**Лабораторна робота № 2**

*Тема:* Створення web-клієнта з використанням Flutter

<https://dou.ua/lenta/articles/flutter-first-steps/>

<https://medium.com/@lumeilin301/flutter-firebase-app-tutorial-part-1-get-started-95cce84939c3>

**Завдання**

Розробити додаток-клієнт для створеної RESTFull API. Визначити та реалізувати усі функціональні можливості, відповідно до API. На сторінках відображення списку основних даних передбачити можливість сортування, пошуку інформації.

**Теоретичні відомості**

**Flutter** - це набір засобів розробки від Google, який дозволяє спеціалістам створювати сучасні кросплатформні мобільні програми IOS та Android з мінімальними витратами часу. Програмістам достатньо створити єдину кодову базу, яка надалі компілюватиметься окремо на кожній ОС без будь-яких програмних доробок. Програми, створені на Flutter, мало чим поступаються нативним у плані продуктивності, при цьому їх розробка обходиться бізнесу значно дешевше. Це дає технології практично безмежні можливості розвитку, завдяки чому вона, незважаючи на відносну «молодість», успішно відвойовує частку ринку React Native. На Flutter було створено багато програмних продуктів великих корпорацій, таких як Google Ads, Alibaba, Ebay і Cryptomaniac, що говорить про спроможність і поширеність технології.

Попри поширене переконання, сам по собі Flutter не є мовою програмування, а для створення програмного забезпечення він використовує **Dart**. Це мова, що інтерпретується, яка також як і SDK була створена корпорацією Google. Реліз Dart відбувся ще восени 2011 року, проте суттєва популярність прийшла до нього дещо пізніше — у 2017 році, коли було анонсовано бета-версію Flutter. Вже в 2021 році в Google повідомили про майбутній вихід Flutter 2, через що інтерес до Dart піднявся на новий рівень. Сьогодні Dart активно використовується для розробки різних програмних продуктів, у тому числі серверних, десктопних, мобільних і веб-додатків. Відмінна риса мови полягає в тому, що написаний на ній код компілюється в байтовий код JAVA, і на машинному рівні фактично не відрізняється від нативного, що відкриває перед розробниками необмежені можливості.

**Хід роботи**

***Установка Flutter і створення базового додатка***

Посилання на покроковий туторіал зі встановлення на Windows:

<https://docs.flutter.dev/get-started/install/windows/mobile>

Посилання на покроковий тутторіал зі встановлення на Linux:

https://docs.flutter.dev/get-started/install/linux

Тут буде покроково описано як встановити на Widnows 10 (64-bit).

Для роботи Flutter має свої системні вимоги:



Для розробки Windows має мати таке встановлене ПЗ:

- [Git for Windows](https://gitforwindows.org/) (2.4 or later)

- [Android Studio](https://developer.android.com/studio/install) 2022.3 (Giraffe) or later to debug and compile Java or Kotlin code for Android. Flutter requires the full version of Android Studio.

Також потрібно налаштувати середовище розробки. Офіційною документацією Flutter рекомендується використовувати Visual Studio Code. Також після встановлення VS Code потрібно встановити відповідне [розширення](https://marketplace.visualstudio.com/items?itemName=Dart-Code.flutter) для розробки на Flutter.

Завантажте та встановіть Flutter

1. Завантажте наведений нижче інсталяційний пакет, щоб отримати останню версію стабільний випуск Flutter SDK.

[Flutter SDK 3.16.5](https://storage.googleapis.com/flutter_infra_release/releases/stable/windows/flutter_windows_3.16.5-stable.zip)

2. Створіть папку, куди можна встановити Flutter.

Розгляньте %USERPROFILE% або [C:\dev](../../../../../E:/C:/dev).

Не встановлюйте Flutter у каталог або шлях, який відповідає одній або двом з наступних умов:

* Шлях містить спеціальні символи або пробіли.
* Шлях вимагає підвищених привілеїв.   
  3. Перемістіть zip-файл у каталог, де ви хочете зберегти Flutter SDK.

4. Розпакуйте файл zip.

Після завершення Flutter SDK має бути в C:\dev\flutter або в іншій вибраній вами папці.

[Оновіть змінну Windows PATH](https://superuser.com/questions/949560/how-do-i-set-system-environment-variables-in-windows-10)

1. Натисніть Windows + S.
2. Введіть environment.
3. Коли відображається Редагувати змінні системного середовища як Найкращий збіг натисніть Відкрити під Відредагуйте змінні системного середовища.
4. Натисніть Про програму.
5. Натисніть Додаткові параметри системи.
6. Натисніть Змінні середовища…
7. Відобразиться діалогове вікно Змінні середовища.
8. У розділі Змінні користувача для <user> перевірте Шлях запис.
9. Якщо запис існує, натисніть Редагувати….
10. Якщо запис не існує, натисніть Новий….
11. Натисніть Новий.
12. Введіть <install-directory>\flutter\bin.
13. Натисніть запис <install-directory>\flutter\bin.
14. Натискайте Перемістити вгору, доки запис Flutter не опиниться вгорі списку.
15. Натисніть ОК.
16. Щоб увімкнути ці зміни, закрийте та знову відкрийте всі наявні командні рядки та екземпляри PowerShell.

Якщо ви встановили всі попередні умови та Flutter SDK, ви зможете почати розробку Flutter Windows для мобільних пристроїв.

Налаштуйте розробку під Android

* Запустіть Android Studio.
* Відобразиться діалогове вікно Ласкаво просимо до Android Studio.
* Дотримуйтеся вказівок майстра налаштування Android Studio.
* Встановіть такі компоненти:
* Платформа Android SDK, API 33.0.0
* Інструменти командного рядка Android SDK
* Інструменти створення Android SDK
* Інструменти платформи Android SDK
* Емулятор Android

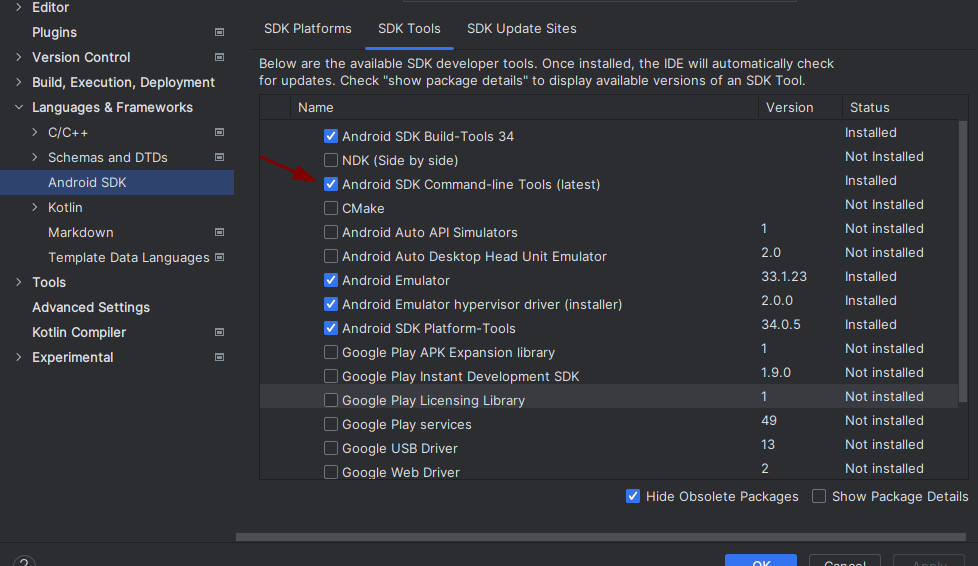
Далі, налаштуйте віртуальний або фізичний пристрій для розробки.  
Віртуальний: <https://docs.flutter.dev/get-started/install/windows/mobile?tab=virtual>

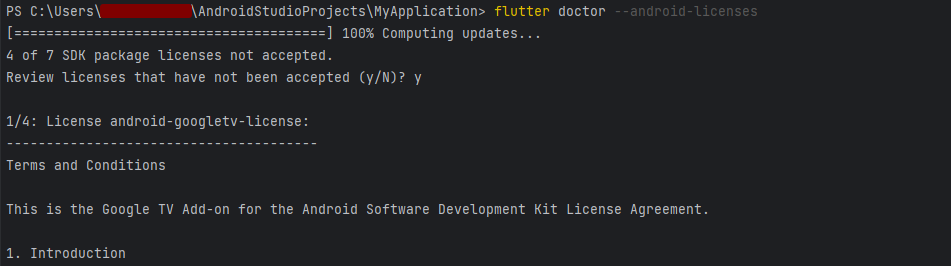
Фізичний: https://docs.flutter.dev/get-started/install/windows/mobile?tab=physical

Після налаштуваня погодьтеся з Android ліцензіями, за допомогою вікна консолі і такої команди:

flutter doctor --android-licenses







Переконайтеся, що на вашому пристрої встановлено Google Chrome браузер, оскільки розробка веб-додатків буде виконуватись лише в ньому(Так сказали дядіньки з гугла).

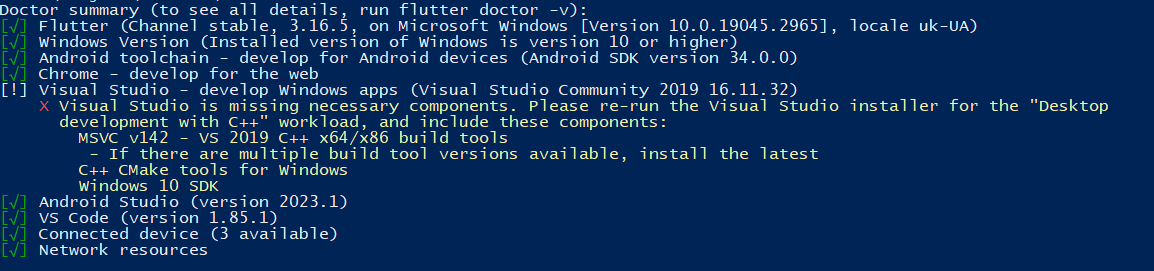
Після виконаних кроків, виконайте наступну команду в консолі:

flutter doctor

Якщо все правильно встановлено і немає ніяких проблем вивід буде приблизно такий:



Якщо виникли помилки:



Скористайтесь <https://stackoverflow.com/questions/71080518/visual-studio-not-installed-this-is-necessary-for-windows-development>

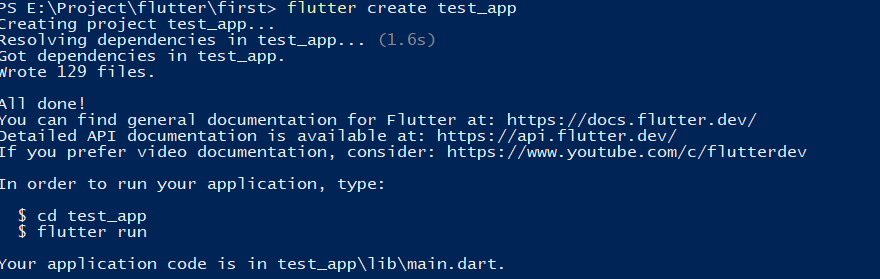
Якшо все дуже класно встановилося, то можна починати розробку. Для цього відкрийте будь-яку зручну для вас папку в VS Code, бажано таку, до якої не треба адмін. прав і в якій нічого немає, та напишіть команду в консоль, для того щоб створити базовий шаблон програми:

flutter create <app\_name>

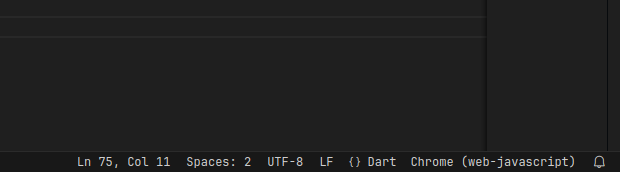
***Важливо!***

Назва папки має називатись за правилами **snake\_case**, тобто не мати великих літер та кожне слово має розділятись “\_” символом. Наприклад:

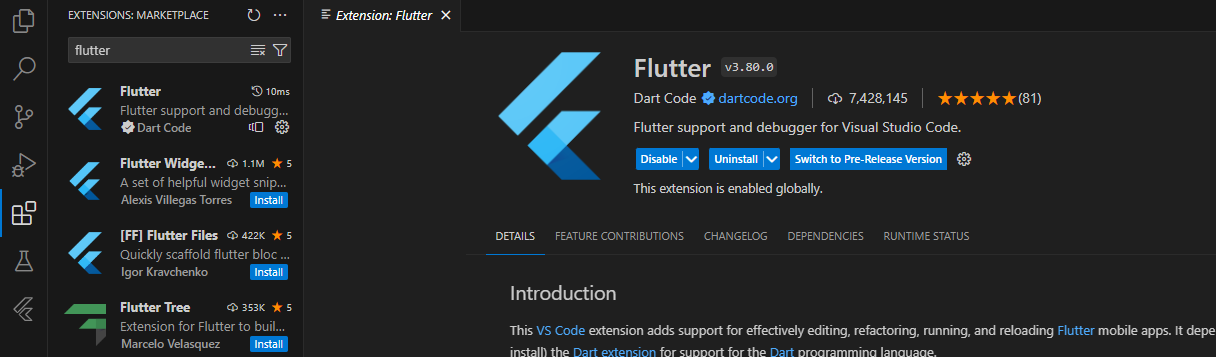
* my\_app
* web\_client
* web\_api
* myApp
* WebClient
* webApi



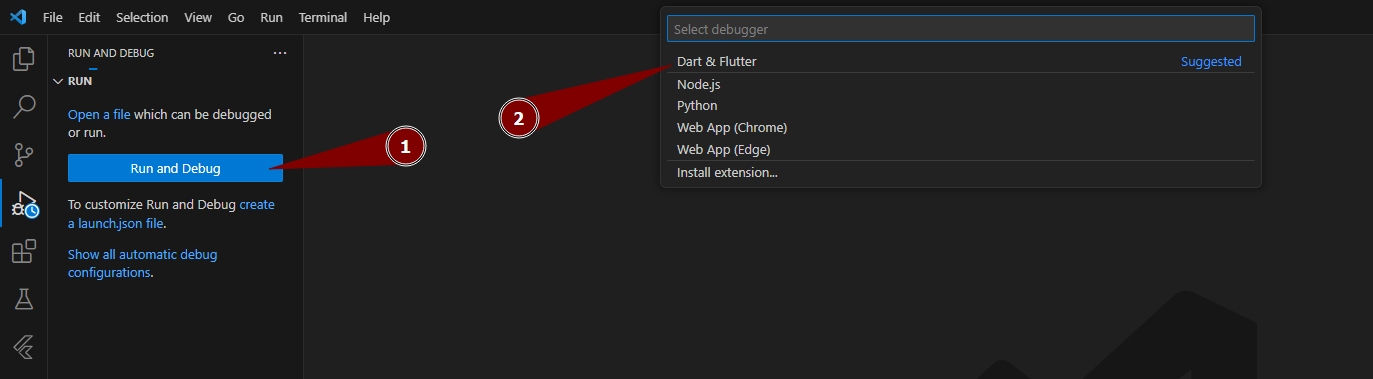
Для компіляції веб додатків потрібно вибрати відповідний пристрій (браузер) в VS Code на нижній панелі редактора, праворуч:

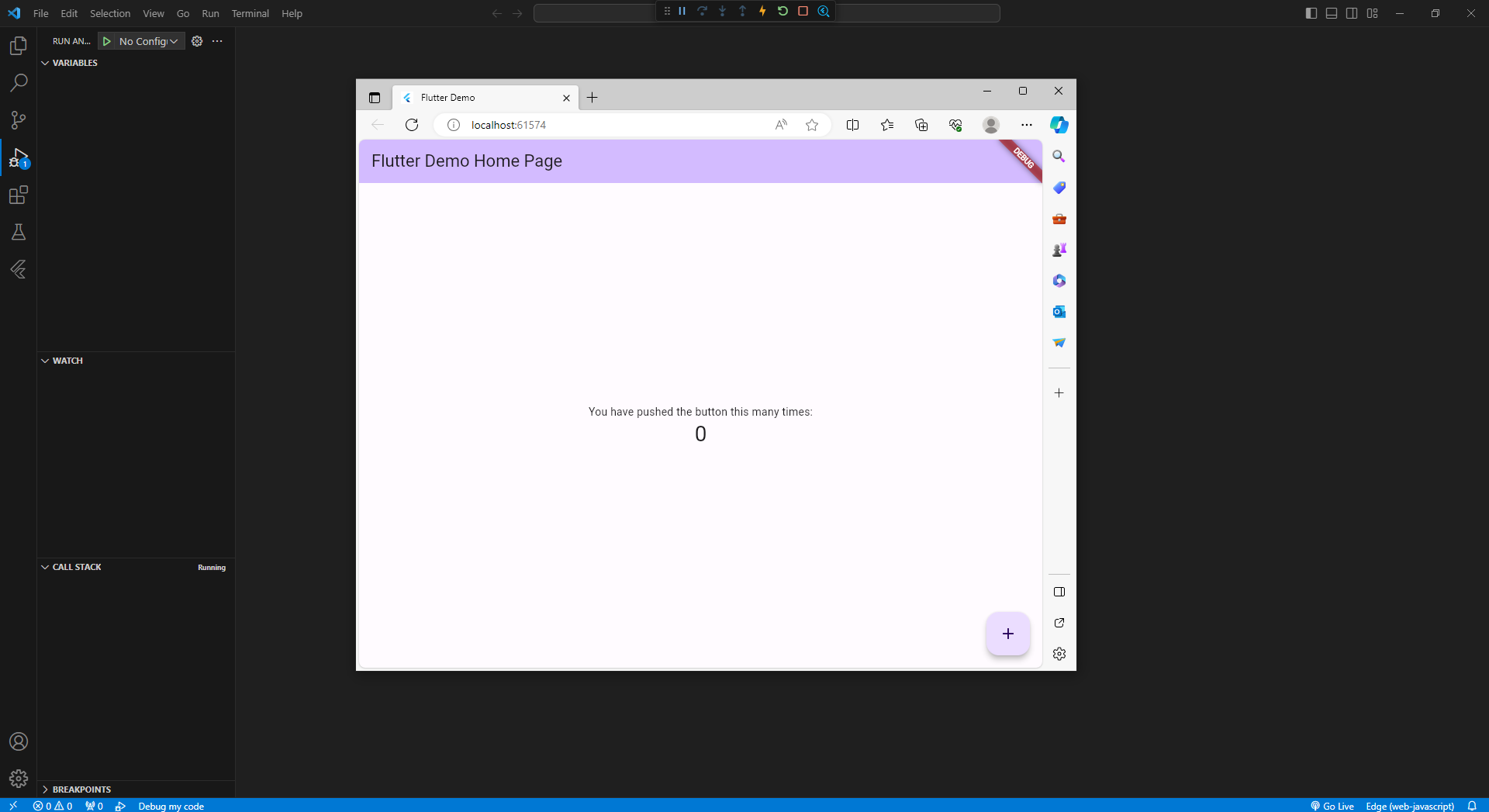


Попередньо встановивши розширення для IDE



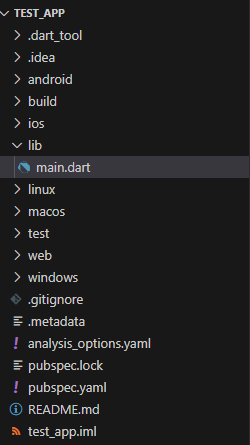
Пробуємо запустити проект:



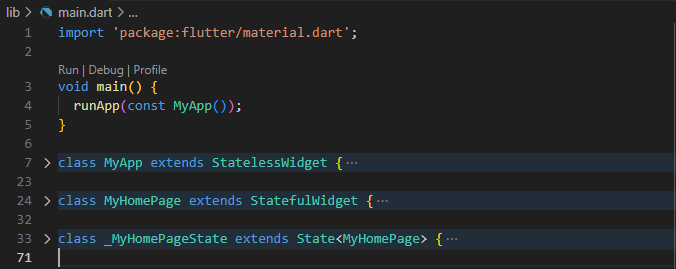


***Структура проекту в Flutter***

Створивши проект можна побачити наступну структуру, де нас буде цікавити файл проекту app\lib\main.dart.



Якщо відкрити цей файл, то можна побачити, що в ньому міститься весь вміст тестового проекту, тобто без використання структуризації, розділення компонентів по різних директоріях для кращого розуміння та можливості зручного використання.



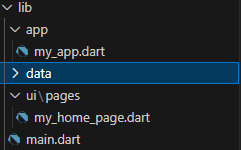
Для кращого використання можна розділити на чотири блоки:

**app/** - для класу створеного додатку

**ui/** - для створенних віджетів та сторінок

**data/** - для даних, можна також моделей

**main.dart** – точка входу в проект де буде створюватись власне додаток



Проте, потрібно пам’ятати що при розбиті коду на різні компоненти потрібно підключати файли для можливості використання вмісту через дерективу import. Наприклад:

import '../ui/pages/my\_home\_page.dart';

// або

import 'package:test\_app/ui/pages/my\_home\_page.dart';

Отже, отримаємо наступні компоненти:

**lib\main.dart**

import 'package:flutter/material.dart';

import 'app/my\_app.dart';

void main() {

  runApp(const MyApp());

}

**lib\app\my\_app.dart**

import 'package:flutter/material.dart';

import 'package:test\_app/ui/pages/my\_home\_page.dart';

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  // This widget is the root of your application.

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      title: 'Flutter Demo',

      theme: ThemeData(

        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),

        useMaterial3: true,

      ),

      home: const MyHomePage(title: 'Flutter Demo Home Page'),

    );

  }

}

В цьому фалі зазначити певні налаштування та вказали, що є сторінка home та задали компонент який відповідає за вміст сторінки.

**lib\ui\pages\my\_home\_page.dart**

import 'package:flutter/material.dart';

class MyHomePage extends StatefulWidget {

  const MyHomePage({super.key, required this.title});

  final String title;

  @override

  State<MyHomePage> createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

  int \_counter = 0;

  void \_incrementCounter() {

    setState(() {

      \_counter++;

    });

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        backgroundColor: Theme.of(context).colorScheme.inversePrimary,

        title: Text(widget.title),

      ),

      body: Center(

        child: Column(

          mainAxisAlignment: MainAxisAlignment.center,

          children: <Widget>[

            const Text(

              'You have pushed the button this many times:',

            ),

            Text(

              '$\_counter',

              style: Theme.of(context).textTheme.headlineMedium,

            ),

          ],

        ),

      ),

      floatingActionButton: FloatingActionButton(

        onPressed: \_incrementCounter,

        tooltip: 'Increment',

        child: const Icon(Icons.add),

      ), // This trailing comma makes auto-formatting nicer for build methods.

    );

  }

}

Для MyHomePage вказали, що буде використання стану, тобто класу, що відповідатиме за логіку роботи сторінки із відповідним вмістом.

Це як пропозиція яким чином можна взаємодіяти із проектом та яку структуру проекту зробити, але в залежності від складності структуру потрібно організовувати відповідно до проекту.

Розглянемо приклад розробки проекту(web\_client\_api) для роботи із АРІ.

***Створення компонентів в Flutter***

Писати на Flutter можна по різному. Проте, рекомендується розбивати код на компоненти, аби зменшити повторюваність коду та збільшити продуктивність додатка, оскільки чим більше різних компонентів, тим легше буде їх перемальовувати за потреби. Тому, напишемо декілька головних файлів для виконання додатка.

**main.dart**

import 'package:flutter/material.dart';

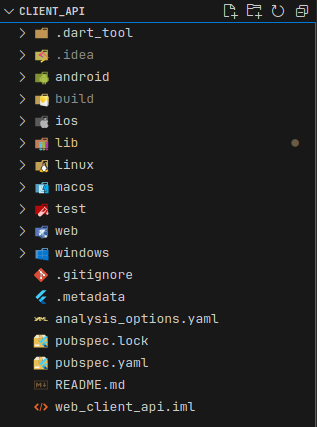
import 'package:web\_client\_api/presentation/pages/home\_page.dart';

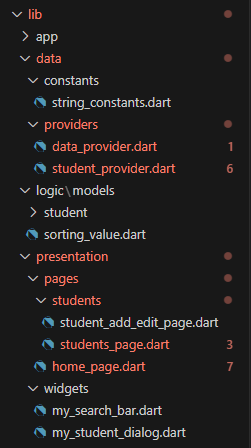
void main() {

runApp(const MyApp());

}

Початок виконання додатка починаєтьсяз функції **main в main.dart** файлі. main.dart має обов’язково знаходитись в кореневій папці виконуваного проекта, тобто в папці **lib.** Саме звідти розпочинається компіляція і в цій папці будуть створюватись усі файли програми. Початкова структура проекта:

Папка lib міститиме наступні вкладеності:



Для того, аби запустити наш додаток, потрібно викликати в цій функції метод **runApp**(), який в свою чергу приймає віджет, який і буде відображено при запуску.

Для цього створюємо віджет MyApp, який і буде описувати кореневу структуру додатка, тобто буде виступати в якості кореня дерева віджетів в програмі.

class MyApp extends StatelessWidget {

const MyApp({super.key});

@override

Widget build(BuildContext context) {

  return MaterialApp(

    title: 'Students api client',

    debugShowCheckedModeBanner: false,

    theme: ThemeData(

    colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),

      useMaterial3: true,

    ),

    home: const HomePage(),

  );

}

}

**home\_page.dart**

import 'package:flutter/material.dart';

import '../../logic/models/sorting\_value.dart';

import 'sessions/sessions\_page.dart';

import 'students/students\_page.dart';

import 'subjects/subjects\_page.dart';

class HomePage extends StatefulWidget {

const HomePage({Key? key}) : super(key: key);

@override

State<HomePage> createState() => \_HomePageState();

}

class \_HomePageState extends State<HomePage> {

int \_currentIndex = 0;

SortingValue \_sortingValue = SortingValue.asc;

final List<Widget> \_pages = [

  const StudentPage(),

  const SubjectPage(),

  const SessionPage(),

];

void \_onItemTapped(int index) {

  setState(() {

    \_currentIndex = index;

  });

}

void \_onSortPressed() {

setState(() {

  \_sortingValue = \_sortingValue == SortingValue.asc

    ? SortingValue.desc

    : SortingValue.asc;

});

\_pages[\_currentIndex] =

  \_updatePageSorting(\_pages[\_currentIndex], \_sortingValue);

}

Widget \_updatePageSorting(Widget page, SortingValue sortingValue) {

  if (page is StudentPage) {

    return StudentPage(sortingValue: sortingValue);

  } else if (page is SubjectPage) {

    return SubjectPage(sortingValue: sortingValue);

  } else if (page is SessionPage) {

    return SessionPage(sortingValue: sortingValue);

  }

  return page;

}

@override

Widget build(BuildContext context) {

  return Scaffold(

    appBar: AppBar(

      title: const Text('Students API Client'),

        actions: [

          IconButton(

            icon: const Icon(Icons.sort),

            onPressed: \_onSortPressed,

          ),

        ],

      ),

      body: \_pages[\_currentIndex],

      bottomNavigationBar: BottomNavigationBar(

        currentIndex: \_currentIndex,

        onTap: \_onItemTapped,

        items: const [

          BottomNavigationBarItem(

            icon: Icon(Icons.people),

            label: 'Students',

          ),

          BottomNavigationBarItem(

            icon: Icon(Icons.book),

            label: 'Subjects',

          ),

          BottomNavigationBarItem(

            icon: Icon(Icons.schedule),

            label: 'Sessions',

          ),

        ],

      ),

    );

  }

}

На цій сторінці описується головний “шаблон” додатка, куди через навігаційну панель будуть підставлятись різні сторінки з списками. Ви могли помітити, що ця сторінка наслідує **StatefulWidget**, а MyApp **StatelessWidget**. То яка ж між ними різниця?

Stateless Widget (Віджет без стану):

* Пояснення: Це, як картина чи плакат, які завжди залишаються незмінними. Наприклад, якщо ви малюєте малюнок з зображенням гори, то вона завжди виглядатиме однаково.
* Приклад: Якщо у вас є віджет, який відображає лого вашого класу чи літаючого метелика, і це лого завжди залишається однаковим, то це може бути Stateless віджет.

Stateful Widget (Віджет із станом):

* Пояснення: Це, як тваринка, яка може реагувати на різні події чи зміни. Наприклад, якщо ви малюєте картину, де ведмідь може пританцьовувати, якщо йому весело, або сидіти, якщо він сумний. Стан визначає його настрій та дії.
* Приклад: Якщо ви створюєте віджет термометра, який показує температуру, і ця температура може змінюватися, то це може бути Stateful віджет.

Тому що, HomePage буде змінювати свій вміст на різні сторінки, вона повинна наслідувати **StatefulWidget.** Сам стан сторінки ми змінюємо за допомогою метода **setState()**, який є вбудованим в клас State. SetState сповіщає про те, що потрібно оновити стан віджета, тобто перемалювати його. Наприклад:

void \_onItemTapped(int index) {

  setState(() {

    \_currentIndex = index;

  });

}

В цьому методі при натиснення на елемент навігаційної панелі setState змінює локальне поле \_currentIndex і після цього перемальовує весь HomePage віджет. Саме тому, сторінки в додатку змінюються і переміщаються між собою.

***Створення класів для роботи з API***

Для того щоб працювати з RESTFull API, потрібно працювати з http запитами. Для цього потрібно до проекта додати відповідний пакет. Виконаємо таку консольну команду:

flutter pub add http

Ця команда додасть пакет до файлу **pubspec.yaml**. В цьому файлі під тегом **dependencies** описані усі пакети які будуть використовуватись під час виконання проекта. Для цього проекта рекомендуєтся використовувати такі пакети:

dependencies:

flutter:

sdk: flutter

http: ^1.1.2

json\_annotation: ^4.8.1

json\_serializable: ^6.7.1

build\_runner: ^2.4.7

logger: ^2.0.2+1

***Важливо!***

Якщо редагуєте файл pubspec.yaml вручну, після редагування обов’язково його збережіть та виконайте команду в консолі:

flutter pub get

Або додати через термінал:

flutter pub add json\_annotation

flutter pub add build\_runner

…

Після додавання пакету, можна починати розробку класу для роботи з апі. Перед цим напишемо абстрактний клас, який буде виступати інтерфейсом для отримання даних.

import '../../logic/models/sorting\_value.dart';

abstract class DataProvider<T> {

const DataProvider();

Future<List<T>?> getAll({

String? searchInput,

SortingValue? sortingInput,

});

Future<T?> getById(int id);

Future<T?> create(T data);

Future<T?> update(int id, T newData);

Future<String> delete(int id);

}

Після написання цього класу, можна написати клас який буде його наслідувати та реалізовувати його методи. Цей клас приймає об’єкт **SortingValue**  як параметр. SortingValue це звичайний перерахований тип, який описано як:  
enum SortingValue { asc, desc }

**student\_provider.dart**

import 'dart:convert';

import 'package:web\_client\_api/data/constants/string\_constants.dart';

import 'package:web\_client\_api/data/providers/data\_provider.dart';

import 'package:web\_client\_api/logic/models/student/student.dart';

import 'package:http/http.dart' as http;

import '../../logic/models/sorting\_value.dart';

class StudentProvider implements DataProvider<Student> {

  final String link = "${hostname}students";

  const StudentProvider();

  @override

  Future<Student?> create(Student data) async {

    final response = await http.post(

      Uri.parse(link),

      body: jsonEncode(data.toJson()),

      headers: <String, String>{

      'Content-Type': 'application/json; charset=UTF-8',

      },

  );

  if (response.statusCode == 201) {

    return Student.fromJson(jsonDecode(response.body));

  }

  return null;

}

@override

Future<String> delete(int id) async {

  final response = await http.delete(Uri.parse("$link/$id"));

  if (response.statusCode == 200) {

    return "Student deleted successfully";flutter pub get

  }

  if (response.statusCode == 404) {

    return "Student not found";

  }

  return "Error while deleting";

}

@override

Future<List<Student>?> getAll({

  String? searchInput,

  SortingValue? sortingInput,

}) async {

  final Map<String, String> queryParams = {};

  if (sortingInput != null) {

    queryParams['\_sort'] = 'firstName';

    queryParams['\_order'] = sortingInput == SortingValue.asc ? 'asc' : 'desc';

  }

  if (searchInput != null && searchInput.isNotEmpty) {

    queryParams['firstName\_like'] = searchInput;

  }

  final Uri uri = Uri.parse(link).replace(queryParameters: queryParams);

  final response = await http.get(uri);

  if (response.statusCode == 200) {

    List<dynamic> data = jsonDecode(response.body);

    List<Student> students =

    data.map((json) => Student.fromJson(json)).toList();

    return students;

  }

  return null;

}

@override

Future<Student?> getById(int id) async {

  final response = await http.get(Uri.parse("$link/$id"));

  if (response.statusCode == 200) {

    return Student.fromJson(jsonDecode(response.body));

  }

  return null;

}

@override

Future<Student?> update(int id, Student newData) async {

  final response = await http.put(

    Uri.parse("$link/$id"),

    body: jsonEncode(newData.toJson()),

    headers: <String, String>{

    'Content-Type': 'application/json; charset=UTF-8',

    },

  );

  if (response.statusCode == 200) {

    return Student.fromJson(jsonDecode(response.body));

  }

  return null;

}

}

В цьому класі реалізовується абстрактний клас DataProvider та виконується робота з API.

Цей клас має поле **link.** Воно являє собою рядок, в який інтерпольовано константу з файлу **string\_constants.dart.** Ось його вміст:  
// Цю адресу використоувати для компіляції під десктоп чи веб.

// 0.0.0.0 використовуєтья для того, щоб можна було зробити запит на дані з телефону.

const hostname = "http://0.0.0.0:3000/";

// Цю адресу використовувати для запитів на апі коли компілюєте під фізичний девайс. Наприклад під Android телефон через USB кабель.

// 192.168.0.109 потрібно змінити на адресу ноута, чи пк з якого компілюєте

const mobileHostname = "http://192.168.0.109:3000/";

Також, слід зазначити, що набагато зручніше працювати з даними використовуючи моделі класів. Саме тому створимо модель для поточного провайдера.

import 'package:json\_annotation/json\_annotation.dart';

part 'student.g.dart';

@JsonSerializable()

class Student {

  int? id;

  String firstName;

  String lastName;

  Student({

    this.id,

    required this.firstName,

    required this.lastName,

  });

  factory Student.fromJson(Map<String, dynamic> json) =>

  \_$StudentFromJson(json);

  Map<String, dynamic> toJson() => \_$StudentToJson(this);

  Student copyWith({

    int? id,

    String? firstName,

    String? lastName,

  }) {

  return Student(

    id: id ?? this.id,

    firstName: firstName ?? this.firstName,

    lastName: lastName ?? this.lastName,

  );

  }

}

Ця модель анотується методом **JsonSerializable()** з відповідного пакета. Це дозволяє пришвидшити розробку, оскільки після виконання команди

dart run build\_runner build

пакет сам згенерує методи для сереалізації та десереалізації об’єкта класа в **JSON.** Також, для вдалої генерації потрібно описати наступний рядок:  
part 'student.g.dart';

в файлі з моделлю. Цей рядок вказує на назву файлу, який потрібно згенерувати.

Вміст цього файлу буде приблизно таким, залежно від моделі:  
// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'student.dart';

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// JsonSerializableGenerator

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Student \_$StudentFromJson(Map<String, dynamic> json) => Student(

  id: json['id'] as int?,

  firstName: json['firstName'] as String,

  lastName: json['lastName'] as String,

);

Map<String, dynamic> \_$StudentToJson(Student instance) => <String, dynamic>{

  'id': instance.id,

  'firstName': instance.firstName,

  'lastName': instance.lastName,

};

***Створення сторінки з списком***

Створимо одну з сторінок для виведення списку даних, які було отримано з API. Для цого використовуємо раніше створений провайдер для студентів. Створимо StatefulWidget.

import 'package:flutter/material.dart';

import 'package:web\_client\_api/data/providers/student\_provider.dart';

import 'package:web\_client\_api/logic/models/sorting\_value.dart';

import 'package:web\_client\_api/logic/models/student/student.dart';

import 'package:web\_client\_api/presentation/pages/students/student\_add\_edit\_page.dart';

import '../../widgets/my\_search\_bar.dart';

import '../../widgets/my\_student\_dialog.dart';

class StudentPage extends StatefulWidget {

  const StudentPage({super.key, this.sortingValue = SortingValue.asc});

  final SortingValue sortingValue;

  @override

  State<StudentPage> createState() => \_StudentPageState();

}

class \_StudentPageState extends State<StudentPage> {

  final \_repository = const StudentProvider();

  final \_searchController = TextEditingController();

  String \_searchQuery = "";

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      body: Column(

        children: [

          MySearchBar(

            searchController: \_searchController,

            onSearched: () {

              setState(() {

                \_searchQuery = \_searchController.text;

              });

            },

            hintText: 'Student name...',

          ),

          FutureBuilder<List<Student>?>(

            future: \_repository.getAll(

              sortingInput: widget.sortingValue,

              searchInput: \_searchQuery,

            ),

            builder: (context, snapshot) {

              if (snapshot.connectionState == ConnectionState.waiting) {

                return const Center(child: CircularProgressIndicator());

              } else if (snapshot.hasError) {

                return Center(child: Text('Error: ${snapshot.error}'));

              } else if (!snapshot.hasData || snapshot.data!.isEmpty) {

                return const Center(child: Text('No students found.'));

              } else {

                List<Student> students = snapshot.data!;

                return Expanded(

                  child: ListView.builder(

                    itemCount: students.length,

                    itemBuilder: (context, index) {

                      final item = students[index];

                      return ListTile(

                        title: Text(

                          '${item.firstName} ${item.lastName}',

                        ),

                        subtitle: Text('ID: ${item.id}'),

                        trailing: Row(

                          mainAxisSize: MainAxisSize.min,

                          children: [

                            IconButton(

                              icon: const Icon(Icons.visibility\_outlined),

                              onPressed: () {

                                showAdaptiveDialog(

                                  context: context,

                                  builder: (context) =>

                                      MyStudentDialog(id: item.id!),

                                );

                              },

                            ),

                            IconButton(

                              icon: const Icon(Icons.edit),

                              onPressed: () async {

                                final result = await Navigator.of(context)

                                    .push<Student?>(MaterialPageRoute(

                                  builder: (ctx) {

                                    return StudentAddEditPage(

                                      studentToEdit: item,

                                    );

                                  },

                                ));

                                setState(() {

                                  if (result != null) {

                                    ScaffoldMessenger.of(context).showSnackBar(

                                        SnackBar(

                                            content: Text(

                                                'Subject ${result.id} updated')));

                                  }

                                });

                              },

                            ),

                            IconButton(

                              icon: const Icon(Icons.delete),

                              onPressed: () async {

                                final response =

                                    await \_repository.delete(item.id!);

                                setState(() {

                                  ScaffoldMessenger.of(context)

                                      .clearSnackBars();

                                  ScaffoldMessenger.of(context).showSnackBar(

                                    SnackBar(content: Text(response)),

                                  );

                                });

                              },

                            ),

                          ],

                        ),

                      );

                    },

                  ),

                );

              }

            },

          ),

        ],

      ),

      floatingActionButton: FloatingActionButton(

        onPressed: () async {

          final result =

              await Navigator.of(context).push<Student?>(MaterialPageRoute(

            builder: (context) => const StudentAddEditPage(),

          ));

          setState(() {

            if (result != null) {

              ScaffoldMessenger.of(context).showSnackBar(SnackBar(

                  content: Text(

                      'Student ${result.firstName} ${result.lastName} created')));

            }

          });

        },

        child: const Icon(Icons.add),

      ),

    );

  }

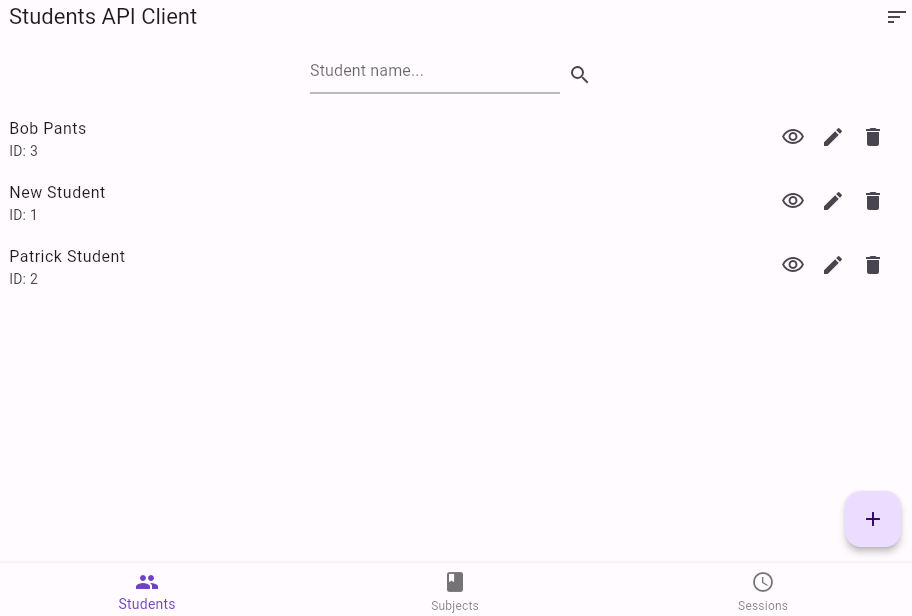
}

Через конструктор віджета з головної сторінки передається значенння сортування, за яким воно буде відбуватись. В класі стану описано поле для роботи з провайдером, контроллер текстового поля та рядкова змінна для пошуку даних в списку.

final \_repository = const StudentProvider();

final \_searchController = TextEditingController();

String \_searchQuery = "";

  
Далі, створюється саме тіло сторінки. Слід зауважити, для виведення даних використовується клас **FutureBuilder**. За його допомоги, можна виводити різні віджети в залежності від того отримано дані з апі чи ні. До прикладу, на цій сторінці поки ми очікуємо на отримання виводиться завантажувальне кільце, а коли дані отримано автоматично змалюється список для кожного елемента списку студентів.

Також, на цій сторінці використовуються авторські компоненти, **MySearchBar** і **MyStudentDialog**. Ось такий вміст відповідних файлів цих віджетів:

import 'package:flutter/material.dart';

class MySearchBar extends StatelessWidget {

  const MySearchBar({

    super.key,

    required this.searchController,

    required this.onSearched,

    this.hintText,

  });

  final TextEditingController searchController;

  final VoidCallback onSearched;

  final String? hintText;

  @override

  Widget build(BuildContext context) {

    return Row(

      mainAxisAlignment: MainAxisAlignment.center,

      children: [

        SizedBox(

          width: 250,

          height: 60,

          child: TextField(

            onSubmitted: (value) => onSearched(),

            decoration: InputDecoration(hintText: hintText),

            controller: searchController,

          ),

        ),

        IconButton(

          onPressed: onSearched,

          icon: const Icon(Icons.search),

        ),

      ],

    );

  }

}

Цей компонент являє собою поле для пошуку, в який переноситься текст для підказки на текстовому полі, текстовий контроллер для текстового поля та метод, який буде виконуватись при натиску на кнопку пошуку.  
Цей компонент можна використовувати багаторазово. За таким ж принципом можна створювати і інші сторінки та компоненти.

import 'package:flutter/material.dart';

import 'package:web\_client\_api/data/providers/student\_provider.dart';

import '../../logic/models/student/student.dart';

class MyStudentDialog extends StatefulWidget {

const MyStudentDialog({Key? key, required this.id}) : super(key: key);

final int id;

@override

State<MyStudentDialog> createState() => \_MyStudentDialogState();

}

class \_MyStudentDialogState extends State<MyStudentDialog> {

final \_repository = const StudentProvider();

@override

Widget build(BuildContext context) {

return Center(

child: Container(

padding: const EdgeInsets.all(16.0),

decoration: BoxDecoration(

color: Colors.white,

borderRadius: BorderRadius.circular(16),

),

constraints: const BoxConstraints.tightFor(width: 200, height: 150),

child: FutureBuilder(

future: \_repository.getById(widget.id),

builder: (context, AsyncSnapshot<Student?> snapshot) {

if (snapshot.connectionState == ConnectionState.done) {

if (snapshot.hasData) {

final Student student = snapshot.data!;

return Column(

mainAxisAlignment: MainAxisAlignment.start,

crossAxisAlignment: CrossAxisAlignment.start,

children: [

const Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text(

'Student',

style: TextStyle(

fontWeight: FontWeight.bold, fontSize: 16),

),

],

),

Text('ID: ${student.id}'),

Text('First name: ${student.firstName}'),

Text('Last name: ${student.lastName}'),

const Spacer(),

Row(

mainAxisAlignment: MainAxisAlignment.end,

children: [

TextButton(

onPressed: () => Navigator.of(context).pop(),

child: const Text('OK'),

)

],

),

],

);

} else if (snapshot.hasError) {

return Text('Error: ${snapshot.error}');

} else {

return const Text('Student not found.');

}

} else {

return const CircularProgressIndicator();

}

},

),

),

);

}

}

Цей компонент являє собою об’єкт діалогового вікна, яке буде показано при натиснені на відповідну кнопку на сторінці з списком студентів.

***Створення сторінки для додавання/редагування елементів списку***

Також, обов’язково потрібно створити сторінку для додавання чи редагування існуючих елементів списку. Створимо таку сторінку для студентів:

import 'package:flutter/material.dart';

import 'package:web\_client\_api/logic/models/student/student.dart';

import '../../../data/providers/student\_provider.dart';

class StudentAddEditPage extends StatefulWidget {

  const StudentAddEditPage({Key? key, this.studentToEdit}) : super(key: key);

  final Student? studentToEdit;

  @override

// ignore: library\_private\_types\_in\_public\_api

  \_StudentAddEditPageState createState() => \_StudentAddEditPageState();

}

class \_StudentAddEditPageState extends State<StudentAddEditPage> {

  late TextEditingController \_firstNameController;

  late TextEditingController \_lastNameController;

  final \_repository = const StudentProvider();

  @override

  void initState() {

    super.initState();

    \_firstNameController =

        TextEditingController(text: widget.studentToEdit?.firstName ?? '');

    \_lastNameController =

        TextEditingController(text: widget.studentToEdit?.lastName ?? '');

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text(widget.studentToEdit == null

            ? 'Add Student'

            : 'Edit Student № ${widget.studentToEdit!.id}'),

      ),

      body: Padding(

        padding: const EdgeInsets.all(16.0),

        child: Column(

          children: [

            TextField(

              controller: \_firstNameController,

              decoration: const InputDecoration(labelText: 'First Name'),

            ),

            TextField(

              controller: \_lastNameController,

              decoration: const InputDecoration(labelText: 'Last Name'),

            ),

            const SizedBox(height: 16),

            ElevatedButton(

              onPressed: () async {

                if (widget.studentToEdit == null) {

                  final newStudent = Student(

                    firstName: \_firstNameController.text,

                    lastName: \_lastNameController.text,

                  );

                  await \_repository.create(newStudent);

                } else {

                  final updatedStudent = widget.studentToEdit!.copyWith(

                    firstName: \_firstNameController.text,

                    lastName: \_lastNameController.text,

                  );

                  await \_repository.update(updatedStudent.id!, updatedStudent);

                }

// ignore: use\_build\_context\_synchronously

                Navigator.of(context).pop();

              },

              child:

                  Text(widget.studentToEdit == null ? 'Add' : 'Save Changes'),

            ),

          ],

        ),

      ),

    );

  }

}

На цій сторінці можна як створити так і редагувати елемент. Все залежить від того, чи буде передано існуючий елемент в якості параметра для віджета.

Для того, аби показати цю сторінку, використаємо клас **Navigator**. За його допомогою можна змінювати поточні сторінки в контексті та пересуватись між ними. Тому, на сторінці StudentsPage виконаємо такий метод при натисненні на кнопку додавання:

floatingActionButton: FloatingActionButton(

  onPressed: () async {

  final result =

  await Navigator.of(context).push<Student?>(MaterialPageRoute(

    builder: (context) => const StudentAddEditPage(),

  ));

  setState(() {

    if (result != null) {

      ScaffoldMessenger.of(context).showSnackBar(SnackBar(

      content: Text(

      'Student ${result.firstName} ${result.lastName} created')));

    }

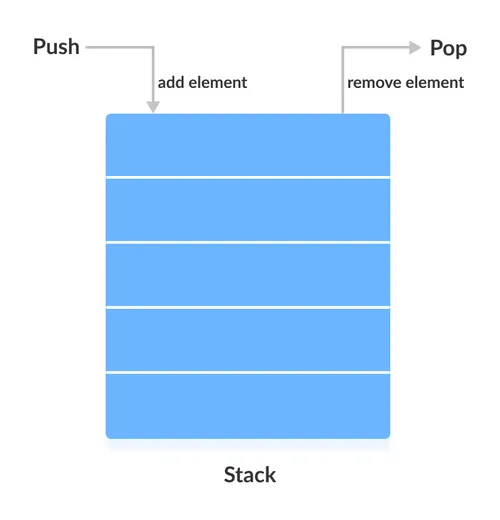
  });

  },

  child: const Icon(Icons.add),

),

Тут використовується метод **push**. За його допомогою сторінка, яка передається як параметр метода заноситься в стек сторінок, поверх поточної.



На цій сторінці розміщено поля для введеня даних для моделі студента. Після додавання/збереження викликаємо метод **pop**. Цей метод, як показано на малюнку, видаляє поточну сторінку зі стеку, та переносить на попередню.

Navigator.of(context).pop(newStudent);

Аргументом цього методу є об’єкт студента, який було створено. Цей об’єкт буде отримано на сторінці StudentsPage коли сторінка додавання закриється.

Кінцевий вигляд сторінок:

