МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**КУРСОВАЯ РАБОТА**

**АГРЕГАЦИЯ ПАКЕТОВ ДАННЫХ**

**ИЗ СЕРВИСОВ ОБМЕНА СООБЩЕНИЯМИ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.Е. Романов

(подпись)

Направление подготовки 01.03.02 «Прикладная математика и информатика»

Направленность (профиль) «Системное программирование и компьютерные технологии» (Математическое и программное обеспечение вычислительных машин)

Научный руководитель

канд. физ.-мат. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.В. Гаркуша

(подпись)

Нормоконтролер

ст. преп. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Харченко

(подпись)

Краснодар

2018

**РЕФЕРАТ**

Курсовая работа 36 с., 5 рис., 6 источников.

МЕССЕНДЖЕРЫ, СОЦИАЛЬНЫЙ СЕТИ, ОБЪЕДИНЕНИЕ СООБЩЕНИЙ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ЕДИНЫЙ СЕРВИС ОБМЕНА СООБЩЕНИЯМИ, ОМНИКАНАЛЬНОСТЬ, МУЛЬТИМЕССЕДЖИНГ, iOS, SWIFT Цель курсовой работы – изучение природы http-запросов к серверу при работе с API популярных сервисов обмена сообщениями и соцсетей, создание клиентской части мобильного приложения. Результат работы - программная реализация приложения, функциями которого будут являться  возможность единовременного получения сообщений и данных из нескольких сервисов, двусторонняя коммуникация в интерфейсе диалога, который по содержанию копирует диалог в соответствующем сервисе.

СОДЕРЖАНИЕ

[Введение 3](#_Toc516320551)

[1 Мультисервис мгновенного обмена сообщениями 4](#_Toc516320552)

[1.1 Идея 4](#_Toc516320553)

[1.2 Объективные недостатки текущих решений](#_Toc516320553) 4

[1.3 Выбор платформы для разработки](#_Toc516320553) 5

[2 Инструменты и теоретическая часть разработки](#_Toc516320556) 6

[2.1 IDE](#_Toc516320557) 6

[2.2 Выбор языка реализации](#_Toc516320561) 9

[2.3 HTTP - запросы](#_Toc516320561) 9

[2.4 Уровень библиотек](#_Toc516320561) 10

2.4.1 Обработка данных с сервера 10

[2.5 API популярных сервисов](#_Toc516320561) 15

[2.5.1 Функции API](#_Toc516320553) 15

[2.6 VK API](#_Toc516320553) 16

[2.7 Facebook API](#_Toc516320553) 18

[3 Программная реализация](#_Toc516320570) 19

[3.1 Общая структура приложения](#_Toc516320571) 19

[3.1.1 Структура ядра](#_Toc516320571) 20

[3.2 Процесс решения задачи с точки зрения разработчика 2](#_Toc516320572)1

[3.2 Реализация взаимодействия с VK API 28](#_Toc516320572)

[4 Итог работы 28](#_Toc516320572)

[4.1 Тестирование 28](#_Toc516320572)

[4.2 Итог разработки 28](#_Toc516320572)

[Заключение 30](#_Toc516320573)

[Список использованных источников 31](#_Toc516320574)

# **ВВЕДЕНИЕ**

Сегодня практически каждый человек имеет учётные записи в нескольких социальных сетях и мессенджерах. К сожалению, в большинстве случаев сервисы никак не интегрированы друг с другом, как это реализовано для e-mail. Для работы с ними необходимо переключатся между несколькими отдельными программами. Делая это снова, пользователь попадает в некий цикл и рискует потраченным временем, особенно при большом количестве сервисов в пользовании. Ситуация усугубляется необходимостью порой пользоваться несколькими аккаунтами одного такого сервиса для разных целей.

Неудобства возникают и в корпоративном сегменте. В сфере продаж мессенджеры давно применяются для консультаций и обработки заказов. А многие интернет-магазины (особенно мелкие) строят всю коммуникацию с клиентами на связке социальной сети с размещенным товаром и мессенджером в качестве непосредственного канала продаж. Но обработка множества обращений из стандартного интерфейса сервиса неэффективна. А специфика работы подразумевает работу с целым набором таких сервисов одновременно, что еще более осложняет процесс.

Цель курсовой работы - создание единого канала связи, в одном окне собирающего популярные сервисы обмена сообщениями с сохранением функциональности каждого из них.

Итог работы – реализация задумки в виде приложения-агрегатора для мобильной платформы.

1. **Мультисервис обмена сообщениями**

**1.1 Идея**

Программные продукты, интегрированные со всеми популярными системами обмена сообщений: Viber, WhatsApp, Telegram, Facebook, VK с возможностью общения с пользователями всех сервисов в одном окне уже существуют на рынке, но в основном это корпоративно-ориентированные решения, недоступные рядовому пользователю.

* 1. **Объективные недостатки текущих реализаций**

Ознакомившись с ситуацией на рынке похожих сервисов, выделяем основные решения: Pact, Livetex, Chat2Desk и тд.

Их особенность - ориентированность на коммерческую аудиторию, где важна коммуникация с клиентами с всех платформ, а, соответственно, использование по платной подписке (10–100$ ежемесячно для одного устройства).

Во многом это компенсируется эксклюзивными функциями: например, во многие продукты включен нейромодуль, в него загружается база знаний. В момент обращения клиента нейромодуль анализирует вопрос и показывает оператору подходящие материалы или поддерживаются чат-боты, которые распознают ключевые слова и присылают заготовленные ответы.

* 1. **Выбор платформы для разработки**

Так как у нас нет ориентированности на коммерческий сегмент, будет оптимальным решением воплотить проект в мобильное приложение, которое всегда под рукой. В настоящий момент имеются две крупнейшие мобильные платформы, рассмотрим обе.

С точки зрения отладки и исправления багов, разработка под iOS очень сильно выигрывает у Android, т.к. предоставляет более детерминированное окружение. Это, по сути, одна ОС от одного производителя, и это всего лишь 4 разрешения экранов для iPhone и 2 разрешения для iPad (retina и non-retina). А Android дает гигантскую энтропию и сильную фрагментацию по версиям ОС, которые надо поддерживать при разработки для большего охвата аудитории. У каждого большого вендора Android-устройств по сути своя, отличная от других версия Android. Каждый производитель предоставляет пользователю свой уникальный интерфейс, но для разработчика это выливается в необходимость иметь порядка 20-30 устройств от всех этих вендоров и постоянном отлове различных специфичных ошибок для каждого из этих устройств, а также еще куча разрешений с различной плотностью, что в конечном итоге является источником трудностей не только разработчика, но и дизайнера интерфейса.

**2 Инструменты и теоретическая часть разработки**

### **2.1 IDE**

Главный инструмент разработчика под macOS и вместе с ней iOS — среда программирования Xcode. Она включает средства для создания приложений для Mac, iPhone, iPad, Apple TV, Apple Watch. Все платформы Apple. Xcode содержит средство построения интерфейса (Interface Builder), редактор кода, поддерживающего все современные средства работы с кодом. Кроме того, не выходя из Xcode, можно протестировать приложение; если оно разрабатывается для внешнего устройства, то его можно запустить в эмуляторе.В систему включены эмуляторы всех устройств, поддерживаемых компанией. В дополнение Xcode содержит графические инструменты для анализа производительности приложения, включая средства диагностирования загруженности CPU, использования накопителя, нагрузки на графический адаптер (со стороны OpenGL).

Текущая версия вобрала в себя: новую версия языка версию языка Swift 4, средство изучения Swift Playground для iPad, новую версию Interface Builder, которая стала быстрее работать и позволяет просматривать макет приложения на разных платформах, без необходимости развертывать на симуляторе само приложение. В Xcode 10 можно разрабатывать приложения для macOS Sierra, iOS 12, watchOS 5 и tvOS 12. Имеются все SDK.

### **2.2 Выбор языка реализации**

Исторически разработка приложений для iOS и Mac велась на Objective-C. Недавно вышедший язык Swift выступает в качестве возможной замены Objective-C. Последний хоть и активно используется, но морально устарел и не имеет будущего. Изучив Swift, можно преуспеть в создании приложений сразу для обеих платформ.

Swift отлично подходит для быстрого развития. Когда команда Apple разрабатывала замену Objective-C, у них было два основных требования: он должен быть прост в изучении и способствовать ускорению цикла разработки приложений.

В итоге, Swift имеет все атрибуты современного языка программирования и определённо превосходит Objective-C по всем фронтам. Основные особенности с точки зрения безопасности:

1. нет неопределенных или неинициализированных переменных,
2. нет ошибок с размерностями массивов,
3. нет ошибок переполнения,
4. явная обработка значений nil (null),
5. автоматическое управление памятью.

Таким образом, появляется возможность тратить больше времени на реализацию идей и меньше - на беспокойство по поводу возможных ошибок, сбоев и конфликтов вашего кода. Кроме того, язык поборол синтаксическую многословность в Objective-C, что упростило запись и чтение. Результат – в разы меньше времени на написание аналогичного кода в Swift. Несмотря на то, что язык высокоуровневый, ориентированный на скорое изучение, он по-настоящему быстр. По словам Apple, Swift до 2.6 раз быстрее, чем Objective-C и почти в 8,4 раза быстрее, чем Python 2.7. Конечная цель – сделать язык быстрее, чем C++. Важно, что Swift не просто быстр, но и наполнен современными языковыми функциями, которые позволяют писать по-настоящему функциональный код. Среди них:

1. дженерики,
2. замыкания,
3. кортежи,
4. итераторы,
5. множественные возвраты,
6. встроенные шаблоны функционального программирования.

Язык безопасен: введение многих из перечисленных возможностей, а также усовершенствование синтаксиса делает Swift безопаснее, нежели Objective-C. Например, улучшение работы с памятью означает меньшее количество возможностей для несанкционированного доступа к данным. Переход к неправильным частям памяти, ошибочное изменение данных также усложнено. Другой пример: более эффективная обработка ошибок значительно уменьшает количество сбоев и появление критических сценариев. Непредсказуемое поведение минимизировано [6]. Swift бесплатный и открытый: через год после появления Swift Apple сделал его  языком с открытым исходным кодом. Хотя это не уникальное явление для современного мира, для компании подобное – редкость. Но шаг со Swift стал оправданным и плодотворным. Как и в случае с любым другим языком с открытым исходным кодом, Swift – полностью в руках сообщества. Пользователи могут предлагать пути исправления ошибок и улучшать функции, помогают переносить приложения за пределы Mac и iOS. В конце концов, пользователи – главная движущая сила языка. Согласно отчету [GitHub Octoverse 2017](https://octoverse.github.com/), Swift является 13-м по популярности среди языков в проектах с открытым исходным кодом. Ресурс [TNW](https://thenextweb.com/dd/2016/03/01/data-shows-demand-for-developers-who-know-swift-has-skyrocketed/) в 2016 году сообщил, что спрос на сотрудников Swift вырос на 600 процентов. К концу года [Upwork](https://www.upwork.com/hiring/for-clients/9-reasons-to-choose-swift-for-ios-app-development/) сообщила, что Swift стал вторым самым быстрорастущим навыком на внештатном рынке труда. И в опросе [Stack Overflow 2017](https://insights.stackoverflow.com/survey/2017) Swift стал четвертым самым любимым языком среди активных разработчиков. У Apple нет причин в ближайшее десятилетие заменять Swift другим языком. Наблюдая сюда 4-летний прогресс, как с точки зрения развития, так и популярности, неизменно растущие продажи устройств и расширение линейки. Основываясь на этих фактах, можно с уверенностью говорить, что потребность в Swift-разработчиках будет расти.

* 1. **HTTP - запросы**

Важнейшей частью работы с API web-сервисов является использование HTTP-запросов. Рассмотрим их специфику и применение.

Итак, HTTP клиент посылает запрос на сервер в форме сообщения-запроса, которое имеет следующий формат:

1. cтрока запроса (обязательный элемент),
2. заголовок (опциональный элемент),
3. пустая строка (обязательный элемент),
4. тело сообщения (опциональный элемент).

Строка запроса начинается с токена метода, после которого следует URL запроса и версия протокола. Элементы отделяются друг от друга пробелами:

Строка-запроса = Метод (пробел) URL запроса (пробел) версия-HTTP (следующая строка)

Метод запроса указывает метод, который должен быть вызван на стороне сервера по указанному индентификатору URL.

В HTTP существует восемь методов:

1. HEAD - используется для получения строки статуса и заголовка от сервера по URL. Не изменяет данные.
2. GET - используется для получения данных от сервера по указанному URL. Не изменяет данные.
3. POST - используется для отправки данных на сервер (например информации о разработчике и т.д.) с помощью форм HTML.
4. PUT - замещает все предыдущие данные на ресурсе новыми загруженными данными.
5. DELETE - удаляет все текущие данные на ресурсе, определённом URL.
6. CONNECT - устанавливает туннельное соединение с сервером по указанному URL.
7. OPTIONS - описывает свойства соединения для указанного ресурса.
8. TRACE - выполняет вызов возвращаемого тестового сообщения с ресурса.
   1. **Уровень библиотек**

Итак, у нас возникла необходимость обрабатывать данные с сервера. Данная задача присутствует почти во всех мобильных приложениях.

Существует нативный инструмент для этого — URLSession, но работать с ним немного сложнее, чем хотелось бы. Для облегчения этого процесса существует фрэймфорк Alamofire — это обвертка над URLSession, которая сильно упрощает жизнь при работе с сервером.

Воспользуемся CocoaPods т.к. с ним очень легко и быстро работать.

CocoaPods — это мощное и одновременно изящное средство управления зависимостями Cocoa-библиотек, которые разработчики используют в своих iOS и MacOS X проект

Нам необходимо получить доступ к новым функция, т.к. это не наши файлы, а отдельная библиотека: для этого прописывается импорт библиотеки Alamofire.

Собственно, сам метод запроса: Request

Далее первым параметром передается URL, по которому будет производится запрос:

"http://jsonplaceholder.typicode.com/posts"

Метод responseJSON говорит о том, что ответ от сервера нам нужен в JSON формате. Далее в замыкании(closure) мы получаем ответ от сервера и выводим его в консоль:

{

response in

print(response)

}

Важно заметить, что код в этом замыкании происходит асинхронно и выполнится после выхода из текущего исполняемого блока кода.

На самом деле мы сделали GET запрос, но нигде этого не указывали. Начиная с Alamofire 4 по умолчанию выполняется GET запрос. Можно его явно указать, заменив соответствующий код на следующий:

request("http://jsonplaceholder.typicode.com/posts", method: .get)

Таким образом в параметре method передается метод запроса и от него зависит, как мы будем общаться с сервером. Основные HTTP-методы упомянуты выше.

Функция request — глобальная функция, поэтому мы можем ее вызывать через Alamofire.request или просто request[3].

Так выглядит полный запрос со всеми параметрами:

request(URLConvertible, method: HTTPMethod, parameters: Parameters?, encoding: ParameterEncoding, headers: HTTPHeaders?),

где HTTPMethod - перечисление, со всеми возможными типами запросов:

public enum HTTPMethod: String {

case options = "OPTIONS"

case get = "GET"

case head = "HEAD"

case post = "POST"

case put = "PUT"

case patch = "PATCH"

case delete = "DELETE"

case trace = "TRACE"

case connect = "CONNECT"

}

ParameterEncoding - тоже протокол с одной функцией:

public protocol ParameterEncoding {

func encode(\_ urlRequest: URLRequestConvertible, with parameters: Parameters?) throws -> URLRequest

}

Он необходим для определения того, в каком виде нам закодировать наши параметры. Разные серверы и запросы требуют определенной кодировки.

Этот протокол реализуют:

1. URLEncoding;
2. JSONEncoding;
3. PropertyListEncoding.

В основном этот параметр не используется, но иногда бывает нужен, в частности JSONEncoding.default для кодировки в JSON формате и PropertyListEncoding.default в XML. HTTPHeaders - это также Dictionary, но другой типизации:

public typealias HTTPHeaders = [String: String]

Headers(заголовки) нам будут необходимы в основном для авторизации.  
На выходе мы получаем объект типа DataRequest — сам запрос. Его мы можем сохранить, передать, как параметр в другую функцию при необходимости, донастроить и отправить. Об этом далее.

**2.4.1 Обработка данных с сервера**

Итак, у нас возникла необходимость обрабатывать данные с сервера. Данная задача присутствует почти во всех мобильных приложениях. Делаем это при помощи библиотеки JSON:

JavaScript Object Notation, или сокращенно JSON является самым распространенным способом связи с сервером и получения информации с него. Он чрезвычайно популярен из-за простоты в использовании и восприятии. Это совсем небольшая библиотека, и с ней очень удобно работать[4].

JSON – один из частных случаев стиля REST.

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат. Отсутствие дополнительных внутренних прослоек означает передачу данных в том же виде, что и сами данные. Т.е. мы не заворачиваем данные в XML, как это делает SOAP и XML-RPC, не используем AMF, как это делает Flash и т.д. Просто отдаем сами данные.  
 Каждая единица информации однозначно определяется URL – это значит, что URL по сути является первичным ключом для единицы данных. Т.е. например третья книга с книжной полки будет иметь вид /book/3, а 35 страница в этой книге — /book/3/page/35. Отсюда и получается строго заданный формат. Причем совершенно не имеет значения, в каком формате находятся данные по адресу /book/3/page/35 – это может быть и HTML, и отсканированная копия в виде jpeg-файла, и документ Microsoft Word.

Как происходит управление информацией сервиса целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол конечно же HTTP. Так вот, для HTTP действие над данными задается с помощью методов: GET, POST и т.д.

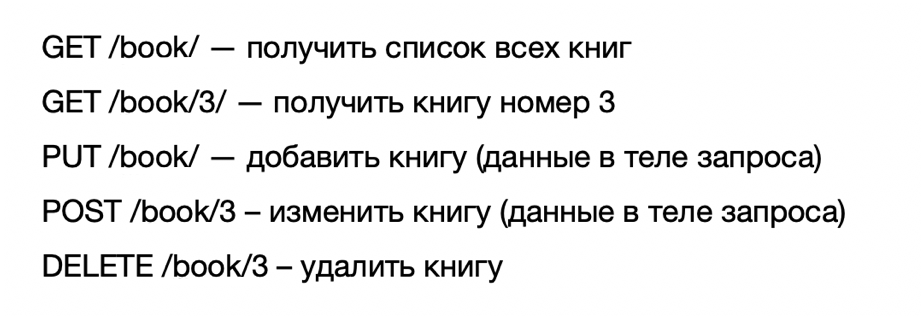
Вот как это будет выглядеть на примере:

Рисунок 1 – Пример работы с HTTP-метода

* 1. **API популярных сервисов**

API (application programming interface) - это набор готовых классов, функций, процедур, структур и констант. Вся эта информация предоставляется самим приложением (или операционной системой). При этом пользователю не обязательно понимать, что это API технология обеспечивает взаимодействие модулей. Цель предоставленной информации – использование этих данных при взаимодействии с внешними программами.

API различных продуктов используются программистами для создания приложений, которые будут взаимодействовать друг с другом.

В общем случае данный механизм применяется с целью объединения работы различных приложений в единую систему. Это достаточно удобно для исполнителей. Ведь в таком случае к другому приложению можно обращаться как к «черному ящику». При этом не имеет значения его внутренний механизм – программист может вообще не знать, что такое API.

* + 1. **Функции API**

В процессе работы элементы механизма API организуют многоуровневую иерархию. При этом подчиненные компоненты также получают подобную структуру. Внутри стандартной сетевой модели OSI выделяют как минимум 7 внутренних уровней. Они классифицируются от физического уровня трансляции бит до приложений, таких как протоколы HTTP и IMAP. Таким образом API верхнего использует функциональность нижнего.

Одним из важных компонентов организации информации при описании API являются библиотеки функций и классов. В их состав входят описания сигнатур и семантики. Здесь API функции – это просто часть механизма интерфейса.

Для вебмастеров и программистов особенно важны Web API. Такие системы управления включают в себя комплект HTTP-запросов. В результате получения таких запросов модуль генерирует строго определенную структуру HTTP-ответов. Для транспортировки информации между ними принято использовать форматы XML или JSON.

Фактически в этом случае название Web API будет синонимом обозначения веб-службы. Иными словами, это определенные программные системы со своими интерфейсами. Для получения конкретного доступа к ним используется идентификация в сети по веб-адресу. Например, при передаче данный на сервер применяется серверный API.

В случае построения программных систем на основе сервис-ориентированной архитектуры именно веб-служба является уровнем формирования модулей.

Для обычных пользователей такие службы являются синонимами абсолютно обычных решений в Интернете. Это может быть почта, поисковая система, сервис хранения файлов, социальных закладок и так далее. В случае необходимости тестирования веб-службы на больших объемах разнообразных данных соответствующий API testing предоставляет механизм для такой объемной работы.

При правильной организации любой клиент может использовать такие службы вне зависимости от типа компьютера, вида браузера и места своего нахождения в сети.

Здесь особенно важно, что открытый API – это возможность дополнения и расширения системы взаимодействия.

* 1. **VK API**

Примером использования в рекламе является API сайта vk.com.

API ВКонтакте — это интерфейс, который позволяет получать информацию из базы данных vk.com с помощью http-запросов к специальному серверу. Вам не нужно знать в подробностях, как устроена база, из каких таблиц и полей каких типов она состоит — достаточно того, что API-запрос об этом «знает». Синтаксис запросов и тип возвращаемых ими данных строго определены на стороне самого сервиса [1].

Рассмотрим отдельно все составляющие запроса:

1. https:// — протокол соединения;
2. api.vk.com/method — адрес API-сервиса;
3. users.get — название метода API ВКонтакте. Методы представляют собой условные команды, которые соответствуют той или иной операции с базой данных — получение информации, запись или удаление. Например, users.get — метод для получения информации о пользователе, video.add — метод для добавления видеозаписи в свой список, likes.delete — метод для удаления отметки «Мне нравится». Все методы разделены на секции. Например, для работы с сообществами Вам нужны методы секции groups, для работы с фотографиями — photos, и так далее;
4. Потом — параметры запроса. После названия метода нужно передать его входные данные (если они есть) — как обычные GET-параметры в http-запросе. В нашем примере мы сообщаем серверу, что хотим получить данные о пользователе с id=*210700286* и формат этих данных должен соответствовать актуальной версии API. Входные параметры всегда перечислены на странице с описанием метода.   
   В ответ сервер вернет JSON-объект с запрошенными данными (или сообщение об ошибке, если что-то пошло не так). JSON — это формат записи данных в виде пар «имя свойства»: «значение».

Структура ответа на запрос каждого метода также строго задана, и при работе с API Вы заранее знаете, что в поле id придет число, а в поле first\_name — строка. Такие правила оговариваются на страницах с описанием метода и соответствующих объектов, которые он возвращает в ответе.

* 1. **Facebook API**

Facebook API - это платформа для создания приложений, доступных для пользователей социальной сети Facebook. API позволяет приложениям использовать социальные связи и информацию о профиле, чтобы сделать приложения более привлекательными, а также публиковать действия в новостной ленте и на страницах профиля Facebook с учетом настроек конфиденциальности отдельных пользователей. С помощью API пользователи могут добавлять социальный контекст в свои приложения, используя данные профиля, друга, страницы, группы, фотографии и события. API использует протокол RESTful, а ответы представлены в формате JSON.

API Graph — это основной способ чтения и записи данных в социальный граф Facebook, используемый приложениями. Все наши SDK и продукты взаимодействуют с API Graph, а остальные наши API являются расширениями API Graph, поэтому понимать принципы работы API Graph очень важно [2].

**3 Программная реализация**

**3.1 Общая структура приложения**

На самом примитивном уровне абстракции приложение, ориентированное на работу с сервером состоит из следующих архитектурных слоев:

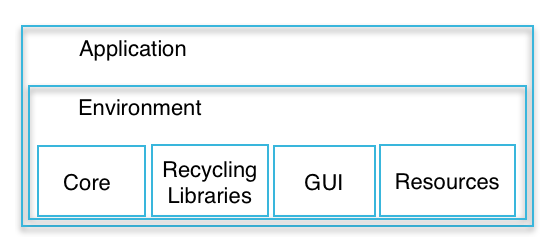
1. Ядро приложения, которое включает в себя компоненты системы, не доступные для взаимодействия с пользователем;
2. Графический пользователь интерфейс;
3. Компоненты повторного использования: библиотеки, визуальные компоненты и другое;
4. Файлы окружения: AppDelegate, .plist и т. д;
5. Ресурсы приложения: графические файлы, звуки, необходимые бинарные файлы. Наиважнейшим условием построение стрессоустойчивой архитектуры является отделение ядра системы от GUI, настолько, что б одно, могло успешно функционировать без другого. Интерфейс же приобретает однозначное толкование как с точки зрения пользователя, так и с точки зрения иерархии классов [5].

Рисунок 2 – Структура приложения

**3.1.1 Структура ядра**  
 Ядро приложения, состоит из следующих слоев:

1. (Start layer) Стартовый слой, определяющий рабочий процесс, начала исполнения программы;
2. (Network layer) Сетевой слой, обеспечивающий механизм транспортного взаимодействия;
3. (API layer) Слой API, обеспечивающий единую систему команд взаимодействия между клиентом и сервером;
4. (Network Cache Layer) Слой сетевого кэширования, обеспечивающий ускорения сетевого взаимодействия клиента и сервера;
5. (Validation Items Layer) Слой валидации данных полученных из сети;
6. (Network Items Layer) Слой сущности данных передаваемых по сети;
7. (Data Model) Модель данных, обеспечивающая взаимодействие сущностей данных;
8. (Local cache layer) Слой локального кеширования, обеспечивающий локальный доступ к уже полученным сетевым ресурсам;
9. (Workflow layer) Слой рабочих процессов, включающий классы и алгоритмы специфичные для данного приложения;
10. (Local storage) Локальное хранилище.

Одна из основных задач стоящие перед разработчиками системы заключается в том чтобы обеспечить взаимно независимое функционирование указанных слоев. Каждый слой должным обеспечивать только выполнение возложенных на него функций. Как правило, слой находящийся на более высоком уровне иерархии не должен иметь представление о специфике реализации других слоев.

**3.2 Процесс решения поставленной задачи с точки зрения разработчика**

Постановка задачи позволяет выделить несколько подзадач, которые могут быть описаны отдельными классами или методами:

а) Загрузка и обработка пакетных данных из сети:

1. проверка и парсинг полученных данных;
2. сохранение данных в виде массивов сообщений с привязкой к ID;
3. отправка сообщения на сервер (метод .post);
4. фильтрация данных по указанным критериям (настройки приложения).

б) Обеспечить связанную работу интерфейса, который состоит из следующих основных форм:

1. главный контроллер (может быть невидимым);
2. форма с последними сообщениями пользователя;
3. форма диалогов;
4. контроллер навигации для перемещения между формами;
5. опционально форма настроек.

в) После запуска приложения на выполнение, производится создание (инстанциирование) объекта отвечающего за загрузку данных (в нашем случае асинхронную) и начинает процесс. Главный контроллер приложения отображает сплеш-скрин, и в это время, формирует форму, которая займет его место по сокрытию сплэш-формы.

г) По окончании загрузки данных, на главный контроллер выводится список последних сообщений пользователя с идентификацией отправителя.

д) По выбору любого сообщения происходит переход в диалог с отправителем. Для отображения диалога, происходит выполнение соответствующего класса с передачей данных в него.

е) Здесь появляется возможность отправки ответного сообщения при помощи соответствующего метода.

**3.3 Реализация взаимодействия с VK API**

Первым делом в метод загрузки главного вью-контроллера viewDidLoad() помещаем post-запрос с методом messages.getLongPollServer, который возвращает объект, содержащий поля key, server, ts. Используя эти данные, мы можем подключиться к серверу быстрых сообщений для мгновенного получения приходящих сообщений и других событий. Далее, по кнопке обновления получаем массив последних сообщений пользователя, используем для этого метод messages.getLongPollHistory.

Он возвращает объект, который содержит поля history, messages, а также groups — массив объектов сообществ и profiles — массив объектов пользователей.

Поле history представляет из себя массив, аналогичный полю updates, получаемому от Long Poll сервера. Поле messages представляет из себя массив личных сообщений – объектов message, которые встречались среди событий с кодом 4 (добавление нового сообщения) из поля history. Каждый объект message содержит набор полей. Первый элемент массива представляет собой общее количество сообщений. Далее предполагается переход пользователя к нужному диалогу по нажати. На сообщение из списка. Как описано выше, здесь происходит переключение на следующий вью-котроллер, описанный классом DialogViewController.

В него передаются необходимые данные при помощи метода prepare(for segue: UIStoryboardSegue, sender: Any?), такие как, например, ID пользователя, с которым происходит диалог.

В описанном классе при его загрузки происходит получение истории сообщений конкретного диалога при помощи метода API messages.GetHistory.

Полученными таким образом сообщениями заполняем массив, а далее - выводим в интерфейс при помощи TableView – формы.

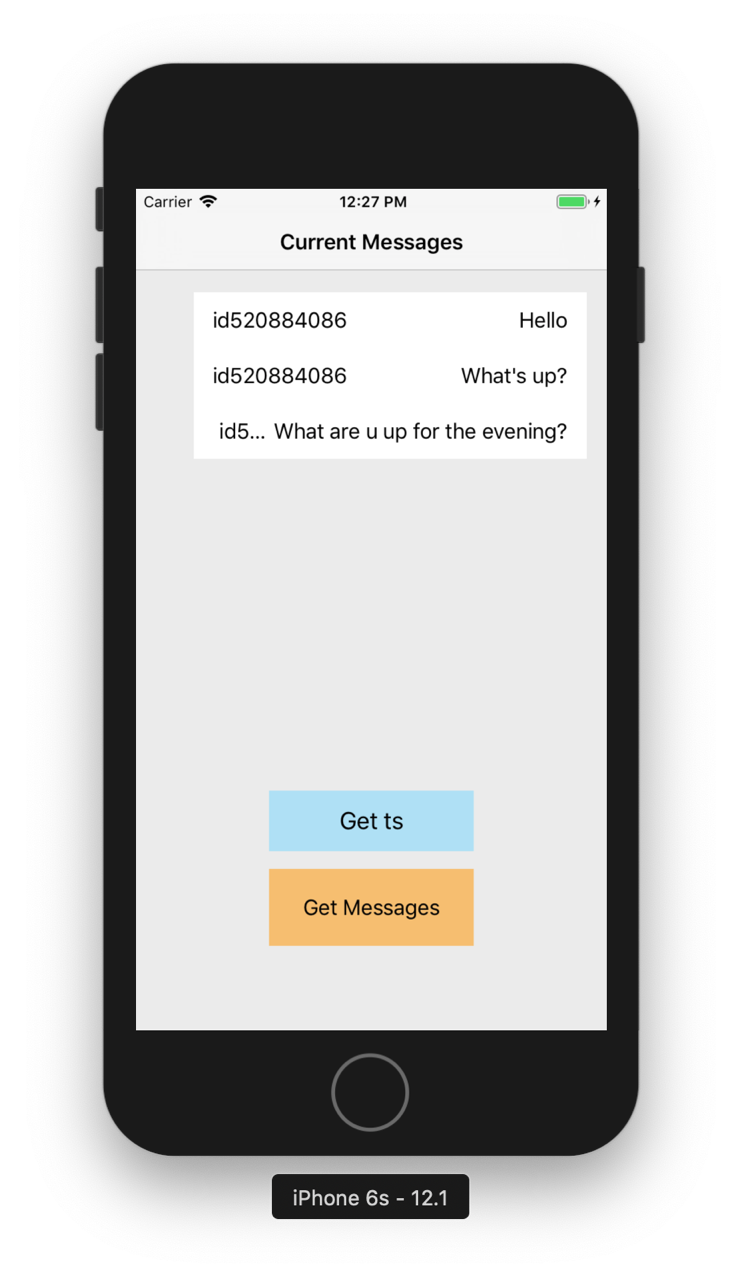
Для реализации функционала отправки сообщения добавляем в интерфейс класса диалога текстовое поле для ввода и кнопку отправки. Введенный текст оправляем при помощи метода messages.send. Сразу после отправки обновляем окно приложения и, если передача успешна, вновь отправленное сообщение отображается в диалоге.

Рисунок 3 – Рост объёма трафика

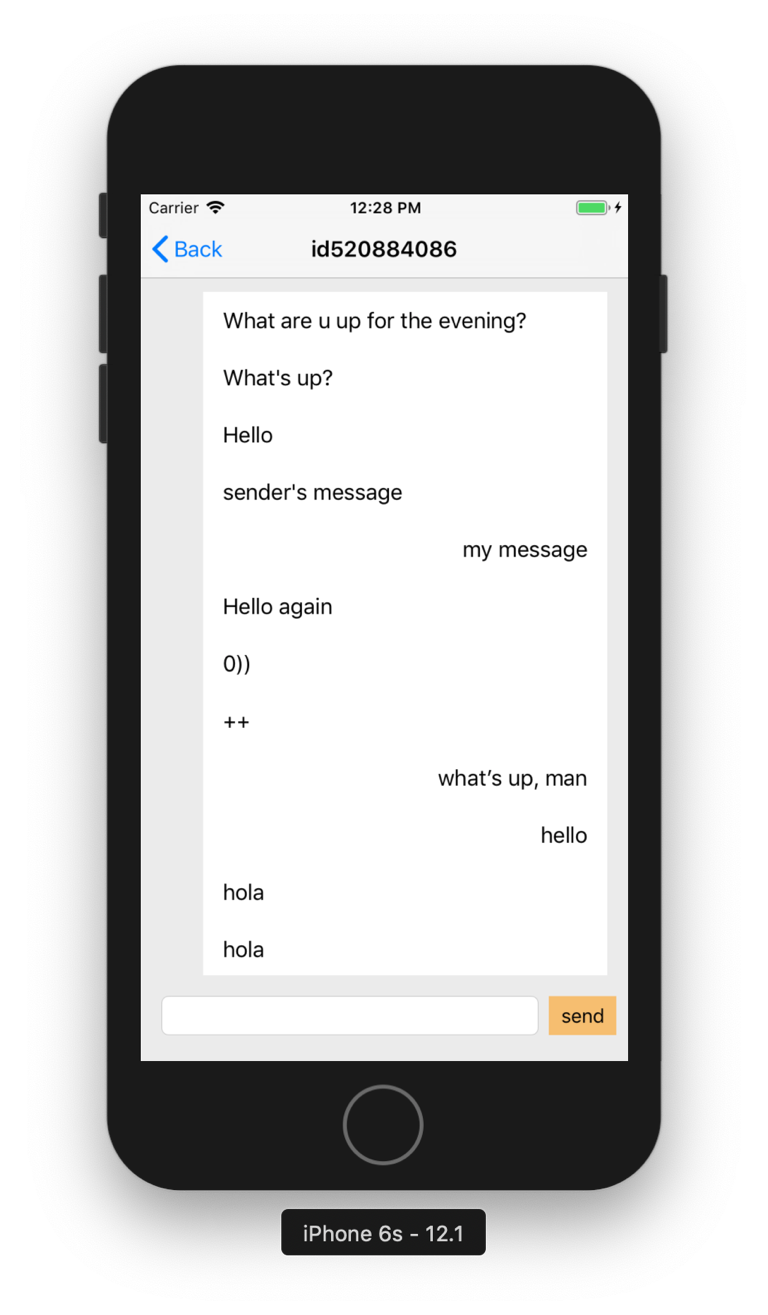


Рисунок 4 – Интерфейс диалога

**4 Итог работы**

### **4.1 Тестирование**

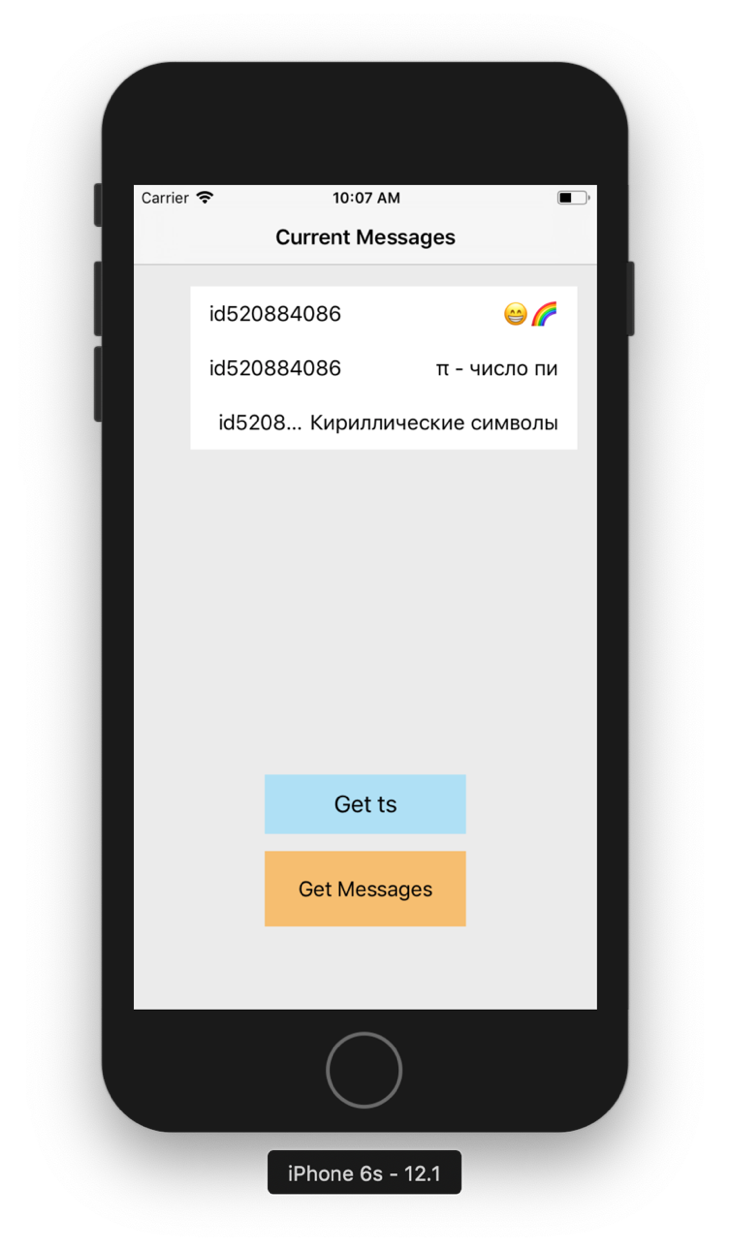
После написания программной реализации, необходимо протестировать приложение. Сделать это можно созданием посылкой на отправку спецсимволов, кириллицы, эмодзи и т.д. Такие символы могут пересылаться некорректно, так как они являются частью URL-адреса, но отправка проходит успешно. Общая работоспособность устройства была протестирована на эмуляторах различных устройств, а также на реальном устройстве (iPhone 6s)

Рисунок 5 – Отправка спецсимволов Unicode

**4.2 Результат разработки**

В результате работы было реализовано два программных класса.

Первый служит для обработки и вывода общего списка событий из всех поддерживаемых сервисов в реальном времени. Второй позволяет работать с диалогом соответствующего сервиса для приема и отправки сообщений в чат с пользователем в этом сервисе.

Итак, разработанное приложение реализует функционал мультимессенджера для двух крупнейших социальных сетей – Вконтакте и Facebook. В связи с ограничениями новейшей версии Graph API в Facebook возможна работа только с сообщениями сообщества. Тем не менее, реализованное приложение выигрывает по затратам памяти ресурсов у клиентов обоих сервисов.

# **ЗАКЛЮЧЕНИЕ**

В рамках курсовой работы были получены следующие результаты:

Был проведен обзор инструментов и навыков, необходимых для iOS - разработки, а также основ проектирования клиент-серверных приложений для мобильных платформ.

Было спроектировано и разработано приложение для операционной системы iOS для мультимесседжинга на мобильном устройстве, соответствующее техническому заданию.

Был проведен обзор применяемых в разрабатываемом приложении технологий, продемонстрирована функциональность приложения в различных условиях использования.

Приложение было протестировано как на реальном устройстве, так и на различных по своим техническим характеристикам и параметрам эмуляторах из среды Xcode.

В перспективе программный продукт может быть расширен следующей функциональностью:

1. объединение одного и того же контакта из разных соцсетей в единый;
2. объединение истории сообщений с одним контактом из разных сервисов в один диалог;
3. выбор между учетными записями в нескольких сервисах, от имени которых ведется активность;
4. минимальная база данных для хранения переписки и вложенных файлов;
5. передача файлов и мультимейдийного контента.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Документация API “Вконтакте” URL – https://vk.com/dev/first\_guide (22.05.2018).
2. Документация API Facebook URL – https://developers.facebook.com/docs (6.11.2018)
3. Alamofire Usage Guide URL – https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md (27.04.2018)
4. Введение в JSON со Swift URL – https://medium.com/@drugoj\_rom/введение-в-json-c-swift-8db1a92a757e/ (10.05.2018)
5. Apple Education. App Development with Swift , 2017. – 1015 с.
6. Apple Inc. The Swift Programming Language, 2014. – 500 с.