СОДЕРЖАНИЕ

Введение 5

1. Обзор существующих решений 7

1.1 Библиотека "JHChainableAnimations" 7

1.2 Библиотека "Presentr" 8

1.3 Библиотека "PresenterKit" 8

1.4 Библиотека "HYBControllerTransitions" 9

2. Основы работы анимаций. Проектирование библиотеки 10

2.1 Принципы работы анимаций в iOS 10

2.2 Механизм работы анимации в iOS 11

2.3 Механизм работы анимаций переходов между видами контроллеров 14

2.4 Архитектура библиотеки 17

3. Реализация библиотеки анимаций 20

3.1 Добавление свойств к неизменяемому объекту 20

3.2 Модификация синтаксиса библиотеки 21

3.3 Особенности разработки анимаций в трехмерном пространстве 22

3.4 Размещение библиотеки в менеджере зависимостей 26

Заключение 30

# Введение

В настоящее время анимации являются неотъемлемой частью пользовательского интерфейса, в том числе и интерфейса мобильных приложений. Если в случае приложений для настольных компьютеров для привлечения внимания пользователя достаточно простого визуального эффекта, то на маленьких экранах мобильных устройств данный способ взаимодействия может остаться незамеченным. Поэтому анимация представляет собой не просто дополнительную функциональность приложения, а один из значимых способов взаимодействия приложения с пользователем.

Наличие анимации выводит приложение на новый уровень коммуникации с пользователями, обеспечивая им наглядность и интерактивность при работе с программой. Однако разработка анимации может негативно влиять на процессы разработки: возникает необходимость написания и поддержки большого количества кода, растет сложность кода, а также появляется необходимость тестирования анимации на различных устройствах. Ситуация должна меняться к лучшему с этой стороны. Среди существующих решений можно отметить качественный набор инструментов и программных решений от компании Apple, однако учитывая современную скорость разработки, это не всегда бывает достаточным.

Наибольшую сложность при разработке представляют анимации представлений модели MVC в момент перехода между ними. Для решения могла бы использоваться библиотека для упрощения разработки анимаций. Именно это и является целью данной работы - разработка библиотеки с открытым исходным кодом для совершения анимаций между представлениями в рамках модели MVC. Для достижения поставленной цели представляется необходимым решить следующие задачи: рассмотреть существующие решения, провести анализ достоинств и недостатков, на основании имеющихся выводов спроектировать архитектуру и разработать библиотеку.

Как известно, для управления библиотеками используются менеджеры зависимостей. Они предоставляют списки библиотек с открытым исходным кодом и позволяют разработчикам проводить интеграцию зависимостей и готовых компонентов. Совместная разработка существенно помогает улучшить качество предоставляемого решения.

Данная работа разделена на три главы, каждая из которых раскрывает информацию, связанную определенным этапом решения задачи по упрощению анимации представлений в рамках модели MVC.

В первой главе проводится обзор существующих библиотек, их достоинств и недостатков, особенностей их работы и главных ограничений, с которыми могут столкнуться разработчики анимаций.

Вторая глава посвящена общим принципам работы и построения анимаций, анимациям перехода между видами контроллеров, а также описанию архитектурного подхода для решения имеющихся задач.

Третья глава описывает наиболее сложные аспекты разработки сложных анимаций, анимаций в трехмерном пространстве, а также особенности работы среды исполнения для модификации синтаксиса языка и добавления свойств к неизменяемым объектам.

В заключении представлены основные результаты работы, а также возможные перспективы дальнейшего развития и использования разрабатываемой библиотеки.

# Обзор существующих решений

Среди существующих мобильных операционных систем наибольшую заботу о пользователях и простоту взаимодействия представляет именно мобильная операционная система от компании Apple – iOS. Данный факт наблюдается в постоянном развитии и совершенствовании технологий для людей с ограниченными возможностями, таких как Voice Over, Display Accommodation, а также в удобной экосистеме для обычных пользователей.

Анимации является неотъемлемой частью данной операционной системы, 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, имеют открытый исходный код и распространяются 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 [9].

"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 Core Animation присутствуют функции Linear, EasyIn, EasyOut, библиотека "JHChainableAnimations" расширяет этот набор с такими как EaseInBounce, EaseOutBounce и EaseInOutBounce [4].

К достоинствам библиотеки можно отнести автоматические изменения точки опоры компонента, изменения матрицы преобразования и другие низкоуровневые возможности для отображаемых компонентов. Также к преимуществам относится имеющееся техническая документация, включающая руководство пользователя. С другой стороны, 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 восьмой версии операционной системы, что определяется доступными программными интерфейсами [7].

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 необходимо 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, как и в случае с предыдущей библиотекой.

## 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 [9]. 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.

Выводы: проанализировав существующие решения возникает необходимость создание более универсальной и более расширяемой библиотеки. Таким решением может стать совершенно новая библиотека для разработки анимаций. Для её разработки стоит рассмотреть общие принципы построения анимаций, а также особенности и сложности построения анимаций в момент перехода между представлениями в рамках модели MVC.

# Основы работы анимаций. Проектирование библиотеки

## 2.1 Принципы работы анимаций в iOS

Анимации построены на принципах того, как операционная система рисует элементы на экране – момент, когда происходит отображение элементов выполняется с задержкой после отдачи предназначенной для этого команды, поэтому "во время отдачи команды отображения система может запомнить и отметить необходимые элементы, которые будут отображены по-новому в следующий момент обновления экрана" [3]. Таким образом, анимацию можно трактовать как набор состояний объекта на протяжении времени.

Без помощи со стороны платформы выполнение анимации считалось бы нетривиальной задачей. Существует множество задач, которые необходимо решать: сложность расчетов, корректный подсчет временных диапазонов, частота обновления экрана, потоки выполнения и многое другое. Реализация большинства задач уже выполнена в стандартной библиотеке пользовательского интерфейса UIKit: 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 ими и остальными вещами, так или иначе связанными с отображением, отвечает важный компонент системы, который Apple называет сервером анимаций [4].

Сервер анимаций работает в параллельном потоке, несмотря на факт того, что отображение элементов происходит на главном потоке работы приложения, называемого потоком пользовательского интерфейса. "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" [3].

Плавность и непрерывность анимаций является фикцией, так как значения меняются путем маленьких инкрементальных изменений, дающих 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: движение или изменение размеров, однако без перескакивания между граничными значениями. Другие виды атрибутов тоже могут подвергаться анимации. Цвет фона может измениться с зеленого на красный, при этом цвета не переключаются моментально, а плавно заменяются один другим.

Отображаемые объекты на экране мобильного телефона имеют общего предка - компонент интерфейса UIView. Он представляет собой 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" типа UIWindow, который не имеет родителя.

Помимо всех остальных свойств, у данного элемента присутствует матрица преобразования, пример использования матрицы преобразования:

(2.1)

Исходным состоянием матрицы является единичное состояние. Матрица служит для определения нового состояния размеров, положения, поворота элемента путем умножения на значения, описывающие текущие свойства элемента. "Для совершения анимации, связанной с местоположением, поворотами и размерами разработчикам стоит использовать матрицу преобразований вместо непосредственного изменения значений, поскольку данный подход позволяет быстрее получить исходные координаты, сбросив матрицу в начальное состояние, что при умножении даст оригинальные значения" [3].

Важным моментом является то, что применение матрицы преобразования не ограничено одним разом. В процессе анимации для получения финального состояния в процессе анимации могут участвовать несколько матриц преобразования, и стоит учитывать, что в общем случае получившийся результат не будет коммутативен.

Процесс анимирования совершается в несколько этапов. В качестве примера можно привести задачу, в которой 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 применении матрицы выполняется расчет 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. Так как анимация была описана, она поступает на исполнение на сервер анимаций;
6. Сервер анимаций выполняет работу по перерисовке экрана, определяя запрошенную анимацию. Впоследствии происходит подсчет начальных и конечных значений, расчет промежуточных значений. В следующие моменты отображения объект будет находиться в промежуточных значениях, которые были получены в результате интерполяции значений на протяжении времени анимации. Пользователь будет наблюдать анимированное состояние объекта;
7. Наступают следующие моменты отображения, в процессе которых объект инкрементально изменяет значения, подвергшиеся анимации. Производится выполнение анимации. 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.

Данный механизм работы остается неизменным также и в случае анимации перехода между видами контроллеров, отличия заключаются в некоторых аспектах конфигурирования и предоставления необходимых объектов для совершения анимации.

## 2.3 Механизм работы анимаций переходов между видами контроллеров

Архитектурный подход, который предлагает Apple, представляет собой паттерн MVC. Model–view–controller (MVC, «модель – представление – контроллер», «модель–вид–контроллер») [6] — 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. Вид данного подхода продемонстрирован на рисунке 1.

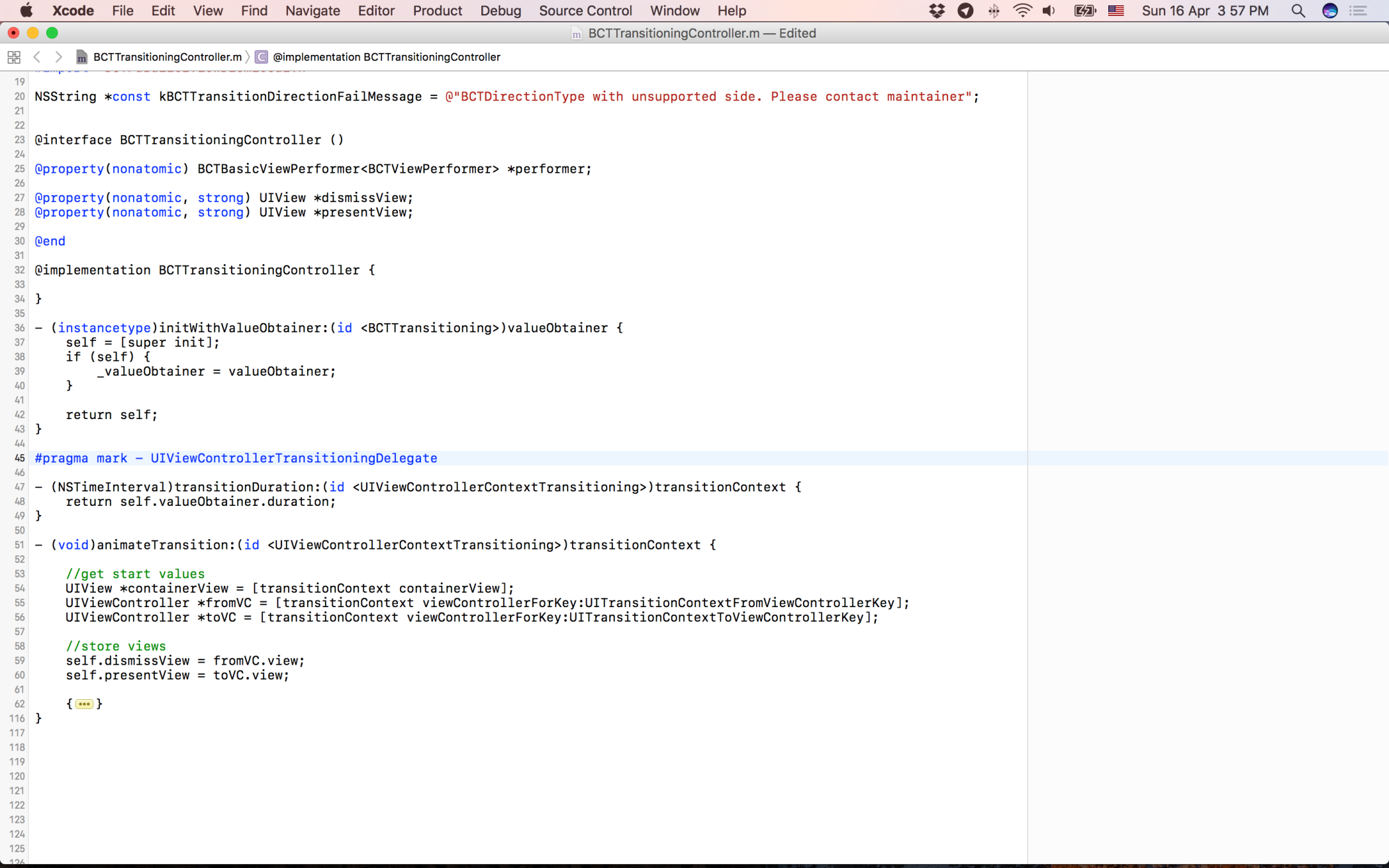


Рисунок 1 - Концепция архитектурного подхода MVC

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. В большинстве случаев ответственным объектом за анимированный показ нового контроллера выступает сам контроллер.

Данный объект будет находиться в свойстве показываемого контроллера transitioningDelegate, необходимым условием является реализация протокола UIViewControllerTransitioningDelegate. 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 [4].

Конфигурирование анимации происходит в главном методе контроллера анимации, который должен удовлетворять протоколу UIViewControllerAnimatedTransitioning. Пример реализации контроллера анимации продемонстрирован в следующем коде:



Листинг 1 - пример реализации контроллера анимации

Основная часть взаимодействия происходит с контекстом анимации. Контекст представляет собой набор данных для смены представлений контроллеров на экране. Он оперирует контейнером для анимации – временным корневым элементом для видов двух контроллеров, временем анимации, видами двух контроллеров, размерами текущего окна. Поэтому все манипуляции производятся именно с контекстом анимации. Они могут включать следующие этапы:

1. Контекст анимации предоставляет указатели на текущий и новый контроллеры, а также на соответствующие виды этих контроллеров. Контроллеры могут идентифицироваться по их видам, один из которых является показываемым, а второй скрываемым, либо идентификация может проводится по классам контроллеров, однако в таком случае наблюдается высокая связанность кода анимации и логики приложения, что не является допустимым в рамках библиотеки;
2. Предоставляется указатель на контейнер, который будет корневым элементом в момент совершения анимации;
3. Производятся манипуляции между видами контроллеров. Они включает в себя добавление и удаление элементов из контейнера, а также их замена на их графические представления для создания необычных эффектов. Например, вид показываемого контроллера должен добавиться как дочерний в контейнер анимации, после чего наступает момент конфигурирования анимации;
4. Выполняется описательная часть анимации, которая может включать оба вида контроллеров или же не взаимодействовать с ними вовсе. Для анимации исчезновения рекомендуется использовать не виды, а их графические представления - снимки текущего состояния вида;
5. Этап конфигурирования никак не связан с моментом исполнения анимаций.

Данная конфигурация не будет выполнена до тех пор, пока сам пользователь не инициирует момент перехода на представление нового контроллера. Тогда и только тогда произойдет взаимодействие с сервером анимаций, который проведет подсчет и реализацию анимации. В случае, если какой-то элемент будет сконфигурирован неправильно или не предоставлен графической системе, будут использоваться стандартные значения и анимации, что является необходимым защитным механизмом в случае невнимательности разработчика.

В процессе детального изучения были выявлены основные моменты, такие как "ленивая" установка свойств для делегатов, увеличение переиспользования кода посредством выделения новых уровней абстракции, для реализации которых требовалась корректно построенная архитектура.

## 2.4 Архитектура библиотеки

В результате анализа существующих решений в их использовании были выявлены серьезные недостатки: отсутствие гибкой архитектуры, избыточная конфигурация, неприемлемый интерфейс взаимодействия с библиотекой. Данные проблемы стали предпосылками для разработки собственной архитектуры, которая бы устраняла большинство некомпетентных и алогичных вещей. Архитектура библиотеки представлена на рисунке 2.

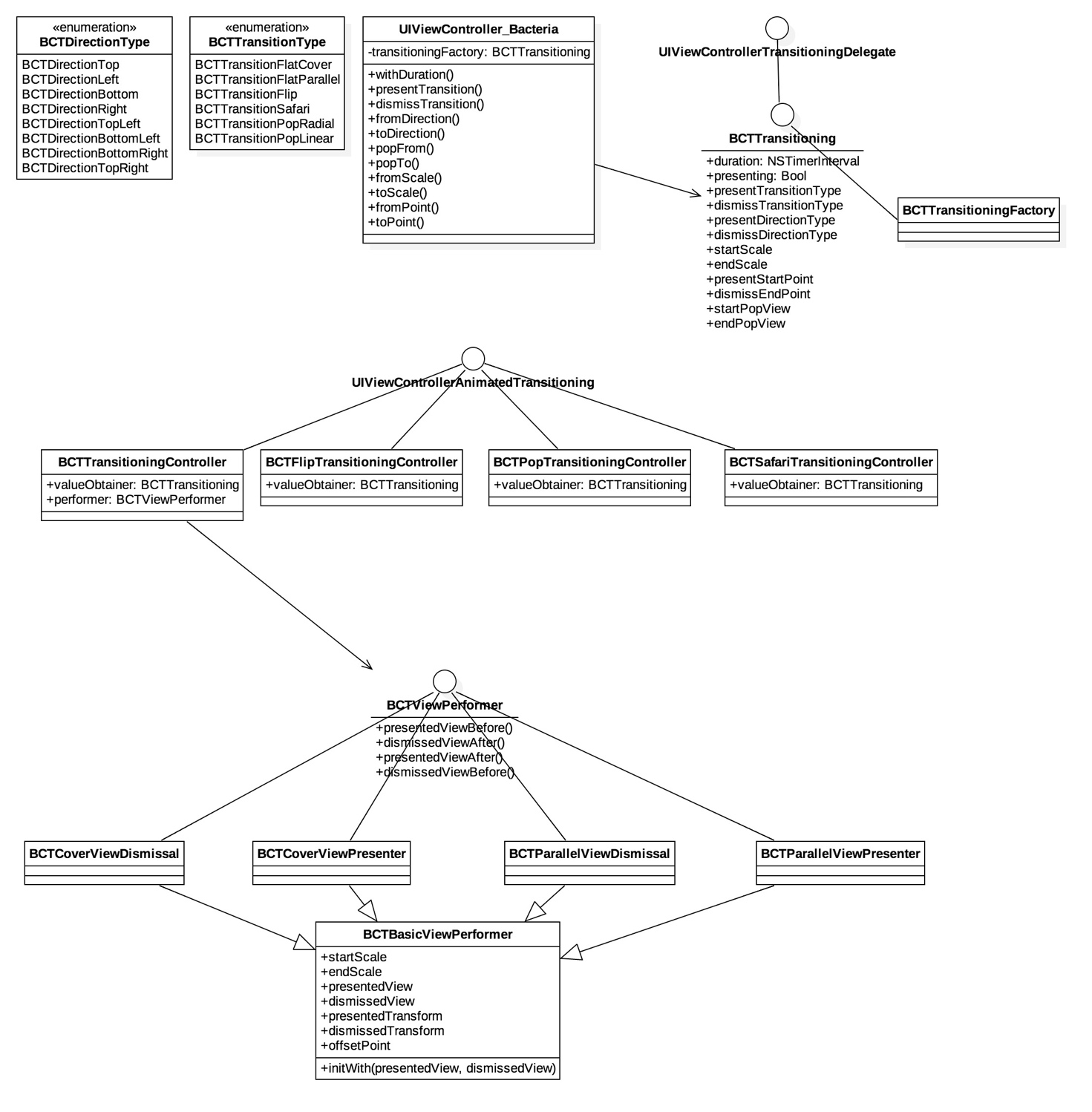


Рисунок 2 - UML-диаграмма классов библиотеки

Процесс анимирования должен быть построен на контроллерах анимации, поэтому конфигурация данных объектов должна происходить неявным образом. В данном случае логика скрыта за приватным свойством transitioningFactory категории UIViewController.

Ответственным за решение предоставления нужного анимационного контроллера для определенного типа анимации должен быть сторонний объект. В этом случае был применен наиболее подходящий шаблон проектирования - фабрика, который возвращает объекты, реализующие интерфейс UIViewControllerAnimatedTransitioning. Данные объекты, принимающие конфигурационные аспекты анимации непосредственно от объекта, скрытого за протоколом BCTTransitioning, будут определять и конфигурировать анимацию по переходу между представлениями. Это является оптимальным решением для анимаций вида выскакивания, переворотов и выпадения.

Для оставшихся двух видов анимации: выталкивания и выхода, необходимо было унифицировать решение задачи из-за ее однотипности. Конфигурация анимации была идентична, отличия заключались в направлении анимации и способа воздействия на вид текущего контроллера. Для наиболее оптимального решения необходимо было скрыть особенности показа за объектами, которые будут определять положения объектов вида UIView до анимации и после нее. Иными словами, данные объекты будут полностью описывать пред- и пост-состояния анимации. В таком случае анимационный контроллер типа BCTTransitionController получает зависимость на объекты, реализующие протокол BCTViewPerformer, которые занимаются определением местоположения объектов вида на протяжении анимации.

Руководствуясь принципом инверсии зависимостей, был выделен протокол BCTViewPerformer, что привело к образованию цепи наследования и все объекты, реализующие протокол BCTViewPerformer, также наследуют базовый класс BCTBasicViewPerformer, который занимается получением значений вида и базовым сохранением. Наследники базового класса ответственные только за подсчет новых координат. Такое решение приводит к тому, что вводятся уровни абстракции и общая ответственность разделяется на классы с меньшими зонами ответственности.

Выводы: архитектура библиотеки решает задачи по координации показа, настройки и выполнению анимации между объектами с разными уровнями ответственности, что позволяет добавить новую функциональность в библиотеку, легко ориентироваться по структуре библиотеки и изменить существующую в случае необходимости.

# Реализация библиотеки анимаций

## 3.1 Добавление свойств к неизменяемому объекту

Первой задачей, требующей решения, была проблема добавления свойств для неизменяемого контроллера. Все контроллеры в процессе разработки iOS приложений наследуют базовый контроллер типа UIViewController. Первые два символа в названии обозначают принадлежность к стандартной библиотеки 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 в этом классе. Это было бы очень хорошим решением, однако неудобным для пользователей библиотеки. Разработчикам необходимо было бы наследовать все контроллеры от базового контроллера библиотеки, что не удобно и не всегда возможно.

Возможно было использовать другой подход. 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() [11].

## 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" [1]. Блоки 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. Известно, что свойство - это метод доступа и метод записи, которые ссылаются на внутреннюю переменную объекта [11]. 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 числом. Далее сразу следует синтаксис вызова блока, после чего из этой анонимной функции возвращается текущий объект, у которого запрашивался доступ к свойству, при котором возвращался этот блок.

Получение нового синтаксиса повысило удобство и читабельность синтаксиса библиотеки.

## 3.3 Особенности разработки анимаций в трехмерном пространстве

В процессе создания библиотеки возникла необходимость отображения плоских элементов с некоторой перспективой для передачи их объема. Сложность представляли именно трехмерные анимации, особенно когда необходимо было создавать ощущение объемного пространства в процессе анимации.

Симуляция трехмерных отображений элементов может быть осуществима несколькими способами:

1. Использование низкоуровневых библиотек, например, OpenGL или упрощенных аналогов – OpenGLKit;
2. Использование другой низкоуровневую библиотеки, призванной заменить OpenGL на устройствах компании Apple, и была представлена в 2014 году - Metal;
3. Использование внутренних составляющих базового компонента UIView.

Для несложных анимаций достаточно использовать функционал базового элемента UIView до определенного этапа. В тот момент, когда его функциональности не хватает, используется внутренний элемент - слой типа CALayer. CA - обозначает принадлежность объекта к библиотеке Core Animation, работающей со всеми элементами на экране, отвечающая за отображение, анимации, работу с изображениями.

Известно, что у каждого элемента UIView находится ровно один связанный с ним CALayer, который занимается отображением этого элемента вида [2]. В свою очередь слои могут образовывать иерархии, подобные тем, какие создают элементы интерфейса, состоящие из базовых элементов. Наиболее часто иерархия базовых элементов однозначно соответствует иерархии слоев, содержащихся в каждом базовом элементе.

Часть свойств и значений UIView являются производными от свойств и значений CALayer, но слои представляют собой больший набор функционала, чем простые элементы вида. Например, свойство transform элемента UIView – структура вида CGAffineTransform для аффинных преобразований, является частью матрицы слоя, связанного с ним – свойством CALayer transform – структурой вида CATransform3D. Матрица преобразования слоя представляет собой более расширенный набор компонент для вычислений нового положения элемента, так как представлена для трехмерного пространства.

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 выбирается из диапазона значений от - 1 / 500 до -1 / 1000 [2].

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 том случае, когда в свойстве уже содержится не матрица преобразования, придется конкатенировать существующую с новой матрицей для перспективы, чтобы сохранить эффект от обеих матриц.

Решением данной задачи служит свойство sublayerTransform - которое представляет собой такую же матрицу, которая "автоматически применяется ко всем элементам, расположенными внизу по иерархии" [4].

Алгоритм установки перспективы имеет следующий вид:

1. Из иерархии достается 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.

Для правильной и последовательной анимации в трехмерном пространстве стоит обратить внимание на свойство слоя - якорь. 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, соответственно диапазон значений от нуля до единицы. 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 Диапазон значений ограничен значениями числа с плавающей точкой одинарной точности. 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 настроить матрицу для трехмерной анимации. На протяжении всей анимации производились конфигурации значений якорей для достижения эффекта "прилипания" элемента к экрану.

## 3.4 Размещение библиотеки в менеджере 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 собой открытый сервис для удобной интеграции библиотек, находящихся в открытом доступе, а также закрытых решений компаний в корпоративные проекты.

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

Swift Package Manager [12] представляет собой удобное решение по интеграции для проектов, написанных на языке Swift, но т.к. библиотека реализована на языке Objective-C, то возникает проблема совместимости.

Причиной того, что не был выбран Carthage стало использование данным менеджером зависимостей заранее скомпилированных динамических библиотек. Это является основным отличием от менеджера зависимостей CocoaPods, который встраивает исходный код библиотеки в компиляцию всего проекта. Таким образом, наиболее подходящим представляется менеджер зависимостей CocoaPods.

Общий алгоритм отправки библиотеки в 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 файла продемонстрирован ниже.



Листинг 2 - пример файла конфигурации библиотеки

В файле конфигурации указывается 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 нужную версию библиотеки с удаленного хранилища, выполняя поиск по метке.

После указываются описание и сопроводительные материалы, в роли которых чаще всего выступают снимки экрана. Далее предоставляется информация об авторе, о библиотеке – адрес веб-страницы, адрес расположения для доступа к исходному коду. 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 [13], поскольку позволяет в проекте использовать код с другой лицензией, что является ключевым моментом для коммерческих проектов.

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 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 рекомендуется добавить небольшое 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 комментариев 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.

Выводы: 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

В ходе выполнения выпускной квалификационной работы была реализована библиотека для реализации анимации представлений в рамках модели MVC. Подробно были проанализированы существующие решения данной задачи, были определены достоинства и недостатки каждого, на основании чего были выдвинуты предпосылки для разработки новой библиотеки.

В начале процесса проектирования были рассмотрены основные этапы и механизмы работы анимаций в операционной системе iOS, особое внимание было уделено описанию особенностей работы анимаций перехода между видами контроллеров. После чего была описана структура возможной библиотеки.

На основании разработанного проекта была реализована библиотека для реализации анимации представлений. В ходе работы были решены проблемы, связанные с синтаксисом библиотеки, а также осуществлено добавление свойств для неизменяемых объектов.

В данный момент библиотека находится в открытом доступе, готова к использованию, прошла проверку и опубликована в менеджере зависимостей CocoaPods. Среди направлений развития стоит отметить расширение текущих возможностей библиотеки. План развития подробно раскрыт в описании и включает следующие этапы: оптимизация исходного кода и поддержка интерактивных переходов.

Важно отметить, что в результате выполнения данной работы были получены практические навыки разработки программного обеспечения, а также закреплены знания в области проектирования информационных систем.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Хиллегасс А. Objective-C Программирование для iOS и MacOS – СПб. : Питер, 2013. 254 с.
2. Nick Lockwood. iOS Core Animation: Advanced Techniques. – Addison-Wesley Professional, cop. 2013. – 355 p.
3. Нойбург М. Программирование для iOS 7. Основы Objective-C, Xcode, Cocoa. Перевод с английского – М. : ООО "И. Д. Вильямс", 2014. 384 с.
4. Руководство по библиотеке Core Animation [Электронный ресурс]. – Режим доступа: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreAnimation\_guide свободный. Язык англ. (дата обращения 24.04.2017)
5. Кочан С. Программирование на Objective-C. – М. : "ЭКОМ Паблишерз", 2016 – 550 с.
6. Model-View-Controller [Электронный ресурс]. – Режим доступа:   https://ru.wikipedia.org/wiki/Model-View-Controller свободный. Язык рус. (дата обращения 20.04.2017)
7. Усов В. Swift. Основы разработки приложений под iOS. – СПб. : Питер, 2016 – 304 с.
8. Apple Inc. iOS Human Interface Guidelines – Apple Inc., cop. 2015. – 431 p.
9. Нахавандипур В. iOS. Приемы программирования. – СПб. : Питер, 2014. – 832 с.
10. Программирование при помощи блоков [Электронный ресурс]. – Режим доступа: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/Blocks/ свободный. Язык англ. (дата обращения 15.03.2017)
11. Программирование при помощи свойств [Электронный ресурс]. – Режим доступа: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/DeclaredProperty.html свободный. Язык англ. (дата обращения 15.03.2017)
12. Swift Package Manager [Электронный ресурс]. – Режим доступа: https://swift.org/package-manager/ свободный. Язык англ. (дата обращения 11.02.2017)
13. Лицензия MIT [Электронный ресурс]. – Режим доступа: <https://opensource.org/licenses/MIT> свободный. Язык англ. (дата обращения 27.03.2017)