# Fueled Movies: A Sample iOS Application

A screenshot of a movie

Description automatically generatedA screenshot of a video game

Description automatically generated

## Overview

* This sample application leverages The Movie Database (TMDB) API to fetch a list of movie titles along with their corresponding images. The movies are displayed in a grid layout, and selecting a movie navigates the user to a detailed screen with additional information about the selected title.

The FueledMovies application demonstrates the following key concepts:

* UI Composition: Built using SwiftUI.
* Data Persistence: Utilizes CoreData to manage persistent storage, caching configuration data and movie images after they are retrieved.
* Networking Integration: Implements REST API communication with a custom AsyncSession class using Swift concurrency.

**Application Overview**

1. **App Initialization**

* Main application file **FueledMoviesApp.swift)**
* Manages the application startup.
* Downloads necessary configuration for API requests.
* Implements a LoadingState to handle UI transitions (loading, completed, error).

1. **Movie Content**

* Sets up the navigation stack. **(MovieContentView.swift)**
* Creates and displays a collection of movies. This is encapsulated by MovieListView (MoviesView.swift)
* Each movie is presented by MovieItemView (MovieItemView.swift). A movie item is represented by a movie poster imaged and a title.
* Interacting with a MovieItem will navigate the user to the MovieDetailView (MovieDetailView.swift). Navigation is achieved by pushing a view onto the navigation stack

1. **Networking**

* Sets up and provides ability to make REST requests **AsyncSession.swift**
* Provides a framework for making GET requests and downloading binary data.
* Utilizes Swift concurrency (async/await) for clean, asynchronous operations.
* Offers extensibility for additional HTTP methods (e.g., POST, DELETE).
* Includes customizable session configurations and error handling.

1. **Service Layer**

* Services provide functionality related to the domain they are abstracting.
* ConfigurationService (ConfigurationService.swift) Requests configuration data from TMDB API and persists it to Core Data to be retrieved as needed
* MovieService (MovieService.swift) Provides functionality to request a list of movies and download and cache poster images for a movie.

1. **Repository Layer**

* Repositories provide access to the data stored in Core Data, handling the conversion of model objects to and from Core Data entities. A key design objective was to abstract the underlying storage implementation, ensuring it remains hidden from higher layers.
* ConfigurationRepository (ConfigurationRepository.swift) provides operations to add and fetch the current configuration from Core Data.
* ImageReposity (ImageRespository.swift) provides operation to add and fetch image data from Core Data

## What to Look For

* MVVM Architecture: The project follows the Model-View-ViewModel (MVVM) design pattern, ensuring a clear separation of concerns. The ViewModel layer handles business logic and state management, making the SwiftUI views simpler and more focused on UI composition.
* SwiftUI Integration: The app demonstrates seamless integration of SwiftUI with MVVM, leveraging features like data binding and environment injection to create a reactive and maintainable UI.
* Service Layer (Domains) and ViewModel Synergy: The ViewModel interacts with Services, ensuring efficient caching and retrieval of configuration and movie data.