# Методические указания к Практической работе №9
# по дисциплине «Разработка кроссплатформенных мобильных приложений»

**Тема работы: Реализация светлой и тёмной тем в Flutter-приложении с использованием ThemeData, ThemeMode и Cubit**

**Цель работы:** освоить динамическую смену светлой и тёмной темы с управлением состоянием темы через Cubit и настройкой кастомных светлой и темной тем приложения.

План практической работы:

- Динамическая смена светлой и темной тем приложения
- Настройка кастомных тем приложения
- Выполнение практической работы №9 в соответствии с заданием

**Последовательность выполнения практической работы:**

1. **Динамическая смена светлой и темной тем приложения**

**Тема (Theme)** во Flutter —коллекция всех стилей приложения, которая обеспечивает ему профессиональный вид. Это все цвета ваших виджетов, все градиенты, а также стили текста, кнопок и т.д. С помощью тем можно выражать фирменный стиль, обеспечивать единообразие отступов и типографики, поддерживать тёмный режим и разделять оформление и бизнес-логику.

Темы уменьшают дублирование кода и облегчают массовые визуальные изменения. По мере масштабирования приложения тема становится единственным источником правды для цветов, типографики, форм элементов, уровней подъёма (elevation), стилей компонентов и пользовательских дизайн-токенов.

Для работы с темой, виджет MaterialApp использует 3 параметра:

- themeMode - принимает геттер режима темы ThemeMode.light или ThemeMode.dark. От данного параметра зависит, какая тема включена: темная или светлая.
- darkTheme - параметр темной темы. Здесь мы указываем какой набор данных ThemeData будет использоваться для темной темы.
- theme - основной параметр ThemeData, который определяет активную тему по умолчанию.

У Flutter 2 способа работы с темой:

- использование стандартной темы из набора Material Design: параметры ThemeData.light() и ThemeData.dark():

```
return MaterialApp(
    theme: ThemeData.light(),   // светлая тема
    darkTheme: ThemeData.dark(),// тёмная тема
    themeMode: themeMode
)
```

Если результат стандартного набора цветовой палитры нас не устраивает, то можно прибегнуть к использованию copyWith и расширить MaterialDesign своими свойствами:

```
theme: ThemeData.light().copyWith(
    scaffoldBackgroundColor: Colors.lightBlue,  //цвет фона
    )
```

- использование кастомных настроек светлой и темной темы (создание собственного класса ThemeData и определение в нем всех необходимых свойств).

**Переключение и сохранение тем во время выполнения**

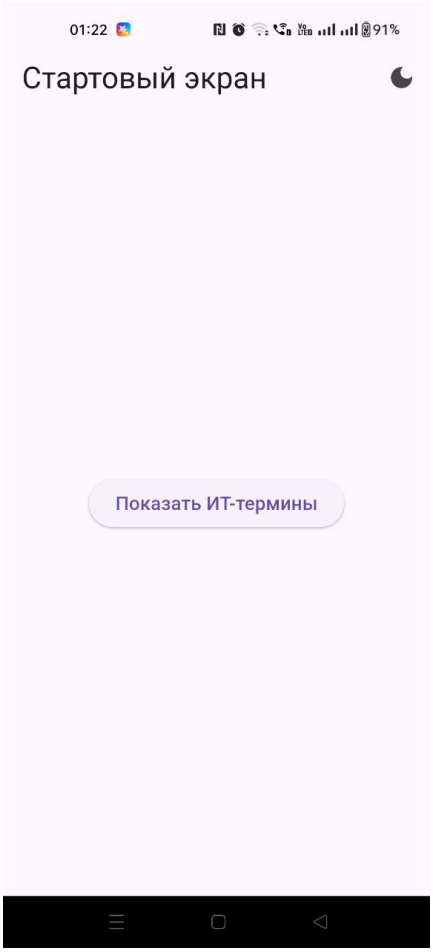
Современные приложения обычно позволяют пользователю переключать светлую и тёмную темы или выбирать собственное оформление. Реализовать такое переключение «на лету» можно, если управлять ThemeMode через систему управления состоянием — например, Provider, Riverpod, Bloc или через наследуемый ValueNotifier.

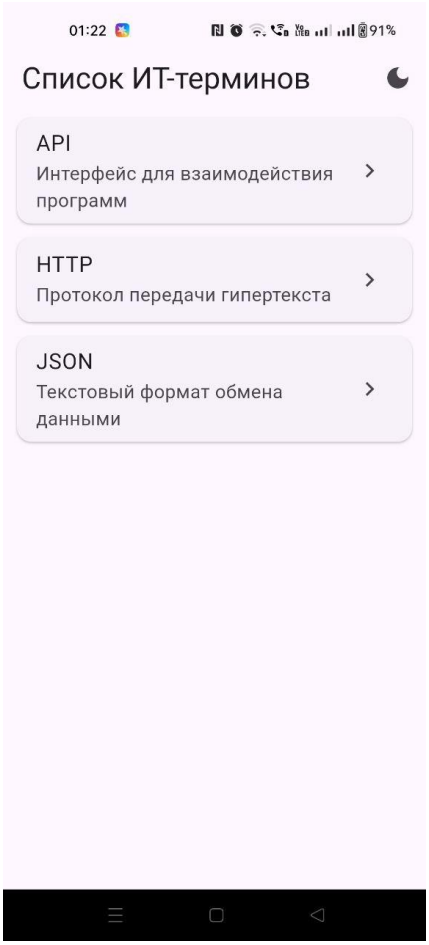
Чтобы выбранная тема сохранялась после закрытия и повторного запуска приложения, предпочтение пользователя можно записывать в SharedPreferences, защищённое хранилище или другое локальное хранилище приложения.

**Пример** приложения с управлением состоянием смены тем с помощью Cubit

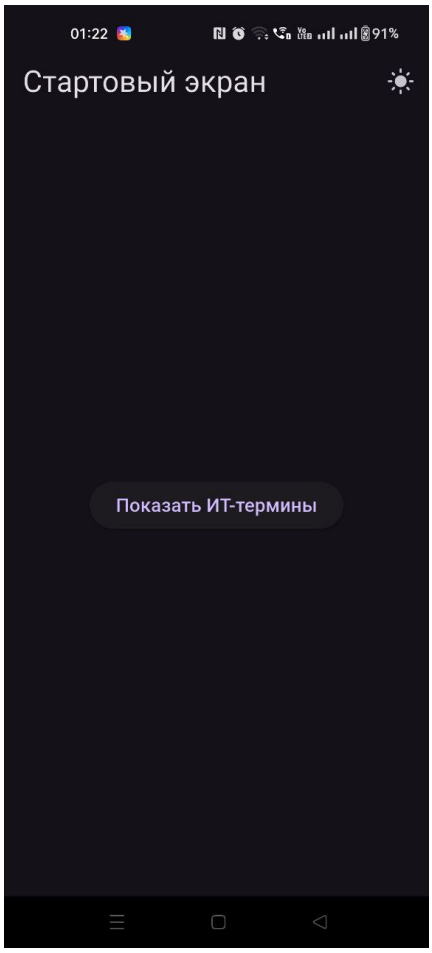
Приложение состоит из 3-х экранов с кнопкой-иконкой для смены тем в AppBar:

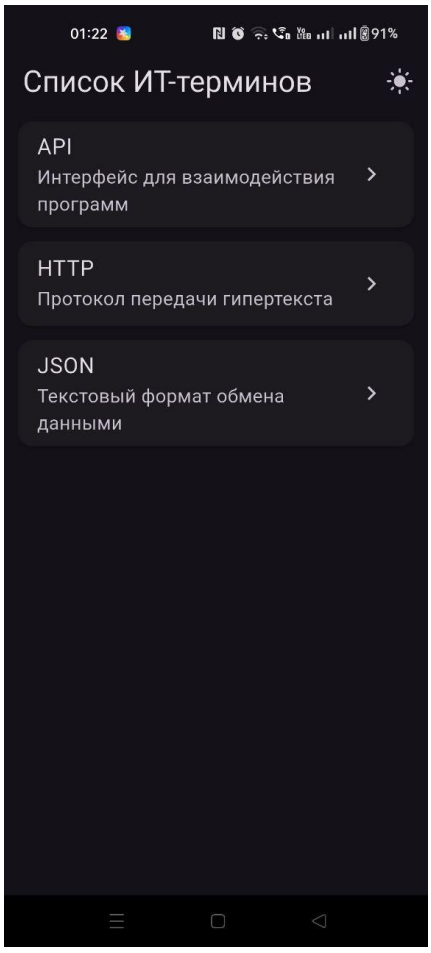
1) Стартовый экран (кнопка «Показать»)
2) Экран списка IT-терминов со строкой поиска по термину
3) Экран детализации термина

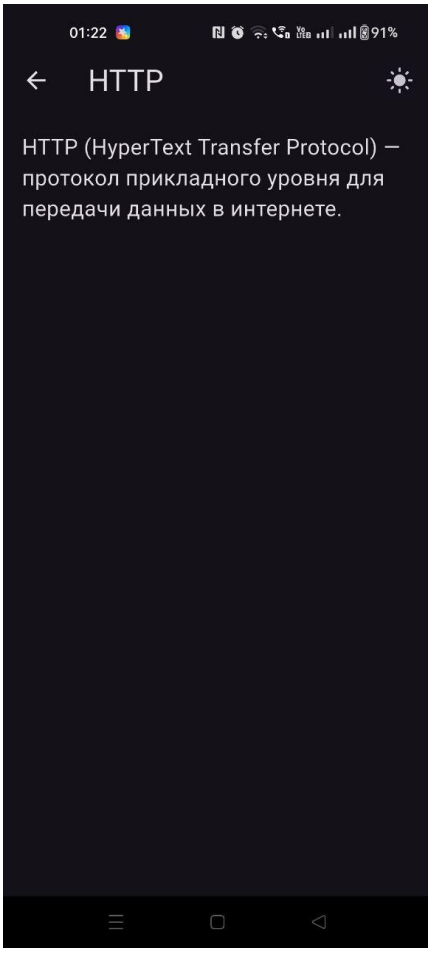




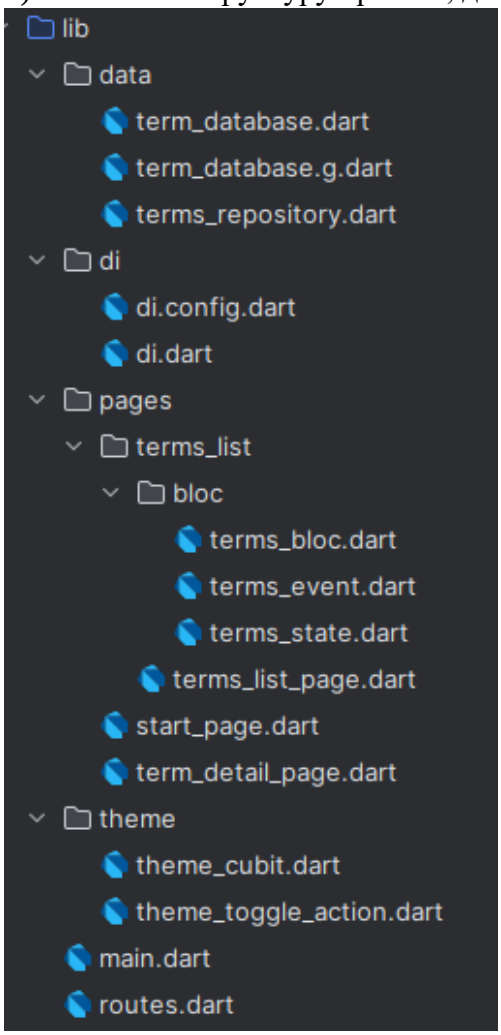


Выполнение:

1) Изменить структуру проекта, добавив папку theme с файлами для управления темой приложения




2) Создаем файл с Cubit для управления состоянием темы (theme/theme_cubit.dart) с аннотацией `@injectable`)

```
3) import 'package:flutter/material.dart';
   import 'package:flutter_bloc/flutter_bloc.dart';
   import 'package:injectable/injectable.dart';

   @injectable
   // Cubit хранит текущее значение ThemeMode
   class ThemeCubit extends Cubit<ThemeMode> {
     ThemeCubit() : super(ThemeMode.light);

     /// Переключить тему
     void toggleTheme(bool isDark) {
       emit(isDark ? ThemeMode.dark : ThemeMode.light);
     }
   }
```

3) Генерируем DI-код

В терминале:

flutter pub run build_runner build --delete-conflicting-outputs

Зависимость для Cubit добавится в файл di/di.config.dart

4) Подключаем ThemeCubit в корне приложения через DI и добавляем в MaterialApp управление иемами

Файл main.dart

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

import 'di/di.dart';
import 'routes.dart';
import 'pages/start_page.dart';
```

```dart
import 'pages/terms_list/terms_list_page.dart';
import 'pages/term_detail_page.dart';
import 'theme/theme_cubit.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();

  // Настраиваем DI
  setupDI();
  runApp(const MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      // Получаем ThemeCubit из DI
      create: (_) => getIt<ThemeCubit>(),
      child: BlocBuilder<ThemeCubit, ThemeMode>(
        builder: (context, themeMode) {
          return MaterialApp(
            title: 'IT Термины (Drift + Theme)',
            debugShowCheckedModeBanner: false,
            theme: ThemeData.light(),    // светлая тема
            darkTheme: ThemeData.dark(),// тёмная тема
            themeMode: themeMode,       // сюда передаём состояние Cubit
            initialRoute: AppRoutes.start,
            routes: {
              AppRoutes.start: (_) => const StartPage(),
              AppRoutes.list: (_) => const TermsListPage(),
              AppRoutes.detail: (_) => const TermDetailPage(),
            },
          );
        },
      ),
    );
  }
}
```

5) Создаем общий виджет-кнопку для AppBar (файл theme/theme_toggle_action.dart)

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

import 'theme_cubit.dart';

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

  @override
  Widget build(BuildContext context) {
    final themeMode = context.watch<ThemeCubit>().state;
    final isDark = themeMode == ThemeMode.dark;

    return IconButton(
      tooltip: isDark ? 'Светлая тема' : 'Тёмная тема',
      icon: Icon(isDark ? Icons.light_mode : Icons.dark_mode),
      onPressed: () {
        context.read<ThemeCubit>().toggleTheme(!isDark);
      },
    );
  }
}
```

6) Добавляем эту кнопку на все страницы в AppBar
Пример страницы start_page.dart

```dart
import 'package:flutter/material.dart';
import '../routes.dart';
import '../theme/theme_toggle_action.dart';

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

  @override
  Widget build(BuildContext context) {
     return Scaffold(
      appBar: AppBar(title: const Text('Стартовый экран'),
        actions: const [
          ThemeToggleAction(), // добавили кнопку
        ],),
      body: Center(
        child:
            ElevatedButton(
              onPressed: () =>
                  Navigator.pushReplacementNamed(context, AppRoutes.list),
              child: const Text('Показать ИТ-термины'),
            ),
        ),
      );
  }
}
```

Аналогично код для кнопки добавляем на все страницы в AppBar

```dart
actions: const [
  ThemeToggleAction(), // добавили кнопку
],
```

**2. Настройка кастомных тем приложения**

Создание единой темы оформления заключается в использовании определенного набора цветов, шрифтов, форм и стилей во всех элементах и разделах приложения. Иными словами, это процесс создания уникального дизайна.

Поскольку большая часть виджетов оформляется в одинаковом стиле, не следует создавать различные стили для каждого виджета в отдельности. Вместо этого нужно определить стиль оформления в одном месте и потом применять его ко всем виджетам. Такой подход позволяет избежать появления повторяющихся фрагментов, что упрощает чтение, понимание и редактирование кода.

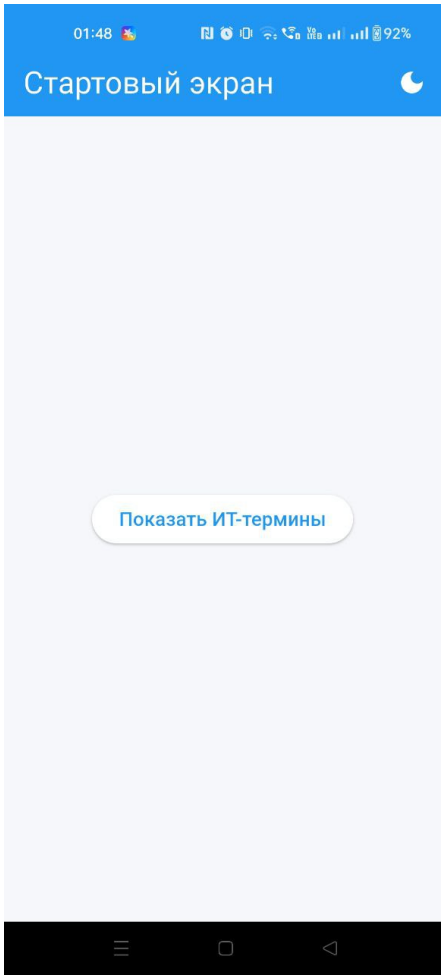
Для этого нужно создать новый файл, например, custom_theme.dart в директории lib/theme и определить в нем необходимые свойства, например:

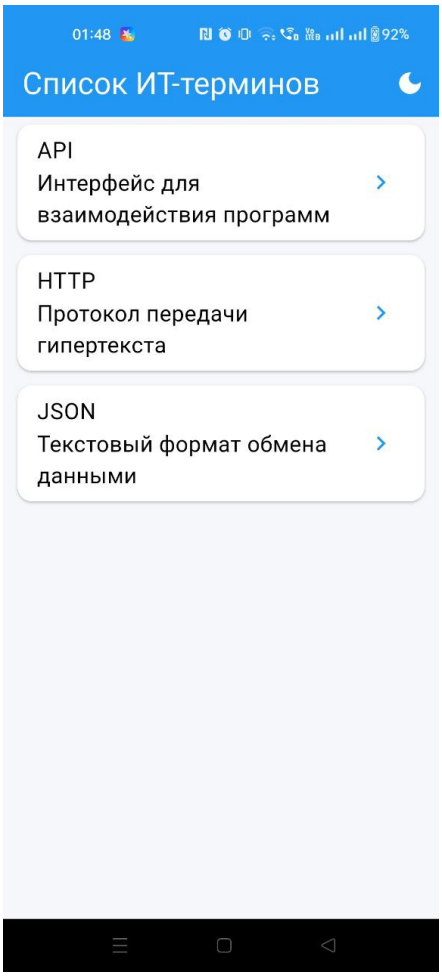
```dart
import 'package:flutter/material.dart';
import 'colors.dart';
class CustomTheme {
  static ThemeData get lightTheme {
    return ThemeData(
      primaryColor: CustomColors.purple,
      scaffoldBackgroundColor: Colors.white,
      fontFamily: 'Montserrat',
      buttonTheme: ButtonThemeData(
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(18.0)),
        buttonColor: CustomColors.lightPurple,
      )
    );}
}
```

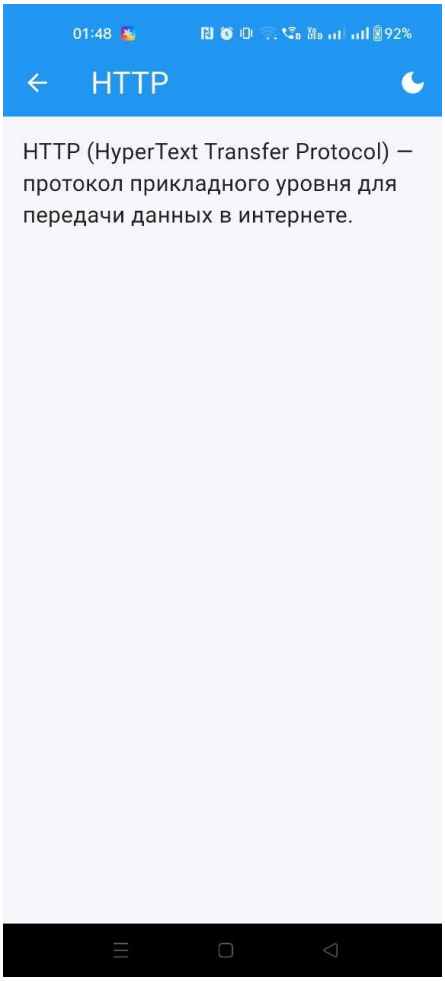
Применим созданную тему оформления к приложению в файле lib/main.dart:

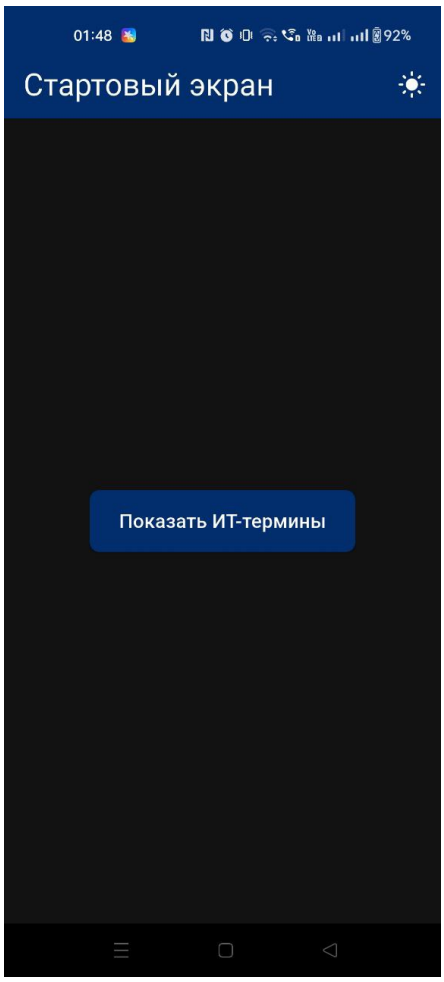
```dart
 Widget build(BuildContext context) {
    return MaterialApp(
      theme: CustomTheme.lightTheme,
    );
}
```

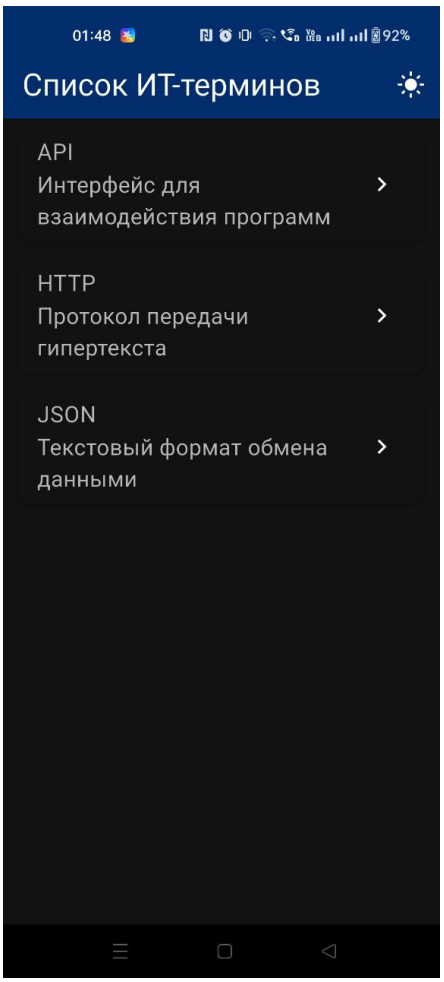
**Пример** приложения с управлением состоянием смены тем с помощью Cubit и настройкой кастомных светлой и темной тем.

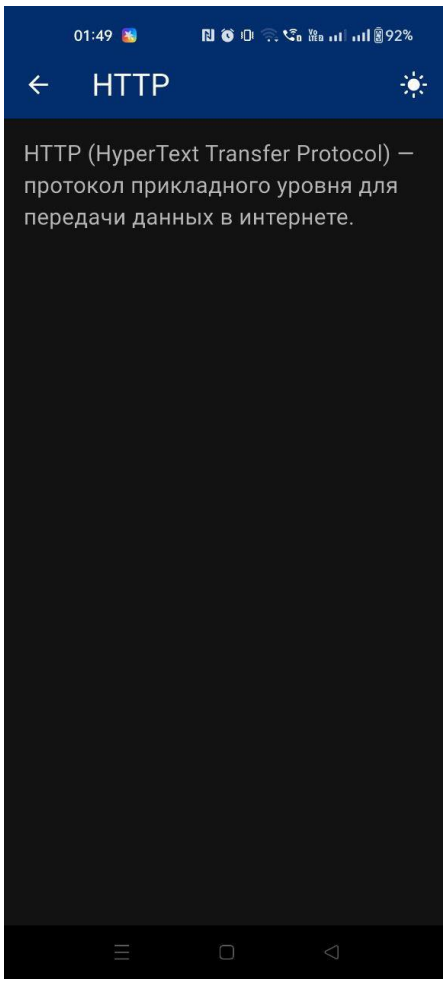


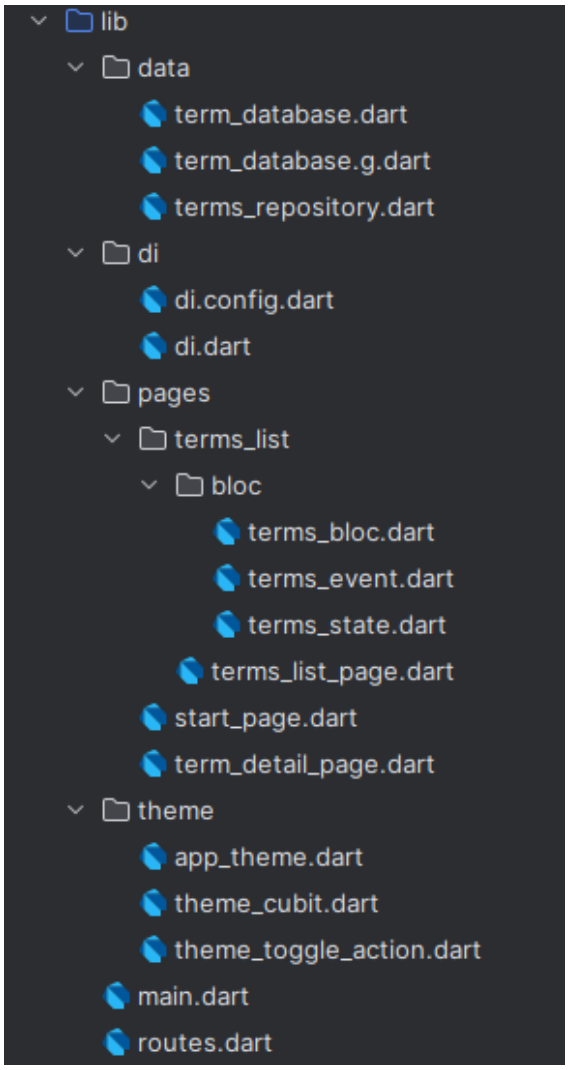



Выполнение:

1) Изменить структуру приложения, добавив файл theme/app_theme.dart в соответствии с рисунком:




2) Изменить программный код приложения, добавив необходимые файлы в приложение, описанное выше.

Файл с кастомными светлой и темной темами (lib/theme/app_theme.dart)

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

class AppTheme {
  // Светлая тема
  static final ThemeData light = ThemeData(
    brightness: Brightness.light,
    primaryColor: Colors.blue,
    scaffoldBackgroundColor: const Color(0xFFF5F7FA),
    appBarTheme: const AppBarTheme(
      backgroundColor: Colors.blue,
      foregroundColor: Colors.white,
      elevation: 0,
    ),
    cardColor: Colors.white,
    listTileTheme: const ListTileThemeData(
      iconColor: Colors.blue,
    ),
    colorScheme: const ColorScheme.light(
      primary: Colors.blue,
      secondary: Colors.blueAccent,
    ),
    textTheme: const TextTheme(
      bodyMedium: TextStyle(fontSize: 16, color: Colors.black87),
      titleMedium: TextStyle(fontSize: 18, fontWeight: FontWeight.w600),
    ),
  );
```

```
  // Тёмная тема
  static final ThemeData dark = ThemeData(
    brightness: Brightness.dark,

    // Основной тёмно-синий цвет
    primaryColor: const Color(0xFF0A2A43),
    // глубокий тёмный синий

    scaffoldBackgroundColor: const Color(0xFF121212),

    appBarTheme: const AppBarTheme(
      backgroundColor: Color(0xFF032E6D), // тёмно-синий AppBar
      foregroundColor: Colors.white,
      elevation: 0,
    ),

    // Карточки в тёмной теме
    cardColor: const Color(0xFF1E1E1E),

    // Иконки внутри ListTile — белые
    listTileTheme: const ListTileThemeData(
      iconColor: Colors.white, // ← белые иконки
      textColor: Colors.white70,
    ),

    colorScheme: const ColorScheme.dark(
      primary: Color(0xFF032E6D), // тёмно-синий
      secondary: Color(0xFF032E6D), // можно оставить таким же
    ),

    textTheme: const TextTheme(
      bodyMedium: TextStyle(fontSize: 16, color: Colors.white70),
      titleMedium: TextStyle(
        fontSize: 18,
        fontWeight: FontWeight.w600,
        color: Colors.white,
      ),
    ),
    elevatedButtonTheme: ElevatedButtonThemeData(
      style: ElevatedButton.styleFrom(
        backgroundColor: Color(0xFF032E6D), // тёмно-синяя кнопка
        foregroundColor: Colors.white,      // белый текст/иконки
        padding: EdgeInsets.symmetric(horizontal: 24, vertical: 14),
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.all(Radius.circular(8)),
        ),
      ),
    ),
  );
}
```

Подключаем кастомные темы в main.dart

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

import 'di/di.dart';
import 'routes.dart';
import 'pages/start_page.dart';
import 'pages/terms_list/terms_list_page.dart';
import 'pages/term_detail_page.dart';
import 'theme/theme_cubit.dart';
import 'theme/app_theme.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();

  // Настраиваем DI
  setupDI();
  runApp(const MyApp());
}
```

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

  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      // Получаем ThemeCubit из DI
      create: (_) => getIt<ThemeCubit>(),
      child: BlocBuilder<ThemeCubit, ThemeMode>(
        builder: (context, themeMode) {
          return MaterialApp(
            title: 'IT Термины (Drift + Theme)',
            debugShowCheckedModeBanner: false,
            theme: AppTheme.light,   // кастомная светлая тема
            darkTheme: AppTheme.dark,// кастомная тёмная тема
            themeMode: themeMode,       // сюда передаём состояние Cubit
            initialRoute: AppRoutes.start,
            routes: {
              AppRoutes.start: (_) => const StartPage(),
              AppRoutes.list: (_) => const TermsListPage(),
              AppRoutes.detail: (_) => const TermDetailPage(),
            },
          );
        },
      ),
    );
  }
}
```

### 3. Задание на практическую работу

Изменить приложение, разработанное в предыдущей практической работе, выполнив следующие действия:

1) Реализовать Cubit для управления темой приложения (светлая/темная)
2) Зарегистрировать Cubit в DI-контейнере
3) Выполнить настройку кастомных светлой и тёмной тем
4) Реализовать переключатели темы в AppBar или в настройках приложения
5) В отчете представить:

– цель работы;

– скриншот структуры проекта;

– скриншоты экранов приложения в светлой и темной теме;

– программный код приложения (класс Cubit, файлы с программным кодом для переключения тем, фрагменты кода UI-слоя с переключением тем, код main.dart).