**Tài liệu Kỹ thuật – Ứng dụng MinCloset**

Phiên bản: 4.0

Ngày: 23 tháng 6 năm 2025

Tác giả: Made by Min

**1. Giới thiệu**

**1.1. Mục đích Tài liệu**

Tài liệu này cung cấp một cái nhìn tổng quan chi tiết về kiến trúc kỹ thuật, các thành phần, thư viện, chức năng, và phương thức hoạt động của ứng dụng MinCloset. 1 Mục tiêu là làm cơ sở cho việc bảo trì, mở rộng và chuyển giao dự án trong tương lai. 2

**1.2. Tổng quan Ứng dụng**

MinCloset là một ứng dụng di động đa nền tảng được xây dựng bằng framework Flutter, hoạt động như một trợ lý tủ đồ thông minh. 3 Ứng dụng cho phép người dùng số hóa tủ quần áo của họ, quản lý các vật phẩm, tạo các bộ trang phục (outfits) ảo, và nhận gợi ý phối đồ hàng ngày được cá nhân hóa bởi Trí tuệ nhân tạo (A.I.). 4

**2. Kiến trúc & Công nghệ**

**2.1. Kiến trúc tổng quan (Clean Architecture)**

Ứng dụng được cấu trúc theo các nguyên tắc của **Clean Architecture** để đảm bảo sự phân tách rõ ràng giữa các tầng, giúp việc phát triển, kiểm thử (testing) và bảo trì trở nên dễ dàng. 5 Luồng dữ liệu và sự phụ thuộc tuân theo mô hình:

**UI (Giao diện) → Notifier (Logic Giao diện) → Use Case (Logic Nghiệp vụ) → Repository (Truy cập Dữ liệu) → Data Source (Nguồn Dữ liệu)** 6

Điều này đảm bảo logic nghiệp vụ ở tầng trong không bị ảnh hưởng bởi sự thay đổi của các chi tiết triển khai ở tầng ngoài (như giao diện hay cơ sở dữ liệu).

**2.2. Quản lý Trạng thái (State Management): Riverpod**

**Riverpod** được chọn làm giải pháp quản lý trạng thái chính cho toàn bộ ứng dụng nhờ tính linh hoạt, an toàn và khả năng tích hợp tốt với kiến trúc bất biến (immutable). 7

* **StateNotifierProvider**: Được sử dụng cho các trạng thái phức tạp, có logic đi kèm (ví dụ: HomePageNotifier, AddItemNotifier).
* **FutureProvider**: Dùng để xử lý các tác vụ bất đồng bộ và cung cấp dữ liệu từ các nguồn như CSDL (ví dụ: outfitsProvider, closetsProvider).
* **Provider**: Dùng để cung cấp các đối tượng không thay đổi, chẳng hạn như các lớp Repository hoặc Service.
* **StateProvider**: Dùng cho các trạng thái đơn giản, có thể thay đổi trực tiếp (ví dụ: mainScreenIndexProvider).

**2.3. Cấu trúc Thư mục**

Dự án được tổ chức theo chức năng và tầng kiến trúc để dễ dàng điều hướng và quản lý. 8 Cấu trúc thư mục chính trong lib:

* constants: Chứa các giá trị hằng số dùng chung (ví dụ: AppOptions).
* domain: Tầng nghiệp vụ, chứa các models và use\_cases.
* helpers: Các lớp hỗ trợ, như db\_helper.dart.
* models: Các lớp mô hình dữ liệu (ví dụ: ClothingItem, Outfit).
* notifiers: Các lớp StateNotifier quản lý logic và trạng thái của từng màn hình/tính năng.
* providers: Nơi định nghĩa và khởi tạo các providers của Riverpod.
* repositories: Tầng truy cập dữ liệu, làm cầu nối giữa Use Case và Data Source.
* screens: Chứa các widget giao diện người dùng, được chia thành pages (các trang chính) và các màn hình con.
* services: Các dịch vụ bên ngoài (ví dụ: gọi API AI, API thời tiết).
* states: Các lớp trạng thái (state) bất biến, được sử dụng bởi các Notifier.
* theme: Định nghĩa giao diện chung cho ứng dụng (app\_theme.dart).
* widgets: Các widget có thể tái sử dụng trong toàn bộ ứng dụng.

**2.4. Thư viện & Dependencies chính**

File pubspec.yaml định nghĩa các thư viện bên thứ ba được sử dụng trong dự án. Các thư viện quan trọng bao gồm:

* flutter\_riverpod, hooks\_riverpod: Quản lý trạng thái.
* sqflite, path\_provider: Lưu trữ dữ liệu trên CSDL SQLite.
* shared\_preferences: Lưu trữ các cài đặt đơn giản.
* google\_generative\_ai: Tương tác với Gemini AI để phân loại và gợi ý.
* pro\_image\_editor: Thư viện chính cho chức năng "Xưởng phối đồ".
* geolocator, geocoding: Lấy vị trí và thông tin địa lý cho chức năng thời tiết.
* http: Thực hiện các cuộc gọi mạng đến API thời tiết.
* equatable: Giúp so sánh các đối tượng và state một cách dễ dàng.

**3. Các Thành phần Cốt lõi**

**3.1. Tầng Dữ liệu (Data Layer)**

* **Models:**
  + ClothingItem: Đại diện cho một vật phẩm quần áo, chứa các thuộc tính như id, tên, danh mục, màu sắc, ảnh, mùa, dịp sử dụng,...
  + Closet: Đại diện cho một tủ đồ, chứa id và tên.
  + Outfit: Đại diện cho một bộ đồ đã phối, chứa id, tên, ảnh, danh sách ID của các vật phẩm, và trạng thái isFixed.
  + CitySuggestion: Dùng trong chức năng tìm kiếm thành phố.
* **Data Sources:**
  + DatabaseHelper: Một lớp singleton quản lý việc khởi tạo, tạo bảng và thực hiện các thao tác CRUD (Create, Read, Update, Delete) trên CSDL SQLite.
  + SharedPreferences: Được sử dụng để lưu các cài đặt của người dùng như tên, thông tin cá nhân, và lựa chọn thành phố cho thời tiết.

**3.2. Tầng Repository**

Tầng Repository đóng vai trò là một API dữ liệu cho ứng dụng. 9 Nó trừu tượng hóa nguồn gốc của dữ liệu (CSDL hay mạng), giúp cho các Use Case không cần quan tâm đến chi tiết triển khai.

* ClosetRepository: Cung cấp các phương thức để lấy/thêm/sửa/xóa tủ đồ.
* ClothingItemRepository: Cung cấp các phương thức cho các vật phẩm.
* OutfitRepository: Cung cấp các phương thức cho các bộ đồ đã lưu.
* WeatherRepository, SuggestionRepository, CityRepository: Đóng gói các cuộc gọi đến WeatherService và SuggestionService.

**3.3. Tầng Nghiệp vụ (Domain Layer)**

Tầng này chứa logic nghiệp vụ cốt lõi của ứng dụng, được đóng gói trong các **Use Cases**. 10

* AnalyzeItemUseCase: Nhận một file ảnh, gọi đến ClassificationService để phân tích và trả về các thuộc tính của vật phẩm.
* GetOutfitSuggestionUseCase: Điều phối việc lấy thông tin thời tiết, danh sách vật phẩm hiện có, và gọi đến SuggestionService để tạo ra một gợi ý trang phục hoàn chỉnh.
* SaveOutfitUseCase: Xử lý việc lưu ảnh của bộ đồ vào bộ nhớ và lưu thông tin bộ đồ vào CSDL.
* Validate...UseCase: Các Use Case dùng để xác thực dữ liệu đầu vào, ví dụ như kiểm tra tên vật phẩm có bị trùng không.

**3.4. Tầng Trình bày (Presentation Layer)**

* **States:** Các lớp trạng thái (ví dụ: HomePageState, OutfitBuilderState) là các lớp bất biến (immutable) chứa tất cả dữ liệu cần thiết để vẽ nên một màn hình.
* **Notifiers:** Các lớp StateNotifier (ví dụ: HomePageNotifier, OutfitBuilderNotifier) chứa logic giao diện. Chúng nhận tương tác từ người dùng, gọi đến các Use Case hoặc Repository, và tạo ra một trạng thái (State) mới để cập nhật UI.
* **UI (Screens/Widgets):**
  + MainScreen: Widget chính chứa BottomAppBar và quản lý việc chuyển đổi giữa 4 trang chính.
  + HomePage: Bảng điều khiển chính, hiển thị thống kê, các hành động nhanh và gợi ý trang phục.
  + ClosetsPage: Quản lý việc hiển thị tất cả vật phẩm và các tủ đồ.
  + OutfitsHubPage: Hiển thị danh sách các bộ đồ đã lưu.
  + OutfitBuilderPage: Màn hình "Xưởng phối đồ".
  + ProfilePage: Hiển thị thông tin người dùng và các thống kê chi tiết.

**4. Phân tích Chức năng Chi tiết**

**4.1. Thêm đồ & Phân loại bằng AI**

Đây là một chức năng cốt lõi được thiết kế lại để tối ưu trải nghiệm.

* **Luồng hoạt động:** Người dùng chọn ảnh (từ camera hoặc thư viện) thông qua GlobalAddButton. Ứng dụng điều hướng đến AnalysisLoadingScreen để gửi tất cả ảnh đã chọn đến Gemini AI một cách đồng thời. 11 Sau khi phân tích xong, ứng dụng sẽ tự động điều hướng đến AddItemScreen (nếu có 1 ảnh) hoặc BatchAddItemScreen (nếu có nhiều ảnh) với toàn bộ dữ liệu đã được AI điền sẵn. 12
* **Thành phần kỹ thuật:**
  + ClassificationService: Chịu trách nhiệm xây dựng câu lệnh (prompt) chi tiết, gửi ảnh và prompt đến API của Google Gemini, và xử lý kết quả JSON trả về.
  + BatchAddItemNotifier: Quản lý trạng thái của toàn bộ luồng thêm hàng loạt, từ việc gọi AI, tạo các ItemNotifierArgs cho từng ảnh, cho đến việc xác thực và lưu tất cả vào CSDL.

**4.2. Xưởng phối đồ (OutfitBuilderPage)**

Chức năng này cho phép người dùng tự do sáng tạo các bộ đồ bằng cách kết hợp các vật phẩm. Giao diện được tái cấu trúc phức tạp để mang lại trải nghiệm tốt nhất.

* **Bố cục:** Màn hình sử dụng Stack để xếp chồng các lớp widget:
  1. **Lớp nền:** ProImageEditor được dùng làm canvas chính.
  2. **Lớp điều khiển:** Một AppBar và một thanh công cụ phụ tùy chỉnh được đặt ở trên cùng, chứa các nút như Lưu, Undo, Redo.
  3. **Lớp chức năng:** Một DraggableScrollableSheet được đặt ở trên cùng, chứa ItemBrowserView để hiển thị danh sách vật phẩm. Bảng trượt này luôn hiển thị một phần và có thể được kéo lên/xuống.
* **Tương tác:**
  1. Các nút bấm trên UI tùy chỉnh sẽ tương tác với ProImageEditor thông qua một GlobalKey.
  2. Việc lưu một bộ đồ được thực hiện bằng cách gọi captureEditorImage() để lấy ảnh kết quả, sau đó hiển thị một dialog tùy chỉnh để lấy tên và các thông tin khác, cuối cùng gọi SaveOutfitUseCase để lưu vào CSDL.
  3. Việc thêm vật phẩm vào canvas được thực hiện bằng cách gọi addLayer() từ \_editorKey.

**4.3. Gợi ý Trang phục hàng ngày**

* **Luồng hoạt động:** Khi người dùng nhấn nút "Gợi ý mới" trên HomePage, HomePageNotifier sẽ gọi GetOutfitSuggestionUseCase. Use case này sẽ:
  1. Xác định vị trí của người dùng (tự động qua geolocator hoặc thủ công qua shared\_preferences).
  2. Gọi WeatherRepository để lấy dữ liệu thời tiết hiện tại.
  3. Gọi ClothingItemRepository để lấy toàn bộ vật phẩm.
  4. Gọi SuggestionService và gửi toàn bộ thông tin trên trong một prompt đến Gemini AI.
  5. Nhận và phân tích phản hồi JSON từ AI để hiển thị cho người dùng.

#### ****Mô tả các Bài Test Chi tiết****

##### **6.2.1. Tầng Domain (Logic nghiệp vụ)**

* **test/domain/analyze\_item\_use\_case\_test.dart**
  + **Mục đích:** Kiểm tra AnalyzeItemUseCase. Nó giả lập (mock) ClassificationService để đảm bảo use case gọi đến service một cách chính xác và trả về kết quả phân tích vật phẩm như mong đợi.
* **test/domain/get\_outfit\_suggestion\_use\_case\_test.dart**
  + **Mục đích:** Kiểm tra GetOutfitSuggestionUseCase. Giả lập các Repository và các platform-interface (Geolocator, Geocoding) để xác minh use case xử lý đúng các kịch bản: lấy thời tiết theo vị trí tự động, lấy theo thành phố đã lưu, và xử lý khi người dùng từ chối quyền truy cập vị trí.
* **test/domain/save\_outfit\_use\_case\_test.dart**
  + **Mục đích:** Kiểm tra SaveOutfitUseCase. Nó giả lập OutfitRepository và path\_provider để đảm bảo use case xử lý đúng việc lưu ảnh bộ đồ vào bộ nhớ và gọi đến repository với dữ liệu Outfit chính xác.
* **test/domain/validate\_item\_name\_use\_case\_test.dart**
  + **Mục đích:** Kiểm tra ValidateItemNameUseCase. Xác minh logic kiểm tra tên vật phẩm trùng lặp trong các trường hợp: thêm mới, chỉnh sửa, và thêm hàng loạt.
* **test/domain/validate\_required\_fields\_use\_case\_test.dart**
  + **Mục đích:** Kiểm tra ValidateRequiredFieldsUseCase. Đảm bảo use case xác định chính xác các trường bắt buộc bị thiếu (tên, tủ đồ, danh mục) khi thêm một hoặc nhiều vật phẩm.

##### **6.2.2. Tầng Notifier (Logic giao diện)**

* **test/notifiers/\*\_notifier\_test.dart**
  + **Mục đích chung:** Các file test này (ví dụ add\_item\_notifier\_test.dart, profile\_page\_notifier\_test.dart) sử dụng ProviderContainer để kiểm tra các StateNotifier một cách độc lập. Chúng giả lập tất cả các phụ thuộc (repositories, use cases) và xác minh rằng việc gọi các phương thức trên notifier sẽ tạo ra các thay đổi State chính xác và tương tác đúng với các phụ thuộc.

##### **6.2.3. Tầng Repository (Truy cập dữ liệu)**

* **test/repositories/\*\_repository\_test.dart**
  + **Mục đích chung:** Các file test này (ví dụ clothing\_item\_repository\_test.dart, outfit\_repository\_test.dart) kiểm tra các lớp Repository. Chúng giả lập DatabaseHelper để đảm bảo các phương thức của repository gọi đúng các lệnh CSDL và chuyển đổi (parse) dữ liệu Map từ CSDL thành các đối tượng Model của ứng dụng một cách chính xác.

##### **6.2.4. Tầng Service (Dịch vụ)**

* **test/services/weather\_service\_test.dart**
  + **Mục đích:** Kiểm tra WeatherService. Nó giả lập http.Client để kiểm tra logic gọi API. Nó xác minh rằng service xây dựng đúng URL, gửi yêu cầu, xử lý phản hồi JSON thành công (status 200) và xử lý lỗi khi thất bại.

##### **6.2.5. Kiểm thử Giao diện (Widget Tests)**

* **test/widgets/\*\_test.dart**
  + **Mục đích chung:** Các file test này (ví dụ stats\_overview\_card\_test.dart, category\_selector\_test.dart) sử dụng WidgetTester để render các widget riêng lẻ vào một môi trường ảo. Chúng kiểm tra xem widget có hiển thị đúng dữ liệu được truyền vào và có phản hồi đúng với các tương tác của người dùng (như nhấn, nhập liệu) hay không.

##### **6.2.6. Kiểm thử Luồng (Flow/Integration Tests)**

* **test/flows/item\_add\_edit\_flow\_test.dart**
  + **Mục đích:** Kiểm tra luồng hoàn chỉnh của người dùng: từ ClosetsPage → nhấn vào một vật phẩm → mở AddItemScreen → sửa tên → lưu lại, và xác minh rằng phương thức updateItem trong repository được gọi với dữ liệu mới.
* **test/flows/navigation\_to\_add\_item\_test.dart**
  + **Mục đích:** Kiểm tra luồng từ MainScreen đến AddItemScreen sau khi chọn ảnh. Nó giả lập image\_picker và các use case để đảm bảo việc điều hướng diễn ra chính xác và màn hình AddItemScreen được điền sẵn dữ liệu từ kết quả phân tích AI giả lập.
* **test/flows/profile\_navigation\_test.dart**
  + **Mục đích:** Kiểm tra một lỗi cụ thể: đảm bảo ProfilePage không bị kẹt ở trạng thái loading khi người dùng điều hướng ra khỏi trang rồi quay trở lại.