Development guidelines

- Architecture & Design Patterns
- Services & API Integration
- Routing Structure
- State Management
- Authentication & Security
- Key Components
- Custom Pipes
- Development Workflow
- Build & Deployment
- Testing
- Code Style & Conventions
- Troubleshooting
- Additional resources

Architecture & Design Patterns

Component-Based Architecture

The application follows Angular's component-based architecture where each feature is encapsulated in its own component with clear responsibilities.

Modular Design

- **Feature Modules**: Components are organized by feature
- **Shared Module**: Common components, pipes, and directives
- **Core Module**: Singleton services and app-wide functionality

Design Patterns Used

1. **Dependency Injection**

Services are injected into components via Angular's DI system:

```
constructor(
  private apiService: ApiService,
  private router: Router
) {}
```

2. **Observer Pattern (RxJS)**

Used extensively for handling asynchronous data:

```
this.apiService.getProducts().subscribe(
  data => this.products = data,
  error => this.handleError(error)
);
```

3. **Guard Pattern**

Route guards protect authenticated routes:

```
{ path: 'cart', canActivate: [AuthGuard], component: CartComponent }
```

4. **Interceptor Pattern**

HTTP interceptors handle cross-cutting concerns:

- Authentication token injection
- Error handling
- Loading states

5. **Service Locator Pattern**

Services are registered at the root level and accessed via DI:

```
@Injectable({ providedIn: 'root' })
export class ApiService { }
```

Services & API Integration

ApiService (`api.service.ts`)

The main service handling all API communications.

Base URLs

```
// Production
BASEURL = "https://onmarket.id/";
API_USERSTATE_SERVER = "https://usapi.onindonesia.id/";
API_USERONDELIVERY_SERVER = "https://onapps-api.onindonesia.id/";
API_OSAS_SERVER = "https://api.ondelivery.id/";
API_ONMARKET_SERVER = "https://api.onmarket.id/";
API_ONBOOKING_SERVER = "https://apisat.ondelivery.id/";
API_STORAGE = "https://storage.onindonesia.id/onmarket/";
```

Key Methods

```
// User Authentication
getUser(uname: string, pwd: string): Observable<any>
verify(token: string): Observable<any>

// Products
getProductList(payload: ProductListPayload): Observable<any>
getProductDetail(id: string): Observable<any>

// Cart Management
addToCart(payload: AddCart): Observable<any>
getCart(): Observable<any>
updateCart(cartId: string, quantity: number): Observable<any>
deleteCart(cartId: string): Observable<any>

// Orders
createOrder(payload: any): Observable<any>
getOrders(): Observable<any>
```

SocketsService (`sockets.service.ts`)

Handles real-time WebSocket connections using Socket.io.

```
// Connect to WebSocket
connect(): void

// Disconnect
disconnect(): void

// Emit events
emitEvent(eventName: string, data?: any): void

// Listen to events
on(eventName: string): Observable<any>
```

AffiliateService (`services/affiliate.service.ts`)

Manages affiliate-specific functionality.

LoggerService (`logger.service.ts`)

Handles application logging with configurable log levels.

Routing Structure

Main Routes

The application uses Angular Router with the following main routes:

```
// Home & Product Routes
{ path: "", component: HomePageComponent }
{ path: "search", component: SearchResultComponent }
{ path: "product", component: ProductDetailsComponent }
{ path: "category", component: CategoryProductComponent }

// Authentication Routes
{ path: "login", component: LoginComponent }
{ path: "signup", component: SignupComponent }
{ path: "signup/otp", component: VerifyPhoneOtpComponent }
{ path: "reset", component: ResetComponent }

// Shopping Routes (Protected)
{ path: "cart", canActivate: [AuthGuard], component: CartComponent }
{ path: "checkout", canActivate: [AuthGuard], component: CheckoutComponent }
{ path: "checkout-finish", canActivate: [AuthGuard], component: CheckoutConfirmationComponent }

// User Profile Routes (Protected)
{
  path: "user",
  canActivate: [AuthGuard],
  component: ProfileComponent,
  children: [
    { path: "profile/detail-profile", component: DetailProfileComponent },
    { path: "profile/address", component: AddressComponent },
    { path: "order", component: OrderComponent },
    { path: "change-password", component: ChangePasswordsComponent },
    { path: "chats", component: ChatComponent },
    { path: "favorite", component: FavoriteComponent },
    { path: "voucher", component: VoucherListComponent },
```

```
    { path: "voucher/history", component: VoucherHistoryComponent }
  ]
}

// Store Routes
{
  path: "store",
  component: PersonalStoreComponent,
  children: [
    { path: "store-profile", component: StoreProfileComponent },
    { path: "store-review", component: ReviewComponent },
    { path: "store-product", component: ProductComponent }
  ]
}

// Affiliate Routes (Protected)
{
  path: "affiliate",
  canActivate: [AuthGuard],
  component: AffiliateComponent,
  children: [
    { path: "register", component: RegisterComponent },
    {
      path: "",
      component: DashboardComponent,
      children: [
        { path: "dashboard", component: HomeComponent },
        { path: "commision", component: AffiliateWithdrawComponent },
        { path: "collection", component: CollectionComponent },
        { path: "payment-setting", component: PaymentSettingComponent }
      ]
    }
  ]
}

// Additional Routes
{ path: "about-us", component: AboutUsComponent }
{ path: "help", component: HelpComponent }
{ path: "faq", component: FaqTwoComponent }
```

```
{ path: "contact", component: ContactPageComponent }


// 404 Route
{ path: "**", redirectTo: "/not-found" }
{ path: "/not-found", component: PageNotFoundComponent }
```

Route Guards

AuthGuard (`auth.guard.ts`) protects routes that require authentication:

- Checks for JWT token in localStorage
- Verifies token with API
- Redirects to home page if not authenticated

State Management

BehaviorSubjects

The application uses RxJS BehaviorSubjects for state management:

ApiService State

```
isUpdateCart = new BehaviorSubject(true);
deletedFromCart = new BehaviorSubject(true);
login = new BehaviorSubject(true);
```

SocketsService State

```
private chatOpenSubject = new BehaviorSubject<boolean>(false);
chatOpen$ = this.chatOpenSubject.asObservable();
```

Local Storage

Persistent data is stored in browser localStorage:

- **jwt** - Authentication token
- **user** - User information
- **cart** - Shopping cart data (backup)

Usage Pattern

```
// Subscribe to state changes
this.apiService.isUpdateCart.subscribe(
  shouldUpdate => {
    if (shouldUpdate) {
      this.loadCart();
    }
  }
);
```

```
// Update state
this.apiService.isUpdateCart.next(true);
```

Authentication & Security

JWT Authentication

The application uses JWT (JSON Web Token) for authentication:

1. **Login Flow**:

```
// User logs in
this.apiService.getUser(username, password).subscribe(
  response => {
    // Store JWT token
    localStorage.setItem('jwt', response.token);
    // Navigate to home
    this.router.navigate(['/']);
  }
);
```

2. **Token Storage**: JWT tokens are stored in localStorage with key `'jwt'`
3. **Token Injection**: The `AuthInterceptor` automatically adds the token to all requests:

```
const token = localStorage.getItem("jwt")
  ? `Bearer ${localStorage.getItem("jwt")}`
  : "";

const authReq = req.clone({
  headers: req.headers.set("authorization", token)
});
```

4. **Token Verification**: AuthGuard verifies tokens before allowing access to protected routes

HTTP Interceptor (`auth.interceptor.ts`)

Handles authentication and error responses:

Features:

- Automatically adds JWT token to request headers

- Handles 401/403 unauthorized responses
- Redirects to login on session expiration
- Shows user-friendly error messages
- Clears localStorage on auth failure

Error Handling:

```
private handleAuthError(err: HttpErrorResponse): Observable<any> {
  if (err.status === 401 || err.status === 403) {
    // Show error notification
    this.Toast.fire({
      title: "Sesi berakhir, mohon login kembali",
      icon: "error"
    });

    // Redirect to login
    this.router.navigate(["/login"]);

    // Clear token
    localStorage.removeItem("jwt");
  }
  return throwError(err);
}
```

Security Best Practices

1. **Route Guards**: Protect sensitive routes with AuthGuard
2. **Token Expiration**: Tokens are verified on each protected route access
3. **Secure Communication**: All API calls use HTTPS
4. **XSS Protection**: Angular's built-in sanitization
5. **CSRF Protection**: Handled by backend API

Key Components

1. Homepage Component

Path: `src/app/components/homepage/`

Main landing page featuring:

- Banner slider
- Product categories
- Featured products
- Promotions
- Flash sales

2. Product Details Component

Path: `src/app/components/product-details/`

Displays detailed product information:

- Product images gallery
- Price and stock information
- Product description
- Seller information
- Reviews and ratings
- Add to cart functionality

3. Cart Component

Path: `src/app/components/cart/`

Shopping cart management:

- List of cart items
- Quantity adjustment
- Item removal
- Price calculation
- Checkout navigation

4. Checkout Component

Path: `src/app/components/checkout/`

Order completion process:

- Shipping address selection
- Payment method selection
- Order summary
- Order confirmation

5. Profile Component

Path: `src/app/components/accounts/profile/`

User profile management with nested routes:

- Personal information
- Address book
- Order history
- Password change
- Favorites/wishlist
- Vouchers

6. Affiliate Dashboard

Path: `src/app/components/affiliate/dashboard/`

Affiliate program features:

- Earnings dashboard with charts
- Commission history
- Withdrawal requests
- Collection management
- Affiliate links generation
- Payment settings

7. Search Result Component

Path: `src/app/components/search-result/`

Product search functionality:

- Search input
- Filters (category, price, rating)
- Sort options
- Infinite scroll
- Result grid/list view

8. Personal Store Component

Path: `src/app/components/personal-store/`

Store management:

- Store profile
- Product listing
- Store reviews
- Store statistics

Custom Pipes

1. PhoneNumberPipe

File: `src/app/pipes/phone-number.pipe.ts`

Formats phone numbers according to Indonesian format.

Usage:

```
{{ phoneNumber | phoneNumber }}
```

2. ShortNumberPipe

File: `src/app/pipes/short-number-pipe.pipe.ts`

Converts large numbers to shortened format (e.g., 1000 → 1K).

Usage:

```
{{ largeNumber | shortNumber }}
```

3. TimeAgoPipe

File: `src/app/pipes/time-ago.pipe.ts`

Converts timestamps to relative time (e.g., "2 hours ago").

Usage:

```
{{ timestamp | timeAgo }}
```

4. TransformCityPipe

File: `src/app/pipes/transform-city.pipe.ts`

Formats city names according to application standards.

Usage:

```
{{ cityName | transformCity }}
```

5. SafePipe

File: `src/app/safe.pipe.ts`

Bypasses Angular's security for trusted content (URLs, HTML, etc.).

Usage:

```
<iframe [src]="url | safe:'resourceUrl'"></iframe>
```

6. NumberPipe

File: `src/app/number.pipe.ts`

Custom number formatting for Indonesian locale.

Usage:

```
{{ price | number }}
```

Development Workflow

Starting Development Server

```
# For Unix/Linux/Mac
npm start

# For Windows
npm run startWindows
```

The app will automatically reload when you make changes to source files.

Code Scaffolding

Generate new components, services, etc. using Angular CLI:

```
# Generate a new component
ng generate component components/new-component

# Generate a new service
ng generate service services/new-service

# Generate a new module
ng generate module modules/new-module

# Generate a new pipe
ng generate pipe pipes/new-pipe

# Generate a new directive
ng generate directive directives/new-directive
```

Linting

Check code quality and style:

```
npm run lint
```

Building for Production

```
npm run build
```

Build artifacts will be stored in the `dist/` directory.

Build & Deployment

Development Build

```
npm run build
```

Production Build

```
ng build --configuration production
```

Build Configuration

Configured in `angular.json`:

```
{
  "configurations": {
    "production": {
      "fileReplacements": [{
        "replace": "src/environments/environment.ts",
        "with": "src/environments/environment.prod.ts"
      }],
      "optimization": true,
      "outputHashing": "all",
      "sourceMap": false,
      "namedChunks": false,
      "extractLicenses": true,
      "vendorChunk": false,
      "buildOptimizer": true,
      "budgets": [...]
    }
  }
}
```

Environment Variables

Update environment files before building:

Development (`src/environments/environment.ts`):

```
export const environment = {
  production: false,
  enableConsoleLogs: true
};
```

Production (`src/environments/environment.prod.ts`):

```
export const environment = {
  production: true,
  enableConsoleLogs: false
};
```

Deployment Options

1. Static Hosting (Recommended)

Deploy to static hosting services:

- **Firebase Hosting**
- **Netlify**
- **Vercel**
- **AWS S3 + CloudFront**
- **GitHub Pages**

2. Web Server

Deploy to traditional web servers:

- **Apache**
- **Nginx**
- **IIS**

Important: Configure server to redirect all routes to `index.html` for Angular routing to work.

Nginx Configuration Example:

```
server {
  listen 80;
  server_name onmarket.id;
  root /var/www/onmarket/dist/sapp;
  index index.html;

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

Testing

Unit Tests

Run unit tests via Karma:

```
npm test
```

Tests are written using Jasmine and located alongside components:

- `*.spec.ts` files

End-to-End Tests

Run e2e tests via Protractor:

```
npm run e2e
```

E2E tests are located in the `e2e/` directory.

Test Configuration

- **Karma**: `karma.conf.js`
- **Protractor**: `e2e/protractor.conf.js`
- **TypeScript**: `tsconfig.spec.json`

Writing Tests

Example component test:

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

  beforeEach(async () => {
    await TestBed.configureTestingModule({
```

```
      declarations: [ CartComponent ]
    }).compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(CartComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

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

  it('should load cart items', () => {
    component.loadCart();
    expect(component.cartItems.length).toBeGreaterThan(0);
  });
});
```

Code Style & Conventions

TypeScript Style Guide

Follow Angular's TypeScript style guide:

- Use 2 spaces for indentation
- Use single quotes for strings
- Use camelCase for variables and functions
- Use PascalCase for classes and interfaces
- Use UPPER_CASE for constants

Component Structure

```
@Component({
  selector: 'app-example',
  templateUrl: './example.component.html',
  styleUrls: ['./example.component.css']
})
export class ExampleComponent implements OnInit {
  // 1. Public properties
  public title: string;

  // 2. Private properties
  private data: any[];

  // 3. Constructor with DI
  constructor(
    private apiService: ApiService,
    private router: Router
  ) {}

  // 4. Lifecycle hooks
  ngOnInit(): void {
    this.loadData();
  }
```

```
  // 5. Public methods
  loadData(): void {
    // Implementation
  }

  // 6. Private methods
  private processData(data: any): void {
    // Implementation
  }
}
```

Naming Conventions

- **Components**: `feature-name.component.ts`
- **Services**: `feature-name.service.ts`
- **Pipes**: `pipe-name.pipe.ts`
- **Guards**: `guard-name.guard.ts`
- **Interfaces**: `interface-name.ts`
- **Modules**: `module-name.module.ts`

Template Guidelines

```
<!-- Use semantic HTML -->
<header>
  <nav>
    <!-- Navigation -->
  </nav>
</header>

<!-- Use Angular directives -->
<div *ngIf="isLoading">Loading...</div>
<ul>
  <li *ngFor="let item of items">{{ item.name }}</li>
</ul>

<!-- Use event binding -->
<button (click)="handleClick()">Click Me</button>
```

```
<!-- Use property binding -->
<img [src]="imageUrl" [alt]="imageAlt">

<!-- Use two-way binding -->
<input [(ngModel)]="searchTerm">
```

SCSS/CSS Guidelines

- Use component-scoped styles
- Follow BEM naming convention for complex components
- Use Angular Material theming
- Avoid `!important` unless absolutely necessary

Troubleshooting

Common Issues and Solutions

1. OpenSSL Error on npm start

Error: `Error: error:0308010C:digital envelope routines::unsupported`

Solution: The project already includes the fix in package.json:

```
NODE_OPTIONS='--openssl-legacy-provider' ng serve
```

2. Module Not Found Errors

Solution: Clear cache and reinstall:

```
rm -rf node_modules package-lock.json
npm cache clean --force
npm install
```

3. Port Already in Use

Error: `Port 4200 is already in use`

Solution: Kill the process or use a different port:

```
# Kill process on port 4200 (Unix/Mac)
lsof -ti:4200 | xargs kill -9

# Or use different port
ng serve --port 4300
```

4. CORS Errors

Issue: API requests blocked by CORS policy

Solution:

- Configure backend to allow frontend origin

- Use proxy configuration during development:

Create `proxy.conf.json`:

```
{
  "/api": {
    "target": "https://api.onmarket.id",
    "secure": true,
    "changeOrigin": true
  }
}
```

Update `angular.json`:

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

5. JWT Token Expired

Issue: Session expired errors

Solution:

- Token verification happens on each protected route
- User is automatically redirected to login
- No manual intervention needed

6. WebSocket Connection Issues

Issue: Socket.io connection fails

Solution:

- Check if WebSocket server is running
- Verify JWT token is valid
- Check network connectivity
- Verify API_ONMARKET_SERVER URL

7. Angular Material Theme Not Applied

Solution: Ensure `custom-theme.scss` is imported in `angular.json`:

```
"styles": [
  "src/custom-theme.scss",
  "src/styles.scss"
]
```

8. Build Memory Issues

Error: `JavaScript heap out of memory`

Solution: Increase Node.js memory limit:

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

Additional resources

Official Documentation

- Angular Documentation
- Angular Material
- RxJS
- TypeScript

Key Dependencies Documentation

- Socket.io Client
- ApexCharts
- SweetAlert2
- Google Maps JavaScript API

Learning Resources

- Angular Tutorial
- Angular University
- RxJS Marbles

Community

- Angular GitHub
- Stack Overflow - Angular