# ‘StarWars’ Application Design Documentation

## Introduction

This document provides a comprehensive overview of the design decisions, architectural choices, trade-offs, and challenges encountered during the development of the SwiftUI application. The goal was to create an engaging and efficient app that showcases planet details while leveraging SwiftUI’s modern capabilities.

## Architectural & Design Decisions

##### Adoption of the MVVM Pattern

##### **Decision**: The Model-View-ViewModel (MVVM) pattern was chosen as the architectural framework.

##### MVVM is well-suited for SwiftUI applications due to its alignment with SwiftUI’s state-driven UI updates. It separates concerns, allowing:

##### Model: Handles data representation and business logic.

##### View: Defines the UI layout and presentation.

* ViewModel: Acts as an intermediary between the Model and View, managing state and business logic.

MVVM promotes cleaner code organization and makes it easier for multiple developers to work on different parts of the application concurrently.

**Drawbacks**: MVVM can lead to increased complexity, especially for small projects. Implementing MVVM requires creating additional layers (Models, Views, and ViewModels), which can introduce more classes and interfaces than necessary for simple applications.

##### Choosing SwiftUI Over UIKit

**Decision**: Opted to use SwiftUI instead of UIKit for building the application.

SwiftUI offers a modern, declarative approach to building user interfaces, which aligns well with contemporary iOS development practices.

Some benefits are seamless integration with Swift’s state management, promoting reactive programming paradigms and aligning with Apple’s strategic direction for UI frameworks, ensuring long-term support and updates.

**Drawbacks**: Debugging in SwiftUI is more challenging compared to UIKit. The error messages can sometimes be less informative, and it can make it harder to pinpoint the exact source of a problem

##### Swift Concurrency Usage

**Decision**: Swift Concurrency was used to handle asynchronous operations, particularly for network requests.

 Swift Concurrency (using async/await) provides a modern, structured way to manage asynchronous tasks, ensuring more readable and maintainable code.

**Drawback**: Requires iOS 15+ and Swift 5 +, limiting compatibility with older systems.

##### Singleton Pattern

**Decision**: Use of Singleton patterns for CoreDataManager, ImageLoader, NetworkManager etc.

Using singleton pattern simplifies access to shared resources and ensures a single instance is used across the application.

**Drawback**: Classes that use singletons are tightly coupled to them, making it difficult to change the singleton’s implementation without affecting all dependent classes.

##### Choosing SwiftUI Navigation View over Coordinator Pattern

**Decision**: Transitioned to NavigationView and NavigationLink for navigation

Initially tried using coordinator pattern to manage navigation. Managing navigation state manually introduced complexity, particularly with maintaining a consistent back navigation experience. By using NavigationView  managed to leverage built-in navigation features, including the back button and swipe-to-go-back gestures, to align with native iOS app experiences.

**Drawback**: SwiftUI’s navigation system can be less flexible than the Coordinator pattern. In SwiftUI, navigation is often tied to view hierarchies and state, which can limit your ability to perform complex navigation scenarios such as conditional navigation.

##### CoreData over Realm for offline capability

**Decision**: Used CoreData to persist planet data locally.

Tried using Realm for data persistency and faced a dependency related issue. Hence used CoreData Eventhough the usage of CoreData adds more overhead and complexity.

**Drawback**: Adds overhead and complexity, especially for simple data storage needs.

##### Image Caching

**Decision**: Implement image caching using local file storage.

This Reduced network usage and speeds up image loading by caching downloaded images.

**Drawback**: Requires managing file storage and cache clearing to prevent excessive storage use.