

Программист мобильных устройств. Цифровые профессии

Дипломный проект по теме:

Редактор фотографий для iOS

Студент: Стрельцов Дмитрий Александрович

Ростов-на-Дону 2024

**Оглавление**

[Введение 3](#_Toc143955438)

[Глава I Теоретическая часть 5](#_Toc143955439)

[Развитие программирования для Apple 8](#_Toc143955440)

[История Swift и почему его используют для разработки на Apple устройствах 9](#_Toc143955441)

[Фрейморк UIKit 11](#_Toc143955442)

[Storyboard или SwiftUI 12](#_Toc143955443)

[UITest и Unit-тесты 16](#_Toc143955444)

[Глава II Практическая часть. Приложение редактор фотографий 17](#_Toc143955445)

[Класс ImagePicker 19](#_Toc143955446)

[Класс ViewController 25](#_Toc143955447)

[Класс EditorViewController 28](#_Toc143955448)

[Класс UIColor+Extra 30](#_Toc143955449)

[Assets и иконка приложения 32](#_Toc143955450)

[Тестирование 33](#_Toc143955451)

[Заключение 34](#_Toc143955452)

[Список использованной литературы 35](#_Toc143955453)

[Приложения 36](#_Toc143955454)

# Введение

В настоящее время приложения для редактирования фотографий на смартфонах имеют огромную важность и пользуются большой популярностью. Они позволяют пользователям улучшать, изменять и преобразовывать свои фотографии прямо на своих устройствах. Вот некоторые причины, почему эти приложения так важны:

1. Удобство использования: Многие приложения для редактирования фотографий на смартфонах имеют простой и интуитивно понятный интерфейс, который позволяет пользователям легко и быстро редактировать свои фотографии. Это делает их доступными для широкой аудитории пользователей.
2. Мобильность: Смартфоны всегда под рукой, поэтому приложения для редактирования фотографий позволяют пользователям редактировать свои снимки в любое время и в любом месте. Нет необходимости использовать компьютер или специализированное оборудование.
3. Разнообразие инструментов и эффектов: Приложения для редактирования фотографий на смартфонах обычно предлагают широкий набор инструментов и эффектов, которые позволяют пользователю создавать уникальные и креативные фотографии. Они могут включать функции обрезки, регулировки яркости и контрастности, применение фильтров и многое другое.
4. Социальные сети и обмен фотографиями: Возможность редактировать фотографии на смартфоне позволяет пользователям легко поделиться своими работами в социальных сетях и других платформах обмена фотографиями. Это позволяет им проявить свою творческую сторону и получить признание от других пользователей.
5. Экономия времени: Редактирование фотографий на смартфоне позволяет пользователям сэкономить время, поскольку они могут сразу же редактировать и делиться фотографиями без необходимости передачи их на компьютер и использования специального программного обеспечения.

В целом, приложения для редактирования фотографий на смартфонах играют важную роль в современной культуре обмена фотографиями, предоставляя пользователям инструменты для творческого выражения и улучшения своих снимков.

Цели проекта по редактированию фотографий:

1. Предоставление пользователям возможности улучшать и изменять свои фотографии: Основная цель такого приложения - предоставить пользователям инструменты и функции, позволяющие им улучшить качество своих фотографий.
2. Создание креативных эффектов и фильтров: Приложение может предлагать различные эффекты и фильтры, которые пользователи могут применять к своим фотографиям для создания уникального визуального стиля.

Данный проект решает проблему удобства использования и интуитивный интерфейс. Важной целью является создание приложения с простым и интуитивно понятным интерфейсом, который позволяет пользователям легко и быстро редактировать свои фотографии. Это помогает пользователям сосредоточиться на творческом процессе, а не на сложностях использования приложения.

Приложение реализовано для Apple устройств, реализовано с помощью языка Swift. Код для приложения написан в программах XCode и Visual Studio Code. Интерфейс приложения реализован с помощью storyboard и с использованием UIKit, а также с помощью дополнительного класса ImagePicklerDelegete.

Приложение представляет из себя главный экран, где пользователь по нажатию на кнопку может загрузить свою картинку или сделать фото, при нажатии на кнопку редактирования, приложение перейдет на новый экран где пользователь сможет отредактировать свою фотографию.

В данном проекте я составлял логику приложения, производил тестирование, занимался дизайном.

# Глава I Теоретическая часть

Для разработки продукции Apple можно использовать несколько языков программирования, включая:

* **Swift**: является основным языком программирования для разработки приложений под платформы Apple, такие как iOS, macOS, watchOS и tvOS. Он разработан Apple и предлагает современный синтаксис и мощные функциональные возможности.
* **Objective-C**: был основным языком программирования для разработки приложений для платформ Apple до появления Swift. Он является объектно-ориентированным языком программирования и все еще используется в некоторых старых проектах или библиотеках.

Эти два языка программирования являются основными для разработки приложений для платформ Apple. Разработчики могут использовать Swift или Objective-C в зависимости от своих предпочтений и требований проекта.

Оба языка программирования, Swift и Objective-C, используются для разработки приложений для платформ Apple, и каждый из них имеет свои преимущества. Выбор между Swift и Objective-C зависит от предпочтений разработчика, опыта и требований проекта. Вот некоторые факторы, которые могут помочь принять решение о выборе языка:

* Swift:

- Более современный язык: Swift был разработан Apple в 2014 году и предлагает более современный и лаконичный синтаксис, что делает код более читаемым и легким в поддержке.

- Безопасность типов: Swift предлагает систему проверки типов на этапе компиляции, что помогает выявить ошибки и предотвратить некоторые типичные ошибки программирования.

- Высокая производительность: Swift обладает высокой производительностью и эффективностью выполнения кода.

- Богатая стандартная библиотека: Swift включает в себя множество функций и структур в стандартной библиотеке, что упрощает разработку приложений.

* Objective-C:

- Большая база кода: Objective-C является старым и широко используемым языком для разработки приложений под платформы Apple. Существует большая база кода и библиотек, написанных на Objective-C, которые могут быть полезны в разработке.

- Обратная совместимость: Objective-C совместим с кодом, написанным на C, что позволяет использовать существующий код и библиотеки на Objective-C.

- Большой опыт: Если у вас уже есть опыт работы с Objective-C или вы присоединяетесь к команде разработки, где используется Objective-C, то выбор Objective-C может быть логичным.

Objective-C - это объектно-ориентированный язык программирования, который был разработан в 1980-х годах и долгое время был основным языком программирования для разработки приложений для платформ Apple. В последние годы он был заменен Swift, но все еще широко используется в существующих проектах и библиотеках.

Objective-C использует синтаксис C с добавлением синтаксических конструкций для объектно-ориентированного программирования. Классы и объекты объявляются с использованием синтаксиса @interface и @implementation.

В Objective-C взаимодействие между объектами осуществляется путем отправки сообщений. Сообщение представляет собой вызов метода на объекте. Синтаксис отправки сообщения выглядит следующим образом: [objectName methodName].

Objective-C поддерживает динамическую типизацию, что означает, что тип объекта может быть определен и изменен во время выполнения программы. Это позволяет создавать гибкий и адаптивный код.

Objective-C использует систему автоматического счетчика ссылок (ARC - Automatic Reference Counting) для управления памятью. ARC автоматически отслеживает ссылки на объекты и освобождает память, когда объект больше не используется.

Objective-C имеет большую базу кода и библиотек, которые разработаны для платформ Apple. Это позволяет использовать существующий код и библиотеки для разработки приложений.

Хотя Objective-C все еще используется в некоторых проектах, Apple активно продвигает Swift в качестве предпочтительного языка программирования для разработки приложений под платформы Apple. Swift предлагает современный синтаксис, безопасность типов и высокую производительность, что делает его привлекательным для новых проектов и разработчиков.

В итоге, выбор между Swift и Objective-C зависит от конкретной ситуации и предпочтений разработчика. Swift является более современным и рекомендуемым языком Apple, но Objective-C все еще имеет свое место в экосистеме разработки приложений для платформ Apple.

В данном проекте большее внимание привлек Swift по всем вышеуказанным преимуществам.

## Развитие программирования для Apple

Apple активно развивает программирование и предлагает различные инструменты для разработчиков. Вот несколько ключевых моментов в развитии программирования для Apple:

**Objective-C**: В течение многих лет Objective-C был основным языком программирования для разработки приложений под платформы Apple. Он предлагал широкие возможности для создания приложений, особенно на iOS и macOS.

**Swift**: В 2014 году Apple представила Swift - свой собственный язык программирования. Swift был разработан для упрощения процесса разработки, повышения безопасности кода и увеличения производительности. Он стал альтернативой Objective-C и быстро стал популярным среди разработчиков.

**Xcode**: Xcode - это интегрированная среда разработки (IDE) от Apple, которая предоставляет разработчикам инструменты для создания приложений под платформы Apple. Она поддерживает языки программирования Objective-C и Swift, а также предлагает набор инструментов для разработки интерфейсов, отладки, тестирования и развертывания приложений.

**App Store**: App Store - это центральный магазин приложений для устройств Apple. Разработчики могут создавать и публиковать приложения в App Store, чтобы пользователи могли легко находить, загружать и устанавливать их на свои устройства.

**UIKit и SwiftUI**: UIKit - это фреймворк для создания пользовательских интерфейсов приложений под iOS и tvOS. Он предлагает набор инструментов для создания интерфейсов, обработки событий и управления жизненным циклом приложений. SwiftUI - это новый фреймворк, представленный Apple в 2019 году, который позволяет создавать интерфейсы приложений с использованием декларативного подхода.

Apple продолжает инвестировать в разработку программного обеспечения и инструментов для программистов, чтобы облегчить процесс создания качественных и инновационных приложений под их платформы.

## История Swift и почему его используют для разработки на Apple устройствах

Swift - это язык программирования, разработанный компанией Apple, который широко используется для разработки приложений для операционных систем iOS, macOS, watchOS и tvOS. История Swift начинается в 2010 году, когда разработчики Apple решили создать новый язык программирования, заменяющий Objective-C.

Официально Swift был представлен на конференции разработчиков Apple в июне 2014 года. Он был создан с целью упростить и улучшить процесс разработки приложений, сделать его более безопасным и эффективным.

Важными особенностями Swift являются:

* Безопасность: Swift предлагает множество механизмов безопасности, которые помогают избежать распространенных ошибок программирования, таких как доступ к нулевому указателю или переполнение буфера.
* Простота использования: Swift предлагает более простый и интуитивный синтаксис по сравнению с Objective-C, что делает его более доступным для новых разработчиков. Он также поддерживает множество современных функций, таких как автоматическое управление памятью, инференцию типов и функциональное программирование, что упрощает разработку и повышает производительность
* Совместимость с Objective-C: Swift полностью совместим с кодом, написанным на Objective-C, и позволяет использовать существующие библиотеки и инструменты.
* Высокая производительность: Swift оптимизирован для работы на платформах Apple и обладает высокой производительностью. Swift работает на прямом уровне с системными библиотеками Apple и использует оптимизацию компилятора LLVM, что делает его очень быстрым языком. Он требует меньше времени для выполнения операций, что позволяет создавать быстрые и отзывчивые приложения.
* Модерн: Swift предлагает множество современных функций, таких как типы данных с неявными значениями, функциональное программирование, обработка ошибок с помощью конструкции try-catch и многое другое.

В результате Swift стал очень популярным языком программирования для разработки приложений для iOS. Он предлагает разработчикам более простой и безопасный способ создания приложений, а также обладает современным и удобным синтаксисом. Он предлагает разработчикам мощный и удобный инструмент для создания современных приложений для Apple-устройств.

Swift является популярным языком программирования для разработки приложений для устройств Apple. Вот некоторые из самых популярных приложений, созданных на Swift:

1. VSCO - это популярное приложение для редактирования фотографий, которое было создано на Swift. Оно предлагает широкий набор инструментов для обработки и редактирования изображений, позволяя пользователям создавать профессионально выглядящие фотографии.
2. LinkedIn - это крупная социальная сеть для профессиональных контактов. Мобильное приложение LinkedIn для iOS также было создано на Swift. Оно предлагает пользователям возможность создавать профили, искать вакансии, общаться с коллегами и расширять свою профессиональную сеть.
3. Yahoo! Weather: Приложение Yahoo! Weather предоставляет актуальную информацию о погоде. Оно было разработано на Swift и предлагает пользователям подробные прогнозы погоды по всему миру, а также красивые фотографии, соответствующие текущим погодным условиям.
4. Clear - это популярное приложение для управления задачами и списками дел. Оно предлагает простой и интуитивно понятный интерфейс, который был разработан на Swift. Пользователи могут создавать, редактировать и отслеживать свои задачи в удобной и организованной форме.
5. Instagram: Одно из самых популярных приложений для обмена фотографиями и видео.
6. Twitter: Социальная сеть для обмена короткими сообщениями и новостями.
7. Slack: Платформа для коммуникации и организации рабочих групп.
8. Pinterest: Социальная сеть для хранения и обмена изображениями и идеями.

Это только некоторые из множества приложений, разработанных на Swift.

## Фрейморк UIKit

UIKit - это фреймворк, предоставляемый Apple для разработки пользовательского интерфейса (UI) в приложениях для операционной системы iOS. Он является одним из основных фреймворков для создания мобильных приложений на платформе iOS с использованием языка программирования Swift.

UIKit предоставляет множество классов и компонентов, которые облегчают создание интерактивных и привлекательных пользовательских интерфейсов. Некоторые из основных классов и компонентов, предоставляемых UIKit, включают:

UIView: Базовый класс для отображения графических элементов на экране. Он предоставляет функциональность для управления размерами, положением и отображением элементов интерфейса.

UIButton: Класс для создания кнопок, которые пользователи могут нажимать для выполнения определенных действий.

UILabel: Класс для отображения текста на экране. Он позволяет настраивать шрифт, цвет и выравнивание текста.

UIImageView: Класс для отображения изображений на экране. Он позволяет загружать и отображать изображения из локальных ресурсов или из сети.

UITableView: Класс для отображения списков данных, которые можно прокручивать вертикально. Он предоставляет функциональность для отображения и настройки каждой строки в таблице.

UICollectionView: Класс для отображения коллекций данных, которые могут быть организованы в виде сетки или пользовательских макетов. Он предоставляет более гибкий способ отображения данных, чем UITableView.

Это только небольшой обзор некоторых классов и компонентов, предоставляемых UIKit. UIKit также предлагает множество других возможностей, таких как обработка жестов, анимации, работа с клавиатурой и многое другое. Он является основным инструментом для разработки пользовательского интерфейса в приложениях для iOS, и знание UIKit является важным для создания качественных и удобных приложений.

## Storyboard или SwiftUI

Storyboard - это графический интерфейсный файл в среде разработки Xcode, который позволяет визуально создавать и организовывать пользовательский интерфейс для приложений iOS с использованием языка программирования Swift.

Storyboard предоставляет удобный способ создания и настройки экранов (View Controllers), а также переходов между ними. Он позволяет разработчикам визуально располагать элементы интерфейса, такие как кнопки, метки, изображения и другие, на экране. Это упрощает процесс создания и настройки пользовательского интерфейса, так как не требуется вручную кодировать каждый элемент и его расположение.

В Storyboard можно создавать различные сцены (Scenes), которые представляют собой отдельные экраны в приложении. Каждая сцена содержит View Controller, который управляет логикой и поведением этого экрана. Разработчики могут легко создавать переходы между сценами, определять анимации переходов и настраивать различные свойства элементов интерфейса.

Storyboard также позволяет использовать возможности Auto Layout для создания адаптивного интерфейса, который будет корректно отображаться на разных устройствах с разными размерами экранов.

Однако, с развитием разных подходов к разработке пользовательского интерфейса, таких как SwiftUI, которая предоставляет альтернативный способ создания интерфейса без использования Storyboard, некоторые разработчики могут предпочитать использовать другие методы для создания интерфейса в своих проектах.

В целом, Storyboard является мощным инструментом для создания и настройки пользовательского интерфейса в приложениях iOS с использованием Swift. Он облегчает процесс разработки и позволяет разработчикам визуально представлять и настраивать интерфейс своего приложения.

SwiftUI - это современный декларативный фреймворк разработки пользовательского интерфейса для создания приложений iOS, macOS, watchOS и tvOS с использованием языка программирования Swift. SwiftUI был представлен Apple в 2019 году и предлагает новый подход к созданию интерфейса, основанный на декларативном программировании.

Основные преимущества SwiftUI:

* Декларативный подход: SwiftUI позволяет описывать пользовательский интерфейс в виде декларативного кода, который описывает, как должен выглядеть интерфейс, а не как его создать шаг за шагом. Это упрощает понимание и поддержку кода, а также позволяет разработчикам создавать интерфейс более эффективно.
* Универсальность: SwiftUI позволяет создавать приложения для разных платформ, таких как iOS, macOS, watchOS и tvOS, используя общий код и компоненты. Это упрощает разработку и поддержку приложений для разных устройств и платформ.
* Автоматическое обновление интерфейса: SwiftUI автоматически обновляет интерфейс при изменении состояния данных, что позволяет создавать реактивные и интерактивные пользовательские интерфейсы. Это упрощает управление состоянием и обновлением пользовательского интерфейса.
* Множество встроенных компонентов: SwiftUI предлагает богатый набор встроенных компонентов для различных элементов интерфейса, таких как кнопки, метки, списки, формы и многое другое. Это упрощает создание и настройку интерфейса без необходимости вручную создавать каждый элемент.
* Разделение интерфейса и логики: SwiftUI отделяет пользовательский интерфейс от логики приложения, позволяя разработчикам сосредоточиться на каждом из аспектов отдельно. Это упрощает понимание и поддержку кода, а также делает его более модульным и переиспользуемым.

SwiftUI предлагает множество других функций и возможностей, таких как анимации, градиенты, стили и многое другое. Он становится все более популярным среди разработчиков и является рекомендуемым подходом для создания пользовательского интерфейса в приложениях Apple.

Однако, важно отметить, что SwiftUI все еще относительно новый фреймворк, и некоторые функции и возможности могут быть ограничены или требовать дополнительной настройки. Кроме того, для поддержки более старых версий iOS может потребоваться использование классического подхода с использованием фреймворка UIKit.

В целом, SwiftUI представляет собой современный и мощный фреймворк для создания пользовательского интерфейса в приложениях Apple. Он предлагает простой и эффективный способ создания интерфейса и позволяет создавать более реактивные и модульные приложения.

Вывод: Storyboard и SwiftUI - два разных подхода к созданию пользовательского интерфейса в приложениях iOS.

Storyboard представляет собой графический интерфейсный файл в Xcode, который позволяет визуально создавать и настраивать пользовательский интерфейс. Он был основным инструментом для создания интерфейса в iOS до появления SwiftUI. Storyboard позволяет разработчикам визуально располагать элементы интерфейса на экране и определять переходы между экранами. В Storyboard используется UIKit, основной фреймворк для создания пользовательского интерфейса в iOS.

SwiftUI представляет собой современный декларативный подход к созданию пользовательского интерфейса в iOS. Он использует декларативный синтаксис, который позволяет описывать интерфейс приложения в виде структуры кода. SwiftUI является более новым и рекомендуемым подходом для создания интерфейса в iOS. SwiftUI предлагает множество удобных функций и возможностей, таких как автоматическое обновление интерфейса при изменении данных и многое другое.

Вопрос о том, что лучше - Storyboard или SwiftUI, зависит от ваших предпочтений и требований проекта. Storyboard может быть более знакомым для разработчиков, которые имеют опыт работы с UIKit, и может быть полезным для простых приложений с простым пользовательским интерфейсом. SwiftUI, с другой стороны, предлагает более современный и гибкий подход, особенно для более сложных и интерактивных пользовательских интерфейсов.

Оба подхода имеют свои преимущества и недостатки, и выбор между ними зависит от ваших личных предпочтений и требований проекта. Некоторые разработчики могут предпочитать использовать SwiftUI, чтобы они могли использовать его преимущества и новейшие функции, тогда как другие могут предпочитать оставаться с Storyboard из-за привычки или потребности в совместимости с более старыми версиями iOS.

В конечном итоге, выбор между Storyboard и SwiftUI зависит от личных предпочтений, опыта и требований проекта. В проекте мной был выбран именно Storyboard, потому что мне было интересно поработать именно с графическим интерфейсом.

## UITest и Unit-тесты

UITest и unit-test - это два разных типа тестирования в Swift.

Unit-тесты (unit-test) - это тестирование отдельных модулей, классов или функций в изоляции от других компонентов приложения. Они предназначены для проверки правильности работы отдельных частей кода, например, функций или методов. Unit-тесты позволяют разработчикам убедиться, что каждая часть кода выполняет свою работу корректно и соответствует ожидаемым результатам. Unit-тесты могут быть автоматизированы и запускаться при каждом изменении кода для быстрой обратной связи о его работоспособности.

В случае Swift, unit-тесты могут быть написаны с использованием фреймворка XCTest, встроенного в Xcode. XCTest предоставляет набор инструментов для написания и запуска unit-тестов в Swift. С помощью XCTest вы можете создавать тестовые случаи, определять ожидаемые результаты и проверять, соответствует ли фактический результат ожидаемому.

UITest (UI-тест) - это тестирование пользовательского интерфейса (UI) приложения. UITest позволяет симулировать пользовательские взаимодействия с приложением, такие как нажатия на кнопки, ввод текста и проверку отображаемых элементов интерфейса. UITest позволяет проверить, что пользовательский интерфейс ведет себя ожидаемым образом и отвечает на взаимодействия пользователя правильно. UITest также может использоваться для проверки навигации в приложении и обработки ошибок. UITest используется для проверки правильности работы пользовательского интерфейса, навигации в приложении и обработки ошибок. В Swift для написания UITest можно использовать фреймворк XCTest, встроенный в Xcode.

Таким образом, основное отличие между UITest и unit-test заключается в том, что unit-тесты тестируют отдельные части кода в изоляции, в то время как UITest тестирует пользовательский интерфейс и взаимодействие пользователя с приложением. Оба типа тестирования важны для создания качественного приложения и обеспечения его надежной работы.

# Глава II Практическая часть. Приложение редактор фотографий

Само приложение представляет собой фильтр для фотографий. Пользователь выбирает фотографию на своем смартфоне, выбирает различные фильтры для нее и редактирует фотографию.

При запуске приложение есть главный экран, на котором есть две кнопки:

* <+> - кнопка, чтобы добавить фотографию или сделать фотография из приложения.
* <редактирование> - при клике на эту кнопку, пользователь переходит на новый экран, где может наложить различные фильтры.

Фильтры накладываются поверх загруженной фотографии, тем самым создавая эффект с различными цветовыми решениями.

Так же в приложении является очень важным, что мы запрашиваем права на получение доступа к галерее пользователя и к его камере. Для этого в файле Info.plist были добавлены дополнительные поля: Privacy – Photo Library User Description (использование фото библиотеки) и Privacy – Camera User Description (использование камеры). Это ключи, которые используются в файле Info.plist для запроса разрешений на использование камеры и фото библиотеки соответственно.

Приложения в iOS должны запросить разрешение пользователя, прежде чем они смогут получить доступ к камере или фото библиотеке на устройстве. Для этого они должны предоставить ясное объяснение пользователю о том, зачем им нужен доступ к камере или фото библиотеке.

Когда пользователь впервые открывает приложение и пытается использовать функции, связанные с камерой или фото библиотекой, iOS отображает системный диалоговый окно, в котором показывается объяснение, которое было указано в соответствующих ключах в Info.plist. Пользователь может разрешить или запретить доступ к камере или фото библиотеке в приложении.

Если бы мы не предоставляли эти ключи и соответствующие объяснения, тогда приложение не сможет получить доступ к камере или фото библиотеке на устройстве пользователя.

Главная страница интерфейса приложения создавался в main.storyboard и выглядит как показано на рисунке (Приложение 1). Были добавлены две кнопки – Button, для добавления фотографий и редактирования. А также ImageView – куда помещается фотография.

При нажатии на кнопку редактирования происходит переход на новый экран (Приложение 2). На котором расположены пять кнопок, при нажатии на которые добавляются различные фильтры на фото. А также сама фотография.

## Класс ImagePicker

Механизм выбора фотографий из фотопленки осуществлен следующим образом: получаем изображение из библиотеки фотографий или непосредственно с камеры с помощью класса ImagePicker. Это небольшой вспомогательный класс, экземпляр которого мы создаем в нашем приложении для выполнения своей работы. Класс ImagePicker, предоставляет функциональность для выбора изображений из галереи или с камеры на устройстве.

В начале кода мы импортируем библиотеки Foundation и UIKit, которые предоставляют базовые функциональности и пользовательский интерфейс соответственно (рис. 1).

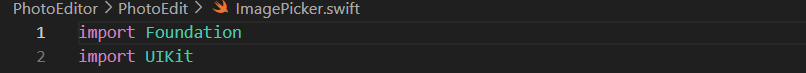


рис. 1 Импорт библиотек

Этот класс объявляет протокол ImagePickerDelegate, который содержит один метод didSelect(image:), принимающий в качестве аргумента изображение типа UIImage. Класс, использующий ImagePicker, должен реализовать этот метод для обработки выбранного изображения (рис. 2).

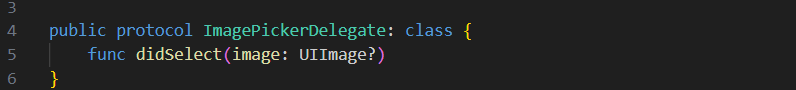


рис. 2 Протокол ImagePickerDelegete

Класс ImagePicker является открытым и наследуется от NSObject, что позволяет использовать его в Swift-приложениях(рис. 3).

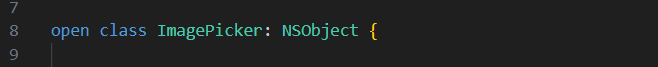


рис. 3 Класс ImagePicker

В этом классе объявлены три приватных свойства: pickerController, presentationController и delegate. pickerController представляет собой экземпляр UIImagePickerController, который будет использоваться для выбора изображений. presentationController - слабая ссылка на UIViewController, который будет использоваться для отображения контроллера выбора изображения. delegate - слабая ссылка на объект, реализующий протокол ImagePickerDelegate. (рис. 4)

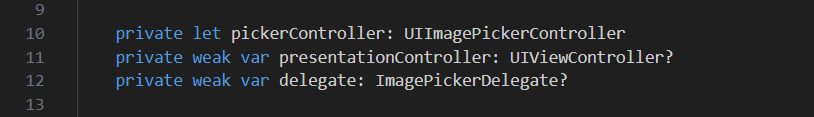


рис. 4 Переменные класса ImagePicker

В инициализаторе класса ImagePicker мы инициализируем pickerController, устанавливаем presentationController и delegate, а также настраиваем pickerController для выбора и редактирования изображений. Здесь мы также устанавливаем делегата pickerController как сам ImagePicker. (рис. 5)

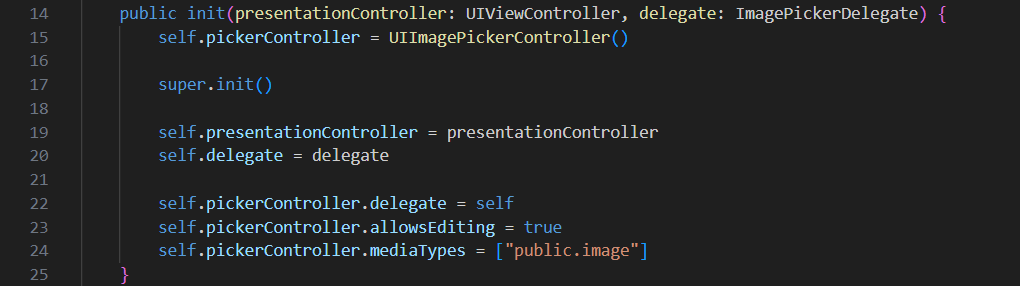


рис. 5 Инициализатор класса ImagePicker

Приватная функция action возвращает UIAlertAction для заданного типа и заголовка. Мы проверяем, доступен ли тип источника изображения, и если да, создаем действие, которое устанавливает тип источника для pickerController и отображает его.(рис. 6)

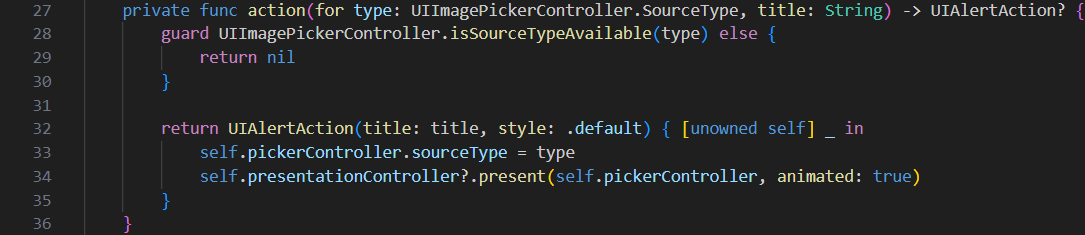


рис. 6 Функция action в классе ImagePicker

Функция percent представляет действие, которое вызывает показ Action Sheet с различными вариантами для захвата или выбора фотографий. Давайте разберемся, что делает каждая часть метода (рис. 7). Этот функция отображает действия пользователя в виде всплывающего окна (UIAlertController) на экране. Функция является публичной и может быть вызвана извне класса или структуры. Она принимает один параметр sourceView типа UIView. sourceView - это представление, из которого должен быть показан Action Sheet.

Сначала создается экземпляр UIAlertController с неизвестным заголовком и сообщением и с заданным стилем .actionSheet, который представляет собой всплывающее окно с кнопками для выполнения действий. UIAlertController используется для отображения предупреждений или Action Sheet пользователю. В данном случае, preferredStyle установлен в .actionSheet, что означает, что предупреждение будет показано как Action Sheet.

Затем метод добавляет различные действия в Action Sheet, основываясь на доступных вариантах, проверяется, есть ли доступные варианты действий для камеры, сохраненных фотографий и библиотеки фотографий с помощью метода self.action(for:title:), который возвращает действие (UIAlertAction) для указанного типа медиа с заданным заголовком. Если действие доступно, оно добавляется в UIAlertController.

Также добавляется кнопка "Cancel", которая просто закрывает UIAlertController.

Если текущее устройство является iPad, то устанавливаются свойства popoverPresentationController для UIAlertController, чтобы определить источник отображения и позволить всплывающему окну указывать стрелку вверх и вниз. Иначе, всплывающее окно будет отображаться в центре экрана. Это гарантирует, что Action Sheet будет показано как Popover на iPad, с sourceView в качестве точки якоря.

Наконец, вызывается метод present(\_:animated:), чтобы отобразить UIAlertController на экране. self.presentationController отвечает за отображение представления в текущем контексте, например, в UINavigationController или UIViewController.

Функция pickerController является приватной (private) и имеет следующую сигнатуру: private func pickerController(\_ controller: UIImagePickerController, didSelect image: UIImage?). Эта функция выполняет определенные действия после выбора изображения из UIImagePickerController. Сначала функция вызывает метод dismiss(animated:completion:) для закрытия UIImagePickerController. Этот метод закрывает контроллер выбора изображения с анимацией (если параметр animated установлен в true). Параметр completion позволяет указать дополнительные действия, которые должны быть выполнены после закрытия контроллера. В данном случае, completion установлен в nil, что означает, что после закрытия контроллера никакие дополнительные действия не выполняются. Затем функция вызывает метод didSelect(image:) у делегата (delegate). Эта функция выполняет две основные операции: закрытие контроллера выбора изображений и передачу выбранного изображения делегату для дальнейшей обработки.

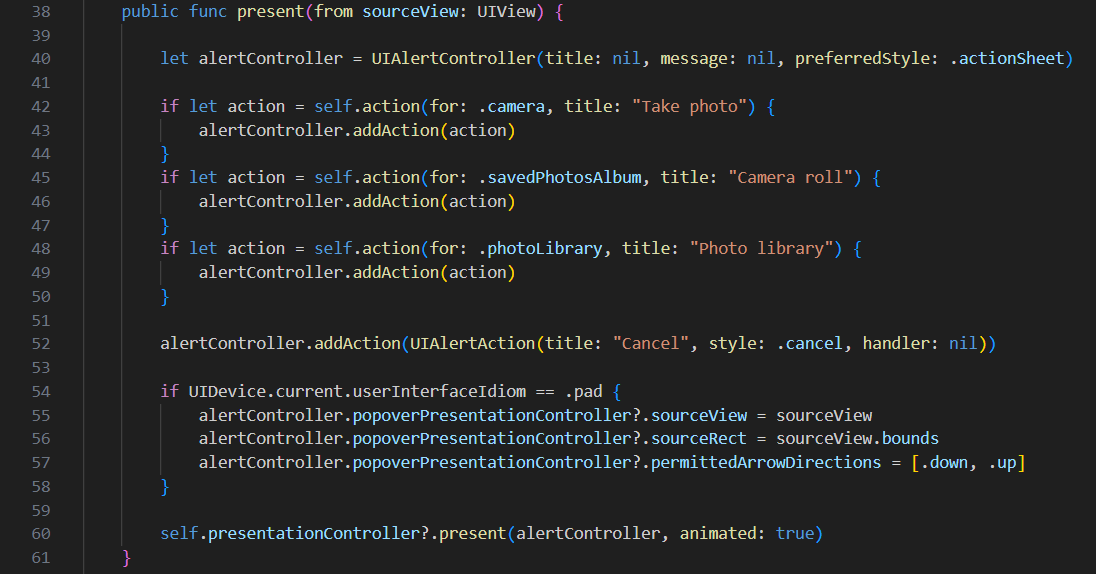


рис. 7 Функция percent в классе ImagePicker

Код extension ImagePicker: UIImagePickerControllerDelegate представляет собой расширение (extension) для класса ImagePicker и реализует протокол UIImagePickerControllerDelegate(рис. 8). Метод imagePickerControllerDidCancel вызывается, когда пользователь отменяет выбор изображения в контроллере выбора изображений. Он передает nil в метод pickerController(\_:didSelect:), чтобы указать, что изображение не было выбрано. Метод imagePickerController вызывается, когда пользователь выбирает изображение в контроллере выбора изображений. Он извлекает выбранное изображение из словаря info, используя ключ .editedImage, и передает его в метод pickerController(\_:didSelect:). Если изображение не может быть извлечено, то передается nil. Код extension ImagePicker: UINavigationControllerDelegate

Данный код представляет собой расширение (extension) для класса ImagePicker и реализует протокол UINavigationControllerDelegate. В данном коде не содержится никаких методов.

Код extension ImagePicker: UINavigationControllerDelegate. Данный код представляет собой расширение (extension) для класса ImagePicker и реализует протокол UINavigationControllerDelegate. В данном коде не содержится никаких методов.

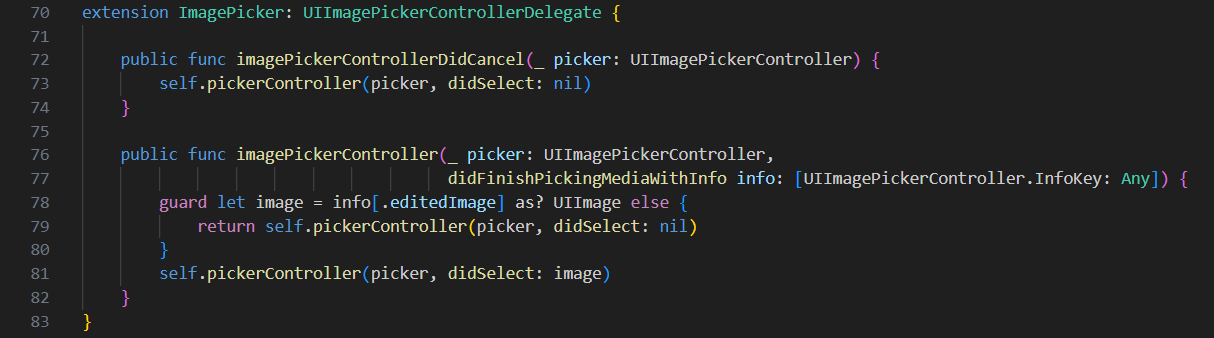


рис. 8 extension для класса ImagePicker

## Класс ViewController

Данный класс на языке Swift импортирует UIKit и содержит реализацию контроллера представления (ViewController) для отображения пользовательского интерфейса и обработки событий. Код import UIKit означает, что класс использует фреймворк UIKit, который предоставляет набор инструментов и классов для создания пользовательского интерфейса в iOS-приложениях. UIKit включает в себя классы для работы с визуальными элементами, такими как кнопки, изображения, текстовые поля, таблицы и другие.

Класс ViewController является подклассом UIViewController, что означает, что он наследует функциональность базового класса UIViewController. Он также реализует протокол UIImagePickerControllerDelegate, который позволяет контроллеру обработать события, связанные с выбором изображений из галереи или камеры.(рис. 9)

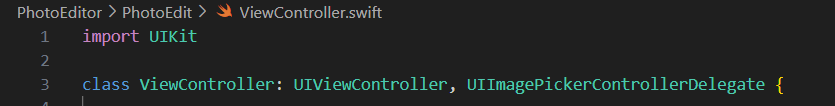


рис. 9 Класс ViewController

В классе ViewController определены аутлеты (IBOutlet) и свойства, которые связываются с элементами пользовательского интерфейса, созданными с использованием Interface Builder. Например, addButton и picture являются аутлетами, связанными с кнопкой и изображением соответственно.(рис.10)



рис.10 Аутлеты

В функции viewDidLoad() происходит инициализация экземпляра класса ImagePicker и его свойства presentationController и delegate. Данный класс используется для выбора изображений из галереи или камеры(рис. 11).

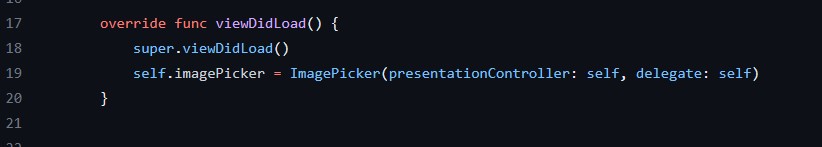


рис.11 Функция viewDidLoad

Также в классе определены две функции: editAction(\_:) и addAction(\_:)(рис. 12). editAction(\_:) открывает экран редактирования изображения, передавая текущее изображение на следующий экран. addAction(\_:) вызывает present(from:) у экземпляра imagePicker, открывая экран выбора изображения.

Функция editAction(\_:) выполняет следующие действия:

1. Создает экземпляр класса UIStoryboard с именем "Editor" и без параметра "bundle". UIStoryboard используется для создания экранов пользовательского интерфейса, которые были созданы с использованием Interface Builder.
2. Создает экземпляр класса EditorViewController с помощью метода instantiateViewController(identifier:) экземпляра storyboard. Этот метод создает экземпляр контроллера представления (View Controller) с заданным идентификатором "Editor".
3. Присваивает свойству picture экземпляра c изображение, которое хранится в свойстве picture.image текущего контроллера представления.
4. Вызывает метод pushViewController (\_:animated:)  у свойства  navigationController  текущего контроллера представления, чтобы открыть экран c с анимацией.

Функция addAction(\_:) выполняет следующие: вызывает метод present(from:) у свойства imagePicker текущего контроллера представления, передавая в качестве аргумента sender (как UIView). Это открывает экран выбора изображения, который позволяет пользователю выбрать изображение из галереи или сделать новое с помощью камеры.



рис. 12 Функции editAction и addAction

В расширении класса ViewController реализуется протокол ImagePickerDelegate, который содержит метод didSelect(image:)(рис. 13). Эта функция вызывается, когда пользователь выбирает изображение из галереи или камеры, и устанавливает выбранное изображение в свойство picture.image для отображения на экране.

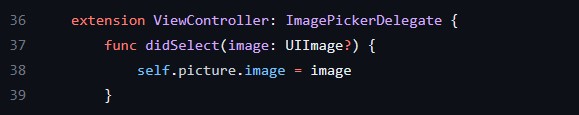


рис.13 расширении класса ViewController

## Класс EditorViewController

Класс EditorViewController является подклассом UIViewController и имеет следующие свойства и методы:

* filterView: слабая ссылка (weak) на экземпляр класса UIView. Это свойство представляет представление (view), которое используется для отображения фильтра внутри контроллера представления.
* picture: публичное свойство типа UIImage?. Это свойство используется для хранения изображения, которое будет отображаться в imageView.
* imageView: слабая ссылка (weak) на экземпляр класса UIImageView. Это свойство представляет представление (view), которое используется для отображения изображения.
* viewDidLoad(): переопределенный метод, вызывается после загрузки контроллера представления в память. В этом методе устанавливается изображение picture в imageView и устанавливается начальный фоновый цвет filterView.
* action1(\_:), action2(\_:), action3(\_:), action4(\_:), action5(\_:): обработчики событий (IBAction), вызываются при нажатии на соответствующие кнопки или элементы пользовательского интерфейса. В каждом из этих методов устанавливается фоновый цвет filterView в соответствии с определенным фильтром (UIColor.Filter).

Класс EditorViewController импортирует фреймворк UIKit с помощью выражения import UIKit. Фреймворк UIKit предоставляет набор инструментов для создания графического пользовательского интерфейса на устройствах Apple, включая представления, контроллеры представлений, графические элементы интерфейса и многое другое.

Функция viewDidLoad() является методом переопределения (override) из класса UIViewController и вызывается после того, как контроллер представления загружается в память(рис. 14). В этой функции выполняются следующие действия:

1. Вызывается метод super.viewDidLoad(), чтобы выполнить базовую реализацию метода viewDidLoad() из родительского класса UIViewController.
2. Устанавливается изображение picture в свойство image объекта imageView. Это означает, что изображение picture будет отображаться в представлении imageView.
3. Устанавливается фоновый цвет filterView с использованием UIColor(red: 0, green: 0, blue: 0, alpha: 0). В данном случае, это черный цвет с прозрачностью 0, что означает, что filterView будет невидимым.

Эта функция выполняется при загрузке контроллера представления и используется для инициализации и настройки элементов пользовательского интерфейса, установки значений по умолчанию и других действий, которые должны быть выполнены перед отображением экрана пользователю.



рис. 14 Класс EditorViewController и функция viewDidLoad

## Класс UIColor+Extra

В классе UIColor+Extra объявлено расширение UIColor и в нем определяется вложенная структура Filter(рис.15).

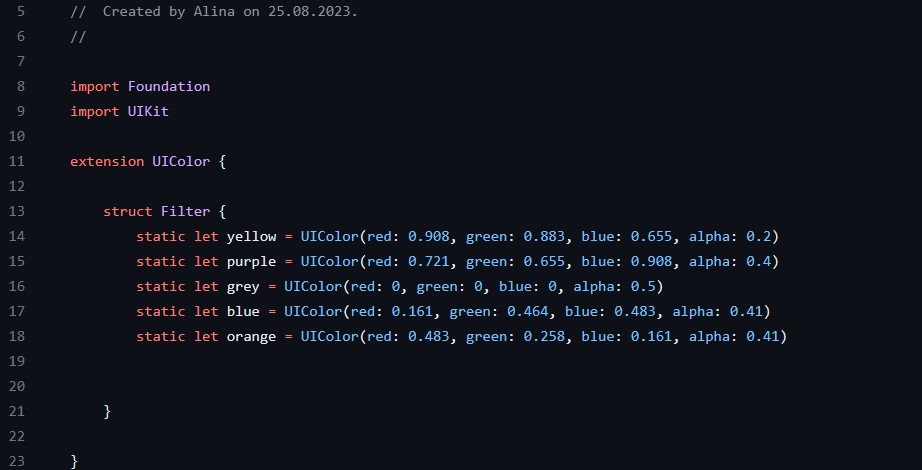


рис. 15 Класс UIColor+Extra

Внутри структуры Filter определены следующие статические свойства типа UIColor:

* yellow: Желтый цвет с определенными значениями красного, зеленого, синего и прозрачности.
* purple: Фиолетовый цвет с определенными значениями красного, зеленого, синего и прозрачности.
* grey: Серый цвет с определенными значениями красного, зеленого, синего и прозрачности.
* blue: Синий цвет с определенными значениями красного, зеленого, синего и прозрачности.
* orange: Оранжевый цвет с определенными значениями красного, зеленого, синего и прозрачности.

Расширение UIColor позволяет добавить новые свойства или методы к существующему типу UIColor без необходимости изменения исходного кода. В данном случае, расширение UIColor используется для определения набора цветов, которые могут быть использованы в контексте фильтров или других задач в приложении.

## Assets и иконка приложения

// тут добавить

## Тестирование

ViewControllerUITests.swift

UI-тестирование:

Unit:

Этот код представляет собой набор тестов (ViewControllerTests) для проверки функциональности ViewController в iOS-приложении.

Определяется класс ViewControllerTests, который наследуется от XCTestCase (базового класса для написания тестов в XCTest).

Внутри ViewControllerTests определена переменная viewController типа ViewController!. Эта переменная будет использоваться для инстанцирования ViewController, который будет тестироваться.

В методе setUp() (вызывается перед каждым тестом) происходит загрузка экземпляра ViewController из основной сториборда. Затем вью контроллер загружается (если ещё не был загружен) вызовом loadViewIfNeeded().

В методе tearDown() (вызывается после каждого теста) происходит очистка переменной viewController.

Следующий метод testEditAction() представляет собой тест для проверки функции editAction() в ViewController.

В блоке "Given" создается MockNavigationController, который используется как корневой контроллер окна приложения.

Затем следует блок "When", где вызывается метод editAction() на экземпляре viewController.

В блоке "Then" проверяется, что navigationController перешел к экземпляру EditorViewController. Если это не так, то тест не пройден.

Вы также можете добавить дополнительные утверждения (assertions), чтобы проверить другие аспекты поведения приложения.

После этого идут дополнительные классы-моки (MockNavigationController и MockImagePicker), которые используются внутри теста.

MockNavigationController имитирует поведение настоящего навигационного контроллера. Он переопределяет метод pushViewController(\_:animated:) для сохранения переданного контроллера в свойство pushedViewController.

MockImagePicker также является подклассом, представляющим фейковый ImagePicker. Он переопределяет метод present(from:) для установки значения presented в true.

Таким образом, эти тесты позволяют автоматизировать проверку функциональности ViewController, включая навигацию между экранами и взаимодействие с фейковым ImagePicker.

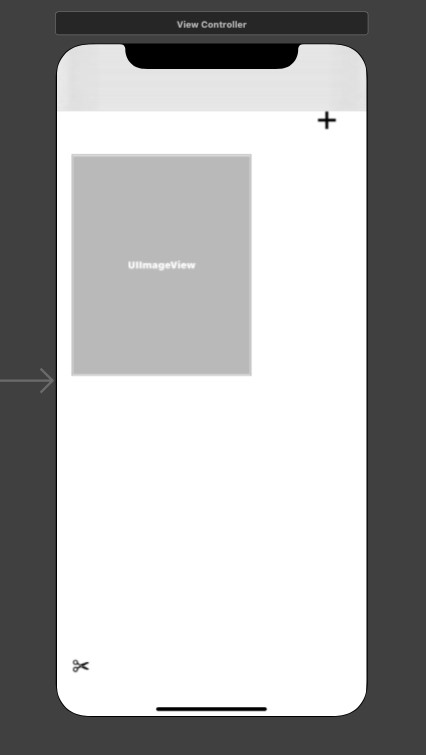
# Заключение

# Список использованной литературы

1. Picker Images whith UIImagePickerController in Swift 5 <https://theswiftdev.com/picking-images-with-uiimagepickercontroller-in-swift-5/>

# Приложения

Приложение 1



Приложение 2

