# Flutter BLoC Base Yapıları Dokümantasyonu

## 1. Genel Bakış

Bu dokümantasyon, Flutter BLoC projesinde kullanılan temel yapıları ve bunların nasıl kullanılacağını açıklar. Bu yapılar, kod tekrarını önlemek, tutarlılık sağlamak ve bakımı kolaylaştırmak için tasarlanmıştır.

## 2. Base State Yapısı

### 2.1 BaseState Sınıfı

abstract class BaseState<T> extends Equatable {  
 final T? \_data;  
 final String? \_message;  
 final bool \_isLoading;  
  
 const BaseState({  
 T? data,  
 String? message,  
 bool isLoading = false,  
 });  
  
 T? get data => \_data;  
 String? get message => \_message;  
 bool get isLoading => \_isLoading;  
  
 @override  
 List<Object?> get props => [\_data, \_message, \_isLoading];  
}

**Özellikler:** - **Generic yapı**: Farklı veri tipleriyle kullanılabilir - **Equatable**: Gereksiz build’leri önler - **Immutable**: State değişikliklerini kontrol eder - **Null safety**: Dart’ın null safety özelliklerini kullanır

### 2.2 State Tipleri

#### 2.2.1 InitialState

class InitialState<T> extends BaseState<T> {  
 const InitialState() : super();  
}

* Bloc’un başlangıç durumu
* Hiçbir veri yüklenmemiş durum
* UI’da genellikle boş bir durum gösterir

#### 2.2.2 LoadingState

class LoadingState<T> extends BaseState<T> {  
 const LoadingState() : super(isLoading: true);  
}

* Veri yükleme durumu
* UI’da loading indicator gösterir
* API çağrısı devam ediyor

#### 2.2.3 LoadedState

class LoadedState<T> extends BaseState<T> {  
 const LoadedState(T data) : super(data: data);  
}

* Veri başarıyla yüklendi
* UI’da veriyi gösterir
* API çağrısı başarılı

#### 2.2.4 ErrorState

class ErrorState<T> extends BaseState<T> {  
 const ErrorState(String message) : super(message: message);  
}

* Hata durumu
* UI’da hata mesajı gösterir
* API çağrısı başarısız

#### 2.2.5 NoContentState

class NoContentState<T> extends BaseState<T> {  
 const NoContentState() : super();  
}

* Veri bulunamadı
* UI’da “veri yok” mesajı gösterir
* API çağrısı başarılı ama veri boş

## 3. API Response Yapısı

### 3.1 ApiResponse Sınıfı

@freezed  
class ApiResponse<T> with \_$ApiResponse<T> {  
 const factory ApiResponse.success(T data) = ApiResponseSuccess<T>;  
 const factory ApiResponse.error(String message, {String? type}) = ApiResponseError<T>;  
 const factory ApiResponse.noContent() = ApiResponseNoContent<T>;  
}

**Özellikler:** - **Freezed**: Immutable ve type-safe yapı - **Pattern matching**: Kolay kullanım - **Generic**: Farklı veri tipleriyle kullanılabilir - **Union types**: Üç farklı durum tanımlar

### 3.2 Kullanım Örnekleri

#### 3.2.1 Success Response

final response = ApiResponse.success(userData);  
response.when(  
 success: (data) => print('Başarılı: $data'),  
 error: (message, type) => print('Hata: $message'),  
 noContent: () => print('Veri yok'),  
);

#### 3.2.2 Error Response

final response = ApiResponse.error('Kullanıcı bulunamadı', type: 'NOT\_FOUND');

#### 3.2.3 NoContent Response

final response = ApiResponse.noContent();

## 4. Repository Mixin

### 4.1 BaseRepositoryMixin

mixin BaseRepositoryMixin {  
 T? parseResponse<T>(ApiResponse<T> response) {  
 return response.maybeWhen(  
 success: (data) => data,  
 orElse: () => null,  
 );  
 }  
  
 ApiResponse<T> handleResponse<T>({  
 required T? Function() parseData,  
 String? errorMessage,  
 String? errorType,  
 }) {  
 try {  
 final data = parseData();  
 if (data == null || (data is Map && data.isEmpty)) {  
 return const ApiResponse.noContent();  
 }  
 return ApiResponse.success(data);  
 } catch (e) {  
 return ApiResponse.error(  
 errorMessage ?? 'Bir hata oluştu',  
 type: errorType,  
 );  
 }  
 }  
  
 ApiResponse<bool> handleBoolResponse({  
 required bool Function() parseData,  
 String? errorMessage,  
 String? errorType,  
 }) {  
 try {  
 final result = parseData();  
 return ApiResponse.success(result);  
 } catch (e) {  
 return ApiResponse.error(  
 errorMessage ?? 'İşlem başarısız',  
 type: errorType,  
 );  
 }  
 }  
}

**Faydaları:** - **Merkezi hata yönetimi**: Tüm repository’lerde aynı hata yönetimi - **Kod tekrarını önleme**: Ortak fonksiyonlar - **Tutarlılık**: Aynı response formatı - **Bakım kolaylığı**: Tek yerden yönetim

### 4.2 Repository Kullanım Örneği

class ProfileRepository with BaseRepositoryMixin {  
 Future<ApiResponse<ProfileModel>> getProfile() async {  
 return handleResponse<ProfileModel>(  
 parseData: () => \_mockProfile, // Gerçek API çağrısı  
 errorMessage: 'Profil alınamadı',  
 );  
 }  
  
 Future<ApiResponse<bool>> updateProfile(ProfileModel profile) async {  
 return handleBoolResponse(  
 parseData: () => \_updateProfile(profile), // Gerçek API çağrısı  
 errorMessage: 'Profil güncellenemedi',  
 );  
 }  
}

## 5. API Call Mixin

### 5.1 HandleApiCallMixin

mixin HandleApiCallMixin {  
 Future<void> handleApiCall<T>({  
 required Future<ApiResponse<T>> Function() apiCall,  
 required void Function(BaseState<T>) emitState,  
 void Function(T data)? onSuccess,  
 void Function(String? message)? onError,  
 void Function()? onNoContent,  
 }) async {  
 emitState(LoadingState<T>());  
  
 final response = await apiCall();  
 response.when(  
 success: (data) {  
 emitState(LoadedState<T>(data));  
 onSuccess?.call(data);  
 },  
 error: (message, type) {  
 emitState(ErrorState<T>(message));  
 onError?.call(message);  
 },  
 noContent: () {  
 emitState(NoContentState<T>());  
 onNoContent?.call();  
 },  
 );  
 }  
  
 Future<void> handleVoidApiCall({  
 required Future<ApiResponse<void>> Function() apiCall,  
 void Function(BaseState<void>)? emitState,  
 required void Function() onSuccess,  
 void Function(String? message)? onError,  
 void Function()? onNoContent,  
 }) async {  
 emitState?.call(LoadingState<void>());  
 final response = await apiCall();  
 response.when(  
 success: (\_) => onSuccess(),  
 error: (message, type) {  
 emitState?.call(ErrorState<void>(message));  
 onError?.call(message);  
 },  
 noContent: () => onNoContent?.call(),  
 );  
 }  
}

**Faydaları:** - **Otomatik state yönetimi**: Loading, success, error durumları otomatik - **Standart yapı**: Tüm bloc’larda aynı pattern - **Callback desteği**: Başarı/hata durumlarında özel işlemler - **Kod tekrarını önleme**: Ortak API çağrı mantığı

### 5.2 Bloc Kullanım Örneği

class ProfileBloc extends Bloc<ProfileEvent, ProfileState> with HandleApiCallMixin {  
 final ProfileRepository \_repository;  
  
 ProfileBloc(this.\_repository) : super(const ProfileState(  
 profileState: InitialState(),  
 detailsState: InitialState(),  
 )) {  
 on<LoadProfileInfo>(\_onLoadProfileInfo);  
 }  
  
 Future<void> \_onLoadProfileInfo(LoadProfileInfo event, Emitter<ProfileState> emit) async {  
 await handleApiCall<ProfileModel>(  
 apiCall: \_repository.getProfile,  
 emitState: (state) => emit(ProfileState(  
 profileState: state,  
 detailsState: this.state.detailsState,  
 )),  
 onSuccess: (data) => print('Profil yüklendi: ${data.name}'),  
 onError: (message) => print('Hata: $message'),  
 );  
 }  
}

## 6. Base Widget Yapıları

### 6.1 BaseBlocBuilder

class BaseBlocBuilder<B extends BlocBase<S>, S, T> extends StatelessWidget {  
 final B bloc;  
 final BaseState Function(S state) stateSelector;  
 final Widget Function(T data) onLoaded;  
 final Widget? onLoading;  
 final Widget Function(String? message)? onError;  
 final Widget? onNoContent;  
  
 const BaseBlocBuilder({  
 super.key,  
 required this.bloc,  
 required this.stateSelector,  
 required this.onLoaded,  
 this.onLoading,  
 this.onError,  
 this.onNoContent,  
 });  
  
 @override  
 Widget build(BuildContext context) {  
 return BlocBuilder<B, S>(  
 bloc: bloc,  
 builder: (context, state) {  
 final baseState = stateSelector(state);  
  
 if (baseState is LoadingState) {  
 return onLoading ?? const Center(child: CircularProgressIndicator());  
 }  
 if (baseState is LoadedState<T>) {  
 return onLoaded(baseState.data!);  
 }  
 if (baseState is ErrorState) {  
 if (onError != null) return onError!(baseState.message);  
 return Center(child: Text(baseState.message ?? 'Bir hata oluştu'));  
 }  
 if (baseState is NoContentState) {  
 return onNoContent ?? const Center(child: Text('İçerik bulunamadı'));  
 }  
 return const SizedBox.shrink();  
 },  
 );  
 }  
}

**Faydaları:** - **Otomatik state yönetimi**: State’e göre otomatik UI güncelleme - **Tutarlı UI**: Tüm sayfalarda aynı loading/error gösterimi - **Kod tekrarını önleme**: Ortak widget yapısı - **Esneklik**: Özel UI override edilebilir

### 6.2 BaseBlocConsumer

class BaseBlocConsumer<B extends BlocBase<S>, S, T> extends StatelessWidget {  
 final B bloc;  
 final BaseState Function(S state) stateSelector;  
 final Widget Function(T data) onLoaded;  
 final void Function(BuildContext context, BaseState<T> state)? onStateChange;  
 final Widget? onLoading;  
 final Widget Function(String? message)? onError;  
 final Widget? onNoContent;  
 final bool Function(S previous, S current)? listenWhen;  
  
 const BaseBlocConsumer({  
 super.key,  
 required this.bloc,  
 required this.stateSelector,  
 required this.onLoaded,  
 this.onStateChange,  
 this.onLoading,  
 this.onError,  
 this.onNoContent,  
 this.listenWhen,  
 });  
  
 @override  
 Widget build(BuildContext context) {  
 return BlocConsumer<B, S>(  
 bloc: bloc,  
 listenWhen: listenWhen,  
 listener: (context, state) {  
 final baseState = stateSelector(state);  
 if (onStateChange != null) {  
 onStateChange!(context, baseState as BaseState<T>);  
 }  
 },  
 builder: (context, state) {  
 final baseState = stateSelector(state);  
  
 if (baseState is LoadingState) {  
 return onLoading ?? const Center(child: CircularProgressIndicator());  
 }  
 if (baseState is LoadedState<T>) {  
 return onLoaded(baseState.data!);  
 }  
 if (baseState is ErrorState) {  
 if (onError != null) return onError!(baseState.message);  
 return Center(child: Text(baseState.message ?? 'Bir hata oluştu'));  
 }  
 if (baseState is NoContentState) {  
 return onNoContent ?? const Center(child: Text('İçerik bulunamadı'));  
 }  
 return const SizedBox.shrink();  
 },  
 );  
 }  
}

**Faydaları:** - **State dinleme**: State değişikliklerinde özel işlemler - **UI güncelleme**: State’e göre otomatik UI değişimi - **Esneklik**: listenWhen ile koşullu dinleme - **Tutarlılık**: Tüm sayfalarda aynı davranış

### 6.3 Widget Kullanım Örnekleri

#### 6.3.1 Basit Kullanım

BaseBlocBuilder<ProfileBloc, ProfileState, ProfileModel>(  
 bloc: context.read<ProfileBloc>(),  
 stateSelector: (state) => state.profileState,  
 onLoaded: (profile) => ProfileCard(profile: profile),  
 onLoading: const Center(child: CircularProgressIndicator()),  
 onError: (message) => Center(child: Text('Hata: $message')),  
 onNoContent: const Center(child: Text('Profil bulunamadı')),  
)

#### 6.3.2 Consumer Kullanımı

BaseBlocConsumer<ProfileBloc, ProfileState, ProfileModel>(  
 bloc: context.read<ProfileBloc>(),  
 stateSelector: (state) => state.profileState,  
 onLoaded: (profile) => ProfileCard(profile: profile),  
 onStateChange: (context, state) {  
 if (state is LoadedState) {  
 ScaffoldMessenger.of(context).showSnackBar(  
 const SnackBar(content: Text('Profil yüklendi!')),  
 );  
 }  
 },  
)

## 7. Dependency Injection

### 7.1 GetIt Kullanımı

final getIt = GetIt.instance;  
  
void setupInjection() {  
 // Repository'ler  
 getIt.registerLazySingleton<ProfileRepository>(() => ProfileRepository());  
 getIt.registerLazySingleton<CardRepository>(() => CardRepository());  
  
 // Bloc'lar  
 getIt.registerFactory<ProfileBloc>(() => ProfileBloc(getIt<ProfileRepository>()));  
 getIt.registerFactory<CardsBloc>(() => CardsBloc(getIt<CardRepository>()));  
}

**Faydaları:** - **Test edilebilirlik**: Mock’lar kolayca değiştirilebilir - **Loose coupling**: Bağımlılıklar gevşek - **Singleton yönetimi**: Tek instance garantisi - **Factory pattern**: Her kullanımda yeni instance

### 7.2 Kullanım Örnekleri

#### 7.2.1 Widget’ta Kullanım

class ProfilePage extends StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 return BlocProvider(  
 create: (context) => getIt<ProfileBloc>()..add(const LoadProfileInfo()),  
 child: const ProfileView(),  
 );  
 }  
}

#### 7.2.2 Test’te Kullanım

setUp(() {  
 getIt.registerSingleton<ProfileRepository>(MockProfileRepository());  
 profileBloc = getIt<ProfileBloc>();  
});

## 8. Error Handling

### 8.1 AppError Sınıfı

class AppError {  
 final String message;  
 final String? code;  
 final dynamic originalError;  
  
 const AppError({  
 required this.message,  
 this.code,  
 this.originalError,  
 });  
  
 @override  
 String toString() => 'AppError: $message (Code: $code)';  
}

### 8.2 Error Handling Stratejisi

// Repository seviyesinde  
try {  
 final response = await apiCall();  
 return handleResponse(data: response);  
} catch (e) {  
 return ApiResponse.error('Ağ hatası: ${e.toString()}');  
}  
  
// Bloc seviyesinde  
await handleApiCall(  
 apiCall: \_repository.getData,  
 emitState: (state) => emit(MyState(dataState: state)),  
 onError: (message) => \_logError(message),  
);

## 9. Best Practices

### 9.1 Kod Organizasyonu

#### 9.1.1 Dosya Yapısı

lib/  
├── core/  
│ ├── base/  
│ │ ├── base\_state.dart  
│ │ ├── base\_repository\_mixin.dart  
│ │ └── handle\_api\_call\_mixin.dart  
│ ├── models/  
│ │ └── api\_response.dart  
│ └── widgets/  
│ ├── base\_bloc\_builder.dart  
│ └── base\_bloc\_consumer.dart  
├── features/  
│ └── profile/  
│ ├── data/  
│ ├── domain/  
│ └── presentation/  
└── main.dart

#### 9.1.2 Naming Conventions

* **State sınıfları**: \*State suffix’i
* **Event sınıfları**: \*Event suffix’i
* **Bloc sınıfları**: \*Bloc suffix’i
* **Repository sınıfları**: \*Repository suffix’i
* **Model sınıfları**: \*Model suffix’i

### 9.2 Performance Optimizasyonu

#### 9.2.1 State Yönetimi

// İyi örnek  
BlocBuilder<ProfileBloc, ProfileState>(  
 buildWhen: (previous, current) =>   
 previous.profileState != current.profileState,  
 builder: (context, state) => ProfileWidget(state.profileState),  
)  
  
// Kötü örnek  
BlocBuilder<ProfileBloc, ProfileState>(  
 builder: (context, state) => ProfileWidget(state.profileState),  
)

#### 9.2.2 Memory Management

// Widget dispose edildiğinde  
@override  
void dispose() {  
 \_bloc.close();  
 super.dispose();  
}

### 9.3 Test Edilebilirlik

#### 9.3.1 Mock Kullanımı

class MockProfileRepository extends Mock implements ProfileRepository {  
 @override  
 Future<ApiResponse<ProfileModel>> getProfile() async {  
 return ApiResponse.success(mockProfile);  
 }  
}

#### 9.3.2 Test Helper’ları

class TestHelpers {  
 static ProfileState createLoadedState(ProfileModel profile) {  
 return ProfileState(  
 profileState: LoadedState(profile),  
 detailsState: const InitialState(),  
 );  
 }  
}

## 10. Örnek Uygulama

### 10.1 Tam Örnek: Profile Feature

#### 10.1.1 Model

@freezed  
class ProfileModel with \_$ProfileModel {  
 const factory ProfileModel({  
 required int id,  
 required String name,  
 required String email,  
 String? avatar,  
 }) = \_ProfileModel;  
}

#### 10.1.2 Repository

class ProfileRepository with BaseRepositoryMixin {  
 Future<ApiResponse<ProfileModel>> getProfile() async {  
 return handleResponse<ProfileModel>(  
 parseData: () => \_fetchProfileFromAPI(),  
 errorMessage: 'Profil alınamadı',  
 );  
 }  
}

#### 10.1.3 Bloc

class ProfileBloc extends Bloc<ProfileEvent, ProfileState> with HandleApiCallMixin {  
 final ProfileRepository \_repository;  
  
 ProfileBloc(this.\_repository) : super(const ProfileState(  
 profileState: InitialState(),  
 detailsState: InitialState(),  
 )) {  
 on<LoadProfileInfo>(\_onLoadProfileInfo);  
 }  
  
 Future<void> \_onLoadProfileInfo(LoadProfileInfo event, Emitter<ProfileState> emit) async {  
 await handleApiCall<ProfileModel>(  
 apiCall: \_repository.getProfile,  
 emitState: (state) => emit(ProfileState(  
 profileState: state,  
 detailsState: this.state.detailsState,  
 )),  
 );  
 }  
}

#### 10.1.4 Widget

class ProfilePage extends StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: const Text('Profil')),  
 body: BaseBlocConsumer<ProfileBloc, ProfileState, ProfileModel>(  
 bloc: context.read<ProfileBloc>(),  
 stateSelector: (state) => state.profileState,  
 onLoaded: (profile) => ProfileCard(profile: profile),  
 onStateChange: (context, state) {  
 if (state is LoadedState) {  
 ScaffoldMessenger.of(context).showSnackBar(  
 const SnackBar(content: Text('Profil yüklendi!')),  
 );  
 }  
 },  
 ),  
 );  
 }  
}

Bu base yapılar, Flutter BLoC projelerinde tutarlı, sürdürülebilir ve test edilebilir kod yazmanızı sağlar. Bu yapıları kullanarak, kod tekrarını önleyebilir, hata yönetimini standartlaştırabilir ve geliştirme sürecini hızlandırabilirsiniz.