1. Dependencies

Required Dependencies (`pubspec.yaml`)

```
dependencies:
  flutter:
    sdk: flutter

  # UI
  cupertino_icons: ^1.0.8
  cached_network_image: ^3.4.1
  sizer: ^3.0.4

  # Code Utility
  provider: ^6.1.2
  path_provider: ^2.0.1
  get_it: ^8.0.0
  injectable: ^2.3.0
  json_annotation: ^4.9.0
  freezed_annotation: ^2.4.1
  dartz: ^0.10.1
  permission_handler: ^11.3.1
  flutter_bloc: ^9.0.0
  auto_route: ^9.2.2

  # Network & Database
  dio: ^5.7.0
  hive: ^2.0.4
  hive_flutter: ^1.0.0
```

Dev Dependencies

```
dev_dependencies:
  flutter_test:
    sdk: flutter
  build_runner:
  flutter_lints: ^4.0.0
  freezed: ^2.4.2
  injectable_generator: ^2.4.0
  json_serializable: ^6.7.1
  auto_route_generator: ^9.0.0
  hive_generator: ^2.0.1
```

---

2. Project Folder Structure

```
lib/
├── _shared/
│   ├── _core/              # Core utilities (errors, failures, value objects)
│   ├── config/             # Application-wide configurations (logger, theme)
│   ├── services/           # Routing, dependency injection, shared services
│   ├── utilities/          # Extension methods, common utilities
│
├── application/            # State management layer (BLoC, Cubit, Provider)
│   ├── {feature1}/
│   ├── {feature2}/
│
├── domain/                 # Business logic (Entities, Use Cases, Repositories)
│   ├── shared/             # Shared domain logic across features
│   ├── {feature1}/
│   ├── {featureN}/
│
├── data/                   # Data Layer (Implements Domain)
│   ├── shared/             # DTOs, repository implementations
│   ├── {feature1}/
│   ├── {featureN}/
│
├── presentation/           # UI Layer (Screens, Widgets)
│   ├── shared/             # Common UI widgets, constants
│   ├── {feature1}/
│   ├── {featureN}/
```

---

3. Core Files

Error

lib/_shared/_core/error.dart

```dart
import 'value_failures.dart';

class NotAuthenticatedError extends Error {}

class UnExpectedValueError extends Error {
  final ValueFailure valueFailure;

  UnExpectedValueError(this.valueFailure);

  @override
  String toString() {
    const explanation =
        "Encountered a ValueFailure at an unrecoverable point. Terminating.";
    return Error.safeToString('$explanation Failure was: $valueFailure');
  }
}
```

Failure

lib/_shared/_core/failure.dart

```
 import 'package:freezed_annotation/freezed_annotation.dart';
part 'failure.freezed.dart';

@freezed
abstract class Failure with _$Failure {
  const factory Failure.networkError() = _NetworkError;

  const factory Failure.unexpected({required String errorMessage}) =
      _Unexpected;

  factory Failure.commonFailure() => const Failure.unexpected(
        errorMessage: "Some error occurred. Please try again",
      );
}
```

Value Failures

`lib/_shared/_core/value_failures.dart`

```dart
import 'package:freezed_annotation/freezed_annotation.dart';

part 'value_failures.freezed.dart';

@freezed
abstract class ValueFailure<T> with _$ValueFailure<T> {
  const factory ValueFailure.empty({
    required T failedValue,
  }) = _Empty<T>;

  const factory ValueFailure.invalidValue({
    required T failedValue,
    required String errorMsg,
  }) = _InvalidValue<T>;

  const factory ValueFailure.mustBeAlphabets({
    required T failedValue,
  }) = _MustBeAlphabets<T>;

  const factory ValueFailure.minLength({
    required T failedValue,
    required int minLength,
  }) = _MinLength<T>;

  const factory ValueFailure.maxLength({
    required T failedValue,
    required int maxLength,
  }) = _MaxLength<T>;

  const factory ValueFailure.containsWhitespace({
    required T failedValue,
  }) = _ContainsWhitespace<T>;

  const factory ValueFailure.multiline({
    required T failedValue,
  }) = _Multiline<T>;

  const factory ValueFailure.alreadyExists({
    required T failedValue,
  }) = _EmailAlreadyExists<T>;

  const factory ValueFailure.listTooShort({
    required T failedValue,
    required int min,
  }) = _ListTooShort<T>;

  const factory ValueFailure.listTooLong({
    required T failedValue,
    required int max,
  }) = _ListTooLong<T>;
}
```

Value Object

lib/_shared/_core/value_object.dart

```
import 'package:dartz/dartz.dart';
import 'package:flutter/foundation.dart';
import 'package:uuid/uuid.dart';

import 'error.dart';
import 'value_failures.dart';

@immutable
abstract class ValueObject<T> {
  const ValueObject();
  Either<ValueFailure<T>, T> get value;

  ///Throws [UnExpectedValueError] containing the [ValueFailure]
  T getOrCrash() {
    return value.fold((f) => throw UnExpectedValueError(f), id);
  }

  bool isValid() => value.isRight();

  Either<ValueFailure<dynamic>, Unit> get failureOrUnit {
    return value.fold(
      (l) => left(l),
      (r) => right(unit),
    );
  }

  @override
  bool operator ==(Object other) {
    if (identical(this, other)) return true;

    return other is ValueObject<T> && other.value == value;
  }

  @override
  int get hashCode => value.hashCode;

  @override
  String toString() => 'Value($value)';
}

class UniqueId extends ValueObject<String> {
  @override
  final Either<ValueFailure<String>, String> value;

  // We cannot let a simple String be passed in. This would allow for possible non-unique IDs.
  factory UniqueId() {
    return UniqueId._(
      right(const Uuid().v1()),
    );
  }

  /// Used with strings we trust are unique, such as database IDs.
  factory UniqueId.fromUniqueString(String uniqueIdStr) {
    //assert(uniqueIdStr != null);
    return UniqueId._(
      right(uniqueIdStr),
    );
  }

  const UniqueId._(this.value);
}
```

AutoRoute (Navigation)

lib/_shared/service/app_router_config.dart

```
import '../config/logger.dart';
import 'package:auto_route/auto_route.dart';
import 'app_router_config.gr.dart';

@AutoRouterConfig(replaceInRouteName: 'Screen|Page,Route')
class AppRouterConfig extends RootStackRouter {
  static final AppRouterConfig _instance = AppRouterConfig._internal();
  factory AppRouterConfig() => _instance;
  AppRouterConfig._internal();

  @override
  RouteType get defaultRouteType => const RouteType.adaptive();

  @override
  List<AutoRoute> get routes {
    Logger.i("routes called");
    return [
      AutoRoute(
        page: LoginRoute.page,
        initial: true,
      ),
      AutoRoute(
        page: RegisterRoute.page,
      ),
      AutoRoute(
        page: HomeRoute.page,
      ),
    ];
  }
}
```

Dependency Injection

lib/_shared/service/d_injection.dart

```
import 'package:get_it/get_it.dart';
import 'package:injectable/injectable.dart';
import 'd_injection.config.dart';

final getIt = GetIt.instance;

@injectableInit
void configureInjection() => GetIt.I.init();
```

Logging

lib/_shared/config/logger.dart

```
import 'dart:developer';
import 'package:flutter/foundation.dart';

class Logger {
  static const String _reset = '\x1B[0m'; // Reset color
  static const String _green = '\x1B[32m'; // _green color
  static const String _red = '\x1B[31m'; // Red color
  static const String _grey = '\x1B[90m'; // Grey color
  static const String _purple = '\x1B[35m';

  static void d(String message, {String? logName}) {
    _printLog(message, logName: logName ?? 'DEBUG', color: _grey, emoji: '');
  }

  static void i(String message, {String? logName}) {
    _printLog(message, logName: logName ?? 'INFO', color: _green, emoji: '');
  }

  static void w(String message, {String? logName}) {
    _printLog(
      message,
      logName: logName ?? 'WARNING',
      color: _purple,
      emoji: '⚠',
    );
  }

  static void e(String message, {String? logName}) {
    _printLog(message, logName: logName ?? 'ERROR', color: _red, emoji: '');
  }

  static void _printLog(
    String message, {
    required String logName,
    required String color,
    required String emoji,
  }) {
    if (kDebugMode) {
      log(
        '$emoji [$logName]:: $color$message$_reset',
        name: logName,
      );
    }
  }
}
```

Theme Configuration

lib/_shared/config/theme_config.dart

```
import 'package:flutter/material.dart';

final appThemeData = {
  'light': AppTheme.lightTheme,
  'dark': AppTheme.darkTheme,
};

class AppTheme {
  AppTheme._();

  static const primaryColor = Color(0xFF4CAF50);
  static const onPrimaryColor = Color(0xFFFFFFFF);
  static const primaryContainer = Color(0xFF81C784);
  static const onPrimaryContainer = Color(0xFF002D00);
```

```
/// Primary Dark:
static const primaryColorDark = Color(0xFF388E3C);
static const onPrimaryColorDark = Color(0xFFFFFFFF);
static const primaryContainerDark = Color(0xFF66BB6A);
static const onPrimaryContainerDark = Color(0xFF002400);

/// Secondary Light:
static const secondaryColor = Color(0xFFFFC107);
static const onSecondaryColor = Color(0xFF000000);
static const secondaryContainer = Color(0xFFFFE082);
static const onSecondaryContainer = Color(0xFF3E2C00);

/// Secondary Dark:
static const secondaryColorDark = Color(0xFFFFB300);
static const onSecondaryColorDark = Color(0xFF3E2C00);
static const secondaryContainerDark = Color(0xFFFFD54F);
static const onSecondaryContainerDark = Color(0xFF402D00);

/// Tertiary Light:
static const tertiaryColor = Color(0xFF8BC34A);
static const onTertiaryColor = Color(0xFF000000);
static const tertiaryContainer = Color(0xFFDCEDC8);
static const onTertiaryContainer = Color(0xFF263700);

/// Tertiary Dark:
static const tertiaryColorDark = Color(0xFF7CB342);
static const onTertiaryColorDark = Color(0xFF1A3700);
static const tertiaryContainerDark = Color(0xFFC5E1A5);
static const onTertiaryContainerDark = Color(0xFF263700);

/// Error Light:
static const errorColor = Color(0xFFD32F2F);
static const onErrorColor = Color(0xFFFFFFFF);
static const errorContainer = Color(0xFFFFCDD2);
static const onErrorContainer = Color(0xFF790000);

/// Error Dark:
static const errorColorDark = Color(0xFFEF9A9A);
static const onErrorColorDark = Color(0xFF5A0000);
static const errorContainerDark = Color(0xFFD32F2F);
static const onErrorContainerDark = Color(0xFF790000);

static final ThemeData lightTheme = ThemeData(
  scaffoldBackgroundColor: Colors.white,
  brightness: Brightness.light,
  textTheme: const TextTheme().apply(
    fontFamily: "Montserrat",
    displayColor: Colors.black54,
    bodyColor: Colors.black87,
  ),
  primaryColor: primaryColor,
  colorScheme: const ColorScheme(
    primary: primaryColor,
    onPrimary: onPrimaryColor,
    primaryContainer: primaryContainer,
    onPrimaryContainer: onPrimaryContainer,
    secondary: secondaryColor,
    onSecondary: onSecondaryColor,
    secondaryContainer: secondaryContainer,
    onSecondaryContainer: onSecondaryContainer,
    tertiary: tertiaryColor,
    onTertiary: onTertiaryColor,
    tertiaryContainer: tertiaryContainer,
    onTertiaryContainer: onTertiaryContainer,
    error: errorColor,
```

```
    onError: onErrorColor,
    errorContainer: errorContainer,
    onErrorContainer: onErrorContainer,
    surfaceDim: Color(0xFFddd8e7),
    surface: Color(0xFFfdf8ff),
    surfaceBright: Color(0xFFfdf8ff),
    surfaceContainerLowest: Color(0xFFffffff),
    surfaceContainerLow: Color(0xFFf7f1ff),
    surfaceContainer: Color(0xFFf1ebfb),
    surfaceContainerHigh: Color(0xFFebe6f5),
    surfaceContainerHighest: Color(0xFFe6e0ef),
    onSurface: Color(0xFF1c1a25),
    onSurfaceVariant: Color(0xFF484456),
    outline: Color(0xFF797488),
    outlineVariant: Color(0xFFc9c3d9),
    inverseSurface: Color(0xFF312f3a),
    onInverseSurface: Color(0xFFf4eefe),
    inversePrimary: Color(0xFFc9beff),
    scrim: Color(0xFF000000),
    shadow: Color(0xFF000000),
    brightness: Brightness.light,
  ),
  inputDecorationTheme: InputDecorationTheme(
    hintStyle: const TextStyle(
      color: Colors.black38,
      fontFamily: "Montserrat",
    ),
    errorMaxLines: 2,
    focusedBorder: OutlineInputBorder(
      borderRadius: BorderRadius.circular(4.0),
      borderSide: const BorderSide(color: primaryColor),
    ),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(4.0),
      borderSide: const BorderSide(color: primaryColor),
    ),
    prefixIconColor: primaryColor,
  ),
  textButtonTheme: TextButtonThemeData(
    style: ButtonStyle(
      foregroundColor: primaryColor.wrapMatProp(),
      backgroundColor: Colors.transparent.wrapMatProp(),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(7.0),
      ).wrapMatProp(),
      minimumSize: const Size(double.minPositive, 25).wrapMatProp(),
    ),
  ),
  outlinedButtonTheme: OutlinedButtonThemeData(
    style: ButtonStyle(
      foregroundColor:
          primaryColor.wrapMatStateColor(disabledColor: Colors.grey[500]!),
      iconColor:
          primaryColor.wrapMatStateColor(disabledColor: Colors.grey[500]!),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(7.0),
        side: const BorderSide(color: primaryColor),
      ).wrapMatProp(),
      side: WidgetStateProperty.resolveWith<BorderSide>((states) {
        final borderColor = primaryColor
            .wrapMatStateColor(
              disabledColor: Colors.grey[500]!,
            )
            .getAbsValue(states: states);
```

```
          return BorderSide(color: borderColor ?? primaryColor);
        }),
        minimumSize: const Size(double.maxFinite, 50).wrapMatProp(),
      ),
    ),
    elevatedButtonTheme: ElevatedButtonThemeData(
      style: ButtonStyle(
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(7.0),
        ).wrapMatProp(),
        backgroundColor:
            primaryColor.wrapMatStateColor(disabledColor: Colors.grey),
        foregroundColor: Colors.white.wrapMatProp(),
        minimumSize: const Size(double.infinity, 50).wrapMatProp(),
      ),
    ),
  );

  static final ThemeData darkTheme = ThemeData(
    scaffoldBackgroundColor: Colors.black,
    brightness: Brightness.dark,
    textTheme: const TextTheme().apply(fontFamily: "Montserrat"),
    primaryColor: primaryColor,
    primaryColorDark: primaryColorDark,
    primaryColorLight: primaryContainer,
    colorScheme: const ColorScheme(
      primary: primaryColorDark,
      onPrimary: onPrimaryColorDark,
      primaryContainer: primaryContainerDark,
      onPrimaryContainer: onPrimaryContainerDark,
      secondary: secondaryColorDark,
      onSecondary: onSecondaryColorDark,
      secondaryContainer: secondaryContainerDark,
      onSecondaryContainer: onSecondaryContainerDark,
      tertiary: tertiaryColorDark,
      onTertiary: onTertiaryColorDark,
      tertiaryContainer: tertiaryContainerDark,
      onTertiaryContainer: onTertiaryContainerDark,
      error: errorColorDark,
      onError: onErrorColorDark,
      errorContainer: errorContainerDark,
      onErrorContainer: onErrorContainerDark,
      surfaceDim: Color(0xFF14121c),
      surface: Color(0xFF14121c),
      surfaceBright: Color(0xFF3a3843),
      surfaceContainerLowest: Color(0xFF0f0d17),
      surfaceContainerLow: Color(0xFF1c1a25),
      surfaceContainer: Color(0xFF201e29),
      surfaceContainerHigh: Color(0xFF2b2834),
      surfaceContainerHighest: Color(0xFF36333f),
      onSurface: Color(0xFFe6e0ef),
      onSurfaceVariant: Color(0xFFc9c3d9),
      outline: Color(0xFF938ea2),
      outlineVariant: Color(0xFF484456),
      inverseSurface: Color(0xFFe6e0ef),
      onInverseSurface: Color(0xFF312f3a),
      inversePrimary: Color(0xFF5f2ff9),
      scrim: Color(0xFF000000),
      shadow: Color(0xFF000000),
      brightness: Brightness.dark,
    ),
  );
}

class MaterialFontSizes {
  // Display
```

```
  // Display
  static const double displayLarge = 57.0;
  static const double displayMedium = 45.0;
  static const double displaySmall = 36.0;

  // Headline
  static const double headlineLarge = 32.0;
  static const double headlineMedium = 28.0;
  static const double headlineSmall = 24.0;

  // Title
  static const double titleLarge = 22.0;
  static const double titleMedium = 16.0;
  static const double titleSmall = 14.0;

  // Body
  static const double bodyLarge = 16.0;
  static const double bodyMedium = 14.0;
  static const double bodySmall = 12.0;

  // Label
  static const double labelLarge = 14.0;
  static const double labelMedium = 12.0;
  static const double labelSmall = 11.0;
}

extension MaterialPropX<T> on T {
  WidgetStateProperty<T> wrapMatProp() =>
      WidgetStateProperty.resolveWith<T>((states) => this);
}

extension MaterialColorPropX<Color> on Color {
  WidgetStateProperty<Color> wrapMatStateColor({
    required Color disabledColor,
    Color? focusedColor, // Add more optional colors for different states
    Color? hoveredColor,
    WidgetState state = WidgetState.disabled, // Default to disabled state
  }) =>
      WidgetStateProperty.resolveWith<Color>(
        (states) {
          if (states.contains(state)) {
            return disabledColor;
          } else if (focusedColor != null &&
              states.contains(WidgetState.focused)) {
            return focusedColor;
          } else if (hoveredColor != null &&
              states.contains(WidgetState.hovered)) {
            return hoveredColor;
          } else {
            return this; // Default state (normal)
          }
        },
      );
}

extension ToValuePropX on WidgetStateProperty<Color> {
  Color? getAbsValue({Set<WidgetState> states = const {WidgetState.disabled}}) {
    return resolve(states);
  }
}

extension TextStyleX on TextStyle {
  TextStyle addMontserratFont() {
    return copyWith(fontFamily: "Montserrat");
    //return GoogleFonts.montserratAlternates(textStyle: this);
  }
```

```
}
```

Utility Methods

lib/_shared/utilities/utility_methods.dart

```
import 'dart:async';

extension StringX on String {
  String capitalize() => this[0].toUpperCase() + substring(1);
}

Function debounce(Function func, {Duration duration = const Duration(milliseconds: 500)}) {
  Timer? timer;
  return () {
    if (timer?.isActive ?? false) timer!.cancel();
    timer = Timer(duration, () => func());
  };
}

String formatDate(DateTime date) {
  return "${date.day}-${date.month}-${date.year}";
}
```

Utility Methods

lib/_shared/utilities/extension_methods.dart

```
import '../_core/value_object.dart';

extension ValueObjectX on ValueObject {
  String? validate() => value.fold(
        (l) => l.maybeMap(
          invalidValue: (e) => e.errorMsg,
          orElse: () => null,
        ),
        (_) => null,
      );
}

extension StringX on String {}
```

Static Analysis Configuration

analysis_options.yaml

```
include: package:flutter_lints/flutter.yaml

analyzer:
  exclude:
    - "**/*.g.dart"
    - "**/*.freezed.dart"

linter:
  rules:
    avoid_print: true
    prefer_relative_imports: true
    always_declare_return_types: true
    require_trailing_commas: true
```

Notes for AI Usage

- This document serves as structured knowledge for setting up a Flutter project.
- When generating project code:
  - Ensure dependencies exist in `pubspec.yaml`.
  - Follow defined folder structure.
  - Use registered services for DI.
  - Apply linting rules from `analysis_options.yaml`.
- For routing:
  - Register routes in `app_router_config.dart`.
  - Ensure `AutoRoute` setup is in place.
- For DI:
  - Register services inside `d_injection.dart`.
  - Ensure `configureInjection()` is called in `main.dart`.
- For utilities:
  - Use `capitalize()`, `debounce()`, and `formatDate()` in relevant places.