MeetMind

Guide Architectural de Référence

Kickstart du Projet - 3 Parties

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)

Partie 2

Liquid Glass UI

Design System - Palette de Couleurs

```swift
extension Color {
    // Gradients principaux (plus corporate)
    static let meetMindPrimary = Color(hex: "4F46E5") // Indigo
    static let meetMindSecondary = Color(hex: "7C3AED") // Purple

    // Glass effects
    static let glassBackground = Color.white.opacity(0.2)
    static let glassBorder = Color.white.opacity(0.6)

    // Meeting types
    static let standupColor = Color.blue
    static let oneOnOneColor = Color.green
    static let reviewColor = Color.orange
    static let planningColor = Color.purple

    // Priority
    static let priorityLow = Color.gray
    static let priorityMedium = Color.orange
    static let priorityHigh = Color.red
    static let priorityUrgent = Color(hex: "DC2626")
}
```

Design System - Typography

```
extension Font {
    // Headings
    static let meetingTitle = Font.system(
        size: 28, weight: .bold, design: .default
    )
    static let cardTitle = Font.system(
        size: 18, weight: .semibold
    )

    // Body
    static let meetingNotes = Font.system(
        size: 16, weight: .regular
    )
    static let actionItem = Font.system(
        size: 15, weight: .medium
    )

    // Special
    static let timestamp = Font.system(
        size: 12, weight: .regular, design: .monospaced
    )
}
```

Components à Créer

Core Components :

- `GlassCard` - Container de base
- `MeetingTypeSelector` - Picker de type
- `ParticipantPill` - Pills pour participants
- `ActionItemRow` - Row avec checkbox
- `DurationPicker` - Picker de durée

Complex Components :

- `MeetingCard` - Card pour timeline
- `NotesEditor` - Éditeur avec auto-save
- `ActionItemsList` - Liste avec completion
- `ParticipantPicker` - Sélection participants

GlassCard - Signature

```
struct GlassCard<Content: View>: View {
    let content: Content

    init(@ViewBuilder content: () -> Content)

    var body: some View {
        content
            .padding()
            .background(.ultraThinMaterial)
            .cornerRadius(20)
            .shadow(...)
            .overlay(borderGradient)
    }
}
```

MeetingTypeSelector - UI

```
[📱 Standup] [👥 1-on-1] [📊 Review] [📋 Planning] [🔄 Retro]
      ↑
Selection avec animation scale + color highlight
```

Features :

- Sélection tactile
- Animation spring au tap
- Couleur de fond selon type
- Durée par défaut appliquée

Views Architecture

```
MeetMindApp
└── ContentView
    ├── TimelineView (liste des meetings)
    │   └── MeetingCard (row)
    ├── EditorView (nouveau/édition meeting)
    │   ├── MeetingTypeSelector
    │   ├── TextEditor (notes)
    │   ├── ParticipantPicker
    │   └── ActionItemsList
    ├── DetailView (lecture meeting)
    │   ├── MeetingHeader
    │   ├── NotesContent
    │   └── ActionItemsList
    └── StatsView (analytics dashboard)
        └── MetricsCards
```

TimelineView - Structure

```
struct TimelineView: View {
    @State private var viewModel = TimelineViewModel()
    @Namespace private var animation

    var body: some View {
        ScrollView {
            LazyVStack(spacing: 20) {
                ForEach(viewModel.meetings) { meeting in
                    MeetingCard(meeting: meeting)
                        .matchedGeometryEffect(
                            id: meeting.id,
                            in: animation
                        )
                }
            }
        }
        .background(gradientBackground)
    }
}
```

EditorView - Features

Composants :
- `TextEditor` natif SwiftUI (notes)
- `MeetingTypeSelector` en header
- `DatePicker` pour date/heure
- `DurationPicker` pour durée
- `ParticipantPicker` (multi-select)
- `ActionItemsList` (ajout/édition)
- Toolbar glass en bas

Auto-save :
- Debouncing avec Task cancellation
- Sauvegarde toutes les 2 secondes
- Indicateur "Saved" subtil

Writing Tools : Intégration automatique iOS 18+

ActionItemsList - Component

```
struct ActionItemsList: View {
    @Binding var items: [ActionItem]

    var body: some View {
        VStack(spacing: 12) {
            ForEach($items) { $item in
                ActionItemRow(item: $item)
            }

            Button("Add Action Item") {
                // Ajouter nouvel item
            }
        }
    }
}

struct ActionItemRow: View {
    @Binding var item: ActionItem

    var body: some View {
        HStack {
            Button { item.isCompleted.toggle() } label: {
                Image(systemName: item.isCompleted ?
                    "checkmark.circle.fill" : "circle")
            }
            TextField("Action item", text: $item.title)
            PriorityBadge(priority: item.priority)
        }
    }
}
```

Animations - Patterns

Hero Animation :

```swift
@Namespace private var animation
.matchedGeometryEffect(id: meeting.id, in: animation)
```

Completion Animation :

```swift
withAnimation(.spring(response: 0.3)) {
    actionItem.isCompleted.toggle()
}
```

Card Flip :

```swift
.rotation3DEffect(
    .degrees(isFlipped ? 180 : 0),
    axis: (x: 0, y: 1, z: 0)
)
```

Calendar Integration UI

```
struct CalendarSyncBanner: View {
    @State private var isSyncing = false

    var body: some View {
        HStack {
            Image(systemName: "calendar.badge.clock")
            Text("Sync with Calendar")
            Spacer()
            if isSyncing {
                ProgressView()
            } else {
                Button("Sync") {
                    Task { await syncCalendar() }
                }
            }
        }
        .padding()
        .background(.thinMaterial)
        .cornerRadius(12)
    }
}
```

Ressources UI - SwiftUI

Documentation officielle :
- SwiftUI Documentation
- SwiftUI Tutorials

WWDC Sessions :
- WWDC23: Animate with springs (10158)
- WWDC23: Demystify SwiftUI performance (10160)
- WWDC22: SwiftUI on iPad: Add toolbars (10069)

Communauté :
- Hacking with Swift - SwiftUI
- SwiftUI Lab

Partie 3

Apple Intelligence & Data

Apple Intelligence - Stack

On-Device :

- Natural Language : Action items extraction, keywords
- EventKit : Calendar integration native
- Core ML : Models custom (optionnel)

iOS 18+ Features :

- App Intents : Siri, Shortcuts, Spotlight
- Writing Tools : Intégration automatique
- Focus Filters : Integration avec Focus modes

App Intents - Vue d'ensemble

Intents à créer :

```swift
struct CreateMeetingIntent: AppIntent
struct ShowUpcomingMeetingsIntent: AppIntent
struct AddActionItemIntent: AppIntent
struct CompleteMeetingIntent: AppIntent
```

Usage :

- "Hey Siri, create a standup meeting for tomorrow"
- "Show my upcoming meetings in MeetMind"
- "Add action item: Follow up with John"

Ressource :

- App Intents Documentation
- WWDC22: Dive into App Intents (10032)

CreateMeetingIntent - Structure

```
struct CreateMeetingIntent: AppIntent {
    static var title: LocalizedStringResource = "Create Meeting"
    static var description = IntentDescription("Schedule a new meeting")

    @Parameter(title: "Title")
    var title: String

    @Parameter(title: "Type")
    var type: MeetingTypeEntity?

    @Parameter(title: "Date")
    var date: Date?

    @Parameter(title: "Participants")
    var participants: [ParticipantEntity]?

    func perform() async throws -> some IntentResult {
        // Créer meeting via MeetingRepository
        // Optionnel : sync avec Calendar
    }
}
```

Entity Types - AppEntity

```
struct MeetingEntity: AppEntity {
    static var typeDisplayRepresentation =
        TypeDisplayRepresentation(name: "Meeting")

    var id: UUID
    var title: String
    var date: Date
    var type: MeetingType

    var displayRepresentation: DisplayRepresentation {
        DisplayRepresentation(
            title: "\(title)",
            subtitle: "\(date.formatted())"
        )
    }
}

struct ParticipantEntity: AppEntity {
    var id: UUID
    var name: String
    var email: String?
}
```

Natural Language - Action Items Extraction

```
actor ActionItemExtractor {
    private let tagger = NLTagger(
        tagSchemes: [.lexicalClass, .nameType]
    )

    func extract(from text: String) async -> [String] {
        var actionItems: [String] = []

        // Détecter patterns d'action items
        let patterns = [
            "TODO:", "Action:", "Follow up",
            "\\w+ needs to", "\\w+ should", "\\w+ will"
        ]

        for pattern in patterns {
            let regex = try? NSRegularExpression(pattern: pattern)
            // Extraire phrases contenant ces patterns
        }

        return actionItems
    }
}
```

Natural Language - Meeting Tone Analysis

```
actor MeetingToneAnalyzer {
    private let tagger = NLTagger(tagSchemes: [.sentimentScore])

    func analyze(_ notes: String) async -> MeetingTone {
        tagger.string = notes

        let (tag, _) = tagger.tag(
            at: notes.startIndex,
            unit: .paragraph,
            scheme: .sentimentScore
        )

        guard let score = tag?.rawValue
            .flatMap(Double.init) else {
            return .neutral
        }

        return MeetingTone.from(score: score)
    }
}

enum MeetingTone {
    case productive, neutral, tense
}
```

Writing Tools - iOS 18

Intégration automatique :

```
swift
TextEditor(text: $meeting.notes)
    // Writing Tools s'activent automatiquement !
```

Use cases :

- Reformuler notes de manière plus professionnelle
- Résumer une longue réunion
- Extraire les points clés
- Créer une table des décisions

Ressource :

- WWDC24: Meet Writing Tools (10168)

EventKit - Calendar Sync

```
import EventKit

actor CalendarService {
    private let store = EKEventStore()

    func requestAccess() async throws -> Bool {
        try await store.requestFullAccessToEvents()
    }

    func createEvent(
        for meeting: Meeting
    ) async throws -> EKEvent {
        let event = EKEvent(eventStore: store)
        event.title = meeting.title
        event.startDate = meeting.scheduledDate
        event.endDate = meeting.scheduledDate
            .addingTimeInterval(meeting.duration)
        event.calendar = store.defaultCalendarForNewEvents

        try store.save(event, span: .thisEvent)
        return event
    }

    func fetchEvents(
        in dateRange: DateInterval
    ) async throws -> [EKEvent] {
        // Fetch depuis Calendar
    }
}
```

Smart Features à Implémenter

Smart Suggestions :
- "Vous avez 3 meetings aujourd'hui, total: 2h30"
- "5 action items en attente depuis 7+ jours"
- "Votre taux de complétion ce mois : 87%"

Meeting Insights :
- Durée moyenne par type de meeting
- Participants les plus fréquents
- Patterns de récurrence

Auto-completion :
- Suggestions de participants basées sur historique
- Suggestions de durée basées sur type
- Templates pour meetings récurrents

Analytics & Stats - Métriques

Time Tracking :
- Temps total en réunion (jour/semaine/mois)
- Breakdown par type de meeting
- Tendances dans le temps

Productivity :
- Action items créés vs complétés
- Taux de complétion par participant
- Délai moyen de complétion

Meetings :
- Nombre de meetings (par type, participant)
- Durée moyenne vs planifiée
- Meetings récurrents vs one-off

Swift Charts - Dashboard

```swift
import Charts

struct MeetingTimeChart: View {
    let meetings: [Meeting]

    var body: some View {
        Chart {
            ForEach(meetings) { meeting in
                BarMark(
                    x: .value("Type", meeting.type.displayName),
                    y: .value("Duration", meeting.duration / 3600)
                )
                .foregroundStyle(meeting.type.color)
            }
        }
        .chartYAxis {
            AxisMarks(values: .automatic) { value in
                AxisValueLabel {
                    Text("\(value.as(Double.self) ?? 0, format: .number)h")
                }
            }
        }
    }
}
```

Focus Filters - iOS 16+

```
struct MeetingFocusFilter: FocusFilterProvider {
    func filter(
        _ meetings: [Meeting],
        in focusMode: FocusMode
    ) -> [Meeting] {
        switch focusMode {
        case .work:
            return meetings.filter {
                $0.type != .oneOnOne
            }
        case .personal:
            return []
        case .doNotDisturb:
            return meetings.filter {
                $0.scheduledDate > Date().addingTimeInterval(3600)
            }
        default:
            return meetings
        }
    }
}
```

Webhooks - Slack/Teams Integration

```
actor WebhookService {
    func postToSlack(
        meeting: Meeting,
        webhookURL: URL
    ) async throws {
        let payload: [String: Any] = [
            "text": "📝 Meeting Notes: \(meeting.title)",
            "blocks": [
                [
                    "type": "section",
                    "text": ["type": "mrkdwn",
                             "text": meeting.notes]
                ],
                [
                    "type": "section",
                    "text": ["type": "mrkdwn",
                             "text": formatActionItems(meeting.actionItems)]
                ]
            ]
        ]

        // POST to webhook
    }
}
```

Privacy - Considérations

Données professionnelles :
- Notes de réunions = confidentielles
- Chiffrement automatique (SwiftData)
- Option : chiffrement bout-en-bout supplémentaire

Calendar Permissions :
- Demander accès Calendar explicitement
- Expliquer pourquoi (sync bidirectionnelle)
- Permettre sync one-way si préféré

Partage :
- Demander confirmation avant email/Slack
- Option d'anonymiser participants
- Logs d'export pour audit

Ressources Apple Intelligence

App Intents :
- App Intents Documentation
- WWDC22: Dive into App Intents (10032)
- WWDC23: Explore enhancements to App Intents (10103)

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

Natural Language :
- NaturalLanguage Documentation

Ressources - Productivity

Documentation :

- Focus Filters
- Webhooks Best Practices

WWDC Sessions :

- WWDC22: Focus on productivity (10109)
- WWDC23: Discover Calendar and EventKit (10052)

Features Optionnelles (Bonus)

Transcription (iOS 17+) :
- Speech framework pour transcrire audio
- Notes automatiques depuis enregistrement

Visual Intelligence :
- Scanner notes manuscrites → digital
- OCR avec Vision framework

Notifications :
- Rappels avant meetings
- Rappels pour action items non complétés
- Daily/weekly summaries

iCloud Collaboration :
- Notes partagées en temps réel
- Commentaires sur action items

Architecture Complète - Recap

```
MeetMindApp
├── Data Layer (SwiftData)
│   ├── Models (@Model classes)
│   └── ModelContainer
├── Domain Layer (Actors)
│   ├── MeetingRepository
│   ├── CalendarSyncManager
│   └── ShareService
├── UI Layer (SwiftUI)
│   ├── Views (Timeline, Editor, Detail, Stats)
│   ├── Components (GlassCard, TypeSelector, etc.)
│   └── ViewModels (@Observable)
└── Intelligence Layer
    ├── App Intents
    ├── NaturalLanguage (Action items, tone)
    ├── EventKit Integration
    └── Analytics
```

Checklist de Démarrage

Jour 1 :
- [ ] Créer projet Xcode avec SwiftData
- [ ] Définir Models (@Model)
- [ ] Setup ModelContainer
- [ ] Implémenter MeetingRepository actor
- [ ] Implémenter CalendarSyncManager actor
- [ ] Tests CRUD basiques

Jour 2 :
- [ ] Créer design system (Colors, Fonts)
- [ ] Implémenter GlassCard
- [ ] Créer TimelineView
- [ ] Créer EditorView
- [ ] ActionItemsList component

Checklist de Démarrage (suite)

Jour 3 :
- [ ] Créer App Intents (Create, Show)
- [ ] Implémenter ActionItemExtractor
- [ ] Implémenter MeetingToneAnalyzer
- [ ] Tester Writing Tools
- [ ] EventKit integration
- [ ] StatsView avec Charts

Finitions :
- [ ] App Icon & Launch Screen
- [ ] Dark mode support
- [ ] Accessibility labels
- [ ] Privacy policy

Points d'Attention

Performance :
- LazyVStack pour longues listes de meetings
- Pagination si 100+ meetings
- Cache pour calendar events

UX :
- Auto-save toujours (jamais perdre des notes)
- Feedback visuel pour sync calendar
- Offline mode (queue sync operations)
- Gestion des conflits de calendrier

Code Quality :
- Actors pour thread-safety
- async/await partout
- Sendable pour types partagés
- Tests pour logique critique

Équivalence avec Mindful

Feature	Mindful	MeetMind
Core Model	JournalEntry	Meeting
Categories	Mood (5 types)	MeetingType (6 types)
Sub-items	Photos	ActionItems
Sync	iCloud	Calendar + iCloud
NLP	Sentiment	Action items extraction
Export	PDF/MD/JSON	PDF/Email/Slack
Stats	Mood trends	Time tracking

Ressources Générales

Documentation Apple :

- Apple Developer Documentation
- EventKit Guide

Communauté :

- Swift Forums
- r/iOSProgramming
- Stack Overflow

Prochaines Étapes

Après l'atelier :

1. Continuer le développement de MeetMind
2. Beta testing avec collègues
3. Intégrer feedback
4. Soumission App Store

Apprentissage continu :

- Regarder WWDC sessions manquées
- Lire EventKit documentation
- Tester les APIs de webhooks
- Participer à des meetups iOS

Questions ?

Références Rapides

SwiftData : developer.apple.com/swiftdata

EventKit : developer.apple.com/eventkit

App Intents : developer.apple.com/app-intents

Natural Language : developer.apple.com/naturallanguage

WWDC : developer.apple.com/videos

Bon Courage ! 🚀

Ready to build MeetMind?

Remember :

- Start simple, iterate often
- Test on real devices
- Privacy first
- Sync is critical