CI/CD flutter key focus

Ứng dụng Ghi chú (Note App) hoặc To-do List có đồng bộ dữ liệu từ một API công khai.

Tại sao ý tưởng này lại hoàn hảo?

- **Có Logic để Unit Test:** Các hàm xử lý việc thêm, sửa, xóa, đánh dấu hoàn thành một ghi chú là đối tượng tuyệt vời cho **Unit Test**.
- **Có Giao diện (UI) để Widget Test:** Các màn hình như danh sách ghi chú, màn hình chi tiết, nút "Thêm mới" rất phù-hợp để viết **Widget Test**. Bạn có thể kiểm tra xem UI có cập nhật đúng khi trạng thái thay đổi không.
- **Có Luồng người dùng để Integration Test:** Luồng hoàn chỉnh từ lúc mở app, thêm một ghi chú mới, quay về màn hình chính và thấy ghi chú đó xuất hiện là một kịch bản lý tưởng cho **Integration Test**.
- **Có yếu tố bên ngoài (API):** Việc gọi API để lấy hoặc gửi dữ liệu cho phép bạn thể hiện kỹ năng **mocking** (giả lập) trong unit test và quản lý **biến môi trường/secrets** (như API key) trong CI/CD.

  Sử dụng API công khai miễn phí như JSONPlaceholder để giả lập việc lấy danh sách "todos" từ server.

Cấu Trúc Dự Án và Quy Trình Làm Việc DevOps

Trước cả khi viết pipeline, hãy thiết lập dự án của bạn một cách chuyên nghiệp.

- **Cấu trúc thư mục rõ ràng:** Chia code của bạn theo các lớp (layers) như `data` (xử lý API, database), `domain` (logic nghiệp vụ), và `presentation` (UI, state management). Cấu trúc này giúp việc viết test cho từng phần trở nên dễ dàng hơn rất nhiều.
- **Sử dụng Branching Model:** Áp dụng một quy trình làm việc Git đơn giản nhưng hiệu quả.
  - `main`: Nhánh chính, chỉ chứa code đã ổn định, sẵn sàng để release.
  - `develop`: Nhánh phát triển, chứa các tính năng đã hoàn thành và đang chờ được release.
  - `feature/<tên-tính-năng>`: Nhánh để phát triển một tính năng mới. Khi xong, bạn sẽ tạo Pull Request để merge vào `develop`.
- **Thiết lập Linter nghiêm ngặt:** Trong file `analysis_options.yaml`, hãy bật các quy tắc linting nghiêm ngặt. Đây là bước tự động hóa chất lượng code đầu tiên và đơn giản nhất.

Triển Khai Pipeline Từng Bước (Key Focus)

Giai đoạn 1: Viết Test trước, tự động hóa sau

Đây là nền tảng của cả quy trình. Một pipeline không có test để chạy thì gần như vô nghĩa.

1. **Viết Unit Test:**
    - Tạo một `repository` hoặc `service` để gọi API từ JSONPlaceholder.
    - Viết unit test cho class này. **Quan trọng:** Sử dụng một thư viện như `mockito` để "mock" `http.Client`. Bạn sẽ kiểm tra xem hàm của mình có xử lý đúng dữ liệu trả về từ API giả lập hay không, chứ không phải gọi API thật.
2. **Viết Widget Test:**
    - Viết test cho màn hình danh sách To-do.
    - Bạn cần giả lập dữ liệu (mock data) và kiểm tra xem `ListView` có hiển thị đúng số lượng item không.
    - Kiểm tra các hành động như nhấn vào nút "Thêm mới" có mở ra màn hình mới hay không.
3. **Viết Integration Test (Nâng cao):**
    - Viết một kịch bản test hoàn chỉnh: Mở app -> Nhấn nút thêm -> Nhập text -> Lưu -> Quay lại và kiểm tra item mới có trong danh sách không.

Giai đoạn 2: Tự động hóa Kiểm tra (CI) với GitHub Actions

Bây giờ, hãy tạo file `.github/workflows/ci.yml`.

- **Mục tiêu:** Đảm bảo mọi code được đẩy lên nhánh `develop` hoặc trong một Pull Request vào `main` đều phải vượt qua kiểm tra chất lượng.
- **Các bước (steps) trong pipeline:**
    1. Checkout code.
    2. Cài đặt Flutter.
    3. Chạy `flutter analyze` để kiểm tra lỗi code.
    4. Chạy `flutter test` để chạy tất cả unit test và widget test.
- **Kết quả:** Pipeline sẽ báo "xanh" (thành công) nếu tất cả các bước trên đều qua, và báo "đỏ" (thất bại) nếu có lỗi. Bạn có thể thiết lập quy tắc bảo vệ nhánh (branch protection rule) để cấm merge code nếu CI thất bại.

Giai đoạn 3: Tự động hóa Build

Mở rộng file CI ở trên hoặc tạo file mới (`build.yml`).

- **Mục tiêu:** Tự động tạo ra file cài đặt (`.aab` cho Android, `.ipa` cho iOS) sau khi code đã vượt qua tất cả các bài test.
- **Các bước thêm vào:** 5. Chạy `flutter build appbundle --release`. 6. **Lưu trữ Artifact:** Sử dụng action `actions/upload-artifact` để lưu file `.aab` vừa

build được. Điều này cho phép bạn tải về file cài đặt trực tiếp từ GitHub Actions sau khi pipeline chạy xong.

Giai đoạn 4: Tự động hóa Deployment (CD)

Đây là bước cuối cùng và ấn tượng nhất.

- **Mục tiêu:** Tự động gửi bản build đến cho người kiểm thử (testers) hoặc triển khai lên các kho ứng dụng.
- **Công cụ:** Kết hợp **GitHub Actions** và **Fastlane**.
- **Lộ trình đề xuất:**
    1. **Deploy lên Firebase App Distribution (Dễ nhất):**
        - Cài đặt Fastlane cho dự án.
        - Tạo một `lane` trong `Fastfile` để build và upload lên Firebase.
        - Trong GitHub Actions, bạn sẽ gọi lệnh `bundle exec fastlane deploy_to_firebase`.
        - **Bảo mật:** Lưu **Firebase App Distribution token** của bạn vào **GitHub Secrets**, tuyệt đối không ghi thẳng vào file workflow.
    2. **Optional | Deploy lên Google Play (Internal Testing Track):**
        - Đây là bước nâng cao hơn. Bạn cần tạo Service Account trên Google Play Console.
        - Dùng Fastlane để upload file `.aab` cùng với metadata (changelog) lên track Internal Testing.
        - Tương tự, file JSON của Service Account phải được lưu trong GitHub Secrets.

Sử dụng Github Action cho CI và FastLane cho CD

- **Phân chia vai trò rõ ràng:** GitHub Actions lo việc chung, Fastlane lo việc chuyên môn. Điều này giúp file cấu hình của bạn (`.yml` và `Fastfile`) gọn gàng và dễ hiểu.
- **Linh hoạt:** Kịch bản Fastlane của bạn có thể chạy ở bất cứ đâu, không chỉ trên GitHub Actions. Bạn có thể chạy nó ngay trên máy tính của mình để kiểm thử. Nếu sau này bạn muốn đổi sang một nền tảng CI/CD khác (như Bitrise, Jenkins), bạn chỉ cần mang kịch bản Fastlane đi theo.
- **Tập trung:** Bạn không cần phải viết các script phức tạp để xử lý việc ký app hay gọi API của Google/Apple ngay trong file YAML của GitHub Actions. Fastlane đã làm hết cho bạn.

Kết luận: Suy nghĩ của bạn là hoàn toàn chính xác. **Dùng GitHub Actions cho CI và Fastlane cho CD** là một phương pháp chuẩn mực, hiệu quả và được rất nhiều lập trình viên Flutter chuyên nghiệp áp dụng.

Về phần Deployment

Firebase App Distribution: Lựa chọn tốt nhất

Đây là công cụ của Google được tạo ra chính xác cho mục đích này: **phân phối các bản build thử nghiệm (pre-release) đến cho testers.** Nó hoàn toàn miễn phí và là cách chuyên nghiệp nhất để thay thế việc deploy lên chợ ứng dụng trong giai đoạn phát triển.

Tại sao nó hoàn hảo cho phần CD?

- **Hoàn toàn miễn phí:** Không có chi phí ẩn nào.
- **Mô phỏng quy trình thực tế:** Quy trình tự động build và tải ứng dụng lên Firebase App Distribution rất giống với việc tải lên các chợ ứng dụng thật. Bạn vẫn phải quản lý phiên bản, ký ứng dụng (signing), và gửi đi.
- **Dễ dàng cho người kiểm thử:** Bạn chỉ cần thêm email của người kiểm thử (có thể là email của chính bạn). Họ sẽ nhận được một email mời, làm theo hướng dẫn để tải app về máy và cài đặt.
- **Tích hợp hoàn hảo với Fastlane:** Fastlane có sẵn các "action" để tự động hóa việc upload lên Firebase App Distribution. Đây là sự kết hợp lý tưởng.
- **Hỗ trợ cả Android và iOS:** Bạn có thể phân phối cả file `.aab`/`.apk` và `.ipa`.

Workflow:

1. **CI (GitHub Actions):** Code được push -> Chạy `analyze` và `test`.
2. **CD (GitHub Actions + Fastlane):**
    - Nếu CI thành công, GitHub Actions sẽ gọi một lệnh của Fastlane (ví dụ: `fastlane deploy_to_firebase`).
    - Fastlane sẽ tự động build ứng dụng, ký và tải nó lên Firebase App Distribution.
    - Bạn (và các tester) sẽ nhận được email thông báo có phiên bản mới để tải về.

Đây là cách đáp ứng 100% yêu cầu của phần CD mà không tốn một đồng nào.

- Thanh search
- Phân loại hashtag
- Thêm ảnh
-