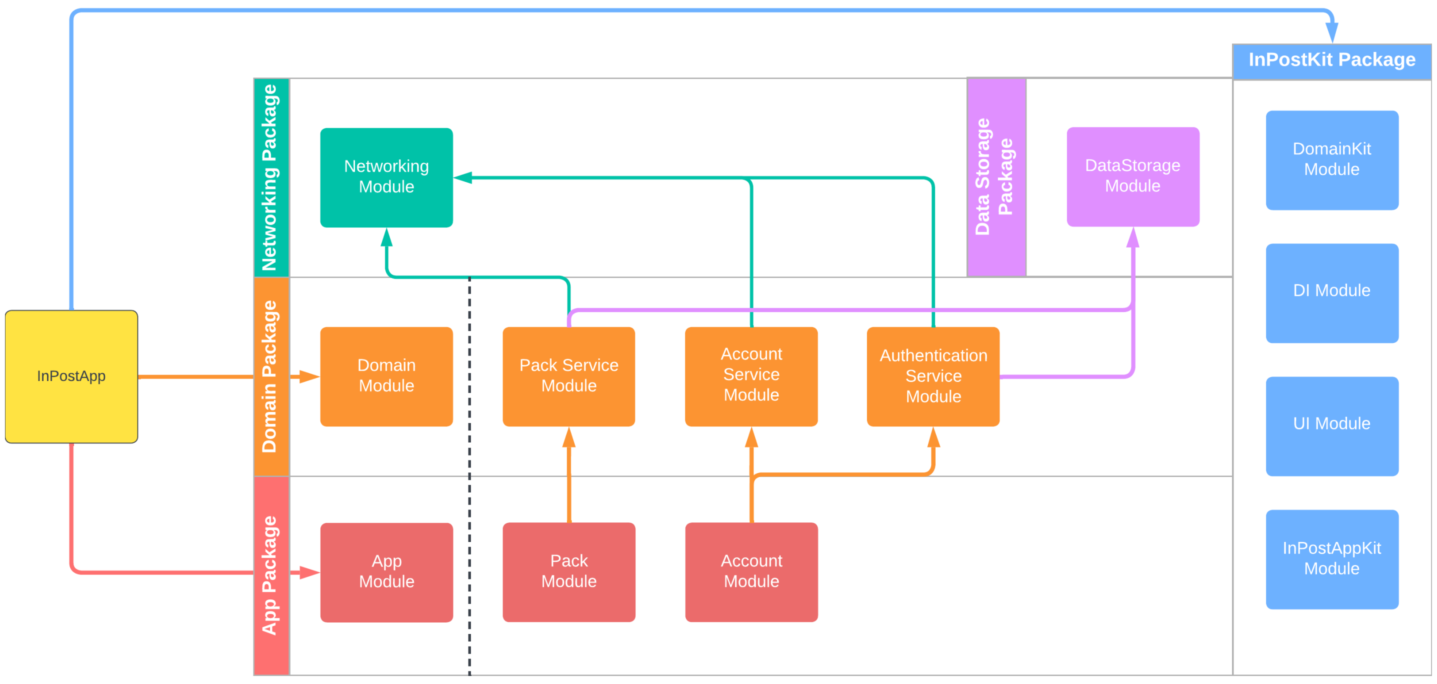
iOS App Architecture Documentation

Disclaimer:
The documentation contains references to “Account Module,” “AccountService Module,” and “AuthenticationService Module” for illustrative purposes. Please note that these modules are not currently implemented in the app but are included as examples to help illustrate the architectural concepts.

Introduction

This document provides an architectural overview of our iOS app, highlighting its package structure and modules. The app relies on Swift Package Manager (SPM) to manage dependencies and is organized into two main packages: “App” and “Domain,” along with supportable packages “InPostKit” and “Networking.”

Graph



App Package

The “App” package includes three types of modules:

Supportable Modules

These modules provide support and core functionality.

Localizable Module

- Manages localization resources to support a multilingual experience.

Main

This module is responsible for the main entry point of the Package.

App Module

- Serves as the entry point for the application.
- Responsible for app setup and configuration.

Feature Modules

These modules encapsulate specific features of the application.

Pack Module

- Including screens such as Pack List or Pack Details.

Account Module

- Including screens such as Login and Account

Domain Package

Domain Packages are responsible for defining data models (DTOs) and implementing business logic.

The “Domain” package is structured into two types of modules:

Main Modules

Domain Module

- The core of the “Domain” package, this module encapsulates the main domain models, entities, and business logic used throughout the application.
- It defines the essential data structures and rules that govern the behavior of our app, ensuring that our business logic is decoupled from UI concerns.

Service Modules

PackService

- PackService is responsible for managing DTOs related to Pack data.

AccountService

- TBD

AuthenticationService

- TBD

InPostKit Package

The “InPostKit” package is a modular component that can be employed not only in our app but also in other applications. It offers a collection of modules designed to enhance functionality and streamline integration into various projects.

Key Modules:

DI (Dependency Injection) Module: “InPostKit” incorporates dependency injection (DI) module. This module is enhanced by the use of the third-party library **SwinjectAutoregister**, which automates the registration of dependencies.

UI Module: The package includes a set of pre-built UI components and fonts.

DomainKit Module: “InPostKit” features “DomainKit,” a module designed to support and assist the “Domain Modules” in our app.

AppKit Module: “InPostKit” features “AppKit,” a module designed to support and assist the “App Modules” in our app.

Reusability:

The primary strength of the “InPostKit” package is its modularity and reusability. By providing a collection of versatile modules, it can be leveraged in other applications to expedite development and maintain consistency in UI and functionality.

Networking Package

The “Networking” package is essential for our app’s communication with external APIs and services. At its core is the “Networking Module,” which serves as our app’s gateway to the digital world.

Networking Module

The “Networking Module” manages HTTP requests and responses, making it possible for our app to interact with external APIs. It handles tasks such as request building, endpoint management, response parsing, and error handling.

A central component of the “Networking Module” is the “APIClient,” which encapsulates API-related functionality. It streamlines the process of making network requests, ensuring secure and efficient data exchange.

DataStorage Package

The “DataStorage” package is responsible for managing the storage of data.

DataStorage Module

Currently, it utilizes UserDefaults for data storage, providing a solution for storing user preferences and application data.

Notice: In the future, it may containg data storage solutions such as CoreData or Realm.