Technical Document for OfflineNotesApp

1. Overview

OfflineNotesApp is an iOS application designed for managing notes with offline capabilities. It supports user authentication (login/signup), note creation, viewing, and detail display with photos. The app uses Core Data for local persistence and follows Clean Architecture principles combined with MVVM for presentation.

---

2. Features

- User authentication: Signup and Login
- Create, read, and display notes with photos
- Offline storage using Core Data
- Reactive UI using SwiftUI and MVVM
- Modularized codebase supporting scalability and maintainability

---

3. Architecture

The app uses **Clean Architecture + MVVM**, split into several layers:

3.1. Layers

Layer	Responsibility	Folder
Presentation	SwiftUI Views + ViewModels that bind data to UI	`Presentation/Views` and `ViewModels`
Domain	Business logic, Use Cases, and Entities	`Domain/UseCase` and `Entities`

Data	Data sources, Repositories, Mappers between Core Data Entities and Domain Entities	`Data/Repository` and `ModelEntityMappers`
Core	Core utilities including Dependency Injection, Core Data stack, Router, Utilities like hashing and photo picker	`Core/`

4. Core Components

4.1 Entities (Domain/Entities)

- `NoteEntity.swift`: Defines the Note domain model with attributes like title, content, date, and associated photos.
- `UserEntity.swift`: Defines the User domain model for authentication.
- `PhotoEntity.swift`: Represents photos attached to notes.

4.2 Use Cases (Domain/UseCase)

- `LoginUseCase.swift`: Handles login logic.
- `SignupUseCase.swift`: Handles user registration.
- `FetchNotesUseCase.swift`: Retrieves saved notes.
- `AddNoteUseCase.swift`: Adds a new note with photos.

4.3 Repositories (Data/Repository)

- Abstract the data access logic, interacting with Core Data storage.

4.4 Model Mappers (Data/ModelEntityMappers)

- Translate Core Data entities to domain entities and vice versa.

4.5 Presentation Layer

- SwiftUI views like `LoginView`, `SignupView`, `HomeView` (notes list), `AddNoteView`, and `NoteDetailView`.
- ViewModels for each screen handle UI logic and call Use Cases.
- Shared UI components and reusable controls (buttons, text fields, etc.)

4.6 Core Layer

- **DIContainer.swift**: Dependency Injection container for managing service instantiations.
- **CoreDataStack.swift**: Core Data setup and management.
- Utilities for photo picking, validation, hashing, and routing.

---

5. Workflow / Functional Flow

1. **User Authentication**
   - On app launch, the user can log in or sign up.
   - Login and Signup ViewModels communicate with respective use cases.
2. **Notes Management**
   - Once authenticated, the user lands on the Home screen displaying notes.
   - Notes are fetched via `FetchNotesUseCase` and displayed in a list.
   - The user can add a new note with photos using `AddNoteView`.
   - New notes are saved locally using Core Data.
   - Notes details can be viewed with photos.
3. **Offline Capability**
   - All notes and user info are stored locally with Core Data.

    ○ The app functions without internet connectivity.

---

6. Functional Diagram

Here is a high-level functional diagram representing the architecture and data flow:

plaintext
CopyEdit

```
+-------------------+         +-----------------+         +------------------+
|    Presentation   |  <----> |    Domain       |  <----> |      Data        |
| (SwiftUI Views +  |         | (Use Cases &    |         | (Repositories &  |
|  ViewModels)      |         |  Entities)      |         |  Mappers)        |
+-------------------+         +-----------------+         +------------------+
        |                            |                            |
        v                            v                            v
  User Interaction            Business Logic &             Core Data Storage
     & UI updates               Application Rules           & Persistence
```

- **Core** utilities (DI container, CoreData stack, Utilities) support all layers and are injected where needed.

---

7. Summary

OfflineNotesApp is a robust, offline-first note-taking iOS application structured with modern architecture practices (Clean Architecture + MVVM). It is designed for scalability, maintainability, and testability with a clear separation of UI, business logic, and data access layers. The use of Core Data ensures persistent storage and offline usage.