**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN TỬ VIỄN THÔNG**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**GetX - State Management và Navigation**

**Sinh viên thực hiện:**

**01. Lê Thành Duy Lớp: 22KTMT2**  **MSSV:**  106220250

**02. Nguyễn Ngọc Duy** **Lớp: 22KTMT2**  **MSSV:**  106220251

**Người hướng dẫn:**

**TS. Nguyễn Duy Nhật Viễn**

**Đà Nẵng, 2025.**

**THUYẾT MINH**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**RESTful API với HTTP và Dio**

BẢNG PHÂN CÔNG CÔNG VIỆC TRONG NHÓM

|  |  |  |  |
| --- | --- | --- | --- |
| STT | HỌ VÀ TÊN | NHIỆM VỤ | KHỐI LƯỢNG |
| 01 | LÊ THÀNH DUY | Tìm hiểu về GetX và State managememt  Tìm hiểu về DI, Obx widget và getx controller  Code phần State management cho app demo và ví dụ, giao diện | 50% |
| 02 | NGUYỄN NGỌC DUY | Tìm hiểu về navigation và routing  Trình bày lợi ích hiệu năng và trường hợp sử dụng  Code phần navigation và routing trong Demo và giao diện | 50% |

Link Github: https://github.com/duypear/Getx\_flutter

**Mục lục**

Contents

[**I. Tổng quan về GetX:** 1](#_Toc213752869)

[**II. State Management với GetX:** 1](#_Toc213752870)

[**1 GetxController và Reactive Programming** 1](#_Toc213752871)

[**1.1 Reactive Programming trong GetX** 1](#_Toc213752872)

[**1.2 GetXController** 2](#_Toc213752873)

[**1.3 Cách khởi tạo và sử dụng Controller** 2](#_Toc213752874)

[**1.4 Lifecycle của GetxController:** 2](#_Toc213752875)

[**1.5 Ưu điểm của GetxController và Reactive Programming** 4](#_Toc213752876)

[**2. Obx Widget và Dependency Injection** 4](#_Toc213752877)

[**2.1 Obx Widget:** 4](#_Toc213752878)

[**2.2 Dependency Injection:** 5](#_Toc213752879)

[**III. Navigation và Routing với GetX :** 6](#_Toc213752880)

[**1. Cấu hình routes:** 6](#_Toc213752881)

[**2. Các phương thức điều hướng cơ bản** 7](#_Toc213752882)

[**3. Truyền dữ liệu giữa các màn hình** 7](#_Toc213752883)

[**4. Quản lý giao diện tạm thời: SnackBar, Dialog, BottomSheet.** 7](#_Toc213752884)

[**5. Ví dụ tổng hợp đơn giản** 8](#_Toc213752885)

[**IV. Performance Benefits và Use Cases của GetX** 9](#_Toc213752886)

[**1. Hiệu năng nổi bật (Performance Benefits)** 9](#_Toc213752887)

[**2. Trường hợp sử dụng phù hợp (Use Cases)** 9](#_Toc213752888)

[**V. Demo ứng dụng mua sắm online:** 10](#_Toc213752889)

[**TÀI LIỆU THAM KHẢO** 10](#_Toc213752890)

**Giới thiệu**

Trong lập trình đa nền tảng với Flutter, việc quản lý trạng thái (state) và điều hướng (navigation) là hai vấn đề quan trọng ảnh hưởng trực tiếp đến khả năng mở rộng, hiệu năng và trải nghiệm người dùng của ứng dụng. Flutter cung cấp nhiều giải pháp cho quản lý trạng thái như setState, Provider, Bloc, Riverpod, tuy nhiên GetX nổi lên như một thư viện gọn nhẹ, dễ sử dụng và hiệu quả cao.

Báo cáo này trình bày tổng quan về thư viện GetX, cách sử dụng GetX cho quản lý trạng thái, điều hướng, và cơ chế dependency injection. Đồng thời, phần cuối sẽ minh họa ứng dụng GetX thông qua một ví dụ thực tế nhỏ.

# **I. Tổng quan về GetX:**

GetX là một thư viện mã nguồn mở của Flutter với ba thành phần chính:

* State Management: quản lý dữ liệu và tự động cập nhật giao diện khi dữ liệu thay đổi.
* Dependency Injection: giúp quản lý, khởi tạo và tái sử dụng các đối tượng trong toàn bộ ứng dụng.
* Route Management (Navigation): cung cấp cơ chế điều hướng mạch lạc, không phụ thuộc BuildContext.

Nhờ cơ chế reactive (lập trình phản ứng), GetX cho phép lập trình viên tập trung vào logic thay vì phải thao tác nhiều với widget tree, giúp mã ngắn gọn hơn, dễ bảo trì và tối ưu hiệu năng.

# **II. State Management với GetX:**

Trong Flutter, “state” (trạng thái) là dữ liệu mô tả giao diện người dùng tại một thời điểm. Khi state thay đổi, giao diện cần được cập nhật tương ứng. Flutter cung cấp nhiều cách quản lý state khác nhau như setState, Provider, Bloc, Riverpod,… Tuy nhiên, với các ứng dụng có quy mô lớn hoặc cần tối ưu hiệu năng, các cách này thường dẫn đến việc viết nhiều mã lặp lại và khó kiểm soát luồng dữ liệu.

GetX mang đến cách tiếp cận đơn giản và nhẹ hơn nhờ khả năng quản lý trạng thái phản ứng (reactive state management). Thay vì phải gọi setState để cập nhật giao diện, GetX tự động theo dõi và cập nhật giao diện mỗi khi dữ liệu thay đổi.

Ví dụ khi sử dụng phương pháp thông thường trong Flutter:

setState(() {

counter++;

});

Phương pháp này yêu cầu widget phải là StatefulWidget và phải gọi lại toàn bộ build() mỗi khi dữ liệu thay đổi. Trong khi đó, GetX giúp rút ngắn và tự động hóa quy trình này thông qua cơ chế phản ứng (reactivity).

## **1 GetxController và Reactive Programming**

### **1.1 Reactive Programming trong GetX**

Reactive Programming (lập trình phản ứng) là mô hình trong đó các thành phần giao diện tự động cập nhật khi dữ liệu thay đổi. Trong GetX, điều này được thực hiện thông qua các biến phản ứng (reactive variables) hay còn gọi là Rx variables.

Một biến trở thành phản ứng khi được thêm hậu tố .obs. Khi giá trị của biến này thay đổi, các widget sử dụng nó (thông qua Obx) sẽ tự động cập nhật giao diện.

Ví dụ:

var count = 0.obs;

var name = "Duy".obs;

var isActive = false.obs;

Ở ví dụ trên, mỗi khi giá trị của count, name hoặc isActive thay đổi, mọi widget đang lắng nghe các biến này sẽ được cập nhật lại tương ứng. Cơ chế này giúp loại bỏ hoàn toàn nhu cầu gọi setState() trong code.

### **1.2 GetXController**

GetxController là lớp được sử dụng để tách biệt phần xử lý logic và dữ liệu ra khỏi giao diện người dùng. Mỗi controller đảm nhận vai trò quản lý trạng thái cho một phần hoặc toàn bộ ứng dụng. Cách tiếp cận này giúp mã nguồn có cấu trúc rõ ràng, dễ bảo trì và tái sử dụng. Khi người dùng thực hiện thao tác, controller sẽ xử lý logic, cập nhật dữ liệu và thông báo cho giao diện biết để hiển thị lại các thay đổi.

Về mặt mô hình, GetxController tương tự như ViewModel trong kiến trúc MVVM (Model–View–ViewModel). Mô hình hoạt động của GetxController có thể mô tả như sau:  
Người dùng thao tác → Controller xử lý → Trạng thái thay đổi → Giao diện tự động cập nhật.

Khi người dùng thực hiện hành động (nhấn nút, nhập dữ liệu,...), controller sẽ xử lý logic, cập nhật giá trị của các biến reactive, và GetX sẽ tự động thông báo cho giao diện hiển thị lại phần liên quan.

Ví dụ:

import 'package:get/get.dart';

class CounterController extends GetxController {

  var count = 0.obs;

  void increment() => count++;

  void decrement() => count--;

}

Biến count trong ví dụ trên là một observable (đối tượng có thể quan sát được). Khi giá trị của count thay đổi, GetX sẽ thông báo cho các widget đang lắng nghe và cập nhật phần hiển thị tương ứng. Điều này được thực hiện thông qua các widget như Obx hoặc GetX.

### **1.3 Cách khởi tạo và sử dụng Controller**

GetX cung cấp nhiều phương thức để khởi tạo và truy cập controller, bao gồm Get.put(), Get.lazyPut(), và Get.find().

* Get.put(): tạo một instance của controller và lưu trữ trong bộ nhớ để sử dụng lại.
* Get.lazyPut(): chỉ tạo controller khi lần đầu tiên được gọi.
* Get.find(): truy cập instance của controller đã được khởi tạo

Ví dụ:

final CounterController controller = Get.put(CounterController());

Obx(() => Text("Giá trị hiện tại: ${controller.count}"));

Khi người dùng nhấn nút tăng hoặc giảm, giá trị count trong controller thay đổi, và Obx() sẽ tự động render lại phần UI chứa nó — hoàn toàn không cần gọi setState() như cách truyền thống.

### **1.4 Lifecycle của GetxController:**

GetxController có các phương thức vòng đời tương tự như widget, giúp kiểm soát việc khởi tạo và hủy dữ liệu:

|  |  |
| --- | --- |
| Phương thức | Chức năng |
| onInit() | Gọi khi controller được khởi tạo lần đầu tiên (dùng để load dữ liệu ban đầu). |
| onReady() | Gọi khi tất cả widget đã render xong (thường dùng để khởi chạy API hoặc animation). |
| onClose() | Gọi khi controller bị xóa khỏi bộ nhớ (dùng để giải phóng tài nguyên, đóng stream). |

**Ví dụ :**

class CounterController extends GetxController {

  var count = 0.obs;

  @override

  void onInit() {

    super.onInit();

    print("Controller đã khởi tạo");

  }

  @override

  void onClose() {

    print("Controller đã bị hủy");

    super.onClose();

  }

}

Nhờ cơ chế này, GetX có thể tự động quản lý bộ nhớ, chỉ giữ controller khi cần và hủy khi không còn sử dụng, giúp ứng dụng hoạt động mượt mà và tránh rò rỉ tài nguyên.

### **1.5 Ưu điểm của GetxController và Reactive Programming**

Việc sử dụng GetxController kết hợp với reactive programming mang lại nhiều lợi ích:

* Không cần gọi setState() trong các widget, giúp mã ngắn gọn và dễ đọc hơn.
* Hiệu năng cao do chỉ những widget liên quan đến biến thay đổi mới được cập nhật lại.
* Tách biệt rõ ràng giữa phần giao diện (View) và phần xử lý logic (Controller), dễ dàng bảo trì và mở rộng.
* Hỗ trợ quản lý trạng thái xuyên suốt navigation, giúp dữ liệu không bị mất khi chuyển giữa các màn hình.

## **2. Obx Widget và Dependency Injection**

Trong mô hình GetX, hai thành phần quan trọng nhất giúp kết nối giữa dữ liệu và giao diện là Obx widget và Dependency Injection (DI). Obx đảm nhiệm vai trò hiển thị và tự động cập nhật giao diện khi dữ liệu thay đổi, trong khi Dependency Injection giúp quản lý và chia sẻ các đối tượng (controller, service, repository, v.v.) trong toàn bộ ứng dụng một cách dễ dàng và hiệu quả.

### **2.1 Obx Widget:**

Obx là một widget đặc biệt của GetX, dùng để lắng nghe các biến phản ứng (reactive variables). Khi giá trị của biến thay đổi, Obx sẽ tự động rebuild phần giao diện chứa nó mà không cần gọi setState(). Đây là cơ chế quan trọng giúp GetX đạt được hiệu năng cao và loại bỏ sự phụ thuộc vào StatefulWidget.

Cấu trúc của Obx:

* Obx nhận vào một hàm (lambda function) trả về một widget.
* Hàm này được gọi lại mỗi khi giá trị của biến phản ứng trong hàm thay đổi.

Cú pháp tổng quát:

* Obx(() => WidgetHiểnThị());

Obx có thể được sử dụng linh hoạt trong bất kỳ widget nào, giúp code trở nên ngắn gọn, rõ ràng và tránh việc phải tạo StatefulWidget chỉ để cập nhật dữ liệu.

Ví dụ dùng Obx Widget:

class CounterController extends GetxController {

  var count = 0.obs; // biến phản ứng

  void increment() => count++;

}

class CounterPage extends StatelessWidget {

  final controller = Get.put(CounterController()); // đăng ký controller

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text('Ví dụ Obx Widget')),

    body: Center(

        child: Obx(() => Text(

          'Giá trị hiện tại: ${controller.count}',

          style: TextStyle(fontSize: 22),

        )),

      ),

      floatingActionButton: FloatingActionButton(

        onPressed: controller.increment,

        child: Icon(Icons.add),

      ),

    );

  }

count là một biến phản ứng (.obs) trong CounterController.

Obx widget lắng nghe giá trị của count. Khi count thay đổi, nội dung Text sẽ được cập nhật tự động.

Không cần dùng setState(), giúp mã ngắn gọn và hiệu năng tốt hơn.

Ưu điểm của Obx:

* Chỉ cập nhật đúng phần giao diện có dữ liệu thay đổi, không làm ảnh hưởng đến các widget khác.
* Giúp viết code ngắn gọn và dễ đọc.
* Không cần tạo StatefulWidget cho những phần giao diện đơn giản.

### **2.2 Dependency Injection:**

Dependency Injection (DI) là một kỹ thuật trong lập trình hướng đối tượng dùng để tách biệt việc khởi tạo và sử dụng đối tượng, giúp mã nguồn dễ quản lý, mở rộng và kiểm thử.  
Trong Flutter, GetX tích hợp sẵn cơ chế DI, cho phép lập trình viên quản lý và chia sẻ các controller hoặc service giữa nhiều phần khác nhau của ứng dụng một cách đơn giản và hiệu quả.

| **Phương thức** | **Mô tả** | **Khi sử dụng** |
| --- | --- | --- |
| **Get.put()** | **Khởi tạo và lưu đối tượng ngay lập tức** | **Dùng khi cần sẵn sàng ngay** |
| **Get.lazyPut()** | **Chỉ khởi tạo khi được gọi lần đầu** | **Dùng cho tài nguyên nặng, ít sử dụng** |
| **Get.putAsync()** | **Dành cho đối tượng cần khởi tạo bất đồng bộ (async)** | **Dùng cho service hoặc API** |

**Ví dụ sử dụng Dependency Injection trong GetX :**

import 'package:get/get.dart';

// Khai báo một lớp dịch vụ hoặc controller

class AuthService extends GetxService {

  var isLoggedIn = false;

  void login() => isLoggedIn = true;

}

// Inject đối tượng vào bộ nhớ của GetX

Get.put(AuthService());

// Lấy lại đối tượng ở một nơi khác trong chương trình

final auth = Get.find<AuthService>();

// Sử dụng đối tượng đã được inject

auth.login();

print(auth.isLoggedIn); // Kết quả: true

**III. Navigation và Routing với GetX :**

Trong Flutter, việc điều hướng (navigation) giữa các màn hình thường được thực hiện bằng Navigator và yêu cầu truyền BuildContext. Tuy nhiên, khi ứng dụng ngày càng phức tạp, cách làm này dễ dẫn đến code rườm rà và khó quản lý.

GetX cung cấp một cơ chế routing và navigation mạnh mẽ, không cần context, giúp điều hướng giữa các màn hình trở nên đơn giản, rõ ràng và dễ kiểm soát hơn.

**1. Cấu hình routes:**

Thay vì sử dụng MaterialApp, GetX sử dụng GetMaterialApp làm widget gốc của ứng dụng.  
Toàn bộ cấu hình các tuyến đường (route) được khai báo trong thuộc tính getPages.

void main() {

  runApp(GetMaterialApp(

    debugShowCheckedModeBanner: false,

    initialRoute: '/',

    getPages: [

      GetPage(name: '/', page: () => HomeScreen()),

      GetPage(name: '/detail', page: () => DetailScreen()),

    ],

  ));

}

* GetMaterialApp thay thế cho MaterialApp, hỗ trợ đầy đủ routing, snackbar, dialog, v.v.
* initialRoute xác định màn hình đầu tiên khi ứng dụng khởi chạy.
* getPages là danh sách các tuyến đường (route) cùng với widget tương ứng.

**2. Các phương thức điều hướng cơ bản**

Các hàm điều hướng trong GetX không cần context và rất dễ gọi từ bất kỳ đâu (bao gồm cả controller):

| **Phương thức** | **Ý nghĩa** | **Dùng khi...** |
| --- | --- | --- |
| Get.to(Widget()) | Điều hướng sang màn hình mới | Push stack |
| Get.back() | Quay lại màn hình trước | Pop |
| Get.off(Widget()) | Thay thế màn hình hiện tại | Không cho quay lại |
| Get.offAll(Widget()) | Xóa toàn bộ stack, mở màn hình mới | Thường dùng sau Login |
| Get.toNamed('/route') | Điều hướng theo tên route | Dùng chung với getPages |
| Get.offNamed('/route') | Thay thế màn hình hiện tại bằng tên route | Không quay lại được |

**3. Truyền dữ liệu giữa các màn hình**

**Truyền dữ liệu bằng arguments**

// Gửi dữ liệu

Get.toNamed('/detail', arguments: {'id': 101, 'name': 'Sản phẩm A'});

// Nhận dữ liệu

final data = Get.arguments;

print(data['name']); // Kết quả: Sản phẩm A

* Dùng để truyền mọi kiểu dữ liệu (Map, object…).
* Cách này dễ dùng cho ứng dụng thực hành.

**4. Quản lý giao diện tạm thời: SnackBar, Dialog, BottomSheet.**

Trong Flutter truyền thống, các thành phần như snackbar, dialog hoặc bottom sheet yêu cầu context, dẫn đến code phức tạp hơn.  
Với GetX, các thành phần này có thể gọi ở bất kỳ đâu. Ví dụ:

// Hiển thị Snackbar

Get.snackbar("Thông báo", "Đã thêm sản phẩm vào giỏ hàng!");

// Hiển thị Dialog

Get.defaultDialog(

  title: "Xác nhận",

  middleText: "Bạn có chắc muốn xóa không?",

  textConfirm: "Đồng ý",

  textCancel: "Hủy",

);

// Hiển thị Bottom Sheet

Get.bottomSheet(

Container(

    color: Colors.white,

    child: Wrap(

      children: [

        ListTile(leading: Icon(Icons.share), title: Text("Chia sẻ")),

        ListTile(leading: Icon(Icons.link), title: Text("Sao chép liên kết")),

      ],

    ),

  ),

);

Điều này giúp tách biệt hoàn toàn phần giao diện và logic, đồng thời tăng tính linh hoạt và khả năng tái sử dụng.

**5. Ví dụ tổng hợp đơn giản**

**Khai báo route:**

GetMaterialApp(

initialRoute: '/',

getPages: [

GetPage(name: '/', page: () => HomePage()),

GetPage(name: '/detail', page: () => DetailPage()),

],

);

**Trong HomePage:**

ElevatedButton(

onPressed: () => Get.toNamed('/detail', arguments: {'title': 'Sản phẩm A'}),

child: Text('Chi tiết'),

);

**Trong DetailPage:**

class DetailPage extends StatelessWidget {

@override

Widget build(BuildContext context) {

final args = Get.arguments;

return Scaffold(

appBar: AppBar(title: Text(args['title'] ?? 'Chi tiết')),

body: Center(child: Text('Nội dung sản phẩm')),

);

}

}

**IV. Performance Benefits và Use Cases của GetX**

**1. Hiệu năng nổi bật (Performance Benefits)**

GetX là một trong những giải pháp quản lý trạng thái và điều hướng có hiệu năng cao nhất trong hệ sinh thái Flutter. Những lợi ích về hiệu năng chính bao gồm:

* **Cập nhật UI theo cơ chế phản ứng chính xác**

Với cơ chế reactive programming sử dụng .obs và Obx, chỉ những widget thực sự phụ thuộc vào biến trạng thái mới được build lại. Điều này giúp tránh việc render lại toàn bộ widget tree khi chỉ một phần nhỏ thay đổi, từ đó cải thiện tốc độ xử lý và giảm tải cho CPU/GPU.

* **Không cần sử dụng setState**

Toàn bộ quá trình cập nhật giao diện có thể thực hiện qua cơ chế .obs và Obx, hoàn toàn không cần gọi setState. Điều này giúp giao diện gọn gàng hơn, tách biệt rõ ràng với logic xử lý.

* **Điều hướng và hiển thị tiện ích không cần context**

GetX cho phép gọi các phương thức như Get.to(), Get.back(), Get.snackbar(), Get.defaultDialog() mà không cần truyền BuildContext. Đây là một điểm mạnh quan trọng giúp các controller không bị ràng buộc với UI, cho phép tổ chức mã nguồn sạch hơn.

* **Tích hợp đa năng, nhẹ và hiệu quả**

GetX tích hợp nhiều tính năng trong một package duy nhất: quản lý trạng thái, điều hướng, dependency injection. Nhờ đó, ứng dụng có thể giảm số lượng thư viện phụ thuộc, giảm kích thước build và tránh xung đột giữa các package.

* **Phù hợp cả với ứng dụng nhỏ lẫn quy mô vừa**

GetX hoạt động rất hiệu quả trong các ứng dụng nhỏ đến trung bình, đặc biệt khi có nhiều thao tác giao diện liên tục. Với quy mô vừa, nếu tổ chức tốt controller và phân chia rõ ràng các màn hình, GetX vẫn duy trì hiệu suất ổn định và tốc độ phản hồi cao.

**2. Trường hợp sử dụng phù hợp (Use Cases)**

* **Ứng dụng nhỏ đến trung bình, cần phát triển nhanh**

Với cú pháp ngắn gọn, GetX rất phù hợp cho các dự án có deadline ngắn, cần ra MVP sớm hoặc chỉ có một đến hai lập trình viên thực hiện. Việc quản lý state và điều hướng nhanh chóng giúp tiết kiệm thời gian và công sức.

* **Ứng dụng có nhiều trạng thái tương tác nhanh**

GetX cho phép cập nhật giao diện mượt mà khi người dùng tương tác liên tục. Đây là lựa chọn phù hợp cho các ứng dụng như: điểm danh, tính điểm, thống kê thời gian thực hoặc các ứng dụng theo dõi dữ liệu nhỏ gọn.

* **Ứng dụng có nhiều màn hình và luồng điều hướng**

Với khả năng điều hướng bằng tên route, truyền dữ liệu, và truy xuất không cần context, GetX cho phép quản lý luồng di chuyển trong ứng dụng một cách hiệu quả và rõ ràng.

* **Ứng dụng yêu cầu tách biệt logic và giao diện**

GetX khuyến khích sử dụng controller để xử lý logic riêng biệt với UI. Điều này giúp mã nguồn dễ bảo trì, dễ mở rộng và có khả năng tái sử dụng cao hơn trong các module khác nhau của ứng dụng.

* **Ứng dụng dùng để demo, thử nghiệm ý tưởng**

GetX giúp lập trình viên tạo ra các nguyên mẫu (prototype) nhanh chóng mà vẫn giữ được cấu trúc rõ ràng. Đây là lựa chọn phù hợp để trình bày ý tưởng hoặc thử nghiệm tính năng với chi phí thời gian thấp.

**V. Demo ứng dụng mua sắm online:**

**1. Code State Mangement with GetX:**

**\* GetX Controller:**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**\* DI and obx:**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**2. Code routing and navigation with GetX**

**\* Routing:**

**A screen shot of a computer code

AI-generated content may be incorrect.A screen shot of a computer code

AI-generated content may be incorrect.**

**\*Navigation:**

**A computer screen shot of a program

AI-generated content may be incorrect.A computer screen shot of text

AI-generated content may be incorrect.**

**A screen shot of a computer code

AI-generated content may be incorrect.A computer screen shot of a computer code

AI-generated content may be incorrect.**

**3. Kết quả app demo:**

Giao diện chính**:**

**Light theme:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Dart theme:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Giao diện sản phẩm :**

**A screen shot of a phone

AI-generated content may be incorrect.**

**Kết quả thanh toán trong giỏ hàng:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Kết quả màn hình cập nhật theo thanh tìm kiếm:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Kết quả khi nhấn vào thêm vào giỏ hàng:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

# **TÀI LIỆU THAM KHẢO**

[1] Flutter Community, “get package,” *pub.dev*, 2025. [Online]. Available: <https://pub.dev/packages/get>  
[2] Flutter Team, *State Management Documentation*, *Flutter.dev*, 2025. [Online]. Available: https://docs.flutter.dev/development/data-and-backend/state-mgmt/simple  
[3] Flutter Community, “Using GetX for navigation and state management,” *Medium*, 2023.  
[4] Reso Coder, “Flutter GetX Tutorial: Simplified State & Navigation,” *YouTube*, 2023.  
[5] K. Pardeshi, “Flutter GetX Guide: Powerful State Management and Routing,” *Mobisoft Infotech Blog*, 2024. [Online]. Available: https://mobisoftinfotech.com/resources/blog/flutter-development/flutter-getx-tutorial  
[6] Flutter Devs, “Mastering Flutter GetX,” *flutterdevs.com*, 2024.  
[7] R. Wane, “Reactive Programming in Flutter with GetX,” *Medium*, 2023.  
[8] Flutter Team, *Navigation and Routing*, *Flutter.dev*, 2025. [Online]. Available: <https://docs.flutter.dev/ui/navigation>