Mindful

Guide Architectural de Référence

Kickstart du Projet - 3 Parties

Mindful : Vue d'Ensemble

Journal personnel intelligent avec Apple Intelligence

Une application de journal intime moderne pour iOS qui utilise Swift 6, SwiftUI, et Apple Intelligence pour vous aider à capturer vos pensées, suivre vos humeurs, et réfléchir sur votre vie.

Features Principales

Core Features

- ✍️ **Journal intime riche** : Texte formaté avec support Markdown
- 📸 **Photos & souvenirs** : Attachez plusieurs photos par entrée
- 😊 **Mood tracking** : Suivez votre humeur avec 5 états émotionnels
- 🏷️ **Tags intelligents** : Tags manuels et auto-tagging avec NLP
- ☁️ **iCloud sync** : Synchronisation automatique entre tous vos appareils
- 🔒 **Privacy-first** : Données chiffrées, aucun serveur tiers

Features Avancées (Jours 2 & 3)

Apple Intelligence (iOS 26+)

- 🤖 Sentiment analysis : Détection automatique de l'humeur pendant la frappe
- ✨ Writing Tools : Amélioration, résumé, et reformulation natifs
- 🗣️ App Intents : "Hey Siri, add a journal entry about my day"
- 🧠 Smart prompts : Rappels contextuels et insights personnalisés

Interface Liquid Glass

- 🪩 Design moderne avec effets glass/blur
- ✨ Animations fluides et naturelles
- 🌓 Support Dark Mode complet

Architecture Globale

```
MindfulApp
├── Data Layer (SwiftData) ← Jour 1
│   ├── @Model JournalEntry
│   ├── @Model Photo
│   └── ModelContainer
├── Domain Layer (Actors) ← Jour 1
│   ├── JournalRepository
│   ├── PhotoProcessor
│   ├── ExportService
│   └── SentimentAnalyzer
├── Presentation Layer (SwiftUI) ← Jour 2
│   ├── Views (Timeline, Editor, Detail, Stats)
│   ├── Components (GlassCard, MoodSelector)
│   └── ViewModels (@Observable)
└── Intelligence Layer ← Jour 3
    ├── App Intents (Siri)
    ├── NaturalLanguage (sentiment, keywords)
    └── Writing Tools (iOS 18)
```

Stack Technique Complète

```
SwiftUI + Swift 6
├── SwiftData (persistence) ← Jour 1
├── Actors (thread-safety) ← Jour 1
├── async/await (concurrency) ← Jour 1
├── NaturalLanguage (AI on-device) ← Jour 3
├── App Intents (Siri integration) ← Jour 3
└── Swift Charts (analytics) ← Jour 2
```

Objectifs :

- Zero data races (Swift 6)
- 100% thread-safe
- Privacy-first (on-device AI)
- Performance optimale

Table des Matières

Partie 1

Backend Local de l'App

Stack Technique Jour 1

- SwiftData : Persistence moderne (remplace Core Data)
- Actors : Thread-safety garantie
- async/await : Toutes les opérations asynchrones
- Sendable : Sécurité compile-time Swift 6

Objectif : Zero data races, 100% thread-safe

Models à Créer

```
@Model class JournalEntry
@Model class Photo

struct Location: Codable, Sendable
struct Weather: Codable, Sendable
enum Mood: String, Codable, Sendable
```

Ressources :

- SwiftData Documentation
- WWDC23: Meet SwiftData
- WWDC23: Model your schema with SwiftData

JournalEntry - Propriétés Clés

```
@Model final class JournalEntry {
    var id: UUID
    var createdAt: Date
    var modifiedAt: Date
    var title: String
    var content: String
    var mood: Mood
    var tags: [String]

    @Relationship(deleteRule: .cascade)
    var photos: [Photo]

    var location: Location?
    var weather: Weather?
    var isFavorite: Bool
}
```

Mood - Enum pour Humeurs

```
enum Mood: String, Codable, Sendable {
    case veryHappy, happy, neutral, sad, verySad

    var emoji: String { /* 😄🙂😐😔😢 */ }
    var displayName: String { /* "Très Joyeux" */ }
    var color: Color { /* .green, .blue, .red */ }
    var score: Int { /* -2 à +2 */ }

    static func from(sentimentScore: Double) -> Mood
}
```

Photo - Model pour Images

```
@Model final class Photo {
    var id: UUID
    var createdAt: Date

    @Attribute(.externalStorage)
    var imageData: Data

    var thumbnailData: Data
    var caption: String?
    var entry: JournalEntry? // Inverse relationship
    var width: Int
    var height: Int
}
```

SwiftData Configuration

```
@MainActor
class DataController {
    static let shared = DataController()
    let container: ModelContainer

    private init() {
        let schema = Schema([
            JournalEntry.self,
            Photo.self
        ])

        let config = ModelConfiguration(
            cloudKitDatabase: .automatic
        )

        container = try! ModelContainer(
            for: schema,
            configurations: config
        )
    }
}
```

Actors à Implémenter

```
actor JournalRepository {
    // CRUD operations
    func create(_ entry: JournalEntry) async throws
    func fetchAll() async throws -> [JournalEntry]
    func update(_ entry: JournalEntry) async throws
    func delete(_ entry: JournalEntry) async throws
    func search(query: String) async throws -> [JournalEntry]
}

actor PhotoProcessor {
    func process(_ image: UIImage) async throws -> ProcessedPhoto
    func generateThumbnail(_ image: UIImage) async -> Data
    func compress(_ data: Data) async -> Data
}

actor ExportService {
    func exportToPDF(_ entries: [JournalEntry]) async throws -> URL
    func exportToMarkdown(_ entries: [JournalEntry]) async throws -> URL
    func exportToJSON(_ entries: [JournalEntry]) async throws -> URL
}
```

JournalRepository - Méthodes Clés

CRUD :

- `create()`, `update()`, `delete()`
- `fetchAll()`, `fetch(from:to:)`, `fetch(mood:)`

Recherche :

- `search(query: String)` - Full-text search
- `fetchFavorites()` - Entrées favorites

Stats :

- `count()` - Nombre total d'entrées
- `currentStreak()` - Jours consécutifs d'écriture
- `moodDistribution()` - Répartition des humeurs

PhotoProcessor - Pipeline

```
UIImage → [compress] → [generateThumbnail] → [upload?]
                                                  ↓
                                            ProcessedPhoto
```

Méthodes :

- `process(_ image: UIImage) async throws -> ProcessedPhoto`
- `processMultiple(_ images: [UIImage]) async throws -> [ProcessedPhoto]`
- `generateThumbnail(size: CGSize) async -> Data`
- `compress(quality: CGFloat) async -> Data`

Performance : *TaskGroup pour traiter plusieurs photos en parallèle*

ExportService - Formats

Formats supportés :
- PDF (avec images et formatting)
- Markdown (texte pur, portable)
- JSON (backup complet)

Méthodes :

```swift
func exportToPDF(_ entries: [JournalEntry]) async throws -> URL
func exportToMarkdown(_ entries: [JournalEntry]) async throws -> URL
func exportToJSON(_ entries: [JournalEntry]) async throws -> URL
func exportAll(formats: [ExportFormat]) async throws -> [URL]
```

Ressource : PDFKit pour génération PDF

Ressources Backend - SwiftData

Documentation officielle :
- SwiftData Overview
- Modeling data with SwiftData

WWDC Sessions :
- WWDC23: Meet SwiftData (10187)
- WWDC23: Model your schema with SwiftData (10195)
- WWDC23: Dive deeper into SwiftData (10196)

Articles :
- SwiftData by Example

Ressources Backend - Concurrency

Swift Concurrency :
- Concurrency Documentation
- Actor Documentation

WWDC Sessions :
- WWDC21: Meet async/await in Swift (10132)
- WWDC21: Protect mutable state with Swift actors (10133)
- WWDC22: Eliminate data races using Swift Concurrency (110351)

Swift Evolution :
- SE-0296: async/await
- SE-0304: Structured concurrency
- SE-0306: Actors