

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра вычислительной математики**

Курсовой проект

**Разработка iOS-приложений на языке Swift
(«Новости»)**

Малиновский Владислав Сергеевич

студент 4 курса 5 группы,
специальности «прикладная математика»

Руководители:

Мандрик Павел Алексеевич,
доцент кафедры вычислительной математики,
кандидат физико-математических наук

Пацино Ирина Владимировна,
специалист
ООО «Системные технологии»

Минск, 2018

ОГЛАВЛЕНИЕ

Введение	3
Глава 1. Процесс разработки приложения	4
Внедрение VK SDK в проект приложения	4
Инициализация SDK. Авторизация пользователя.....	5
Networking logic. Вызов методов API.....	6
Парсинг json.....	8
Архитектура приложения.....	8
Создание ячеек с помощью XIB файлов.....	10
Заполнение ячеек реальной информацией.....	10
Дизайн приложения.....	11
Дополнительная функциональность.....	11
Глава 2. Результаты работы.....	12
Заключение.....	18
Список используемой литературы.....	19

ВВЕДЕНИЕ

На сегодняшний день мобильные устройства и планшеты от компании Apple занимают значительную долю продаж на мировом рынке умных устройств. Все эти устройства от компании Apple оснащаются операционной системой iOS. Данная операционная система является одной из самых популярных операционных систем и с долей в 40% уступает лишь операционной системе Android. Поэтому разработка под операционную систему iOS – это одно из самых популярных направлений в сфере информационных технологий.

Первоначально, разработка приложений под iOS велась с помощью языка программирования Objective-C. Это объектно-ориентированный язык программирования, который был построен на основе языка C и парадигм Smalltalk. Однако с течением времени был создан язык программирования Swift. Данный язык задумывался как более легкий для чтения и устойчивый к ошибкам язык, нежели Objective-C. На сегодняшний день Swift представлен версией 5.0. И в процессе разработки нашего приложения мы будем использовать последнюю версию языка Swift.

Тема моего приложения – новости. Новостные приложения являются одними из самых востребованных в AppStore. Однако новостные приложения могут быть разной направленности: это может быть приложение для новостного портала или, например, для новостей из социальных сетей. Принимая во внимание популярность социальных сетей в нынешнее время, было решено разработать приложение для новостей из социальной сети ВКонтакте. Данная социальная сеть – самая популярная на территории стран СНГ. Разработка приложения проходила с использованием VK SDK. Данная SDK упрощает использование API ВКонтакте в iOS-приложениях.

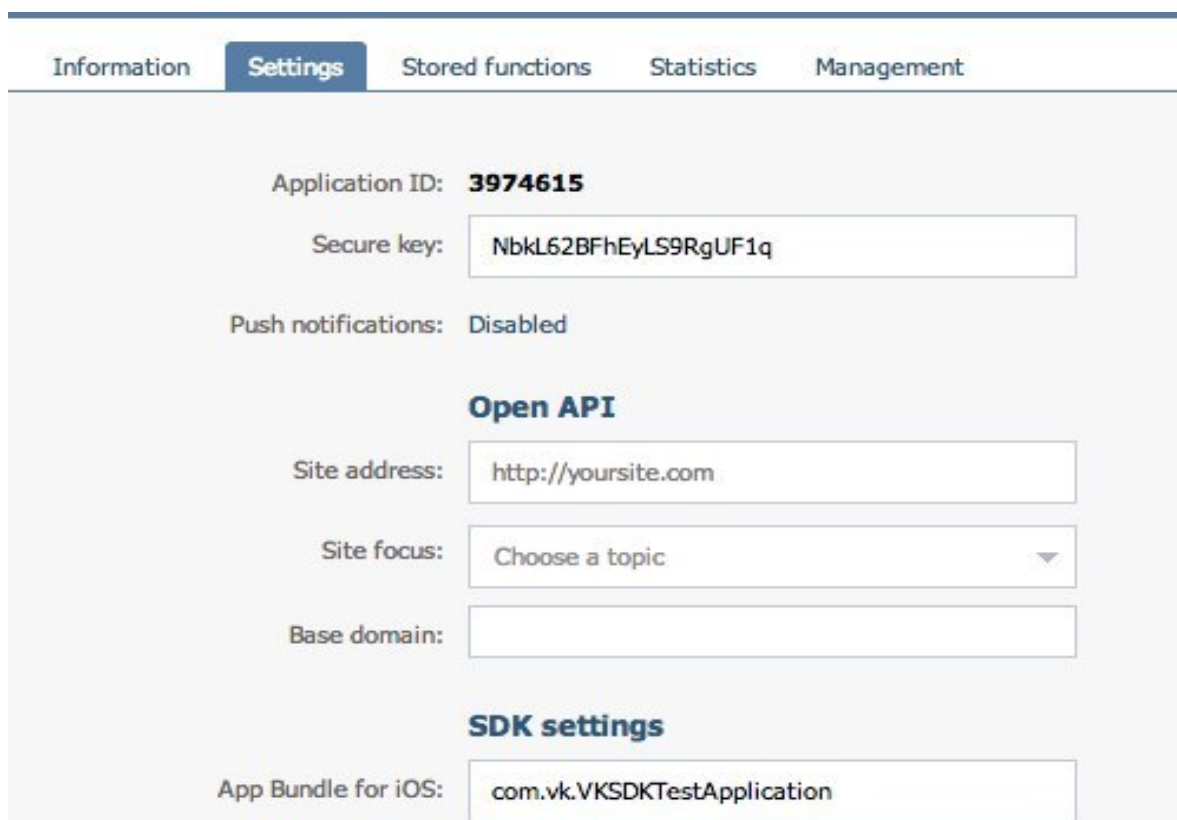
В процессе разработки были затронуты следующие темы: авторизация, работа с запросами, с потоками, следование шаблону проектирования Clean Swift, работа с форматом json, загрузка и кэширование изображений, автоматическая размерность ячеек, UITableView, NSLayoutAnchor, AutoLayout, UICollectionView, работа со сторонними библиотеками.

ГЛАВА 1. ПРОЦЕСС РАЗРАБОТКИ ПРИЛОЖЕНИЯ

1.1 Внедрение VK SDK в проект приложения

При установке VK SDK мы следовали официальной документации.

В первую очередь, требовалось провести подготовку к использованию SDK. Для этого создаем standalone-приложение. Мы его создаем для того, чтобы у нас была возможность использовать API ВКонтакте. В процессе создания standalone-приложения мы задаем название будущего приложения, а также указываем App Bundle ID нашего XCode проекта. На этом конфигурация standalone-приложения окончена.



The screenshot displays the 'Settings' tab of a configuration interface. At the top, there are five tabs: 'Information', 'Settings' (selected), 'Stored functions', 'Statistics', and 'Management'. Below the tabs, the 'Application ID' is listed as '3974615'. The 'Secure key' is shown in a text box with the value 'NbkL62BFhEyLS9RgUF1q'. The 'Push notifications' status is 'Disabled'. A section titled 'Open API' contains three fields: 'Site address' with the value 'http://yoursite.com', 'Site focus' with a dropdown menu showing 'Choose a topic', and 'Base domain' which is empty. Below this is a section titled 'SDK settings' with a field for 'App Bundle for iOS' containing the value 'com.vk.VKSDKTestApplication'.

Далее необходимо было настроить URL-схему нашего приложения для настройки авторизации. URL-схема является удобным способом передачи данных в iOS-приложения. С помощью URL-схем можно, например запустить браузер Safari из приложения. VK SDK требует, чтобы URL-схема имела вид VK+APP_ID (APP_ID генерируется при создании standalone-приложения). Настройка URL-схем проходит в info.plist файле нашего XCode проекта.

После подготовительных шагов идет непосредственное подключение фреймворка в наш проект. Установка VK SDK проводилась с помощью Carthage – менеджера управления зависимостями. Carthage нужен, чтобы встраивать в наше приложения библиотеки сторонних разработчиков. Одним

из преимуществ Carthage (в отличие от CocoaPods) является то, что он не вносит никаких изменений в наш проект и работает обособленно от нашего проекта. Т.е. в любой момент, мы с легкостью можем отказаться от каких-либо изменений. При работе с Carthage нужно создать Cartfile. В этом документе мы указываем строку подключения нашего фреймворка и главный заголовочный файл. На этом установка VK SDK завершена.

```
#import <VKSdkFramework/VKSdkFramework.h>
```

 строка подключения

```
#import <VKSdkFramework/VKSdkFramework.h>
```

 главный заголовочный файл

1.2 Инициализация SDK. Авторизация пользователя

Инициализируем SDK с помощью нашего APP_ID для любого делегата:

```
[VKSdk initializeWithDelegate:delegate andAppId:YOUR_APP_ID];
```

Для авторизации пользователя создаем Cocoa Touch Class AuthViewController.swift и связанный с ним Storyboard AuthViewController.storyboard. На нашем экране входа создаем кнопку, по нажатию на которую будем переходить на форму входа в наше приложение. Далее мы изменяем стиль кнопки и с помощью constraints закрепляем кнопку на экране, а также устанавливаем фиксированную высоту. Затем привязываем нашу кнопку к файлу AuthViewController.swift и добавляем IBAction. На данном этапе функция не делает ничего полезного, а просто выводит стандартное сообщение.

После этого нам необходимо проверить, доступна ли предыдущая сессия. Для этого используем асинхронный метод wakeUpSession. Данный метод принимает один параметр – Scope. С этим параметром передается список прав доступа. Права доступа определяют возможность использования токенов для работы с тем или иным разделом данных. Мы будем использовать offline токен. Этот токен позволяет иметь доступ к API в любое время. Если обобщить, то метод wakeUpSession пытается извлечь токен из хранилища и проверить, разрешено ли приложению использовать токен доступа к пользователю.

```
NSArray *SCOPE = @[@"friends", @"email"];

[VKSdk wakeUpSession:SCOPE completeBlock:^(VKAuthorizationState state,
NSError *error) {
    if (state == VKAuthorizationAuthorized) {
        // Authorized and ready to go
    } else if (error) {
        // Some error happend, but you may try later
    }
}];
```

После вызова асинхронного метода, проверяем значения параметра `VKAuthorizationState`. Он может находиться в одном из следующих состояний:

- 1) `VKAuthorizationInitialized` – означает, что SDK готов к работе, и мы можем авторизовать пользователя с помощью метода `+authorize`.
- 2) `VKAuthorizationAuthorized` – означает, что с предыдущей сессией все в порядке и мы можем продолжить работу с данными пользователя.
- 3) `VKAuthorizationError` – означает, что во время проверки произошла ошибка.

Проверка осуществляется с помощью следующего кода:

```
[VKSdk wakeUpSession:SCOPE completeBlock:^(VKAuthorizationState state,
NSError *err) {
    if (state == VKAuthorizationAuthorized) {
        // authorized
    } else {
        // auth needed
    }
}];
```

Переходим непосредственно к авторизации. За авторизацию отвечает метод `VKSdk authorize` с параметром `scope`. Логика переключения между экранами будет обрабатываться в файле `AppDelegate` с помощью делегирования. Для этого создаем делегат `AuthServiceDelegate`. И для него создаем три функции: `authServiceShouldShow`, `authServiceSignIn`, `authServiceDidSignInFail`. Метод `authServiceShouldShow` в качестве параметра будет принимать `viewController`. Данный метод будет отображать интерфейс для входа в приложения (либо для регистрации, если у пользователя нет аккаунта).

После завершения процесса авторизации мы хотим, чтобы наше приложение отображало новостную ленту. Для этого создаем новый `CocoaTouchClass` с именем `FeedViewController` и новый `storyboard` с таким же именем. Сам `FeedViewController` мы хотим отобразить как `NavigationViewController`.

1.3 Networking logic. Вызов методов API

На текущем этапе мы хотим получать данные из Интернета (с сервиса VK). Сервис VK использует API (Application programming interface), который возвращает нам данные в формате json. Чтобы обратиться к методу API ВКонтакте, нам необходимо выполнить POST или GET запрос следующего вида:

```
Chttps://api.vk.com/method/METHOD_NAME?PARAMETERS%access_token=ACCESS_TOKEN%v
=V
```

Он состоит из нескольких частей:

`method_name` (обязательно) – название метода API, к которому хотим обратиться.

`parameters` (опционально) – входные параметры соответствующего метода API, последовательность пар `name=value`, разделенных амперсандом.

`access_token` (обязательно) – ключ доступа.

`V` (обязательно) – используемая версия API. Использование этого параметра применяет некоторые изменения в формате ответа различных методов. Этот параметр следует передавать со всеми запросами.

Для получения новостной ленты мы будем использовать метод `newsfeed.get`, который возвращает данные, необходимые для показа списка новостей для текущего пользователя. Сетевую логику в нашем XCode проекте мы будем описывать в отдельном файле `NetworkService.swift`. В нем создаем функцию `getFeed()`. Для того, чтобы создать веб-адрес, воспользуемся классом `UrlComponents`. Данный класс позволяет создавать URL из составных частей:

`Scheme` – протокол (`http` или `https`)

`Host` – `api.vk.com`

`Path` – ключевой фрагмент, определяет к какому методу обращаемся.

`QueryItems` – параметры

Наш первый запрос примет следующий вид:

```
https://api.vk.com/method/newsfeed.get?filters=post,photo&access_token=0dfh38576jdue7493034jgkgld9595dk44&v=5.92
```

Далее мы будем совершенствовать получение данных. Создадим структуру, которая будет хранить в себе статические сегменты URL запроса (`scheme`, `host`). Определим метод `request`, который будет создавать URL из представленного `path` и `parameters`. Затем этот URL используется для создания `URLRequest`, а этот `URLRequest` будет использоваться для создания `session.dataTask`. Данная задача затем выполняется, запускается `http` запрос и результат возвращается через `completion` блок. И на выходе мы получаем либо данные, либо сообщение об ошибке. Всю эту информацию мы получаем в `json` формате.

1.4 Парсинг json

Создадим модель данных `FeedResponse`, которая будет представлять массив новостных постов. А затем создадим модель данных, которая уже будет представлять непосредственно сам пост с новостью. Эти модели данных будут реализованы как структура и подписаны под протокол `Decodable`, чтобы мы могли преобразовывать json в нашу модель данных.

```
struct FeedResponse: Decodable {
    var items: [FeedItem]
}

struct FeedItem: Decodable {
    let sourceId: Int
    let posted: Int
    let text: String?
    let date: Double
    let comments: CountableItem?
    let likes: CountableItem?
    let reposts: CountableItem?
    let views: CountableItem?
}

struct CountableItem: Decodable {
    let Count: Int
}
```

Прочтение json формата будем осуществлять с помощью `JsonDecoder()`.

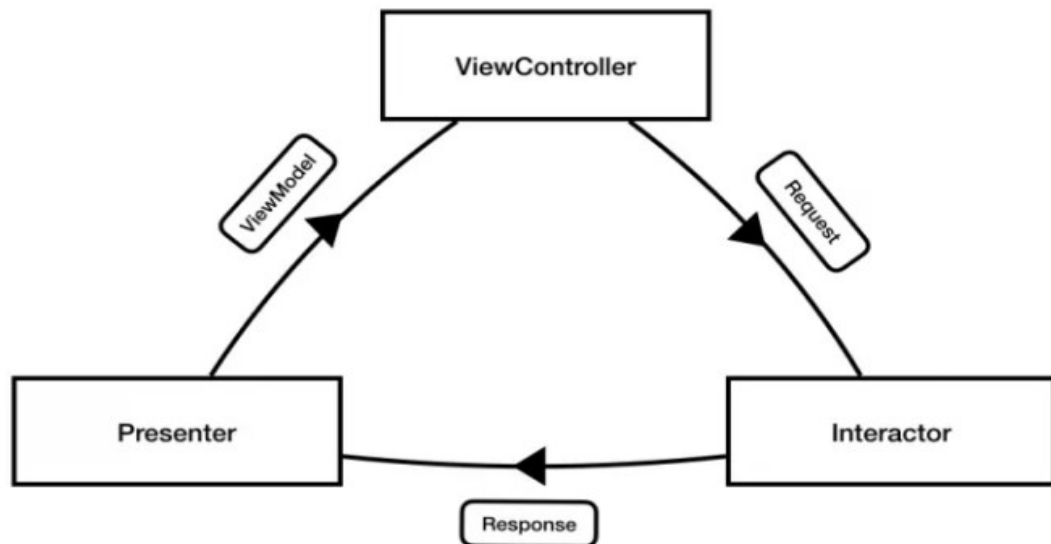
1.5 Архитектура приложения

До текущего момента мы сделали, безусловно, нужную, но не самую интересную работу. И все самое интересное будет впереди. Мы будем создавать посты со всей информацией (количество просмотров, лайков и т.д.), будем загружать и кэшировать изображения, регулировать размер ячейки в зависимости от размера поступающих данных (фотографии, текста) и наведем красоту в нашем приложении (красивые кнопки, градиент и т.д.).

В современной разработке необходимо следовать шаблонам проектирования. Следование определенным шаблонам облегчает разработку, код становится более понятным. В нашем приложении мы будем применять архитектуру `Clean Swift`.

Из чего же состоит архитектура `Clean Swift`? Данная архитектура включает в себя 8 модулей, но из них активно использовать мы будем лишь 3 –

ViewController, Interactor, Presenter (сокращенно VIP). Взаимодействие между тремя этими файлами происходит циклично и передача данных основана на протоколах. Все это позволяет при изменении какого-либо компонента просто подменить его на другой.



Поясним на простейшем примере:

Например, пользователь нажимает на кнопку. Жест касания входит через `IBAction` во `ViewController`. `ViewController` создает объект запроса и отправляет его в `Interactor`. `Interactor` ловит этот запрос и осуществляет какой-то конкретный сценарий в соответствии с бизнес логикой, создает объект результата и передает его в `Presenter`. `Presenter` формирует объект с отформатированными для отображения пользователю данными и отправляет его во `ViewController`. И `ViewController` отображает результаты пользователю.

Теперь более подробно про 3 компонента.

ViewController

Отвечает за отображение подготовленных данных и взаимодействие с пользователем, но не отвечает за положение `view` и его `layouts`.

`ViewController` действует как делегат для всего слоя представления. Он знает, когда текст в `UITextField` был изменен или пользователь щелкнул по ячейке.

Interactor

Содержит бизнес логику. Может делать сетевые запросы, обращаться к базе данных. После отработки логики, Interactor должен передать данные для их подготовки в Presenter.

Presenter

Presenter обрабатывает данные для показа пользователю. Отображает ответ в виде моделей, подходящих для отображения. Затем он передает ViewModel обратно во ViewController для отображения пользователю.

Далее мы переходим к созданию ячеек новостной ленты. Для этого нам понадобится UITableView.

1.6 Создание ячеек с помощью XIB файлов

На данном этапе наша ячейка с новостью будет состоять из трех составных частей: верхняя часть, где будет храниться заголовок новости и дата создания новости, основная часть, которая будет содержать основной текст новости, и нижняя часть, в которой будет находиться информация о числе лайков, просмотров, комментариев и репостов. Для верхней и нижней части создаем два view, а для центральной части создаем label. Воспользуемся Autoresizing для закрепления положения на экране. После этого добавляем необходимые элементы из библиотеки объектов в наши составные части. Группируем элементы, настраиваем отступы. Получаем макет новостной ячейки.

1.7 Заполнение ячеек реальной информацией

Мы уже создали IBOutlet для каждого элемента ячейки. Теперь необходимо передать им данные. Создадим модель данных, которая будет содержать информацию для этих элементов. Создаем протокол FeedCellViewModel. В нем создаем все свойства, которыми обладает наша ячейка. Далее создаем функцию set, которой будет передаваться параметр, подписанный под протокол FeedCellViewModel. В этой функции происходит передача данных от параметра к IBOutlet. Эта функция будет вызываться в методе CellForRowAt.

Самое тяжелое в заполнении ячеек данными – это загрузка изображений. Все это делается в асинхронных потоках, т.к. например изображение может оказаться большим, а Интернет соединений медленным, и наше приложение просто «зависнет» на время загрузки изображения. Однако все изображения загружаются каждый раз, когда мы их запрашиваем. Но что если попадаете одна и та же картинка? Она будет загружена каждый раз заново. Это не очень хорошо с точки зрения производительности. Поэтому мы будем кэшировать

изображения. Также в нашем приложении мы будем оптимизировать картинки по размеру и округлять их, если данная картинка является изображением пользователя или группы.

1.8 Дизайн приложения

Для лучшего вида нашего приложения (в частности наших ячеек) мы будем использовать Card view. Далее мы будем реализовать функционал для автоматического определения размерности новостных ячеек в зависимости от объема поступающего контента. Для этого используем структуру CGRect, которая содержит в себе размеры объекта, а также координаты, где объект должен находиться.

Одним из условий нашего курсового проекта являлась разработка не только через storyboard, но и через код. Например, мы закрепляли элементы на экране с помощью класса NSLayoutConstraint. Также мы используем Auto Layout. Благодаря этому исчезает необходимость в подгонке размеров приложения под различные устройства. Кроме того, в приложении присутствует функционал, благодаря которому новости большого объема отображаются в ленте не полностью, а частично. Но мы можем развернуть новость полностью благодаря реализации кнопки «Развернуть полностью».

Также в посте может присутствовать более одной фотографии и нам нужно отобразить их все. Для этого используем UICollectionViews. Однако наша галерея фотографий будем все еще отображаться не совсем корректно (набор очень маленьких фотографий), поэтому будем использовать UICollectionViewCustomLayout.

Далее, как и в любом приложении, в котором присутствует авторизация, нам нужно отображать информацию о текущем пользователе. В нашем приложении мы будем показывать аватар пользователя в правом верхнем углу.

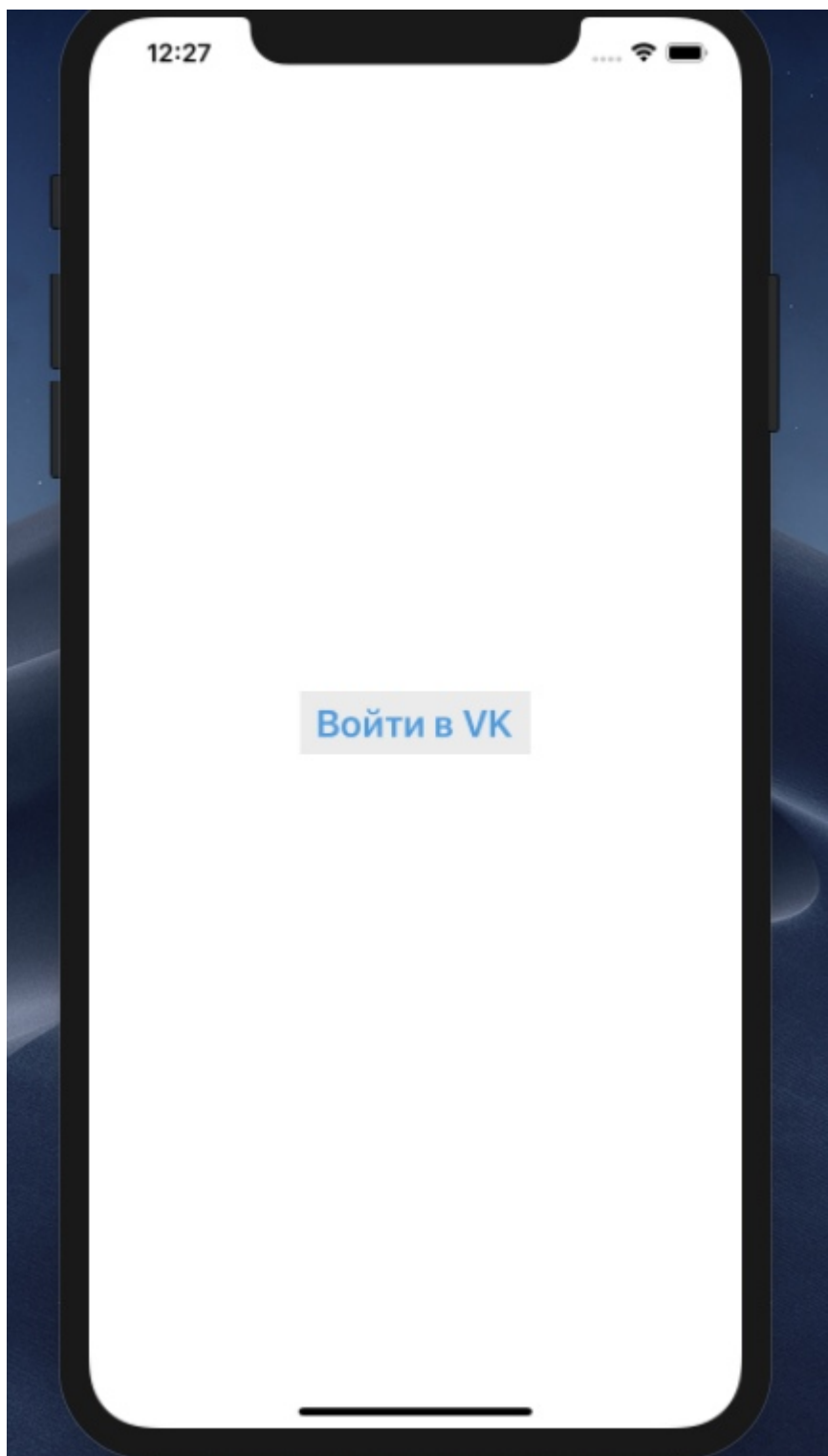
И в самом конце добавляем Gradient View для заднего фона нашего приложения.

1.9 Дополнительная функциональность

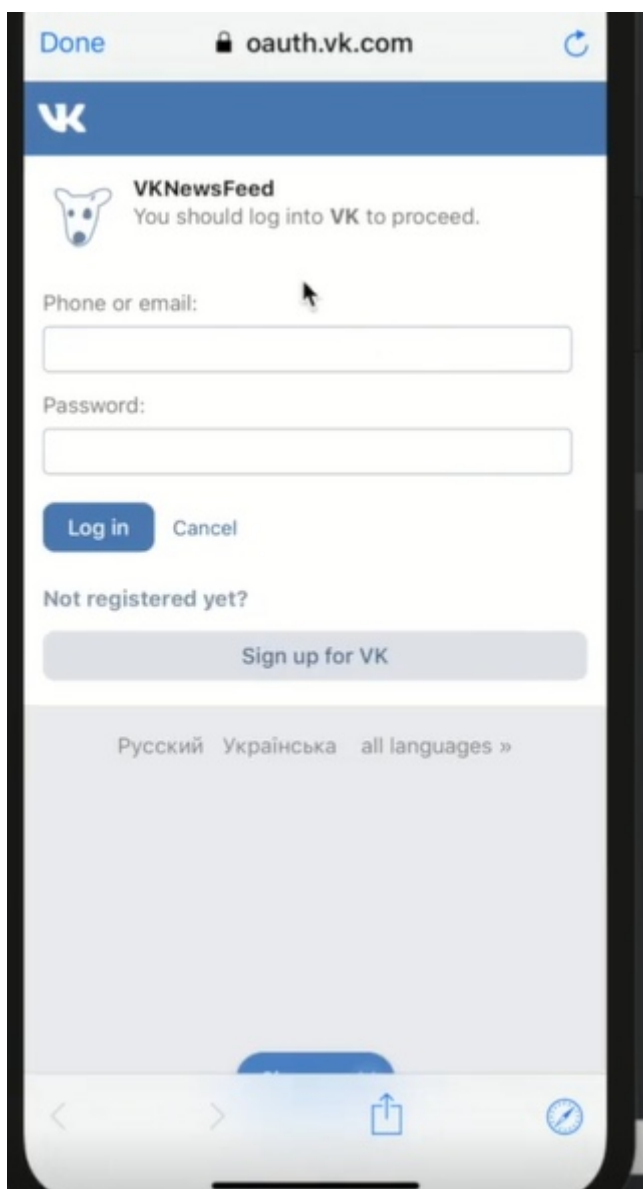
Также в приложении реализована такая функциональность, как обновление постов (например, если добавились новые новости) и загрузка предыдущих постов, которые оказались скрытыми новыми новостями.

ГЛАВА 2. РЕЗУЛЬТАТЫ РАБОТЫ

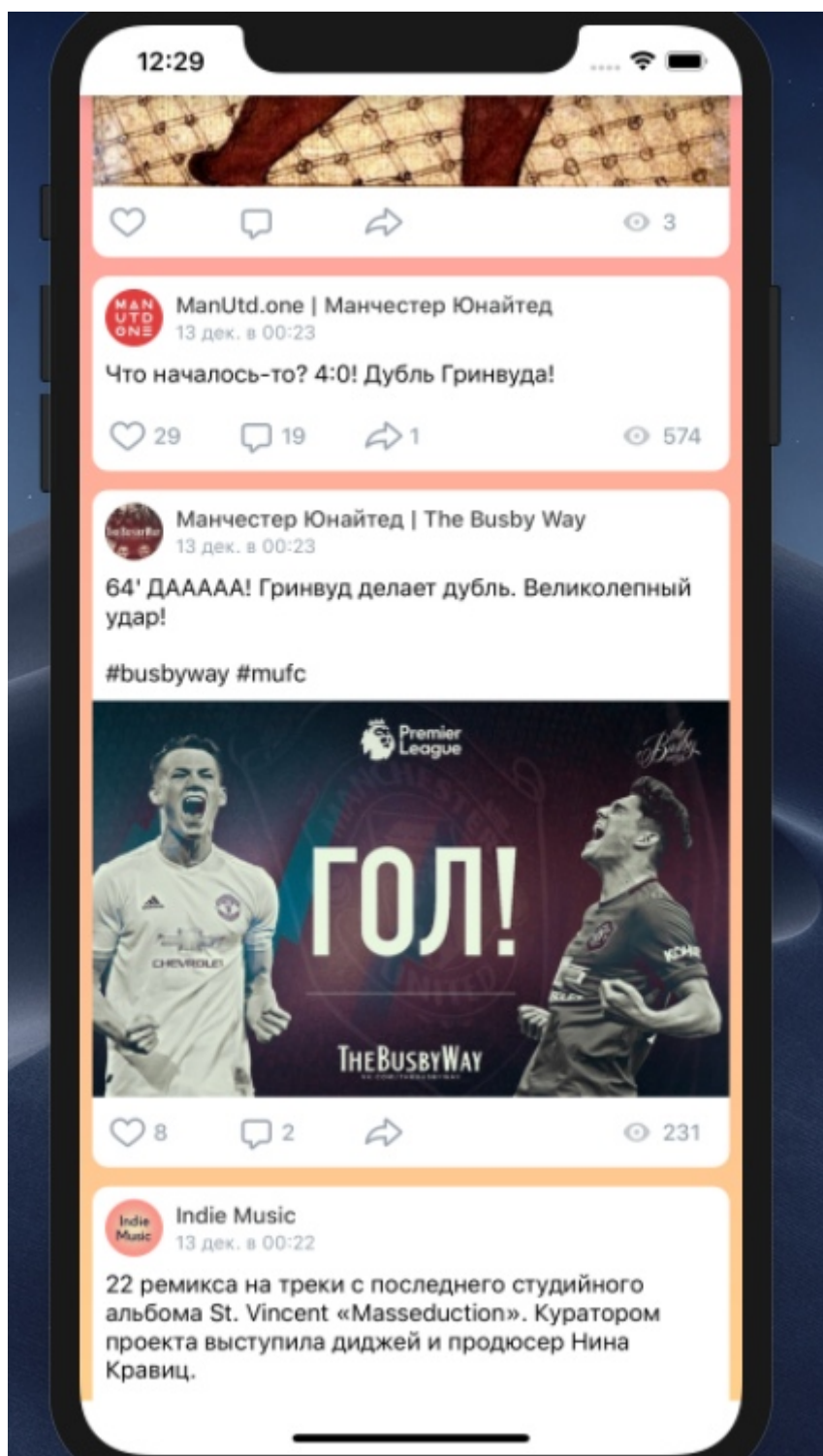
Стартовый экран



Авторизация через веб-интерфейс



Новостная лента



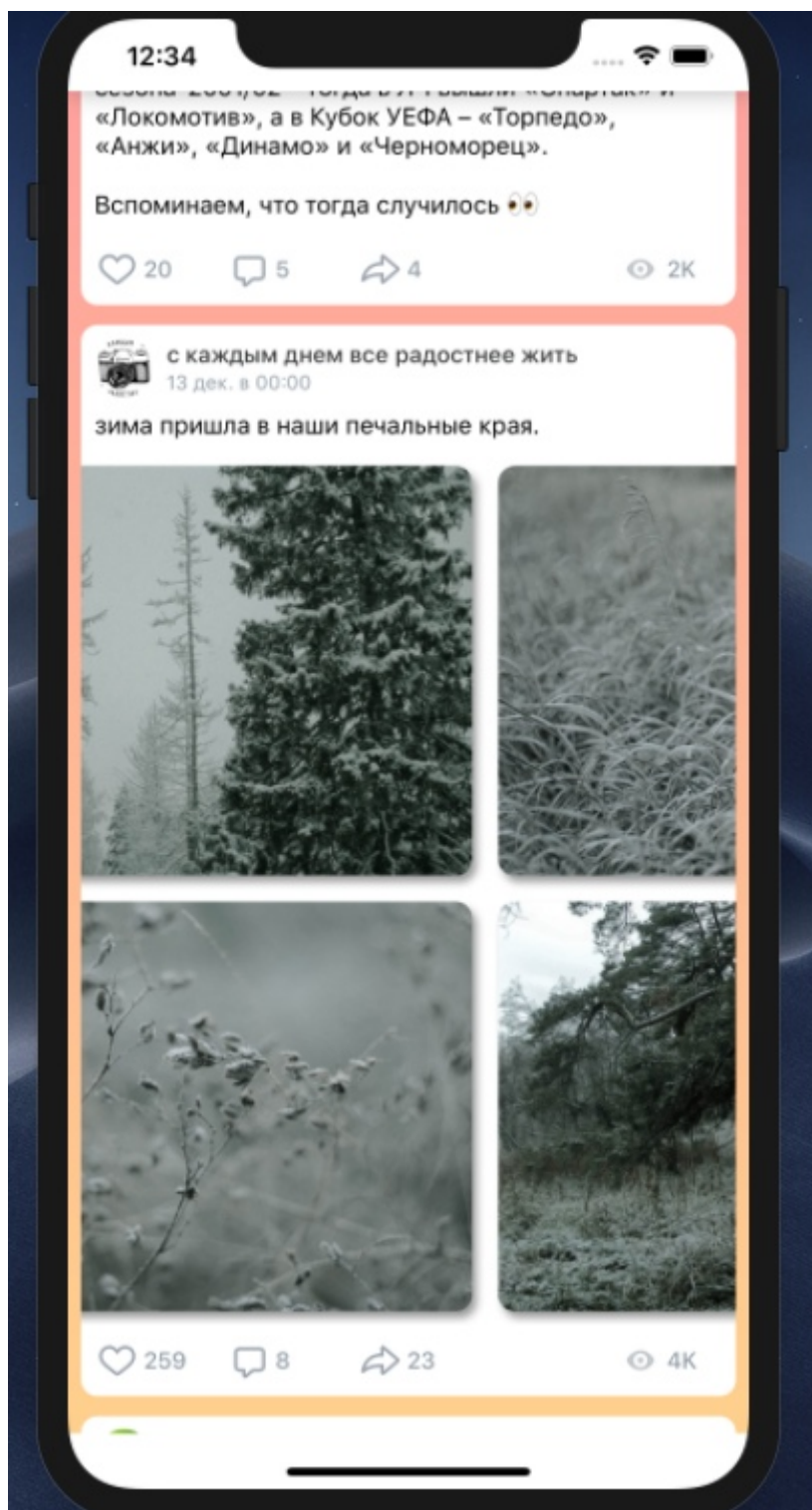
Пост большого размера



Пост большого размера полностью



Пост с галереей изображений



ЗАКЛЮЧЕНИЕ

Таким образом, мы создали простое новостное приложение (новостная лента) для социальной сети. В процессе разработки были затронуты все основные темы, которые стояли в плане обучения.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Интернет-адрес: <https://developer.apple.com>
2. Усов В. Основы разработки приложений под iOS и macOS
3. Интернет-адрес: https://vk.com/dev/ios_sdk