Core Data in iOS

Understanding Data Persistence

What is CoreData?

1. Not Just Database - Complete object graph and persistence framework
2. Cross Platform - Works on iOS, macOS, watchOS, and tvOS
3. Features - Data Modeling, relationships, validations
4. Integration - Seamless UIKit and SwiftUI

Core Data Stack

Flow of Data

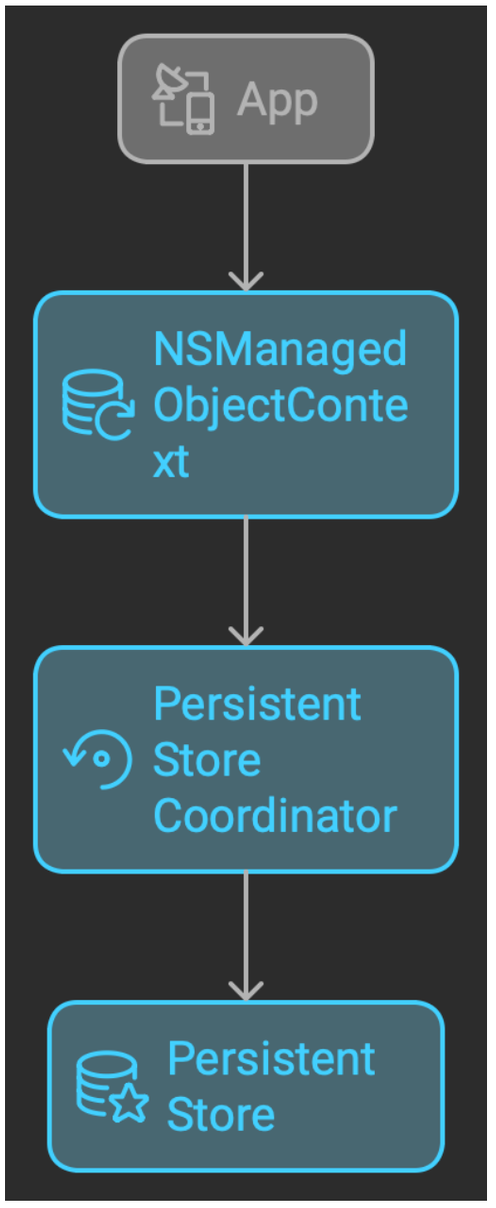
1. App reacts with

   NSManagedObjectContext

2. Context requests data through

   coordinator

3. Coordinator fetches from store



Core Data Stack Components

NSManagedObjectContext

1. Temporary scratch pad for objects

2. Track changes to objects

3. Handles undo/redo operations

4. Validates changes before saving.

Core Data Stack Components

Persistent Store Coordinator

1. Mediates between context and store
2. Handles data fetching and saving
3. Manages multiple stores if needed
4. Coordinates migrations

Core Data Stack Components

Persistent Store

1. Actual storage on disk (SQLite)
2. Handles data persistence
3. Manages file format
4. Supports different store types

Core Data Stack Components

SQLite Store

Type: NSSQLiteStoreType

Characteristics: Default option, efficient for large data sets, supports migrations.

Use Case: Most iOS apps, especially those with substantial data or complex relationships.

Binary Store

Type: NSBinaryStoreType
Characteristics: Entire store is loaded into memory, faster for smaller datasets.
Use Case: Apps with smaller data sets that need quick access and don't require incremental saving.

In-Memory Store

Type: NSInMemoryStoreType
Characteristics: Data exists only in RAM, lost when app terminates.
Use Case: Temporary data storage, caching, unit testing

4. XML Store (macOS only)

Type: NSXMLStoreType
Characteristics: Data stored in XML format, human-readable.
Use Case: macOS apps where data interoperability or human-readability is important.

```
let container = NSPersistentContainer(name: "MyDataModel")
let storeDescription = container.persistentStoreDescriptions.first
storeDescription?.type = NSSQLiteStoreType // or other store types
container.loadPersistentStores { (storeDescription, error) in
    // Handle errors
}
```

Setting Up Core Data

```
struct PersistenceController {
    static let shared = PersistenceController()
    let container: NSPersistentContainer

    init(inMemory: Bool = false) {
        container = NSPersistentContainer(name: "PersonalJournal")

        if inMemory {
            container.persistentStoreDescriptions.first!.url =
                URL(fileURLWithPath: "/dev/null")
        }

        container.loadPersistentStores { description, error in
            if let error = error {
                fatalError("Error: (error.localizedDescription)")
            }
        }
    }
}
```

Fetching Data: Two Approaches

@FetchRequest

```
struct JournalListView: View {
    @FetchRequest(
        sortDescriptors: [
            NSSortDescriptor(
                keyPath: \JournalEntry.date,
                ascending: false
            )
        ]
    ) var entries: FetchedResults<JournalEntry>
}
```

@NSFetchRequest

```
func fetchEntries() {
    let request = JournalEntry.fetchRequest()
    request.sortDescriptors = [
        NSSortDescriptor(
            keyPath: \JournalEntry.date,
            ascending: false
        )
    ]
    entries = try? viewContext.fetch(request)
}
```

CRUD Operations

Create

```
let entry = JournalEntry(context: viewContext)
entry.title = "New Entry"
try? viewContext.save()
```

Update

```
entry.title = "Updated Title"
try? viewContext.save()
```

Delete

```
viewContext.delete(entry)
try? viewContext.save()
```