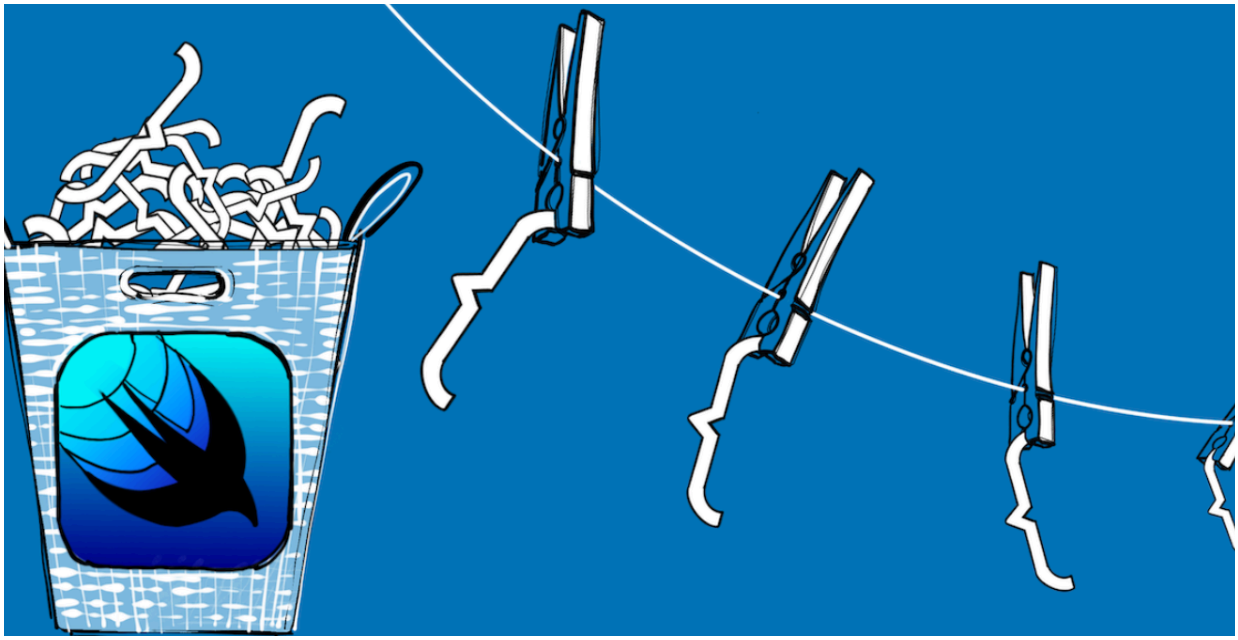
SwiftUI. Декларативная верстка

Добрый вечер, меня зовут Гладунов Егор, и, как вы уже знаете, я IOS разработчик. Мое сегодняшнее выступление нацелено на то, чтобы познакомить вас с новым подходом в построении интерфейсов в IOS-приложениях. Но зайду я немного издалека.



Подпись

SwiftUI - это Фреймворк от компании Apple, предоставляющий новый подход к разработке интерфейсов. Он был представлен полтора года назад на презентации WWDC2019. Поддерживается с версии IOS13. Ранее в приложениях мы использовали фреймворк UIKit,

предоставляющий огромное дерево классов UI-элементов. Нужно было его знать хотя бы примерно, чтобы понимать, у какого элемента какие свойства есть и так далее. Само построение интерфейса происходило в окне Interface Builder’a, а взаимодействие с кодом через связи типо IBOutlet и IBAction. SwiftUI вкратце - это декларативная верстка напрямую из кода. Итак, давайте пройдемся по основным пунктам

Да что такое, этот ваш SwiftUI?

Декларативный подход - первый пункт нашего списка. Все, что было ранее, было императивным подходом. Вот вам очень простой пример для понимания разницы - пример с бутербродом.

Императивный стиль:

- Прийти на кухню
- Приготовить инструменты
- Достать необходимые продукты
- Отрезать кусок хлеба нужной толщины
- Отрезать условную колбасу (Определенное количество колец)
- Отрезать условный сыр (Также определенное количество кусков)
- Сложить ингредиенты в определенном порядке
- PROFIT???

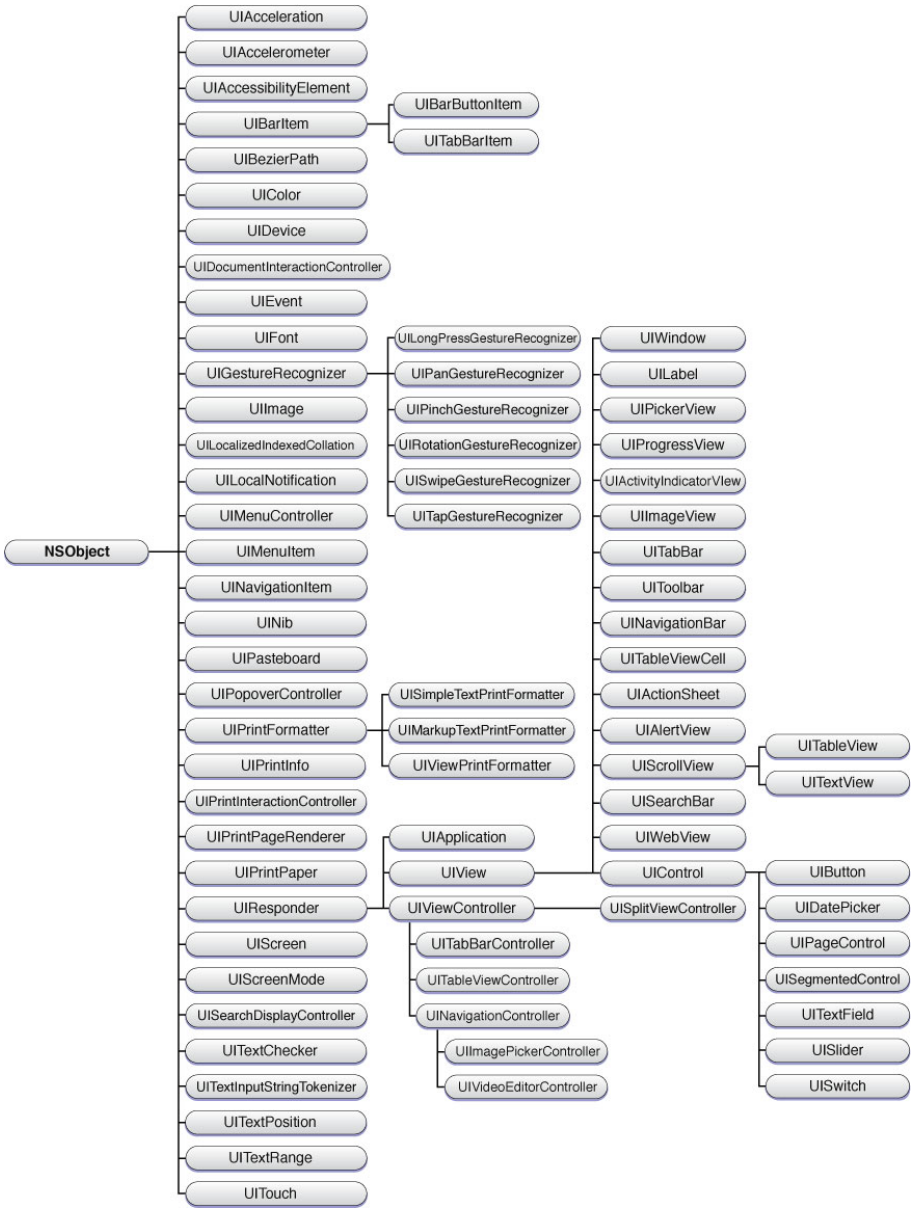
А теперь декларативный стиль:

- Сказать «Хочу бутерброд с колбасой и сыром»
- Указать, сколько ингредиентов должно быть в бутерброде
- Получить готовый бутерброд
- PROFIT

То есть, в императивном подходе мы контролируем каждый шаг, настраиваем свойства объектов, констрейнты и так далее, а в декларативном просто говорим , какой элемент хотим видеть и что в нем должно быть. Наглядно я покажу это позже на примере кода.

Обновленные UI-элементы

Ранее UI-элементы в UIKit это классы, огромное дерево классов



Порядок наследования четко определяет свойства и методы элементов. В SwiftUI все UI-элементы - это структуры, которые реализуют протокол View. При реализации протокола структура и получает все необходимые для кастомизации поля. Уникальные поля каждая структура получает свои, но у многих элементов они повторяются. Также некоторые элементы были переименованы, некоторые изменены и добавлены новые.

Например, контейнеры по осям и спейсеры.

Верстка напрямую из кода

И что с того? - спросят некоторые. Мы и сейчас можем это делать, на крайний случай есть куча библиотек, например, SnapKit. И будут неправы.

Даже прибегая к верстке через код, вы все равно пользуетесь UIKit, вы вынуждены представлять координатную плоскость, расставлять констрейнты, прорисовывать мысленно расположение элементов. В SwiftUI все в разы проще.

Во-первых, с представлением интерфейса помогает новое окно Canvas, в реальном времени показывающее интерфейс и позволяющее даже запустить его для проверки каких-либо действий, не требующих взаимодействия с данными - например, нажатия кнопок, анимации и другое.

Во-вторых, каждый новый элемент стремится к центру экрана, если ему не мешает другой элемент или правила контейнера/модификатора. Эта логика и правила контейнеров как раз и помогают избавиться от связей и констреинтов.

Какие же плюсы всего этого?

- Нет крашей из-за того, что вы забыли обновить связь между аутлетом и переменной
- Вы сразу видите результат своей верстки в том виде, в каком он будет на экране устройства. Например, в Interface Builder не отображались даже закругленные края у вьюх, здесь же все изменяется в реальном времени
- Интерфейс отрисовывается быстрее
- «Реактивность». Если какая-либо вью завязана на переменную, то при изменении значения этой переменной, весь экран моментально перерисуется.
- Увеличенная скорость разработки и внесения изменений. Вы можете спокойно удалить кусок View и ничего не сломается, ведь между элементами нет никаких констрейнтов. Или наоборот, добавление элементов не повлечет за собой долгие исправления, как было в Interface Builder.

Также стоит отметить, что Apple отошли от своего продвижения архитектуры MVC как было в UIKit. В SwiftUI они за MVVM, это видно даже в примерах кода на их официальном сайте.

Анимации

Еще одним преимуществом SwiftUI перед UIKit являются транзишены, то есть анимации. Любое передвижение или изменение вью на экране

является анимированным, что не скажешь о UIKit, где все анимации приходилось делать через UIView,animate().

Практика

Теперь перейдем к наглядным примерам.

- Анимация пружинки
- Экран логина
- Реактивность
- Кастомные вью