1. **Phân biệt StatelessWidget và StatefulWidget:**

* **StatelessWidget:**

StatelessWidget là một widget không thay đổi trạng thái (immutable). Nó được xây dựng bằng cách chỉ định một lần và không thể thay đổi dữ liệu hoặc trạng thái bên trong nó sau khi được khởi tạo.

Khi dữ liệu hoặc trạng thái của ứng dụng thay đổi, StatelessWidget sẽ được xây dựng lại hoàn toàn từ đầu.

StatelessWidget phù hợp cho các phần tử giao diện đơn giản và không thay đổi, ví dụ như hiển thị dữ liệu tĩnh hoặc biểu tượng.

Ví dụ: Text, Image, Icon, Button.

* **StatefulWidget:**

StatefulWidget là một widget có thể thay đổi trạng thái (mutable). Nó có thể thay đổi dữ liệu và trạng thái bên trong widget sau khi khởi tạo ban đầu.

StatefulWidget phù hợp cho các phần tử giao diện có trạng thái động và phụ thuộc vào dữ liệu hoặc tương tác người dùng, ví dụ như form nhập liệu, danh sách có thể cuộn, trạng thái xử lý tiến trình.

StatefulWidget chứa hai lớp con quan trọng: StatefulWidget chính (stateful widget) và State (lớp trạng thái). StatefulWidget chịu trách nhiệm quản lý việc xây dựng giao diện và cập nhật State khi trạng thái thay đổi.

Khi dữ liệu hoặc trạng thái của StatefulWidget thay đổi, chỉ State được xây dựng lại, còn StatefulWidget không thay đổi.

Ví dụ: Form, ListView, Checkbox.

1. **Design**
   1. **Drawer(là phần sidebar)**
   * Cách khai báo:

Scaffold(

drawer: Drawer(

child: // Populate the Drawer in the next step.

),

);

Truyền key:

final GlobalKey<ScaffoldState> \_scaffoldKey = GlobalKey<ScaffoldState>();*Biến toàn cục(tương tự pop-up ở web , bấm phía ngoài thì state = false)*

return Scaffold(

      key: \_scaffoldKey,

Scaffold: là -> A blue rectangle with white text

Description automatically generated

* + Trong Drawer sẽ thường là ListView

drawer: Drawer(

        child: ListView(

          padding: EdgeInsets.zero,

          children: [

            const DrawerHeader(

              decoration: BoxDecoration(

                color: Colors.blue,

              ),

              child: Text('Drawer Header'),

            ),

            ListTile(

              title: const Text('Item 1'),

              onTap: () {

                // Handle item 1 click

                \_scaffoldKey.currentState?.openEndDrawer(); *Sử dụng key để đóng hoặc mở drawer*

              },

            ),

            ListTile(

              title: const Text('Item 2'),

              onTap: () {

                // Handle item 2 click

                \_scaffoldKey.currentState?.openEndDrawer();

              },

            ),

            // Add more ListTiles for additional items

          ],

        ),

      ),

* 1. **Snackbar(là phần toast nhỏ ở phía dười cùng màn hình)**

**A black rectangular object with white border

Description automatically generated**

Tạo Snackbar =>

class SnackBarToast extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Center(

      child: ElevatedButton(

        onPressed: () {

          const snackBarMessage = SnackBar(content: Text('Yay! A SnackBar!'));

          // Find the ScaffoldMessenger in the widget tree

          // and use it to show a SnackBar.

          ScaffoldMessenger.of(context).showSnackBar(snackBarMessage); *Show snackbar khi bấm button*

        },

        child: const Text('Show SnackBar'),

      ),

    );

  }

}

* 1. **Update the UI based on orientation(thay đổi UI theo chiều đthoai)**
  + Cách khai báo:

return OrientationBuilder(

      builder: (context, orientation) {  
 }

Sử dụng GridView để hiển thị list nếu mhinh dọc thì 2 cột , màn hình ngang thì 3 cột => khai báo cột crossAxisCount

* + Cách khai báo GridView:

return GridView.count(

          crossAxisCount: orientation == Orientation.portrait ? 2 : 3,

          // Generate 100 widgets that display their index in the List.

          children: List.generate(100, (index) {

            return Center(

              child: Text(

                'Item $index',

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

              ),

            );

          }),

        );

* 1. **Theme**
  + Cách khai báo:

void main() {

  runApp(

    MaterialApp(

      theme: ThemeData(

        primaryColor: Colors.blue,

        colorScheme: ColorScheme.fromSwatch(

          primarySwatch: Colors.red,

          accentColor: Colors.orange,

        ),

        textTheme: const TextTheme(

          headline6: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),

          // Add more text styles as needed

        ),

      ),

      home: MyApp(),

    ),

  );

}

* 1. **Tabs**
  + Cách khai báo:
* return DefaultTabController(
* length: 3, *Số lượng tab*
* child:

Sử dụng

class TabExample extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return DefaultTabController(

        length: 3,

        child: Scaffold(

          appBar: AppBar(

            bottom: const TabBar(

              tabs: [

                Tab(icon: Icon(Icons.directions\_car)),

                Tab(icon: Icon(Icons.directions\_transit)),

                Tab(icon: Icon(Icons.directions\_bike)),

              ],

            ),

            title: const Text("Tab Demo"),

          ),

          body: const TabBarView(

            children: [

              Icon(Icons.directions\_car),

              Icon(Icons.directions\_transit),

              Icon(Icons.directions\_bike),

            ],

          ),

        ));

  }

}

1. **Form**
   1. **Form**
   * Build a form with validation

Khởi tạo statefulWidget :

Mỗi class extends từ statefulWidget thì đại diện cho 1 form

class exampleForm extends StatefulWidget {

  @override

  State<StatefulWidget> createState() => //return Form

}

// Create a Form widget.

class MyCustomForm extends StatefulWidget {

  const MyCustomForm({super.key});

  @override

  MyCustomFormState createState() => MyCustomFormState();

}

Sử dụng key để xác định form đang sử dụng và validate

final \_formKey = GlobalKey<FormState>();

khởi tạo form

 @override

  Widget build(BuildContext context) {

    // Build a Form widget using the \_formKey created above.

    return Form(

      key: \_formKey,

      child: Column(

        crossAxisAlignment: CrossAxisAlignment.start,

        children: [

          TextFormField(

            // The validator receives the text that the user has entered.

            validator: (value) {

              if (value == null || value.isEmpty) {

                return 'Please enter some text';

              }

              return null;

            },

          ),

          Padding(

            padding: const EdgeInsets.symmetric(vertical: 16),

            child: ElevatedButton(

              onPressed: () { *Check khi bấm button*

                // Validate returns true if the form is valid, or false otherwise.

                if (\_formKey.currentState!.validate()) {

                  // If the form is valid, display a snackbar. In the real world,

                  // you'd often call a server or save the information in a database.

                  ScaffoldMessenger.of(context).showSnackBar(

                    const SnackBar(content: Text('Processing Data')),

                  );

                }

              },

              child: const Text('Submit'),

            ),

          ),

        ],

      ),

    );

  }

Kết quả:

A screenshot of a computer

Description automatically generated

* 1. **Text Field**
  + Cách khai báo

TextField(

              decoration: InputDecoration( *Decorate cho Textfield*

                border: OutlineInputBorder(),

                hintText: 'Enter a search term',

              ),

            ),

Sự khác nhau giữa text field và text form field   
A screenshot of a cell phone

Description automatically generated

* 1. **Handle changes(tương tự watcher của vuejs(bắt sự thay đổi))**
  + Cách khởi tạo

final myController = TextEditingController();

  @override

  void dispose() {

    myController.dispose();

    super.dispose();

  }

  @override

  void initState() {

    super.initState();

    // Start listening to changes.

    myController.addListener(\_printLatestValue);

  }

  void \_printLatestValue() {

    print('Second text field: ${myController.text}');

  }

Sử dụng:

Padding(

            padding: const EdgeInsets.symmetric(vertical: 16),

            child: TextField(

              decoration: const InputDecoration(

                border: OutlineInputBorder(),

                hintText: 'Enter a search term',

              ),

              controller: myController, *Sử dụng*

            ),

          ),

* 1. **Retrieve the value of a text field(tương tự v-model)**
  + Cách khởi tạo: Tìm cách tối ưu hơn nếu form nhiều trường ?????????

 final TextEditingController field1Controller = TextEditingController();

  final TextEditingController field2Controller = TextEditingController();

  final TextEditingController field3Controller = TextEditingController();

  @override

  void dispose() {

    field1Controller.dispose();

    field2Controller.dispose();

    field3Controller.dispose();

    super.dispose();

  }

Sử dụng:

 TextFormField(

            // The validator receives the text that the user has entered.

            validator: (value) {

              if (value == null || value.isEmpty) {

                return 'Please enter some text';

              }

              return null;

            },

            controller: field1Controller, *Gán mỗi field là 1 controller*

Xử lý button :

onPressed: () {

                // Validate returns true if the form is valid, or false otherwise.

                if (\_formKey.currentState!.validate()) {

                  // If the form is valid, display a snackbar

                  // ScaffoldMessenger.of(context).showSnackBar(

                  //   const SnackBar(content: Text('Processing Data')),

                  // );

                  //If the form is valid showing arlert

                  showDialog(

                      context: context,

                      builder: (context) {

                        return AlertDialog(

                          // Retrieve the text the that user has entered by using the

                          // TextEditingController.

                          content: Column(

                            crossAxisAlignment: CrossAxisAlignment.start,

                            mainAxisSize: MainAxisSize.min,

                            children: [

                              Text('Field 1: ${field1Controller.text}'),

                              Text('Field 2: ${field2Controller.text}'),

                              Text('Search term: ${field3Controller.text}'),

                            ],

                          ),

                          actions: [

                            TextButton(

                              onPressed: () {

                                Navigator.pop(context);

                              },

                              child: const Text('Close'),

                            ),

                          ],

                        );

                      });

                }

              },

1. **Gestures**
   1. **InkWell(text có thể tap)**
   * Sự khác nhau giữa opTap() và onPressed(): onTap() thường được sử dụng trong InkWell hoặc GestureDetector đơn giản hơn là 1 text có thể được tap . Còn onPressed() thường sử dụng ở button như FlatButton, RaisedButton
   * Cách khởi tạo:

class InkWellExample extends StatelessWidget {

  const InkWellExample({super.key});

  @override

  Widget build(BuildContext context) {

    // The InkWell wraps the custom flat button widget.

    return InkWell(

      // When the user taps the button, show a snackbar.

      onTap: () {

        ScaffoldMessenger.of(context).showSnackBar(const SnackBar(

          content: Text('Tap'),

        ));

      },

      child: const Padding(

        padding: EdgeInsets.all(12),

        child: Text(

            'Flat Button'), //Text can't tap , but have InkWell Text can tap

      ),

    );

  }

}

* 1. **GestureDetector**
  + Cách khởi tạo

class MyButton extends StatelessWidget {

  const MyButton({super.key});

  @override

  Widget build(BuildContext context) {

    // The GestureDetector wraps the button.

    return GestureDetector(

      // When the child is tapped, show a snackbar.

      onTap: () {

        const snackBar = SnackBar(content: Text('Tap'));

        ScaffoldMessenger.of(context).showSnackBar(snackBar);

      },

      // The custom button

      child: Container( *Sử dụng container để trang trí text like button*

        padding: const EdgeInsets.all(12),

        decoration: BoxDecoration(

          color: Colors.lightBlue,

          borderRadius: BorderRadius.circular(8),

        ),

        child: const Text('My Button'),

      ),

    );

  }

}

* 1. **Swipe(vuốt)**
  + Cách Khởi tạo :

return Dismissible(

            // Each Dismissible must contain a Key. Keys allow Flutter to

            // uniquely identify widgets.

            key: Key(item),

            // Provide a function that tells the app

            // what to do after an item has been swiped away.

            onDismissed: (direction) { *Hành động vuốt để xóa*

              // Remove the item from the data source.

              setState(() {

                items.removeAt(index);

              });

              // Then show a snackbar.

              ScaffoldMessenger.of(context)

                  .showSnackBar(SnackBar(content: Text('$item dismissed')));

            },

            // Show a red background as the item is swiped away.

            background: Container(color: Colors.red),

            child: ListTile(

              title: Text(item),

            ),

          );

1. **Image**
   1. **Image Network**
   * Cách khai báo:

Image.network()

A screen shot of a computer

Description automatically generatedKết quả:

* 1. **Fade in images**
  + Cách khai báo:

FadeInImage.memoryNetwork(

        placeholder: kTransparentImage,

        image: 'https://picsum.photos/250?image=9',

      ),

* 1. **cached images (tránh việc khi dung ứng dụng thì sẽ load hình lại)**
  + cách khai báo:

CachedNetworkImage(

        placeholder: (context, url) => const CircularProgressIndicator(),

        imageUrl: 'https://picsum.photos/250?image=9',

      ),

1. **List**
   1. **Horizon List**
   * Cách khai báo:

ListView(

        // This next line does the trick.

        scrollDirection: Axis.horizontal, *Khai Báo Horizon list*

        children: <Widget>[

          Container(

            width: 160,

            color: Colors.red,

          ),

          Container(

            width: 160,

            color: Colors.blue,

          ),

          Container(

            width: 160,

            color: Colors.green,

          ),

          Container(

            width: 160,

            color: Colors.yellow,

          ),

          Container(

            width: 160,

            color: Colors.orange,

          ),

        ],

      ),

1. **Navigation**
   1. **Animate a widget across screens(tương tự pop-up nhưng toàn màn hình)**
   * Cách khai báo:

Hero(

            tag: 'imageHero',

            child: Image.network(

              'https://www.lamborghini.com/sites/it-en/files/DAM/lamborghini/facelift\_2019/model\_detail/augmented-reality/huracan/huracan\_tecnica/hura\_tecnica\_ar\_01.png',

            ),

          ),

class ImageForClick extends StatelessWidget {

  const ImageForClick({super.key});

  @override

  Widget build(BuildContext context) {

    return Center(

        child: GestureDetector( *Sử dụng* *GestureDetector để có sự kiện tap vào hình ảnh*

      onTap: () {

        Navigator.push(context, MaterialPageRoute(builder: (context) {

          return const DetailScreen();

        })); *Navigation.Push dung để chuyển hướng , ở đây là để mở DetailScreen*

      },

      child: Hero(

        tag: 'imageHero',

        child: Image.network(

          'https://picsum.photos/250?image=9',

        ),

      ),

    ));

  }

}

class DetailScreen extends StatelessWidget {

  const DetailScreen({super.key});

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      body: GestureDetector(

        onTap: () {

          Navigator.pop(context);

        },

        child: Center(

          child: Hero(

            tag: 'imageHero',

            child: Image.network(

              'https://www.lamborghini.com/sites/it-en/files/DAM/lamborghini/facelift\_2019/model\_detail/augmented-reality/huracan/huracan\_tecnica/hura\_tecnica\_ar\_01.png',

            ),

          ),

        ),

      ),

    );

  }

}

* 1. **Navigate to a new screen and back**
  + Cách Khởi tạo:

class ButtonTo2ndSrceen extends StatelessWidget {

  const ButtonTo2ndSrceen({super.key});

  @override

  Widget build(BuildContext context) {

    return Center(

        child: ElevatedButton(

      onPressed: () {

        Navigator.push(

          context,

          MaterialPageRoute(builder: (context) => const SecondPage()),*MaterialPageRoute Chuyển hướng sang SecondPage()*

        );

      },

      child: const Text('Go to 2nd Page'),

    ));

  }

}

Second Page:

class SecondPage extends StatelessWidget {

  const SecondPage({super.key});

  @override

  Widget build(BuildContext context) {

    return Scaffold(

        appBar: AppBar(title: const Text("Navigation 2nd")),

        body: const Welcome2ndPage());

  }

}

class Welcome2ndPage extends StatelessWidget {

  const Welcome2ndPage({super.key});

  @override

  Widget build(BuildContext context) {

    return const Center(child: Text("This is 2nd Page of Navigation"));

  }

}

* 1. **Navigate with named routes**
  + Cách khởi tạo:

void main() {

  runApp(MaterialApp(

    home: const MyApp(),

    //Định nghĩa name cho các page

    routes: {'/second': (context) => const SecondPage()},

  ));

}

Sử dụng

Navigator.pushNamed(

          context,

          '/second',

        );

Lưu ý

Navigator.pushNamed(context, '/'); //Auto là trang home

* 1. **Pass arguments to a named route(Đẩy data từ trang này sang trang khác)**
  + Cách khởi tạo:

Khai báo đầu vào của trang 2 :

class SecondPage extends StatelessWidget {

  final String data;

  const SecondPage({Key? key, required this.data}) : super(key: key);

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: const Text("Navigation 2nd")),

      body: Welcome2ndPage(data: data), *Truyền data vào Welcome2ndPage*

    );

  }

}

Truyền đi ở trang 1 :

void main() {

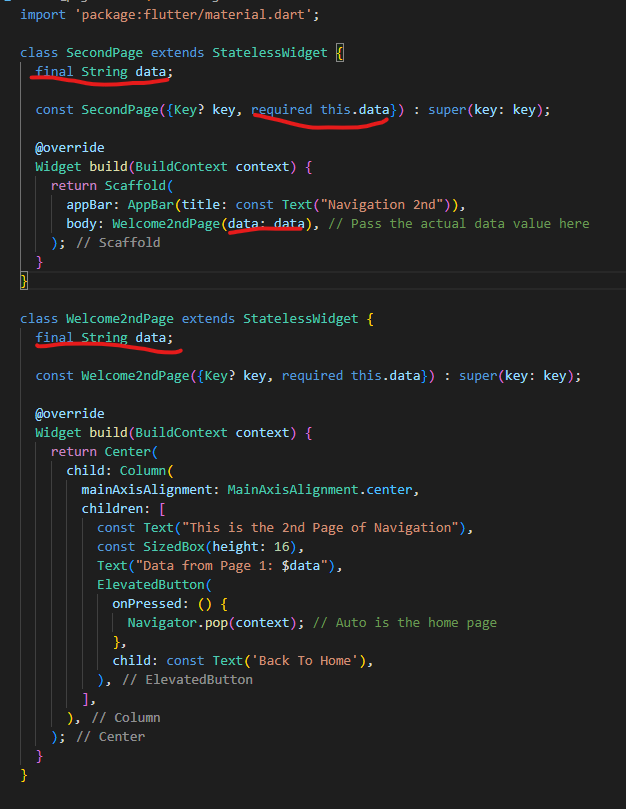
  runApp(MaterialApp(

    home: const MyApp(),

    routes: {'/second': (context) => const SecondPage(data: '')},

  ));

}

ở trang 2 :

Cách đẩy dữ liệu từ trang 1 :

DataForPass(fieldController: fieldController),

          ElevatedButton(

            onPressed: () async {

              String enteredText = fieldController.text;

              final result = await Navigator.push(

                context,

                MaterialPageRoute(

                    builder: (context) => SecondPage(data: enteredText)),

              ); *đẩy dữ liệu đi*

              if (result != null) { *check nếu được back về từ trang đã đẩy đi*

                ScaffoldMessenger.of(context).showSnackBar(

                  SnackBar(content: Text('Data received: $result')),

                );

              }

            },

            child: const Text('Go to 2nd Page'),

          ),

Dùng data ở trang nhận :

class Welcome2ndPage extends StatelessWidget {

  final String data;

  const Welcome2ndPage({Key? key, required this.data}) : super(key: key); *Nhận Data*

  @override

  Widget build(BuildContext context) {

    return Center(

      child: Column(

        mainAxisAlignment: MainAxisAlignment.center,

        children: [

          const Text("This is the 2nd Page of Navigation"),

          const SizedBox(height: 16),

          Text("Data from Page 1: $data"), *Dùng data*

          ElevatedButton(

            onPressed: () {

              Navigator.pop(

                  context, 'Data in page 2'); // Back to redirected page *Đẩy data về trang 1*

            },

            child: const Text('Back To Home'),

          ),

        ],

      ),

    );

  }

}

Nhận data ở trang 1 :

if (result != null) { // nếu được pop, tức chuyển về từ trang 2

                ScaffoldMessenger.of(context).showSnackBar(

                  SnackBar(content: Text('Data received: $result')),

                );

              }

1. **Networking**
   1. **Fetch data from the internet (GET METHOD)**
   * Cách khởi tạo:

Tạo model nhận giá trị từ api

class Album {

  final int userId;

  final int id;

  final String title;

  const Album({

    required this.userId,

    required this.id,

    required this.title,

  });

  factory Album.fromJson(Map<String, dynamic> json) {

    return Album(

      userId: json['userId'],

      id: json['id'],

      title: json['title'],

    );

  }

}

Gọi API:

Future<Album> fetchAlbum() async {

  final response = await http

      .get(Uri.parse('https://jsonplaceholder.typicode.com/albums/1')); *call api*

  if (response.statusCode == 200) {

    // If the server did return a 200 OK response,

    // then parse the JSON.

    return Album.fromJson(jsonDecode(response.body)); **parse json**

  } else {

    // If the server did not return a 200 OK response,

    // then throw an exception.

    throw Exception('Failed to load album');

  }

}

* **Future**: là một hoạt động bất đồng bộ đại diện cho 1 model , thực hiện các tác vụ bất đồng bộ sẽ có trong tương lai, thực hiện mà không crash ctrinh

Sử dụng dữ liệu:

Sử dụng statefulWidget

class Fetch extends State<FetchingDataExample> {

  late Future<Album> futureAlbum; *khai báo futureAlbum có thể null , được khởi tạo sau , khi dùng thì mới khởi tạo*

  @override

  void initState() {

    super.initState();

    futureAlbum = fetchAlbum(); *futureAlbum được gán bằng kqua khi gọi hàm fetchAlbum*

  }

  @override

  Widget build(BuildContext context) {

    return Center(

      child: FutureBuilder<Album>(

          future: futureAlbum,

          builder: (context, snapshot) {

            if (snapshot.hasData) { *Kiểm tra trạng thái nếu có dữ liệu và không có dữ liệu*

              return Text(snapshot.data!.title);

            } else if (snapshot.hasError) {

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

            }

            // By default, show a loading spinner.

            return const CircularProgressIndicator(); *nếu có dữ liệu , nhưng chưa fetch xong thì xuất hiện progress bar*

          }),

    );

  }

}

* 1. **Add authorization headers**

Future<Album> fetchAlbum() async {

  final response = await http.get(

    Uri.parse('https://jsonplaceholder.typicode.com/albums/1'),

    // Send authorization headers to the backend.

    headers: { *token*

      HttpHeaders.authorizationHeader: 'Basic your\_api\_token\_here',

    },

  );

  final responseJson = jsonDecode(response.body);

  return Album.fromJson(responseJson);

}

* 1. **Parse JSON in the background(Xử lý nhiều dữ liệu)**
  + Cách khởi tạo:

class Photo {

  final int albumId;

  final int id;

  final String title;

  final String url;

  final String thumbnailUrl;

  const Photo({

    required this.albumId,

    required this.id,

    required this.title,

    required this.url,

    required this.thumbnailUrl,

  });

  factory Photo.fromJson(Map<String, dynamic> json) {

    return Photo(

      albumId: json['albumId'] as int,

      id: json['id'] as int,

      title: json['title'] as String,

      url: json['url'] as String,

      thumbnailUrl: json['thumbnailUrl'] as String,

    );

  }

}

Gọi API:

Future<List<Photo>> fetchPhotos(http.Client client) async {

  final response = await client

      .get(Uri.parse('https://jsonplaceholder.typicode.com/photos'));

  // Use the compute function to run parsePhotos in a separate isolate.

  return compute(parsePhotos, response.body); *compute dùng để chạy hàm parse photo một cách riêng biệt , thay vì fetch xong mới parse , thì compute fetch ảnh nào xử lý ảnh đó có nghĩa là fetch ảnh nào thì gọi parsephotos cho ảnh đó*

}

// A function that converts a response body into a List<Photo>.

List<Photo> parsePhotos(String responseBody) {

  final parsed = jsonDecode(responseBody).cast<Map<String, dynamic>>();

  return parsed.map<Photo>((json) => Photo.fromJson(json)).toList();

}

Hiển thị photo :

class PhotosList extends StatelessWidget {

  const PhotosList({super.key, required this.photos});

  final List<Photo> photos;

  @override

  Widget build(BuildContext context) {

    return GridView.builder(

      gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(

        crossAxisCount: 2,

      ), *SliverGridDelegateWithFixedCrossAxisCount dung để điểu chỉnh số cột và tự động căn chỉnh chiều cao cho hình*

      itemCount: photos.length,

      itemBuilder: (context, index) {

        return Image.network(photos[index].thumbnailUrl);

      },

    );

  }

}

* 1. **Send data to the internet (POST METHOD)**
  + Cách khởi tạo:

Model:

class CreateAlbum {

  final int id;

  final String title;

  const CreateAlbum({required this.id, required this.title});

  factory CreateAlbum.fromJson(Map<String, dynamic> json) {

    return CreateAlbum(

      id: json['id'],

      title: json['title'],

    );

  }

}

API:

Future<CreateAlbum> createAlbum(String title) async {

  final response = await http.post(

    Uri.parse('https://jsonplaceholder.typicode.com/albums'),

    headers: <String, String>{

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

    },

    body: jsonEncode(<String, String>{

      'title': title,

    }),

  );

  if (response.statusCode == 201) {

    // If the server did return a 201 CREATED response,

    // then parse the JSON.

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

  } else {

    // If the server did not return a 201 CREATED response,

    // then throw an exception.

    throw Exception('Failed to create album.');

  }

}

Sử dụng:

class CreateAlbumDisplay extends State<CreateAlbumExample> {

  final TextEditingController \_controller = TextEditingController();

  Future<CreateAlbum>? \_futureAlbum;

  @override

  Widget build(BuildContext context) {

    return Container(

      alignment: Alignment.center,

      padding: const EdgeInsets.all(8),

      child: (\_futureAlbum == null) ? buildColumn() : buildFutureBuilder(), *nếu model có dữ liệu hiển thị title , ngược lại hiển thị form điền title*

    );

  }

  Column buildColumn() {

    return Column(

      mainAxisAlignment: MainAxisAlignment.center,

      children: <Widget>[

        TextField(

          controller: \_controller,

          decoration: const InputDecoration(hintText: 'Enter Title'),

        ),

        ElevatedButton(

          onPressed: () {

            setState(() {

              \_futureAlbum = createAlbum(\_controller.text); *gán value vào model , gọi api post*

            });

          },

          child: const Text('Create Data'),

        ),

      ],

    );

  }

  FutureBuilder<CreateAlbum> buildFutureBuilder() {

    return FutureBuilder<CreateAlbum>(

      future: \_futureAlbum,

      builder: (context, snapshot) {

        if (snapshot.hasData) {

          return Text(snapshot.data!.title);

        } else if (snapshot.hasError) {

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

        }

        return const CircularProgressIndicator();

      },

    );

  }

}

* 1. **Update data over the internet (UPDATE METHOD)**

Sử dụng lại model của Create

Gọi api:

Future<CreateAlbum> updateAlbum(String title) async {

  final response = await http.put(

    Uri.parse('https://jsonplaceholder.typicode.com/albums/1'),

    headers: <String, String>{

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

    },

    body: jsonEncode(<String, String>{

      'title': title,

    }),

  );

  if (response.statusCode == 200) {

    // If the server did return a 200 OK response,

    // then parse the JSON.

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

  } else {

    // If the server did not return a 200 OK response,

    // then throw an exception.

    throw Exception('Failed to update album.');

  }

}

* 1. **Delete data on the internet(DELETE METHOD)**
  + Cách khai báo:

Sử dụng lại model CreateAlbum

Future<CreateAlbum> deleteAlbum(String id) async {

  final http.Response response = await http.delete(

    Uri.parse('https://jsonplaceholder.typicode.com/albums/$id'),

    headers: <String, String>{

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

    },

  );

  if (response.statusCode == 200) {

    // If the server did return a 200 OK response,

    // then parse the JSON. After deleting,

    // you'll get an empty JSON `{}` response.

    // Don't return `null`, otherwise `snapshot.hasData`

    // will always return false on `FutureBuilder`.

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

  } else {

    // If the server did not return a "200 OK response",

    // then throw an exception.

    throw Exception('Failed to delete album.');

  }

}

Sử dụng:

class DeleteAlbumDisplay extends State<DeleteAlbumExample> {

  late Future<CreateAlbum> \_futureAlbum;

  @override

  void initState() {

    super.initState();

    \_futureAlbum = fetchAlbumUpdate(); *Sử dụng lại hàm fetch update album*

  }

  @override

  Widget build(BuildContext context) {

    return FutureBuilder<CreateAlbum>(

      future: \_futureAlbum,

      builder: (context, snapshot) {

        // If the connection is done,

        // check for response data or an error.

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

          if (snapshot.hasData) {

            return Column(

              mainAxisAlignment: MainAxisAlignment.center,

              children: <Widget>[

                Text(snapshot.data?.title ?? 'Deleted'),

                ElevatedButton(

                  child: const Text('Delete Data'),

                  onPressed: () {

                    setState(() {

                      \_futureAlbum = deleteAlbum(snapshot.data!.id.toString());

*Truyền Id của model , gọi hàm xóa*

                    });

                  },

                ),

              ],

            );

          } else if (snapshot.hasError) {

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

          }

        }

        // By default, show a loading spinner.

        return const CircularProgressIndicator();

      },

    );

  }

}

* 1. **Web Socket (Real-Time)**
  + Cách khai báo:

final \_channel = WebSocketChannel.connect(

    Uri.parse('wss://echo.websocket.events'), *Connect với websocket*

  );

Sử dụng:

class WebSocketDisplay extends State<WebSocketExample> {

  final TextEditingController \_controller = TextEditingController();

  final \_channel = WebSocketChannel.connect(

    Uri.parse('wss://echo.websocket.events'),

  );

  @override

  Widget build(BuildContext context) {

    return Padding(

        padding: const EdgeInsets.all(20),

        child: Column(

          crossAxisAlignment: CrossAxisAlignment.start,

          children: [

            Form(

              child: TextFormField(

                controller: \_controller,

                decoration: const InputDecoration(labelText: 'Send a message'),

              ),

            ),

            const SizedBox(height: 24),

            StreamBuilder(

              stream: \_channel.stream,

              builder: (context, snapshot) {

                return Text(snapshot.hasData ? '${snapshot.data}' : '');

              },

            ),*Lắng nghe dữ liệu từ socket server và display*

            FloatingActionButton(

              onPressed: \_sendMessage,

              tooltip: 'Send message',

              child: const Icon(Icons.send),

            ),

          ],

        ));

  }

  void \_sendMessage() {

    if (\_controller.text.isNotEmpty) {

      \_channel.sink.add(\_controller.text); *Gửi data lên server sink.add()*

    }

  }

  @override

  void dispose() {

    \_channel.sink.close(); *được gọi tự động khi StatefulWidget tree được tái xây dựng(đổi sang một trạng thái khác) => đóng kết nối , giải phóng bộ nhớ*

    \_controller.dispose(); *Giải phong bộ nhớ*

    super.dispose(); *gọi dispose cả ở lớp cha*

  }

}

A diagram of a system

Description automatically generated Lưu ý statefulWidget tree:

1. **Permisstion and Optimization**
   * Sử dụng const và final khi khai báo các giá trị không thay đổi.
   * Sử dụng ListView.builder thay vì ListView để hiển thị danh sách lớn, giúp tiết kiệm bộ nhớ.
   * Sử dụng setState một cách cẩn thận và chỉ khi cần thiết. Sử dụng các giải pháp quản lý trạng thái như Provider hoặc GetX để tránh việc gọi setState nhiều lần.
   * Sử dụng const trong việc tạo các widget không thay đổi.
   * Sử dụng const trong việc tạo các list, map, set không thay đổi.
   * Sử dụng FutureBuilder hoặc StreamBuilder để tải dữ liệu từ API một cách hiệu quả.
   * Sử dụng cả hai hàm dispose và didUpdateWidget để giải phóng tài nguyên khi widget bị hủy.
   * Tối ưu hóa hình ảnh và tệp tin đa phương tiện bằng cách sử dụng các công cụ nén và tối ưu hóa hình ảnh như flutter\_image\_compress hoặc flutter\_svg để giảm kích thước tệp và tăng tốc độ tải xuống.
   * Sử dụng cơ chế caching cho dữ liệu tải từ API để giảm số lần gọi API và tăng tốc độ tải dữ liệu.
2. **Google Map**

* Cách Khởi tạo:

Thêm gói google\_maps\_flutter

Đăng kí key google map: https://console.cloud.google.com/

Cấu hình file manifest

<meta-data

android:name="com.google.android.geo.API\_KEY"

android:value="YOUR\_API\_KEY" />

Cấu hình file AppDelegate.swift

GMSServices.provideAPIKey("YOUR\_API\_KEY")

* A screen shot of a computer

  Description automatically generatedSử dụng:

1. **Provider**

* Cách khởi tạo:

Tạo model để hứng dữ liệu:

class User {

  final String name;

  final int age;

  User(this.name, this.age);

}

Khởi tạo provider:

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return ChangeNotifierProvider<UserProvider>(

*ChangeNotifierProvider<UserProvider> có ý nghĩa là tạo ra một Provider của lớp UserProvider để cung cấp dữ liệu cho các widget trong ứng dụng*

      create: (\_) => UserProvider*(),(anonymous function) để tạo một phiên bản mới của UserProvider khi cần thiết.*

      child: MaterialApp(

        title: 'Ứng dụng quản lý người dùng',

        initialRoute: '/',

        routes: {

          '/': (context) => const UserListScreen(),

          '/add': (context) => const AddUserScreen(),

        },

      ),

    );

  }

}

class UserProvider with ChangeNotifier {

  final List<User> \_users = [];

  List<User> get users => \_users;

  void addUser(User user) {

    \_users.add(user);

    notifyListeners();

  }

  void updateUser(User oldUser, User updatedUser) {

    final index = \_users.indexOf(oldUser);

    if (index != -1) {

      \_users[index] = updatedUser;

      notifyListeners();

    }

  }

  void deleteUser(User user) {

    \_users.remove(user);

    notifyListeners();

  }

}

**notifyListeners() :** notifyListeners() được gọi trong lớp kế thừa từ ChangeNotifier, gọi **notifyListeners()**, Provider sẽ thông báo rằng dữ liệu đã thay đổi và yêu cầu các widget nghe lên Provider cập nhật giao diện.

@override

  Widget build(BuildContext context) {

    var userProvider = Provider.of<UserProvider>(context); *truy cập đối tượng UserProvider từ Provide từ cây widget hiện tại và lưu trữ nó trong biến userProvider.*

    var users = userProvider.users; *truy cập danh sách người dùng từ đối tượng* ***UserProvider*** *thông qua getter* ***users****.*

    return Scaffold(

      appBar: AppBar(

        title: const Text('Danh sách người dùng'),

      ),

      body: Column(

        children: [

          Expanded(

            child: ListView.builder(

              itemCount: users.length,

              itemBuilder: (context, index) {

                var user = users[index];

                return ListTile(

                  title: Text(user.name),

                  subtitle: Text('Tuổi: ${user.age}'),

                  onLongPress: () {

                    \_showOptionsDialog(context, user);

                  },

                );

              },

            ),

          ),

          FloatingActionButton(

            onPressed: () {

              Navigator.pushNamed(context, '/add');

            },

            child: const Icon(Icons.add),

          ),

        ],

      ),

    );

  }

Thay đổi value trong provider:

Màn hình add user:

TextFormField(

                controller: \_ageController,

                decoration: const InputDecoration(labelText: 'Tuổi'),

                keyboardType: TextInputType.number,

                validator: (value) {

                  if (value!.isEmpty) {

                    return 'Vui lòng nhập tuổi';

                  }

                  return null;

                },

              ),

              const SizedBox(height: 16.0),

              ElevatedButton(

                onPressed: () {

                  if (\_formKey.currentState!.validate()) {

                    var name = \_nameController.text;

                    var age = int.parse(\_ageController.text);

                    var userProvider =

                        Provider.of<UserProvider>(context, listen: false);*Khai báo*

                    userProvider.addUser(User(name, age)); *gọi hàm add*

                    Navigator.pop(context);

                  }

                },

                child: const Text('Thêm'),

              ),

1. **GetX**

* Điểm đặc biệt :
  + State management và redirect dễ sử dụng hơn
* Cách khởi tạo:

get: ^4.1.4

Controller:

Tạo và quản lý các biến, thuộc tính và phương thức liên quan đến trạng thái của ứng dụng

import 'package:get/get.dart';

class CounterController extends GetxController {

  var count = 0.obs; *phương thức .obs được gọi trên một giá trị để tạo một biến thuộc loại "observable". Có nghĩa là biến sẽ được theo dõi và tự động cập nhật giao diện khi giá trị của nó thay đổi.*

  void increment() {

    count++;

  }

  void redirectToDetailScreen() {

    Get.toNamed('/detail', arguments: count.value);

  }

}

class MyApp extends StatelessWidget {

  MyApp({super.key});

  final CounterController counterController = Get.put(CounterController()); *Get.put(CounterController()) đăng ký CounterController với GetX. Cho phép sử dụng counterController trong toàn bộ ứng dụng thông qua cơ chế dependency injection của GetX. Bất kỳ widget nào cần truy cập và sử dụng CounterController có thể lấy nó bằng cách sử dụng Get.find() hoặc Get.put(CounterController())*

  @override

  Widget build(BuildContext context) {

    return GetMaterialApp(

      title: 'GetX State and Redirect Example',

      initialRoute: '/',

      routes: {

        '/': (context) => HomeScreen(),

        '/detail': (context) => const DetailScreen(),

      },

    );

  }

}

@override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: const Text('Home Screen'),

      ),

      body: Center(

        child: Column(

          mainAxisAlignment: MainAxisAlignment.center,

          children: [

            Obx(() => Text(

                  'Count: ${counterController.count.value}',

                  style: const TextStyle(fontSize: 24),

                )), ***Obx*** *để theo dõi sự thay đổi của biến* ***count*** *trong* ***CounterController*** *và cập nhật giao diện khi giá trị thay đổi.*

            const SizedBox(height: 16),

            ElevatedButton(

              onPressed: () {

                counterController.increment();

              },

              child: const Text('Increment'),

            ),

            const SizedBox(height: 16),

            ElevatedButton(

              onPressed: () {

                counterController.redirectToDetailScreen(); *Chuyển trang của GetX đơn giản để sử dụng*

              },

              child: const Text('Go to Detail'),

            ),

          ],

        ),

      ),

    );

  }