**Тестирование и отладка приложений на TCA**

Тестирование является важной частью разработки мобильных приложений, особенно в архитектуре TCA. TCA ориентирована на предоставление модульности и изолированности компонентов, что делает тестирование каждого компонента проще и более эффективным. Тесты помогают:

• Проверить корректность работы редьюсеров, эффектов и состояния.

• Убедиться, что состояние и действия правильно взаимодействуют друг с другом.

• Легко моделировать поведение разных частей системы без необходимости запускать приложение полностью.

TCA предоставляет все необходимые инструменты для написания юнит-тестов, интеграционных тестов и тестов пользовательского интерфейса, делая процесс тестирования простым и удобным.

**2. Основы тестирования в TCA**

Тестирование в TCA в основном делится на тестирование редьюсеров (и их логики), тестирование эффектов (асинхронных операций) и тестирование взаимодействий между состоянием, действиями и эффектами. Основные компоненты TCA, которые нужно тестировать:

• **State** — правильность обновления состояния.

• **Actions** — обработка действий и их преобразование в новые состояния.

• **Reducer** — бизнес-логика приложения, изменение состояния на основе действий.

• **Effects** — асинхронные операции и их правильное выполнение.

Для тестирования TCA удобно использовать инструмент **TestStore**, который позволяет тестировать редьюсеры, действия и эффекты в изоляции, используя заранее подготовленные данные и зависимости.

**3. Тестирование редьюсеров**

Основной инструмент для тестирования редьюсеров в TCA — это **TestStore**, который помогает имитировать различные действия и проверять, как изменяется состояние в ответ на эти действия.

**Пример юнит-теста для редьюсера:**

import XCTest

import ComposableArchitecture

class AppTests: XCTestCase {

    func testAddTask() {

        let store = TestStore(

            initialState: TaskListState(tasks: []),

            reducer: taskListReducer,

            environment: TaskListEnvironment()

        )

        store.send(.addTask(Task(id: 1, name: "New Task"))) {

            $0.tasks = [Task(id: 1, name: "New Task")]

        }

    }

    func testRemoveTask() {

        let store = TestStore(

            initialState: TaskListState(tasks: [Task(id: 1, name: "Old Task")]),

            reducer: taskListReducer,

            environment: TaskListEnvironment()

        )

        store.send(.removeTask(Task(id: 1, name: "Old Task"))) {

            $0.tasks = []

        }

    }

}

В этом примере:

• Мы создаем TestStore, который инициирует начальное состояние.

• В методе send посылаем действие (например, добавление или удаление задачи).

• В блоке expect указываем, как должно измениться состояние после выполнения действия.

**Тестирование редьюсеров с ошибками:**

Редьюсеры могут возвращать ошибки или результаты с ошибками (например, при асинхронных запросах). Для тестирования этих случаев можно использовать подходы, которые имитируют ошибки:

func testNetworkError() {

    let store = TestStore(

        initialState: AppState(),

        reducer: appReducer,

        environment: AppEnvironment(

            networkClient: MockNetworkClient(error: NSError(domain: "", code: 0, userInfo: nil)),

            mainQueue: DispatchQueue.main.eraseToAnyScheduler()

        )

    )

    store.send(.fetchData) {

        $0.data = "Error"

    }

}

Здесь:

• Используется мок-сервис, который возвращает ошибку.

• Мы проверяем, что при ошибке приложение отобразит состояние ошибки.

**4. Тестирование эффектов**

TCA позволяет тестировать асинхронные операции с помощью Effect. Асинхронные операции можно тестировать, моделируя их результаты с помощью тестовых эффектов.

**Пример теста для асинхронного эффекта:**

func testFetchDataSuccess() {

    let environment = AppEnvironment(

        networkClient: MockNetworkClient(data: "Mocked Data"),

        mainQueue: DispatchQueue.main.eraseToAnyScheduler()

    )

    let store = TestStore(

        initialState: AppState(),

        reducer: appReducer,

        environment: environment

    )

    store.send(.fetchData)

    store.receive(.dataLoaded("Mocked Data")) {

        $0.data = "Mocked Data"

    }

}

Здесь:

• Используется мок для сетевого клиента, который имитирует успешный ответ.

• Мы проверяем, что при вызове действия fetchData приложение корректно обрабатывает асинхронный ответ.

**Тестирование ошибок в эффектах:**

Можно также тестировать ошибки, которые могут возникнуть при асинхронных запросах.

func testFetchDataFailure() {

    let environment = AppEnvironment(

        networkClient: MockNetworkClient(error: NSError(domain: "", code: 0, userInfo: nil)),

        mainQueue: DispatchQueue.main.eraseToAnyScheduler()

    )

    let store = TestStore(

        initialState: AppState(),

        reducer: appReducer,

        environment: environment

    )

    store.send(.fetchData)

    store.receive(.dataFailed(NSError(domain: "", code: 0, userInfo: nil))) {

        $0.data = "Error"

    }

}

Здесь:

• Мок-сервис возвращает ошибку.

• Мы проверяем, что при ошибке будет установлено состояние с сообщением об ошибке.

**5. Тестирование пользовательского интерфейса (UI) с Snapshot-тестами**

Тестирование пользовательского интерфейса в TCA можно выполнить с помощью **Snapshot-тестов**. Эти тесты делают “снимок” интерфейса на разных стадиях работы приложения и сравнивают его с эталонным снимком.

Пример создания Snapshot-теста с использованием библиотеки SnapshotTesting:

import SnapshotTesting

import SwiftUI

func testMainViewSnapshot() {

    let view = MainView(store: store)

    assertSnapshot(matching: view, as: .image)

}

Здесь:

• Мы создаем представление (например, MainView) с переданным состоянием из TestStore.

• Используем assertSnapshot для создания снимка текущего состояния UI и его сравнения с эталонным изображением.

Snapshot-тесты полезны для проверки корректности UI, особенно в приложениях с динамическим изменением интерфейса.

**6. Инструменты для отладки в TCA**

TCA предоставляет несколько инструментов для отладки, которые упрощают процесс поиска и устранения ошибок:

1. **Логирование состояний и действий:**

Можно добавлять вывод логов, чтобы отслеживать, какие действия происходят и как меняется состояние:

let appReducer = Reducer<AppState, AppAction, AppEnvironment> { state, action, environment in

    print("Action: \(action), State: \(state)")

    // Обработка действий

    return .none

}

2. **Инспектор состояния в реальном времени (интеграция с Xcode):**

При использовании инструментов Xcode для отладки можно добавлять код, который будет выводить текущее состояние приложения, действия и эффекты.

3. **Использование** TestStore **с** record **для записи поведения:**

В TestStore можно использовать метод record, чтобы записывать последовательность действий и состояний в процессе тестирования:

store.record { store in

    store.send(.fetchData)

}

**7. Заключение**

Тестирование и отладка приложений на TCA — это важная часть процесса разработки, которая помогает обеспечить качество и стабильность приложения. Использование TestStore для тестирования редьюсеров и эффектов, а также snapshot-тестов для проверки пользовательского интерфейса, позволяет создавать надежные тесты и быстро выявлять ошибки. Вдобавок, TCA предоставляет удобные инструменты для отладки, что упрощает процесс разработки и повышает уверенность в правильности работы приложения.