Clean Swift TDD. Частина 5. Завершення VIP-циклу

У моєму останньому пості ви дізналися, як протестувати Presenter і правильно відформатувати дані для показу користувачу. Також ми розглянули метод displayFetchedOrders() компонента **View Controller**. Головним завданням цього методу є формування моделі представлення з отриманих Замовлень та визначення змісту IBOutlets-посилань.

Але ми залишили метод displayFetchedOrders() порожнім, тому що основну увагу приділили тестуванню **Presenter**. Ми повинні були переконатися, що дата Замовлення відповідає заданому формату, а також імена и прізвища об'єднані в єдине ціле.

Зараз давайте закінчимо тест displayFetchedOrders() для того, аби побачити екран з переліком Замовлень.

Ми будемо дотримуватись тих самих кроків **TDD**, які я описав в останніх своїх постах:

1. ізолювати залежності
2. написати перший тест
3. визначити кордони
4. реалізувати логіку

Оскільки межі вже встановлені за допомогою IBOutlets і IBActions, крок 3 не є обов' язковим. Це робить наше життя ще простіше!

Презентація отриманих Замовлень у вигляді таблиці

Пам'ятаєте невидимий UI компонент, який взаємодіє з контролером за допомогою IBOutlets і IBActions? Після отримання введених користувачем даних викликається метод IBAction. Разом з цим, контролер визначає вміст для відповідних IBOutlet (наприклад, text=() для елемента UILabel має сеттер-метод).

Ізолювати залежності

На екрані ListOrders елемент UITableView має IBOutlet-посилання. Він інкапсулює все, що повинно бути зроблено для показу користувачу вибраного Замовлення. Таким чином, UITableView є зовнішня залежність до ListOrdersViewController.

Давайте відслідкуємо цю залежність в нашому ListOrdersViewControllerTests:

```
class TableViewSpy: UITableView {
     // MARK: Method call expectations
     var reloadDataCalled = false

     // MARK: Spied methods
     override func reloadData() {
          reloadDataCalled = true
     }
}
```

Найпростіший спосіб показати нові отримані Замовлення полягає у простому перезавантаженні таблиці. Отже, щоб записати факт застосування методу reloadData() доцільно задіяти spy-тест.

Якщо reloadData() викликається, ви можете бути впевнені в тому, що таблиця Замовлень оновлюється. Про це подбає UIKit від Apple. Так що вам не доведеться тестувати метод reloadData() елемента UITableView.

Тестування

Тест ListOrdersViewControllerTests дуже простий.

```
func testShouldDisplayFetchedOrders() {
     // Given
     let tableViewSpy = TableViewSpy()
     sut.tableView = tableViewSpy
    let displayedOrders =
[ListOrders_FetchOrders_ViewModel.DisplayedOrder(id:
"abc123", date: "6/29/07", email: "amy.apple@clean-
swift.com", name: "Amy Apple", total: "$1.23")]
     let viewModel =
ListOrders_FetchOrders_ViewModel(displayedOrders:
displayedOrders)

     // When
     sut.displayFetchedOrders(viewModel)
```

```
    // Then
    XCTAssert(tableViewSpy.reloadDataCalled, "Displaying
fetched orders should reload the table view")
}
```

По-перше, ми створили `TableViewSpy`. По-друге, ми створили об'єкт `ListOrders_FetchOrders_ViewModel`, що містить масив `Замовлень` для презентації. Для цілей тестування, одного `DisplayedOrder` буде достатньо.

Тоді ми просто викличемо метод `displayFetchedOrders()`. Тест закінчено, метод `reloadData()` елемента `TableViewSpy` був викликаний.

Реалізація логіки

Ось реалізація методу `displayFetchedOrders()`, який робить тестовий прохід:

```
var displayedOrders:
[ListOrders_FetchOrders_ViewModel.DisplayedOrder] = []

func displayFetchedOrders(viewModel:
ListOrders_FetchOrders_ViewModel) {
    displayedOrders = viewModel.displayedOrders
    tableView.reloadData()
}
```

Ми використовуємо `UITableView` для презентації отриманих `Замовлень`. Так як же працює `UITableView`?

В контролер необхідно додати деякі методи `UITableViewDataSource` і `UITableViewDelegate`, які повинні мати доступ до отриманих `Замовлень`.

Для того, щоб полегшити це завдання, давайте використаємо приватну змінну `displayedOrders` для зберігання відформатованих `Замовлень`, які ми отримали у вигляді `моделі представлення`. Коли вам потрібен доступ до більш пізніх `Замовлень – кількості і змісту`, ви можете використовувати `displayedOrders`.

Не забудьте викликати метод `reloadData()` елемента `UITableView`.

Таблиця – кільскість секцій і рядків

Я використовую просту таблицю, оскільки моя мета полягає в тому, щоб показати вам **`Clean Swift`** і **`TDD`**, а не як побудувати красиву табличну презентацію. Тому ми маємо тільки одну секцію, а число рядків дорівнює кількості `Замовлень` для показу.

Тестування

Ось тест на кількість секцій:

```
func testNumberOfSectionsInTableViewShouldAlwaysBeOne() {
    // Given
    let tableView = sut.tableView
```

```
    // When
    let numberOfSections =
sut.numberOfSectionsInTableView(tableView)

    // Then
    XCTAssertEqual(numberOfSections, 1, "The number of table
view sections should always be 1")
}
```

Ми просто викликаємо метод `numberOfSectionsInTableView()` і стверджуємо, що завжди буде повертатись `1`.

Реалізація логіки

Реалізація методу `numberOfSectionsInTableView()` повертає значення `= 1`:

```
override func numberOfSectionsInTableView(tableView:
UITableView) -> Int {
    return 1
}
```

Тестування

Тестування кількості рядків достатньо тривіальна:

```
func
testNumberOfRowsInAnySectionShouldEqaulNumberOfOrdersToDispla
y() {
     // Given
     let tableView = sut.tableView
     let testDisplayedOrders =
[ListOrders_FetchOrders_ViewModel.DisplayedOrder(id:
"abc123", date: "6/29/07", email: "amy.apple@clean-
swift.com", name: "Amy Apple", total: "$1.23")]
     sut.displayedOrders = testDisplayedOrders

     // When
     let numberOfRows = sut.tableView(tableView,
numberOfRowsInSection: 0)

     // Then
     XCTAssertEqual(numberOfRows, testDisplayedOrders.count,
"The number of table view rows should equal the number of
orders to display")
}
```

Спочатку створюється масив `DisplayedOrder` для тестування цілей. Потім викликається метод `tableView:numberOfRowsInSection()`. Ми стверджуємо, що значення, яке повертається, дорівнює числу `Замовлень` у масиві.

Реалізація логіки

Реалізація дуже проста. Необхідно вказати кількість отриманих Замовлень у displayedOrders масиві.

```
override func tableView(tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
     return displayedOrders.count
}
```

Тестування

Швидше за все, ви вже знаєте, як налаштувати вигляд елементів таблиці. Згадайте про [невидимий UI компонент](#).

Після того, як контролер приймає від **Presenter** перелік отриманих, відформатованих Замовлень у вигляді моделі представлення даних він поновлює контент у таблиці. Остання налаштовує свої рядки, встановивши значення для текстових елементів UILabel. Таким чином, ми повинні переконатися, що текст встановлено правильно. Це означає, що text=() викликає сеттер-метод з правильним аргументом.

```
func testShouldConfigureTableViewCellToDisplayOrder() {
     // Given
     let tableView = sut.tableView
     let testDisplayedOrders =
[ListOrders_FetchOrders_ViewModel.DisplayedOrder(id:
"abc123", date: "6/29/07", email: "amy.apple@clean-
swift.com", name: "Amy Apple", total: "$1.23")]
```

```
    sut.displayedOrders = testDisplayedOrders

    // When
    let indexPath = NSIndexPath(forRow: 0, inSection: 0)
    let cell = sut.tableView(tableView,
cellForRowAtIndexPath: indexPath)

    // Then
    XCTAssertEqual(cell.textLabel?.text, "6/29/07", "A
properly configured table view cell should display the order
date")
    XCTAssertEqual(cell.detailTextLabel?.text, "$1.23", "A
properly configured table view cell should display the order
total")
}
```

Реалізація логіки

Метод `tableView:cellForRowAtIndexPath()` повинен бути вам знайомий. Спочатку ми знаходимо `Замовлення` для презентації у відповідному рядку за допомогою спеціального індексу шляху. Потім пробуємо дістати існуючий вільний рядок з черги по певному ідентифікатору. Якщо такого рядку немає ми повинні його створити.

Тепер, коли у нас є рядок, нам достатньо просто встановити `textLabel` і `detailTextLabel` для `дати замовлення` і `обсягу` відповідно.

```
override func tableView(tableView: UITableView,
cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {
     let displayedOrder = displayedOrders[indexPath.row]
     var cell =
tableView.dequeueReusableCellWithIdentifier("OrderTableViewCe
ll")
     if cell == nil {
          cell = UITableViewCell(style: .Value1,
reuseIdentifier: "OrderTableViewCell")
     }

     cell?.textLabel?.text = displayedOrder.date
     cell?.detailTextLabel?.text = displayedOrder.total

     return cell!
}
```

Висновки

Вітаємо! Ви тільки що повністю реалізували функцію з архітектури **Clean Swift**, дотримуючись правил **VIP-циклу** з використанням **TDD**.

Ви можете знайти повний вихідний код з тестами на GitHub.

Тепер ви повністю озброєні для вирішення будь-яких вимог на рівні Senior iOS Developer!

Хочете поділитися моїми попередніми постами про тестування? Нижче я навожу перелік їх усіх.

Тестування для архітектури **Clean Swift**:

- How Clean Swift makes it obvious which methods need to be tested
- Testing View Controller – Part 1
- Testing View Controller – Part 2
- Testing Business Logic in Interactor
- Testing Presentation Logic in Presenter

Тестування через розробку (**TDD**):

- Clean Swift. TDD. Частина 1. View Controller
- Clean Swift. TDD. Частина 2. Interactor
- Clean Swift. TDD. Частина 3. Worker
- Clean Swift. TDD. Частина 4. Presenter
- Clean Swift. TDD. Частина 5. Завершення VIP-циклу

Відстеження зовнішніх залежностей:

- Все, що вам потрібно знати про стеження
- Різні типи тестових двійників