## Directory Conventions

- **core/**: Shared components used across the application (layouts, guards, interceptors)
- **features/**: Feature-specific modules with their own components, services, and models
- **assets/**: Static files like images, icons, and fonts
- **public/**: Files served as-is (favicon.ico, robots.txt, etc.)

# Routing Configuration

## Route Structure

The application uses Angular's file-based routing with lazy loading:

```
// app.routes.ts
export const routes: Routes = [
  // Main layout routes (with header and footer)
  {
    path: '',
    loadChildren: () => import('./core/layout/main-layout/main-layout.component'),
    children: [
      {
        path: '',
        loadChildren: () => import('./features/landing/landing.component'),
      },
      {
        path: 'blog',
```

```

    loadComponent: () => import('./features/blog/blog-list/blog-list.component'),
  },
  {
    path: 'blog/:slug',
    loadComponent: () => import('./features/blog/blog-detail/blog-detail.component'),
  },
],
},

// Auth routes (without layout)
{
  path: 'login',
  loadComponent: () => import('./features/login/login.component'),
},
{
  path: 'register',
  loadComponent: () => import('./features/register/register.component'),
},

// Catch-all redirect
{ path: '**', redirectTo: '' },
];

```

## Available Routes

Route	Component	Description	Layout
/	LandingComponent	Homepage	With layout
/blog	BlogListComponent	Blog listing page	With layout
/blog/:slug	BlogDetailComponent	Individual blog post	With layout
/login	LoginComponent	User login	No layout
/register	RegisterComponent	User registration	No layout

## Routing Features

- **Lazy Loading:** All routes use dynamic imports for code splitting
- **Layout Strategy:** Main routes use a layout wrapper, auth routes don't
- **Dynamic Parameters:** Blog detail uses `:slug` parameter

- **Catch-all Route:** Redirects unknown routes to homepage
- 

# Core Components

## Main Layout Component

**Location:** `src/app/core/layout/main-layout/main-layout.component.ts`

The main layout component wraps content pages with a consistent header and footer.

```
@Component({
  selector: 'app-main-layout',
  standalone: true,
  imports: [RouterOutlet, HeaderComponent, FooterComponent],
  templateUrl: './main-layout.component.html',
  styleUrls: ['./main-layout.component.css'],
})
export class MainLayoutComponent {}
```

### Template Structure:

```
<app-header></app-header>
<main>
  <router-outlet></router-outlet>
</main>
<app-footer></app-footer>
```

## Header Component

**Location:** `src/app/core/layout/header/header.component.ts`

The header component provides navigation and category browsing functionality.

### Features:

- Category dropdown menu with multiple product categories
- Mobile-responsive navigation
- Keyboard accessibility (ESC key closes dropdown)

- Dynamic category data structure

### Key Properties:

```
isCategoryOpen: boolean = false;  
activeCategoryId: number = 0;  
categories: Array<{ name: string; items: Array<{ name: string }> }>;
```

### Key Methods:

- `toggleCategory()`: Opens/closes category dropdown
- `setActiveCategory(index: number)`: Changes active category tab
- `slugify(input: string)`: Converts category names to URL-friendly slugs
- `@HostListener('document:keydown.escape')`: Closes dropdown on ESC key

## Footer Component

**Location:** `src/app/core/layout/footer/footer.component.ts`

Provides footer content for the application.

---

## Feature Modules

### Landing Page

**Location:** `src/app/features/landing/`

The landing page showcases the platform's main features and product categories.

#### Key Features:

- Feature highlights with icons
- Product category showcase with emojis
- Factory highlights carousel
- Business opportunity (BSO) sections with image carousels
- Call-to-action sections

#### Key Properties:

```
features: Array<{ title, description, icon, alt }>;
ctaCategories: Array<{ label, emoji }>;
factoryHighlights: Array<{ title, description, imageUrl }>;
bsoSections: Array<{ title, slides: Array<{ title, imageUrl }> }>;
```

### Carousel Methods:

- `nextBso(idx: number)`: Advances to next slide
- `prevBso(idx: number)`: Goes to previous slide

## Blog Feature

The blog feature is a complete content management system with listing and detail views.

### Blog List Component

**Location:** `src/app/features/blog/blog-list/`

Displays blog posts with a highlighted carousel and full list view.

#### Features:

- Carousel for highlighted blogs (auto-play, 5-second interval)
- Grid view of all blog posts
- Loading states
- Signal-based reactive state management

#### Key Signals:

```
highlightedBlogs = signal<Blog[]>([]);
allBlogs = signal<Blog[]>([]);
currentSlide = signal(0);
isLoading = signal(true);
```

#### Lifecycle:

- `ngOnInit()`: Loads blog data
- `ngOnDestroy()`: Cleans up auto-play interval

### Blog Detail Component

**Location:** `src/app/features/blog/blog-detail/`

Displays individual blog post content with full details.

# Blog Card Component

**Location:** `src/app/features/blog/components/blog-card/`

Reusable component for displaying blog preview cards.

## Inputs:

```
@Input() blog: Blog;
```

# Blog Service

**Location:** `src/app/features/blog/services/blog.service.ts`

Service for fetching blog data from the API.

**API Endpoint:** `https://demosunshineapi.onindonesia.id/api/cms/blog`

## Methods:

```
getBlogs(pageIndex: number, pageSize: number): Observable<BlogListResponse>  
getLatestBlogs(count: number): Observable<Blog[]>  
getAllBlogs(): Observable<Blog[]>  
getBlogDetail(id: string): Observable<BlogDetail>
```

## Usage Example:

```
this.blogService.getLatestBlogs(3).subscribe({  
  next: (blogs) => this.highlightedBlogs.set(blogs),  
  error: (err) => console.error('Error loading blogs:', err)  
});
```

# Blog Models

**Location:** `src/app/features/blog/models/blog.model.ts`

## Interfaces:

```
interface Blog {  
  id: number;  
  key: string;  
  title: string;  
  url: string;
```

```
image: BlogImage;
dept: string;
category: string;
description: string;
createdAt: string;
updatedAt: string;
createdBy: string;
tiktok_embed: string | null;
youtube_embed: string | null;
ig_embed: string | null;
}

interface BlogDetail extends Blog {
  content: string;
  updatedBy: string;
}

interface BlogImage {
  name: string;
  status: string;
  url: string;
  uid: string;
}

interface BlogListResponse {
  rows: Blog[];
  count: number;
}
```

# Login Page

**Location:** `src/app/features/login/`

Provides user authentication interface.

## Features:

- Username and password inputs
- Form validation
- LocalStorage for session management (basic implementation)
- Navigation to homepage after successful login

## Key Properties:

```
username: string = "";  
password: string = "";
```

## Key Methods:

- `onLogin()`: Validates credentials and redirects to homepage

# Register Page

**Location:** `src/app/features/register/`

Provides user registration interface.

---

# Services and Data Models

## Service Pattern

Services in this application follow Angular's dependency injection pattern:

```
@Injectable({  
  providedIn: 'root', // Singleton service  
})  
export class BlogService {  
  private http = inject(HttpClient); // Modern inject() API  
  
  // Service methods...  
}
```

## Data Flow

1. **Component** initiates data request
2. **Service** makes HTTP call to API
3. **RxJS Observable** streams response
4. **Component** subscribes and updates UI with signals

## Example Flow:

```
// In component
ngOnInit() {
  this.blogService.getAllBlogs().subscribe({
    next: (blogs) => this.allBlogs.set(blogs),
    error: (err) => console.error(err)
  });
}
```

# API Integration

Currently, the application connects to:

- **Blog API:** `https://demosunshineapi.onindonesia.id/api/cms/blog`

To switch to a local API:

```
// In blog.service.ts
private readonly apiUrl = 'http://localhost:3601/api/cms/blog';
```

# Angular Configuration

## Application Configuration

**Location:** `src/app/app.config.ts`

```
export const appConfig: ApplicationConfig = {
  providers: [
    provideBrowserGlobalErrorListeners(), // Global error handling
    provideZonelessChangeDetection(),    // Modern zoneless mode
    provideRouter(routes),                // Routing
    provideClientHydration(withEventReplay()), // SSR hydration
    provideHttpClient(withFetch())        // HTTP with fetch API
  ]
};
```

# Server Configuration

**Location:** `src/app/app.config.server.ts`

Provides server-specific configuration for SSR.

# TypeScript Configuration

**Strict Mode Enabled:**

- `strict`: true
- `noImplicitOverride`: true
- `noPropertyAccessFromIndexSignature`: true
- `noImplicitReturns`: true
- `noFallthroughCasesInSwitch`: true

**Target:** ES2022 (modern JavaScript features)

# Build Configuration

**Location:** `angular.json`

**Key Settings:**

- **Output mode:** `server` (SSR enabled)
- **Budget limits:** 500kB warning, 1MB error for initial bundle
- **Source maps:** Enabled in development
- **Optimization:** Enabled in production

---

# Development Workflow

## Development Commands

```
# Start dev server
npm start
# or
ng serve
```

```
# Build for production
```

```
npm run build
```

```
# Run unit tests
```

```
npm test
```

```
# or
```

```
ng test
```

```
# Watch mode for development
```

```
npm run watch
```

```
# or
```

```
ng build --watch --configuration development
```

```
# Serve SSR build
```

```
npm run serve:ssr:onmarket
```

## Code Generation

Generate new components using Angular CLI:

```
# Generate a new component
```

```
ng generate component features/my-feature
```

```
# Generate a service
```

```
ng generate service features/my-feature/services/my-service
```

```
# Generate a guard
```

```
ng generate guard core/guards/auth
```

```
# Generate an interceptor
```

```
ng generate interceptor core/interceptors/auth
```

## Code Formatting

The project uses Prettier for consistent code formatting.

**Configuration** (from package.json):

```
"prettier": {
  "printWidth": 100,
  "singleQuote": true,
  "overrides": [
    {
      "files": "*.html",
      "options": {
        "parser": "angular"
      }
    }
  ]
}
```

### Format code:

```
# Format all files
npx prettier --write .

# Format specific files
npx prettier --write "src/**/*.ts"
```

## Hot Reload

The development server supports hot module replacement (HMR):

- Changes to TypeScript, HTML, and CSS files trigger automatic reload
- No manual refresh needed
- Fast rebuild times

---

## Build and Deployment

### Production Build

Build the application with optimizations:

```
npm run build
```

This creates:

- `dist/onmarket/browser/`: Client-side bundles
- `dist/onmarket/server/`: Server-side bundles

## Build Output

The production build includes:

- **Optimized bundles**: Minified and tree-shaken
- **Output hashing**: Cache-busting for static assets
- **Source maps**: Available for debugging (if enabled)
- **SSR server**: Express server for rendering

## Deployment Options

### Option 1: Node.js Server

Deploy the built application to a Node.js server:

```
# Build
npm run build

# Deploy dist/ folder to server

# On server, run:
PORT=4000 node dist/onmarket/server/server.mjs
```

### Option 2: PM2 Process Manager

Use PM2 for production deployment:

```
# Install PM2 globally
npm install -g pm2

# Start with PM2
pm2 start dist/onmarket/server/server.mjs --name onmarket

# Save PM2 configuration
pm2 save
```

```
# Set up auto-start on reboot
pm2 startup
```

## Option 3: Docker

Create a `Dockerfile`:

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY dist/ ./dist/
ENV PORT=4000
EXPOSE 4000
CMD ["node", "dist/onmarket/server/server.mjs"]
```

Build and run:

```
docker build -t onmarket-frontend .
docker run -p 4000:4000 onmarket-frontend
```

## Environment Variables

The application supports environment configuration:

- `PORT`: Server port (default: 4000)
- `pm_id`: PM2 process ID (auto-detected)

**Example:**

```
PORT=8080 node dist/onmarket/server/server.mjs
```

## Coding Conventions

## TypeScript Guidelines

1. **Use strict typing:** Avoid `any` type
2. **Use modern syntax:** Use signals, `inject()`, and standalone components
3. **Prefer const:** Use `const` for variables that don't change
4. **Use arrow functions:** For callbacks and event handlers

#### Good:

```
const user = signal<User | null>(null);
const http = inject(HttpClient);
```

#### Bad:

```
let user: any;
var service = new HttpClient();
```

## Component Guidelines

1. **Standalone components:** Always use `standalone: true`
2. **Signal-based state:** Prefer signals over traditional properties for reactive state
3. **OnPush strategy:** Use with signals for optimal performance (implicit with zoneless)
4. **Input/Output decorators:** Use for component communication

#### Example Component:

```
@Component({
  selector: 'app-my-component',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './my-component.html',
  styleUrls: ['./my-component.css']
})
export class MyComponent {
  data = signal<string[]>([]);

  @Input() title: string = '';
  @Output() itemClicked = new EventEmitter<string>();
}
```

## Template Guidelines

1. **Use structural directives:** `@if`, `@for`, `@switch` (Angular 17+ syntax)
2. **Async pipe:** Use for observables in templates
3. **Track by:** Always use `track` with `@for` loops
4. **Accessibility:** Include ARIA labels and semantic HTML

### Example Template:

```
@if (isLoading()) {  
  <p>Loading...</p>  
} @else {  
  <ul>  
    @for (item of items(); track item.id) {  
      <li>{{ item.name }}</li>  
    }  
  </ul>  
}
```

## Service Guidelines

1. **Injectable decorator:** Use `providedIn: 'root'` for singleton services
2. **Modern inject API:** Use `inject()` instead of constructor injection
3. **RxJS operators:** Use pipe and operators for stream manipulation
4. **Error handling:** Always handle errors in subscriptions

### Example Service:

```
@Injectable({  
  providedIn: 'root',  
})  
export class DataService {  
  private http = inject(HttpClient);  
  
  getData(): Observable<Data[]> {  
    return this.http.get<Data[]>('/api/data').pipe(  
      catchError(error => {  
        console.error('Error loading data:', error);  
        return of([]);  
      })  
    );  
  }  
}
```

```
}
```

# CSS Guidelines

1. **Component styles:** Keep styles scoped to components
2. **Global styles:** Only for resets and typography
3. **BEM naming:** Use BEM (Block Element Modifier) convention for class names
4. **Responsive design:** Use mobile-first approach

## Example:

```
/* Component styles */  
.blog-card {  
  padding: 1rem;  
}  
  
.blog-card__title {  
  font-size: 1.5rem;  
}  
  
.blog-card__title--highlighted {  
  color: #ff6b6b;  
}
```

# File Naming Conventions

- **Components:** `component-name.component.ts`
- **Services:** `service-name.service.ts`
- **Models:** `model-name.model.ts`
- **Guards:** `guard-name.guard.ts`
- **Interceptors:** `interceptor-name.interceptor.ts`

# Git Commit Guidelines

Use conventional commit messages:

```
feat: Add blog detail page  
fix: Resolve routing issue in header  
docs: Update README with deployment instructions
```

```
style: Format code with Prettier
refactor: Simplify blog service logic
test: Add tests for login component
chore: Update dependencies
```

# Troubleshooting

## Common Issues

### Issue: "Module not found" errors

**Solution:** Ensure all dependencies are installed:

```
rm -rf node_modules package-lock.json
npm install
```

### Issue: Port already in use

**Solution:** Kill the process on port 4200/4000 or use a different port:

```
# Kill process on Mac/Linux
lsof -ti:4200 | xargs kill

# Or use different port
ng serve --port 4201
```

### Issue: SSR hydration errors

**Solution:** Ensure client and server rendering produce identical HTML. Common causes:

- Date/time rendering differences
- Random data generation
- Browser-only APIs (window, document)

**Fix:** Use platform checks:

```
import { isPlatformBrowser } from '@angular/common';
import { PLATFORM_ID, inject } from '@angular/core';
```

```
export class MyComponent {
  private platformId = inject(PLATFORM_ID);

  ngOnInit() {
    if (isPlatformBrowser(this.platformId)) {
      // Browser-only code
    }
  }
}
```

## Issue: CORS errors in development

**Solution:** Configure a proxy in `angular.json`:

```
"serve": {
  "options": {
    "proxyConfig": "proxy.conf.json"
  }
}
```

Create `proxy.conf.json`:

```
{
  "/api": {
    "target": "https://demosunshineapi.onindonesia.id",
    "secure": false,
    "changeOrigin": true
  }
}
```

## Issue: Build fails with memory error

**Solution:** Increase Node.js memory:

```
NODE_OPTIONS="--max-old-space-size=4096" npm run build
```

# Performance Optimization

1. **Lazy loading:** All routes are already lazy-loaded

2. **OnPush change detection:** Use signals for automatic OnPush optimization
3. **Image optimization:** Use Angular's image directive:

```
import { NgOptimizedImage } from '@angular/common';

// In template
<img ngSrc="path/to/image.jpg" width="500" height="300" priority>
```

4. **Bundle analysis:** Analyze bundle size:

```
ng build --stats-json
npx webpack-bundle-analyzer dist/onmarket/browser/stats.json
```

## Debugging Tips

1. **Angular DevTools:** Install Angular DevTools browser extension
2. **Source maps:** Enable in development for debugging
3. **Console logging:** Use structured logging:

```
console.group('Blog Service');
console.log('Loading blogs...');
console.table(blogs);
console.groupEnd();
```

4. **RxJS debugging:** Use `tap` operator:

```
this.http.get('/api/data').pipe(
  tap(data => console.log('Received:', data)),
  catchError(err => {
    console.error('Error:', err);
    return of([]);
  })
)
```

---

## Additional Resources

### Official Documentation

- [Angular Documentation](#)
- [Angular CLI](#)
- [RxJS Documentation](#)
- [TypeScript Documentation](#)

## Recommended Reading

- [Angular Signals Guide](#)
- [Angular SSR Guide](#)
- [Angular Performance Best Practices](#)

## Community

- [Angular GitHub](#)
  - [Angular Blog](#)
  - [Stack Overflow - Angular Tag](#)
- 

# Maintenance and Updates

## Updating Dependencies

Check for outdated dependencies:

```
npm outdated
```

Update Angular and related packages:

```
ng update @angular/core @angular/cli
```

Update all dependencies:

```
npm update
```

# Security Audits

Run security audit:

```
npm audit
```

Fix security vulnerabilities:

```
npm audit fix
```

---

## Support and Contact

For questions or support with this application:

1. Check this documentation first
  2. Review Angular official documentation
  3. Search existing GitHub issues
  4. Contact the development team
- 

## Changelog

### Version 1.0.0 (Current)

#### Features:

- Landing page with feature showcase
- Blog system with list and detail views
- Login and registration pages
- SSR support
- Zoneless change detection
- Signal-based state management

#### Technology Stack:

- Angular 21.0.8
- TypeScript 5.9.3
- Express 5.2.1

- RxJS 7.8.2
- 

# License

[Add your license information here]

---

**Last Updated:** February 2026

**Document Version:** 1.0.0

**Maintained by:** OnMarket Development Team