Разработка библиотеки для реализации анимации представлений в рамках модели MVC

ОГЛАВЛЕНИЕ

Введение 2

1. Обзор существующих решений 4

1.1 Библиотека "JHChainableAnimations" 4

1.2 Библиотека "Presentr" 5

1.3 Библиотека "PresenterKit" 5

1.4 Библиотека "HYBControllerTransitions" 6

2. Основы работы анимаций. Проектирование библиотеки 7

2.1 Принципы работы анимаций в iOS 7

2.2 Механизм работы анимации в iOS 9

2.3 Механизм работы анимаций переходов между видами контроллеров 12

2.4 Архитектура библиотеки 13

3. Реализация библиотеки анимаций 16

3.1 Добавление свойства к неизменяемому объекту 16

3.2 Модификация синтаксиса библиотеки 17

3.3 Особенности разработки анимаций в трехмерном пространстве 19

3.4 Организация отправки в менеджер зависимостей 22

Заключение 26

Список источников 27

То, что было - но чему не нашлось места 28

# Введение

Стоит упомянуть такие слова как - выходом работы…, актуальность, цель, данная работа посвящена, первая глава о…, вторая глава о…, третья глава о…

За последние десятилетия пользовательский интерфейс изменился не один раз. В настоящее время анимации являются неотъемлемой частью пользовательского интерфейса, в том числе и мобильного. Если в случае приложений для стационарных компьютеров для привлечения внимания пользователя достаточно подсветить элемент, то на маленьких экранах мобильны устройств данный факт может остаться незамеченным. Поэтому анимация представляет из себя не просто элемент атрибутики приложения, а новый способ взаимодействия.

Наличие анимации выводит приложение на новый уровень коммуникации с пользователями, наглядно и интерактивно взаимодействуя с ними. В тот же момент, сама разработка анимации сказывается негативно на разработчиках: приходится писать и поддерживать большое количество кода, заниматься настройкой и тестированием анимации, следить за качеством и правильностью. Ситуация должна меняться к лучшему с этой стороны тоже. Хотя компания Apple беспокоится о разработчиках, предоставляя достаточные уровни абстракции, и позволяет не прорисовывать анимацию, высчитывая промежуточные значения, а только заниматься описательной частью, это не всегда достаточно и удобно.

Наибольшую сложность представляет анимация представлений контроллеров в момент перехода между ними. Решением данной проблемы разработчиков является библиотека для совершения анимаций.

Создание количество библиотек с открытым исходным кодом становится все популярнее, так как они решают задачи, упрощая взаимодействия с различными сервисами и службами, базами данных, с облачными хранилищами. Для управления библиотеками используются менеджеры зависимостей. Они предоставляют списки библиотек и позволяют разработчикам проводить интеграцию зависимостей.

Целью данной работы является разработка библиотеки для реализации анимации представлений в рамках модели MVC.

Данная работа разделена на три главы, каждая из которых раскрывает информацию, связанную определенным этапом создания библиотеки для анимации представлений в рамках модели MVC.

В первой главе проводится обзор существующих решений, их достоинств и недостатков, а также

Во второй главе описывается

В третьей главе

# Обзор существующих решений

Анимации является неотъемлемой частью операционной системы компании Apple iOS, и для их эффективной реализации необходимо иметь представление о средствах, которые оптимизируют данный процесс. Далее будут рассмотрены основные библиотеки, которые используются для упрощения реализации анимаций всех типов в iOS, и доступны под открытой лицензией

## 1.1 Библиотека "JHChainableAnimations"

Библиотека для совершения цепочных преобразований "JHChainableAnimations" разработана на языке Objective-C, характеризуется минимальными ограничениями на использование, обеспечивает быструю и доступную связь несколько различных анимаций, используя цепочку вызовов блоков. Также позволяет эффективно изменять обычные компоненты простым образом (смещения, повороты и масштабирование), предлагает удобства и простоту написания анимаций различной сложности, даже при их большой вложенности.

"JHChainableAnimations" позволяет использовать различные функции для управления временем анимации (т.е. функции, определяющие скорость изменения свойств компонента UIView с течением времени), которые значительно расширяют стандартный набор. Например, в стандартной библиотеке CoreAnimation присутствуют такие функции как Linear, EasyIn, EasyOut, библиотека "JHChainableAnimations" расширяет этот набор с такими фунциями как EaseInBounce, EaseOutBounce и EaseInOutBounce [3].

К достоинствам библиотеки можно отнести автоматические изменения точки опоры компонента, изменения матрицы преобразования, и другие низкоуровневые возможности для отображаемых компонентов. Также к преимуществам можно отнести имеющуюся техническую документацию, включающую руководство пользователя. Тем не менее, нет возможности совершать анимированные переходы между различными видами контроллеров, что является определенным недостатком.

## 1.2 Библиотека "Presentr"

Библиотека для выполнения показа модальных видов контроллеров "Presentr" разработана на языке Swift и ее использование доступно только с 8 версии операционной системы, что определяется доступными программными интерфейсами [4].

Принцип работы библиотеки следующий: необходимо создать объект презентации типа Presentr, что может рассматриваться как недостаток, поскольку необходимо постоянно следить за жизненным циклом этого объекта. После чего вызывается метод для анимационного показа вида нового контроллера, в который необходимо передать множество параметров, включая собственно объект. Все остальные необходимые действия выполнит библиотека.

Главным достоинством является удобство установки вида и типа анимации посредством изменения свойств у объекта анимации. Эта библиотека обладает подробной и исчерпывающей документацией. Однако, отсутствуют возможности для конфигурирования показов немодальных контроллеров.

## 1.3 Библиотека "PresenterKit"

Библиотека для анимационного показа видов контроллеров "PresenterKit" разработана на языке Swift, ограничения сходны с библиотекой "Presentr". Позволяет значительно упростить показ различных видов новых контроллеров. Ключевая особенность - упрощенная возможность настройки показа новых контроллеров, где достаточно указать тип конфигурации. Присутствует сгенерированная документация с достаточным процентом покрытия.

К недостаткам можно отнести: во-первых, отсутствие нововведений, так как присутствует упрощение взаимодействия со стандартными компонентами. В данном случае, эта библиотека является лишь удобным интерфейсом для взаимодействия. Во-вторых, сам процесс анимирования вида нового контроллера происходит в несколько этапов.

## 1.4 Библиотека "HYBControllerTransitions"

Библиотека для анимации "HYBControllerTransitions" разработана на языке Objective-C, поддерживает версии операционной системы, начиная с iOS 7. Главная особенность - предоставление большого количества нестандартных переходов, поддержка различных способов для анимирования коллекций. Из недостатков можно отметить следующие: необходимость контроля за жизненными циклами объектов в памяти, а также передачу большого количества параметров в методы и блоки, отсутствие детальной технической документации.

Здесь должен быть текст, который служит мостиком к следующей главе

# Основы работы анимаций. Проектирование библиотеки

## 2.1 Принципы работы анимаций в iOS

Анимации построены на интересном факте того, как операционная система рисует элементы на экране - момент, когда происходит отображение элементов не выполняется именно тогда, когда отдаются предназначенные для этого команды. Во время отдачи команды отображения, система запоминает и отмечает необходимые элементы, которые будут отображены в следующий момент обновления экрана. Таким образом, анимация является набором состояний объекта с течением времени

Без помощи со стороны платформы, выполнение анимации считалось бы нетривиальной задачей. Существует множество задач, которые необходимо решать: сложность расчетов, верный подсчет временных диапазонов, частота обновления экрана, потоки выполнения и многое другое. К счастью, разработчикам предоставлена помощь: непосредственно реализовывать анимацию не стоит, достаточно описательной части, которая отправляется на выполнение. Именно из-за этого для анимаций существует название - анимации "по требованию". За управление ними и остальными вещами, так или иначе связанных с отображением отвечает важный компонент системы, который Apple называет сервером анимаций.

Сервер анимаций работает в параллельном потоке. Разработчикам не приходится беспокоится о деталях его работы, в общем случае код, не отвечающий за отображение элементов на экран выполняется независимо и параллельно с анимацией.

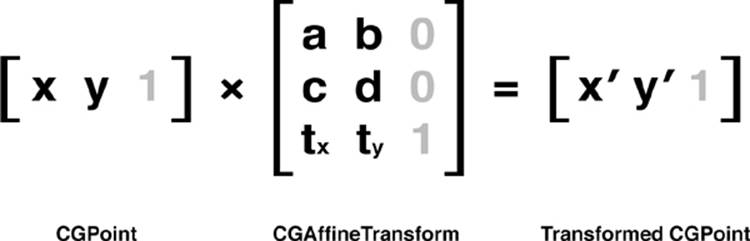
Плавность и непрерывность анимаций является фикцией, так как значения меняются путем маленьких инкрементальных изменений, дающие иллюзию плавности и непрерывного изменения. Устройство перерисовывает и обновляет экран периодично с большой частотой, что и обеспечивает качество анимаций. В этом процессе, выполняются минимально доступные инкрементальные изменения, точно между моментами перерисовки экрана.

## 2.2 Механизм работы анимации в iOS

Анимация - процесс изменения атрибута на протяжении времени. В общем случае, это будет видимый элемент чего-либо на экране. Изменение атрибута может быть позиционным: что-то двигается или меняет свои размеры, но не перескакивая между значениями, а плавно изменяя их. Другие виды атрибутов тоже могут подвергаться анимации. Цвет фона может измениться с зеленого на красный, не переключая цвета моментально, а плавно заменяя один другим.

Отображаемые объекты на экране мобильного телефона имеют общего предка - компонент интерфейса UIView. Он представляет из себя квадрат, который имеет следующие свойства: цвет фона, размеры, позицию на экране, компонент прозрачности и т. д. От данного компонента наследуются большинство остальных видимых элементов на экране. Все компоненты на экране организуют дерево компонентов: каждый элемент имеет ровно одного родителя и множество дочерних элементов. Корнем дерева является элемент окно, который не имеет родителя.

Помимо всех остальных свойств, у данного элемента присутствует матрица преобразования.



Матрица преобразования служит для определения нового состояние размеров, положения, поворота элемента. Для совершения анимации, связанной с местоположением, поворотами и размерами, разработчикам стоит использовать матрицу преобразований, вместо непосредственного изменения значений.

Процесс анимирования совершается в несколько этапов. Рассмотрим небольшой пример. Предположим, что необходимо изменить положение из 1-го во 2-е при использовании матрицы преобразования. Важной частью будет понимание того, что видит пользователь в данный момент на экране:

1. Объект находится в положении 1, именно там его и видит пользователь
2. При помощи матрицы происходит создание нового местоположения объекта, но момент отображения на экране еще не произошел, поэтому элемент отображается в 1-ом положении. Пользователь по-прежнему наблюдает объект в том же местоположении.
3. Вместе с описанием нового местоположения, описывается анимация по изменению положения объекта.
4. Код, не связанный с анимацией, продолжает выполняться до момента отображения.
5. Наступает момент отображения, если бы не было анимации, объект отображался в положении 2. Постольку поскольку наша анимация была описана, она поступает на исполнение на сервер анимаций
6. Сервер анимаций выполняет работу по перерисовке экрана, встречая анимацию, получает начальное и конечное значение и выполняет подсчет промежуточных значений. В следующие моменты отображения объект будет находиться в промежуточных значениях, которое были получены в результате интерполяции значений на протяжении времени анимации. Пользователь будет наблюдать анимированное состояние объекта
7. Наступают следующие моменты отображения. Выполняется анимация. Данное состояние анимации носит название "in-flight".
8. Анимация заканчивается, отображая элемент в положении 2, где его и наблюдает пользователь

Важной составной частью является понятие функций анимации - изинга (от англ. easing — ослабление, смягчение). Предположим, что некоторые объекты имеют условный больший вес. Например, кнопка должна ощущаться тяжелее, чем строка текста. Тяжелые объекты должны иметь большую инерцию, чем легкие, и это влияет на тип анимации и непосредственно на изинг — математическую модель ускорения или замедления. Для одного типа анимации применяется экспоненциальный тип замедления (замедление — это анимация выхода), для другого — синусоидальное ускорение (анимация входа) и т. п. Для каждого конкретного случая скорость, длительность анимации, модели скорости и типы анимаций выбираются индивидуально. Это обеспечивает имитацию живого взаимодействия.

## 2.3 Механизм работы анимаций переходов между видами контроллеров

Архитектурный подход, который предлагает Apple представляет из себя паттерн MVC. Model-view-controller (MVC, «модель- представление-контроллер», «модель-вид-контроллер») — схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные. Таким образом, вид на экране связан с контроллером, который держит связь с моделью.

Механизм работы анимаций переходов основан на анимации между видом текущего контроллера и видом показываемого контроллера. Для осуществления анимации перехода, необходимо установить ссылку на ответственный объект, делегат для показываемого контроллера.

Данный объект будет находиться в свойстве показываемого контроллера transitioningDelegate. Он должен удовлетворять протоколу UIViewControllerTransitioningDelegate. Иными словами, ответственный объект должен возвращать объекты контроллеров анимации, которые определяют продолжительность и порядок анимации. В методе, который определяет порядок анимации, расположена необходимая логика для совершения анимации.

Конфигурирование анимации происходит в главном методе:

-(void)animateTransition: (id < UIViewControllerContextTransitioning >) transitionContext { }. Параметром которого является контекст анимации. Данный контекст оперирует контейнером для анимации - временный корневым элементом для видов двух контроллеров, временем анимации, и прочими параметрами. Весь остальной этап состоит из нескольких шагов:

1. Получаются текущий и новый контроллеры, после чего соответствующие виды этих контроллеров. Контроллеры могут идентифицироваться по их видам, один из которых является показываемым, а второй скрываемым.
2. Формируется контейнер для совершения анимации.
3. Производятся манипуляции между видами контроллеров. Например, показываемый вид добавляется как дочерний в контейнер анимации, после чего стоит сконфигурировать его анимацию.
4. Производится анимация обоих видов, в которой можно использовать как сами виды, так и их графические представления - снимки текущего состояния данных видов в графическом формате.
5. Этап конфигурирования никак не связан с моментом исполнения анимаций.

В действительности, когда происходит момент перехода

При совершении непосредственно перехода система автоматически подставит возвращаемые значения от данных объектов и выполнит анимацию. В случае того, что какой-то элемент будет сконфигурирован неправильно или не предоставлен графической системе, будут использоваться стандартные значения и анимации.

## 2.4 Архитектура библиотеки

Рассмотрев вышеперечисленные библиотеки, оказалось, что для решения текущих задач по показу и сокрытию видов контроллеров, данных библиотек недостаточно. Во-первых, часть из них не реализуют работу с анимациями по переходу между видами контроллеров - JHChainableAnimations, Presentr. Во-вторых, те из них: PresenterKit и HYBControllerTransitions, которые предоставляют достаточно хороший функционал, имеют проблемы с удобством пользования и конфигурирования.

Основываясь на знаниях особенностей и реализации анимации, стоило разработать архитектуру библиотеки. Как было сказано, процесс анимирования должен быть построен на контроллерах анимации. Поэтому, в первую очередь процесс их конфигурирования должен происходить неявным образом. То есть ответственным за решение предоставления нужного анимационного контроллера должен отвечать сторонний объект. В этом случае, стоит применить паттерн разработки фабрика. Данный объект будет принимать в себя все конфигурационные аспекты анимации, и отдавать нужные непосредственно анимационным контроллерам. Это отлично подходит для анимаций типа выскакивания, переворотов, и выпадение контроллера, типа водопада. Однако для оставшихся двух типов: выталкивания и выхода контроллера с любой стороны, необходимо было унифицировать решение задачи, из-за ее однотипности. Решено было скрыть особенности показа за объектами, которые будут определять положения объектов до анимации и после нее. Иными словами, данные объекты будут полностью описывать нашу анимацию. Таким образом анимационные контроллеры двух типов получили зависимость на так называемые перформеры, которые занимаются тем самым определением местоположения объектов анимации. Глупо завязывать контроллеры анимации на конкретном виде перформеров. Руководствуясь принципом инверсии зависимостей, был выделен интерфейс для перформеров, а сами они были организованы в цепь наследования с базовым, который занимается получением значений и их сохранением. Остальные четыре типа перформера выполняют задачи местоположения видимых элементов. Тогда, убирается один из ненужных анимационных контроллеров, делегируя всю ответственность на перформеры.

Отсюда, можно сделать вывод, что архитектура библиотеки получилась гибкая. Для введения нового типа анимации, достаточно ввести новый тип контроллера анимации, возвращать его фабрикой. Чтобы расшиирить базовые типы, достаточно добавить один или несколько перформеров, и предоставить им выполнять манипуляции с анимацией.

# Реализация библиотеки анимаций

## 3.1 Добавление свойства к неизменяемому объекту

Разработка программного обеспечения связана с потенциальными проблемами и задачами, которые должны быть решены. Наиболее интересными задачами для решения оказались именно технические.

Первой задачей, требующей решения была проблема с добавлением свойств для неизменяемого контроллера. Все контроллеры в проекте создании iOS приложений наследуют базовый контроллер типа UIViewController. Первые два символа в названии обозначают принадлежность к обширному стандартному фреймворку для создания пользовательских интерфейсов - UIKit. Однако у сторонних разработчиков нет доступа для изменения исходного кода базового типа.

Первоначальной идеей было вынесения функциональности уровнем выше - создание наследника базового контроллера и предоставление функционала ему. Это было бы очень хорошим решением, однако очень неудобным для библиотеки. Разработчиками необходимо было бы наследовать все контроллеры это базового контроллера библиотеки, что не удобно, и не всегда возможно.

Выбран был другой вариант. Инструментом реализации была выбрана среда исполнения Objective-C - Runtime. Для добавления свойства к неизменяемому объекту, стоило выделить приватную категорию класса. Категория класса - функциональность, которая добавляется к уже существующим классам, заменяя наследование. В этой приватной категории выполняется переопределение методов доступа и записи через реализацию ассоциативных свойств - установка которых происходит непосредственно через обращения к среде исполнения с указанием селектора свойства в качестве ассоциативного ключа и типом ссылки для владения объекта, например objc\_getAssociatedObject() или objc\_setAssociatedObject()

## 3.2 Модификация синтаксиса библиотеки

Текущий синтаксис языка был разработан в начале 80-х годов, с тех пор мало что изменилось. Поскольку язык Objective-C изначально являлся лишь библиотекой языка C и вносил концепцию объектов, был спроектирован синтаксис, который позволял отличать вызовы функций языка C от посылки сообщений в языке Objective-C.

В рамках библиотеки для увеличения удобства ее использования, стоило немного изменить синтаксис на более читабельный и информативный. Можно было обратиться к языку C, чтобы получить иной синтаксис для библиотеки, однако не получится создать цепочку вызовов, так как в языке С не присутствует концепция объектов.

Другим ограничением было то, что с объектами, конфигурация которых требовалась, возможно только работать через посылку сообщения языка Objective-C, что делает невозможным использование чистого C.

Проблема решилась при использовании блоков в качестве возвращающих значений. Блок - это указатель на анонимную функцию, аналог замыкания, которые были представлены компанией Apple в 2012 году. Синтаксис вызова блока очень похож на вызов функции языка C. Таким образом, возможно передавать в блок необходимые параметры, тогда блок может быть результирующим значением сообщения, но все же это далеко от необходимого синтаксиса.

Оставалась проблема предоставления блоков. Решением здесь стало использование методов доступа свойств, синтаксис которых отлично подходил. Известно, что свойство - это метод доступа и метод записи, которые ссылаются на внутреннюю переменную объекта. Свойство имеет метод доступа, соответствующий имени свойству, и метод записи, похожий на имя свойства в формате "set" + имя свойства.

Полная концепция синтаксиса библиотеки представляется данным образом: имеется оболочка свойства, которая возвращает блок с требуемыми параметрами, допустим с одним числом. Далее сразу следует синтаксис вызова блока. Также из самого блока возвращается текущий объект, у которого запрашивался доступ к свойству.

Получение нового синтаксиса повысило удобство и читабельность синтаксиса библиотеки.

## 3.3 Особенности разработки анимаций в трехмерном пространстве

В процессе создания библиотеки сложности возникали не только со стороны аспектов реализации синтаксиса библиотеки или организации структурной схемы. Сложность также представляли анимации, особенно когда необходима была анимировать объекты, создавая ощущение объемного пространства.

Симуляция трехмерных отображений элементов может быть осуществима несколькими способами: использование наиболее низкоуровневых библиотек, например, OpenGL или ее упрощенных аналогов OpenGLKit, возможно было затронуть другую низкоуровневую библиотеку, которая специально была разработана под устройства Apple и была представлена в 2014 году - Metal, однако задача решилась путем использования внутренних составляющих базового компонента UIView.

Для анимаций достаточно использовать функционал базового элемента UIView до определенного этапа. В тот момент, когда его функциональности не хватает используется внутренний элемент - слой или CALayer. CA - обозначает отношение объекта к библиотеке CoreAnimation, которая отвечает за отображение и анимирование элементов на экране. У каждого элемента UIView находится ровно один связанный с ним CALayer, который занимается отображением базового компонента.

Часть свойств и значений UIView являются производными от свойств и значений CALayer, но слои представляют из себя куда больший набор функционала, чем простые элементы вида. Например, свойство transform элемента UIView, представляющее из себя структуру вида CGAffineTransform, для аффинных преобразований является частью матрицы слоя, лежащего под ним, представленной как свойство transform, ссылающееся на структуру вида CATransform3D. Матрица преобразования слоя представляет из себя более расширенный набор компонент для вычислений нового положения элемента, так как представлена для трехмерного пространства. Также, слои могут образовывать иерархии, наподобие тем, что представляют из себя элементы интерфейса, состоящие из базовых элементов. Наиболее часто, иерархия базовых элементов однозначно соответствует иерархии слоев, содержащихся в каждом базовом элементе.

При правильной настройке матрицы трехмерного преобразования должный эффект может не появится. Это связано с тем, что помимо настройки самой матрицы для трехмерного преобразования, нужно установить правильную перспективу. Именно перспектива, или угол обзора на сцену объектов в трехмерном пространстве создает необходимый эффект.

Для создания перспективы достаточно указать исходную матрицу преобразования слоя и установить значение, непосредственно отвечающее за перспективу - элемент матрицы m34, имеющий соответствующее положение в матрице - 3-его столбца и 4-ой строки. Наиболее удачно выбранная перспектива получается из диапазона значений от - 1 / 500 до -1 / 1000. Установить матрицу можно непосредственно на свойство transform слоя, однако такой подход заставляет выставлять это свойство в каждую матрицу каждого слоя и держать эти матрицы в согласованности. А в случае того, что в свойстве уже содержится не матрица преобразования, придется конкатенировать существующую с новой матрицей для перспективы, чтобы сохранить эффект от обоих матриц.

Решением данной задачи служит свойство sublayerTransform - которое представляет из себя такую же матрицу, которая автоматически применяется ко всем элементам, расположенными внизу по иерархии. Общий алгоритм получается следующим:

1. Их иерархии достаются элемент вида и его внутренний слой
2. В свойство матрицы трехмерного преобразования слоя устанавливаются необходимые значения
3. В свойство sublayerTransform слоя родительского элемента настраивается эффект перспективы
4. Производится анимация матрицы слоя для достижения нужного эффекта.

Интересными свойствами, с которым пришлось разобраться в процессе создания библиотеки, были якоря. Якоря присутствуют у слоев базовых элементов, и обеспечивают настройку опорных точек анимации. У слоя есть два якоря - один работает в плоскости, а второй обеспечивает настройку в трехмерном пространстве.

Первый, или anchorPoint, позволяет определять относительно какой точки будет происходить поворот элемента, смещение, изменение размеров. Выражается в относительных координатах с левого верхнего угла. Значением по умолчанию является середина элемента - точка с координатами [0.5, 0.5].

Второй якорь, или anchorPointZ, позволяет сконфигурировать слой, а также базовый элемент, находящийся над ним, в положении оси OZ. Значение 0.0 является значением по умолчанию, увеличение обозначает смещение в сторону пользователя, уменьшение - отдаление от пользователя.

Доступ к функциональности слоя потребовался для симуляции анимаций в трехмерном пространстве, например, переворот в пространстве вида одного контроллера в сторону вида другого контроллера. В данном случае необходимо было установить эффект перспективы, после чего грамотно настроить матрицу для трехмерной анимации. На протяжении всей анимации приходилось заниматься настройкой якорей, чтобы присутствовал эффект "притяжения" элемента к экрану.

## 3.4 Организация отправки в менеджер зависимостей

Важной частью доставки библиотеки до конечных пользователей, которыми являются разработчики, был этап организации и отправки библиотеки в общее хранилище библиотек в менеджере зависимостей.

Менеджер зависимостей представляет из себя сервис по удобной интеграции библиотек, находящихся в открытом доступе, а также приватных решений компаний в существующие проекты.

Для платформы iOS существуют на данный момент три главных менеджера зависимостей: CocoaPods, Carthage, Swift Package Manager.

Swift Package Manager представляет из себя удобное решение по интеграции для проектов, написанных на языке Swift, исходя из факта реализации библиотеки на языке Objective-C, является не совместимым.

Причиной тому, что не был выбран Carthage послужило большая распределенность. Отличительной чертой данного менеджера зависимостей от CocoaPods служит использование предварительно скомпилированных библиотек, а не их исходного кода с последующей компиляцией на этапе построения проекта или решения. В свою очередь Carthage распространен среди сообществ Swift-разработчиков, что отталкивает от его использования.

Поэтому был выбран менеджер зависимостей CocoaPods. Именно он предоставляет наиболее широкий функционал среди остальных, поскольку появился раньше остальных. Общий алгоритм отправки библиотеки в данный менеджер зависимостей выглядит следующим образом:

1. Подготовка исходного кода библиотеки
2. Настройка специального конфигурационного файла
3. Выполнение локальной проверки библиотеки
4. Предоставление документации, инструкции к проекту, его визуализация (опциональный этап)
5. Отправка библиотеки
6. Проверка на боевом проекте (опциональный этап)

Этап 1. Подготовка исходного кода. На данном этапе исходный код библиотеки собирается в едином месте, проводится его ревизия и реструктуризация. Данный этап выполняется с целью повышения удобства анализа исходного кода библиотеки для разработчиков проекта. Также стоит расположить все материалы библиотеки в определенной директории, для того, чтобы в дальнейшем было удобно указывать их местоположение.

Этап 2. Далее настраивается специальный конфигурационный файл. Пример этого файла приведен на рисунке.



В данном файле устанавливается название библиотеки, по которому остальные разработчики могут осуществлять поиск, указывается версия библиотеки. Очень важно отметить, что библиотека непосредственно не хранится в каком-либо хранилище библиотек, поэтому необходимую версию стоит отметить меткой в системе контроля версий, чтобы менеджер зависимостей мог установить библиотеки с удаленного хранилища, выполняя поиск по метке.

После устанавливаются описание, сопроводительные материалы, в роли которых чаще всего выступают снимки экрана. Ниже предоставляется информация об авторе, о библиотеке - домашняя страница, адрес расположения. Очень важным моментом является лицензирование. Для использования, изучения и модификации библиотеки она должна распространяться под одной из существующих лицензий, дающих право на это. В данном случае была выбрана лицензия MIT.

Последней частью настройки данного файла является указание пути для получения исходных файлов библиотеки, необходимые зависимости на стандартные библиотеки, поставляющиеся вместе с комплектом для разработки. Также указывается ограничение на версию операционной системы.

Этап 3. Запускается локальная проверка библиотеки при помощи командной строки с указанием файл спецификации - "pod lib lint example.podspec". В процессе выполняется построение тестового проекта с подключенной библиотекой для проверки работоспособности библиотеки, осуществляется доступ к сайту библиотеки и производится валидация всех остальных полей файла.

Этап 4. Для создания хорошей библиотеки необходимо предоставить сопроводительные материалы, которые помогут разработчикам вникнуть и разобраться в устройстве библиотеки. На данном этапе производится создание обширного описания библиотеки с помощью специального синтаксиса в файл README.md. Также, не будет лишним предоставить необходимое количество снимков работы библиотеки, например, в файлах анимации - gif. Процесс написания документации может быть существенно оптимизирован при использовании должного стиля написания исходного кода программы, путем использования инструментов для генерации документации на основе комментарий в исходном коде библиотеки.

Этап 5. Данный этап включает в себя отправку файла конфигурации библиотеки в репозиторий менеджера зависимостей, из которого пользователи смогут устанавливать себе в зависимости библиотеку.

Этап 6. Данный этап является полностью опциональным, но тем не менее рекомендуется к выполнению: выполняется создание тестового проекта и подключение библиотеки, что позволит проверить работоспособность библиотеки к интеграции и предоставляемый функционал.

Организация и отправка библиотеки в менеджер зависимостей является простым и удобным способом поделиться наработками, решениями конкретных проблем, новыми подходами к тем или иным задачам. Наибольшую сложность представляет этап правильной настройки файла конфигурации, остальная часть требует правильной последовательности для достижения цели.

# Заключение

Ключевые моменты

Разработка

Необходимость

Использование

Сложности

Результат

# Список источников

1 - Model-View-Controller [Электронный ресурс]. – Режим доступа:  https://ru.wikipedia.org/wiki/Model-View-Controller#cite\_note-1  свободный. Язык рус. (дата обращения 10.03.2015)

# То, что было - но чему не нашлось места

Для разработки на iOS используются свои поставщики библиотек исходного кода - менеджеры зависимостей. На данный момент существует два главных менеджера зависимостей - CocoaPods и Carthage. Главная отличительная черта - в степени связанности и способе подключение, которую они предоставляют. Менеджер зависимостей Carthage более модульный и не создает дополнительных проектов для интегрируемых библиотек, что в свою очередь происходит при использовании менеджера зависимостей CocoaPods.

Библиотеки облегчает рутинные задачи разработчикам, такие как взаимодействие с базой данных, написание "чистого" кода, обработка изображений, облегченное обработка ответов сервера, упрощение создания анимаций.

Об этом говорит популярность скриптовых языков таких как JavaScript, Python. Не перестают развиваться библиотеки исходного кода для других направлений и технологий, иначе именуемые пакетами, например, для C# используется менеджер пакетов NuGet, для серверной технологии Node.JS используется свой менеджер пакетов Node.JS Package Manager

Animation is crucial to the character of the iOS interface. It isn’t just cool and fun; it clarifies that something is changing or responding.

For example, one of my first apps was based on a macOS game in which the user clicks cards to select them. In the macOS version, a card was highlighted to show it was selected, and the computer would beep to indicate a click on an ineligible card. On iOS, these indications were insufficient: the highlighting felt weak, and you can’t use a sound warning in an environment where the user might have the volume turned off or be listening to music. So in the iOS version, animation is the indicator for card selection (a selected card waggles eagerly) and for tapping on an ineligible card (the whole interface shudders, as if to shrug off the tap)

В процессе ознакомления с существующими решениями возникла необходимость создать качественную, простую и удобную библиотеку для получения нестандартных анимаций. Однако стоило углубиться в механизм работы анимаций, работы анимаций переходов между видами контроллеров

В тот момент, когда ваш код выполнен и система простаивает, она может заняться новой отрисовкой элементов

Анимации работают по такому же принципу, и как часть этого процесса. Когда анимация заказывается на исполнение, она не начинается до того момента, пока система не возьмется перерисовывать экран.

Этапы: цели, задачи, преимущества, недостатки?, особенности, требования

Основываясь на недостатках вышеперечисленных библиотек для анимации и необходимых знаниях совершения анимаций, возникла необходимость разработать новую библиотеку для анимаций, которая бы удовлетворяла следующим требованиям:

1. Простота и удобство использования
2. Элегантность настройки
3. Более читабельный синтаксис

НИКИТА ~ 4903 Слова Без списка литературы