Тестування через розробку в архітектурі Clean Swift. View Controller

У публікації про архітектуру Clean Swift iOS у додатка CleanStore з прикладу була лише одна сцена і дуже проста-бізнес логіка, а переходу до будь-яких інших сцен не було.

Сьогодні ми додамо нову сцену **ListOrders**, у якій будуть знаходитись Замовлення користувача. Користувач може натиснути кнопку “+” для переходу на сцену **CreateOrder**, яка була створена нами раніше.

Ви навчитеся керувати цією новою функцією за допомогою **розробки через тестування (TDD)**.

Створення UI

iOS-додатку без інтерфейсу користувача не потрібна бізнес-логіка, адже користувачеві просто немає з чим взаємодіяти. Коли ви впроваджуєте нову функцію за допомогою розробки через тестування, гарною ідеєю буде почати з UI.

Спершу створіть нову сцену за допомогою шаблонів Clean Swift Xcode:

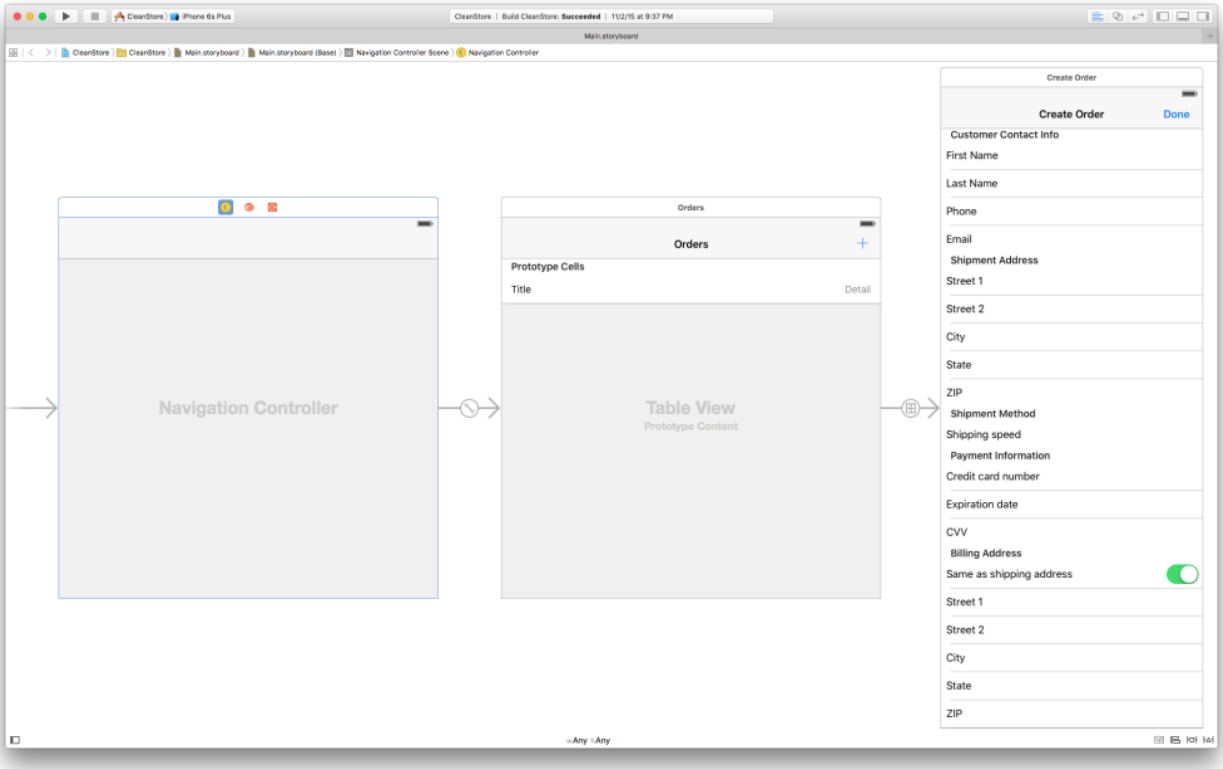
- У Project Navigator в групі Scenes створіть групу під назвою ListOrders.
- Всередині цієї групи створіть новий файл за допомогою шаблону iOS -> Clean Swift -> Scene.
- У поле Name введіть ListOrders.
- Для поля Subclass of виберіть UITableViewController.
- Створіть папку під назвою ListOrders, у якій буде зберігатися сцена. Хорошою ідеєю буде створити відповідність між файловою системою та групами в Xcode, щоб всі файли не лежали в корені проекту.
- Не забудьте встановити прапорець CleanStoreTests.

Тепер створіть новий TableViewController у Storyboard:

- drag-and-drop новий TableViewController для нового NavigationController.
- Зробіть NavigationController новим InitialViewController в Attributes Inspector.
- В Identity Inspector для нового TableViewController:
  - введіть Class = **ListOrdersViewController**.
  - введіть Storyboard ID = **ListOrdersViewController**.
  - встановіть прапорець Use Storyboard ID для Restoration ID.

- Спроектуйте UITableView:
  - введіть UITableView = **Orders**.
  - додайте UIBarButtonItem праворуч від панелі навігації і виберіть **Add** для System Item. Кнопка повинна перетворитися на “+”.
  - виберіть **Right Details** в якості стиля комірки UITableView.
  - введіть **OrderTableViewCell** в якості ідентифікатора комірки UITableView.
  - створіть Show segue під назвою **ShowCreateOrderScene** за допомогою **CreateOrderViewController**.

У результаті Storyboard матиме наступний вигляд:



Переконайтеся, що для файла Storyboard і файлів сцени відмічено **CleanStoreTests** для Target Membership у File Inspector. Це необхідно для того, аби тести могли бачити класи і обробляти ViewController із Storyboard.

Зберіть та запустіть додаток аби переконатися, що все працює правильно. На першому екрані повинна відобразитися сцена **ListOrders**. Після натискання кнопки "+" повинна відобразитися сцена **CreateOrder**, яку ми створили раніше.

Створення юніт-тестів

Для генерування юніт-тестів можна скористатися темплейтами Clean Swift Xcode. Вони генерують юніт-тести для **View Controller**, **Interactor**, **Presenter** і **Worker**. Усі типові налаштування вже виконані, тому вам залишається лише почати писати тести, які відповідають концепції TDD.

- У Project Navigator в групі **Scenes** створіть групу з ім'ям **ListOrders**.
- Створіть у цій групі новий файл, а потім виберіть **Clean Swift** і Unit Tests.
- Введіть **ListOrders** у Scene Name.
- Створіть нову папку з ім'ям **ListOrders**, у яку будуть зберігатися нові юніт-тести.
- Відмітьте лише ціль **CleanStoreTests**. Юніт-тестам повинні бути недоступні цілі тестування.

У нашому проекті повинні відображатися наступні файли юніт-тестів:

- ListOrdersViewControllerTests.swift
- ListOrdersInteractorTests.swift
- ListOrdersPresenterTests.swift
- ListOrdersWorkerTests.swift

Що слід тестувати в першу чергу?

З чого розпочати? З **View Controller**? **Interactor**? **Presenter**?

Що слід тестувати в першу чергу у середовищі **TDD**?

Після того, як ListOrdersViewController завантажився і користувач його побачив, потрібно якнайшвидше показати список Замовлень. Для цього в методі viewDidLoad() потрібно сказати **Interactor** про необхідність підхопити Замовлення. З цього можна й почати. Адже поведінка мобільного додатку сильно залежить від дій користувача.

Давайте напишемо тест, щоб пересвідчитися, що ви викликаєте гіпотетичний метод під назвою fetchOrders() всередині viewDidLoad().

Ізолювання залежностей

Давайте трохи поговоримо про те, чого ви намагаєтеся досягнути, використовуючи **TDD**.

Вам потрібно отримати список `Замовлень`, коли завантажується `view`, щоб якнайшвидше показати його користувачу. Отримання списку `Замовлень` — це бізнес-логіка. Тому потрібно, щоб **`Interactor`** отримав його у `viewDidLoad()`.

Не забувайте, що ми тестуємо `ListOrdersViewController`, тому нас не цікавить, що `ListOrdersInteractor` робить після того, як ми попросили його загрузити список `Замовлень`, а також те, як саме це відбувається.

`ListOrdersViewController` навіть не знає, що він звертається до **`Interactor`**. Йому всього лише потрібно знати, що обмін даними відбувається через `output`, який відповідає вимогам протоколу `ListOrdersViewControllerOutput`.

Давайте ізолюємо **залежність компонентів (component dependency)**, створивши `ListOrdersViewControllerOutputSpy`.

```
class ListOrdersViewControllerOutputSpy:
ListOrdersViewControllerOutput {
    // MARK: Method call expectations
    var fetchOrdersCalled = false
```

```
    // MARK: Spied methods
    func fetchOrders(request: ListOrders_FetchOrders_Request)
{
      fetchOrdersCalled = true
    }
}
```

Окрім того, потрібно ізолювати залежність данних (data dependancy). Для цього додамо модель `ListOrders_FetchOrders_Request` **у файл** `ListOrdersModels.swift`**. Залишимо її порожньою — поки що нас цікавить лише список** Замовлень**, а не критерії його пошуку чи сортування.**

```
struct ListOrders_FetchOrders_Request {
}
```

Написання тесту

Після цього напишемо тест `testShouldFetchOrdersWhenViewIsLoaded()`.

```
func testShouldFetchOrdersWhenViewIsLoaded() {
    // Given
    let listOrdersViewControllerOutputSpy =
ListOrdersViewControllerOutputSpy()
    sut.output = listOrdersViewControllerOutputSpy

    // When
    loadView()
```

```
    // Then

XCTAssert(listOrdersViewControllerOutputSpy.fetchOrdersCalled
, "Should fetch orders when the view is loaded")
}
```

Для початку замінемо результати видачі `ListOrdersViewController` **на** `ListOrdersViewControllerOutputSpy`**.**

Викличемо `loadView()`**, щоб додати параметр** `view`**, який належить View Controller в ієрархію** `view` **кореневого вікна.**

Метод `loadView()` **досить простий:**

```
func loadView() {
    window.addSubview(sut.view)
    NSRunLoop.currentRunLoop().runUntilDate(NSDate())
}
```

Тепер просто потрібно вказати, що метод `fetchOrders()` **справді був викликаний.**

Визначення меж

Розмістимо метод fetchOrders() **між** ListOrdersViewController **і** ListOrdersInteractor. **Додамо метод** fetchOrders() **у протоколи** ListOrdersViewControllerOutput **і** ListOrdersInteractorInput.

```
protocol ListOrdersViewControllerOutput {
    func fetchOrders(request:
ListOrders_FetchOrders_Request)
}

protocol ListOrdersInteractorInput {
    func fetchOrders(request:
ListOrders_FetchOrders_Request)
}
```

Додамо порожню імплементацію fetchOrders() **у** ListOrdersInteractor. **Нам просто необхідно впевнитися, що View Controller звертається до результатів свого** output **із запитом на отримання списку** Замовлень. **Що саме він робить, нас поки що не цікавить. Ми перевіримо цей метод згідно TDD, коли писатимемо тести для** ListOrdersInteractor.

```
func fetchOrders(request: ListOrders_FetchOrders_Request) {
}
```

Суть TDD полягає в тому, щоб не зловживати написанням коду аж поки він не знадобиться. Порожній метод fetchOrders() **потрібен для виконання вимог перевірки відповідності протоколу, яку виконує компілятор.**

Впровадження логіки

Повернімося до тесту `View Controller`. Покищо наш тест закінчується невдачею, адже ми ще не впровадили `юз-кейси`. Давайте зробимо так, аби все працювало.

Додамо метод `fetchOrdersOnLoad()` в `ListOrdersViewController` і викличемо його у `viewDidLoad()`. З `fetchOrdersOnLoad()` викличемо `fetchOrders()` для `output`.

```
override func viewDidLoad() {
    super.viewDidLoad()
    fetchOrdersOnLoad()
}

func fetchOrdersOnLoad() {
    let request = ListOrders_FetchOrders_Request()
    output.fetchOrders(request)
}
```

Ось дві причини розміщати завдання всередині приватного методу:

- вирогідно в майбутньому ми захочемо виконувати багато тасків з `viewDidLoad()`. Навряд чи нам захочеться, щоб усі вони висіли у `viewDidLoad()`.
- ім'я методу `fetchOrdersOnLoad()` описує конкретне завдання.

Запустимо тести ще раз — тепер все працює. `Розробка через тестування` — це весело, еге ж? Код і тести доступні на GitHub.

Підведемо підсумки

Давайте згадаємо, що ми робили, використовуючи принципи тестування через розробку з Clean Swift.

- ми ізолювали залежність компонента ListOrdersInteractor, створивши ListOrdersViewControllerOutputSpy.
- ми написали тест testShouldFetchOrdersWhenViewIsLoaded(), який перевіряє, що список Замовлень отримується під час завантаження view.
- ми впровадили порожній метод fetchOrders() у ListOrdersInteractor. Такий підхід допоможе нам зрозуміти, що тестувати та впроваджувати далі.
- ми завершили додаванням методу fetchOrdersOnLoad() і його викликом у viewDidLoad(). Тест було пройдено.

Наступного разу ми впровадимо метод fetchOrders() у ListOrdersInteractor і створимо OrdersWorker для CRUD.