

Flutter 바이트코딩 튜토리얼

- ✓ 서브태스크가 있는 TODO 앱 만들기

Flutter 바이브코딩을 사용하여 서브태스크 기능이 있는 TODO 앱을 만드는 실전 튜토리얼입니다.

학습 목표

- Flutter 바이브코딩 개발 환경 구성
- Gemini CLI를 활용한 PRD 및 프로젝트 자동 생성
- PLAN.md와 TASKS.md를 활용한 반복적 개발 프로세스
- setState와 InheritedModel을 사용한 상태 관리
- shared_preferences를 사용한 로컬 데이터 저장

튜토리얼 개요

이 튜토리얼에서는 다음 기능을 가진 TODO 앱을 만듭니다:

- TODO 항목 추가, 수정, 삭제
- 각 TODO에 서브태스크 추가 가능
- 완료 상태 체크
- 로컬 저장소에 데이터 영구 저장
- setState와 ChangeNotifier를 활용한 상태 관리

사전 요구사항

- macOS 환경 (Linux/Windows는 WSL2 권장)
- 터미널 사용 경험
- Flutter에 대한 기본 이해

1. 환경 구성

1.1. Flutter 바이브코딩 환경 설치

먼저 `install.sh` 스크립트를 실행하여 필요한 도구들을 설치합니다.
(자세한 내용은 `SETUP.ko.md` 문서 참고)

```
# 스크립트에 실행 권한 부여
chmod +x install.sh doctor.sh

# Flutter 바이브코딩 환경 설치
./install.sh

# 터미널 재시작 또는 환경 변수 다시 로드
source ~/.zshrc # zsh의 경우
# 또는
source ~/.bashrc # bash의 경우
```

1.2. 환경 검증

`doctor.sh` 스크립트로 설치된 환경을 검증합니다.
(자세한 내용은 `SETUP.ko.md` 문서 참고)

```
./doctor.sh
```

1.3. Gemini CLI 설정

Gemini CLI를 사용하기 위해 API 키를 설정합니다.

```
gemini
```

Gemini CLI가 실행되면 `/auth` 를 입력하여 인증을 진행합니다:

1. Google AI Studio에서 API 키 발급 (<https://aistudio.google.com/apikey>)
2. API 키 입력
3. 기본 모델 선택

1.4. Gemini CLI Firebase 확장 설치 (선택사항)

Firebase와 통합하려면 Firebase 확장을 설치합니다.

(이번 튜토리얼에서는 TODO앱을 로컬에만 저장하기 때문에 불필요합니다.)

```
gemini extensions install https://github.com/gemini-cli-extensions/firebase/
```


2. Gemini CLI로 PRD 자동 구성

2.1. PRD란?

PRD(Product Requirements Document)는 제품 요구사항 문서로, 개발할 앱의 기능과 요구사항을 정의합니다.

2.2. Gemini CLI로 PRD 생성

작업 디렉토리를 만들고, 터미널에서 작업 디렉토리로 이동 후 gemini cli로 PRD를 생성합니다.

서브태스크 기능이 있는 TODO 앱의 PRD를 작성해줘.

다음 요구사항을 포함해줘:

- TODO 항목 추가, 수정, 삭제
- 각 TODO에 여러 개의 서브태스크 추가 가능
- TODO와 서브태스크의 완료 상태 체크
- shared_preferences를 사용한 로컬 데이터 저장
- setState와 InheritedModel을 사용한 상태 관리
- Material Design 3 UI

PRD는 PRD.md 파일로 저장해줘.

2.3. PRD 검토 및 수정

생성된 `PRD.md` 파일을 검토하고 필요시 수정합니다.

PRD에 포함되어야 할 주요 섹션:

1. 프로젝트 개요: 앱의 목적과 목표
2. 주요 기능: 상세한 기능 목록
3. 기술 스택: Flutter, shared_preferences, setState, InheritedModel
4. 데이터 모델: Todo와 SubTask 모델 구조
5. UI/UX 요구사항: 화면 구성과 사용자 흐름
6. 제약사항: 로컬 저장만 사용, 인증 없음 등

3. Gemini CLI로 Flutter 프로젝트 생성

3.1. FVM을 사용한 프로젝트 생성

gemini에서 FVM으로 Flutter 프로젝트를 생성하고 초기 설정을 진행하기 위해 자연어로 요청합니다.

```
fvm flutter 스테이블 버전을 사용해서 subtask_todo라는 프로젝트를 생성해줘
```

3.2. 의존성 추가

`pubspec.yaml` 에 필요한 의존성을 추가합니다.

`subtask_todo` 프로젝트에 로컬 데이터를 저장하기 위해 `shared_preferences` 의존성을 추가해줘

또는 수동으로 추가:

```
dependencies:  
  flutter:  
    sdk: flutter  
  shared_preferences: ^2.2.2
```

```
# 의존성 설치  
fvm flutter pub get
```

3.3. 프로젝트 구조 확인

생성된 프로젝트의 기본 구조를 확인합니다.

```
todo_app/  
├── lib/  
│   └── main.dart  
├── test/  
├── pubspec.yaml  
├── README.md  
└── ...
```

4. PLAN.md, TASKS.md 루프로 바이브코딩

4.1. 바이브코딩이란?

바이브코딩은 AI와 협업하여 개발하는 방식으로, 계획(PLAN.md)과 작업(TASKS.md)을 반복적으로 업데이트하며 개발을 진행합니다.

핵심 개념:

- **PLAN.md**: 전체 개발 계획
- **TASKS.md**: 구체적인 작업 목록과 진행 상황

4.2. PLAN.md 작성

Gemini CLI로 전체 개발 계획을 작성합니다.

PRD.md를 참고해서 PLAN.md 파일을 작성해줘.

다음 내용을 포함해줘:

1. 프로젝트 아키텍처 (폴더 구조, 레이어 구성)
2. 데이터 모델 설계 (Todo, SubTask 클래스)
3. 상태 관리 전략 (setState vs ChangeNotifier 사용 시점)
4. 로컬 저장소 구현 방법 (shared_preferences 활용)
5. UI 화면 구성 (홈 화면, 상세 화면, 추가/수정 다이얼로그)
6. 구현 순서 (우선순위별)"

4.3. TASKS.md 작성

PLAN.md를 기반으로 구체적인 작업 목록을 작성합니다.

PLAN.md를 참고해서 TASKS.md 파일을 작성해줘.

각 작업을 체크리스트 형태로 작성하고,

작업 상태를 [], [~], [x]로 표시할 수 있게 해줘.

- []: 미완료
- [~]: 진행 중
- [x]: 완료

4.5. 바이브코딩 루프 실행

이제 Gemini CLI와 함께 반복적으로 개발을 진행합니다.

Step 1: 첫 번째 작업 시작

PLAN.md와 TASKS.md를 참고해서
페이지 1의 Todo 모델 클래스를 구현해줘.
lib/models/todo.dart 파일을 생성하고,
다음은 포함해줘:

- 모든 필드 (`id`, `title`, `description`, `isCompleted`, `subtasks`, `createdAt`)
- `fromJson`, `toJson` 메서드
- `copyWith` 메서드

구현 후 TASKS.md를 업데이트해줘.

Step 2: 코드 검토 및 다음 작업

이제 SubTask 모델을 구현해줘.
lib/models/subtask.dart 파일을 생성하고,
Todo 모델과 비슷하게 구현해줘.
완료되면 TASKS.md를 업데이트해줘.

Step 3: 저장소 서비스 구현

이제 `StorageService`를 구현해줘.

`lib/services/storage_service.dart` 파일을 생성하고,
`shared_preferences`를 사용해서 `Todo` 목록을 저장하고 불러오는 기능을 구현해줘.

- `saveTodos(List<Todo> todos)`: 저장
- `loadTodos()`: `Future<List<Todo>>` - 불러오기

`TASKS.md`도 업데이트해줘.

Step 4: 상태 관리 구현

TodoInheritedModel을 구현해줘.

lib/models/todo_inherited_model.dart 파일을 생성하고,
InheritedModel을 사용해서 다음 기능을 구현해줘:

- Todo 목록 관리
- addTodo, updateTodo, deleteTodo 메서드
- toggleTodoComplete 메서드
- StorageService와 연동하여 자동 저장
- aspect 기반으로 특정 Todo만 업데이트할 수 있도록 구현

TASKS.md도 업데이트해줘.

Step 5: UI 구현

홈 화면을 구현해줘.

`lib/screens/home_screen.dart` 파일을 생성하고,
다음을 포함해줘:

- AppBar with 제목
- `TodoInheritedModel`을 사용한 TODO 목록 표시
- `FloatingActionButton`으로 TODO 추가
- `ListView`로 TODO 항목 표시

그리고 `lib/main.dart`를 수정해서
`TodoInheritedModel`로 전역 상태를 제공하고
홈 화면을 표시해줘.

`TASKS.md`도 업데이트해줘.

Step 6: 반복 개발

위와 같은 방식으로 TASKS.md의 모든 항목을 완료할 때까지 반복합니다.

TASKS.md를 보고 다음으로 할 작업을 진행해줘.

4.7. 바이트코딩 팁

1. 작은 단위로 작업: 한 번에 하나의 기능만 구현
2. 자주 테스트: 각 기능 구현 후 바로 테스트
3. **TASKS.md** 업데이트: 진행 상황을 계속 업데이트
4. 명확한 요청: Gemini CLI에게 구체적으로 요청
5. 코드 검토: 생성된 코드를 반드시 검토하고 이해

4.8. InheritedModel의 장점

왜 InheritedModel을 사용하나요?

InheritedModel은 Flutter에서 제공하는 고급 상태 관리 도구로, 다음과 같은 장점이 있습니다:

1. 효율적인 리빌드: Aspect를 사용하여 필요한 위젯만 선택적으로 리빌드
2. 외부 의존성 없음: Flutter SDK에 내장되어 있어 별도 패키지 불필요
3. 세밀한 제어: 어떤 위젯이 언제 업데이트될지 정확히 제어 가능
4. 성능 최적화: 불필요한 리빌드를 줄여 앱 성능 향상
5. 학습 가치: Flutter의 위젯 트리와 상태 전파 메커니즘을 깊이 이해

InheritedModel vs Provider

특징	InheritedModel	Provider
의존성	없음 (내장)	외부 패키지 필요
학습 곡선	낮음	높음

5. 실행 및 테스트 후 이슈 대응

5.1. 앱 실행

iOS 시뮬레이터에서 실행

iOS 시뮬레이터로 실행해줘

Android 에뮬레이터에서 실행

Android 에뮬레이터로 실행해줘

Chrome에서 실행 (웹)

웹으로 빌드 후 크롬브라우저에서 실행해줘

5.2. 핫 리로드 사용

앱 실행 중 코드를 수정한 후:

- **r** 키: 핫 리로드 (화면 새로고침)
- **R** 키: 핫 리스타트 (앱 재시작)
- **q** 키: 앱 종료

5.3. 테스트 작성 (선택사항)

단위 테스트 예시

```
// test/models/todo_test.dart
import 'package:flutter_test/flutter_test.dart';
import 'package:todo_app/models/todo.dart';

void main() {
  group('Todo', () {
    test('should create Todo with default values', () {
      final todo = Todo(title: 'Test Todo');

      expect(todo.title, 'Test Todo');
      expect(todo.description, '');
      expect(todo.isCompleted, false);
      expect(todo.subtasks, isEmpty);
    });

    test('should convert to JSON and back', () {
      final todo = Todo(
        title: 'Test Todo',
        description: 'Test Description',
      );

      final json = todo.toJson();
      final fromJson = Todo.fromJson(json);

      expect(fromJson.title, todo.title);
      expect(fromJson.description, todo.description);
    });
  });
}
```

테스트 실행

```
# 모든 테스트 실행  
fvm flutter test
```

```
# 특정 테스트 파일 실행  
fvm flutter test test/models/todo_test.dart
```

```
# 커버리지 리포트 생성  
fvm flutter test --coverage
```

5.6. Gemini CLI로 이슈 해결

이슈가 발생하면 Gemini CLI에게 도움을 요청합니다.

앱을 실행했는데 다음 에러가 발생해:
[에러 메시지 붙여넣기]

어떻게 해결할 수 있을까?
관련 코드도 수정해줘.

6. 문서화

└

6.1. README.md 작성

프로젝트의 README.md를 작성합니다.

이 프로젝트의 README.md 파일을 작성해줘.

다음 내용을 포함해줘:

- 프로젝트 소개
- 주요 기능
- 기술 스택
- 설치 및 실행 방법
- 프로젝트 구조
- 스크린샷 (스크린샷 추가 가이드)
- 라이선스








6.2. 문서화 팁

1. 명확성: 다른 개발자가 이해할 수 있도록 명확하게 작성

7. 마치며

축하합니다! Flutter 바이브코딩으로 서브태스크 기능이 있는 TODO 앱을 완성했습니다.

이 튜토리얼에서 배운 내용:

-  Flutter 바이브코딩 환경 구성
-  Gemini CLI를 활용한 AI 협업 개발
-  PLAN.md와 TASKS.md를 사용한 체계적인 개발
-  setState와 InheritedModel을 활용한 효율적인 상태 관리
-  Aspect 기반 선택적 위젯 리빌드
-  shared_preferences로 로컬 데이터 저장
-  디버깅과 이슈 해결 방법

감사합니다!

Happy Flutter Vibe Coding! 🚀