ĐẠI HỌC PHENIKAA
TRƯỜNG CÔNG NGHỆ THÔNG TIN
*****

BÁO CÁO BÀI TẬP LỚN

MÔN HỌC: LẬP TRÌNH CHO THIẾT BỊ DI ĐỘNG

TÊN ĐỀ TÀI: ỨNG DỤNG THEO DÕI THÓI QUEN

Giảng viên hướng dẫn: Nguyễn Xuân Quế

Mã học phần : CSE702027
Lớp: 1-1-25(N05)
Họ và tên sinh viên : Nguyễn Văn Trường **MSV:** 23010371

HÀ NỘI – 2025

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến thầy Nguyễn Xuân Quế, giảng viên môn Lập trình cho thiết bị di động, người đã tận tình giảng dạy, hướng dẫn và hỗ trợ em trong suốt quá trình học tập và thực hiện bài tập lớn này. Nhờ những kiến thức, kinh nghiệm và sự chỉ bảo tận tâm của thầy, em đã hiểu rõ hơn về quy trình phát triển ứng dụng di động, từ khâu thiết kế giao diện đến lập trình và kiểm thử sản phẩm.

Bên cạnh đó, em cũng xin chân thành cảm ơn các thầy cô trong khoa Công nghệ Thông tin đã tạo điều kiện thuận lợi cho em trong quá trình học tập và nghiên cứu.

Mặc dù đã cố gắng hoàn thiện bài tập lớn một cách tốt nhất, nhưng do kiến thức và kinh nghiệm thực tế còn hạn chế, bài làm chắc chắn không tránh khỏi những thiếu sót. Em rất mong nhận được những ý kiến đóng góp quý báu của thầy để em có thể rút kinh nghiệm và hoàn thiện hơn trong các dự án sau này.

Em xin chân thành cảm ơn!

Mục Lục

1. MỞ ĐẦU

1.1. Giới thiệu đề tài

Trong đời sống hiện đại, việc chủ động phát triển và hoàn thiện bản thân là một nhu cầu tất yếu. Nền tảng của quá trình này chính là việc xây dựng và duy trì các thói quen tốt, từ việc chăm sóc sức khỏe thể chất (như uống đủ nước, tập thể dục) đến trau dồi tri thức (như đọc sách, học ngoại ngữ).

Tuy nhiên, thống kê thực tế cho thấy việc duy trì thói quen là một thử thách rất lớn. Rất nhiều người bắt đầu với một động lực cao, nhưng nhanh chóng từ bỏ chỉ sau vài ngày. Một trong những lý do chính dẫn đến thất bại là việc đặt mục tiêu ban đầu quá lớn, quá tham vọng, dẫn đến cảm giác choáng ngợp và áp lực.

Để giải quyết vấn đề này, phương pháp "thói quen nhỏ" (mini habits) đã được chứng minh là mang lại hiệu quả cao. Bằng cách tập trung vào những hành động cực kỳ nhỏ, dễ dàng thực hiện (ví dụ: "chống đẩy 1 cái" thay vì "tập gym 1 tiếng"), người dùng có thể vượt qua sự trì hoãn, dần dần xây dựng động lực và tạo nên sự kiên trì bền bỉ.

Chính vì lý do đó, đề tài **"Xây dựng ứng dụng theo dõi thói quen"** được thực hiện. Mục đích của ứng dụng là cung cấp một công cụ di động đơn giản, thân thiện và tập trung, giúp người dùng bắt đầu và duy trì các thử thách nhỏ. Ứng dụng hoạt động như một người bạn đồng hành, ghi nhận tiến độ và cổ vũ người dùng, từ đó tạo tiền đề cho những thay đổi tích cực và lâu dài.

1.2. Mục tiêu của đề tài

Đề tài này tập trung vào việc hoàn thành các mục tiêu chính sau đây:

- **Về công nghệ:** Tìm hiểu và áp dụng thành thạo ngôn ngữ lập trình **Dart** và framework đa nền tảng **Flutter** để xây dựng một ứng dụng di động hoàn chỉnh, có khả năng chạy mượt mà trên cả hai hệ điều hành Android và iOS từ một cơ sở mã (codebase) duy nhất.
- **Về chức năng:** Triển khai các chức năng cốt lõi cho một ứng dụng theo dõi thói quen, bao gồm:
    - Cho phép người dùng tạo và quản lý hai loại hình thói quen: **Thói quen Hàng ngày** (lặp lại không giới hạn) và **Thử thách** (có số ngày mục tiêu cụ thể).
    - Xây dựng hệ thống theo dõi tiến độ trực quan, cho phép người dùng đánh dấu (tick) hoàn thành theo ngày (bao gồm cả các ngày trong quá khứ và tương lai).
    - Hiển thị các chỉ số tạo động lực quan trọng như **Chuỗi (streak)** ngày hoàn thành liên tiếp.
    - Tự động nhận diện và hiển thị màn hình chúc mừng (với hiệu ứng) khi người dùng hoàn thành một "Thử thách".
    - Cung cấp các màn hình phụ trợ quan trọng như Thống kê, Hồ sơ người dùng, và Cài đặt.
- **Về kỹ thuật lập trình:** Áp dụng các kỹ thuật phát triển ứng dụng hiện đại, đặc biệt là mô hình Quản lý Trạng thái (State Management) bằng gói **Provider**, để đảm bảo luồng dữ liệu trong ứng dụng luôn logic, hiệu quả và dễ dàng bảo trì.

1.3. Phạm vi của đề tài

Để đảm bảo hoàn thành mục tiêu cốt lõi trong thời gian của một bài tập lớn, đề tài tập trung vào các phạm vi sau:

- **Phạm vi chức năng:** Ứng dụng tập trung vào các luồng nghiệp vụ chính đã hoàn thiện, bao gồm: Màn hình Chào mừng, Màn hình chính (Danh sách thói quen), Màn hình Chi tiết (Lịch), Màn hình Thống kê (với các bộ lọc),

Màn hình Hồ sơ (nhập/lưu thông tin), Màn hình Cài đặt (Đa ngôn ngữ Anh/Việt, Chế độ Sáng/Tối) và Màn hình Thông tin.

- **Giới hạn kỹ thuật:**
  - **Lưu trữ dữ liệu:** Đề tài tập trung chính vào trải nghiệm người dùng (UI/UX) và kiến trúc quản lý trạng thái. Do đó, toàn bộ dữ liệu của ứng dụng (danh sách thói quen, hồ sơ, cài đặt) được lưu trữ tạm thời **trong bộ nhớ (in-memory state)** khi ứng dụng chạy, thông qua các Provider.
  - **Hệ quả:** Dữ liệu sẽ bị mất khi người dùng tắt hoàn toàn ứng dụng (khởi động lại từ đầu).
  - **Các tính năng ngoài phạm vi:** Các tính năng nâng cao như lưu trữ dữ liệu vĩnh viễn (sử dụng SQLite, Firebase hoặc shared_preferences) và gửi **thông báo đẩy (push notification)** (mặc dù đã có giao diện chọn giờ) được xác định là hướng phát triển trong tương lai và nằm ngoài phạm vi của bài tập này.

2.PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

Chương này trình bày chi tiết về quá trình phân tích các yêu cầu nghiệp vụ của ứng dụng, các công nghệ được lựa chọn để triển khai, và thiết kế kiến trúc phần mềm được áp dụng để xây dựng dự án.

2.1. Phân tích Yêu cầu

2.1.1. Yêu cầu chức năng (Functional Requirements)

Ứng dụng "Mini Habit Challenge" cần đáp ứng các yêu cầu chức năng sau:

- **Quản lý Thói quen:**
  - **Tạo Thói quen:** Người dùng có thể tạo một thói quen mới.
  - **Phân loại Thói quen:** Hệ thống phân biệt 2 loại thói quen:

        - *Hàng ngày (Daily):* Thói quen lặp lại vô thời hạn (ví dụ: Uống nước).
        - *Thử thách (Challenge):* Thói quen có mục tiêu số ngày cụ thể (ví dụ: 7 ngày đọc sách).
    - **Chi tiết Thói quen:** Khi tạo, người dùng có thể nhập Tên, chọn Loại, đặt Số ngày (nếu là Thử thách), và chọn Giờ nhắc nhở (tùy chọn).
- **Theo dõi Tiến độ:**
    - **Đánh dấu Hàng ngày:** Người dùng có thể đánh dấu (tick/un-tick) hoàn thành thói quen cho ngày hôm nay trực tiếp từ màn hình chính.
    - **Đánh dấu trong Lịch:** Trong màn hình chi tiết, người dùng có thể xem lịch và đánh dấu hoàn thành cho các ngày trong quá khứ hoặc tương lai.
    - **Tính toán Chuỗi (Streak):** Hệ thống phải tự động tính toán và hiển thị số ngày hoàn thành liên tiếp (chuỗi) cho mỗi thói quen.
- **Hoàn thành và Thống kê:**
    - **Phản hồi Hoàn thành:** Khi một "Thử thách" hoàn thành đủ số ngày, ứng dụng hiển thị màn hình chúc mừng (với hiệu ứng pháo hoa) và di chuyển thói quen đó ra khỏi danh sách chính.
    - **Xem Thống kê:** Người dùng có thể truy cập màn hình Thống kê để xem:
        - *Bộ lọc:* Lọc tiến độ theo "Tổng quan", "Hàng ngày", hoặc "Thử thách".
        - *Tiến độ Tổng:* Hiển thị tỷ lệ % hoàn thành của tất cả thói quen trong ngày hôm nay (ở tab "Tổng quan").
        - *Danh sách Hoàn thành:* Hiển thị danh sách các "Thử thách" đã hoàn thành.
- **Quản lý Người dùng và Cài đặt:**
    - **Hồ sơ (Profile):** Người dùng có thể nhập và lưu (tạm thời) các thông tin cá nhân như Tên, Ngày sinh, Cân nặng, Chiều cao.
    - **Cài đặt (Settings):**
        - *Đa ngôn ngữ:* Người dùng có thể chuyển đổi ngôn ngữ hiển thị giữa Tiếng Việt và Tiếng Anh.

        - *Chế độ Giao diện:* Người dùng có thể chọn chế độ Sáng (Light), Tối (Dark), hoặc theo Hệ thống (System).
    - **Reset Ứng dụng:** Người dùng có thể xóa toàn bộ dữ liệu (thói quen, hồ sơ, cài đặt) để đưa ứng dụng về trạng thái ban đầu.
- **Giao diện và Điều hướng:**
    - **Màn hình Chào mừng (Welcome):** Hiển thị màn hình giới thiệu khi khởi động.
    - **Điều hướng chính (Bottom Navigation):** Chuyển đổi giữa 3 màn hình chính: Thử thách, Thống kê, Hồ sơ.
    - **Điều hướng phụ (Drawer):** Cung cấp lối tắt đến các màn hình Cài đặt, Thông tin (About), và chức năng Reset.
    - **Banner Động lực:** Hiển thị ảnh banner ở màn hình Thử thách và Hồ sơ để tăng tính thẩm mỹ và tạo động lực.

2.1.2. Yêu cầu phi chức năng (Non-functional Requirements)

- **Đa nền tảng:** Ứng dụng phải chạy được trên cả hai hệ điều hành Android và iOS từ một cơ sở mã nguồn duy nhất.
- **Hiệu suất:** Ứng dụng phải khởi động nhanh, các thao tác (cuộn, tick, chuyển trang) phải mượt mà, không bị giật, lag.
- **Giao diện người dùng (UI):** Giao diện phải hiện đại (theo chuẩn Material 3), sạch sẽ, trực quan, và có tính thẩm mỹ cao (sử dụng màu sắc, font chữ, và khoảng trắng hợp lý).
- **Dễ bảo trì:** Cấu trúc code phải được tổ chức rõ ràng, tách biệt giữa logic nghiệp vụ (business logic) và giao diện (UI) để dễ dàng sửa lỗi và nâng cấp.
- **Trải nghiệm người dùng (UX):** Ứng dụng phải cung cấp phản hồi ngay lập tức cho các hành động của người dùng (ví dụ: tick vào thói quen, lưu hồ sơ).

2.2. Công nghệ sử dụng

Để đáp ứng các yêu cầu trên, các công nghệ và thư viện sau đã được lựa chọn:

- **Ngôn ngữ lập trình: Dart (phiên bản 3.x)**
  - Ngôn ngữ lập trình hiện đại, tối ưu hóa cho UI, được phát triển bởi Google.
- **Framework: Flutter (phiên bản 3.x)**
  - Framework phát triển ứng dụng di động đa nền tảng, cho phép xây dựng giao diện đẹp, mượt mà và biên dịch trực tiếp ra mã máy (native code).
- **Quản lý Trạng thái: provider**
  - Đây là một trong những giải pháp Quản lý Trạng thái (State Management) được Google chính thức khuyến nghị.
  - *Lý do chọn:* provider sử dụng cơ chế ChangeNotifier đơn giản, dễ học, hiệu suất cao và giúp tách biệt hoàn toàn logic nghiệp vụ (ví dụ: HabitProvider) ra khỏi các tệp giao diện (các tệp trong screens/).
- **Các gói (Packages) hỗ trợ chính:**
  - **google_fonts**: Dễ dàng tích hợp và sử dụng các font chữ tùy chỉnh (như Poppins) từ thư viện Google Fonts để đồng bộ giao diện.
  - **flutter_localizations và intl**: Bộ công cụ chuẩn của Flutter để triển khai đa ngôn ngữ. Chúng tôi sử dụng các tệp .arb (Application Resource Bundle) để quản lý chuỗi văn bản cho Tiếng Việt (app_vi.arb) và Tiếng Anh (app_en.arb).
  - **percent_indicator**: Thư viện phổ biến để vẽ các thanh tiến độ tuyến tính (Linear) và vòng tròn (Circular) được sử dụng trong Màn hình Thống kê.
  - **table_calendar**: Thư viện mạnh mẽ để tạo giao diện Lịch, được sử dụng trong Màn hình Chi tiết Thói quen để người dùng có thể xem và tương tác với các ngày trong quá khứ/tương lai.
  - **confetti**: Thư viện tạo hiệu ứng pháo hoa, được sử dụng trong Màn hình Chúc mừng để tăng trải nghiệm tích cực cho người dùng.
  - **uuid**: Thư viện tiện ích để tạo ra các Chuỗi ID duy nhất (ví dụ: id cho mỗi Habit), đảm bảo không có sự trùng lặp khi thêm, xóa, hoặc cập nhật thói quen.

2.3. Thiết kế Cấu trúc Dữ liệu (Data Model)

Mô hình dữ liệu cốt lõi của ứng dụng xoay quanh đối tượng Habit.

Lớp Habit (trong lib/models/habit.dart):

- id (String): ID định danh duy nhất (tạo bởi uuid).
- name (String): Tên thói quen (do người dùng nhập).
- type (HabitType): Một enum (kiểu liệt kê) xác định là HabitType.daily (Hàng ngày) hay HabitType.challenge (Thử thách).
- startDate (DateTime): Ngày bắt đầu của thói quen (thường là ngày tạo).
- totalDays (int?): Số ngày mục tiêu (có thể null nếu là HabitType.daily).
- reminderTime (TimeOfDay?): Giờ nhắc nhở trong ngày (có thể null nếu không đặt).
- completionLog (Map<DateTime, bool>): **Đây là cấu trúc dữ liệu quan trọng nhất.**
    - *Lý do chọn Map thay vì List*: Một List<bool> đơn giản (như [true, false, true]) không thể lưu trữ thông tin ngày tháng. Bằng cách dùng Map, chúng ta có thể ánh xạ một **ngày cụ thể** (chỉ lấy YYYY-MM-DD, bỏ qua giờ) với trạng thái hoàn thành của nó (ví dụ: [28/10/2025: true, 29/10/2025: false]). Cấu trúc này cho phép:
        1. Tính toán Chuỗi (streak) một cách chính xác.
        2. Cho phép người dùng tick/un-tick vào bất kỳ ngày nào trên lịch (TableCalendar).

2.4. Thiết kế Kiến trúc (Architecture)

Ứng dụng sử dụng kiến trúc **Quản lý Trạng thái Tập trung (Centralized State Management)** thông qua Provider.

- **Nguyên tắc:** Toàn bộ trạng thái (dữ liệu) và logic nghiệp vụ của ứng dụng được đặt trong các lớp ChangeNotifier (gọi là "Provider"). Giao diện (UI)

chỉ có nhiệm vụ "đọc" (read) dữ liệu từ Provider để hiển thị, và "gọi" (call) các hàm trong Provider khi người dùng tương tác.

- **Các Provider chính:**
    1. **HabitProvider:** "Bộ não" chính, quản lý _habits và _completedHabits. Cung cấp các hàm như addHabit(), deleteHabit(), toggleDateCompletion(), và logic nghiệp vụ (như _handleHabitCompletionCheck để xử lý khi hoàn thành).
    2. **SettingsProvider:** Quản lý trạng thái cài đặt (themeMode và locale).
    3. **ProfileProvider:** Quản lý trạng thái hồ sơ người dùng (tên, ngày sinh, v.v.).
- **Luồng hoạt động (Data Flow):**
    1. **Khởi tạo:** main.dart khởi tạo MultiProvider ở cấp cao nhất, "bọc" toàn bộ ứng dụng (MyApp).
    2. **Lắng nghe:**
        - MaterialApp (trong main.dart) "xem" (watch) SettingsProvider để tự động cập nhật Theme và Ngôn ngữ.
        - Các màn hình (ví dụ: ChallengeListScreen) dùng Consumer hoặc context.watch<HabitProvider>() để "xem" danh sách thói quen và tự động vẽ lại (rebuild) khi có thay đổi (ví dụ: khi thêm 1 thói quen mới).
    3. **Hành động:**
        - Khi người dùng nhấn nút "Lưu" (trong ProfileScreen), nút này sẽ gọi context.read<ProfileProvider>().updateProfile(...).
        - Provider cập nhật dữ liệu của nó và gọi notifyListeners().
        - Tất cả các widget đang "xem" provider đó sẽ tự động cập nhật theo dữ liệu mới.

Kiến trúc này đảm bảo code được tách biệt rõ ràng (Separation of Concerns), dễ đọc, dễ kiểm thử và dễ mở rộng.

2.5. Thiết kế Cấu trúc Thư mục

Dự án được tổ chức theo cấu trúc thư mục tiêu chuẩn của Flutter, giúp phân tách rõ ràng các thành phần:

```
lib/
├── l10n/                    # Chứa các tệp .arb cho đa ngôn ngữ
│   ├── app_en.arb
│   └── app_vi.arb
├── models/
│   └── habit.dart       # Định nghĩa Lớp (Class) dữ liệu
├── providers/
│   ├── habit_provider.dart
│   ├── profile_provider.dart
│   └── settings_provider.dart
├── screens/
│   ├── about_screen.dart
│    ├── challenge_list_screen.dart # Màn hình chính
│   ├── completion_screen.dart
│   ├── habit_detail_screen.dart
│    ├── main_screen.dart        # Khung sườn (BottomNav, Drawer)
│   ├── profile_screen.dart
│   ├── settings_screen.dart
│   ├── statistics_screen.dart
│   └── welcome_screen.dart
├── widgets/
```

```
|    └── add_habit_dialog.dart # Các widget con
tái sử dụng
└── main.dart            # Điểm bắt đầu của ứng
dụng
```

3.TRIỂN KHAI CÁC CHỨC NĂNG CHÍNH

Chương này trình bày chi tiết các giải pháp kỹ thuật và đoạn mã (code snippet) tiêu biểu được sử dụng để triển khai các chức năng cốt lõi của ứng dụng "Theo Dõi Thói Quen".

3.1. Kiến trúc Quản lý Trạng thái với Provider

Để tách biệt logic nghiệp vụ (business logic) ra khỏi giao diện người dùng (UI), dự án đã áp dụng mô hình Quản lý Trạng thái Tập trung (Centralized State Management) bằng gói provider.

1. Cung cấp Trạng thái (Providing State): Tất cả các "bộ não" của ứng dụng được khởi tạo ở cấp cao nhất thông qua MultiProvider trong tệp main.dart. Điều này đảm bảo mọi màn hình bên dưới đều có thể truy cập và chia sẻ cùng một nguồn dữ liệu.

Đoạn mã khởi tạo MultiProvider trong main.dart :

```
// lib/main.dart
void main() {
  runApp(
    MultiProvider(
      providers: [
```

```
        // Quản lý nghiệp vụ Thói quen
        ChangeNotifierProvider(create:
(context) => HabitProvider()),
        // Quản lý Cài đặt (Theme, Ngôn ngữ)
        ChangeNotifierProvider(create:
(context) => SettingsProvider()),
        // Quản lý Hồ sơ người dùng
        ChangeNotifierProvider(create:
(context) => ProfileProvider()),
      ],
      child: const MyApp(),
    ),
  );
}
```

2. Mô hình ChangeNotifier: Mỗi provider là một lớp (class) kế thừa (extends) ChangeNotifier. Khi dữ liệu bên trong provider thay đổi (ví dụ: người dùng thêm một thói quen), hàm notifyListeners() sẽ được gọi.

Ví dụ về addHabit trong HabitProvider :

```
// lib/providers/habit_provider.dart
class HabitProvider with ChangeNotifier {
  List<Habit> _habits = [];

  void addHabit(...) {
    final newHabit = Habit(...);
    _habits.add(newHabit);
```

```
    // Thông báo cho tất cả các widget đang "nghe" để cập nhật
    notifyListeners();
  }
}
```

3. Lắng nghe Trạng thái (Listening to State): Giao diện (UI) sử dụng Consumer hoặc context.watch<T>() để "lắng nghe" các thay đổi. Khi notifyListeners() được gọi, chỉ những widget đang lắng nghe này mới được xây dựng lại (rebuild), giúp tối ưu hiệu suất.

Consumer trong challenge_list_screen.dart :

```
// lib/screens/challenge_list_screen.dart
@override
Widget build(BuildContext context) {
  // Consumer sẽ tự động rebuild khi HabitProvider thay đổi
  return Consumer<HabitProvider>(
    builder: (context, provider, child) {
      final dailyHabits = provider.dailyHabits;
      // ...
      return SingleChildScrollView(...);
    },
  );
}
```

4. Gọi Hành động (Calling Actions): Khi người dùng thực hiện một hành động (ví dụ: nhấn nút), UI sẽ gọi hàm từ provider bằng context.read<T>() (hoặc Provider.of<T>(listen: false)). Việc này chỉ gọi hàm mà không đăng ký lắng nghe, tránh việc rebuild không cần thiết.

Gọi hành động trong profile_screen.dart:

```
// lib/screens/profile_screen.dart
void _saveProfile() {
  // Chỉ gọi hàm, không "nghe"
  final profile =
context.read<ProfileProvider>();
  profile.updateProfile(name:
_nameController.text, ...);
}
```

3.2. Triển khai Logic Nghiệp vụ Cốt lõi

Các logic phức tạp nhất của ứng dụng nằm trong HabitProvider và model Habit.

1. Logic Tính toán Chuỗi (Streak): Để tính chuỗi ngày liên tiếp, một hàm getter (hàm lấy) tên streak đã được viết trong lib/models/habit.dart.

- **Thuật toán:**
    1. Khởi tạo currentStreak = 0.
    2. Lấy ngày hôm nay (today).
    3. Kiểm tra: Nếu completionLog[today] không phải true (tức là hôm nay chưa làm), thuật toán sẽ bắt đầu đếm lùi từ yesterday (hôm qua). Nếu today là true, nó bắt đầu đếm từ today.
    4. Sử dụng vòng lặp while, thuật toán kiểm tra completionLog[dateToCheck] == true.

5. Nếu đúng, tăng currentStreak và lùi dateToCheck về 1 ngày (subtract(Duration(days: 1))).
6. Vòng lặp dừng lại khi completionLog trả về false hoặc null (không có dữ liệu).

2. Logic Xử lý Hoàn thành Thử thách (Challenge Completion): Đây là một quy trình tự động được kích hoạt mỗi khi người dùng tick vào một thói quen.

- **Kích hoạt (Trigger):** Cả hai hàm toggleTodayCompletion() và toggleDateCompletion() trong HabitProvider đều gọi một hàm private là _handleHabitCompletionCheck(habit).
- **Kiểm tra (Check):** Hàm này kiểm tra thuộc tính habit.isChallengeFullyCompleted. Thuộc tính này (trong habit.dart) sẽ đếm số lượng true trong completionLog và so sánh với totalDays.
- **Hành động (Action):** Nếu isChallengeFullyCompleted trả về true:
  1. Thói quen được xóa khỏi danh sách chính: _habits.remove(habit).
  2. Thói quen được thêm vào danh sách hoàn thành: _completedHabits.add(habit).
  3. Một "cờ sự kiện" (event flag) được đặt: _justCompletedHabit = habit.
  4. Hàm notifyListeners() được gọi.

3. Cơ chế Cờ sự kiện (Event Flag) cho Màn hình Chúc mừng: Để tránh lỗi (như lỗi "Null check" đã gặp), việc hiển thị màn hình chúc mừng được xử lý bằng cờ _justCompletedHabit:

- Cả ChallengeListScreen và HabitDetailScreen đều "watch" (xem) provider.justCompletedHabit.
- Khi cờ này *không* null, UI sẽ sử dụng WidgetsBinding.instance.addPostFrameCallback(...) để lên lịch điều hướng *sau khi* quá trình build hiện tại hoàn tất.
- Nó điều hướng (Navigator.push) đến CompletionScreen và truyền habit.name vào.
- Ngay sau khi điều hướng, nó gọi provider.clearJustCompletedHabit() để reset cờ về null, đảm bảo màn hình chúc mừng chỉ hiện 1 lần.

3.3. Triển khai Giao diện Tương tác Phức tạp

1. Tích hợp Lịch (TableCalendar) trong Màn hình Chi tiết: Để cho phép người dùng tương tác với lịch sử, thư viện table_calendar đã được tích hợp trong habit_detail_screen.dart.

- **Hiển thị Dữ liệu:** Thuộc tính quan trọng nhất là eventLoader. Nó nhận một ngày (DateTime) làm đầu vào. Bên trong hàm này, chúng ta truy cập habit.completionLog[_dateOnly(day)]. Nếu là true, hàm sẽ trả về một danh sách ["completed"], và TableCalendar sẽ tự động vẽ một dấu chấm (marker) bên dưới ngày đó.
- **Xử lý Tương tác:** Thuộc tính onDaySelected được sử dụng. Khi người dùng nhấn vào một ngày, hàm này sẽ được gọi. Bên trong, chúng ta gọi provider.toggleDateCompletion(habit.id, selectedDay) để cập nhật trạng thái, Provider sẽ notifyListeners(), và Lịch sẽ tự động vẽ lại (rebuild) với dữ liệu mới.

2. Điều hướng chính (MainScreen) và IndexedStack: Ứng dụng sử dụng một cấu trúc Scaffold lồng nhau. MainScreen đóng vai trò là "vỏ" (shell) chính.

- MainScreen chứa Scaffold, Drawer, và BottomNavigationBar.
- body của MainScreen là một IndexedStack.
- *Lý do chọn IndexedStack*: Thay vì xây dựng lại (rebuild) các màn hình (Challenge, Statistics, Profile) mỗi khi chuyển tab, IndexedStack giữ cho cả 3 màn hình **luôn tồn tại trong bộ nhớ (keep alive)**. Điều này giúp tăng hiệu suất và giữ nguyên trạng thái cuộn (scroll position) khi người dùng chuyển tab qua lại.
- AppBar cũng được quản lý động trong MainScreen, hiển thị tiêu đề và nút bấm tương ứng với _selectedIndex (tab đang được chọn).

3. Đa ngôn ngữ và Chế độ Sáng/Tối (Localization & Theming): Đây là một ví dụ điển hình của việc Provider điều khiển MaterialApp.

- SettingsScreen (UI) gọi settingsProvider.updateLocale(Locale('en')) hoặc settingsProvider.updateThemeMode(ThemeMode.dark).
- SettingsProvider (Logic) thay đổi biến của nó và gọi notifyListeners().
- MaterialApp (trong main.dart) đang "xem" (consume) SettingsProvider.

- Khi có thông báo, MaterialApp rebuild lại với giá trị mới: locale: settingsProvider.locale và themeMode: settingsProvider.themeMode.
- Kết quả là toàn bộ cây widget bên dưới MaterialApp ngay lập tức thay đổi ngôn ngữ và chủ đề mà không cần khởi động lại.

4.KẾT QUẢ VÀ GIAO DIỆN ỨNG DỤNG

Chương này trình bày kết quả trực quan của dự án "Mini Habit Challenge". Các giao diện được xây dựng bằng framework Flutter, áp dụng triệt để bộ thiết kế **Material 3 (M3)** và chủ đề màu sắc tùy chỉnh (dựa trên màu **Vàng Hổ Phách - Amber**) để mang lại trải nghiệm người dùng đồng nhất, năng động và hiện đại.
Tất cả các màn hình đều hỗ trợ đầy đủ Chế độ Sáng (Light Mode), Chế độ Tối (Dark Mode) và song ngữ (Tiếng Việt/Tiếng Anh) như đã thiết kế.

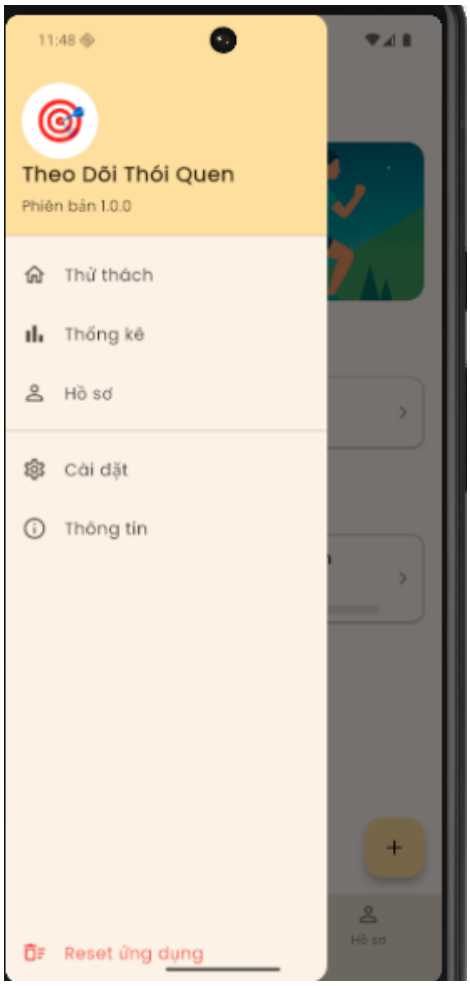
4.1. Màn hình Chào mừng (Welcome Screen)

Đây là màn hình đầu tiên người dùng nhìn thấy khi khởi động ứng dụng. Màn hình được thiết kế với bố cục hiện đại, sử dụng Spacer để đẩy nội dung chính (biểu tượng và văn bản giới thiệu) lên trên và nút "Bắt đầu ngay" xuống dưới cùng, đảm bảo tương thích với nhiều kích cỡ màn hình.

4.2. Giao diện và Điều hướng chính

Sau khi qua màn hình Chào mừng, người dùng sẽ vào MainScreen, đây là "khung sườn" chính của ứng dụng.

- **Thanh điều hướng dưới (Bottom Navigation Bar):** Cung cấp 3 tab chính: "Thử thách" (trang chủ), "Thống kê", và "Hồ sơ". Thanh điều hướng được tùy chỉnh màu sắc (primaryContainer) để đồng bộ với chủ đề.
- **Menu trượt (Navigation Drawer):** Được truy cập từ icon (☰) ở Màn hình chính. Drawer chứa các điều hướng phụ quan trọng như "Cài đặt", "Thông tin" và chức năng "Reset ứng dụng". UserAccountsDrawerHeader được tùy chỉnh màu sắc và font chữ để nổi bật trên nền.

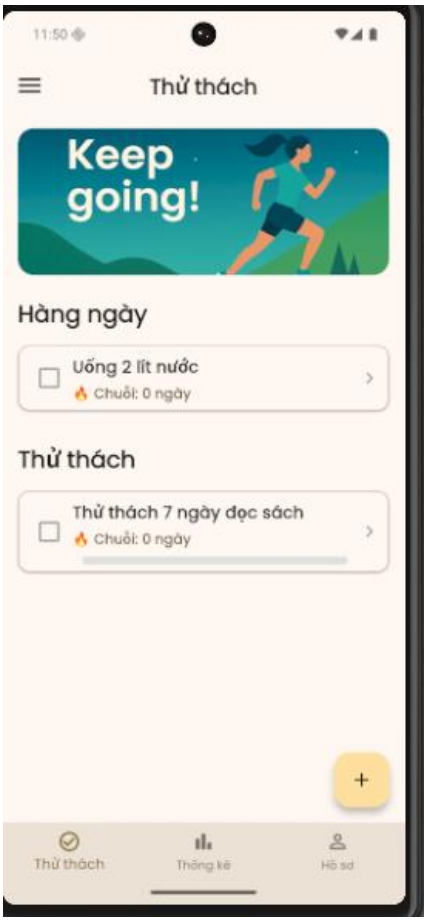


4.3. Màn hình chính: Thử thách (Challenge Screen)

Đây là màn hình cốt lõi của ứng dụng, nơi người dùng tương tác hàng ngày. Bố cục SingleChildScrollView chứa các thành phần:

1. **Banner Động lực:** Một tấm ảnh Image.asset với BoxFit.cover được đặt trên cùng để tạo cảm hứng.
2. **Phân loại Danh sách:** Giao diện được chia rõ ràng thành 2 mục "Hàng ngày" và "Thử thách" bằng các tiêu đề Text in đậm.
3. **Thẻ Thói quen (Habit Card):**
    - Mỗi thói quen được hiển thị dưới dạng một Card tùy chỉnh, sử dụng màu primaryContainer và đường viền (Border) tinh tế (thay vì dùng elevation - bóng mờ) để tạo giao diện phẳng, sạch sẽ.
    - Nút IconButton (Checkbox) bên trái cho phép người dùng tick/un-tick nhanh (gọi provider.toggleTodayCompletion).

- Hiển thị thông tin tên thói quen và "Chuỗi" (streak) với icon 🔥.



Khi người dùng nhấn nút FloatingActionButton (nút ➕), một hộp thoại (AlertDialog) sẽ xuất hiện. Hộp thoại này sử dụng StatefulWidget (AddHabitDialog) để quản lý các lựa chọn của người dùng (Loại, Giờ) một cách linh hoạt.

4.4. Màn hình Chi tiết Thói quen (Habit Detail)

Khi người dùng nhấn vào một Thẻ Thói quen, họ sẽ được điều hướng đến Màn hình Chi tiết.

- **Thẻ Thống kê:** Hiển thị thông tin chi tiết: Chuỗi hiện tại, Ngày bắt đầu, và Giờ nhắc nhở.
- **Lịch tương tác (TableCalendar):**
    - Triển khai gói table_calendar để hiển thị lịch.

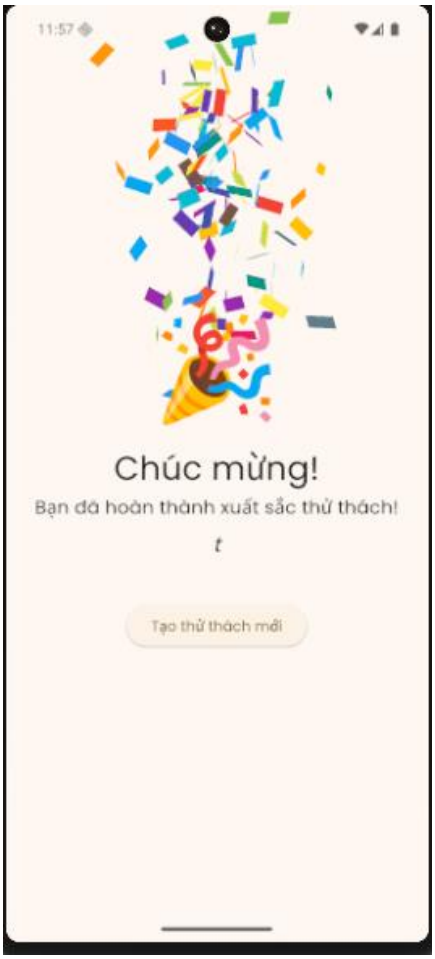
- Sử dụng eventLoader để đọc dữ liệu từ habit.completionLog và vẽ các dấu chấm (marker) bên dưới những ngày đã hoàn thành.
- Người dùng có thể nhấn vào bất kỳ ngày nào (quá khứ hoặc tương lai) để gọi hàm provider.toggleDateCompletion.

- **Nút chức năng:** Màn hình có một nút FloatingActionButton "Tick Hôm nay" và một IconButton "Xóa" trên AppBar (kèm Dialog xác nhận).

4.5. Màn hình Chúc mừng (Completion Screen)

Khi người dùng hoàn thành một "Thử thách" (không phải thói quen "Hàng ngày"), cờ _justCompletedHabit trong HabitProvider sẽ được kích hoạt. Giao diện (cả Màn hình chính và Màn hình Chi tiết) sẽ lắng nghe cờ này và Navigator.push đến Màn hình Chúc mừng.

Màn hình này sử dụng gói confetti để tạo hiệu ứng pháo hoa, mang lại phản hồi tích cực cho người dùng.



4.6. Màn hình Thống kê (Statistics Screen)

Màn hình này được thiết kế để cung cấp cái nhìn tổng quan về tiến độ của người dùng. Nó sử dụng DefaultTabController để tạo 3 tab lọc:

1. **Tab "Tổng quan":** Hiển thị một vòng tròn CircularPercentIndicator (từ gói percent_indicator) thể hiện tỷ lệ % hoàn thành trong ngày (lấy từ provider.overallTodayProgress).
2. **Tab "Hàng ngày" & "Thử thách":** Liệt kê tiến độ của từng thói quen đang hoạt động.
3. **Danh sách Hoàn thành:** Ở cuối tab "Thử thách", ứng dụng hiển thị danh sách các thói quen đã được chuyển từ _habits sang _completedHabits.

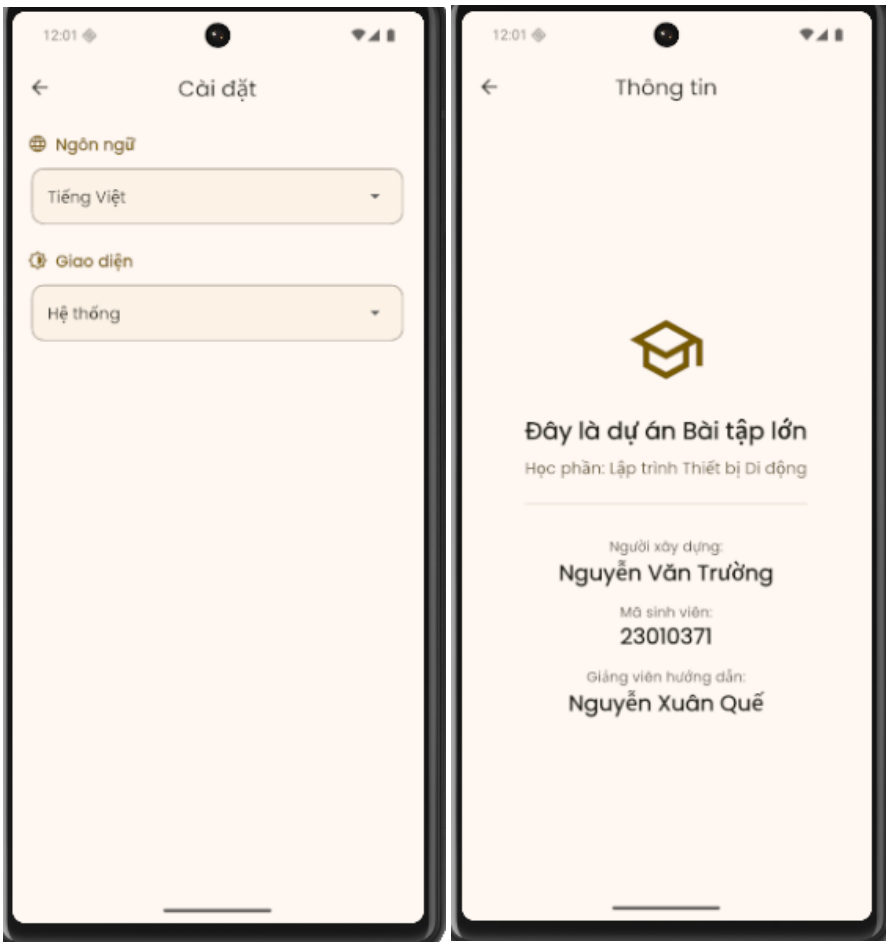
4.7. Màn hình Hồ sơ (Profile Screen)

Màn hình này cho phép người dùng nhập thông tin cá nhân. Nó được triển khai bằng StatefulWidget để quản lý các TextEditingController.

- Dữ liệu được tải từ ProfileProvider vào Controller khi màn hình khởi chạy (didChangeDependencies).
- Khi nhấn "Lưu thay đổi", dữ liệu từ Controller được gửi ngược lại ProfileProvider (gọi provider.updateProfile).
- Giao diện sử dụng các InputDecorator và _buildTextField helper để tạo biểu mẫu nhất quán, sạch sẽ.

4.8. Màn hình Cài đặt và Thông tin (Settings & About)

- **Màn hình Cài đặt:** Cung cấp 2 DropdownButton (nút thả xuống). Khi người dùng thay đổi lựa chọn (ví dụ: chọn "Tiếng Anh"), hàm settingsProvider.updateLocale được gọi. MaterialApp ở gốc sẽ tự động rebuild lại toàn bộ ứng dụng với ngôn ngữ mới.
- **Màn hình Thông tin:** Màn hình Scaffold đơn giản, sử dụng Center và Column để hiển thị thông tin chi tiết của bài tập lớn một cách trang trọng.



5.KẾT LUẬN VÀ ĐÁNH GIÁ

5.1. Kết quả đạt được

Qua quá trình thực hiện bài tập lớn, đề tài "Xây dựng ứng dụng theo dõi thói quen" đã hoàn thành được các mục tiêu chính đề ra:

1. **Xây dựng thành công ứng dụng đa nền tảng:** Ứng dụng đã được xây dựng hoàn chỉnh bằng ngôn ngữ Dart và framework Flutter, có khả năng biên dịch và chạy mượt mà trên cả hai hệ điều hành Android và iOS từ một cơ sở mã nguồn duy nhất.
2. **Hoàn thiện các chức năng cốt lõi:** Đã triển khai thành công toàn bộ các chức năng nghiệp vụ trọng tâm, bao gồm:
    - Hệ thống tạo và phân loại thói quen (Hàng ngày / Thử thách).

- Logic theo dõi tiến độ phức tạp, bao gồm việc tính toán **Chuỗi (streak)** và đánh dấu (tick) linh hoạt qua Lịch (TableCalendar).
- Quy trình tự động xử lý khi hoàn thành một Thử thách (hiển thị pháo hoa, di chuyển dữ liệu sang mục Thống kê).

3. **Áp dụng kiến trúc phần mềm hiện đại:** Đã áp dụng thành công mô hình Quản lý Trạng thái (State Management) bằng **Provider**. Kiến trúc này giúp tách biệt rõ ràng logic (trong các Provider) và giao diện (trong các Screen), giúp code sạch sẽ, dễ bảo trì và mở rộng.
4. **Hoàn thiện trải nghiệm người dùng (UI/UX):** Ứng dụng có giao diện hiện đại theo chuẩn **Material 3**, sử dụng chủ đề màu sắc tùy chỉnh (Vàng Hổ Phách) nhất quán, hỗ trợ đầy đủ Chế độ Sáng/Tối và song ngữ (Tiếng Việt/Tiếng Anh), mang lại trải nghiệm thân thiện và chuyên nghiệp.

5.2. Hạn chế của đề tài

Bên cạnh những kết quả đạt được, trong khuôn khổ của một bài tập lớn, dự án vẫn còn tồn tại một số hạn chế kỹ thuật quan trọng cần được nhìn nhận rõ:

1. **Lưu trữ dữ liệu tạm thời:** Hạn chế lớn nhất là toàn bộ dữ liệu của ứng dụng (danh sách thói quen, hồ sơ, cài đặt) đều được lưu trữ tạm thời trong bộ nhớ (in-memory state) thông qua các Provider. Điều này đồng nghĩa với việc **toàn bộ dữ liệu sẽ bị mất** khi người dùng tắt hoàn toàn ứng dụng (thoát khỏi đa nhiệm).
2. **Thiếu tính năng Thông báo:** Mặc dù giao diện (UI) cho phép người dùng chọn "Giờ nhắc nhở" khi tạo thói quen, nhưng logic nghiệp vụ để lập lịch và gửi thông báo đẩy (push notification) cục bộ vẫn **chưa được triển khai**. Đây là một thiếu sót lớn đối với một ứng dụng thói quen.
3. **Thiếu đồng bộ hóa:** Ứng dụng hoạt động hoàn toàn ngoại tuyến (offline). Dữ liệu không được đồng bộ lên đám mây (cloud), khiến người dùng không thể khôi phục dữ liệu hoặc sử dụng chung tài khoản trên nhiều thiết bị.

5.3. Hướng phát triển trong tương lai

Từ những hạn chế đã nêu, đề tài mở ra nhiều hướng phát triển và nâng cấp giá trị trong tương lai:

1. **Triển khai Lưu trữ Vĩnh viễn (Persistence):** Đây là ưu tiên hàng đầu.
    - Sử dụng gói **shared_preferences** để lưu trữ các dữ liệu đơn giản, ít thay đổi như Cài đặt (Theme, Ngôn ngữ) và Hồ sơ người dùng (Tên, Cân nặng).
    - Tích hợp cơ sở dữ liệu **sqflite** (một CSDL SQL cục bộ) để lưu trữ danh sách các thói quen (_habits, _completedHabits) một cách bền bỉ.
2. **Hoàn thiện chức năng Nhắc nhở:** Tích hợp gói **flutter_local_notifications** để lập lịch (schedule) các thông báo lặp lại hàng ngày dựa trên reminderTime mà người dùng đã chọn khi tạo thói quen.
3. **Nâng cấp Trải nghiệm:**
    - Cho phép người dùng tùy chỉnh sâu hơn (ví dụ: chọn Icon, Màu sắc riêng cho từng thói quen).
    - Xây dựng màn hình Thống kê chi tiết hơn, sử dụng biểu đồ (ví dụ: gói fl_chart) để trực quan hóa tiến độ theo thời gian.
4. **Đồng bộ đám mây (Cloud Sync):** Tích hợp với **Firebase (Firestore)** để lưu trữ dữ liệu người dùng trực tuyến, cho phép đồng bộ hóa và khôi phục dữ liệu trên nhiều thiết bị.

5.4. Bài học kinh nghiệm

Qua quá trình thực hiện dự án từ con số 0 đến khi hoàn thiện, bản thân đã rút ra được nhiều bài học và kinh nghiệm thực tiễn quý báu:

- **Hiểu sâu về Flutter:** Nắm vững cách xây dựng bố cục (layout) phức tạp, điều hướng (Navigator), và cách Flutter xử lý vòng đời của StatefulWidget.
- **Tầm quan trọng của Quản lý Trạng thái:** Hiểu rõ tại sao không nên để logic nghiệp vụ lẫn lộn với UI. Việc sử dụng Provider (với ChangeNotifier,

Consumer, context.read/watch) là một kỹ năng then chốt để xây dựng các ứng dụng lớn, có tổ chức.

- **Tư duy Hệ thống:** Học được cách thiết kế kiến trúc phần mềm trước khi viết code, từ việc định nghĩa Model (cấu trúc dữ liệu Map<DateTime, bool>), phân tách trách nhiệm cho từng Provider, đến việc thiết kế luồng dữ liệu (data flow) logic.
- **Hoàn thiện Giao diện:** Làm chủ được hệ thống thiết kế Material 3, cách tùy chỉnh ThemeData (màu sắc, font chữ), và tầm quan trọng của việc hỗ trợ đa ngôn ngữ (l10n) và đa chủ đề (Sáng/Tối) để tạo ra một sản phẩm chuyên nghiệp.