MeetMind

Guide Architectural de Référence

Kickstart du Projet - 3 Parties

MeetMind : Vue d'Ensemble

Assistant de réunions intelligent avec Apple Intelligence

Une application professionnelle pour iOS qui transforme vos réunions en actions concrètes. Prise de notes intelligente, extraction automatique d'action items, et intégration native avec Calendar.

Features Principales

Core Features

- 📝 **Notes structurées** : Prise de notes pendant la réunion avec auto-save
- ✅ **Action items** : Extraction et suivi des tâches avec assignation
- 👥 **Participants** : Gestion des participants avec avatars et rôles
- 📅 **Calendar sync** : Synchronisation bidirectionnelle avec iOS Calendar
- ☁️ **iCloud sync** : Vos meetings sur tous vos appareils
- 🔒 **Privacy-first** : Données chiffrées, confidentialité garantie

Features Avancées (Jours 2 & 3)

Apple Intelligence (iOS 26+)

- 🤖 Action items extraction : Détection automatique depuis les notes
- 🎯 Meeting tone analysis : Analyse du ton (productif, neutre, tendu)

Architecture Globale

```
MeetMindApp
├── Data Layer (SwiftData) ← Jour 1
│   ├── @Model Meeting
│   ├── @Model ActionItem
│   ├── @Model Participant
│   └── ModelContainer
├── Domain Layer (Actors) ← Jour 1
│   ├── MeetingRepository
│   ├── CalendarSyncManager
│   ├── ShareService
│   └── ActionItemExtractor
├── Presentation Layer (SwiftUI) ← Jour 2
│   ├── Views (Timeline, Editor, Detail, Stats)
│   ├── Components (GlassCard, TypeSelector)
│   └── ViewModels (@Observable)
└── Intelligence Layer ← Jour 3
    ├── App Intents (Siri)
    ├── EventKit (Calendar sync)
    ├── NaturalLanguage (action items, tone)
    └── Webhooks (Slack, Teams)
```

Stack Technique Complète

```
SwiftUI + Swift 6
├── SwiftData (persistence) ← Jour 1
├── Actors (thread-safety) ← Jour 1
├── async/await (concurrency) ← Jour 1
├── EventKit (Calendar integration) ← 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
- Intégration calendrier native

Table des Matières

Partie 1

Backend Local de l'App

Stack Technique Jour 1

- SwiftData : Persistence moderne
- 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 Meeting
@Model class ActionItem
@Model class Participant

struct MeetingType: Codable, Sendable
struct RecurrenceRule: Codable, Sendable
enum Priority: String, Codable, Sendable
```

Ressources :

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

Meeting - Model Principal

```
@Model final class Meeting {
    var id: UUID
    var createdAt: Date
    var scheduledDate: Date
    var duration: TimeInterval

    var title: String
    var notes: String
    var type: MeetingType

    @Relationship(deleteRule: .cascade)
    var actionItems: [ActionItem]

    @Relationship(deleteRule: .nullify)
    var participants: [Participant]

    var isRecurring: Bool
    var recurrenceRule: RecurrenceRule?
    var calendarEventID: String?
}
```

MeetingType - Enum pour Types

```
enum MeetingType: String, Codable, Sendable {
    case standup, oneOnOne, review, planning, retrospective, other

    var displayName: String {
        /* "Stand-up", "1-on-1", "Review"... */
    }

    var icon: String {
        /* SF Symbol names */
    }

    var color: Color {
        /* .blue, .green, .orange... */
    }

    var defaultDuration: TimeInterval {
        /* 15min pour standup, 30min pour 1-on-1... */
    }
}
```

ActionItem - Model pour Actions

```
@Model final class ActionItem {
    var id: UUID
    var createdAt: Date
    var title: String
    var notes: String?

    var assignee: Participant?
    var dueDate: Date?
    var priority: Priority
    var isCompleted: Bool
    var completedAt: Date?

    var meeting: Meeting?
}

enum Priority: String, Codable, Sendable {
    case low, medium, high, urgent

    var color: Color { /* ... */ }
    var icon: String { /* ... */ }
}
```

Participant - Model pour Personnes

```
@Model final class Participant {
    var id: UUID
    var name: String
    var email: String?
    var role: String?
    var avatarData: Data?

    @Relationship(inverse: \Meeting.participants)
    var meetings: [Meeting]
}
```

RecurrenceRule - Struct pour Récurrence

```
struct RecurrenceRule: Codable, Sendable {
    enum Frequency: String, Codable {
        case daily, weekly, biweekly, monthly
    }

    var frequency: Frequency
    var endDate: Date?
    var occurrences: Int? // Ou nombre d'occurrences

    func nextOccurrence(after date: Date) -> Date? {
        // Calculer prochaine occurrence
    }
}
```

SwiftData Configuration

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

    private init() {
        let schema = Schema([
            Meeting.self,
            ActionItem.self,
            Participant.self
        ])

        let config = ModelConfiguration(
            cloudKitDatabase: .automatic
        )

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

Actors à Implémenter

```
actor MeetingRepository {
    func create(_ meeting: Meeting) async throws
    func fetchAll() async throws -> [Meeting]
    func fetchUpcoming() async throws -> [Meeting]
    func fetchPast() async throws -> [Meeting]
    func update(_ meeting: Meeting) async throws
    func delete(_ meeting: Meeting) async throws
}

actor CalendarSyncManager {
    func syncToCalendar(_ meeting: Meeting) async throws
    func syncFromCalendar() async throws -> [Meeting]
    func handleConflicts() async throws
}

actor ShareService {
    func exportToPDF(_ meeting: Meeting) async throws -> URL
    func sendEmail(_ meeting: Meeting, to: [String]) async throws
    func postToSlack(_ meeting: Meeting) async throws
}
```

MeetingRepository - Méthodes Clés

CRUD :

- `create()`, `update()`, `delete()`
- `fetchAll()`, `fetchUpcoming()`, `fetchPast()`

Filtres :

- `fetch(type: MeetingType)` - Par type
- `fetch(participant: Participant)` - Par participant
- `search(query: String)` - Full-text search

Stats :

- `count()` - Nombre total de meetings
- `totalDuration()` - Temps total en réunion
- `completionRate()` - % action items complétés

CalendarSyncManager - Integration

```
iOS Calendar ↔ CalendarSyncManager ↔ MeetingRepository
                          ↓
                  Conflict Resolution
```

Méthodes :

- `syncToCalendar(_ meeting: Meeting) async throws`
- `syncFromCalendar(dateRange: DateInterval) async throws`
- `resolveConflict(local: Meeting, remote: EKEvent) async throws`
- `subscribeToCalendarChanges() async`

Ressource : EventKit framework

ShareService - Multi-Canal

Formats d'export :
- PDF (notes formatées + action items)
- Email (résumé + attachement PDF)
- Slack/Teams (via webhooks)
- JSON (backup)

Méthodes :

```swift
func shareMultiple(
    meeting: Meeting,
    channels: [ShareChannel]
) async throws -> [ShareResult]
```

TaskGroup : Partage parallèle sur tous les canaux

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)

EventKit :
- EventKit Documentation
- WWDC23: Meet EventKit (10052)

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)