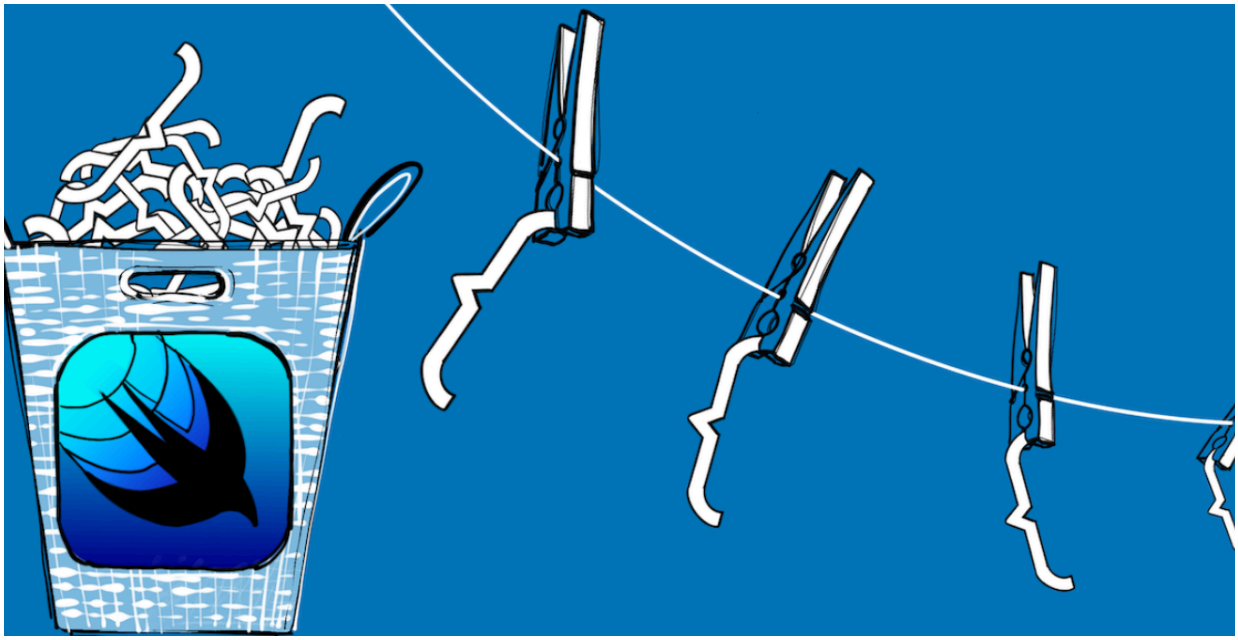
SwiftUI. Декларативная верстка

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???

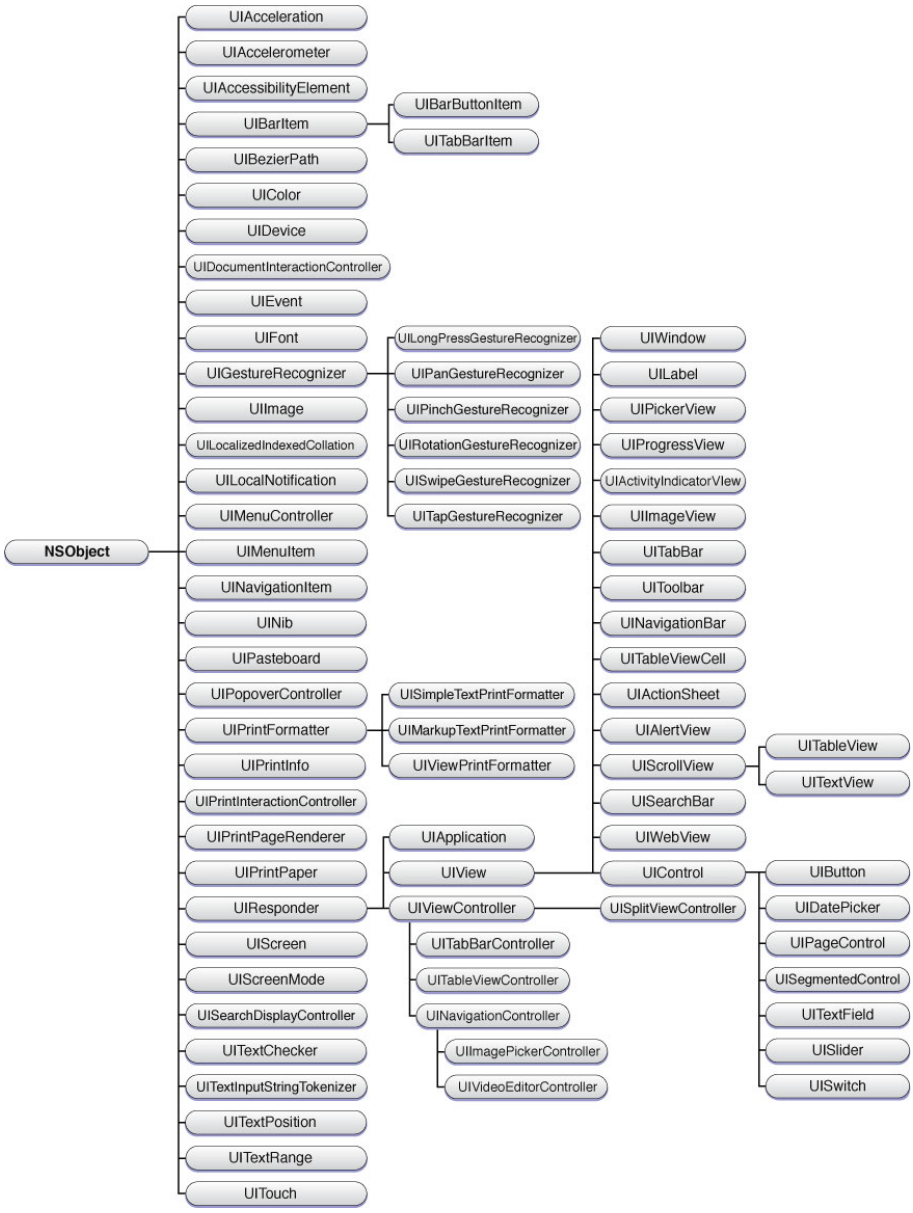
А теперь декларативный стиль:

- 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().

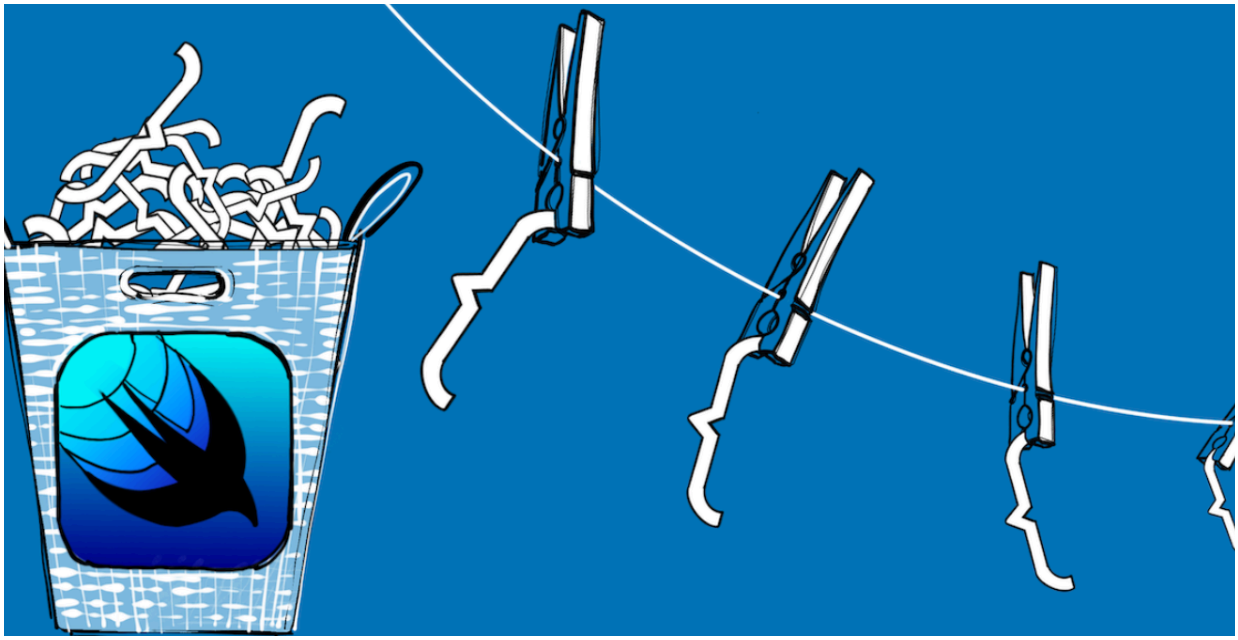
Практика

Теперь перейдем к наглядным примерам.

- Анимация пружинки
- Экран логина
- Реактивность
- Кастомные вью

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???

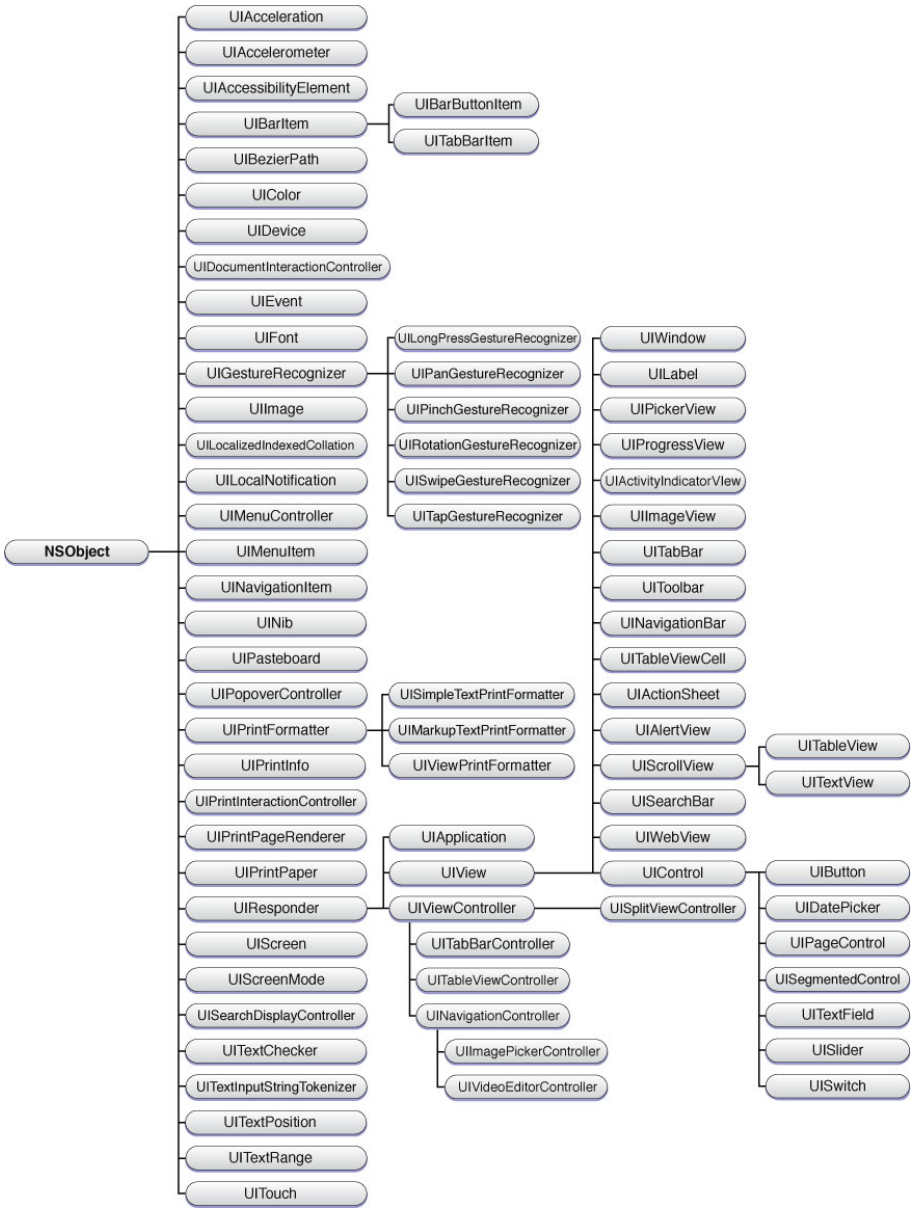
А теперь декларативный стиль:

- 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

IOS

UI-элементы в UIKit

Документация по UI-элементам: название, описание, возможности. Описывается фреймворк UIKit, подходит для SwiftUI лишь частично.

UIButton

Обычная кнопка. Можно указывать текст, шрифты, цвет самой кнопки, цвет краев, стили для разных состояний: Enabled, Disabled, Highlighted. Можно не указывать текст и цвет, тогда кнопка будет невидимой. Можно установить картинку вместо текста и цвета.

UILabel

Текст, он и в Африке текст. Можно изменять текст, фон, шрифты. Можно использовать Attributed Text.

UITextField

Поле для ввода текста, можно изменять текст в нем, плейсхолдер, цвета, границы, стили краев. Можно сделать защищенным(звездочки вместо текста). Можно указывать максимальную длину и тип клавиатуры. При вводе текст идет в одну строку и «прокручивается» по мере ввода.

UITextView

Похоже на текстфилд, но нет плейсхолдеров и текст пишется в несколько строк. Текст можно скроллить.

UIScrollView

Контейнер для контента, который не помещается на экран. Можно делать вертикальный и горизонтальный скролл. Эффект «Пружинки» отключается.

UITableView

Таблица с ячейками. Можно использовать стандартные ячейки, можно делать кастомные. Можно отключать пружинку. СТРОГО ВЕРТИКАЛЬНЫЙ СКРОЛЛ. Ячейки всегда имеют одинаковую ширину - ширину самой таблицы.

UICollectionView

Похоже на таблицу, но с некоторыми отличиями. Можно делать не только вертикальный, но и горизонтальный скролл(одно из двух), разную ширину ячеек. В остальном та же таблица.

UIStackView

Контейнер, можно засовывать любые вьюхи. Можно выбирать тип стека - вертикальный или горизонтальный, распределение контента, отступы, центрирование.

UIPickerView

Вьюха в виде барабана, который можно прокручивать для выбора варианта(Например, выбор времени на будильнике). В качестве значений используется массив. Есть разновидность для выбора даты или времени, в таком случае нужно указать только максимальную и минимальную дату.

UIProgressView

Прогресс бар. Можно изменять цвета, уровень заполнения(с анимацией и без).

UIActivityIndicatorView

Классический лоадер в виде круга из палочек, можно изменять цвет.

UIImageView

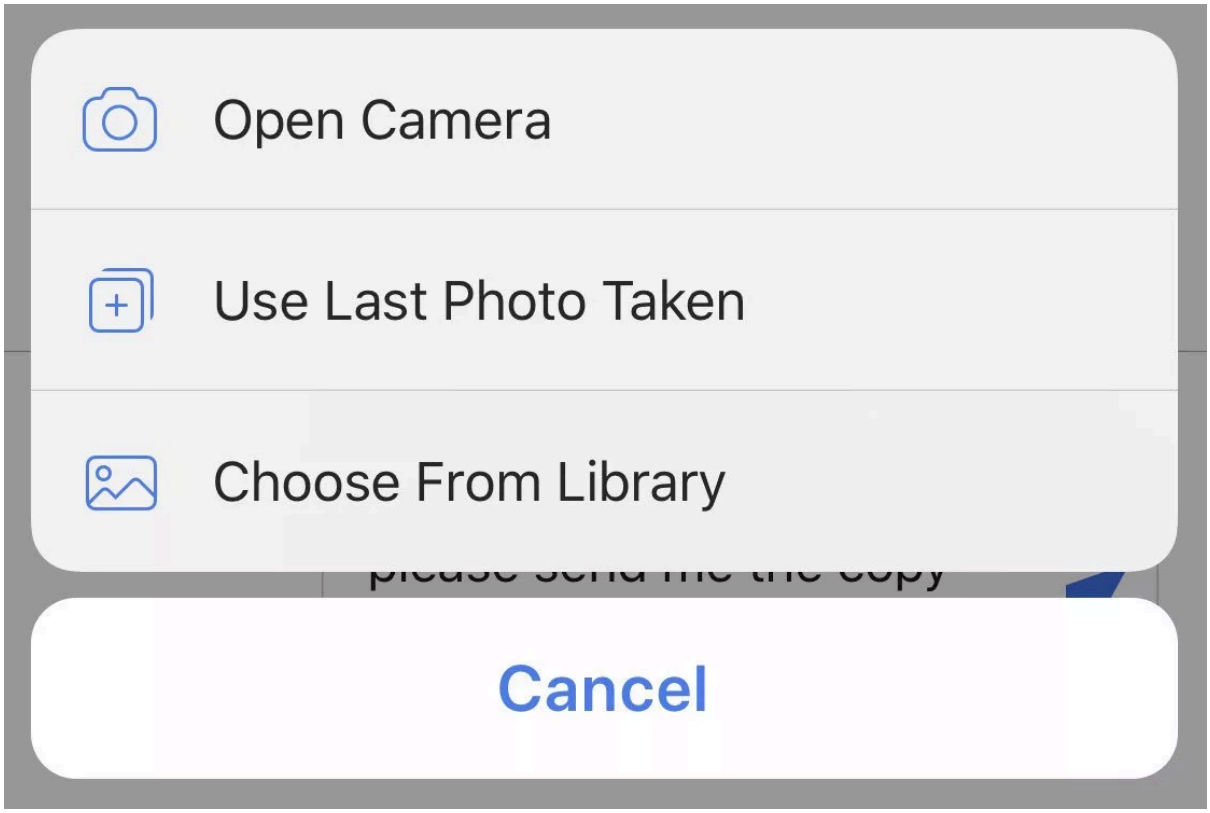
Контейнер для картинки. Можно изменять тип размещения контента(растягивать для заполнения, центрировать, отдалять картинку для умещения и т.д.)

UITabBar

Нижний бар с кнопками-вкладками. При переключении на другую вкладку состояние текущей сохраняется и отображается на том же месте при возврате.

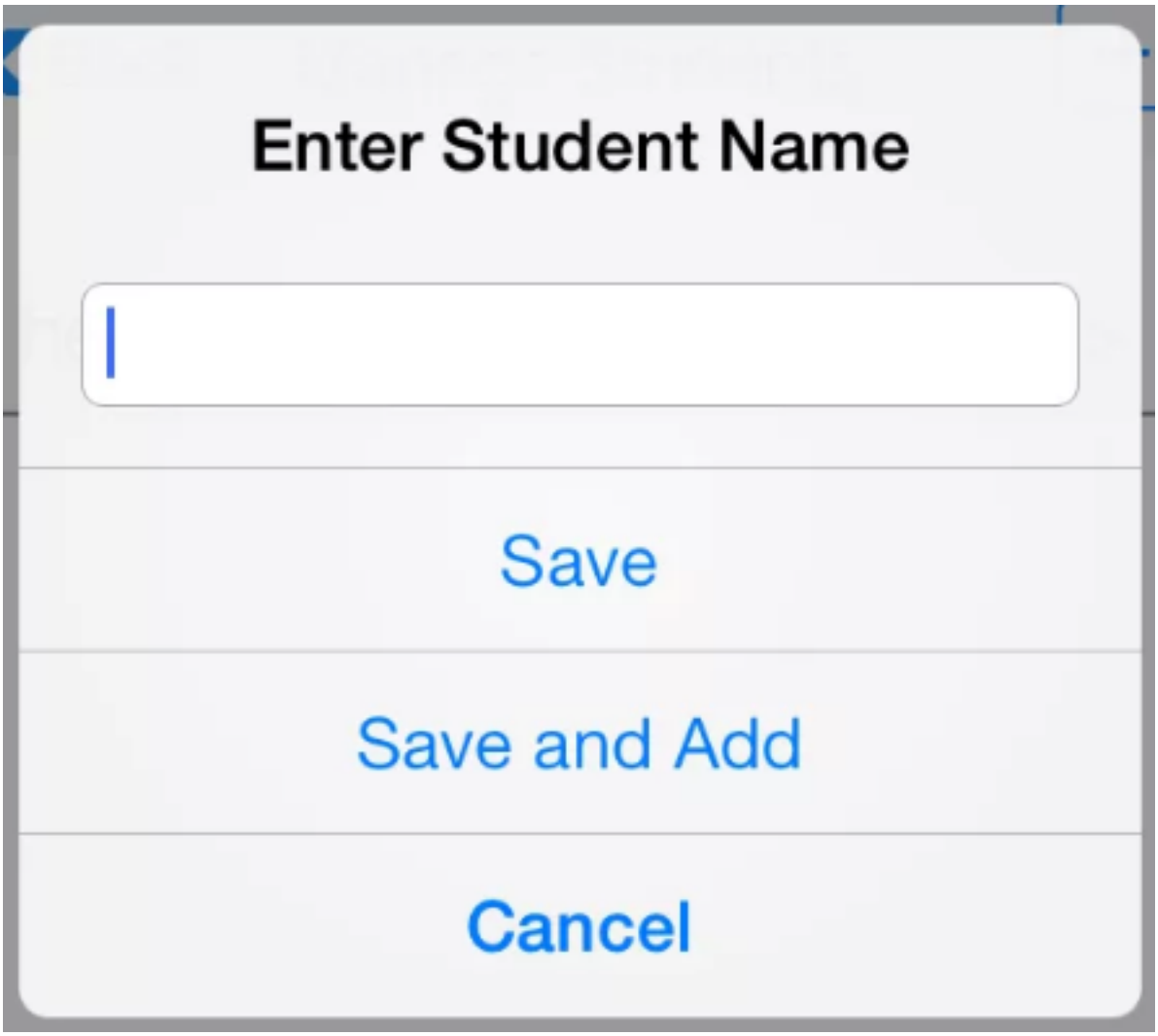
UIActionSheet

В IOS нет боттом щитов, но есть такие вьюхи. Это нативные вьюшки, рекомендуется по возможности использовать их.



UIAlertView

Нативная системная модалка. Можно дабавлять в нее различные элементы - кнопки, текстфилды, лейблы и т.д.



UISearchBar

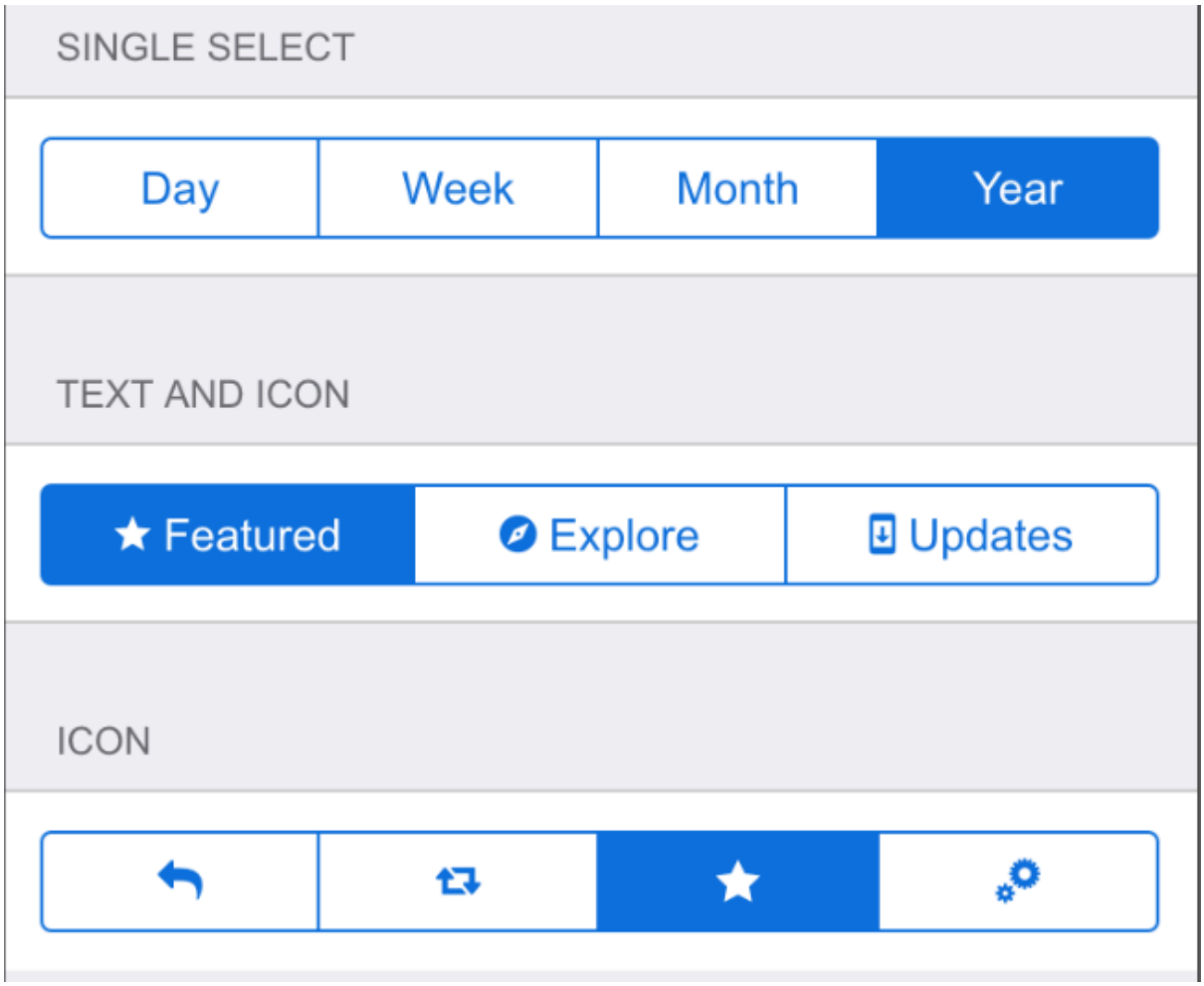
Текстфилд с небольшими надстройками для реализации поиска

UIWebView

Вебвью. Урезанный браузер внутри приложения. Часть js кода не работает

UISegmentedControl

Сегмент контроллер, можно указывать разное количество элементов, текст элементов, цвета.



UISlider

Ползунок. Можно изменять цвет заполнения, цвет самой точки контрола, цвет подложки. Точки на нем расставлять нельзя, это приходится делать вручную через другие элементы(Например, накладывая сверху UIStackView с UILabel’ами).

UISwitch

Стандартный свитч. Можно менять цвет активного и неактивного состояния, цвет контрола. НИКАКИХ ТРЕХПОЗИЦИОННЫХ СВИТЧЕЙ, НЕ ВЕРЬ ИГОРЮ.

UIGestures

Жесты, которые можно вешать на вьюхи и обрабатывать.

-UILongPressGestureRecognizer - распознание долгого нажатия

-UIPanGestureRecognizer - распознание жеста «нажать и потянуть»

-UIPinchGestureRecognizer - распознание жеста зуммирования

-UIRotationGestureRecognizer - распознание жеста кручения двумя пальцами

-UISwipeGestureRecognizer - распознание свайпа

-UITapGestureRecognizer - распознание тапа

IOS

UI-элементы в UIKit

Документация по UI-элементам: название, описание, возможности. Описывается фреймворк UIKit, подходит для SwiftUI лишь частично.

UIButton

Обычная кнопка. Можно указывать текст, шрифты, цвет самой кнопки, цвет краев, стили для разных состояний: Enabled, Disabled, Highlighted. Можно не указывать текст и цвет, тогда кнопка будет невидимой. Можно установить картинку вместо текста и цвета.

UILabel

Текст, он и в Африке текст. Можно изменять текст, фон, шрифты. Можно использовать Attributed Text.

UITextField

Поле для ввода текста, можно изменять текст в нем, плейсхолдер, цвета, границы, стили краев. Можно сделать защищенным(звездочки вместо текста). Можно указывать максимальную длину и тип клавиатуры. При вводе текст идет в одну строку и «прокручивается» по мере ввода.

UITextView

Похоже на текстфилд, но нет плейсхолдеров и текст пишется в несколько строк. Текст можно скроллить.

UIScrollView

Контейнер для контента, который не помещается на экран. Можно делать вертикальный и горизонтальный скролл. Эффект «Пружинки» отключается.

UITableView

Таблица с ячейками. Можно использовать стандартные ячейки, можно делать кастомные. Можно отключать пружинку. СТРОГО ВЕРТИКАЛЬНЫЙ СКРОЛЛ. Ячейки всегда имеют одинаковую ширину - ширину самой таблицы.

UICollectionView

Похоже на таблицу, но с некоторыми отличиями. Можно делать не только вертикальный, но и горизонтальный скролл(одно из двух), разную ширину ячеек. В остальном та же таблица.

UIStackView

Контейнер, можно засовывать любые вьюхи. Можно выбирать тип стека - вертикальный или горизонтальный, распределение контента, отступы, центрирование.

UIPickerView

Вьюха в виде барабана, который можно прокручивать для выбора варианта(Например, выбор времени на будильнике). В качестве значений используется массив. Есть разновидность для выбора даты или времени, в таком случае нужно указать только максимальную и минимальную дату.

UIProgressView

Прогресс бар. Можно изменять цвета, уровень заполнения(с анимацией и без).

UIActivityIndicatorView

Классический лоадер в виде круга из палочек, можно изменять цвет.

UIImageView

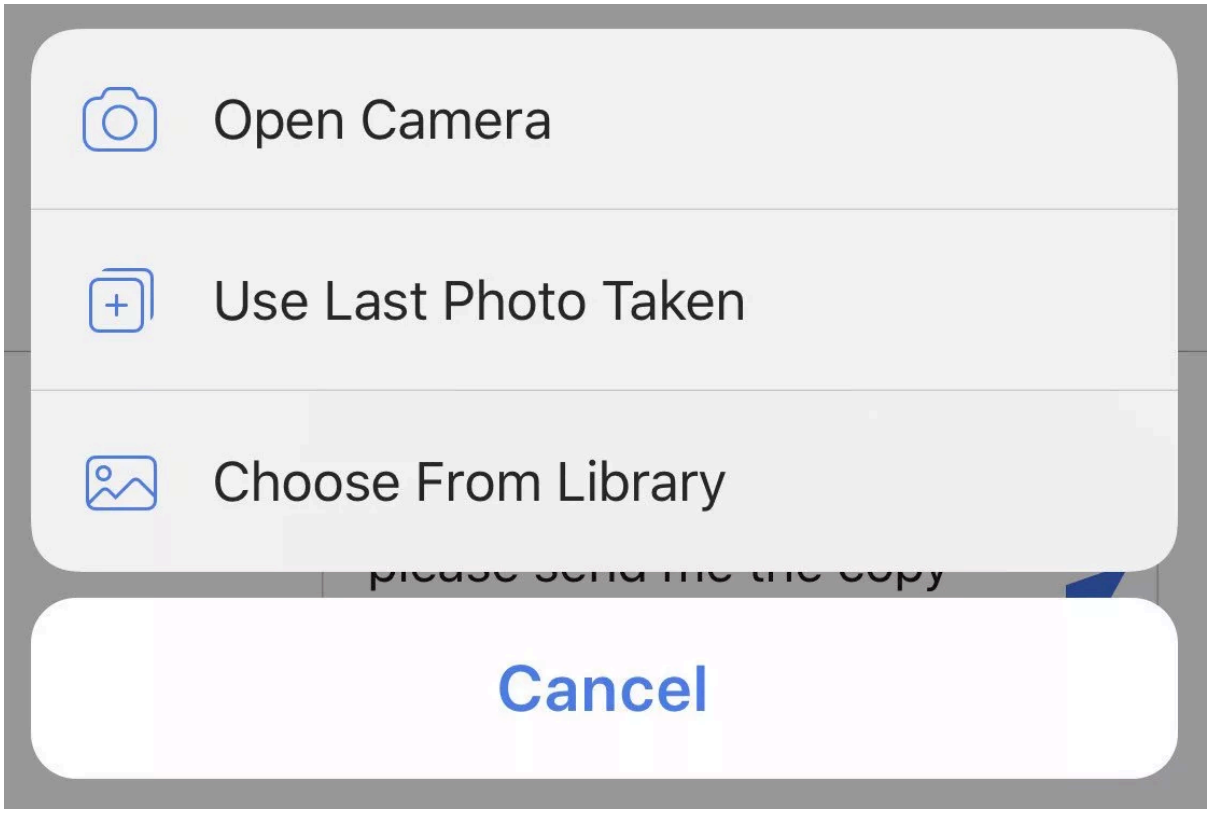
Контейнер для картинки. Можно изменять тип размещения контента(растягивать для заполнения, центрировать, отдалять картинку для умещения и т.д.)

UITabBar

Нижний бар с кнопками-вкладками. При переключении на другую вкладку состояние текущей сохраняется и отображается на том же месте при возврате.

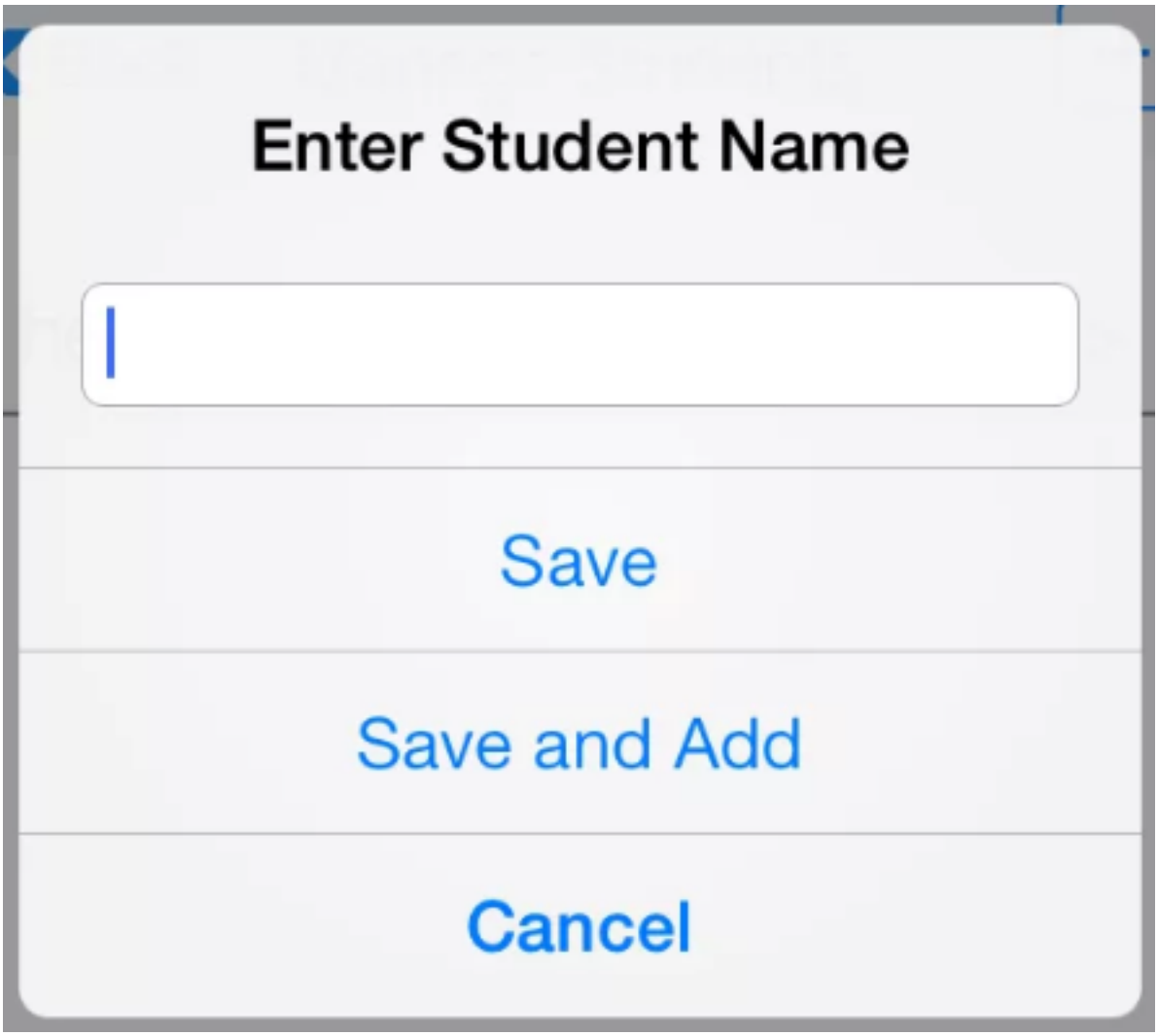
UIActionSheet

В IOS нет боттом щитов, но есть такие вьюхи. Это нативные вьюшки, рекомендуется по возможности использовать их.



UIAlertView

Нативная системная модалка. Можно дабавлять в нее различные элементы - кнопки, текстфилды, лейблы и т.д.



UISearchBar

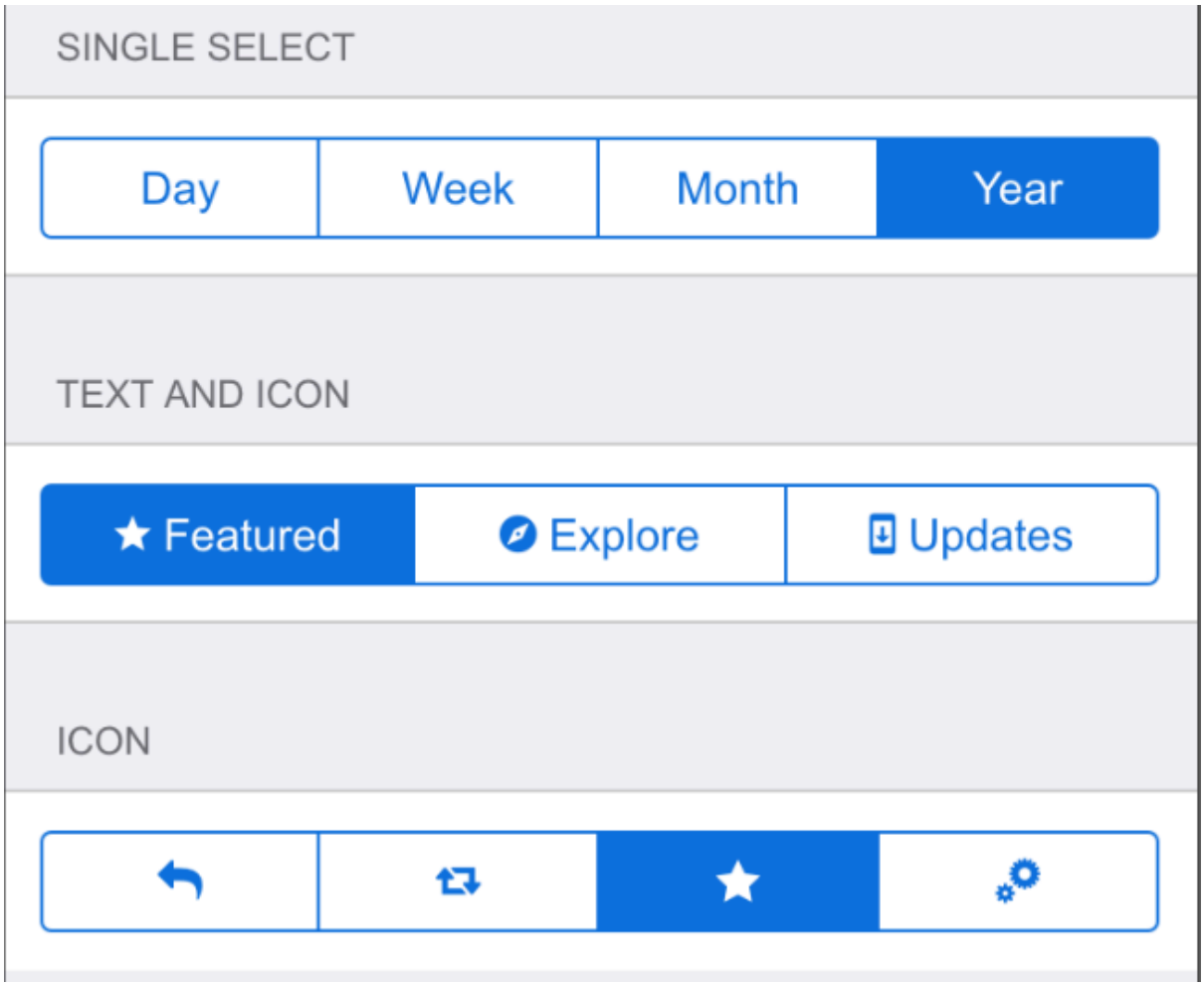
Текстфилд с небольшими надстройками для реализации поиска

UIWebView

Вебвью. Урезанный браузер внутри приложения. Часть js кода не работает

UISegmentedControl

Сегмент контроллер, можно указывать разное количество элементов, текст элементов, цвета.



UISlider

Ползунок. Можно изменять цвет заполнения, цвет самой точки контрола, цвет подложки. Точки на нем расставлять нельзя, это приходится делать вручную через другие элементы(Например, накладывая сверху UIStackView с UILabel’ами).

UISwitch

Стандартный свитч. Можно менять цвет активного и неактивного состояния, цвет контрола. НИКАКИХ ТРЕХПОЗИЦИОННЫХ СВИТЧЕЙ, НЕ ВЕРЬ ИГОРЮ.

UIGestures

Жесты, которые можно вешать на вьюхи и обрабатывать.

-UILongPressGestureRecognizer - распознание долгого нажатия

-UIPanGestureRecognizer - распознание жеста «нажать и потянуть»

-UIPinchGestureRecognizer - распознание жеста зуммирования

-UIRotationGestureRecognizer - распознание жеста кручения двумя пальцами

-UISwipeGestureRecognizer - распознание свайпа

-UITapGestureRecognizer - распознание тапа