OnMart - E-Commerce Mobile Application Documentation

Table of Contents

Project Overview

OnMart is a full-featured React Native e-commerce mobile application that provides a comprehensive shopping experience for both customers and affiliate marketers. The app supports iOS and Android platforms and features a modern, responsive UI built with TypeScript.

Key Capabilities

- **Shopping Experience**: Browse products, search, filter by categories, add to cart, and checkout
- **User Management**: Authentication, profile management, address management
- **Order Management**: Track orders, view order history, manage payments
- **Social Features**: Chat with sellers, product ratings and reviews
- **Affiliate Program**: Complete affiliate marketing system with dashboard, collection management, and payment tracking
- **Wishlist**: Save favorite products for later
- **Vouchers**: Apply discount vouchers and track voucher history
- **Store Management**: Browse stores and view store details
- **Real-time Features**: Socket.io integration for real-time updates

---

Features

Customer Features

1. **Product Discovery**
   - Homepage with featured products and categories
   - Product search with autocomplete
   - Category-based browsing
   - Product filtering and sorting
   - Store browsing
2. **Product Details**
   - High-quality product images with zoom
   - Detailed product information
   - Product variants (size, color, etc.)
   - Product reviews and ratings
   - Store information
   - Related products
3. **Shopping Cart**

- Add/remove products
- Update quantities
- Apply vouchers
- View total price with discounts

4. **Checkout Process**
   - Multiple address support
   - Shipping method selection
   - Payment method integration
   - Order summary
5. **Order Management**
   - Order tracking
   - Order history
   - Order status updates
   - Payment status
6. **User Account**
   - Profile management
   - Address book
   - Wishlist
   - Notification preferences
   - Account deletion
7. **Communication**
   - Chat with sellers
   - Real-time messaging
   - Order inquiries

Affiliate Features

1. **Affiliate Dashboard**
   - Earnings overview
   - Performance metrics
   - Commission tracking
2. **Collection Management**
   - Create product collections
   - Manage collection items
   - Custom collection banners
3. **Link Generation**
   - Generate affiliate links
   - Track link performance
4. **Payment Management**
   - Withdrawal requests
   - Payment history
   - Tax information

Technology Stack

Core Technologies

- **React Native**: 0.75.2
- **React**: 18.3.1
- **TypeScript**: 5.0.4
- **Node.js**: >= 18

Key Libraries

State Management

- **Redux Toolkit**: @reduxjs/toolkit ^2.2.7
- **React Redux**: ^9.1.2
- **Redux Persist**: ^6.0.0

Navigation

- **React Navigation**: ^6.1.18
  - Native Stack Navigator
  - Bottom Tabs Navigator
  - Drawer Navigator

API & Data

- **Apollo Client**: ^3.5.6 (GraphQL)
- **Axios**: ^1.7.7 (REST API)
- **Socket.io Client**: ^4.8.1 (Real-time)

UI Components

- **React Native Reanimated**: ^3.15.1
- **React Native Gesture Handler**: ^2.18.1
- **React Native SVG**: ^15.9.0
- **React Native Vector Icons**: ^10.1.0
- **React Native Maps**: 1.20.0
- **Gorhom Bottom Sheet**: ^5

Media & Camera

- **React Native Vision Camera**: ^4.5.3
- **React Native Image Picker**: ^7.1.2
- **React Native Image Resizer**: ^3.0.11
- **React Native Video**: ^6.7.0

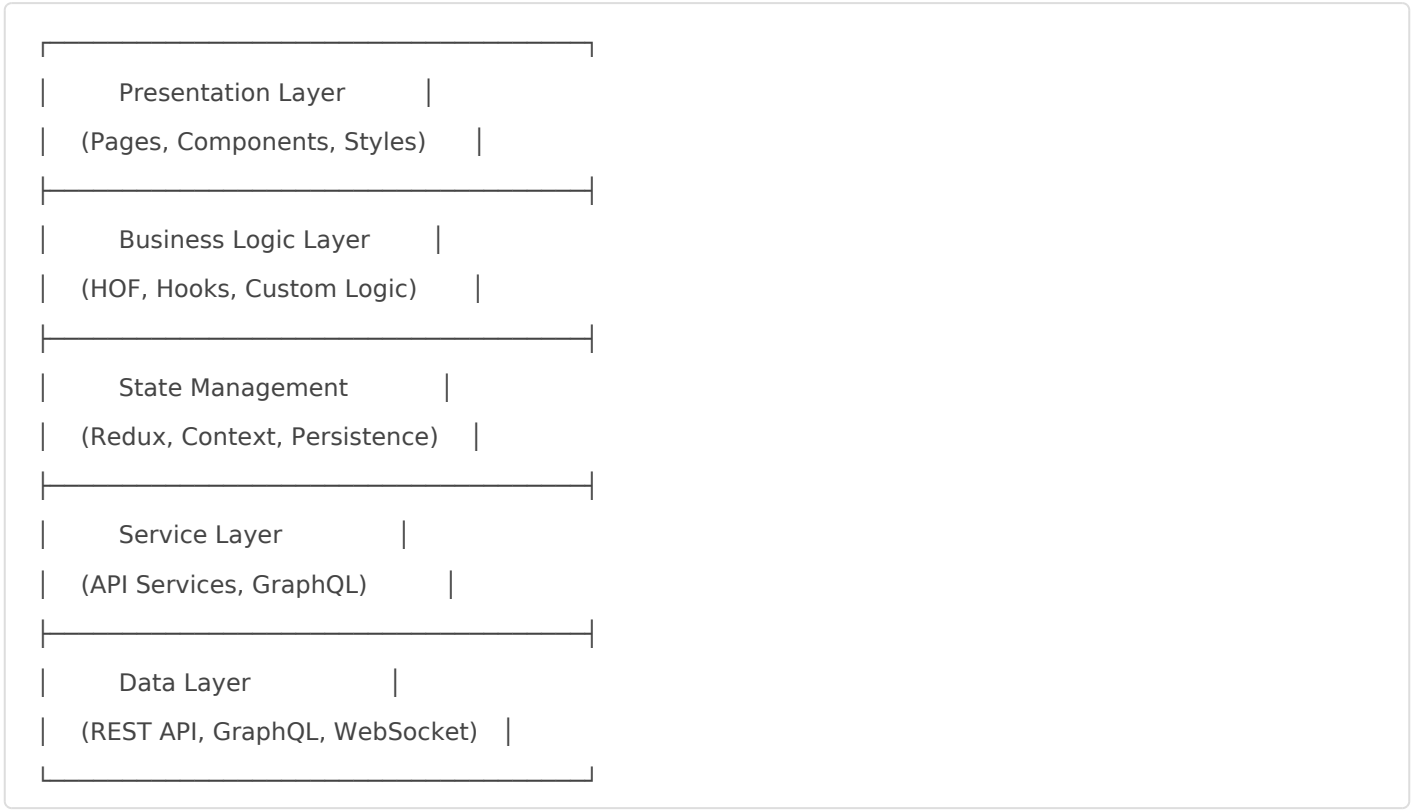
Utilities

- **Moment.js**: ^2.30.1
- **i18next**: ^23.14.0
- **React Native Toast Notifications**: ^3.4.0
- **React Native Share**: ^12.0.3

Project Architecture

OnMart follows a modular architecture with clear separation of concerns:

Architecture Layers



Design Patterns

1. **Atomic Design**: Component library organized as Atoms → Molecules → Organisms
2. **Higher-Order Functions (HOF)**: Business logic separated from UI components
3. **Service Layer Pattern**: Centralized API calls in service modules
4. **Container/Presentational Pattern**: Separation of data handling and UI rendering

---

Getting Started

Prerequisites

Before you begin, ensure you have the following installed:

1. **Node.js** (>= 18.x)
2. **Yarn** (3.6.4 - managed via packageManager)
3. **React Native CLI**
4. **Xcode** (for iOS development, macOS only)
5. **Android Studio** (for Android development)
6. **CocoaPods** (for iOS dependencies)

Environment Setup

Follow the React Native Environment Setup guide for your operating system.

Installation

1. **Clone the repository**

```
git clone https://github.com/ondeliveroper/OnMart.git
cd OnMart
```

2. **Install dependencies**

```
yarn install
```

3. **Install iOS dependencies** (macOS only)

```
cd ios
pod install
cd ..
```

4. **Configure environment**

```
# For development
yarn dev

# For production
yarn prod
```

This copies the appropriate `.env.dev` or `.env.prod` file to `.env`

Running the Application

Development Mode

1. **Start Metro bundler**

```
yarn start
```

2. **Run on iOS** (in a new terminal)

```
yarn ios
```

3. **Run on Android** (in a new terminal)

```
yarn android
```

Production Build

Android

```
yarn android:build
```

Output: `android/app/build/outputs/apk/release/app-release.apk`

iOS Build through Xcode or use Fastlane (requires configuration)

Project Structure

```
OnMart/
├── android/                # Android native code
```

```
├── ios/                    # iOS native code
├── src/                    # Application source code
│   ├── api/                # API integration
│   │   ├── services/       # Service modules
│   │   ├── axios-instance.ts  # Axios configuration
│   │   ├── axios-request.ts   # Request wrapper
│   │   └── endpoints.ts       # API endpoints
│   ├── assets/             # Static assets (images, icons, fonts)
│   ├── components/         # Reusable components
│   │   ├── atoms/          # Basic UI elements
│   │   ├── molecules/      # Composed components
│   │   └── organisms/      # Complex components
│   ├── contexts/           # React Context providers
│   │   ├── theme.tsx       # Theme context
│   │   └── product-card-bottom-sheet/ # Bottom sheet context
│   ├── graphql/            # GraphQL operations
│   │   ├── queries/        # GraphQL queries
│   │   └── mutations/      # GraphQL mutations
│   ├── hooks/              # Custom React hooks
│   ├── i18n/               # Internationalization
│   │   ├── locales/        # Translation files
│   │   └── index.ts        # i18n configuration
│   ├── navigation/         # Navigation configuration
│   │   ├── app-navigator.tsx # Main navigator
│   │   ├── root-navigation.tsx # Navigation root
│   │   └── components/     # Navigation components
│   ├── pages/              # Application screens
│   │   ├── home/
│   │   ├── product-detail/
│   │   ├── cart/
│   │   ├── checkout/
│   │   ├── profile/
│   │   ├── affiliate-*/    # Affiliate pages
│   │   └── ...
│   ├── store/              # Redux store
│   │   ├── actions/        # Action creators
│   │   ├── reducers/       # Reducers
│   │   ├── root-reducer.ts   # Root reducer
│   │   └── index.ts        # Store configuration
```

```
│   ├── types/              # TypeScript type definitions
│   └── utils/              # Utility functions
├── __tests__/              # Test files
├── patches/                # Package patches
├── .env.dev                # Development environment variables
├── .env.prod               # Production environment variables
├── App.tsx                 # Application entry point
├── index.js                # React Native entry point
├── package.json            # Dependencies and scripts
└── tsconfig.json           # TypeScript configuration
```

Key Directories Explained

/src/api/services/

Contains service modules for API integration:

- `auth-service.ts` - Authentication endpoints
- `product-service.ts` - Product operations
- `cart-service.ts` - Shopping cart operations
- `checkout-service.ts` - Checkout process
- `order-service.ts` - Order management
- `profile-services.ts` - User profile operations
- `wishlist-service.ts` - Wishlist operations
- `affiliate-service.ts` - Affiliate program operations
- `chat-service.ts` - Chat functionality

/src/components/

Follows Atomic Design methodology:

- **Atoms**: Button, Card, Container, TextInput, DatePicker, Dropdown, SkeletonLoader
- **Molecules**: BackButton, CartButton, Badge, Rating, SearchButton, PageInfo
- **Organisms**: Complex multi-component structures

/src/pages/

Each page represents a screen in the app:

- Structure: `page-name/page-name.tsx` (UI) + `page-name-hof.ts` (logic)
- Components specific to a page are in `page-name/components/`
- Styles are in `page-name/style.ts`

/src/store/

Redux store configuration:

- **actions**: Action creators and action types
- **reducers**: State reducers (login, user, profile, counter, affiliate)
- Integrated with Redux Persist for state persistence

Development Guidelines

Code Organization

1. **Page Structure**

```
page-name/
├── page-name.tsx          # UI component
├── page-name-hof.ts       # Business logic (Higher-Order Function)
├── style.ts               # Styles
└── components/            # Page-specific components
    ├── component-a.tsx
    └── component-b.tsx
```

2. **Component Structure**

```
import React from 'react';
import { View, Text } from 'react-native';

interface ComponentProps {
  // Props definition
}

const Component: React.FC<ComponentProps> = (props) => {
  // Component logic
  return (
    <View>
      {/* JSX */}
    </View>
  );
};
```

```
export default Component;
```

3. **HOF Pattern**

```
const usePageHOF = () => {
  // State management
  // Business logic
  // API calls

  return {
    datas: {
      // Data to expose
    },
    methods: {
      // Methods to expose
    }
  };
};

export default usePageHOF;
```

Naming Conventions

- **Files**: kebab-case (e.g., `product-detail.tsx`)
- **Components**: PascalCase (e.g., `ProductCard`)
- **Functions/Variables**: camelCase (e.g., `handleSubmit`)
- **Constants**: UPPER_SNAKE_CASE (e.g., `API_BASE_URL`)
- **Types/Interfaces**: PascalCase (e.g., `ProductType`)

TypeScript Guidelines

1. **Always define types for props**

```
interface ButtonProps {
  title: string;
  onPress: () => void;
  disabled?: boolean;
}
```

2. **Use type imports**

```
import type { ProductType } from '../types';
```

3. **Avoid `any` type** - use `unknown` if type is truly unknown

Styling Guidelines

1. **Use theme for consistency**

```
const styles = createStyles(theme);
```

2. **Create reusable style functions**

```
export const createStyles = (theme: Theme) =>
  StyleSheet.create({
    container: {
      flex: 1,
      backgroundColor: theme.colors.background,
    },
  });
```

3. **Responsive design**: Use percentages and flex for layouts

---

API Integration

REST API (Axios)

Configuration

API is configured in `src/api/axios-instance.ts`:

- Base URL from environment variables
- Request/response interceptors
- Authentication token injection

Making API Calls

Service modules in `src/api/services/` provide type-safe API calls:

```
// Example: Product Service
import { getProductDetail } from '../api/services/product-service';

const fetchProduct = async (productId: string) => {
  try {
    const response = await getProductDetail(productId);
    // Handle response
  } catch (error) {
    // Handle error
  }
};
```

Available Services

- **AuthService**: login, signup, logout, password reset
- **ProductService**: list products, product details, search
- **CartService**: add to cart, update cart, remove from cart
- **CheckoutService**: create order, payment processing
- **OrderService**: order history, order details, track order
- **ProfileService**: get/update profile, address management
- **WishlistService**: add/remove from wishlist
- **AffiliateService**: affiliate operations, commission tracking
- **ChatService**: send/receive messages

GraphQL (Apollo Client)

Configuration

Apollo Client is configured in the app with:

- GraphQL endpoint
- Authentication headers
- File upload support (apollo-upload-client)

Queries and Mutations

Located in `src/graphql/`:

```
// Example Query
import { useQuery } from '@apollo/client';
import { GET_PRODUCTS } from '../graphql/queries';
```

```
const { data, loading, error } = useQuery(GET_PRODUCTS, {
  variables: { categoryId: '123' }
});
```

WebSocket (Socket.io)

Real-time features use Socket.io:

- Chat messages
- Order status updates
- Notifications

State Management

Redux Store Structure

```
{
  login: {
    isLoggedIn: boolean,
    token: string,
    // ...
  },
  user: {
    userId: string,
    userInfo: UserType,
    // ...
  },
  profile: {
    // User profile data
  },
  counter: {
    cartCount: number,
    wishlistCount: number,
    notificationCount: number,
  },
```

```
  affiliate: {
    // Affiliate program data
  }
}
```

Using Redux

Accessing State

```
import { useSelector } from 'react-redux';

const MyComponent = () => {
  const isLoggedIn = useSelector((state: RootState) => state.login.isLoggedIn);
  const cartCount = useSelector((state: RootState) => state.counter.cartCount);

  // Component logic
};
```

Dispatching Actions

```
import { useDispatch } from 'react-redux';
import { setUser } from '../store/actions';

const MyComponent = () => {
  const dispatch = useDispatch();

  const handleLogin = (user: UserType) => {
    dispatch(setUser(user));
  };

  // Component logic
};
```

Redux Persist

State is automatically persisted to AsyncStorage. Configured in `src/store/index.ts`.

Context API

Used for:

- **ThemeContext**: Theme management
- **PCBottomSheetProvider**: Product card bottom sheet

Navigation

Navigation Structure

```
Root Navigator
├── Auth Stack (if not logged in)
│   ├── OnBoarding
│   ├── Login
│   ├── SignUp
│   └── ForgotPassword
└── Main App (if logged in)
    ├── Bottom Tab Navigator
    │   ├── Home Stack
    │   ├── Wishlist
    │   ├── Cart
    │   └── Profile
    └── Other Stacks
        ├── Product Detail Stack
        ├── Checkout Stack
        ├── Order Stack
        ├── Affiliate Stack
        └── Chat Stack
```

Navigation Types

Define navigation types in `src/types/route.ts`:

```
export type RootStackParamList = {
  Home: undefined;
  ProductDetail: { productId: string };
  // ...
};
```

Navigating Between Screens

```
import { useNavigation } from '@react-navigation/native';

const MyComponent = () => {
  const navigation = useNavigation();

  const goToProduct = (productId: string) => {
    navigation.navigate('ProductDetail', { productId });
  };

  const goBack = () => {
    navigation.goBack();
  };
};
```

Deep Linking

Configure deep linking in `react-native.config.js` and navigation configuration for handling external links.

Component Library

Atomic Design Structure

Atoms (Basic Building Blocks)

Button

```
import { Button } from '../../components';

<Button
  title="Submit"
  onPress={handleSubmit}
  variant="primary"
  disabled={false}
/>
```

Card

```
import { Card } from '../../components';

<Card>
  <Text>Card Content</Text>
</Card>
```

TextInput

```
import { TextInput } from '../../components';

<TextInput
  placeholder="Enter text"
  value={text}
  onChangeText={setText}
  leftIcon={<Icon />}
  rightIcon={<Icon />}
/>
```

ProductCard

```
import { ProductCard } from '../../components';

<ProductCard
  product={productData}
  onPress={handleProductPress}
  onAddToCart={handleAddToCart}
/>
```

SkeletonLoader

```
import { SkeletonLoader } from '../../components';

<SkeletonLoader width={100} height={20} borderRadius={4} />
```

Molecules (Composite Components)

BackButton

```
import { BackButton } from '../../components';

<BackButton onBackPress={handleBack} />
```

CartButton

```
import { CartButton } from '../../components';

<CartButton
  count={cartCount}
  onPress={handleCartPress}
/>
```

Rating

```
import { Rating } from '../../components';

<Rating
  rating={4.5}
  count={120}
  size="small"
/>
```

SearchButton

```
import { SearchButton } from '../../components';

<SearchButton onPress={handleSearch} />
```

Creating Custom Components

Follow the atomic design principle:

1. Determine the component level (atom/molecule/organism)
2. Create in appropriate directory
3. Export from `components/index.ts`
4. Add TypeScript types
5. Use theme for styling

Styling and Theming

Theme Structure

Theme is defined in `src/contexts/theme.tsx`:

```
{
  colors: {
    primary: string,
    secondary: string,
    background: string,
    text: string,
    error: string,
    success: string,
    // ...
  },
  spacing: {
    small: number,
    medium: number,
    large: number,
  },
  margin: {
    small: number,
    medium: number,
    large: number,
  },
  padding: {
    small: number,
    medium: number,
    large: number,
  },
```

```
  typography: {
    fontSize: {...},
    fontWeight: {...},
  },
  // ...
}
```

Using Theme

```
import { useTheme } from '../../contexts';

const MyComponent = () => {
  const { theme } = useTheme();

  const styles = createStyles(theme);

  return (
    <View style={styles.container}>
      {/* Component content */}
    </View>
  );
};

const createStyles = (theme: Theme) => StyleSheet.create({
  container: {
    backgroundColor: theme.colors.background,
    padding: theme.padding.medium,
  },
  text: {
    color: theme.colors.text,
    fontSize: theme.typography.fontSize.medium,
  },
});
```

Responsive Design

Use dimensions and flex for responsive layouts:

```
import { Dimensions } from 'react-native';

const { width, height } = Dimensions.get('window');

const styles = StyleSheet.create({
  container: {
    width: width * 0.9,
  },
});
```

Internationalization (i18n)

Configuration

i18n is configured using `i18next` and `react-i18next`.

Configuration: `src/i18n/index.ts` Translations: `src/i18n/locales/en.json`

Adding Translations

1. Add keys to `src/i18n/locales/en.json`:

   ```
   {
     "welcome": "Welcome to OnMart",
     "login": {
       "title": "Login",
       "email": "Email Address",
       "password": "Password"
     }
   }
   ```

2. Use in components:

   ```
   import { useTranslation } from 'react-i18next';

   const MyComponent = () => {
     const { t } = useTranslation();
   ```

```
  return (
    <Text>{t('welcome')}</Text>
    <Text>{t('login.title')}</Text>
  );
};
```

Adding New Languages

1. Create new translation file: `src/i18n/locales/[lang].json`
2. Import in `src/i18n/index.ts`
3. Add to resources configuration

Environment Configuration

Environment Variables

Three environment files:

- `.env` - Active environment (copied from .env.dev or .env.prod)
- `.env.dev` - Development configuration
- `.env.prod` - Production configuration

Available Variables

```
API_AUTH_URL=https://auth-api.example.com
API_BASE_URL=https://api.example.com
URL=https://app.example.com
MAP_KEY=YOUR_GOOGLE_MAPS_API_KEY
COLLECTION_BANNER=https://storage.example.com/default-banner.png
```

Using Environment Variables

```
import { API_BASE_URL, MAP_KEY } from '@env';

const apiUrl = API_BASE_URL;
const mapKey = MAP_KEY;
```

Switching Environments

```
# Switch to development
yarn dev

# Switch to production
yarn prod
```

Testing

Running Tests

```
# Run all tests
yarn test

# Run tests in watch mode
yarn test --watch

# Run tests with coverage
yarn test --coverage
```

Test Structure

Tests are located in `__tests__/` directory.

Writing Tests

```
import React from 'react';
import { render } from '@testing-library/react-native';
import MyComponent from '../MyComponent';

describe('MyComponent', () => {
  it('renders correctly', () => {
    const { getByText } = render(<MyComponent />);
    expect(getByText('Hello')).toBeTruthy();
  });
});
```

Build and Deployment

Android Build

Debug Build

```
yarn android
```

Release Build

```
yarn android:build
```

Output: `android/app/build/outputs/apk/release/app-release.apk`

Signing Configuration

Configure signing in `android/app/build.gradle`:

```
signingConfigs {
    release {
        storeFile file('your-keystore.keystore')
        storePassword 'your-store-password'
        keyAlias 'your-key-alias'
        keyPassword 'your-key-password'
    }
```

```
}
```

iOS Build

Debug Build

```
yarn ios
```

Release Build

1. Open `ios/OnMart.xcworkspace` in Xcode
2. Select "Product" → "Archive"
3. Follow distribution steps

Code Signing

- **iOS**: Configure signing in Xcode
- **Android**: Configure keystore

App Distribution

- **iOS**: App Store Connect / TestFlight
- **Android**: Google Play Console / Firebase App Distribution

---

Common Issues and Troubleshooting

Metro Bundler Issues

Problem: Metro bundler cache issues

Solution:

```
yarn start --reset-cache
```

Pod Install Failures (iOS)

Problem: CocoaPods installation fails

Solution:

```
cd ios
pod deintegrate
pod install
cd ..
```

Android Build Failures

Problem: Gradle build fails

Solution:

```
cd android
./gradlew clean
cd ..
yarn android
```

Package Conflicts

Problem: Package version conflicts

Solution:

```
rm -rf node_modules
rm yarn.lock
yarn install
```

Native Module Linking Issues

Problem: Native modules not found

Solution:

```
# Reinstall dependencies
yarn install

# iOS
cd ios && pod install && cd ..

# Android - usually auto-links
```

Common Error Messages

1. **"Unable to resolve module"**
   - Clear Metro cache: `yarn start --reset-cache`
   - Reinstall dependencies
2. **"Command failed: gradlew.bat"**
   - Check Android SDK installation
   - Update Gradle version
3. **"CocoaPods could not find compatible versions"**
   - Update CocoaPods: `pod repo update`
   - Update pod dependencies: `cd ios && pod update && cd ..`

Best Practices

Performance Optimization

1. **Use React.memo for pure components**

```
export default React.memo(MyComponent);
```

2. **Use useCallback for functions passed as props**

```
const handlePress = useCallback(() => {
  // Handler logic
}, [dependencies]);
```

3. **Use useMemo for expensive calculations**

```
const expensiveValue = useMemo(() => {
  return computeExpensiveValue(data);
```

```
}, [data]);
```

4. **Optimize FlatList with proper keys and props**

```
<FlatList
  data={items}
  keyExtractor={(item) => item.id}
  removeClippedSubviews={true}
  maxToRenderPerBatch={10}
  windowSize={10}
/>
```

5. **Use SkeletonLoader for loading states**

Security Best Practices

1. **Never commit sensitive data**
   - Use environment variables
   - Add sensitive files to `.gitignore`
2. **Validate user input**
3. **Use HTTPS for all API calls**
4. **Implement proper authentication**
5. **Store tokens securely** (using AsyncStorage with encryption)

Code Quality

1. **Run linter before committing**

```
yarn lint
```

2. **Follow TypeScript best practices**
3. **Write meaningful commit messages**
4. **Comment complex logic**
5. **Keep functions small and focused**

Git Workflow

1. **Create feature branches**

```
git checkout -b feature/my-feature
```

2. **Commit frequently with clear messages**

```
git commit -m "Add product search functionality"
```

3. **Pull before pushing**

```
git pull origin main
git push origin feature/my-feature
```

Component Development

1. **Start with the simplest implementation**
2. **Make components reusable**
3. **Use TypeScript for type safety**
4. **Separate concerns (UI vs logic)**
5. **Test edge cases**

Additional Resources

React Native Documentation

- React Native Docs
- React Navigation
- Redux Toolkit

Learning Resources

- React Native Express
- TypeScript Handbook

Community

- React Native Community
- Stack Overflow - React Native

Conclusion

This documentation provides a comprehensive guide to understanding, developing, and maintaining the OnMart mobile application. For any questions or issues not covered here, please contact the development team.

Version: 1.0.4 **Last Updated**: February 2026 **Maintained By**: OnDelivery Operations Team

---

Revision #2
Created 5 February 2026 05:02:56 by ondeliveloper
Updated 5 February 2026 05:03:44 by ondeliveloper