Flutter E-Commerce Application - Code Reference

Table of Contents

Project Overview

This is a Flutter-based e-commerce application built with modern architecture patterns and best practices. The app provides a complete shopping experience with product browsing, authentication, cart management, and user profiles.

Key Features:
- User authentication and authorization
- Product catalog with categories
- Shopping cart functionality
- Responsive Material Design UI
- Dark/Light theme support
- RESTful API integration
- State management with BLoC pattern

Architecture

The application follows a **Clean Architecture** pattern with **Feature-Driven Development**:

lib/
■■■ config/ # App configuration
■■■ core/ # Core utilities and services
■■■ features/ # Feature modules
■ ■■■ auth/ # Authentication feature

■ ■■■ products/ # Products feature
■ ■■■ cart/ # Shopping cart feature
■ ■■■ profile/ # User profile feature
■ ■■■ splash/ # Splash screen feature
■■■ main.dart # App entry point

Architecture Layers:
- **Presentation Layer**: UI components, BLoCs, and pages
- **Domain Layer**: Business logic and repositories
- **Data Layer**: Models, repositories, and API services

Dependencies

Core Dependencies

State Management

flutter_bloc: ^8.1.4
equatable: ^2.0.5

Network

dio: ^5.4.1
pretty*dio*logger: ^1.3.1

Local Storage

shared_preferences: ^2.2.2

UI Components

cached*network*image: ^3.3.1
flutter_svg: ^2.0.10+1
shimmer: ^3.0.0

Utils

intl: ^0.19.0
logger: ^2.0.2+1

Development Dependencies

flutter_lints: ^4.0.0
mockito: ^5.4.4
build_runner: ^2.4.8
bloc_test: ^9.1.6

Project Structure

Directory Organization

ecommerce_app/
■■■ android/ # Android platform code
■■■ ios/ # iOS platform code
■■■ lib/ # Main Dart code
■ ■■■ config/ # App configuration
■ ■ ■■■ app_routes.dart
■ ■ ■■■ app_theme.dart
■ ■■■ core/ # Core utilities
■ ■ ■■■ network/
■ ■ ■■■ api_service.dart
■ ■■■ features/ # Feature modules
■ ■ ■■■ auth/
■ ■ ■ ■■■ data/
■ ■ ■ ■ ■■■ repositories/
■ ■ ■ ■ ■■■ auth_repository.dart
■ ■ ■ ■■■ presentation/
■ ■ ■ ■■■ bloc/
■ ■ ■ ■ ■■■ auth_bloc.dart
■ ■ ■ ■■■ pages/
■ ■ ■ ■■■ login_page.dart
■ ■ ■ ■■■ register_page.dart
■ ■ ■■■ products/
■ ■ ■ ■■■ data/
■ ■ ■ ■ ■■■ models/
■ ■ ■ ■ ■ ■■■ product_model.dart
■ ■ ■ ■ ■■■ repositories/
■ ■ ■ ■ ■■■ product_repository.dart
■ ■ ■ ■■■ presentation/
■ ■ ■ ■■■ bloc/
■ ■ ■ ■ ■■■ products_bloc.dart
■ ■ ■ ■■■ pages/
■ ■ ■ ■■■ products_page.dart
■ ■ ■ ■■■ product*detail*page.dart
■ ■ ■■■ cart/
■ ■ ■ ■■■ presentation/
■ ■ ■ ■■■ bloc/
■ ■ ■ ■ ■■■ cart_bloc.dart
■ ■ ■ ■■■ pages/
■ ■ ■ ■■■ cart_page.dart
■ ■ ■■■ profile/

■ ■ ■ ■■■ presentation/
■ ■ ■ ■■■ pages/
■ ■ ■ ■■■ profile_page.dart
■ ■ ■■■ splash/
■ ■ ■■■ presentation/
■ ■ ■■■ pages/
■ ■ ■■■ splash_page.dart
■ ■■■ main.dart
■■■ test/ # Test files
■■■ pubspec.yaml # Dependencies

Core Components

Main Application Entry Point

```
// lib/main.dart
import 'package:flutter/material.dart';
import 'package:flutter*bloc/flutter*bloc.dart';

void main() {
WidgetsFlutterBinding.ensureInitialized();
runApp(const MyApp());
}

class MyApp extends StatelessWidget {
const MyApp({super.key});

@override
Widget build(BuildContext context) {
final apiService = ApiService();
final authRepository = AuthRepository(apiService: apiService);
final productRepository = ProductRepository(apiService: apiService);

return MultiBlocProvider(
providers: [
BlocProvider(
create: (context) => AuthBloc(authRepository: authRepository)
..add(CheckAuthStatus()),
),
BlocProvider(
create: (context) => ProductsBloc(productRepository: productRepository),
),
BlocProvider(create: (context) => CartBloc()),
],
child: MaterialApp(
title: 'E-Commerce App',
theme: AppTheme.lightTheme,
darkTheme: AppTheme.darkTheme,
themeMode: ThemeMode.system,
debugShowCheckedModeBanner: false,
initialRoute: AppRoutes.splash,
onGenerateRoute: AppRoutes.onGenerateRoute,
```

```
),
);
}
}
```

Application Routes

```
// lib/config/app_routes.dart
class AppRoutes {
static const String splash = '/';
static const String login = '/login';
static const String products = '/products';
static const String cart = '/cart';

static Route onGenerateRoute(RouteSettings settings) {
switch (settings.name) {
case splash:
return MaterialPageRoute(builder: (_) => const SplashPage());
case login:
return MaterialPageRoute(builder: (_) => const LoginPage());
case products:
return MaterialPageRoute(builder: (_) => const ProductsPage());
case cart:
return MaterialPageRoute(builder: (_) => const CartPage());
default:
return MaterialPageRoute(
builder: (_) => Scaffold(
body: Center(
child: Text('No route defined for ${settings.name}'),
),
),
);
}
}
}
```

Features

1. Authentication Feature

Location: `lib/features/auth/`

Components:
- `AuthBloc`: Manages authentication state
- `AuthRepository`: Handles authentication logic
- `LoginPage`: User login interface
- `RegisterPage`: User registration interface

Key Events:
- `LoginRequested`: User login attempt

- `LogoutRequested`: User logout
- `CheckAuthStatus`: Verify authentication status

Key States:
- `AuthInitial`: Initial state
- `AuthLoading`: Loading state
- `Authenticated`: User is logged in
- `Unauthenticated`: User is not logged in
- `AuthError`: Authentication error

2. Products Feature

Location: `lib/features/products/`

Components:
- `ProductsBloc`: Manages product state
- `ProductRepository`: Handles product data
- `ProductModel`: Product data model
- `ProductsPage`: Product listing
- `ProductDetailsPage`: Product details

Key Events:
- `LoadProducts`: Load all products
- `LoadProductById`: Load specific product
- `LoadCategories`: Load product categories
- `LoadProductsByCategory`: Load products by category

Key States:
- `ProductsInitial`: Initial state
- `ProductsLoading`: Loading state
- `ProductsLoaded`: Products loaded successfully
- `ProductLoaded`: Single product loaded
- `CategoriesLoaded`: Categories loaded
- `ProductsError`: Error state

3. Cart Feature

Location: `lib/features/cart/`

Components:
- `CartBloc`: Manages cart state
- `CartItem`: Cart item model
- `CartPage`: Shopping cart interface

Key Events:
- `AddToCart`: Add product to cart
- `RemoveFromCart`: Remove product from cart
- `UpdateQuantity`: Update product quantity
- `ClearCart`: Clear entire cart

Key States:
- `CartInitial`: Initial state
- `CartLoading`: Loading state
- `CartLoaded`: Cart loaded with items
- `CartError`: Error state

State Management

The application uses **BLoC (Business Logic Component)** pattern for state management with the `flutter_bloc` package.

BLoC Pattern Structure

```
// Event -> BLoC -> State
abstract class Event extends Equatable {
const Event();
@override
List get props => [];
}

abstract class State extends Equatable {
const State();
@override
List get props => [];
}

class Bloc extends Bloc {
Bloc() : super(InitialState()) {
on(_onEvent);
}

Future _onEvent(Event event, Emitter emit) async {
// Handle event and emit states
}
}
```

BLoC Providers Setup

```
MultiBlocProvider(
providers: [
BlocProvider(create: (context) => AuthBloc(...)),
BlocProvider(create: (context) => ProductsBloc(...)),
BlocProvider(create: (context) => CartBloc()),
],
child: MaterialApp(...),
)
```

Network Layer

API Service

Location: `lib/core/network/api_service.dart`

The API service uses **Dio** for HTTP requests with the following features:

- **Base Configuration:**
- Base URL: `https://dummyjson.com`
- Timeout: 5 seconds connect, 3 seconds receive
- JSON content type headers

- **Interceptors:**
- **PrettyDioLogger**: Request/response logging
- **Auth Interceptor**: Automatic token injection
- **Token Refresh**: Automatic token refresh on 401 errors

- **Methods:**
- `get()`: HTTP GET requests
- `post()`: HTTP POST requests
- `put()`: HTTP PUT requests
- `delete()`: HTTP DELETE requests

```
class ApiService {
static const String baseUrl = 'https://dummyjson.com';
static const String tokenKey = 'authtoken';
final Dio _dio;

ApiService() : _dio = Dio(BaseOptions(
baseUrl: baseUrl,
connectTimeout: const Duration(seconds: 5),
receiveTimeout: const Duration(seconds: 3),
headers: {
'Content-Type': 'application/json',
'Accept': 'application/json',
},
)) {
_setupInterceptors();
}

void _setupInterceptors() {
// Pretty logging
_dio.interceptors.add(PrettyDioLogger(...));

// Auth interceptor
_dio.interceptors.add(InterceptorsWrapper(
onRequest: (options, handler) async {
final token = await _getToken();
if (token != null) {
options.headers['Authorization'] = 'Bearer $token';
}
return handler.next(options);
},
onError: (error, handler) async {
if (error.response?.statusCode == 401) {
await _refreshToken();
// Retry request with new token
}
return handler.reject(error);
},
));
```

```
}
}
```

Data Models

Product Model

Location: `lib/features/products/data/models/product_model.dart`

```
class ProductModel extends Equatable {
final int id;
final String title;
final double price;
final String description;
final String category;
final List images;
final List reviews;

const ProductModel({
required this.id,
required this.title,
required this.price,
required this.description,
required this.category,
required this.images,
required this.reviews,
});

factory ProductModel.fromJson(Map json) {
return ProductModel(
id: json['id'] as int,
title: json['title'] as String,
price: (json['price'] as num).toDouble(),
description: json['description'] as String,
category: json['category'] as String,
images: List.from(json['images'] ?? []),
reviews: (json['reviews'] as List)
.map((review) => Review.fromJson(review))
.toList(),
);
}

Map toJson() {
return {
'id': id,
'title': title,
'price': price,
'description': description,
'category': category,
'image': images,
'reviews': reviews?.map((review) => review?.toJson()).toList(),
};
```

```
}

@override
List get props => [id, title, price, description, category, images, reviews];
}
```

Review Model

```
class Review extends Equatable {
final int rating;
final String? comment;

const Review({
required this.rating,
required this.comment,
});

factory Review.fromJson(Map json) {
return Review(
rating: (json['rating'] as num).toInt(),
comment: json['comment'] as String?,
);
}

Map toJson() {
return {
'rate': rating,
'count': comment,
};
}

@override
List get props => [rating, comment];
}
```

Cart Item Model

```
class CartItem extends Equatable {
final ProductModel product;
final int quantity;

const CartItem({
required this.product,
required this.quantity,
});

double get total => product.price * quantity;

@override
List get props => [product, quantity];
}
```

UI Components

Theme Configuration

Location: `lib/config/app_theme.dart`

The application supports both light and dark themes with Material 3 design:

```
class AppTheme {
static ThemeData get lightTheme {
return ThemeData(
useMaterial3: true,
brightness: Brightness.light,
colorScheme: ColorScheme.fromSeed(
seedColor: Colors.blue,
brightness: Brightness.light,
),
appBarTheme: const AppBarTheme(
centerTitle: true,
elevation: 0,
),
cardTheme: CardTheme(
elevation: 2,
shape: RoundedRectangleBorder(
borderRadius: BorderRadius.circular(8),
),
),
inputDecorationTheme: InputDecorationTheme(
border: OutlineInputBorder(
borderRadius: BorderRadius.circular(8),
),
contentPadding: const EdgeInsets.symmetric(
horizontal: 16,
vertical: 14,
),
),
elevatedButtonTheme: ElevatedButtonThemeData(
style: ElevatedButton.styleFrom(
padding: const EdgeInsets.symmetric(
horizontal: 24,
vertical: 12,
),
shape: RoundedRectangleBorder(
borderRadius: BorderRadius.circular(8),
),
),
),
);
}

static ThemeData get darkTheme {
// Similar configuration with dark brightness
}
}
```

Key UI Features

- **Material 3 Design**: Modern Material Design components
- **Responsive Layout**: Adapts to different screen sizes
- **Theme Support**: Light and dark theme switching
- **Loading States**: Shimmer effects and loading indicators
- **Error Handling**: User-friendly error messages
- **Navigation**: Intuitive navigation between screens

Testing

Test Structure

Location: `test/`

test/
■■■ features/
■ ■■■ cart/
■ ■ ■■■ presentation/
■ ■ ■■■ bloc/
■ ■ ■■■ cart*bloc*test.dart
■ ■■■ products/
■ ■■■ data/
■ ■ ■■■ repositories/
■ ■ ■■■ product*repository*test.dart
■ ■ ■■■ product*repository*test.mocks.dart
■ ■■■ presentation/
■ ■■■ bloc/
■ ■■■ products*bloc*test.dart
■ ■■■ products*bloc*test.mocks.dart
■■■ widget_test.dart

Testing Dependencies

dev_dependencies:
flutter_test:
sdk: flutter
mockito: ^5.4.4
build_runner: ^2.4.8
bloc_test: ^9.1.6

BLoC Testing Example

// test/features/cart/presentation/bloc/cart*bloc*test.dart
import 'package:bloc*test/bloc*test.dart';
import 'package:flutter*test/flutter*test.dart';

```
void main() {
group('CartBloc', () {
late CartBloc cartBloc;

setUp(() {
cartBloc = CartBloc();
});

tearDown(() {
cartBloc.close();
});

test('initial state is CartInitial', () {
expect(cartBloc.state, isA());
});

blocTest(
'emits [CartLoading, CartLoaded] when AddToCart is added',
build: () => cartBloc,
act: (bloc) => bloc.add(AddToCart(product: mockProduct)),
expect: () => [
isA(),
isA(),
],
);
});
}
```

Configuration

Environment Configuration

The application uses the following configuration:

- **API Base URL**: `https://dummyjson.com`
- **Flutter SDK**: `^3.5.0`
- **Target Platforms**: Android, iOS, Web, macOS, Linux, Windows

Build Configuration

Android: `android/app/build.gradle`
- Minimum SDK: 21
- Target SDK: 34
- Compile SDK: 34

iOS: `ios/Runner/Info.plist`
- Deployment Target: 12.0
- Supported orientations: Portrait, Landscape

Development Setup

1. **Install Flutter SDK** (version 3.5.0 or higher)
2. **Install Dependencies**:
flutter pub get
3. **Generate Mock Files** (for testing):
flutter packages pub run build_runner build
4. **Run Tests**:
flutter test
5. **Run Application**:
flutter run

Best Practices

Code Organization

1. **Feature-First Architecture**: Organize code by features rather than layers
2. **Separation of Concerns**: Keep UI, business logic, and data separate
3. **Dependency Injection**: Use constructor injection for dependencies
4. **Immutable Models**: Use `Equatable` for value equality

State Management

1. **Single Source of Truth**: Each feature has its own BLoC
2. **Unidirectional Data Flow**: Events → BLoC → States → UI
3. **Predictable State Changes**: All state changes go through BLoC

Error Handling

1. **Graceful Degradation**: Handle errors without crashing
2. **User-Friendly Messages**: Show meaningful error messages
3. **Retry Mechanisms**: Allow users to retry failed operations

Performance

1. **Lazy Loading**: Load data only when needed
2. **Caching**: Cache network responses and images
3. **Memory Management**: Dispose of resources properly

API Endpoints

The application integrates with the DummyJSON API:

- **Base URL**: `https://dummyjson.com`
- **Authentication**: `POST /auth/login`
- **Products**: `GET /products`
- **Product Details**: `GET /products/{id}`
- **Categories**: `GET /products/categories`
- **Products by Category**: `GET /products/category/{category}`

Deployment

Android Deployment

1. **Build APK**:
flutter build apk --release

2. **Build App Bundle**:
flutter build appbundle --release

iOS Deployment

1. **Build iOS App**:
flutter build ios --release

2. **Archive and Upload** via Xcode

Web Deployment

1. **Build Web App**:
flutter build web --release

2. **Deploy** to web server or hosting platform

Conclusion

This Flutter e-commerce application demonstrates modern mobile development practices with:

- **Clean Architecture** with feature-driven development
- **BLoC Pattern** for state management
- **RESTful API Integration** with proper error handling
- **Material Design** with theme support
- **Comprehensive Testing** strategy
- **Cross-Platform Support** for multiple platforms

The codebase is well-structured, maintainable, and follows Flutter best practices for building scalable applications.

Generated on: ${new Date().toLocaleDateString()}
Flutter Version: 3.5.0
Dart Version: 3.0.0