설계 문서

1. 개요

이 문서는 GitHub Explorer 애플리케이션의 아키텍처, 상태관리, 네트워킹, 주요 기능들, 테스트전략을 설명합니다.

2. 아키텍처

애플리케이션은 MVVM (Model-View-ViewModel) 아키텍처 패턴을 사용하여 다음과 같은 구조를 갖습니다:

- 모델 (Model): 데이터 구조를 정의합니다.
  - `GitHubUser` 와 `GitHubRepo` 클래스가 사용자 데이터를 모델링합니다.
- 뷰 (View): 사용자 인터페이스를 구성합니다.
  - `HomeScreen`과 `DetailScreen`이 UI를 정의합니다.
- 뷰모델 (ViewModel): 비즈니스 로직과 상태 관리를 처리합니다.
  - `UserViewModel(userViewModelProvider)`과 `RepoViewModel(repoViewModelProvider)`로 상태를 관리합니다.

3. 상태 관리

3.1. 상태 관리의 필요성

Flutter 애플리케이션에서 상태 관리는 필수적입니다. 특히 이 프로젝트는 GitHub API를 사용하여 사용자 및 저장소 목록을 가져와야 하며, 데이터를 안전하게 관리하고 UI에 반영하는 작업이 중요합니다. 이를 위해 상태 관리를 선택할 때 다음과 같은 기준을 고려했습니다:

- 비동기 데이터 처리: API 호출을 통해 데이터를 가져오고 처리하는 작업이 주를 이루며, 이를 위한 비동기 상태 관리가 필요합니다.
- 전역 상태 관리: 여러 화면에서 공통으로 사용하는 상태를 관리할 수 있는 전역 상태 관리 솔루션이 필요합니다.
- 유지보수성 및 확장성: 복잡도가 증가하더라도 코드 유지보수가 용이해야 하며, 확장 가능한 구조를 제공해야 합니다.
- 테스트 용이성: 비즈니스 로직과 상태 관리를 쉽게 테스트할 수 있어야 합니다.

3.2. 상태 관리 라이브러리 비교

1) Riverpod

- Riverpod은 `Provider`의 발전된 버전으로, Flutter 상태 관리 라이브러리 중 하나입니다.
- 장점
  - 글로벌 접근: `context` 없이 전역적으로 상태 관리가 가능해 어디서나 접근할 수 있습니다.
  - 의존성 주입 및 관리: 서비스 및 의존성 주입이 간편하며, 의존성 간의 관계를 쉽게 관리할 수 있습니다.
  - 비동기 상태 관리: `AsyncNotifier`와 같은 도구를 사용해 비동기 데이터 처리 및 상태 관리가 간편합니다.
  - 타입 안전성: Riverpod은 타입 안전성을 보장해 상태 관리 시 발생할 수 있는 오류를 컴파일 타임에 잡아낼 수 있습니다.
  - 테스트 용이성: `ProviderContainer`를 통해 테스트 코드 작성이 용이하며, 상태 및 비즈니스 로직을 분리하기 쉽습니다.

- 단점
    - 학습 곡선: Riverpod의 개념과 문법이 처음 접하는 사람에게는 다소 복잡할 수 있습니다.
    - 생태계: Bloc이나 GetX만큼 널리 사용되지는 않으므로, 생태계와 관련 리소스가 다른 옵션보다 적을 수 있습니다.

2) Bloc

- Bloc 패턴은 Flutter 애플리케이션에서 상태를 관리하는 또 다른 대표적인 방법입니다. 주로 복잡한 애플리케이션에 사용됩니다.
- 장점
    - 단방향 데이터 흐름: Bloc은 데이터 흐름을 명확하게 관리할 수 있어 큰 규모의 애플리케이션에서 구조를 유지하기 쉽습니다.
    - 테스트 용이성: 이벤트 기반 상태 전환이 명확하기 때문에 테스트 작성이 용이합니다.
    - 커뮤니티 및 리소스: Bloc은 Flutter 생태계에서 널리 사용되며, 다양한 자료와 커뮤니티 지원이 잘 되어 있습니다.
- 단점
    - 복잡성: 작은 규모의 애플리케이션에 사용하기에는 다소 과한 구조를 요구할 수 있으며, 코드가 길어지는 경향이 있습니다.
    - 러닝 커브: 이벤트와 상태를 정의하고 관리하는 방식은 초기에 러닝 커브가 높을 수 있습니다.

3) GetX

- GetX는 Flutter에서 빠르고 간편하게 상태 관리, 라우팅, 의존성 주입을 할 수 있는 올인원 솔루션입니다.
- 장점
    - 간결함: GetX는 코드가 매우 간결하며, 상태 관리 및 라우팅, 의존성 주입을 단일 라이브러리로 처리할 수 있습니다.
    - 러닝 커브: 다른 상태 관리 라이브러리에 비해 배우기 쉽고, 빠르게 적용할 수 있습니다.
    - 성능: 상태 변경에 대한 반응성이 뛰어나고, 코드가 간결해 성능 최적화에도 유리합니다.
- 단점
    - 구조적이지 않음: 프로젝트가 커질수록 코드 구조가 복잡해지고, 유지보수성이 떨어질 수 있습니다.
    - 타입 안전성 부족: 타입 안전성 측면에서 Riverpod이나 Bloc에 비해 약할 수 있으며, 대규모 프로젝트에서는 유지보수가 어렵습니다.
    - 커뮤니티 및 생태계: Riverpod이나 Bloc만큼의 강력한 생태계를 가지고 있지 않아, 대규모 프로젝트에서는 문제가 될 수 있습니다.

3.3. Riverpod 선택 이유

Riverpod을 선택한 이유는 이 프로젝트의 요구사항에 가장 적합한 상태 관리 솔루션이기 때문입니다. 주요 이유는 다음과 같습니다.

1. 비동기 상태 관리의 용이성: `AsyncNotifier`를 활용해 GitHub API와의 비동기 작업을 쉽게 관리할 수 있습니다. 또한 로딩 상태 유지 및 에러 처리에 유리합니다.

2. 글로벌 상태 관리: Riverpod은 전역적으로 상태를 관리하기 쉽고, 여러 화면에서 상태를 쉽게 공유할 수 있습니다.
3. 테스트 용이성: `ProviderContainer`를 사용해 `BuildContext` 없이 상태를 테스트할 수 있어, 단위 테스트 및 통합 테스트 작성이 용이합니다.
4. 타입 안전성: 타입 오류를 줄이고 컴파일 타임에 오류를 잡아낼 수 있어 안정적인 애플리케이션 개발이 가능합니다.
5. 유연성과 확장성:
    - 상태 관리 로직을 UI 로직에서 분리할 수 있어 MVVM 패턴을 적용하기 용이하며, 유지보수가 쉽습니다.
    - 작은 규모의 프로젝트부터 큰 규모의 프로젝트까지 확장성이 뛰어납니다.

> Riverpod은 이 프로젝트의 비동기 데이터 처리, 전역 상태 관리, 테스트 요구사항을 만족시켜 주는 최적의 선택입니다. Bloc은 강력하지만, 이 프로젝트의 복잡도에 비해 과한 선택일 수 있으며, GetX는 간편하지만 확장성에서 부족할 수 있습니다.

4. 네트워킹

- Dio: 네트워크 요청을 처리하기 위해 Dio를 사용합니다. Dio는 HTTP 요청과 응답을 쉽게 관리할 수 있으며, 인터셉터를 통해 요청과 응답을 로깅합니다.
- 인터셉터: 요청과 응답을 로깅하기 위해 Dio의 인터셉터를 사용합니다. 이를 통해 네트워크 통신을 모니터링하고, 디버깅을 용이하게 합니다.
- Logger: 로깅을 처리하기 위해 Logger 패키지를 사용합니다. 이는 디버깅과 애플리케이션 상태 모니터링에 유용합니다.

5. 주요 기능들

5.1. 페이지네이션

- 사용자 목록이 길어질 경우 성능을 유지하면서 데이터를 동적으로 로드합니다. 사용자 목록 스크롤 시, 일정 지점에 도달하면 추가 데이터를 로드하여 사용자 경험을 향상시킵니다.

5.2. 광고 배너

- 사용자 목록의 특정 위치(10번째, 20번째, 30번째 항목)마다 광고 배너를 표시합니다. 배너는 클릭 시 지정된 웹사이트로 이동합니다.

5.3. 다크 모드 지원

- 앱은 다크 모드를 지원하며, 사용자의 환경에 맞게 테마를 조정합니다. `lightTheme`과 `darkTheme`으로 정의된 테마를 사용하여 사용자 환경에 맞는 테마를 제공합니다.

5.4. 폰트 고정 지원

- 앱은 디바이스의 폰트 설정에 영향을 받지않고 고정된 폰트 크기를 유지합니다.

6. 테스트 전략

***(Simulator 또는 Emulator 를 실행합니다.)**

6.1. 단위 테스트 (Unit Test) : 각 컴포넌트의 비즈니스 로직을 검증합니다.

- `UserViewModel(userViewModelProvider)` 과
- `RepoViewModel(repoViewModelProvider)` 의 상태 관리 로직을 검증합니다.

```
flutter test
```

6.2. 통합 테스트 (Integration Test) : 모듈 간 상호작용을 검증합니다.

- 사용자 목록이 제대로 로드되고 화면에 표시되는지,
- 스크롤링과 페이지네이션에 따라서 목록을 추가로 불러드리는지,
- 상세화면으로 넘어가는지 테스트합니다.

```
flutter test integration_test/integration_test.dart
```

6.3. E2E 테스트 (E2E Test) : 애플리케이션 전체 흐름을 검증합니다.

- 홈화면에서 로드된 사용자 목록을 스크롤링으로 탐색하고, 상세 화면으로 이동하는 과정 전체를 테스트합니다.

```
flutter test integration_test/e2e_test.dart
```