

PHẦN 1: NGÔN NGỮ DART

Mục tiêu: Làm quen với ngôn ngữ Dart, cú pháp, kiểu dữ liệu nullable, lập trình bất đồng bộ.

Công cụ: IDE Online <https://dartpad.dev/?>, Android studio

Bài tập 1: Viết chương trình Hello World để kiểm tra IDE

Bài tập 2:

- Khai báo lớp SinhVien bao gồm các thuộc tính:

+ id, ten: bắt buộc.

+ ngaySinh, que_Quan: Không bắt buộc

- Yêu cầu:

1. Viết hàm khởi tạo SinhVien cho các thuộc tính đã khai báo (Sử dụng named parameter).

2. Ghi đè hàm toString sẵn có để trả về thông tin của một sinh viên.

3. Viết lớp QL_SinhVien để quản lý một danh sách sinh viên với các phương thức add, edit, delete với đối số là một SinhVien.

Bài tập 3:

1. Viết hàm để kiểm tra một danh sách có chứa các phần tử giống nhau hay không (để đơn giản, có thể sử dụng danh sách số nguyên).

2. Nghịch lý Ngày sinh nhật (Birthday Paradox) được phát biểu như sau: “Trong một nhóm có 23 người bất kỳ, xác suất để có hai người có cùng một ngày sinh nhật là không ít hơn $\frac{1}{2}$ ”.

Hãy kiểm tra lại giả thuyết trên bằng thực nghiệm.

Bài tập 4:

1. Sử dụng phương thức map của List để chuyển một danh sách các chuỗi thành danh sách chiều dài của các chuỗi đó.

2. Sử dụng phương thức forEach của List để in ra bình phương các phần tử của một danh sách.

Bài tập 5: Future Type

1. Viết hàm lateNum trả về một số nguyên ngẫu nhiên sau khi thực hiện lời gọi hàm 1s.

2. Kiểm tra số trả về bởi hàm lateNum là chẵn hay lẻ (in kết quả ra màn hình).

Hướng dẫn:

1.

- Google keyword: “Generate random numbers in Dart”
- Hàm `lateNum` có thể viết như sau:

```
import 'dart:math';  
Future<int> lateNum(){  
  var one = Duration(seconds: 1);  
  return Future.delayed(one).then(  
    (value) => Random().nextInt(1000),  
  );  
}
```

Bài tập 6: Stream, async, async*, await

- Viết hàm trả về một `Stream<int>` gồm 10 số nguyên.
- Viết hàm tính tổng 10 số nguyên của Stream

Bài tập 7: Stream, async, await, StreamController

- Viết lớp `MyStream<T>`

```
class MyStream<T>{  
  StreamController<T> _streamController = StreamController();  
  Stream<T> get stream => _streamController.stream;  
  void addEvent(T event){  
    _streamController.sink.add(event);  
  }  
  void dispose(){  
    _streamController.close();  
  }  
}
```

- Tính tổng 10 số nguyên (ứng với 10 Event) do một đối tượng thuộc lớp `MyStream` sinh ra.

Bài tập 8

Viết hàm nhận vào một chuỗi, trả về tần số xuất hiện của từng từ trong chuỗi đó
Hướng dẫn:

- Sử dụng Map để chứa tần số đếm của các từ `Map<String, int> outputMap = {}`
- Chia chuỗi đã cho thành các từ bằng phương thức `split` trên chuỗi:

```
List<String> words = sentence.toLowerCase().split(RegExp(r"[\s,]+"));
```

- Duyệt danh sách từ thu được để đếm:

```
for (String word in words) {  
  outputMap[word] = (outputMap[word] ?? 0) + 1;  
}
```

}

Bài tập 9

Đọc hiểu và làm các bài tập ở trang sau:

<https://gist.github.com/skdblog1999/38ef784799f4975050fa2b52c9a8b95b>

PHẦN 2: FLUTTER CƠ BẢN

Bài tập 1:

Sinh viên chú ý: Cần thực hiện bài tập này ở nhà, thời gian cài đặt chừng 1 buổi nếu mạng tốt, không thể thực hiện trên lớp.

Bài tập này rất quan trọng và nó ảnh hưởng đến toàn bộ các bài tập khác trong môn học, đề nghị các bạn sinh viên nghiêm túc thực hiện đúng theo thời gian qui định.

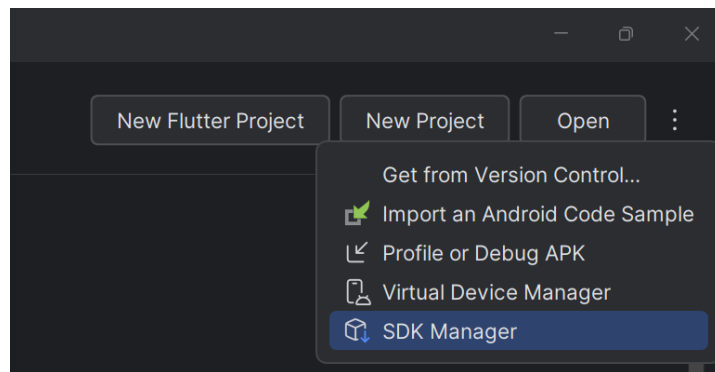
Bắt buộc sinh viên phải thực hiện bài tập này ở nhà

Nếu máy tính cấu hình yếu, ram dưới 8GB có thể không thực hiện được việc cài đặt này.

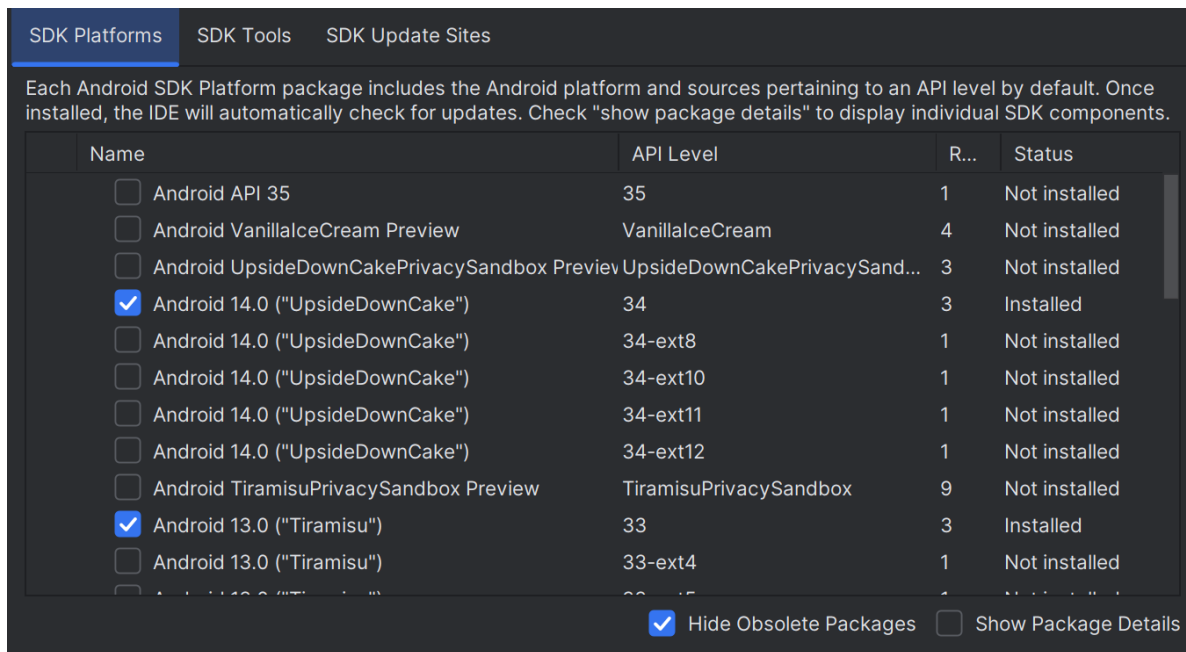
Sinh viên cài đặt các thành phần sau đây (Chú ý các đường dẫn không được đặt tên bằng tiếng Việt có dấu, không chừa khoảng trắng khi trong các đường dẫn, tên thư mục. SV chú ý: Cài đặt theo đúng thư mục đã hướng dẫn, không được cài đặt theo thư mục mặc định.

1. Cài đặt Android Studio theo đường dẫn: C:\Android\AndroidStudio
2. (Bước tốn thời gian nhất) Chạy Android Studio và cài đặt thư viện:

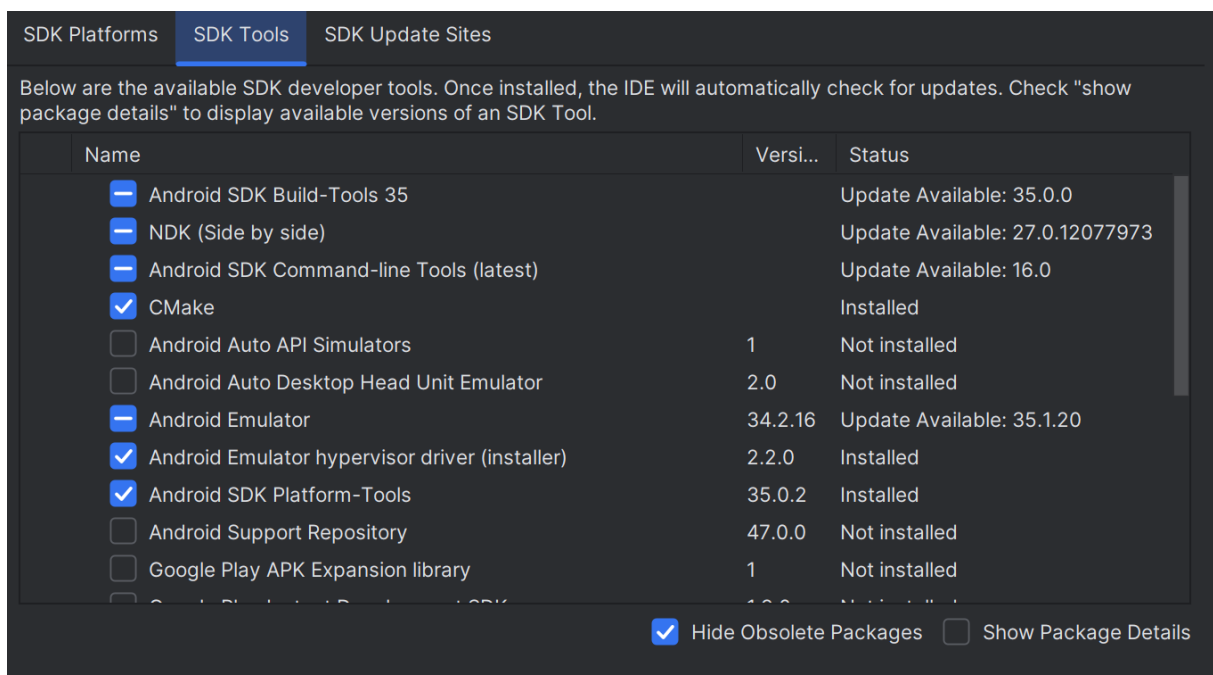
Vào mục Setting (để ý dấu ba chấm ở góc trên bên phải hay trái), tìm SDK Manager:

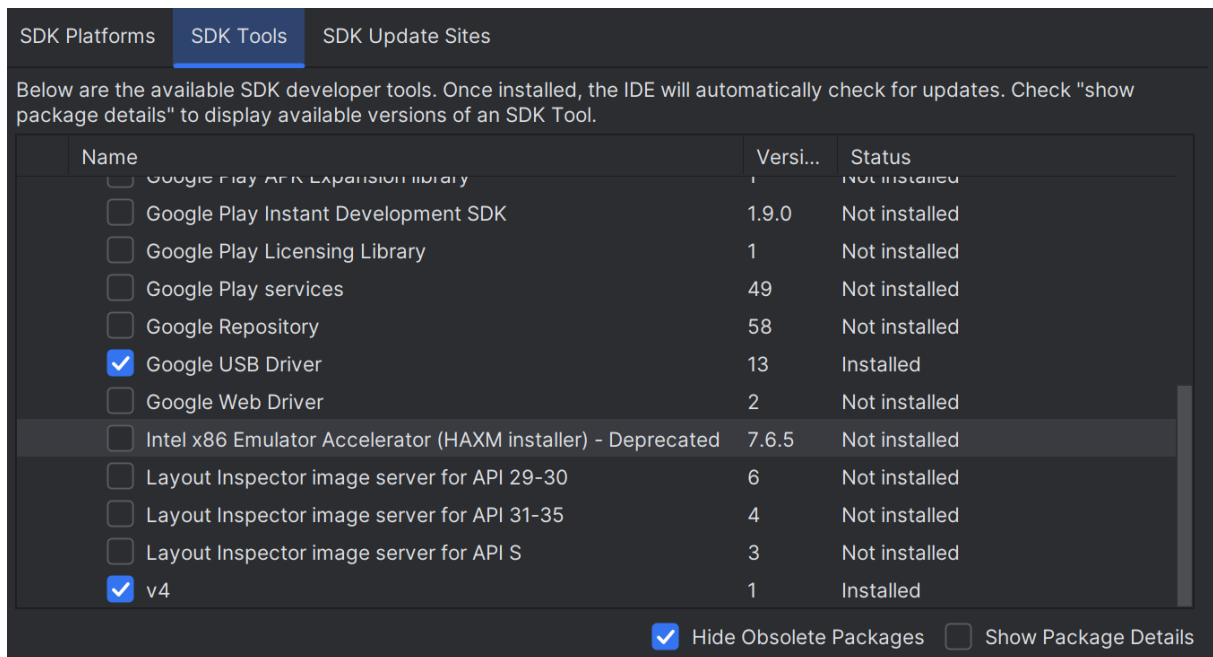


- + Phần Android SDK Location chọn lại: C:\Android\SDK
- + Platform: Chọn Android 13, 14 (Mỗi version chỉ cần một lựa chọn)



+ Phần SDK Tools: Chọn như hình sau. Chú ý các máy sử dụng CPU AMD thì bỏ chọn Intel X86 Emulator...., còn các máy CPU Intel thì chọn phần này.

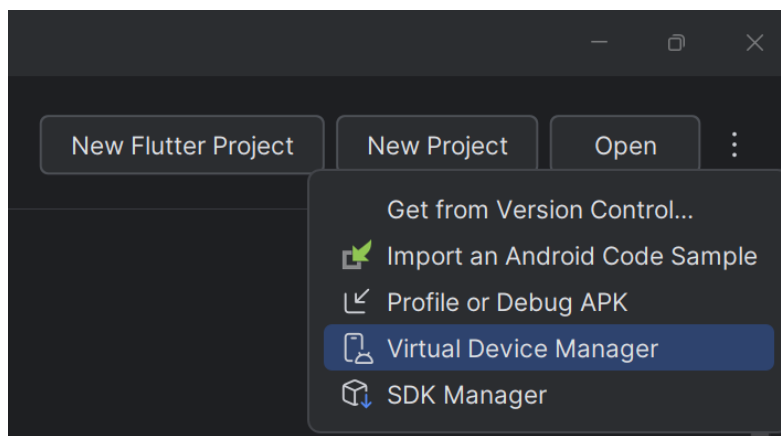




Click Apply --> OK và chờ cài đặt (Thời gian khoảng 30 phút đến 2h tùy thuộc vào tốc độ mạng).

3. Cài đặt máy ảo: Có thể cài đặt máy ảo trong Android Studio hay BlueStack. Nếu có bị sự cố/lỗi có thể vào nhóm Zalo môn học để hỏi và được giải đáp.

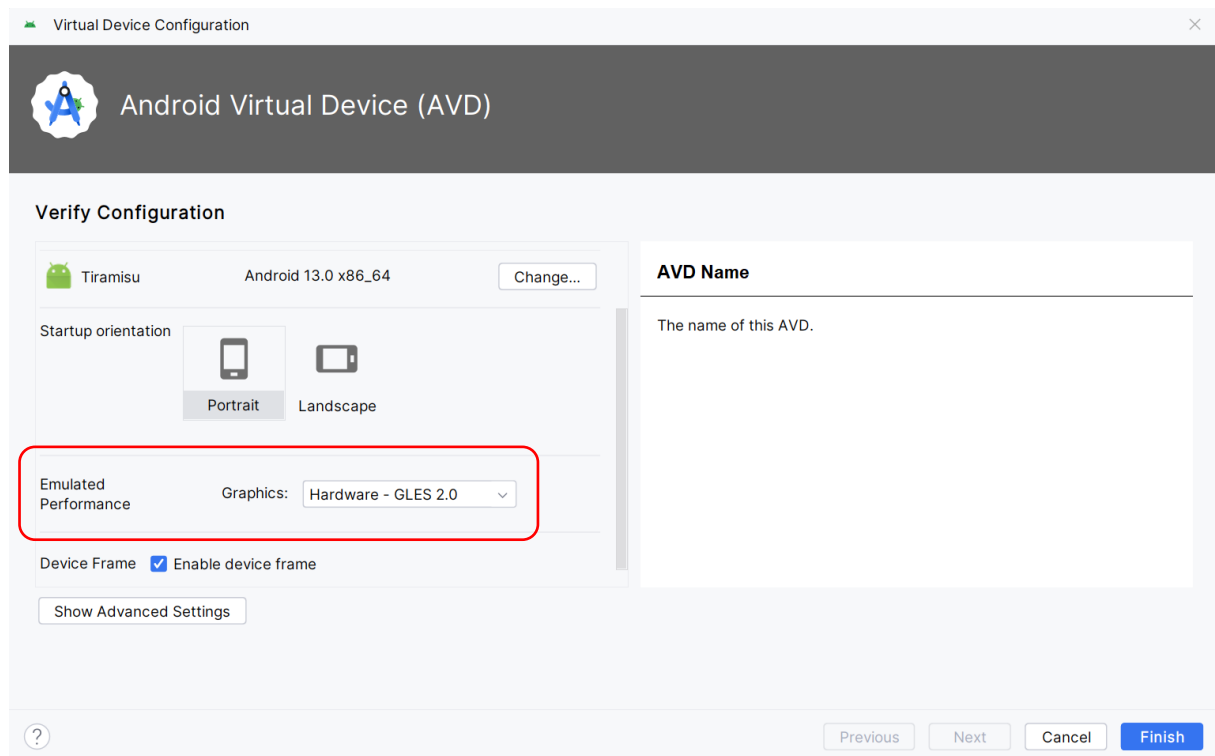
b. Máy ảo Android: Vào mục Setting --> Virtual Device Manager



Chú ý đối với một số máy không thể chạy máy ảo trên Virtual Box/VMWare thì có thể bật tính năng ảo hóa trong Windows thông qua “Turn Windows Features On or Off”. Các tính năng cần bật lên là:

- + Hyper-V
- + Windows Hypervisor Platform
- + Virtual Machine Platform

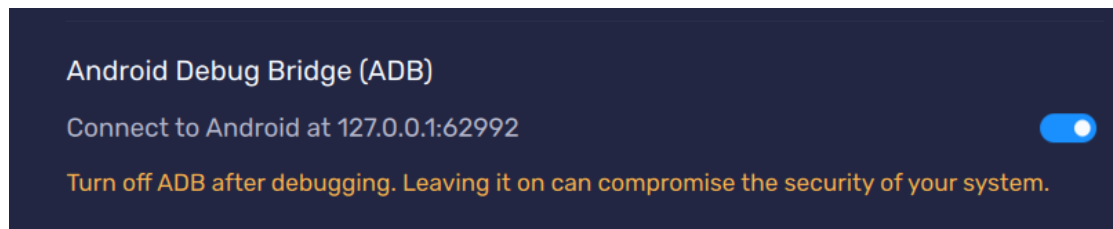
Ngoài ra tùy vào sự tương thích của phần cứng mà máy ảo có thể phải bật (Máy ảo sẽ chạy nhanh hơn) hoặc tắt sự hỗ trợ của phần cứng về Graphic cho máy ảo tức là chuyển từ Hardware sang Software



b. Máy ảo BlueStack: Sau khi cài đặt xong phần máy ảo chú ý một số phần sau đây:

- + Phần Display: Thiết lập độ phân giải và DPI như thông số của một điện thoại thật.

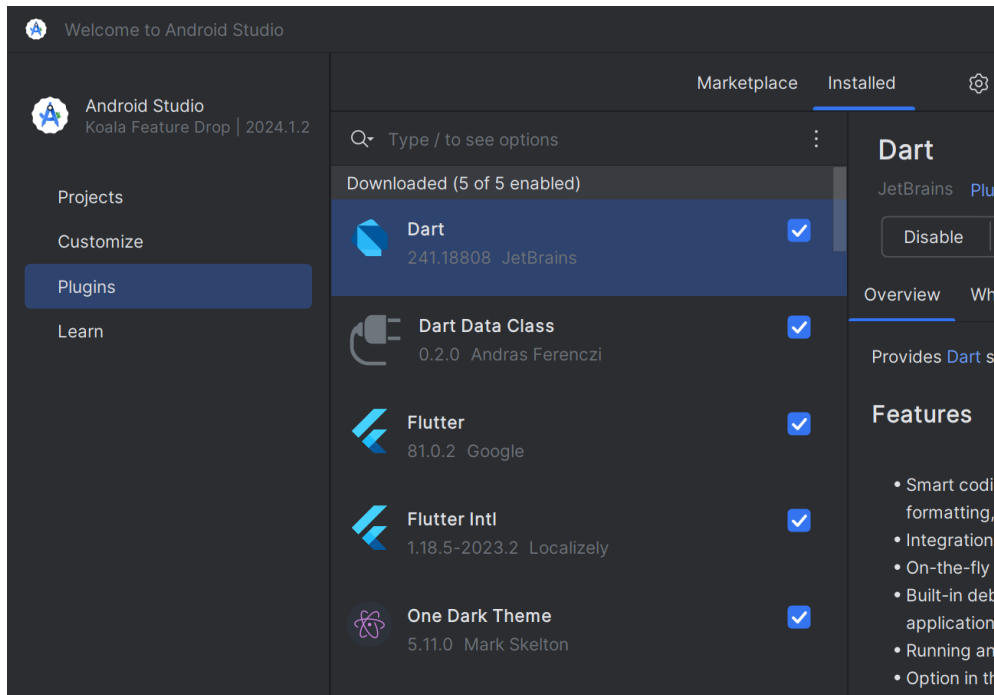
- + Phần Advanced (Nâng cao): Bật Android Debug Bridge, chú ý tới địa chỉ cổng localhost của máy ảo (địa chỉ này thay đổi ở mỗi lần chạy máy ảo)



- + Thiết lập biến môi trường trong Windows cho thư mục platform-tools (trong thư mục sdk) để chạy câu lệnh adb trên cmd hoặc terminal nhằm kết nối máy ảo và Android Studio:

Ví dụ: adb connect 127.0.0.1:62992

4. Cài đặt Plugin Flutter, Dart



5. Tải SDK Flutter, giải nén vào thư mục C:\Android

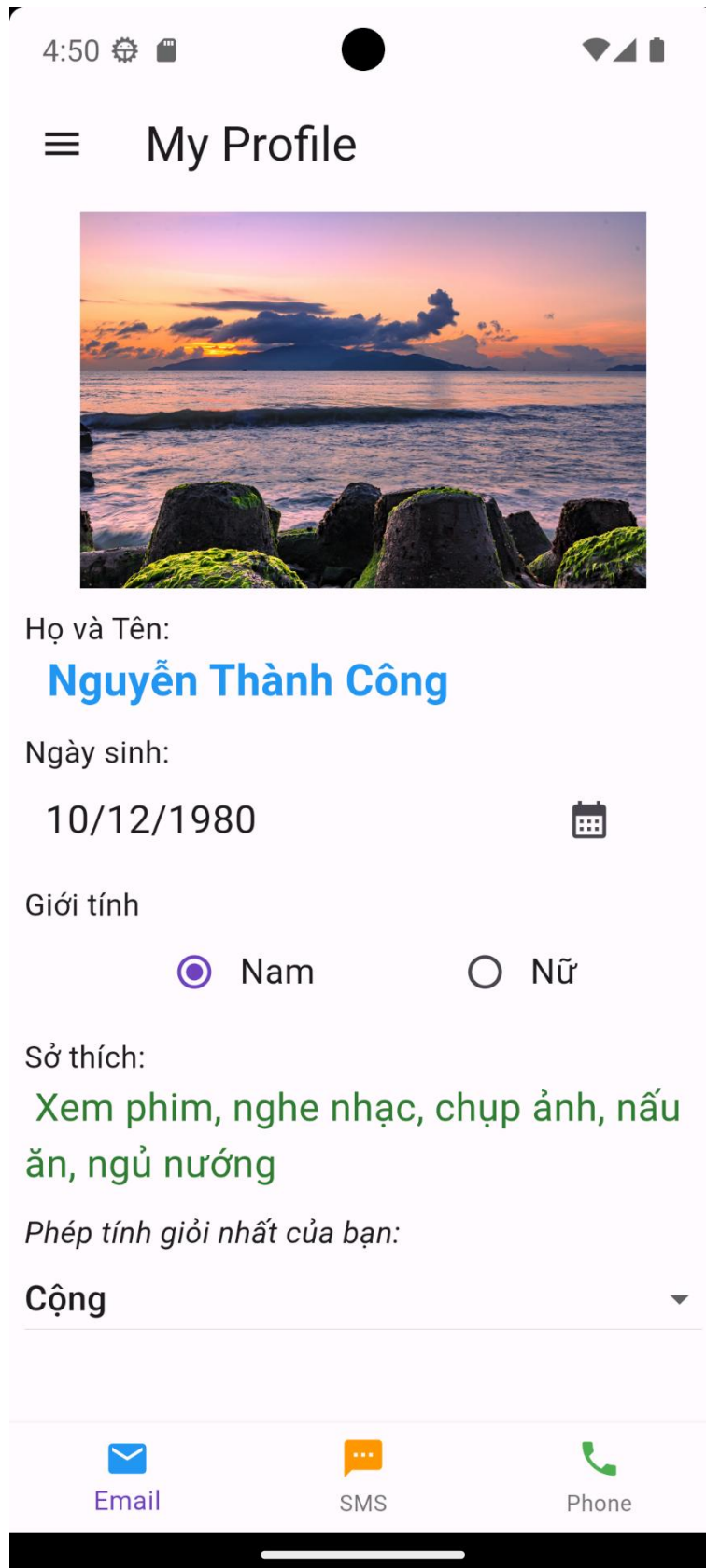
6. Tạo Flutter Project mới để test (Chú ý project này sẽ được dùng để thực hành trong suốt khóa học, do đó sinh viên cần tạo project theo các thông tin sau đây:

(SV chú ý thông tin phần Project name và Project location, Organization)

Project name:	thao_vy6312456_flutter		
Project location:	C:\code\flutter\thao_vy6312456_flutter		...
Description:	A new Flutter project.		
Project type:	Application		
Organization:	vn.edu.ntu.thaovy		
Android language:	<input checked="" type="radio"/> Java	<input type="radio"/> Kotlin	
iOS language:	<input type="radio"/> Objective-C	<input checked="" type="radio"/> Swift	
Platforms:	<input checked="" type="checkbox"/> Android <input checked="" type="checkbox"/> iOS <input type="checkbox"/> Linux <input type="checkbox"/> MacOS <input type="checkbox"/> Web <input type="checkbox"/> Windows		
When created, the new project will run on the selected platforms (others can be added later).			
<input type="checkbox"/> Create project offline			

Bài tập 2: Ứng dụng hiển thị Profile cá nhân

Hiển thị các thông tin cá nhân đơn giản như: Họ tên, Ngày sinh, Quê quán, sở thích, giới tính trên một trang của ứng dụng Flutter.



Hướng dẫn:

- Trang ứng dụng là một StatefulWidget.

- Vẽ Widget Tree của ứng dụng

- Gợi ý:

- Thiết kế giao diện theo Column Layout.
- Ảnh đại diện: Lưu ảnh trong assets của ứng dụng.
- Để hiển thị ảnh có kích thước như mong muốn và căn giữa màn hình, bọc Image trong Container, sau đó tiếp tục bọc Container trong một Center.

```
Center(  
  child: Container(  
    height: size*2/3,width: size,  
    child: Image.asset(myProfile.imageAssest)),  
),
```

- Muốn kích thước ảnh hiển thị phụ thuộc vào kích thước thiết bị, sử dụng MediaQuery.of(context) để truy vấn kích thước của thiết bị:

```
MediaQuery.of(context).size.width
```

- Sử dụng các SizedBox widget để tạo khoảng cách giữa các Widget khác trong ứng dụng. Thuộc tính *width* cho phép thiết lập khoảng cách theo chiều ngang, thuộc tính *height* cho phép thiết lập khoảng cách theo chiều dọc
- Sử dụng Text để hiển thị nội dung văn bản.
- Phần chọn giới tính: Sử dụng 2 Radio Widget. Nguyên tắc sử dụng Radio như sau:
 - Mỗi Radio có hai thuộc tính: value và groupValue. Value là thuộc tính có giá trị không đổi; groupValue là thuộc tính có giá trị thay đổi. Khi groupValue == value thì Radio tương ứng sẽ được chọn.
 - onChange(value)=>{setState(){...}}: Callback nhận vào giá trị value là giá trị value của Radio đang chọn.
- Chọn ngày tháng năm sinh: Sử dụng hàm showDatePicker
- Phép tính giới nhất: Sử dụng DropdownButton Widget. Nguyên tắc sử dụng như sau:
 - items: danh sách các DropdownMenuItem. Mỗi DropdownMenuItem có thuộc tính child chính là thành phần hiển thị; value chính là giá trị của thành phần hiển thị, value này có giá trị cố định.
 - value (chú ý là value của DropdownButton) có giá trị thay đổi, khi giá trị này bằng với một trong các giá trị của các DropdownMenuItem thì DropdownMenuItem tương ứng sẽ được lựa chọn.

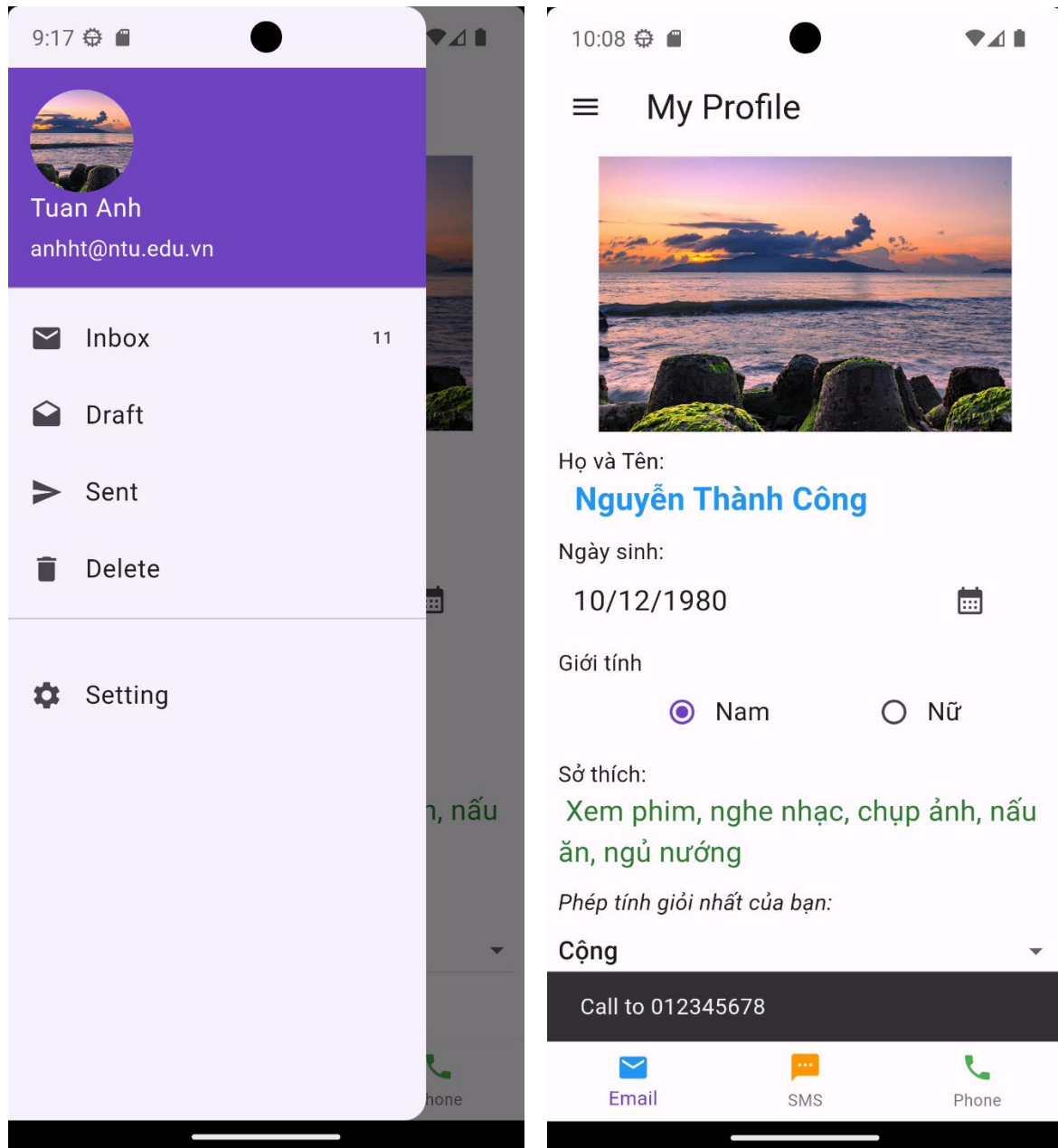
Bài tập 3:

Mục đích:

- Thiết kế giao diện của ứng dụng Flutter

- Thực hành các thành phần của Scaffold: AppBar, BottomAppBar, FloatingActionButton, Drawer, SnackBar, BottomSheet...

Thiết kế Ứng dụng với giao diện như hình dưới đây.



Hướng dẫn:

1. Sinh viên tìm hiểu các thành phần của Drawer:

- children: Gồm DrawerHeader và các item (có thể sử dụng ListTile cho mỗi item)

- DrawerHeader: sử dụng UserAccountsDrawerHeader

- Ảnh trong DrawerHeader: thuộc tính currentAccountPicture của UserAccountsDrawerHeader là một CircleAvatar

Thiết kế Drawer: Viết lớp ProfileDrawer là một StatelessWidget. Phương thức build trả về một Drawer

- Thành phần child của Drawer là một ListView (hoặc Column).
- Thành phần children gồm:

```
o UserAccountsDrawerHeader(  
  currentAccountPictureSize: Size.square(75),  
  currentAccountPicture: CircleAvatar(  
    backgroundImage: AssetImage("asset/images/binh_minh.jpg"),  
    // child: Text(  
    //   "A",  
    //   style: TextStyle(fontSize: 30),  
    // ),  
  ),  
  accountName: Text("Tuan Anh"),  
  accountEmail: Text("anhht@ntu.edu.vn"),  
o  
o ListTile(  
  leading: Icon(Icons.mail),  
  title: Text("Inbox"),  
  trailing: Text("11"),  
),
```

2. Tìm hiểu BottomNavigationBar: Các thành phần của BottomNavigationBar

- currentIndex: index của item đang được active.

- items: Danh sách các BottomNavigationBarItem.

- Mỗi item có một index, item đầu tiên có index 0.

- onTap:(value){}: Callback nhận value là index của item đang chọn. Call back này dùng để xử lý các sự kiện liên quan như mở một trang ứng dụng.

Bài tập 4: Phát triển dựa trên bài tập 2

Thêm vào nút chỉnh sửa để mở một trang mới cho phép chỉnh sửa các thông tin như họ tên, quê quán, ngày sinh.

Hướng dẫn:

- o Cả hai trang của ứng dụng đều là các StatefulWidget.
- o Định nghĩa một class Profile cho đối tượng là state trong widget đầu tiên

```
class Profile{  
  String hoTen;  
  String queQuan;  
  DateTime ngaySinh;
```

```
String soThich;
String imageAssest;
```

```
Profile({this.hoTen, this.queQuan, this.ngaySinh, this.soThich,
this.imageAssest});
}
```

- Khởi tạo trạng thái (state) cho trang thứ nhất:

```
@override
void initState() {
  // TODO: implement initState
  super.initState();
  myProfile = Profile(
    hoTen: "Trần Thành Công",
    ngaySinh: DateTime(2000,9,11),
    imageAssest: 'assets/bai_reu/bt1.jpg',
    queQuan: 'Nha Trang, Khánh Hòa',
    soThich: 'Xem phim, nghe nhạc, cafe với bạn bè, chụp ảnh trong giờ
rảnh rỗi'
  );
}
```

- Hiện thị ngày sinh từ đối tượng ngaySinh:

```
Text(
  '${myProfile.ngaySinh.day}/${myProfile.ngaySinh.month}/${myProfile.ngaySinh.year}',
  style: TextStyle(fontSize: 18,),
),
```

- Thiết kế trang 2 là một StatefulWidget có một thuộc tính là một đối tượng Profile:

```
class ProfileEditPage extends StatefulWidget {
  Profile profileEdit;
  ProfileEditPage(this.profileEdit);

  @override
  _ProfileEditPageState createState() => _ProfileEditPageState();
}
```

- Phương thức xử lý sự kiện khi bấm vào nút “Chỉnh sửa”:

```
Future<void> _editProfile() async{
  // Mở trang ProfileEditPage và chờ trả về một đối tượng Profile
  Profile editedProfile = await Navigator.push(
    context, MaterialPageRoute(
      builder:(context) => ProfileEditPage(myProfile),
    )
  );
  // Gọi phương thức setState để cập nhật giao diện ứng dụng
  setState(() {
    myProfile.hoTen = editedProfile.hoTen;
    myProfile.ngaySinh = editedProfile.ngaySinh;
    myProfile.queQuan = editedProfile.queQuan;
    myProfile.soThich = editedProfile.soThich;
  });
}
```

- Sử dụng TextField nhập văn bản:

```
TextField buildTextField(String label, TextEditingController controller,
bool readOnly) {
  return TextField(
```

```

        controller: controller,
        readOnly: readOnly ,
        maxLines: null, // để có thể nhập nhiều dòng trong TextField
        decoration: InputDecoration(
          labelText: label,
          border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(5)
          )
        ),
      );
    }
  }
}

```

- Sử dụng phương thức `showDatePicker` để hiển thị lịch chọn ngày:

```

Future<DateTime> _selectDate(DateTime ngaySinh) async{
  final DateTime _pickedDate = await showDatePicker(
    context: context,
    initialDate: ngaySinh,
    firstDate: DateTime.now().subtract(Duration(days: 365*50)),
    lastDate: DateTime.now().add(Duration(days: 365*50)),
  );
  return _pickedDate;
}

```

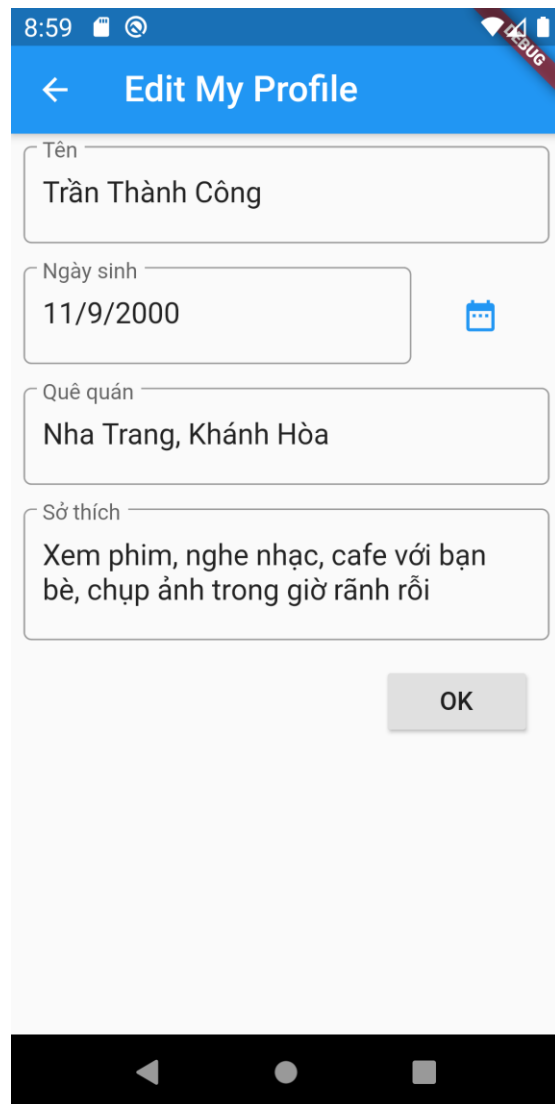
- Sử dụng Row Layout để hiển thị TextField và FlatButton trên cùng một dòng. Sử dụng Expanded để bọc TextField.

```

Row(
  children: [
    Expanded(child: buildTextField("Ngày sinh", ngaySinhController, true)),
    FlatButton(
      onPressed: () async{
        DateTime _selectedDate = await
          _selectDate(widget.profileEdit.ngaySinh);
        setState(() {
          widget.profileEdit.ngaySinh = _selectedDate;
          ngaySinhController.text = _textFromDate(_selectedDate);
        });
      },
      child: Icon(Icons.date_range, color: Colors.blue,)
    ),
  ],
),

```

-



Chú ý: Nên bọc tất cả các Widget của ứng dụng vào một SingleChildScrollView để giao diện vẫn hiển thị đúng và không bị lỗi khi nhập nội dung quá dài hoặc bàn phím của ứng dụng xuất hiện thêm khi nhập liệu.

Bài tập 5: Tương tự như bài tập 2, Sinh viên làm theo 3 version:

version 1: Dữ liệu lưu bằng SharedPreferences, quản lý trạng thái bằng setState.

version 2: Dữ liệu lưu bằng SharedPreferences, quản lý trạng thái bằng Provider.

version 3: Dữ liệu lưu bằng file .json

Hướng dẫn:

1. Version 1:

- Cài đặt lớp ProfilePreference hỗ trợ việc đọc, ghi dữ liệu preference.

- Các trang ứng dụng được cài đặt như sau:

```
class PreferenceSetStateApp extends StatelessWidget {
  @override
```

```

Widget build(BuildContext context) {
  return MaterialApp(
    home: ProfilePreferenceSetStatePage(),
  );
}

class ProfilePreferenceSetStatePage extends StatefulWidget {
  @override
  _ProfilePreferenceSetStatePageState createState() =>
  _ProfilePreferenceSetStatePageState();
}

class _ProfilePreferenceSetStatePageState extends
State<ProfilePreferenceSetStatePage> {
  Profile profile;
  bool loadData = false;
  bool error = false;
  @override
  Widget build(BuildContext context) {
    if(!error)
      if(loadData)
        return buildProfileWidget();
      else
        return buildLoadingWidget();
    else
      return
        errorBuildWidget();
  }

  void _loadData() async{
    try{
      profile = await ProfilePreference.readPreference();
      setState(() {
        loadData = true;
      });
    }catch(e){
      setState(() {
        error = true;
      });
      print("Lỗi đọc dữ liệu");
    }
  }

  Widget buildProfileWidget() {}

  Widget buildLoadingWidget() {}

  Widget errorBuildWidget() {}
}

```

2. Version 2:

Cài đặt lớp SharedPrefHelper kế thừa từ lớp ChangeNotifier:

```

class SharedPrefHelper extends ChangeNotifier{
  Profile _profile = Profile();

  SharedPrefHelper(){
    profile.hoTen = "Chưa có tên";
  }
}

```



```

        profile.queQuan = "Nhập quê quán";
        profile.ngaySinh = DateTime.now();
        profile.soThich = "Thích đủ thứ";
        profile.imageAssest = 'assets/bai_reu/bt1.jpg';
    }

    void getSharedProfile() async{
        SharedPreferences sharedPreferences = await SharedPreferences.getInstance();
        profile.hoTen = sharedPreferences.getString('hoTen') ?? "Chưa có tên";
        profile.queQuan = sharedPreferences.getString("queQuan") ?? "Nhập quê quán";
        profile.ngaySinh = sharedPreferences.getString("ngaySinh")!=null?
            DateTime.parse(sharedPreferences.getString("ngaySinh")) : DateTime.now();
        profile.soThich = sharedPreferences.get("soThich") ?? "Thích đủ thứ";
        notifyListeners();
    }

    Profile get profile => _profile;

    void updateProfile(Profile newProfile) async{
        _profile = newProfile;
        SharedPreferences sharedPreferences = await SharedPreferences.getInstance();
        sharedPreferences.setString("hoTen", _profile.hoTen);
        sharedPreferences.setString('queQuan', _profile.queQuan);
        sharedPreferences.setString('ngaySinh', _profile.ngaySinh.toString());
        sharedPreferences.setString('soThich', _profile.soThich);
        notifyListeners();
    }
}

```

Widget ứng dụng là một StatelessWidget có phương thức build trả về một ChangeNotifierProvider:

```

class ProfilePreferenceProviderApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return ChangeNotifierProvider<SharedPrefHelper>(
            create:(context) {
                SharedPrefHelper sharedPrefHelper = SharedPrefHelper();
                sharedPrefHelper.getSharedProfile();
                return sharedPrefHelper;
            },
            builder:(context, child) => MaterialApp(
                title: "Profile version 2",
                theme: ThemeData(
                    primarySwatch: Colors.blue,
                    visualDensity: VisualDensity.adaptivePlatformDensity,
                ),
                home: ProfilePageVer2(),
            ),
        );
    }
}

```

Lớp ProfilePageVer2 chỉ cần là một StatelessWidget:

```

class ProfilePageVer2 extends StatelessWidget {
    @override

```

```

Widget build(BuildContext context) {
  double size = MediaQuery.of(context).size.width*0.75;
  SharedPreferences prefHelper = context.watch<SharedPreferences>();
  Profile myProfile = prefHelper.profile;
  .....
// Phương thức được gọi trong sự kiện nút bấm “Chỉnh sửa”
Future<void> _editProfile(BuildContext context, Profile profile) {
  // Mở trang ProfileEditPage và chờ trả về một đối tượng Profile
  Navigator.push(
    context, MaterialPageRoute(
      builder:(context) => ProfileEditPageVer2(profile),
    )
  );
}
}

```

- ProfileEditPageVer2 là một StatefulWidget có sự kiện nút bấm “Ok” như sau:

```

RaisedButton(
  onPressed: () {
    SharedPreferences sharePref = context.read<SharedPreferences>();
    _savedProfile(sharePref);
  },
  child: Text("OK"),
),

void _savedProfile(SharedPreferences sharedPrefHelper) {
  widget.profileEdit.ten = tenController.text;
  widget.profileEdit.queQuan = queQuanController.text;
  widget.profileEdit.soThich = soThichController.text;
  sharedPrefHelper.updateProfile(widget.profileEdit);
  Navigator.pop(context);
}

```

3. Version 3:

Cài đặt lớp hỗ trợ đọc ghi file: Sử dụng Template Method để cài đặt.

```

import 'dart:io';
import 'package:path_provider/path_provider.dart';
// Lớp được cài đặt chung cho tất cả các trường hợp đọc/ghi file text.
abstract class FileHelper{
  String fileName;

  FileHelper({this.fileName});

  String initialContent(); // Phương thức trừu tượng sẽ được implement ở các lớp con

  Future<String> get _localPath async{
    var directory = await getApplicationDocumentsDirectory();
    return directory.path;
  }

  Future<File> get _localFile async{
    String path = await _localPath;
    return File('$path/$fileName');
  }

  Future<File> writeString(String str) async{
    File file = await _localFile;

```

```

        return file.writeAsString(str);
    }

    Future<String> readFile() async{
        try{
            File file = await _localFile;
            if(!file.existsSync()) {
                print('file chưa tồn tại: ${file.absolute}');
                await file.writeAsString(initialContent());
            }
            String content = await file.readAsString();
            return content;
        } catch(e){
            print('Lỗi khi đọc file');
            return null;
        }
    }
}

```

// Lớp cài đặt riêng cho từng trường hợp.

```

const String congViecFile = "profile.json";
class FileProfileHelper extends FileHelper{

    FileProfileHelper():super(fileName: congViecFile);
    @override
    String initialContent() {
        return '{"profile":{}}';
    }
}

```

- Các lớp quản lý dữ liệu:

```

class ProfileJson{
    String hoTen;
    String queQuan;
    DateTime ngaySinh;
    String soThich;
    String imageAsset;

    ProfileJson(
        {this.hoTen, this.queQuan, this.ngaySinh, this.soThich, this.imageAsset});

    factory ProfileJson.fromJson(Map<String, dynamic> json){
        return ProfileJson(
            hoTen: json['hoTen'],
            queQuan: json['queQuan'],
            ngaySinh: DateTime.parse(json['ngaySinh']),
            soThich: json['soThich'],
            imageAsset: json['imageAsset']
        );
    }
}

Map<String, dynamic> toJson(){
    return {
        'hoTen':this.hoTen,
        'queQuan':this.queQuan,
        'ngaySinh': ngaySinh,
        'soThich':this.soThich,
        'imageAsset':this.imageAsset
    };
}

```

```

}

class ProfileDatabase extends ChangeNotifier{
  ProfileJson _profile;
  FileProfileHelper fileHelper = FileProfileHelper();
  ProfileDatabase(){
    _profile = ProfileJson(
      hoTen: "",
      ngaySinh: DateTime.now(),
      queQuan: '',
      soThich: '',
      imageAsset: "assets/bai_reu/bt1.jpg"
    );
  }

  ProfileJson get profile => _profile;

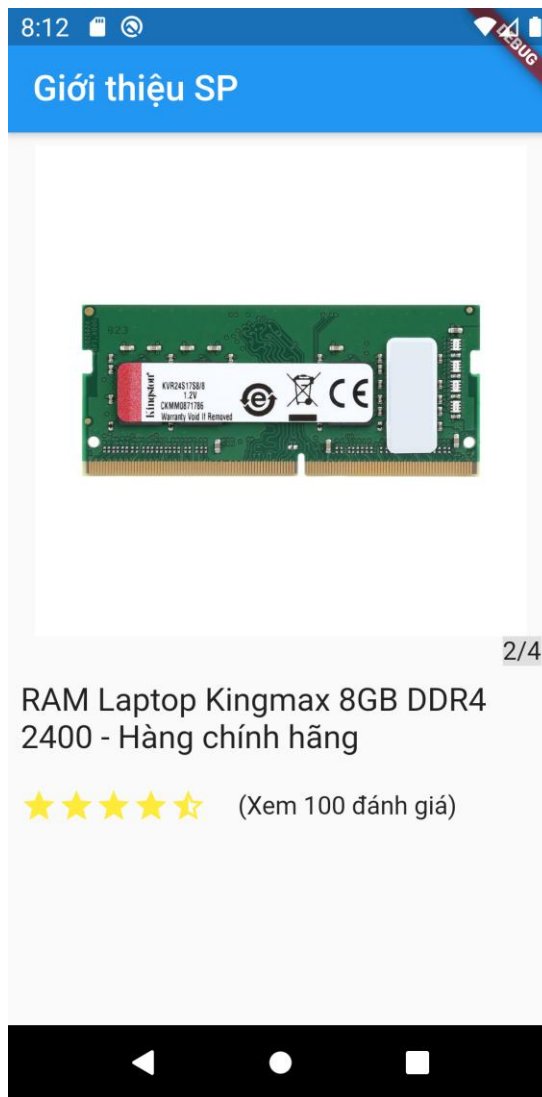
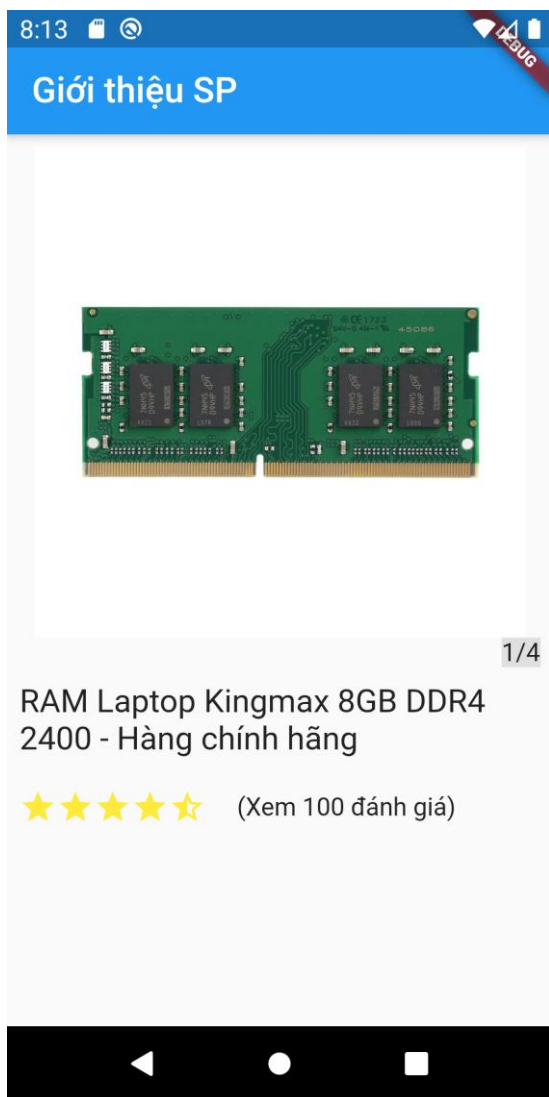
  void readProfile() async{
    String content = await fileHelper.readFile();
    Map data = json.decode(content)['profile'];
    if(data.isNotEmpty) {
      _profile = ProfileJson.fromJson(data);
      notifyListeners();
    }
  }

  void updateProfile(ProfileJson newProfile) async{
    _profile = newProfile;
    notifyListeners();
    String content = json.encode(newProfile);
    await fileHelper.writeString(content);
  }
}

```

- Tương tự như Version 2, sử dụng ChangeNotifierProvider để bọc trang ứng dụng (MaterialApp widget)

Bài tập 6: Sử dụng GestureDetector Thiết kế giao diện giới thiệu sản phẩm của một trang TMĐT



Hướng dẫn:

- Sử dụng CarouselSlider (pub.dev: carousel_slider) để hiển thị các ảnh. Các ảnh có thể xuất hiện theo hiệu ứng trượt (slide) khi người sử dụng thực hiện thao tác vuốt trên màn hình. Khai báo package này trong tập tin pubspec.yaml
- Sử dụng StatefulWidget để thiết kế trang ứng dụng (Sinh viên tự giải thích vì sao không sử dụng StatelessWidget để thiết kế).
- Copy các ảnh của ứng dụng trong assets, khai báo các đường dẫn trong tập tin pubspec.yaml.
- Các state trong lớp State của ứng dụng:

```
class _SanPhamPageState extends State<SanPhamPage> {
  List<String> images = [
    'assets/ram/ram1.jpg',
    'assets/ram/ram2.jpg',
    'assets/ram/ram3.jpg',
    'assets/ram/ram4.jpg',
  ];
  int imagePos=0;
```

Hiển thị ảnh bằng CarouselSlider:

```

CarouselSlider.builder(
  itemCount: images.length,
  options: CarouselOptions(
    height: with_image*0.9,
    viewportFraction: 1,
    onPageChanged: (index, reason) {
      // Cập nhật số thứ tự ảnh
      setState(() {
        imagePos = index;
      });
    },
  ),
  itemBuilder: (context, index) {
    return Container(
      height: with_image*0.9, width: with_image*0.9,
      //margin: EdgeInsets.all(5),
      child: Image.asset(images[index]),
    );
  },
),

```

Hiển thị số thứ tự ảnh:

```

Row(
  children: [
    Expanded(child: SizedBox()),
    Container(
      color: Colors.grey[300],
      child: Text('${imagePos+1}/${images.length}'),
    ),
  ],
),

```

Bài tập 7: Form, Dialog

Hướng dẫn:

- Chỉ cần sử dụng StatelessWidget để thiết kế trang ứng dụng
- Sử dụng Form để bọc các TextFormField, DropdownButtonFormField.
- Khai báo một GlobalKey, sau đó gán cho thuộc tính key của Form để có thể truy xuất đến Form:

```
GlobalKey<FormState> _formState = GlobalKey<FormState>();
```

- Khai báo đối tượng MatHang để chứa thông tin nhập trên Form

```

class MatHang{
  String tenMH,loaiMH;
  int soLuong;
  MatHang({this.tenMH, this.loaiMH, this.soLuong});
}

List<String> loaiMHs = ['Tivi', 'Điện thoại', 'Laptop'];

class MyForm extends StatelessWidget {
  GlobalKey<FormState> _formState = GlobalKey<FormState>();
  MatHang mh = MatHang();
  ...
}

```

```
body: Form(
  key: _formState,
  autovalidateMode: AutovalidateMode.disabled,
  ...
)
```

- TextFormField mặt hàng được thiết kế như sau:

```
TextFormField(
  decoration: InputDecoration(
    labelText: 'Mặt hàng',
    border: OutlineInputBorder(
      borderRadius: BorderRadius.all(Radius.circular(10))
    )
  ),
  onSaved: (newValue) {mh.tenMH=newValue;},
  validator: (value) => value.isEmpty ? "Chưa có tên mặt hàng" : null,
),
```

Callback onSaved chỉ được gọi khi `_formState.currentState.save()` được gọi

- DropdownButtonFormField:

```
DropdownButtonFormField<String>(
  items: loaiMHs.map((loaiMH) => DropdownMenuItem<String>(
    child: Text(loaiMH),
    value: loaiMH,
  )).toList(),
  onChanged: (value) {
    mh.loaiMH=value;
  },
  validator: (value) => value ==null? "Chưa chọn loại mặt hàng" : null,
  decoration: InputDecoration(
    labelText: 'Loại mặt hàng',
    border: OutlineInputBorder(
      borderRadius: BorderRadius.all(Radius.circular(10))
    )
  ),
),
```

Phương thức *map* trên List chuyển một Item trong List thành một đối tượng cần thiết khác (trong trường hợp này là một `DropdownMenuItem<String>`)

- Thiết kế nút bấm và sự kiện trên nút bấm:

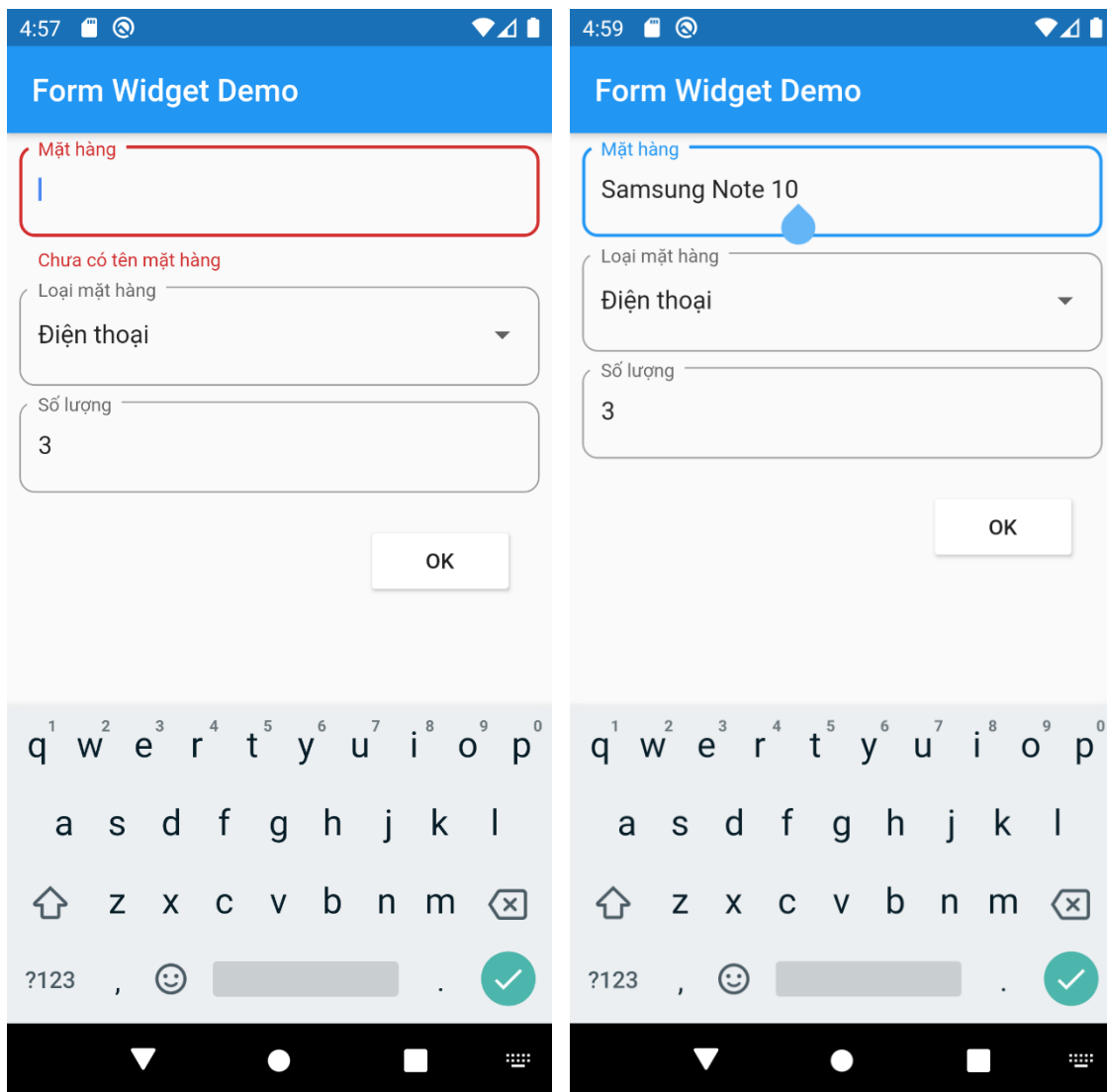
```
Row(
  children: [
    Expanded(child: SizedBox(), flex: 1,),
    RaisedButton(
      child: Text('OK'),
      color: Colors.white,
      onPressed: () {
        if(_formState.currentState.validate()) {
          _formState.currentState.save();
          _showAlertDialog(context);
        }
      },
    ),
    SizedBox(width: 20,),
  ],
),

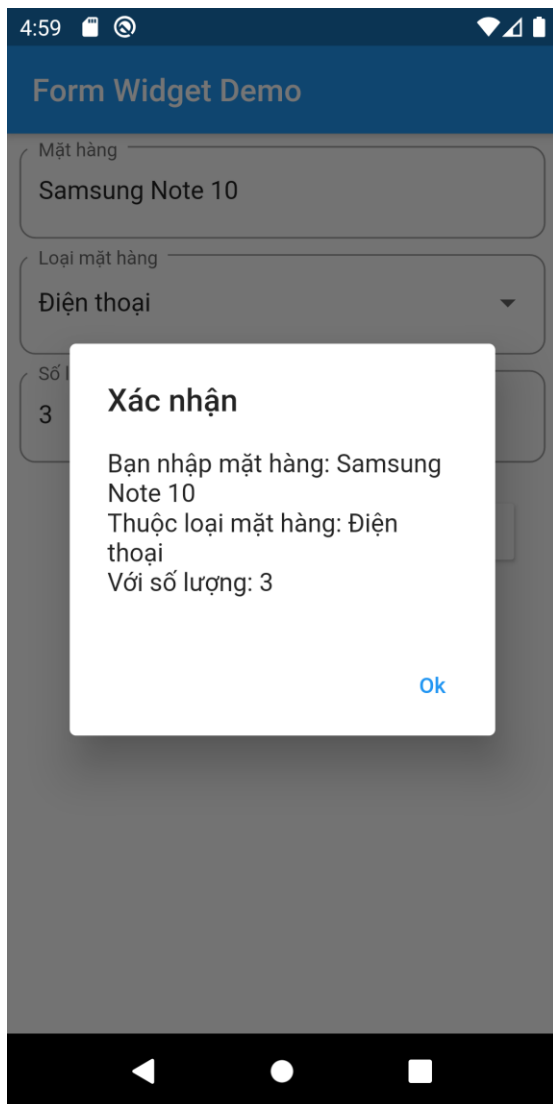
void _showAlertDialog(BuildContext context){
  AlertDialog alertDialog = AlertDialog(
```

```

title: Text('Xác nhận'),
content: Text("Bạn nhập mặt hàng: ${mh.tenMH}\n"
  "Thuộc loại mặt hàng: ${mh.loaiMH}\n"
  "Với số lượng: ${mh.soLuong}"),
actions: [
  FlatButton(
    onPressed: () => Navigator.pop(context),
    child: Text('Ok'))
],
);
showDialog(
  context: context,
  builder:(context) => alertDialog,
);
}

```





Bài tập 8: Xử lý dữ liệu Json và ListView

Đọc file json trong assets hoặc trong thư mục ứng dụng và hiển thị trên ListView.

Hướng dẫn:

- Tạo file users.json trong assets với nội dung như sau:

```
[
  {
    "name": "Tuấn",
    "email": "tuan@gmail.com"
  },
  {
    "name": "Thanh",
    "email": "thanh@gmail.com"
  },
  {
    "name": "Tùng",
    "email": "tung@gmail.com"
  },
  {
    "name": "Dũng",
    "email": "dung@gmail.com"
  },
]
```

```

    {
      "name": "Thảo",
      "email": "thao@gmail.com"
    }
  ]

```

- Để đọc file json thành một danh sách các đối tượng, ta sử dụng import hai thư viện:

```

import 'package:flutter/services.dart' show rootBundle;
import 'dart:convert';

```

Thư viện đầu tiên dùng để đọc file json, thư viện thứ hai dùng để encode và decode dữ liệu json.

- Khai báo hai lớp User như sau, phương thức khởi tạo factory User.fromJson tạo một User từ một đối tượng Map (dữ liệu key/value dạng json). Phương thức toJson hỗ trợ việc chuyển một đối tượng User thành một Map.

```

class User{
  String name, email;

  User({this.name, this.email});
  factory User.fromJson(Map<String, dynamic> json){
    return User(
      name: json['name'],
      email: json['email']
    );
  }

  Map<String, dynamic> toJson(){
    return{
      'name':name,
      'email':email
    };
  }
}

```

- Đọc file json và chuyển thành một danh sách các User:

```

Future<List<User>> fetchUserFromJson(String path) async{
  String userJson = await rootBundle.loadString(path);
  List<dynamic> list = jsonDecode(userJson) as List;
  return list.map((user) => User.fromJson(user)).toList();
}

```

- Sử dụng FutureBuilder để xây dựng giao diện ứng với dữ liệu được trả về bởi phương thức async

```

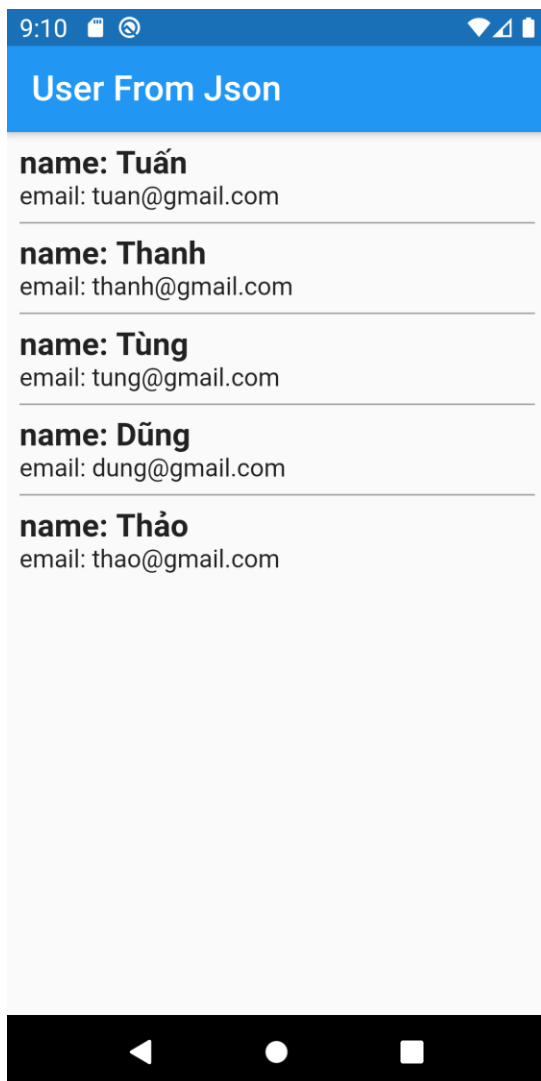
body: FutureBuilder<List<User>>(
  initData: [],
  future: fetchUserFromJson(path),
  builder: (context, snapshot) {
    if(snapshot.hasError)
      return Center(child:Text("Lỗi xảy ra"));
    else
      return snapshot.hasData?
        Padding(
          padding: const EdgeInsets.all(8.0),
          child: ListView.separated(
            itemBuilder:(context, index) => Column(
              crossAxisAlignment: CrossAxisAlignment.start,

```

```

        children: [
          Text('name: ${snapshot.data[index].name}',
            style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
          ),
          Text('email: ${snapshot.data[index].email}'),
        ],
      ),
      separatorBuilder: (context, index) => Divider(
        color: Colors.grey,
        thickness: 1,
      ),
      itemCount: snapshot.data.length
    ),
  ),
): Center(child: CircularProgressIndicator());
},
),

```



Bài tập 8bis: Mở rộng bài tập 6: Cho phép chỉnh sửa, nhập thêm thông tin user.

Gợi ý:

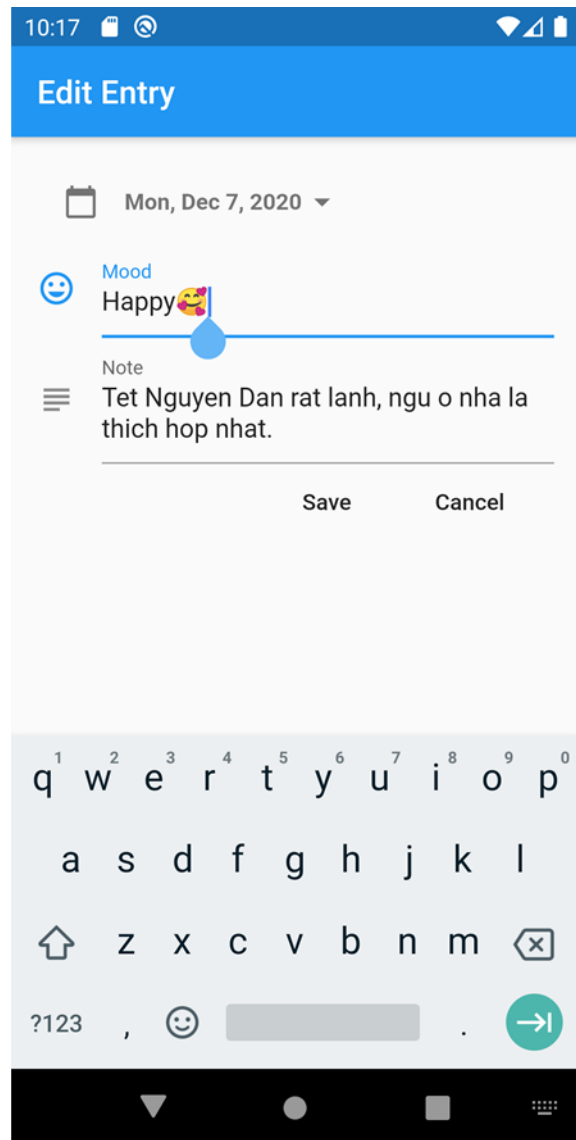
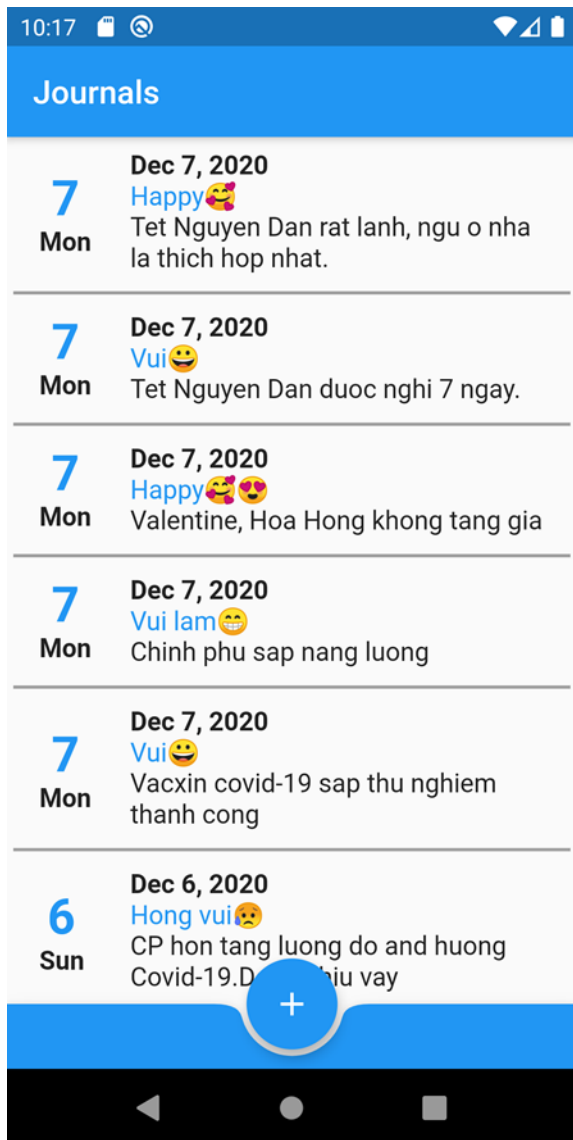
- Viết thêm lớp hỗ trợ đọc ghi file như bài tập số 4.

- Sử dụng Provider để quản lý trạng thái ứng dụng.

Bài tập 9: Đọc ghi file trong Flutter:

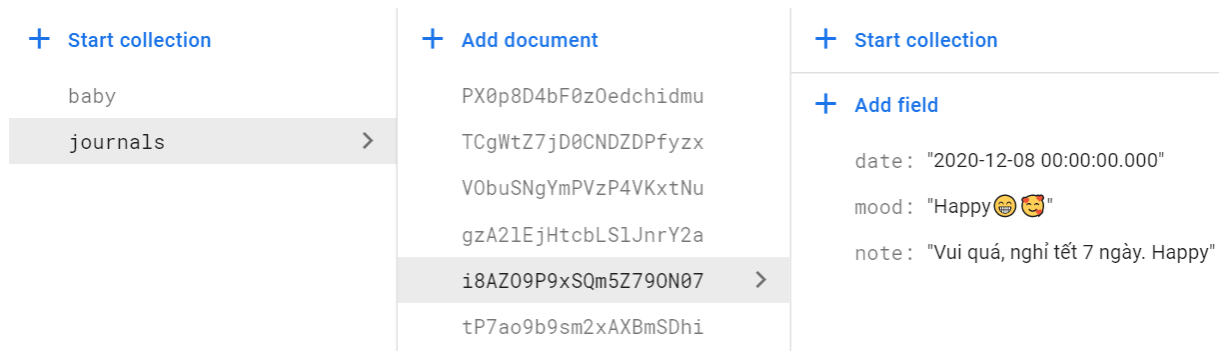
Thiết kế ứng dụng quản lý công việc trên Flutter, dữ liệu được tổ chức thành file json và ghi vào bộ nhớ cục bộ trên điện thoại.

Bài tập 10: Sử dụng Firebase Firestore:



Hướng dẫn:

1. Thiết lập kết nối firebase với ứng dụng.
2. Tạo một Collection có tên là "journals" và nhập vài document dữ liệu mẫu



3. Cài đặt các lớp dữ liệu:

3.1 Lớp JournalDoc có hỗ trợ jsonEncode và jsonDecode:

```
class JournalDoc{
  String date;
  String mood;
  String note;
  JournalDoc({this.date, this.mood, this.note});

  factory JournalDoc.fromJson(Map<String, dynamic> json)=>
    JournalDoc(
      date: json['date'],
      mood: json['mood'],
      note: json['note'],
    );

  Map<String, dynamic> toJson()=>
  {
    'date': date,
    'mood':mood,
    'note':note
  };
}
```

3.2 Lớp dữ liệu có chứa DocumentReference để cập nhật dữ liệu (Document):

```
class JournalSnapshot{
  JournalDoc doc;
  DocumentReference reference;
  JournalSnapshot({this.doc, this.reference});

  JournalSnapshot.fromJson(DocumentSnapshot snapshot):
    doc = JournalDoc.fromJson(snapshot.data()),
    reference = snapshot.reference;

  Future<void> update(String date, String mood, String note) async{
    return await reference.set({
      'date': date,
      'mood':mood,
      'note':note
    });
  }
}
```

3.3 Lớp tương tác với dữ liệu Firestore:

```
class FirestoreDatabaseJournal{
  Stream<List<JournalSnapshot>> getJournalsFromFirebase(){
    Stream<QuerySnapshot> streamQuerySnapshot =
```

```

FirebaseFirestore.instance.collection("journals").snapshots();
    return streamQuerySnapshot.map((QuerySnapshot querySnapshot) =>
        querySnapshot.docs.map((DocumentSnapshot documentSnapshot) =>
            JournalSnapshot.fromSnapshot(documentSnapshot)
        ).toList()
    );
}

Future<void> addJournal(JournalDoc journal) async{
    return await FirebaseFirestore.instance.collection("journals")
        .add({
            'date': journal.date,
            'mood': journal.mood,
            'note': journal.note
        }).then((value) => print("Đã thêm"));
}

Future<void> deleteJournal(JournalSnapshot journalSnapshot)
{
    return journalSnapshot.reference.delete();
}
}

```

4. Trang ứng dụng, khởi tạo firebase

```

class JournalAppPageFirebase extends StatefulWidget {
    @override
    _JournalAppPageFirebaseState createState() =>
    _JournalAppPageFirebaseState();
}

class _JournalAppPageFirebaseState extends State<JournalAppPageFirebase> {
    bool _initialized = false;
    bool _error = false;
    @override
    Widget build(BuildContext context) {
        if(_error)
            return Container(
                color: Colors.white,
                child: Center(
                    child: Text(
                        "Lỗi kết nối",
                        style: TextStyle(fontSize: 16),
                        textDirection: TextDirection.ltr,
                    ),
                ),
            );
        else
            if(_initialized)
                return Provider<FirestoreDatabaseJournal>(
                    create:(context) => FirestoreDatabaseJournal(),
                    child: MaterialApp(
                        debugShowCheckedModeBanner: false,
                        title: 'Journal App Firebase',
                        theme: ThemeData(
                            primarySwatch: Colors.blue,
                            bottomAppBarColor: Colors.blue,
                        ),
                        home: JournalPageProvider(),
                    ),
                );
            else
                return Container(
                    color: Colors.white,
                    child: Center(
                        child: Text(
                            "Chưa khởi tạo",
                            style: TextStyle(fontSize: 16),
                            textDirection: TextDirection.ltr,
                        ),
                    ),
                );
    }
}

```

```

        else return Container(
          color: Colors.white,
          child: Center(
            child: Text(
              "Đang kết nối",
              style: TextStyle(fontSize: 16),
              textDirection: TextDirection.ltr,
            ),
          ),
        );
      }

void initializeFlutterFire() async {
  try {
    await Firebase.initializeApp();
    setState(() {
      _initialized = true;
    });
  } catch(e) {
    _error = true;
  }
}

@override
void initState() {
  initializeFlutterFire();
  super.initState();
}
}

```

5. Để sử dụng `Stream<List<JournalSnapshot>>`, `JournalPageProvider` (là một `StatelessWidget`) sử dụng một `StreamBuilder<List<JournalSnapshot>>`:

```

Widget build(BuildContext context) {
  FirestoreDatabaseJournal fireStoreDatabaseJournal =
    Provider.of<FirestoreDatabaseJournal>(context);

  .....

  body: StreamBuilder<List<JournalSnapshot>>(
    stream: fireStoreDatabaseJournal.getJournalsFromFirebase(),
    builder: (context, snapshot) {
      if(snapshot.hasData)
        return _buildListViewSeparated(snapshot.data, context);
      return Center(child: Text("Đang tải dữ liệu"));
    },
  ),
)

```

Bài tập 11: SQLite

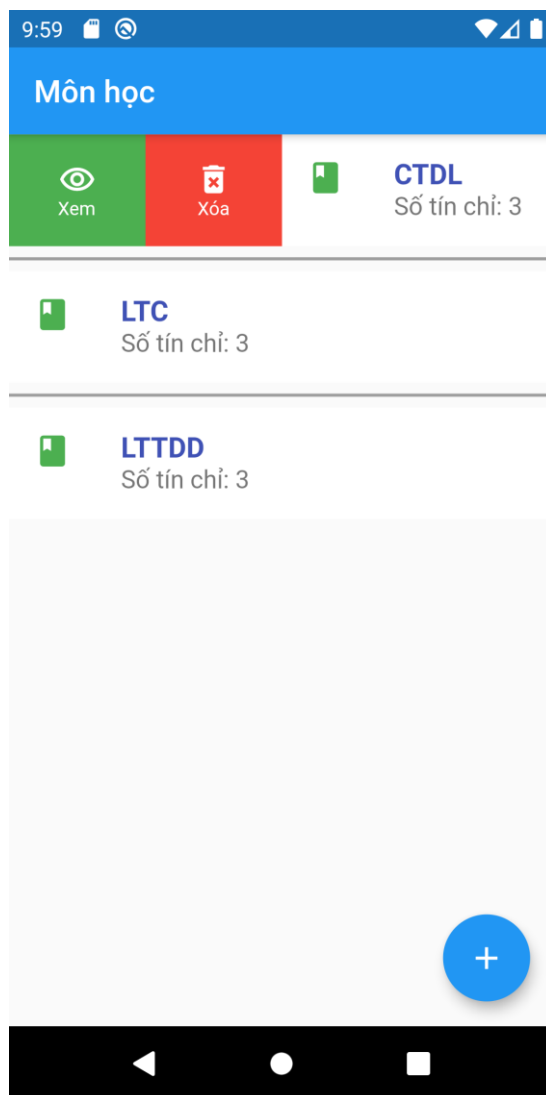
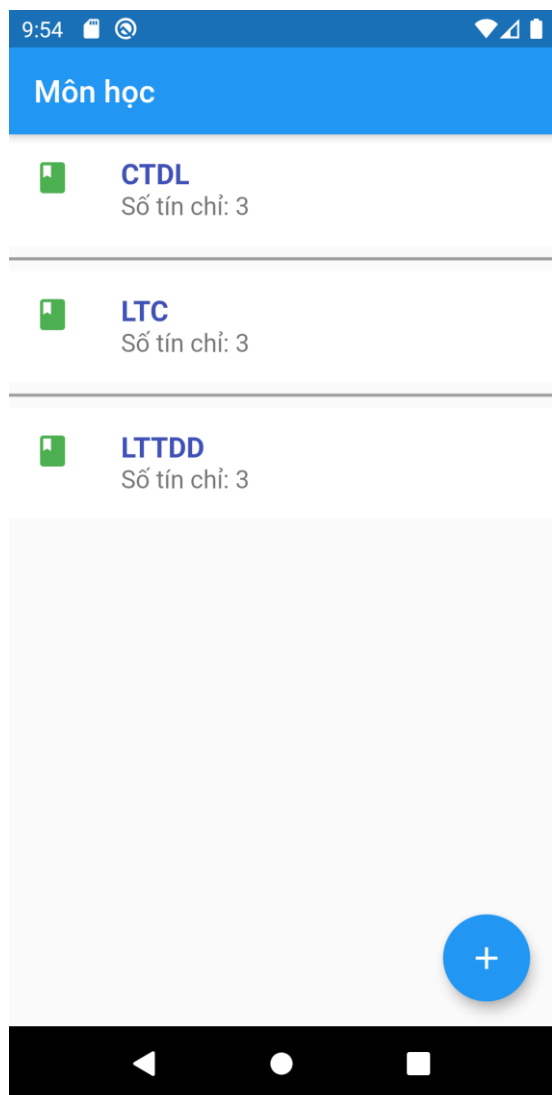
- Ứng dụng quản lý điểm các môn học

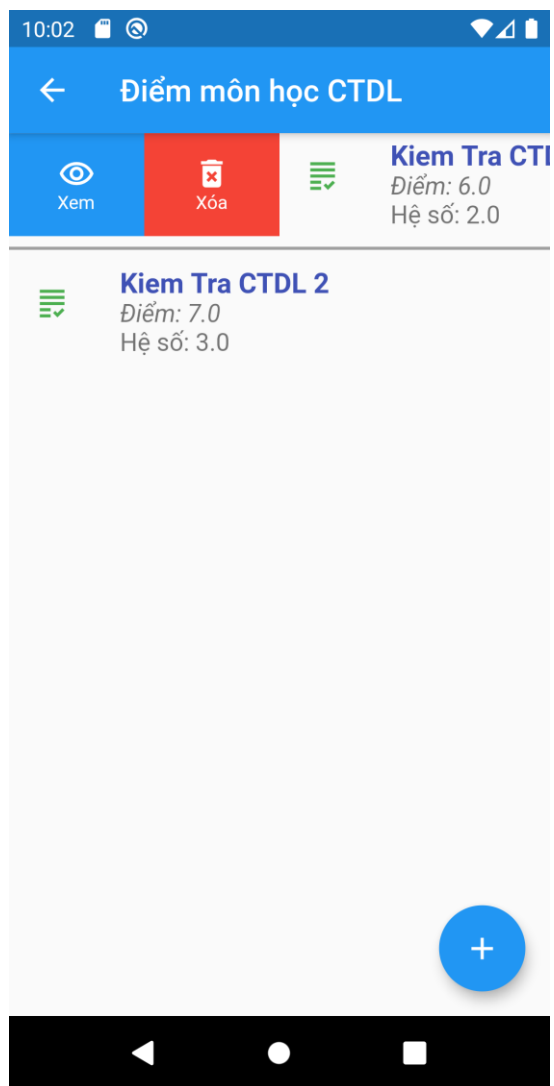
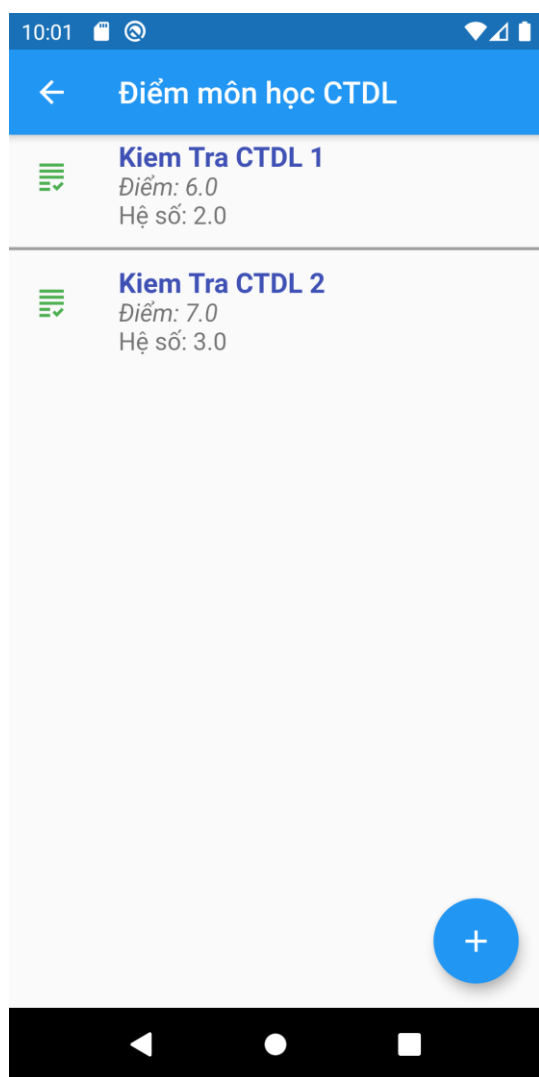
Hướng dẫn: Cài đặt các thư viện sau:

```

sqflite: ^1.3.2+2
path: ^1.7.0
provider: ^4.3.3
flutter_slidable: ^0.5.7

```





10:04

10:06

←

Cập nhật điểm

←

Nhập điểm

Tên bài kiểm tra

Kiem Tra CTDL 1

Tên bài kiểm tra

Hệ số

2.0

Hệ số

Điểm

6.0

Điểm

Cập nhật

Cancel

Thêm

Cancel

1

2

3

-

4

5

6

⌋

7

8

9

⌫

,

0

.

✓

Bài tập 12: Ứng dụng Fruit Ap

Các trang bên client



Các trang bên Admin

11:03

DSSP



Mô tả: Hồng giòn Đà Lạt



id: 06

Tên: Me

Giá: 35000

Mô tả: Me chua, đảm bảo không bị Tào Tháo rượt

đa xanh Tiền Giang

Cập nhật

Xóa



id: 10

Tên: Mận

Giá: 32000

Mô tả: Mận đặc sản Tây Bắc



id: 01

Tên: Xoài

Giá: 25000

Mô tả: Xoài cát Hòa Lộc loại 1, rất chua



id: 03

Tên: Sầu riêng


Giá: 95000

Mô tả: Sầu riêng Ri6 Cái be, Tiền

+

11:03

Cập nhật sản phẩm



Chọn ảnh

Id

10

Tên

Mận

Giá

32000

Mô tả

Mận đặc sản Tây Bắc

Cập nhật

11:04

← Thêm sản phẩm

Chọn ảnh

Id

Tên

Giá

Mô tả

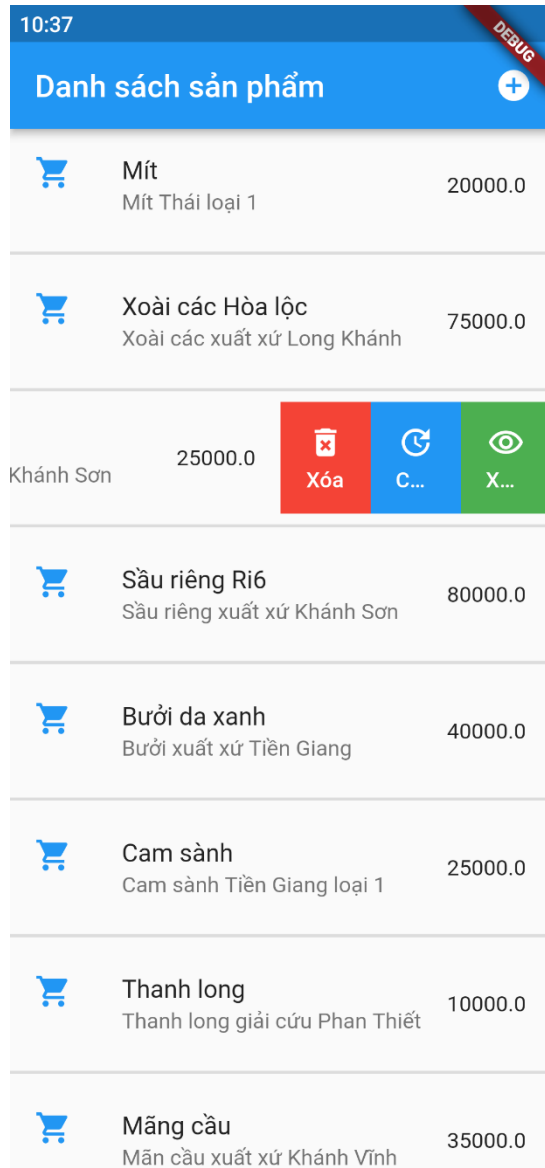
Thêm

Yêu cầu:

- Dữ liệu được lưu trữ trên firebase: Cloud Firestore, Firebase Storage/ Supabase (Nên sử dụng Supabase để lưu trữ dữ liệu, tệp tin ảnh)
- Dữ liệu phía client được truy cập theo Future
- Dữ liệu phía Admin được truy cập thời gian thực theo Stream/Channel. Sinh viên nên tìm hiểu cách truy cập dữ liệu thời gian thực theo Channel

Bài tập 13:

Thiết kế ứng dụng để hiển thị danh sách sản phẩm theo giao diện sau đây



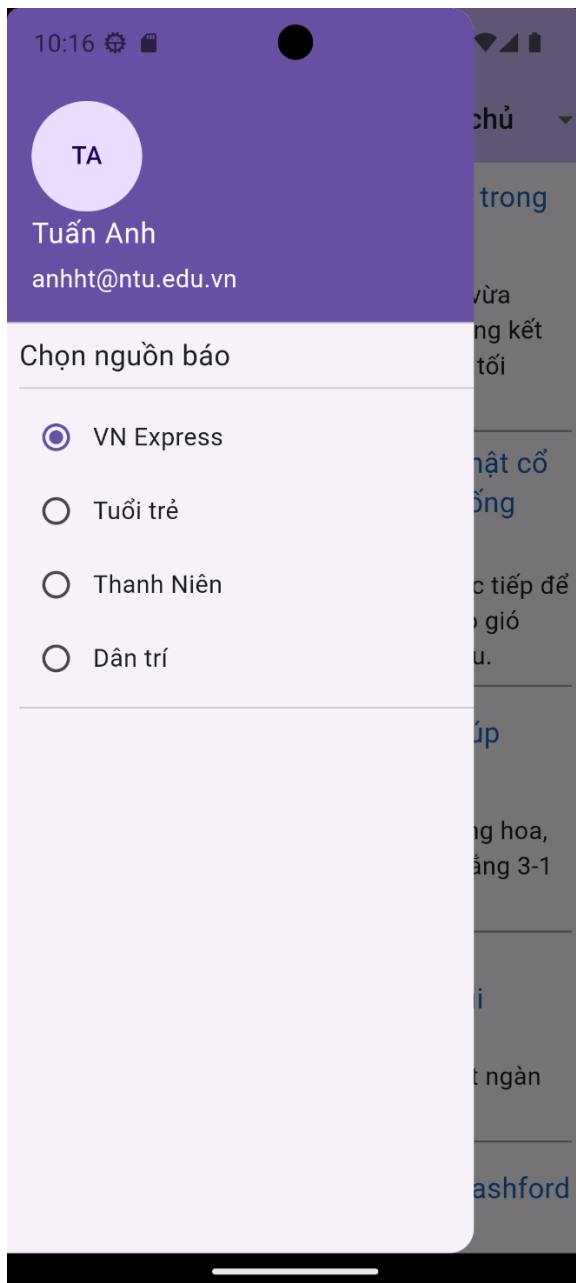
Hướng dẫn:

- Danh sách sản phẩm được lưu trong bộ nhớ RAM
- Sử dụng Provider/GetX để quản lý trạng thái là danh sách các sản phẩm như đã hướng dẫn trên lớp
- Sử dụng ListView để hiển thị danh sách dữ liệu (bọc ListView trong Consumer)
- Sử dụng thư viện Flutter Slidable để hiển thị từng mục của danh sách (Chú ý thuộc tính child của Slidable là một ListTile)
- Sinh viên có thể viết chức năng cho các nút hoặc không.

Bài tập tổng hợp 14:

Viết ứng dụng đọc báo RSS của trang VN Express.net





Yêu cầu: Về cơ bản sinh viên phải thực hiện được hai trang:

1. Hiển thị các mục rss item của một kênh rss trên một danh sách.
2. Hiển thị chi tiết một item khi chạm vào item của trang 1.
3. Các thành phần Drawer, DropdownButton là phần nâng cao tùy chọn

Hướng dẫn:

1. Sử dụng google search tìm kiếm các kênh báo RSS. Tìm hiểu cấu trúc của một kênh RSS (Các từ khóa: VNExpress RSS, Tuoitre RSS...)
2. Tải kênh RSS và chuyển nội dung của mỗi kênh thành một danh sách các Map<String, dynamic>
3. Chuyển danh sách các Map<String, dynamic> thành một danh sách các RssItem. Mỗi RssItem gồm:
 - a. title: Tiêu đề của bài báo

- b. imageUrl: liên kết ảnh của bài báo
 - c. description: Mô tả bài báo
 - d. link: liên kết của bài báo
 - e. pubDate: Thời gian xuất bản của bài báo
4. Viết controller: RssController thực hiện các công việc
- a. Lấy RssItem
 - b. Danh sách các kênh báo
 - c. Danh sách các nguồn báo

Một số code tham khảo:

1. Đọc Rss từ một kênh

```
2. Future<List<dynamic>?> _fetchRSS(String rssURL) async{6
    final response = await http.get(Uri.parse(rssURL));
    if(response.statusCode == 200){
        final xml2Json = Xml2Json();
        xml2Json.parse(utf8.decode(response.bodyBytes)); // Phân tích nội
dung xml của rss
        String rssJson = xml2Json.toParker(); // Chuyển nội dung của rss
thành chuỗi json
        Map<String, dynamic> jsonData = jsonDecode(rssJson); //chuyển
chuỗi json thành đối tượng Map
        return jsonData["rss"]["channel"]["item"]; // Trả về danh sách
cách Rss Item dạng Map
    }
    print("Lỗi đọc rss");
    return Future.error("Lỗi đọc Rss");
}
```

Bài tập tổng hợp 15:

Viết ứng dụng quản lý chi tiêu đơn giản trên Flutter

Yêu cầu: Dữ liệu quản lý local hoặc trên cloud.