HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÁO CÁO MÔN THỰC TẬP CƠ SỞ

Đề tài: Xây dựng phần mềm ghi chú đơn giản

Giảng viên hướng dẫn	**: Kim Ngọc Bách**
Họ và tên sinh viên	**: Lê Minh Quang**
Mã sinh viên	**: 07**
Lớp	**: D21CQCN02-**

Xây dựng phần mềm ghi chú đơn giản

LỜI CẢM ƠN

Lời đầu tiên, em xin được gửi lời cảm ơn sâu sắc đến **Học viện Công nghệ Bưu chính Viễn thông** đã đưa môn học vào trong chương trình giảng dạy. Đặc biệt, chúng em xin gửi lời cảm ơn chân thành nhất đến giảng viên bộ môn - thầy **Kim Ngọc Bách**, vì đã truyền đạt những kiến thức quý báu trong suốt thời gian họctập vừa qua.

Tuy nhiên, dù đã rất cố gắng nhưng do thời gian có hạn nên chắc rằng bài báo cáo của em khó tránh khỏi những thiếu sót. Em rất mong nhận được sự thông cảm và đóng góp ý kiến của quý thầy để bài báo cáo của em được hoàn thiện hơn.

Chúng em xin chân thành cảm ơn!

Hà Nội, ngày 28 tháng 05 năm 2024

Sinh viên thực hiện

Lê Minh Quang – B21DCCN626

Chương 1. Đặt Vấn Đề

1.1. Giới thiệu tổng quan:

Trong thời đại công nghệ số hiện nay, việc quản lý và lưu trữ thông tin cá nhân một cách hiệu quả là vô cùng quan trọng. Các ứng dụng ghi chú (note app) đã trở thành công cụ thiết yếu giúp người dùng tổ chức thông tin, quản lý công việc và lưu trữ những ý tưởng một cách thuận tiện và nhanh chóng. Sự phổ biến của các thiết bị di động cùng với sự phát triển mạnh mẽ của các nền tảng ứng dụng di động đã tạo điều kiện cho sự ra đời của nhiều ứng dụng ghi chú với các tính năng đa dạng và phong phú.

1.2. Ý nghĩa và ứng dụng:

Ứng dụng ghi chú không chỉ đơn thuần là nơi để ghi lại các thông tin ngắn gọn, mà còn có thể được sử dụng để lập kế hoạch, theo dõi tiến độ công việc, lưu trữ thông tin quan trọng. Việc phát triển một ứng dụng ghi chú hiệu quả, dễ sử dụng và tích hợp nhiều tính năng hữu ích có thể giúp người dùng tối ưu hóa việc quản lý thông tin cá nhân và nâng cao hiệu suất làm việc.

1.3. Mục tiêu báo cáo:

Dự án Note App được xây dựng nhằm mục đích tạo ra một ứng dụng ghi chú đơn giản nhưng hiệu quả, đáp ứng nhu cầu cơ bản của người dùng về việc lưu trữ và quản lý thông tin cá nhân. Ứng dụng sẽ có các chức năng chính như:

- Thêm ghi chú mới.
- Sửa đổi ghi chú hiện có.
- Xóa ghi chú.
- Tìm kiếm ghi chú theo từ khóa.

1.4. Phạm vi báo cáo:

Báo cáo này sẽ trình bày quá trình xây dựng và phát triển ứng dụng Note App, từ việc tìm hiểu các công nghệ liên quan đến việc áp dụng chúng trong thực tế. Báo cáo được chia thành các phần chính như sau:

- Giới thiệu về lý thuyết và công nghệ sử dụng trong dự án.
- Ứng dụng lý thuyết và công nghệ vào việc phát triển ứng dụng.
- Quá trình cài đặt và thử nghiệm ứng dụng.
- Đánh giá kết quả đạt được, những khó khăn gặp phải và hướng phát triển trong tương lai.

Chương 2. Giới Thiệu Về Lý Thuyết/Công Nghệ

2.1. Giới Thiệu Về Flutter

Flutter là gì?

Flutter là SDK (sử dụng ngôn ngữ Dart) do Google tạo ra nhằm tạo ra các giao diện chất lượng cao trên iOS và Android trong khoảng thời gian ngắn. Tại thời điểm ra mắt vào năm 2018, Flutter chủ yếu hỗ trợ phát triển ứng dụng di động. Nhưng hiện nay, Flutter hỗ trợ phát triển trên 6 nền tảng sau: iOS, Android, web, Windows, MacOS và Linux.

Ưu điểm và nhược điểm:

Những ưu điểm nổi bật và nhược điểm cần lưu ý.

Dưới đây là một số điểm nổi trội của Flutter trong vai trò một khung phát triển đa nền tảng:

- Hiệu suất gần với phát triển ứng dụng gốc. Flutter sử dụng ngôn ngữ lập trình Dart và biên dịch thành mã máy. Các thiết bị máy chủ hiểu được mã này, điều này đảm bảo hiệu suất nhanh và hiệu quả.
- Kết xuất nhanh, nhất quán và có thể tùy chỉnh. Thay vì dựa vào các công cụ kết xuất theo nền tảng, Flutter sử dụng thư viện đồ họa Skia nguồn mở của Google để kết xuất UI. Điều này mang đến cho người dùng phương tiện trực quan nhất quán cho dù họ sử dụng nền tảng nào để truy cập ứng dụng.
- Công cụ thân thiện với nhà phát triển. Google đã xây dựng Flutter chú trọng vào tính dễ sử dụng. Với các công cụ như tải lại nóng, nhà phát triển có thể xem trước các thay đổi mã sẽ như thế nào mà không bị mất trạng thái. Các công cụ khác như widget inspector giúp dễ dàng trực quan hóa và giải quyết các vấn đề với bố cục UI.

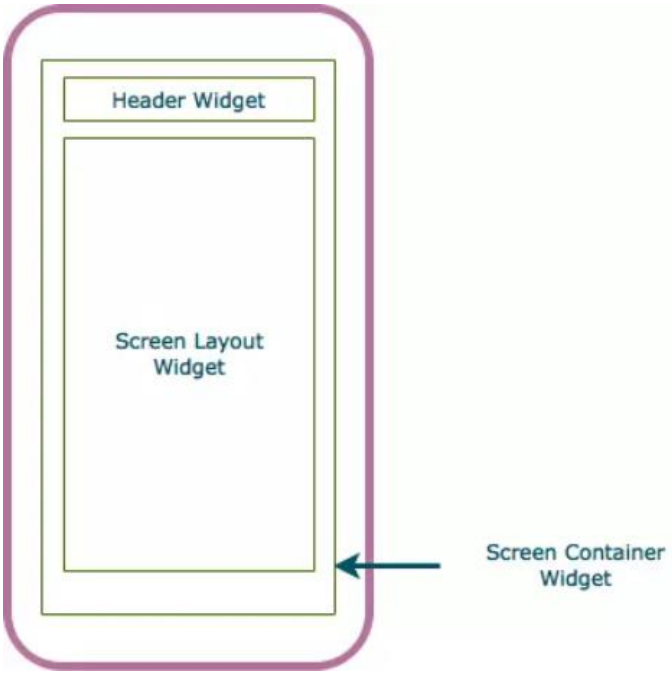
Các thành phần chính của Flutter:

2.1.1. Widgets

Flutter widgets được dựng lên với framework hiện đại mà nguồn cảm hứng được lấy từ *React*. Tất cả mọi thứ trong một ứng dụng Flutter đều là một Widget trong Flutter, từ những thứ đơn giản như Text hay Button cho đến các Screen Layouts.

Widgets sẽ mô tả view của người dùng như thế nào với cấu hình và trạng thái hiện tại. Khi trạng thái của Widget thay đổi, chúng sẽ được rebuild.

Các widgets sẽ được sắp xếp theo thứ tự phân cấp (gọi là Tree Widgets) hiển thị trên màn hình. Một sự sắp xếp tối thiểu các widgets trông như sau:



2.1.2. Dart

Flutter sử dụng ngôn dữ Dart – một ngôn ngữ do Google phát triển. Dart là một static type language nên nó là AOT (Ahead of Time), compile xong hết rồi mới chạy. Trong khi đó nó cũng là JIT (Just in Time) giống như các dynamic type language. Khi dev thì nó sử dụng JIT để hỗ trợ Hot Load và build release thì dùng AOT để tối ưu hiệu năng như một native code bình thường.

2.2. Giới Thiệu Về SQLite

2.2.1. SQLite là gì?

SQLite là hệ quản trị cơ sở dữ liệu (DBMS) tương tự MySQL,… . Điểm nổi bật của SQLite so với các DBMS khác là nó khá gọn, nhẹ, đơn giản, đặc biệt không cần mô hình server-client, không cần cài đặt, cấu hình hay khởi động nên sẽ không có user, password hay phân quyền trong SQLite database. Dữ liệu đươc lưu ở 1 file duy nhất.

Do đó, SQLite không được sử dụng với các hệ thống lớn nhưng với những hệ thống ở quy mô vùa và nhỏ thì SQLite không thua các DBMS khác về chức năng hay tốc độ.

2.2.2. Ưu điểm

- Không yêu cầu một quuy trình, hệ thống máy chủ riêng biệt
- Không cần cấu hình, có nghĩa là không cần thiết lập hoặc quản trị
- Một cơ sở dữ liệu hoàn chỉnh được lưu trữ trong một file disk đa nền tảng (cross-platform disk file) → phù hợp lưu trữ trên một điện thoại cá nhân
- Có trọng lượng nhẹ và rất nhỏ, dưới 400KiB nếu được cấu hình đầy đủ
- SQLite là khép kín (self-contanied), nghĩa là không phụ thuộc vào bên ngoài
- Các transaction hoàn toàn tuân thủ ACID
- Hỗ trợ hầu hết các tính năng ngôn ngữ truy vấn theo tiêu chuẩn SQL2
- SQLite được viết bằng ANSI-C và cung cấp API đơn giản và dễ sử dụng

⇨ *Như vậy SQLite là công cụ hữu ích dễ caching data và lưu trữ dưới dạng local, hoàn toàn phù hợp trong project này.*

2.2.3. Nhược điểm

- Việc phân quyền truy cập cơ sở dữ liệu chỉ có thể là quyền truy cập file của hệ thống
- Một số tính năng không được hỗ trợ: RIGHT JOIN; TRIGGER; phân quyền GRANT và REVOKE

2.3. Giới Thiệu Về GetX

2.3.1. GetX là gì?

Tổng quan về GetX, các tính năng nổi bật của GetX.

GetX là một thư viện quản lý trạng thái và điều hướng dễ sử dụng, được phát triển bởi thư viện GetX. Nó cung cấp một cách dễ dàng để quản lý trạng thái ứng dụng và điều hướng các màn hình.

2.3.2. State Management

Về cơ bản Get hoạt động cũng giống như Rx, cũng có các Observable và các component để

lắng nghe thay đổi của Observable

- Để khai báo một Observable với Get cũng hết sức đơn giản, có ba cách:

```
The first is using Rx{Type}.

    var count = RxString();


The second is to use Rx and type it with Rx<Type>

    var count = Rx<String>();


The third, more practical and easier approach, is just to add an .obs to
your variable.

    var count = 0.obs;

    Rx<int> count = 0.obs;
```

- GetxController

Mỗi một màn hình sẽ có một Controller extends từ GetxController. Controller này sẽ khai báo các Observable và xử lý toàn bộ logic của màn hình đó. Ex

```
class Controller extends GetxController {

  var count = 0.obs;

  void increment() {

    counter.value++;

  }

}
```

- Để lắng nghe các Observable ở trong View thì có thể sử dụng một trong các cách sau

1. GetX Component

```
GetX<Controller>(
```

```
  builder: (value) {
    print("count  rebuild");
    return Text('${controller.count.value}');
  },
),
```

Khi value của observable count thay đổi Widget sử dụng nó sẽ tự động được update mà không hề ảnh hưởng đến các widget khác trong tree widget

2. GetBuilder

Đây là cách simple để sử dụng Get State Manager.

- Trong controller

```
// Create controller class and extends GetxController

class Controller extends GetxController {
  int counter = 0;
  void increment() {
    counter++;
    update(); // use update() to update counter variable on UI when increment be called
  }
}
```

- Trong View

```
GetBuilder<Controller>(
  init: Controller(), // INIT IT ONLY THE FIRST TIME
  builder: (controller) => Text(
    '${controller.counter}',
```

```
  ),
)
```

Đối với GetBuilder chúng ta không cần khai báo kiểu Observable cho các field mà sử dụng trực tiếp kiểu cần dùng. Tuy nhiên cần phải gọi hàm update() có sẵn trong GetXController để update đến các Widget

Nếu bạn đã navigate đến màn hình khác nhưng vẫn cần sử dụng data của màn hình cũ thì chỉ cần đơn giản gọi controller của màn hình đó ra sử dụng. Đây là một điểm tuyệt vời của Get

Class a => Class B (has controller X) => Class C (has controller X)

```
class PageA extends GetWidget<ControllerA> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: FlatButton(
          onPress() =>   Navigator.push(context,
MaterialPageRoute(builder: (BuildContext context) => ClassB()));
        ),
      ),
}


class PageB extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: GetBuilder<ControllerA>(
```

```
          builder: (s) => Text('${s.counter}'),
        ),
      ),
}

// Hoặc sử dụng Get.find()

class PageC extends StatelessWidget {
ControllerA _controllerA = Get.find();
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child:  => Text('${_controllerA.counter}'),
      ),
}
```

3.Obx()

Cách cuối cùng là sử dụng Obx() componet. Đây sẽ là cách chủ đạo tôi dùng trong project này

```
// controller
class HomeController extends GetxController {
  var count = 0.obs;
  void increment() {
    counter.value++;
  }
```

```
}

class HomePage extends GetWidget<HomeController> {

  @override
  Widget build(BuildContext context) {
    return Column(
        children: [
             Obx(() => Text(_controller.count.value.toString()),
             FooterWidget(),
        ]
    );
  }
}



class FooterWidget extends GetWidget {
  DetailController _controller = Get.find();

  @override
  Widget build(BuildContext context) {
    return Center(
        child: Obx(() => Text(_controller.count.value.toString())
        );
  }
```

```
}
```

Với Getx - Obx các widget con sẽ dễ dàng sử dụng Controller của widget cha. Việc update data sẽ được tự động mỗi khi observable được thay đổi value. Cực kỳ dễ dàng phải không nào

- Binding

Còn một thành phần khác cực kỳ quan trọng của Get nữa đó là Binding.

Tương tự như Controller mỗi màn hình cũng sẽ có một Binding đây là nơi cung cấp các dependencies (repository, usecase, ...) cho màn hình đó kể cả controller cũng được provide ở đây.

```
class DetailBinding extends Bindings {
  @override
  void dependencies() {
    Get.lazyPut<DetailController>(() => DetailController());
  }
}
```

Binding sẽ được attach khi config route ở App.

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      enableLog: true,
      debugShowCheckedModeBanner: false,
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
```

```
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      initialBinding: DependenciesBinding(),
      initialRoute: '/search',
      getPages: [
        GetPage(
            name: '/search',
            page: () => SearchPage(),
            binding: SearchBinding()),
        GetPage(
            name: '/detail', page: () => DetailPage(), binding:
DetailBinding())
      ],
    );
  }
}
```

⇨ *Vậy để quản lý state với Get mỗi một màn hình trong Flutter sẽ gồm 3 component:* ***Controller*** *class(Extends từ GetXController),* ***Binding*** *class (Extend từ Binding) và* ***Widget*** *class(Extends từ GetWidget)*

2.3.3. Navigation Manager: Quản lý route và navigation trong ứng dụng.

GetX sẽ giúp chúng ta viết ít code hơn và tăng tốc độ làm việc với cú pháp điều hướng đơn giản

Ví dụ đơn giản: điều hướng đến 1 page nào đó

Using Flutter's Navigator:

```
Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => SomeScreen()),
);
```

Using GetX:

```
Get.to(SomeScreen());
```

Hơn nữa, trong GetX đã hỗ trợ sẵn cách hiện ra dialog, snackbar, bottomsheet mà không cần truyền vào context

2.3.4. Dependencies Manager: Cung cấp giải pháp dependencies injection tuyệt vời

Dependency management trong GetX là một trong những tính năng mạnh mẽ giúp quản lý và cung cấp các đối tượng phụ thuộc (dependencies) một cách hiệu quả trong ứng dụng Flutter. GetX cung cấp một cơ chế đơn giản và dễ sử dụng để quản lý sự phụ thuộc, giúp tránh được việc phải tạo các đối tượng phụ thuộc một cách thủ công hoặc phải sử dụng các thư viện phức tạp khác.

Các Khái Niệm Chính trong Dependency Management của GetX

- Dependency Injection (DI):
    - GetX cho phép dễ dàng cung cấp các đối tượng phụ thuộc tới các lớp khác thông qua Dependency Injection.
    - Với DI, có thể tạo và quản lý vòng đời của các đối tượng phụ thuộc một cách tự động.
- Bindings:
    - Bindings là cách GetX cung cấp để liên kết các phụ thuộc với các phần của ứng dụng.
    - Bindings giúp bạn khởi tạo và cung cấp các đối tượng phụ thuộc cho các trang (pages) hoặc các bộ điều khiển (controllers).

3. Phần Ứng Dụng Lý Thuyết/Công Nghệ Mình Tìm Hiểu

3.1. Giới Thiệu Bài Toán

Yêu cầu của ứng dụng:

Các chức năng chính như thêm, sửa, xóa, tìm kiếm các note.

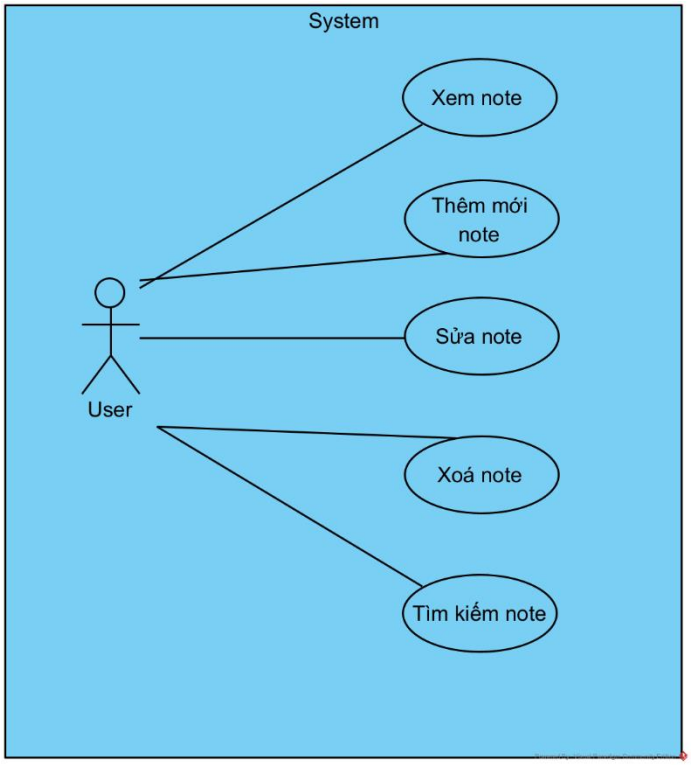
Phạm vi và giới hạn của dự án:

Project này được xây dựng chỉ để hoạt động trên điện thoại cá nhân. Lưu trữ các ghi chú mà không cần phân quyền, cấp quyền vào từng mục

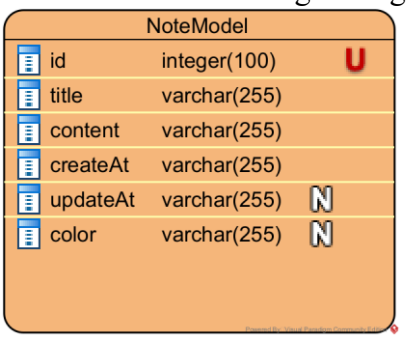
3.2. Giải Thích Cách Áp Dụng

Đầu tiên, ta sẽ phân tích chức năng của ứng dụng Note này

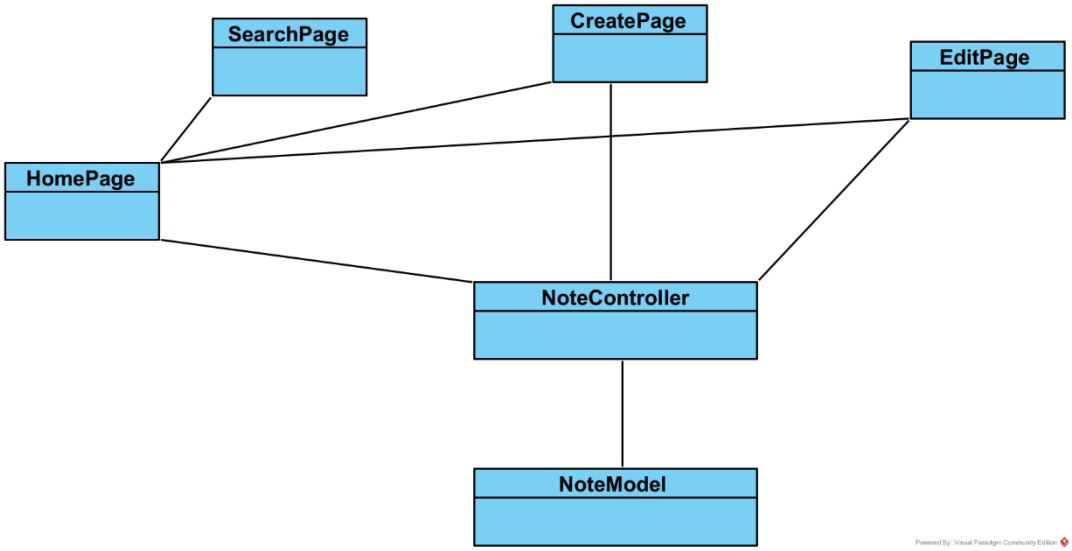
- UseCase



Thứ hai, ta sẽ phải thiết kế model Note có thể có những trường như thế nào



Thứ ba, có thể thiết kế sơ đồ lớp hệ thống như sau:



4. Cài Đặt và Thử Nghiệm

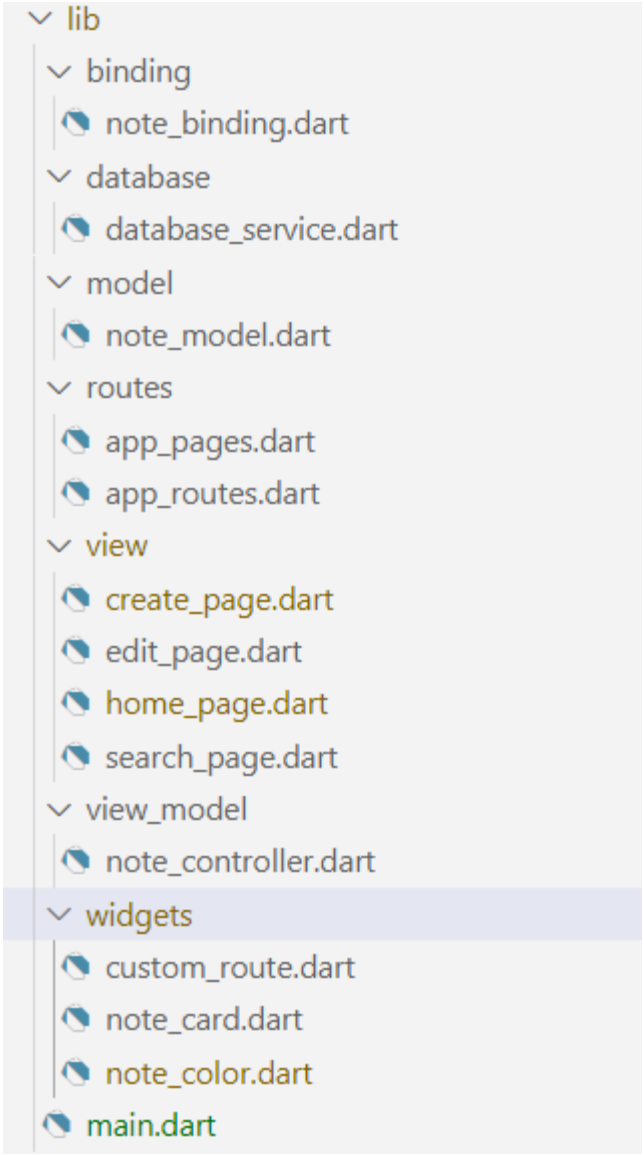
4.1.Môi Trường Phát Triển:

Vào file pubspec.yaml để thêm các thư viện cần thiết vào hệ thống

```yaml
dependencies:
  cupertino_icons: ^1.0.6
  flutter:
    sdk: flutter
  formatted_text: ^4.0.1
  get: ^4.6.6
  intl: ^0.19.0
  path: ^1.9.0
  sqflite: ^2.3.3+1
```

4.2. Cài Đặt Ứng Dụng

Ở đây, tôi sẽ áp dụng mô hình MVVM



4.2.1 Lưu Trữ Dữ Liệu Với Sqflite

Tạo 1 class DatabaseService để quản lý quy trình lưu trữ, truy xuất, quản lý dữ liệu

```
class DatabaseService {}
```

Để có thể chắc chắn tạo 1 database duy nhất, ta áp dụng cách sau:

```
//Singleton pattern
  static final DatabaseService _databaseService =
DatabaseService._internal();
  factory DatabaseService() => _databaseService;
  DatabaseService._internal();
```

Tạo bảng Notes, để lưu trữ notes của người dùng

```
//init database
  static Database? _database;
  final tableName = 'Notes'; //chỉ tạo 1 bảng lưu note
```

```
  Future<Database> get database async {
    if (_database != null) return _database!;
    _database = await _initDB();
    return _database!;
  }

  Future<Database> _initDB() async {
    final databasePath = await getDatabasesPath();
    final path = join(databasePath, tableName);

    return await openDatabase(
      path,
      version: 1,
      onCreate: _createDB,
    );
  }

  Future<void> _createDB(Database db, int version) async {
    await db.execute('''
    CREATE TABLE $tableName (
      id INTEGER PRIMARY KEY autoincrement,
      title TEXT,
      content TEXT,
      createAt TEXT,
      updateAt TEXT,
      color TEXT
    )'''
    );
  }
```

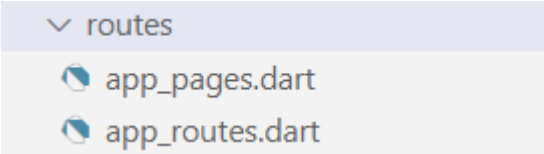
Các methods thêm, sửa, xoá được trình bày dưới đây trong class DatabaseService

```
//insert a note to the {note} table

  Future<int> insertNote(NoteModel note) async {
    final db = await _databaseService.database;
    return await db.insert(
      tableName,
      note.toMap(),
      conflictAlgorithm: ConflictAlgorithm.replace,
    );
  }
```

```dart
  //delete a note from the {note} table

  Future<void> deleteNote({required int id}) async {
    final db = await _databaseService.database;
    await db.delete(
      tableName,
      where: 'id = ?',
      whereArgs: [id],
    );
  }

  //update a note in the {note} table

  Future<void> updateNote(NoteModel note) async {
    final db = await _databaseService.database;
    await db.update(tableName, note.toMap(),
        where: 'id = ?',
        whereArgs: [note.id],
        conflictAlgorithm: ConflictAlgorithm.rollback);
  }

  // fetch all notes from the {note} table

  Future<List<NoteModel>> fetchAllNotes() async {
    final db = await _databaseService.database;
    final notes = await db.rawQuery(
        'SELECT * FROM $tableName ORDER BY createAt DESC, updateAt DESC');
//this query will return all of notes
    return notes.map((note) =>
NoteModel.fromSqliteDatabase(note)).toList();
  }

  // search note from the {note} table

  Future<List<NoteModel>> searchNote(String word) async {
    final db = await _databaseService.database;
    final searchingNotes = await db.query('''
     $tableName WHERE title LIKE '%$word%'
      ''');
    return searchingNotes
        .map((note) => NoteModel.fromSqliteDatabase(note))
        .toList();
  }
```

4.2.2. Cách GetX Quản Lý

- Quản lý route



- File app_routes.dart dùng để tạo tên đường dẫn theo format command của GetX

```
part of 'app_pages.dart';

abstract class Routes {
  Routes();

  static const HOMEPAGE = _Paths.HOMEPAGE;
  static const CREATEPAGE = _Paths.CREATEPAGE;
  static const EDITPAGE = _Paths.EDITPAGE;
  static const SEARCHPAGE = _Paths.SEARCHPAGE;

}

abstract class _Paths {

  static const HOMEPAGE = '/home';
  static const CREATEPAGE = '/create';
  static const EDITPAGE = '/edit';
  static const SEARCHPAGE = '/search';

}
```

- File app_routes.dart tạo ra class AppPages tạo các đường dẫn theo chuẩn GetX

```
part 'app_routes.dart';

class AppPages{
  AppPages._();

  static const INITIAL = Routes.HOMEPAGE;

  static final routes=[
    GetPage(name: _Paths.HOMEPAGE, page:() => HomePage()),
```

```
    GetPage(name: _Paths.CREATEPAGE, page:() => CreatePage()),
    GetPage(name: _Paths.EDITPAGE, page:(){
      final args = Get.arguments as NoteModel;
      return EditPage(args);

    }),
    GetPage(name: _Paths.SEARCHPAGE, page:() => SearchPage()),

  ];
}
```

- Quản lý trạng thái
    - Trong **binding/note_binding.dart** quản lý class NoteController sẽ sống trong suốt quá trình ứng dụng hoạt động

```
class NoteBinding extends Bindings {
  @override
  void dependencies() {
    Get.put(NoteController(), permanent: true);
  }
}
```

- Class NoteController kế thừa GetxController được tạo ra để lắng nghe sự kiện khi người dùng muốn tạo (đồng thời chọn màu bất kì từ list có sẵn cho backgr note), sửa, xoá note

```
class NoteController extends GetxController {
  DatabaseService db = DatabaseService();

  final _titleController =
Rx<TextEditingController>(TextEditingController());
  final _contentController =
Rx<TextEditingController>(TextEditingController());
  final randomColor = ColorNote.randomColorHex().obs;

  //fetch notes from local database
  Future<List<NoteModel>>? allNotes;



  @override
```

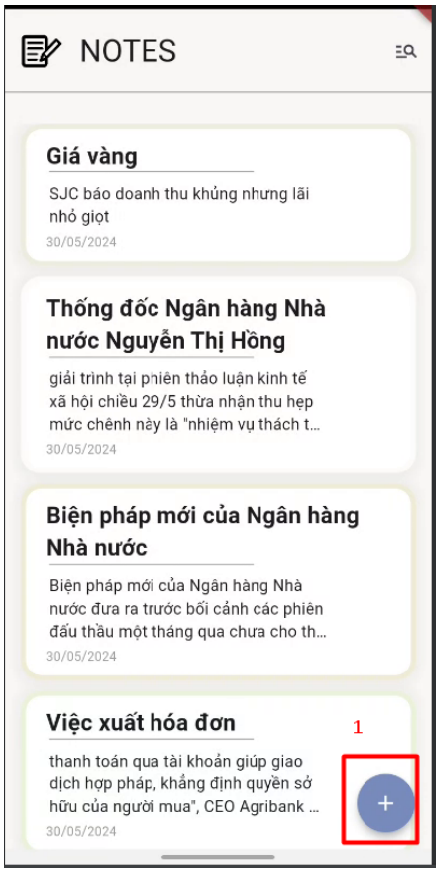
```
void onInit() {
  super.onInit();
  debugPrint('NoteController is initialized');

  //fetchNotes();
}

// void fetchNotes() {
//   // fetch notes from database
//   allNotes = db.fetchAllNotes();
// }

@override
void onReady() {
  // TODO: implement onReady
  super.onReady();
  debugPrint('NoteController is onReady');
}

//fetch data if note is not null
void fetchCurrentNote(NoteModel currentNote){
  titleController.text = currentNote.title;
  contentController.text = currentNote.content;
  update();
}

@override
void onClose() {
  // TODO: implement onClose
  super.onClose();
  debugPrint('NoteController is onClose');
}

//constructor

//getter
TextEditingController get titleController => _titleController.value;
TextEditingController get contentController => _contentController.value;

//method

/*******************
 * SAVE NEW NOTE
```

```
   * ***************/
  Future<void> _onSave() async {
    final title = titleController.text;
    final content = contentController.text;

    await db.insertNote(
      NoteModel(
        title: title,
        content: content,
        createAt: DateFormat('dd/MM/yyyy
HH:mm:ss').format(DateTime.now()),
        color: randomColor.value,
      ),
    );
  }

  void saveNewNote(){
    _onSave();
  }

  /*******************
   * SAVE CHANGES
   * ***************/

  Future<void> _onEdit(NoteModel currentNote) async {
    final title = titleController.text;
    final content = contentController.text;

    await db.updateNote(
      NoteModel(
          id: currentNote.id,
          updateAt: DateFormat('dd/MM/yyyy
HH:mm:ss').format(DateTime.now()),
          title: title,
          content: content,
          createAt: currentNote.createAt,
          color: currentNote.color,
          ),
    );
  }

  void saveEdit(NoteModel currentNote){
    _onEdit(currentNote);
```

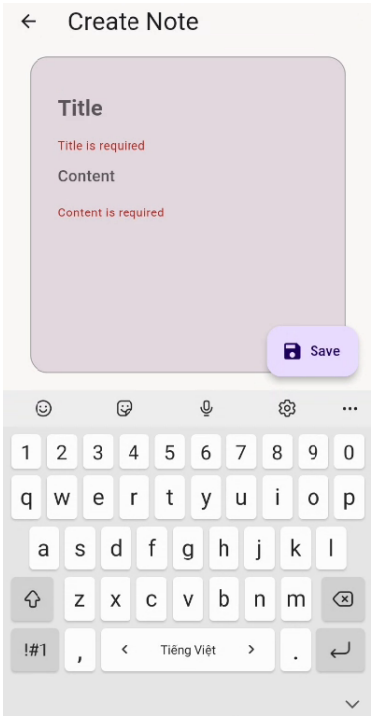
```
  }

  /*******************
   * DELETE NOTE
   * ***************/
  Future<void> delete_by_index(int id) async {
    await db.deleteNote(id: id);
  }

  void deleteNote(NoteModel note){
    delete_by_index(note.id!);
  }
}
```

4.3. Thử Nghiệm Các Chức Năng

4.3.1. Thêm mới ghi chú

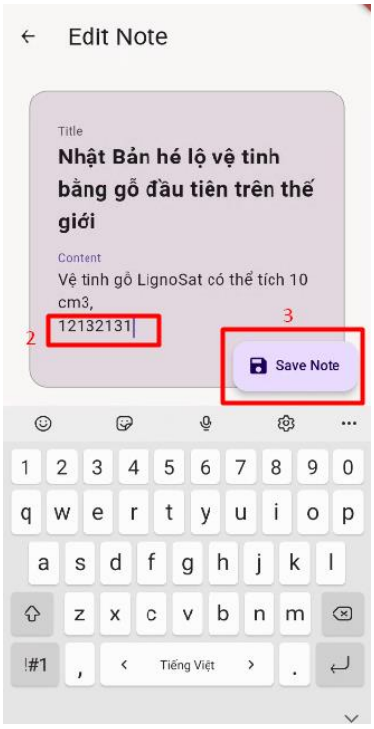


TH1: không đủ title hoặc content



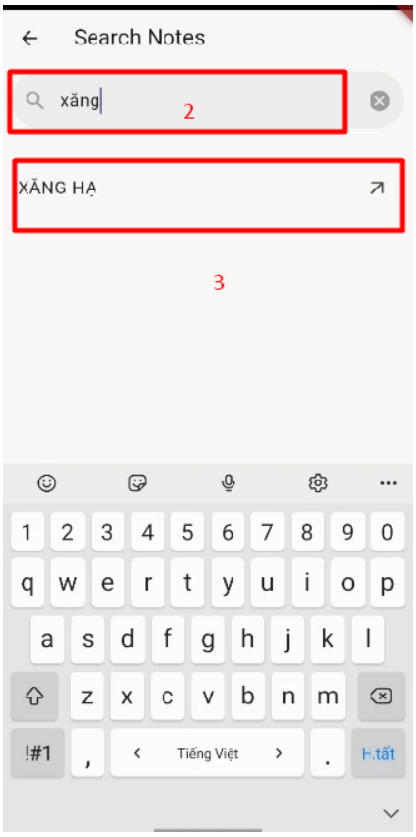
TH2: nếu đủ

4.3.2. Sửa ghi chú



4.3.3: Xoá ghi chú

4.3.4. Tìm kiếm ghi chú



5. Kết Luận

5.1. Kết Quả Đạt Được

LINK GIT: https://github.com/noname1288/Note_app_GetX

Sau quá trình nghiên cứu và phát triển, dự án Note App đã hoàn thành với các chức năng chính như thêm, sửa, xóa và tìm kiếm ghi chú. Ứng dụng được xây dựng trên nền tảng Flutter và sử dụng các package như Sqflite để lưu trữ cơ sở dữ liệu cục bộ và GetX để quản lý trạng thái và route. Những kết quả cụ thể đạt được bao gồm:

- **Giao diện người dùng**: Ứng dụng có giao diện thân thiện, dễ sử dụng và tương thích tốt trên nhiều thiết bị di động.
- **Chức năng cơ bản**: Tất cả các chức năng chính của ứng dụng như thêm, sửa, xóa và tìm kiếm ghi chú đều hoạt động ổn định.
- **Quản lý dữ liệu**: Cơ sở dữ liệu được quản lý hiệu quả bằng Sqflite, đảm bảo dữ liệu được lưu trữ an toàn và truy xuất nhanh chóng.
- **Quản lý trạng thái**: Sử dụng GetX giúp quản lý trạng thái ứng dụng một cách mạch lạc và dễ dàng mở rộng trong tương lai.

5.2. Khó Khăn và Hạn Chế

Trong quá trình phát triển ứng dụng, có một số khó khăn và hạn chế đã gặp phải:

- **Khả năng mở rộng**: Mặc dù ứng dụng đã hoàn thành các chức năng cơ bản, nhưng khả năng mở rộng để thêm các tính năng phức tạp hơn như đồng bộ hóa dữ liệu qua đám mây, nhắc nhở theo lịch trình còn hạn chế.
- **Hiệu suất**: Hiệu suất của ứng dụng có thể bị ảnh hưởng khi số lượng ghi chú tăng lên, đặc biệt là khi thực hiện các thao tác tìm kiếm trên một cơ sở dữ liệu lớn.
- **Kinh nghiệm và kiến thức**: Do hạn chế về kinh nghiệm và kiến thức ban đầu, việc tìm hiểu và áp dụng các công nghệ mới như Flutter và GetX đã gặp phải nhiều khó khăn và tốn nhiều thời gian.

5.3. Bài Học Kinh Nghiệm

Qua quá trình thực hiện dự án, tôi đã rút ra được nhiều bài học kinh nghiệm quý báu:

- **Kiến thức và kỹ năng**: Nâng cao kiến thức về Flutter, GetX và Sqflite, cũng như các kỹ năng lập trình ứng dụng di động.
- **Quản lý thời gian**: Tầm quan trọng của việc lập kế hoạch và quản lý thời gian hiệu quả trong quá trình phát triển dự án.
- **Giải quyết vấn đề**: Phát triển kỹ năng giải quyết vấn đề và tìm kiếm thông tin, tài liệu hỗ trợ từ cộng đồng lập trình.

6. Tài Liệu Tham Khảo

Danh Sách Tài Liệu Tham Khảo: Các sách, bài báo, tài liệu online đã tham khảo để thực hiện dự án.

Web:
https://docs.flutter.dev/
https://pub.dev/packages/get
https://pub.dev/packages/sqflite
https://m3.material.io/

Blog:
https://viblo.asia/p/gioi-thieu-ve-flutter-bWrZnNxrZxw
https://viblo.asia/p/flutter-sqlite-database-4dbZN1rkKYM
https://magz.techover.io/2022/04/05/sqlite-trong-flutter/