Министерство науки и высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»

ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

На правах рукописи

УДК 004.75

БУЛЬДИН ИЛЬЯ ДМИТРИЕВИЧ

СОЗДАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ SIP-ТЕЛЕФОНИИ НИЯУ МИФИ

Выпускная квалификационная работа бакалавра

Направление подготовки 09.03.01 Информатика и вычислительная техника

|  |
| --- |
| Выпускная квалификационная работа защищена  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г. |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Секретарь ГЭК \_\_\_\_\_\_\_\_\_\_\_\_\_ |

г. Москва

2023

Студент-дипломник \_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Бульдин И.Д. /

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Романов Н.Н. /

Рецензент \_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Овчаренко Е.С. /

Заведующий кафедрой №12 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Иванов М.А. /

**АННОТАЦИЯ**

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc122817728)

[1 Обзорная часть 7](#_Toc122817729)

[1.1 Описание языков программирования для разработки iOS приложений 7](#_Toc122817730)

[1.2 Использование императивного и декларативного подхода при разработке iOS приложений 9](#_Toc122817731)

[1.3 Выбор архитектурного паттерна 11](#_Toc122817732)

[2 Теоретическая часть 13](#_Toc122817733)

[2.1 SIP-телефония 13](#_Toc122817734)

[2.2 Анализ конкурентов и определение требований к системе 14](#_Toc122817735)

[2.2.1 Анализ существующих решений от конкурентов 14](#_Toc122817736)

[2.2.2 Определение требований к мобильному приложению 15](#_Toc122817737)

[2.3 Архитектура мобильного приложения 17](#_Toc122817738)

[ПРИЛОЖЕНИЕ А. Листинг кода, демонстрирующий работу паттерна Coordinator 19](#_Toc122817739)

# **ВВЕДЕНИЕ**

В наше время мобильные телефоны стали неотъемлемой частью нашей жизни, и возможность общения с помощью интернета стала доступна для большинства людей. SIP-телефония представляет собой технологию, которая позволяет осуществлять звонки через интернет.

Таким образом, актуальной научной задачей является изучение существующих решений в области создания мобильных приложений SIP-телефонии, анализ их функциональности и разработка мобильного приложения SIP-телефонии для операционной системы iOS, которое будет удовлетворять требованиям современной SIP-телефонии, учитывая специфику использования внутри НИЯУ МИФИ.

Целью работы является разработка мобильного приложения SIP-телефонии, которое будет обладать необходимым функционалом, удобным интерфейсом, высокой степенью надежности и предоставлять сотрудникам удобный инструмент для общения с помощью SIP-телефонии.

Для достижения поставленной цели необходимо решить следующие задачи:

* Изучение существующих решений в области мобильных приложений SIP-телефонии, анализ функциональности и возможностей этих приложений;
* Выбор технологий и инструментов, которые должны использоваться при разработке;
* Формулирование требований к разрабатываемому ПО;
* Проектирование UX и UI;
* Разработка архитектуры и интерфейса мобильного приложения, учитывая требования современной SIP-телефонии и особенности использования на мобильных устройствах с операционной системой iOS;
* Реализация функциональности мобильного приложения, обеспечивающей выполнение основных сформулированных требований;
* Тестирование разработанного ПО (локальное и с помощью службы бета-тестирования iOS приложений TestFlight)

Был проведен анализ существующих решений и разработана архитектура приложения. Для решения поставленной задачи были использованы современные технологии и инструменты, такие как Swift, UIKit, SnapKit и среда разработки Xcode, а также Abto VoIP SIP SDK для удобной работы с SIP-протоколом.

В рамках данной работы было разработано мобильное приложение, которое позволяет осуществлять звонки и общаться с помощью SIP-протокола. В приложении реализованы функции, такие как авторизация, каталог абонентов НИЯУ МИФИ, добавление контактов в раздел «Избранное», Multi Accounting, возможность осуществления звонков и т.д. Разработанное приложение прошло тестирование и отладку, что позволило убедиться в его стабильной работе и функциональности.

Разработка мобильного приложения SIP-телефонии для платформы iOS является актуальной и перспективной задачей. Одним из основных преимуществ использования SIP-телефонии является экономия средств на телефонные разговоры. Также SIP-телефония обеспечивает гибкость в настройке телефонной сети и возможность удаленной работы.

# **1 Обзорная часть**

## **1.1 Описание языков программирования для разработки iOS приложений**

Язык программирования является важным инструментом для разработки программного обеспечения, в том числе и мобильных приложений. При разработке iOS приложений существует несколько языков, которые могут быть использованы. В данной главе рассмотрены основные языки, которые можно использовать для разработки iOS приложений, их особенности, возможности и ограничения. Также приведено сравнение этих языков, обоснован выбор наиболее подходящего.

Для разработки iOS приложений можно использовать следующие языки программирования:

* Swift – язык программирования, созданный Apple для разработки iOS, macOS, watchOS и tvOS. Он является официальным языком разработки приложений для Apple и предлагает множество удобных функций и возможностей для разработки;
* Objective-C – язык программирования, который был ранее использован как основной язык разработки приложений для iOS, но сейчас уступает место Swift. Objective-C является объектно-ориентированным языком и использует динамическую типизацию;
* Kotlin/Native – язык программирования, разработанный компанией JetBrains, который позволяет написать кросс-платформенное приложение одной кодовой базой и скомпилировать его в нативный код для iOS;
* C++ – мощный язык программирования, который может использоваться для разработки iOS приложений с помощью таких инструментов как как Clang и LLVM. Однако, так как C++ является низкоуровневым языком, его использование требует большего опыта и знаний, чем более высокоуровневые языки, такие как Swift или Objective-C. В дополнение к этому, C++ поддерживается Apple неофициально, что может привести к нестабильной работе и проблемам с компиляцией;
* Другие языки, такие как Java и Python, не являются официально поддерживаемыми Apple и могут быть очень сложными для использования

Разработка кросс-платформенного приложения не рассматривается организацией. В целом, для нативной разработки iOS приложений рекомендуется использовать Swift или Objective-C.

Swift является более современным и удобным в использовании языком программирования, чем Objective-C. Он имеет синтаксис с высокой читабельностью, использует статическую типизацию, безопасен и предлагает больше современных функций и возможностей, чем Objective-C. Кроме того, Swift быстрее и эффективнее, чем Objective-C, благодаря использованию оптимизированных структур данных и алгоритмов. Поэтому для разработки мобильного приложения SIP-телефонии НИЯУ МИФИ был выбран язык программирования Swift.

## **1.2 Использование императивного и декларативного подхода при разработке iOS приложений**

UIKit и SwiftUI – это два разных фреймворка, используемых для создания пользовательского интерфейса в iOS-приложениях.

UIKit является стандартным фреймворком для разработки пользовательского интерфейса в iOS с момента выпуска первой версии iOS. Он предоставляет большое количество готовых элементов интерфейса, таких как кнопки, таблицы, коллекции и т.д., а также множество инструментов для управления их взаимодействием и анимацией. Однако UIKit требует большого количества кода для реализации даже простых интерфейсов, что может усложнить разработку и увеличить вероятность ошибок.

SwiftUI – это новый фреймворк, введенный Apple в 2019 году. Он предназначен для создания пользовательского интерфейса с помощью декларативного языка описания, который позволяет описывать интерфейс в терминах его состояний и поведения, а не в терминах низкоуровневых действий, которые нужно выполнить. Это позволяет создавать интерфейсы быстрее и проще, так как не нужно писать много кода для реализации отдельных элементов интерфейса.

В целом, оба фреймворка могут быть использованы для создания пользовательского интерфейса в iOS-приложениях, SwiftUI является более современным и удобным в использовании из-за его декларативного подхода и автоматической обработки изменений состояния. Однако предпочтение при разработке было отдано UIKit.

Далее описаны причины выбора императивного подхода для создания интерфейса iOS приложения:

* Более ранняя поддержка. UIKit уже был в iOS гораздо раньше, чем SwiftUI, поэтому он более широко распространен и лучше поддерживается;
* Больше контроля. UIKit предоставляет больше возможностей для настройки и контроля поведения элементов интерфейса. Это может быть полезно при создании таблиц с динамической высотой ячеек в зависимости от количества текста;
* Больше гибкости и возможностей. UIKit предоставляет больше функциональности и возможностей, чем SwiftUI, включая различные элементы интерфейса, анимации и инструменты управления контентом. Это может быть полезно при создании сложных интерфейсов с множеством индивидуальных элементов.

## **1.3 Выбор архитектурного паттерна**

Выбор архитектурного паттерна - важный шаг в разработке приложения для iOS. Архитектурный паттерн определяет структуру приложения и способ взаимодействия его различных частей. Правильный выбор архитектурного паттерна может упростить разработку, улучшить поддержку кода и повысить гибкость программного кода.

Существует несколько популярных архитектурных паттернов, которые можно использовать в приложениях для iOS, таких как MVC (Model-View-Controller), MVP (Model-View-Presenter), MVVM (Model-View-ViewModel) и VIPER. Каждый из этих паттернов имеет свои сильные и слабые стороны, и выбор паттерна зависит от конкретных требований и целей приложения.

Проведено сравнение различных архитектурных паттернов, их ключевые характеристики, чтобы повысить масштабируемость, поддерживаемость и тестируемость программного кода. Разница между MVC, MVP и MVVM несущественна. Controller «знает» и о модели, и о слое представления. Presenter «знает» о модели и зависит от слоя представления. ViewModel похожа на Presenter, но она отвязана от пользовательского интерфейса и слоя данных, потому что преобразует данные, полученные из модели в формат, удобный для слоя представления.

Использование архитектурного паттерна MVС для приложения SIP-телефонии может быть хорошим выбором по нескольким причинам:

* Понятное и удобное разделение логики на изолированные компоненты. Это облегчает внесение изменений и поддержку приложения.
* Возможность переиспользования кода;
* Тестируемость. Разделение задач в MVC облегчает независимое тестирование различных частей приложения, что помогает обеспечить надежность и отсутствие ошибок в приложении;
* Поддерживаемость. Модель MVC облегчает обслуживание приложения с течением времени, поскольку изменения в модели или контроллере не требуют изменений в слое представления, и наоборот

В целом, архитектурный паттерн MVC может помочь сделать разработку приложения SIP-телефонии более эффективной и удобной, что может быть особенно важно для приложения, которое должно быть надежным и отзывчивым. Кроме того, MVC – это широко используемый и устоявшийся архитектурный паттерн, который знаком многим разработчикам, что может облегчить поиск ресурсов и поддерживаемость кода.

# **2 Теоретическая часть**

## **2.1 SIP-телефония**

Телефония с использованием Session Initial Protocol (SIP) – это метод совершения и приема телефонных звонков через Интернет с использованием широкополосного соединения и учетной записи SIP.

SIP – это стандартизированный протокол, используемый для инициирования, поддержания, изменения и завершения сеансов связи в реальном времени, включающих видео, голос, обмен сообщениями и другие виды связи между устройствами. SIP обычно используется в сочетании с другими протоколами, такими как RTP (Real-Time Transport Protocol) и RTCP (Real-Time Transport Control Protocol).

SIP использует сообщения, известные как SIP-requests и SIP-responses, для инициирования и управления сеансами связи. Эти сообщения отправляются между SIP-клиентами, которые представляют собой устройства или программы, использующие SIP, и SIP-серверами, которые представляют собой серверы, маршрутизирующие и управляющие SIP-запросами. SIP часто используется в системах VoIP (Voice Over IP) для установления и управления голосовыми и видеозвонками через Интернет.

Одним из основных преимуществ SIP-телефонии является то, что она позволяет пользователям совершать и принимать звонки из любого места, где есть широкополосное соединение. Это может быть особенно полезно для предприятий и организаций, сотрудники которых работают удаленно или в разных местах.

Существует множество различных приложений для SIP-софтфонов и аппаратных устройств, которые позволяют пользователям использовать преимущества SIP-телефонии. Далее в пункте 2.2 проведен анализ конкурентов и сформулированы требования для разрабатываемой системы.

## **2.2 Анализ конкурентов и определение требований к системе**

### **2.2.1 Анализ существующих решений от конкурентов**

Хотя существует множество мобильных приложений SIP-софтфонов для операционной системы iOS, они не всегда подходят для внутренних звонков. Это связано с тем, что такие приложения, как правило, предназначены для общего пользования, а не адаптированы к специфическим потребностям организации.

Прямым конкурентом является решение от компании Grandstream – GS Wave Lite. Wave Lite – это мобильное приложение, которое позволяет пользователям совершать и принимать телефонные звонки, используя широкополосное соединение и учетную запись SIP.

Одно из преимуществ Wave Lite заключается в том, что оно имеет широкий функционал и большое количество пользовательских настроек. Приложение включает такие функции, как переадресация вызовов, голосовая почта и текстовые сообщения. Еще одним преимуществом Wave Lite является поддержка HD аудио и видео звонков. На рисунке 2.2.1.1 представлен интерфейс мобильного приложения GS Wave Lite.

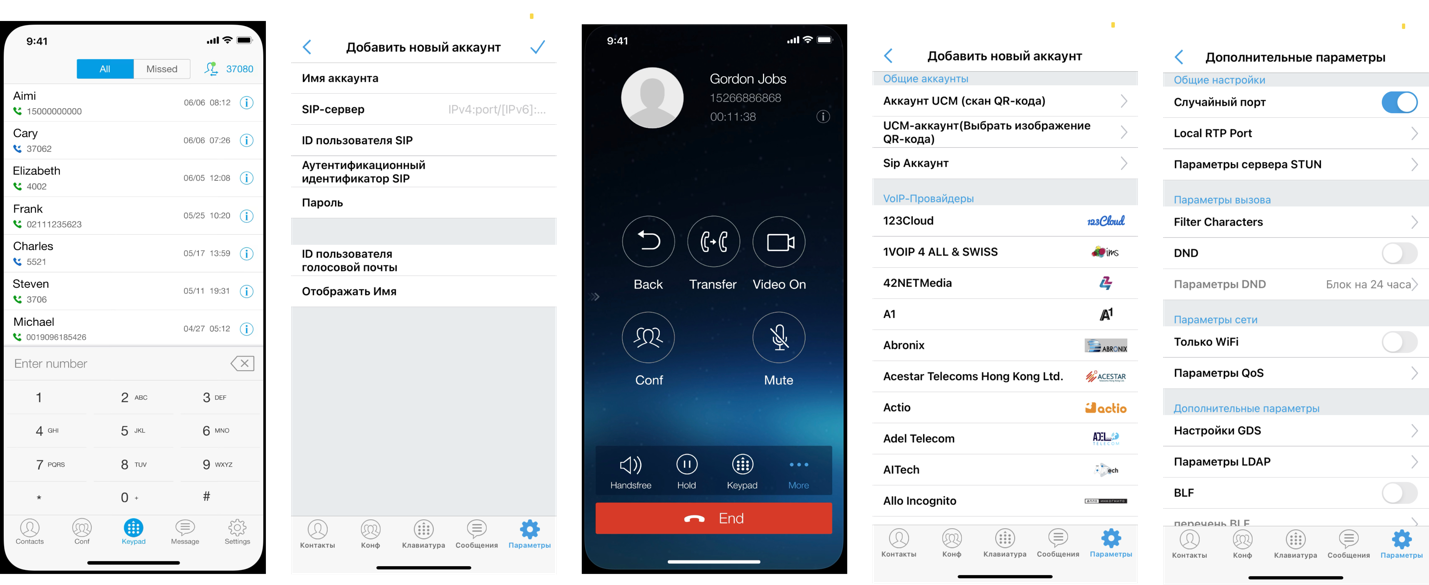


Рис. 2.2.1.1 - Интерфейс мобильного приложения GS Wave Lite

Однако есть и несколько недостатков использования Wave Lite. Одним из них является то, что это приложение стороннего производителя, и поэтому оно не так хорошо интегрируется с внутренними системами в организации, как пользовательское решение. Wave Lite не предусматривает решения одного из основных требований к системе – наличие каталога сотрудников организации, а из-за большого количества настроек процесс авторизации, например, может стать трудной задачей для пользователя.

Еще одним недостатком является то, что Wave Lite может не обеспечивать такой же уровень безопасности, как пользовательское решение.

Кроме того, НИЯУ МИФИ уже предоставляет собственное решение – VoIP MEPhI – для операционной системы Android. Наличие нескольких клиентов мобильного приложения от разных производителей может затруднить эффективное общение пользователей.

Учитывая преимущества и недостатки, для организации может быть лучше создать собственное мобильное приложение для SIP-телефонии. Это позволит адаптировать приложение к своим конкретным потребностям, обеспечить его хорошую интеграцию и предложить дополнительные функции и возможности, которых нет в готовых приложениях для SIP-софтфонов. В пункте 2.2.2 главы 2 определяются требования к разрабатываемому мобильному приложению

### **2.2.2 Определение требований к мобильному приложению**

Система должна отвечать следующим требованиям:

* Поддержка нескольких учетных записей. Приложение должно позволять пользователям настраивать и управлять несколькими SIP-аккаунтами, включая возможность переключения между ними по мере необходимости;
* Каталог абонентов и подразделений НИЯУ МИФИ. Приложение должно реализовывать функционал навигации и поиска по каталогу, а также кэширование ранее посещенных уровней вложенности каталога для доступа в ситуациях отсутствия сети;
* Каталог избранных пользователей. Приложение должно предоставлять пользователю возможность добавления абонентов в список «Избранное» для быстрого доступа к интересующим контактам;
* Недавние звонки. Система должна сохранять недавно совершенные звонки, а также отображать их в стандартном приложении «Телефон» на iOS устройствах;
* Высококачественные аудио звонки;
* Возможности настройки. Приложение должно позволять пользователям настраивать различные параметры приложения, например, доступность SIP-аккаунта для звонка;
* Простота использования. Приложение должно быть простым в использовании и иметь удобным интерфейсом, позволяющим пользователям быстро совершать и принимать звонки;
* Поддерживаемость. Исходный код программы должен соответствовать основным принципам разработки, таким как SOLID, KISS, DRY и т.д., а также быть задокументирован.

## **2.3 Проектирование UX и UI системы**

## **2.4 Архитектура мобильного приложения**

В качестве архитектуры приложения была выбрана архитектура MVC+Coordinator. На рисунке 2.4.1 представлена схема MVC (Model-View-Controller).

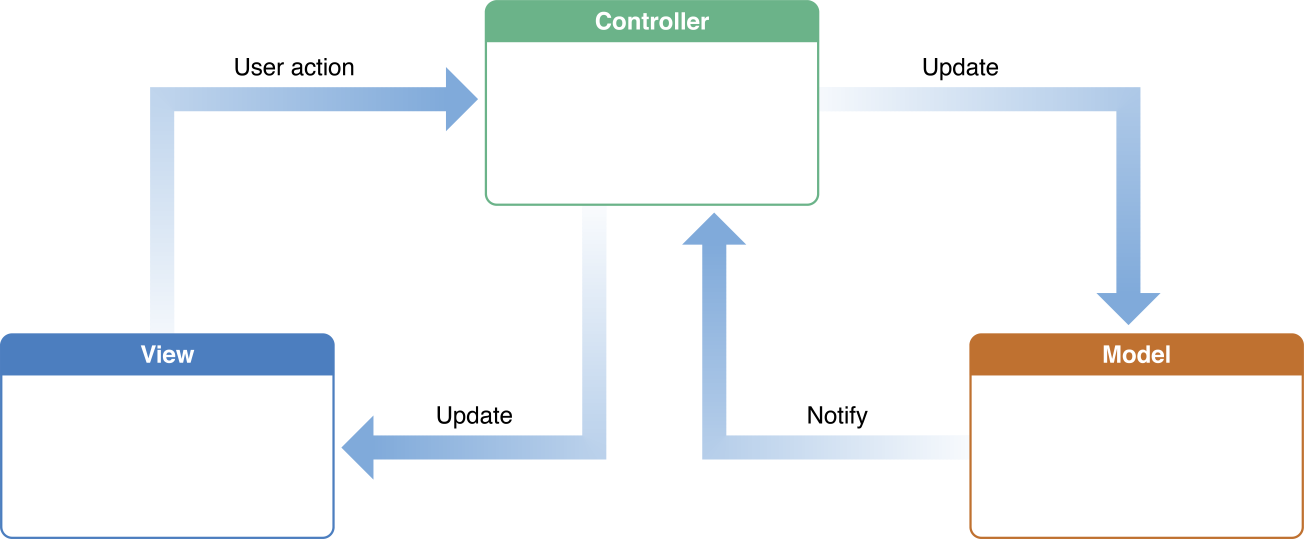


Рисунок 2.4.1 – Схематическое изображение архитектуры MVC

Достоинства выбранной архитектуры:

* Архитектура является понятной для программиста, поддержка кода не будет требовать много времени на понимание работы с архитектурной частью приложения;
* Равномерная нагрузка на Model, View и Controller;
* Переиспользуемость и расширяемость программного кода

В рамках мобильного приложения VoIP MEPhI, в котором не требуется реализация большого количества экранов, MVC является удобным вариантом с точки зрения скорости разработки и дальнейшей поддержки кода.

Для инкапсуляции логики навигации в мобильном приложении используется паттерн Coordinator. Вместо прямого вызова UIViewController’а из другого UIViewController’а используется отдельная изолированная и невидимая для разных Flow сущность. На рисунке 2.2.2 представлена упрощенная схема работы с паттерном Coordinator.

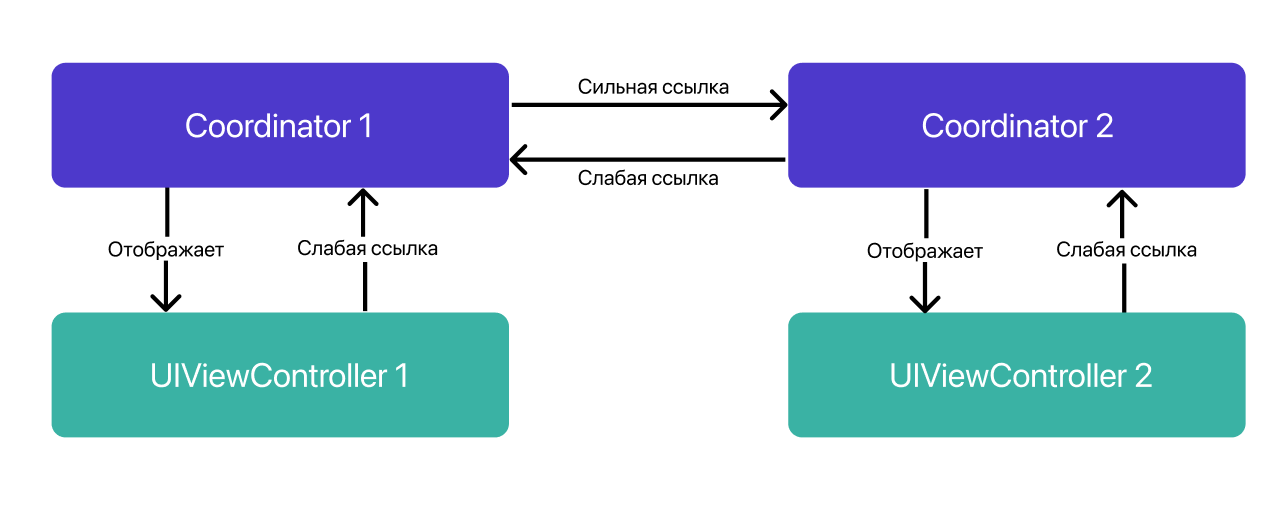


Рисунок 2.4.2 – Схематическое изображение паттерна Coordinator

Из схемы видно, что существует «родительский» Coordinator 1 и «дочерний» – Coordinator 2. В Swift управление памятью происходит посредством механизма ARC (Automatic Reference Counting), т.е. объект удаляется из памяти, если количество сильных ссылок на него равно нулю. Поэтому существуют и слабые ссылки, чтобы избежать ситуации «Reference cycle», когда объекты ссылаются друг на друга по кругу и не дают друг другу удалиться из памяти.

Часть исходного кода, отображающая работу паттерна Coordinator, представлена в Приложение А.

# **ПРИЛОЖЕНИЕ А. Листинг кода, демонстрирующий работу паттерна Coordinator**

**A1 – CoordinatorProtocol.swift**

**import** UIKit

**protocol** FlowCoordinatorProtocol: AnyObject {

**var** parentCoordinator: MainCoordinatorProtocol? { **get** **set** }

}

**protocol** CoordinatorProtocol: FlowCoordinatorProtocol {

**var** rootViewController: UIViewController { **get** **set** }

**func** start() -> UIViewController

**@discardableResult** **func** resetToRoot() -> **Self**

}

**extension** CoordinatorProtocol {

**var** navigationRootViewController: UINavigationController? {

**get** {

(rootViewController **as**? UINavigationController)

}

}

**func** resetToRoot() -> **Self** {

navigationRootViewController?.popToRootViewController(animated: **false**)

**return** **self**

}

}

**A2 – ApplicationCoordinator.swift**

**import** UIKit

**protocol** MainCoordinatorProtocol: CoordinatorProtocol {

**var** recentCoordinator: RecentCoordinatorProtocol { **get** }

**var** dialingCoordinator: DialingCoordinatorProtocol { **get** }

**var** catalogCoordinator: CatalogCoordinatorProtocol { **get** }

**var** profileCoordinator: ProfileCoordinatorProtocol { **get** }

}

**class** ApplicationCoordinator: MainCoordinatorProtocol {

**var** parentCoordinator: MainCoordinatorProtocol?

**lazy** **var** recentCoordinator: RecentCoordinatorProtocol = RecentCoordinator()

**lazy** **var** dialingCoordinator: DialingCoordinatorProtocol = DialingCoordinator()

**lazy** **var** catalogCoordinator: CatalogCoordinatorProtocol = CatalogCoordinator()

**lazy** **var** profileCoordinator: ProfileCoordinatorProtocol = ProfileCoordinator()

**lazy** **var** rootViewController: UIViewController = UITabBarController()

**func** start() -> UIViewController {

**let** recentVC = recentCoordinator.start()

recentCoordinator.parentCoordinator = **self**

recentVC.tabBarItem = UITabBarItem(title: "Недавние", image: .init(systemName: "clock"), tag: 0)

**let** dialingVC = dialingCoordinator.start()

dialingCoordinator.parentCoordinator = **self**

dialingVC.tabBarItem = UITabBarItem(title: "Клавиши", image: .init(systemName: "phone"), tag: 1)

**let** catalogVC = catalogCoordinator.start()

catalogCoordinator.parentCoordinator = **self**

catalogVC.tabBarItem = UITabBarItem(title: "Каталог", image: .init(systemName: "books.vertical"), tag: 2)

**let** profileVC = profileCoordinator.start()

profileCoordinator.parentCoordinator = **self**

profileVC.tabBarItem = UITabBarItem(title: "Профиль", image: .init(systemName: "person.crop.circle"), tag: 3)

**let** tabBarController = rootViewController **as**! UITabBarController

**let** tabBar = tabBarController.tabBar

tabBar.tintColor = .yinyang

tabBar.backgroundColor = .systemBackground

tabBarController.viewControllers = [recentVC, dialingVC, catalogVC, profileVC]

**return** rootViewController

}

}

**A3 – CatalogCoordinator.swift**

**import** UIKit

**protocol** CatalogCoordinatorProtocol: CoordinatorProtocol {

**func** goNextUnit(with filterCode: String)

}

**class** CatalogCoordinator: CatalogCoordinatorProtocol {

**weak** **var** parentCoordinator: MainCoordinatorProtocol?

**var** rootViewController: UIViewController = .init()

**func** start() -> UIViewController {

**let** catalogVC = CatalogViewController(coordinator: **self**)

**let** catalogNavigationController = UINavigationController(rootViewController: catalogVC)

rootViewController = catalogNavigationController

**return** rootViewController

}

**func** goNextUnit(with filterCode: String) {

**let** vc = CatalogViewController(coordinator: **self**, filterCode: filterCode)

(rootViewController **as**? UINavigationController)?.pushViewController(vc, animated: **true**)

}

}