📄 DEVELOPMENT_PLAN.md

Проект: MoodTail

Цель: Реализация внутреннего MVP приложения для отслеживания настроения собаки на базе SwiftUI + MVVM + Core Data.

Прицел: Масштабируемость (CloudKit, мульти-питомцы, аналитика).

🧭 Этапы разработки MoodTail MVP

1. 📁 Стартовая структура проекта

Цель: Подготовить фундамент для масштабируемой архитектуры.

- [ ] Создать Xcode-проект (SwiftUI App Lifecycle)
- [ ] Настроить схемы: `Debug`, `Release`
- [ ] Настроить Bundle ID, запуск на симуляторе
- [ ] Создать структуру каталогов:

```
├── App
├── Core
│   ├── Storage
│   └── AppState
├── Modules
│   ├── MoodLogger
│   ├── HistoryTracker
│   └── AdviceEngine
├── SharedUI
├── Resources
└── Tests
```

- [ ] Подключить:

- SwiftLint
- SwiftFormat (опционально)
- CI (GitHub Actions / Fastlane — при необходимости)

---

2. 🧠 Модель данных и Core Data

Цель: Определить основу хранения данных.

- [ ] Модели:
  - `MoodType` (enum)
  - `MoodEntry`: `id`, `date`, `type`, `note`, `petId`
  - `Pet`: имя, фото, цель отслеживания
- [ ] Настроить Core Data через `CoreDataStack`
  - `MoodEntity`, `PetEntity`
- [ ] Реализовать `MoodStorageProtocol` и `MoodStorageImpl`:
  - `save(entry:)`
  - `fetchAll(for:)`
  - `fetchLast(for:)`
  - `delete(entry:)`

---

3. 🔌 Архитектура и Dependency Injection

Цель: Обеспечить масштабируемую архитектуру и независимость компонентов.

- [ ] Внедрить MVVM: View ↔ ViewModel ↔ Storage
- [ ] Создать:
  - `AppState`: `ObservableObject` с `@MainActor`
  - `PetContextProvider`
  - `MoodHistoryProvider` (абстракция истории)
- [ ] DI:

- Базово — через `@EnvironmentObject` (`AppState`)
- Расширенно — через `Resolver` / `Factory` (опционально)

---

4. 🧩 Бизнес-логика модулей

Цель: Инкапсулировать функциональность по SRP.

▸ MoodLogger

- [ ] ViewModel: логика выбора и сохранения настроения
- [ ] Сценарии: добавление, редактирование, сброс

▸ HistoryTracker

- [ ] ViewModel: загрузка, фильтрация, группировка по датам
- [ ] Поставщик истории: `MoodHistoryProvider`

▸ AdviceEngine

- [ ] Ввод: `[MoodEntry]`, `GoalTrackingModel`
- [ ] Вывод: `AdviceModel?`
- [ ] Логика: проверка паттернов (например, "3 тревоги подряд")

---

5. 🖼️ UI и дизайн-система

Цель: Сформировать единый визуальный язык и переиспользуемые компоненты.

- [ ] Подключить кастомные шрифты, палитру цветов
- [ ] UI-компоненты:
  - `MoodButton`
  - `MoodCard`
  - `AdviceView`
  - `ProgressRing`
  - `EmptyStateView`

- [ ] Экранная структура:
  - `MoodLoggerView`
  - `HistoryView`
  - `AdviceView`
  - Переходы между ними — через `NavigationStack`

---

6. 💾 Интеграция состояния

Цель: Обеспечить связность UI ↔ ViewModel ↔ Model

- [ ] `MoodLoggerViewModel` → вызывает `MoodStorage.save()`
- [ ] `HistoryTrackerViewModel` → читает через `MoodHistoryProvider`
- [ ] `AdviceEngine` вызывается после сохранения/чтения
- [ ] Внедрение AppState (текущий питомец, состояние онбординга)

---

7. 🚪 Онбординг и навигация

Цель: Замкнуть пользовательский поток от первого запуска до ввода данных.

- [ ] `OnboardingView`: добавление питомца, цели
- [ ] `GoalSelectionView`: выбор, что важно отслеживать
- [ ] Навигация через:
  - `NavigationStack`
  - `@State`, `@Binding`, `@EnvironmentObject`

---

8. ✅ MVP-тестирование

Цель: Проверить жизнеспособность основных сценариев.

- [ ] Поток: `Logger → Core Data → History → Advice`
- [ ] Проверка:
  - Пустых состояний (`NoMoodYetView`)
  - Ошибок сохранения и чтения

- [ ] Unit-тесты:
  - `MoodStorageTests`
  - `AdviceEngineTests`
  - `HistoryTrackerTests`

---

9. 🎁 Подготовка внутреннего MVP

Цель: Финализация для демонстрации или TestFlight.

- [ ] Очистить проект от моков и временных заглушек
- [ ] Добавить:
  - Подтверждение действий (модалки)
  - Мягкие анимации
  - Уведомления об успехе (`+1 забота`)
- [ ] Документация:
  - `KNOWN_ISSUES.md`
  - `TODO_NEXT.md`
  - `README.md`

✅ Результат

Компонент	Состояние
MoodLogger	✅ Реализован
История настроений	✅ Отображается
Рекомендации	✅ Генерируются
UI	✅ На базе Design System
Масштабируемость	✅ Подготовлена
MVP	✅ Работоспособный

➡️ SOON Подготовка к следующему этапу

Готовность к:

- TestFlight (после финальной отладки)
- Подключению iCloud / CloudKit
- Внедрению мульти-питомцев (`PetEntity`, `PetContext`)
- Расширению аналитики и визуализаций (`Charts`, `Insights`)

Если хочешь, могу оформить это в `Notion`, `.md` или как PDF-презентацию для команды.

Спросить ChatGPT

Инструменты

ChatGPT может допускать ошибки. Проверьте важную информацию. См. настройки cookie-файлов.