Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

TWITTER-КЛИЕНТ ДЛЯ ПЛАТФОРМЫ IOS

БГУИР КР 1-40 01 01 204 ПЗ

Студент: гр. 951002 Будович И.В.

Руководитель: асс. Мельникова Е.В.

Минск 2021

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2021 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Будовичу Ивану Витальевичу

1. Тема работы Twitter-клиент для платформы iOS

2. Срок сдачи студентом законченной работы––01.06.2021 г.–––

3. Исходные данные к работе язык программирования Swift

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.

1. Анализ прототипов, литературных источников и моделирование предметной области;

2. Анализ требований к программному средству и разработка функциональных требований;

3. Проектирование программного средства;

4. Создание (конструирование) программного средства;

5. Тестирование, проверка работоспособности и анализ полученных результатов;

6. Руководство по использованию;

Список используемой литературы

Заключение

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. " Twitter-клиент для платформы iOS ", А1, схема программы, чертеж.

6. Консультант по курсовой работе

Мельникова Е. В.

7. Дата выдачи задания 10.02.2021

8. Календарный график работы над курсовой работой на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1 к 01.03.2021 – 15 % готовности работы;

разделы 2, 3 к 15.03.2021 – 30 % готовности работы;

разделы 4, 5 к 15.04.2021 – 60 % готовности работы;

раздел 6 к 15.05.2021 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 20.05.2021– 100 % готовности работы.

Защита курсовой работы с 01.06.2021 по 09.06.2021 г.––––––––––––––––––––

РУКОВОДИТЕЛЬ–––––– Мельникова Е. В.

(подпись)

Задание принял к исполнению 10.02.2021–––\_\_\_\_\_\_––

(дата и подпись студента)

Содержание

[Содержание 4](#_Toc72677687)

[Введение 6](#_Toc72677688)

[1 Анализ прототипов, литературных источников и моделирование предметной области 7](#_Toc72677689)

[1.1 Протокол HTTP 7](#_Toc72677690)

[1.2 Клиентское приложение для социальной сети 8](#_Toc72677691)

[1.3 Twitter API 8](#_Toc72677692)

[1.4 Основные Twitter-клиенты для платформы iOS. 9](#_Toc72677693)

[1.5 Постановка задачи 12](#_Toc72677694)

[2 Анализ требований к программному средству и разработка функциональных требований 14](#_Toc72677695)

[2.1 Описание функциональных требований 14](#_Toc72677696)

[2.2 Описание прочих требований 14](#_Toc72677697)

[3 Проектирование программного средства 16](#_Toc72677698)

[3.1 Разработка алгоритма авторизации 16](#_Toc72677699)

[3.2 Разработка алгоритма взаимодействия пользователя с новостной лентой 16](#_Toc72677700)

[3.3 Разработка алгоритма загрузки изображений 17](#_Toc72677701)

[4 Конструирование программного средства 18](#_Toc72677702)

[4.1 Выбор архитектуры приложения 18](#_Toc72677703)

[4.2 Проектирование графического интерфейса 18](#_Toc72677704)

[4.3 Проектирование сетевого взаимодействия 19](#_Toc72677705)

[4.4 Структура модулей программы 20](#_Toc72677706)

[4.5 Описание класса AppDelegate 20](#_Toc72677707)

[4.6 Описание класса SceneDelegate 21](#_Toc72677708)

[4.7 Описание класса LoginController 22](#_Toc72677709)

[4.8 Описание класса FeedController 23](#_Toc72677710)

[4.9 Описание класса TweetCreationController 25](#_Toc72677711)

[4.10 Описание класса UserController 26](#_Toc72677712)

[4.11 Описание класса UserListController 28](#_Toc72677713)

[4.12 Описание структур модели данных 29](#_Toc72677714)

[4.13 Описание классов работы с сетью 30](#_Toc72677715)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 32](#_Toc72677716)

[5.1 Тестирование алгоритма авторизации 32](#_Toc72677717)

[5.2 Тестирование функциональности новостной ленты 33](#_Toc72677718)

[5.3 Тестирование взаимодействия с профилем пользователя 35](#_Toc72677719)

[6 Руководство по использованию 37](#_Toc72677720)

[Заключение 40](#_Toc72677721)

[Список литературы 41](#_Toc72677722)

[Приложение A 42](#_Toc72677723)

[Приложение Б 43](#_Toc72677724)

[Приложение В 46](#_Toc72677725)

[Листинг файла AppDelegate.swift 46](#_Toc72677726)

[Листинг файла SceneDelegate.swift 46](#_Toc72677727)

[Листинг файла ImageDownloadOperation.swift 48](#_Toc72677728)

[Листинг файла ImageDownloader.swift 49](#_Toc72677729)

[Листинг файла TwitterService.swift 50](#_Toc72677730)

[Листинг файла Tweet.swift 50](#_Toc72677731)

[Листинг файла User.swift 51](#_Toc72677732)

[Листинг файла LoginController.swift 52](#_Toc72677733)

[Листинг файла FeedController.swift 53](#_Toc72677734)

[Листинг файла MainTabBarController.swift 59](#_Toc72677735)

[Листинг файла UserListController.swift 60](#_Toc72677736)

[Листинг файла TwitterCreationController.swift 63](#_Toc72677737)

[Листинг файла UserController.swift 64](#_Toc72677738)

[Листинг файла TweetCell.swift 72](#_Toc72677739)

[Листинг файла UserCell.swift 76](#_Toc72677740)

Введение

В современном мире возросла роль социальных сетей: они используются для общения, обмена данными и для рекламы. Основными социальными сетями в 2021 году являются Facebook, Twitter, Instagram. Важно предоставить пользователю удобный доступ к сетевому ресурсу на любом устройстве, в том числе и на мобильном.

Многие социальные сети позволяют стороннему разработчику получить ограниченный доступ к API для взаимодействия клиента с сервером с помощью протокола HTTP/HTTPS и таких методов как GET, POST и др.

Для своей курсовой работы я выбрал социальную сеть Twitter.

В ходе выполнения курсовой работы я постараюсь найти решение таким техническим вопросам как:

1. Работа с протоколом HTTP/HTTPS;
2. Использование сторонних API для взаимодействия с сервером.
3. Особенности разработки ПО для мобильных платформ.
4. Работа с асинхронными запросами.

В этой пояснительной записке отображены следующие этапы написания курсовой работы:

1. Анализ прототипов, литературных источников и моделирование предметной области;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Создание (конструирование) программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию.
7. Анализ прототипов, литературных источников и моделирование предметной области
   1. Протокол HTTP

Протокол HTTP появился как инструмент в рамках технологии Веб, которая обеспечивает доступ к контенту в глобальной сети Интернет.

* + 1. Метод GET

GET - это запрос информации, расположенной на сервере по указанному URL. GET - наиболее распространенный метод поиска с помощью браузеров документов для визуализации. Результат запроса GET может представлять собой, например, файл, доступный для сервера, результат выполнения программы или CGI-сценария, выходную информацию аппаратного устройства и т.д.

Если клиент пользуется в своем запросе методом GET, сервер отвечает строкой состояния, заголовками и затребованными данными. Если сервер не может обработать запрос вследствие ошибки или отсутствия полномочий, он, как правило, посылает в информационном разделе ответа текстовое пояснение.

* + 1. Метод POST

Метод POST позволяет посылать на сервер данные в запросе клиента. Эти данные направляются в программу обработки данных, к которой сервер имеет доступ (например, в CGI-сценарий). Метод POST может использоваться во многих приложениях. Например, его можно применять для передачи входных данных для: сетевых служб (таких как телеконференции); программ с интерфейсом в виде командной строки; аннотирования документов на сервере; выполнения операций в базах данных.

Данные, посылаемые на сервер, находятся в теле содержимого запроса клиента. По завершении обработки запроса POST и заголовков сервер передает тело содержимого в программу, заданную URL.

* 1. Клиентское приложение для социальной сети

Клиентское приложение – это программа, работающая на компьютере пользователя и обеспечивающая интерактивное взаимодействие системы с пользователем.

Для клиентского приложения для социальной сети Twitter характерен следующий функционал:

* Просмотр выложенных пользователем записей, которые могут содержать ссылки, фото и видео, так называемых «твитов».
* Создание собственных «твитов».
* Отметка понравившихся «твитов», называемая «лайком».
* Возможность комментирования «твитов».
* Отправление сообщений пользователю и просмотр полученных сообщений.
* Возможность подписаться на другого пользователя.

Приложения для мобильных устройств должны поддерживать работу с сенсорным экраном и быть адаптивными.

* 1. Twitter API

API Twitter можно использовать для программного извлечения и анализа данных, а также для участия в работе Twitter [1].

Этот API обеспечивает доступ к множеству различных ресурсов, включая следующие:

* твиты;
* пользователи;
* прямые сообщения;
* списки;
* тенденции;
* средства массовой информации;
* места;

API Twitter в настоящее время состоит из двух поддерживаемых версий, а также разных уровней доступа.

Стандартная версия Standart v1.1 предоставляет доступ к следующим ресурсам:

* публикация и взаимодействие с твитами;
* получение временных рамок;
* публикация и получение личных сообщений;
* управление и извлечение информации из общедоступного аккаунта;
* создание списков и управление ими;
* подписка, поиск и доступ к пользователям;
* получение «трендов»[1].
  1. Основные Twitter-клиенты для платформы iOS.
     1. Официальное приложение Twitter



Рисунок 1.1 – снимок экрана официального приложения Twitter

Официальное приложение [3] имеет весь функционал, доступный пользователю с помощью веб-сайта.

Преимущества:

* Не имеет ограничений по запросам.

Недостатки:

* Встроенная реклама в виде продвигаемых записей.
  + 1. UberSocial



Рисунок 1.2 – снимок экрана приложения UberSocial

Клиент UberSocial [4] считается одним из самых популярных клиентов Twitter.

Преимущества:

* Отсутствует реклама.

Недостатки:

* Неудобный интерфейс.
* Приложение не получает обновления около двух лет.
  + 1. Twitterific



Рисунок 1.3 – снимок экрана приложения Twitterific

Приложение Twitterific [5] является удобным аналогом официального приложения

Преимущества:

* Отсутствие продвигаеммых рекламных записей.
* Приятный нативный пользовательский интерфейс.

Недостатки:

* Реклама в виде банера.
  1. Постановка задачи

Программное средство должно представлять собой клиентское приложение для мобильного устройства под управлением операционной системы iOS. Программа должна обеспечивать выполнение перечисленных ниже функций:

* + авторизация пользователя;
  + работа с твитами;
  + иные взаимодействия с другими пользователями.

Программное средство должно использовать сетевые протоколы, взаимодействовать с сервеной частью. Действия, выполняемые пользователем на клиентском приложении, можно отследить с помощью официального приложения или веб-сайта социальной сети.

Для нативной разработки на платформе iOS используются два языка программирования: Objective-C и Swift. Я выбрал язык программирования Swift в связи с простым и удобными синтаксисом, типобезопасностью и удобной работой с парсером формата JSON.

В качестве среды разработки я выбрал Xcode, так как он поддерживает создание отдельных контроллеров с помощью редактора интерфейса.

1. Анализ требований к программному средству и разработка функциональных требований
   1. Описание функциональных требований

Программное средство для функциональности должно работать с аккаунтом авторизованного пользователя. Twitter API не позволяет сторонним приложениям на прямую требовать от пользователя ввода логина и пароля через приложение. Для этого должен использоваться протокол OAuth 1.0a с вводом пользовательских данных через защищенный браузер, в случае платформы iOS – Safari. [1][2]

Twitter – социальная сеть для микроблогинга – общения с помощью небольших записей (твитов), а значит, работа с записями должна быть обязательна в программном средстве. Основные виды работы с записями – создание записей, их оценка и ретвит.

Также должна быть возможность взаимодействия с пользователями, в частности, просмотр доступных записей, просмотр пользователей, на которых он подписан и которые подписаны на него.

Для моего программного средства я определил следующие функциональные требования:

* 1. Программное средство должно поддерживать авторизацию пользователя, при повторном входе в приложение авторизация не требуется, кроме случая, когда пользователь вышел из аккаунта во время предыдущего сеанса работы;
  2. Программное средство должно отображать новостную ленту Twitter, в которой пользователь может выбрать понравивуюся запись, сделать ретвит записи.
  3. Программное средство должно иметь возможность создания новых записей пользователем;
  4. Программное средство должно иметь возможность перехода на страницы других пользователей;
  5. **Описание прочих требований**

При выполнении работы определяются дополнительные требования:

* Приложение выглядеть приемлимо на устройствах разного размера.
* Программное средство должно поддерживать светлое и темное оформление, которое зависит от текущих настроек системы.
* Программное средство должно поддерживать портретный (высота экрана больше ширины) режим.
* Интерфейс программного средства должен быть простым и удобным.
* Программное средство должно работать на устройстве под управлением операционной системы iOS версии 13 и новее.

1. Проектирование программного средства
   1. Разработка алгоритма авторизации

Разрабатываемое средство должно иметь доступ к аккуанту пользователя в социальной сети Twitter. Ранее сказано, что сервис не позволяет сторонним разработчикам напрямую принимать логин и пароль и посылать их на сервер. Для обхода этого ограничения используется протокол OAuth 1.0a. [2]

Для работы с API используются четыре токена: два токена разработчика и два токена пользователя. Токены разработчика известны заранее, их программист получает при регистрации своего приложения на портале разработчика Twitter. Там же разработчик указывает адрес обратного вызова.

При запуске приложения проверяется, нет ли на устройстве сохраненных токенов. Если их нет, то загружается экран с кнопкой «Войти при помощи Twitter». При нажатии на нее, открывается модальное окно браузера Safari с вкладкой на сайте Twitter API. Там пользователь вводит свои логин и пароль. После чего срабатывает переход по адресу обратного вызова, указывающего на приложение. Модальное окно закрывается и приложение либо уведомляет об ошибке, либо получает токены пользователя, сохраняет их в файл и загружает главное окно.

Если на устройстве есть токены, то посылается запрос с подтверждением их актуальности, хотя обычно токены Twitter не устаревают. Если пришло подтверждение, то загружается главное окно, иначе загружается экран с кнопкой «Войти при помощи Twitter». [6]

Схема алгоритма представлена на рисунке Б.1.

* 1. Разработка алгоритма взаимодействия пользователя с новостной лентой

Новостная лента – ключевая часть программного средства. В ней находятся записи. Важно предусмотреть правильное взаимодействие пользователя с лентой и программного средства с сетевыми запросами.

Изначально, пользователю загружается некоторое число записей, данные загружаются в модель, интерпретируются графическим интерфейсом.

Если пользователь просмотрел все записи, необходимо отправить запрос и загрузить еще записи. При этом нужно исключить возможность повторного запроса на момент загрузки с помощью логического флага.

Пользователь может захотеть обновить ленту, прокрутив сверху вниз вверху таблицы. В таком случае, отправляется запрос, в ответе на который приходят либо пустые записи, либо данные в формате JSON, которые загружаются в модель.

Пользователь может также отметить понравившуюся запись, убрать отметку, сделать «ретвит» и отменить его. В каждом случае посылаются свои запросы, в случае их успешного выполнения, обновляется пользовательский интерфейс.

Взаимодействие с иконкой пользователя должно открыть его профиль.

Схема алгоритма представлена на рисунке Б.2.

* 1. Разработка алгоритма загрузки изображений

Во время загрузки данных о записи, программное средство также получает ссылку на иконку пользователя, которую нужно вставить в ячейку таблицы. При этом нужно иметь в виду, что иконки могут повторятся, и не имеет смысла загружать несколько картинок по одной ссылке, с другой стороны, методы настройки ячеек будут вызываться быстрее, чем загрузка одной картинки. В таком случае, имеется смысл отслеживать начавшиеся загрузки с помощью операций. Операции будут помещаться в словарь по ссылке-ключу. Когда картинка загрузилась, она помещается в кеш, а операция удаляется из словаря. Таким образом исключается загрузка нескольких картинок по одной ссылке.

Схема алгоритма представлена на рисунке Б.3.

1. Конструирование программного средства
   1. Выбор архитектуры приложения

В качестве архитектурного решения для своего программного средства я выбрал архитектуру MVP, известную также как Модель – Представление –Представитель [7].

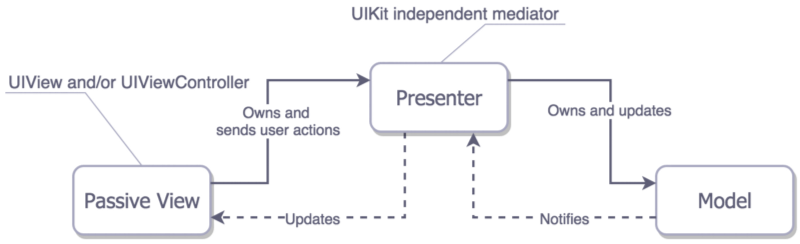


Рисунок 4.1 – схема шаблона MVP

Представление – это элементы графического интерфейса, они отвечают за отображение. В представление включается контроллер, принимающий события от пользователя и перенаправляющий их Представителю.

Представитель – класс, обновляющий Представление и хранящий модель. Он взаимодействует с сетевым слоем, таким образоб разгружая контроллер.

Модель – это классы и структуры, в которых хранятся данные, например, о записях, и которые реализуют бизнес-логику приложения, например, взаимодействуют с сетями.

* 1. Проектирование графического интерфейса

Для создания графического интерфейса я использовал фреймворк UIKit – основной фреймворк для разработки интерфейсов для iOS и iPadOS[8].

Программное средство имеет одно окно, но большое количество контроллером, между которыми происходит навигация.

Изначально, при загрузке приложения пользователь может переключаться между двумя контроллерами – новостной лентой и экраном пользователя. При взаимодействии пользователя с элементами интерфейса, могут создаваться новые контроллеры поверх основных в навигационном контроллере. При этом пользователь может вернуться к корневым контроллерам с помощью полосы навигации. Пользователь также может переключиться между двумя основными навигационными контроллерами.

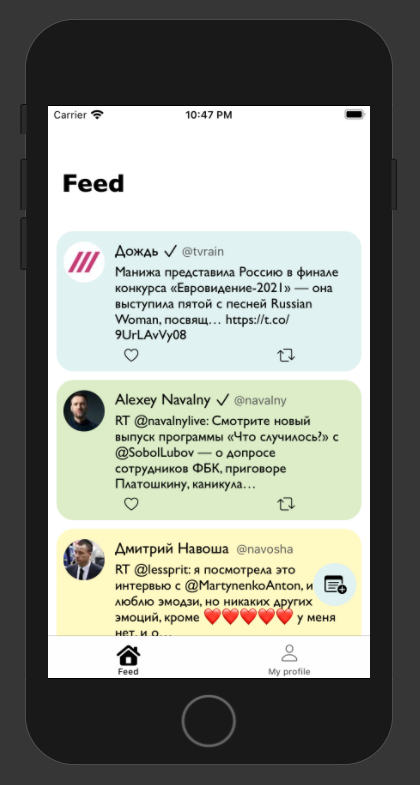


Рисунок 4.2 – новостная лента, внизу переключение между другим контроллером навигации

* 1. Проектирование сетевого взаимодействия

В пункте 3.3 упоминалось об операциях, которые используются для загрузки изображений. Операции реализуют класс Operation, от которого необходимо наследоваться, чтобы установить свои задачи. Для загрузки по сети используются системные классы URLSession и URLSessionDataTask. [8]

Достоинством языка программирования Swift можно считать развитые лямбда-выражения – замыкания, с помощью которых можно выполнять задачи не сразу, а после выполнения другой операции – загрузки картинки или успешного запроса из TwitterAPI.

После изучения TwitterAPI я решил воспользоваться сторонней библиотекой Swifter для упрощения взаимодействия с API. Внутри он использует системные классы, как, например, ASWebAuthenticationService для авторизации.

* 1. Структура модулей программы

Программа состоит из следующих файлов:

1. Файлы делегата приложения и сцены AppDelegate.swift, SceneDelegate.swift;
2. Файлы классов для работы с сетями ImageDownloadOperation.swift, ImageDownloader.swift, TwitterService.swift;
3. Файлы структур для модели Tweet.swift, User.swift;
4. Файлы классов-контроллеров LoginController.swift, MainTabBarController.swift, UserController.swift, FeedController.swift, UserListController.swift, TwitterCreationController.swift.
5. Файлы классов-представителей LoginPresenter.swift, UserPresenter.swift, FeedPresenter.swift, UserListPresenter.swift, TwitterCreationPresenter.swift.
6. Файлы протоколов (интерфейсов) представления LoginViewProtocol.swift, UserViewProtocol.swift, FeedViewProtocol.swift, UserListViewProtocol.swift, TwitterCreationViewProtocol.swift.
7. Классы для представления TweetCell.swift, UserCell.swift.
   1. Описание класса AppDelegate

Класс AppDelegate создается автоматически и необходим для нормального функционирования приложений на основе фрейморка UIKit. Здесь пользователь может обрабатывать необходимые ему особенности приложения, как, например, настроить нижнюю панель главного меню или поведение приложения при открытии ссылки, что я и использовал при организации авторизации. Данное поведение описано в методе application (\_ app:, open url:, options:)

Таблица 4.1 Методы класса AppDelegate для работы с сетью

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| application (\_ app:, open url:, options:) | Обработка открывающейся ссылки | func application(\_ app: UIApplication, open url: URL, options: [UIApplication.OpenURLOptionsKey : Any] = [:]) -> Bool | app | Приложение |
| url | Ссылка |
| options | Дополнительные настройки открытия ссылки |

* 1. Описание класса SceneDelegate

Класс SceneDelegate также создается системной для запуска приложения. В новых версиях iOS он выполняет функцию AppDelegate. Здесь я переопределил поведение создания корневого контроллера: изначально приложенение загружает токены из файла, проверяет их с помощью запроса, в случае подтверждения открывается главное окно, иначе – окно авторизации.

Таблица 4.2 Методы класса SceneDelegate для работы с сетью

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| scene(\_, willConnectTo session:, options:) | Подключение к сцене (создание корневого контроллера для окна приложения) | func scene(\_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions: UIScene.ConnectionOptions) | scene | Сцена |
| session | Сессия сцены |
| connectionOptions | Дополнительные настройки подключения |

Продолжение таблицы 4.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| scene(\_:, openURLContexts:) | Метод обработки открывающейся ссылки | func scene(\_ scene: UIScene, openURLContexts URLContexts: Set<UIOpenURLContext>) | scene | Сцена |
| URLContexts | Контекст настроек открытия ссылки |
| isLoggedIn (completion:) | Проверка, существуют ли токены и актуальны ли они | func isLoggedIn(completion: @escaping(Bool) -> ()) | completion | Замыкание для открытия нужного окна в случае подтверждения или ошибки запроса |

* 1. Описание класса LoginPresenter

Данный представитель связан с контроллером (экраном) авторизации, который приложение открывает, если нужно провести авторизацию. Приложение отправляет запрос на авторизацию, открывается модальное окно браузера Safari.

Таблица 4.3 Методы класса LoginPresenter для работы с сетью

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| login() | Обработка регистрации по ссылке | func login() | - | - |
| save(credentials:) | Сохранение токенов пользователя | func save(credentials: Credential.OAuthAccessToken) | credentials | Токены пользователя |

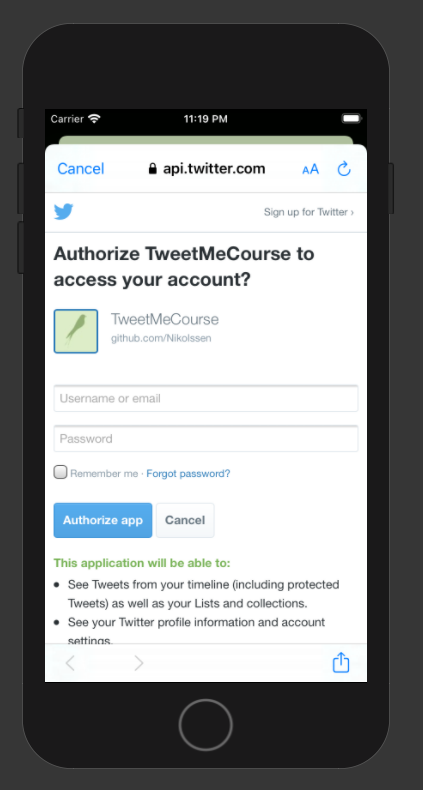


Рисунок 4.3 – модальное окно браузера Safari

* 1. Описание класса FeedPresenter

Класс реализует настройку данных для новостной ленты. Изначально загружается 10 записей, однако затем могут догрузиться более старые, а также пользователь может обновить ленту и загрузить более новые.

Таблица 4.4 Методы класса FeedPresenter

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| loadNewTweets() | Загрузка новых записей | func loadNewTweets() | - | - |
| loadMoreTweets() | Загрузка более старых записей | func loadMoreTweets() | - | - |

Продолжение таблицы 4.4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| loadInitialTweets(action:) | Загрузка первых записей | func loadInitialTweets(action: (() -> Void)?) | action | Замыкание для выполнения действий после загрузки |
| imageForTweet(at:, action: ) | Загрузка картинки для записи | func imageForTweet(at index: Int, action: @escaping ((UIImage) -> Void)) | index | Индекс записи, для которой загружается картинка |
| action | Замыкание для установления изображения |
| likeTweet(at: ) | Отправка запроса оценки записи | func likeTweet(at index: Int) | index | Индекс записи, которую нужно оценить |
| retweetTweet(at : ) | Отправка запроса ретвита записи | func retweetTweet(at index: Int) | index | Индекс записи, ретвит которой нужно сделать |

* 1. Описание класса TweetCreationPresenter

Класс отвечает за создание записи пользователем, он посылает строку с сообщением на сервер с помощью POST-запроса.

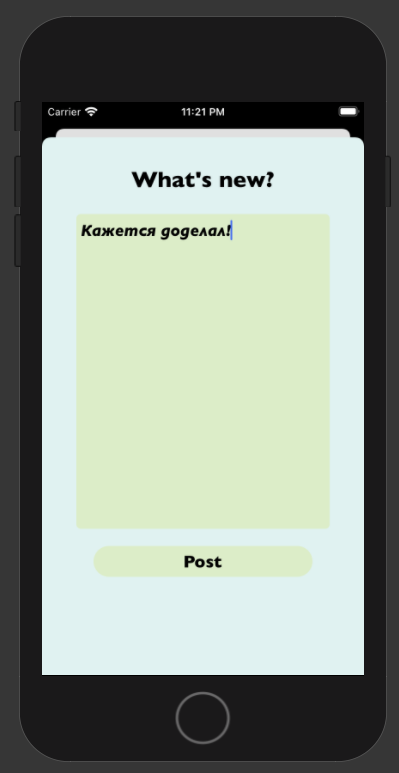


Рисунок 4.4 – контроллер создания записи

Таблица 4.5 Методы класса TweetCreationPresenter

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| createTweet(text: ) | Отправка запроса на создание записи | func createTweet(text: String) | text | Сообщение для создания записи |

* 1. Описание класса UserPresenter

Данный класс регулирует отображение профиля пользователя социальной сети, а также его записи. Пользователь программного средства может подписаться, отписаться, оценить записи. Если это профиль самого пользователя, то вместо подписки/отписки доступна возможность выйти из аккаунта.

Таблица 4.6 Методы класса UserPresenter

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| loadInitialTweets(action:) | Загрузка первых записей | func loadInitialTweets(action: (() -> Void)?) | action | Замыкание для выполнения действий после загрузки |
| imageForTweet(at:, action: ) | Загрузка картинки для записи | func imageForTweet(at index: Int, action: @escaping ((UIImage) -> Void)) | index | Индекс записи, для которой загружается картинка |
| action | Замыкание для установления изображения |
| retweetTweet(at: ) | Отправка запроса на ретвит записи | func retweetTweet(at index: Int) | index | Индекс записи, ретвит которой нужно сделать |
| likeTweet(at: ) | Отправка запроса оценки записи | func likeTweet(at index: Int) | index | Индекс записи, которую нужно оценить |
| loadNewTweets() | Загрузка новых записей | func loadNewTweets() | - | - |
| loadMoreTweets() | Загрузка более старых записей | func loadMoreTweets() | - | - |
| loadUser() | Загрузка данных о пользователе | func loadUser() | - | - |
| followAction() | Подписка, отписка или выход из аккаунта | func followAction() | - | - |

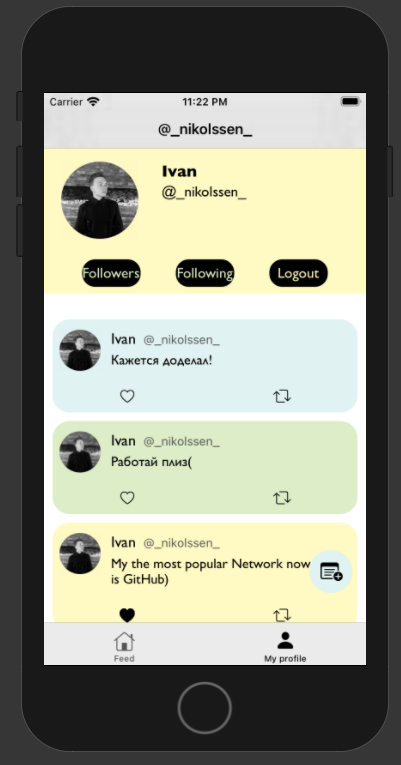


Рисунок 4.5 – контроллер пользователя

* 1. Описание класса UserListPresenter

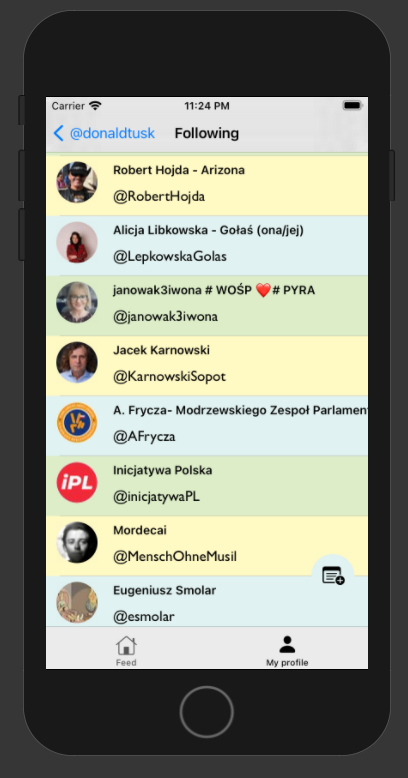


Рисунок 4.6 – список пользователей, подписанных на пользователя @donaldtusk

Класс UserListPresenter координирует таблицу с пользователями. При нажатии ячейку таблицы загружается контроллер выбранного пользователя.

Таблица 4.7 Методы класса UserListPresenter

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| loadUsers() | Загрузка пользователей | func loadUsers() | - | - |
| imageForUser(at:, action: ) | Загрузка картинки для записи | func imageForUser(at index: Int, action: @escaping ((UIImage) -> Void)) | index | Индекс записи, для которой загружается картинка |
| action | Замыкание для установления изображения |

* 1. Описание структур модели данных

Модели данных в программном средстве представлены записями и данными пользователя. Для этого используются структуры Tweet, User.

Таблица 4.8 Свойства структуры Tweet

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение параметра** |
| text | String | Текст записи |
| tweetID | String | Индентификатор записи |
| favorited | Bool | Флаг, отмечена ли запись как понравившаяся |
| retweeted | Bool | Флаг, сделан ли ретвит записи |
| user | User | Данные о пользователе |

Таблица 4.9 Свойства структуры User

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение параметра** |
| screenName | String | Логин пользователя |
| userPhotoLink | String | Ссылка на картинку пользователя |
| id | String | Идентификатор пользователя |

Продолжение таблицы 4.9

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение параметра** |
| name | String | Имя пользователя |
| verified | Bool | Флаг, верифицирован пользователь (например, если он знаменитость) |
| following | Bool | Флаг, подписан ли пользователь программного средства на обновения данного пользователя |

* 1. Описание классов работы с сетью

Для работы с сетью используется сторонний класс Swifter [9]. Для загрузки картинок используется класс ImageDownloader, который сохраняет изображение в кеш. Операция, загружающая картинки, описывается классом ImageDownloadOperation.

Класс TwitterService хранит экземпляры классов Swifter и ImageDownloader в качестве статических свойств, а также идентификатор пользователя.

Таблица 4.10 Методы класса ImageDownloadOperation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| init(url:) | Метод-конструктор | init(url: String) | url | Ссылка для загрузки |
| cancel() | Отмена загрузки | override func cancel() | - | - |
| main() | Основная функция операции | override func main() | - | - |

Таблица 4.11 Методы класса ImageDownloader

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| loadImage(for:, completion:) | Получение изображения по ссылке или из кеша, добавление нового замыкания для операции, либо её удаление из словаря операций | func loadImage(for url: String, completion: @escaping ((UIImage) -> Void)) | completion | Замыкание для обработки результата после загрузки изображения |

В результате этапа конструирования создано программное средство. Схема программы представлена в приложении А. Листинг программы представлен в приложении В.

1. Тестирование, проверка работоспособности и анализ полученных результатов

В данном разделе я остановлюсь на тестировании программного средства. В таблицах 5.1, 5.2 и 5.3 представлены результаты тестирования программного средства.

* 1. Тестирование алгоритма авторизации

Таблица 5.1 Тестирование алгоритма авторизации

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Авторизация в новом приложении | Запуск приложения, нажатие на кнопку, ввод логина и пароля, подтверждение | Загрузка экрана авторизации, демонстрация модального окна браузера, загрузка основого экрана. | Тест пройден |
| 2 | Ввод неверных данных | Запуск приложения, нажатие на кнопку, ввод неверного логина или пароля, подтверждение | Переход в браузере на страницу о неверном пароле, закрытие модального окна выдает уведомление о неудачной авторизации | Тест пройден |
| 3 | Выход из аккаунта в приложении и повторный вход | В авторизированном режиме зайти на свой профиль и нажать кнопку “Logout”, после чего нажать на кнопку и ввести логин и пароль | По нажатии на кнопку происходит переход на экран авторизации, демонстрация модального окна браузера, загрузка основого экрана. | Тест пройден |
| 4 | Повторный вход в авторизирован-ном режиме | В авторизированном режиме закрыть приложение и запустить заново | Запуск основного экрана | Тест пройден |

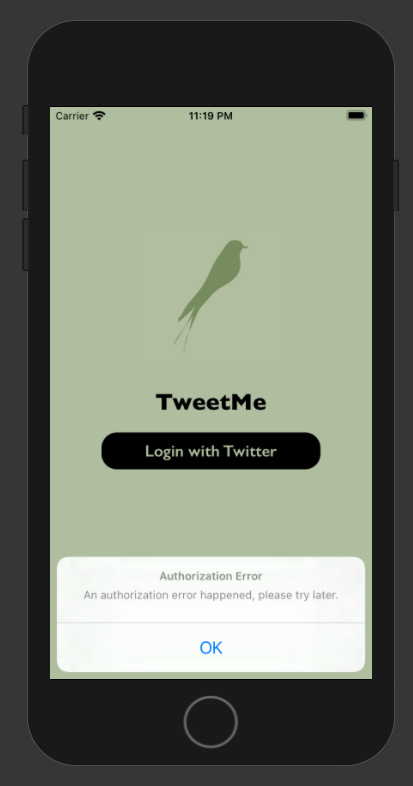


Рисунок 5.1 – уведомление об ошибке авторизации

* 1. Тестирование функциональности новостной ленты

Таблица 5.2 Тестирование функциональности новостной ленты

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Загрузка новостной ленты | Запуск приложения в авторизированном режиме | Десять загруженных записей | Тест пройден |
| 2 | Отметка понравившейся записи | Нажатие на «сердце» возле записи | Изменение «сердца» на залитое, в другом клиенте (например, на сайте <https://twitter.com/home>) отметка также появилась | Тест пройден |
| 3 | Отмена отметки понравившейся записи | Нажатие на залитое «сердце» возле записи | Изменение «сердца» на пустое, в другом клиенте (например, на сайте <https://twitter.com/home>) отметка исчезла | Тест пройден |

Продолжение таблицы 5.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 4 | Создание записи | Нажатие на кнопку «создать запись», ввод текста, отправка. Обновление ленты. | Открывается модальное окно, в него вводится текст, по нажатии на кнопку текст отправляется, в другом клиенте можно увидеть запись. При обновлении ленты запись также появляется | Тест пройден |
| 5 | Ретвит записи | Нажатие на «две стрелки» возле записи | Изменение стиля кнопки на полужирный, в другом клиенте запись появилась | Тест пройден |
| 6 | Отмена ретвита записи | Нажатие на полужирные «две стрелки» возле записи | Изменение стиля кнопки на обычный, в другом клиенте запись исчезла | Тест пройден |
| 7 | Переход на экран пользователя | Нажатие на иконку пользователя | Открывается новый экран пользователя | Тест пройден |
| 8 | Загрузка старых записей | Прокрутка до низа таблицы | Появляются записи | Тест пройден |
| 9 | Обновление таблицы | Жест сверху вниз вверху таблицы | Появление колеса загрузки, исчезновение колеса, появление новых записей (если они есть) | Тест пройден |

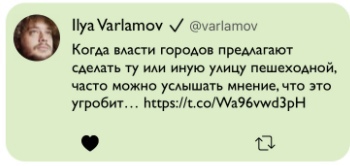


Рисунок 5.2 – отметка понравившейся записи в ячейке записи



Рисунок 5.3 – отметка понравившейся записи в другом клиенте после отметки в программном средстве

* 1. Тестирование взаимодействия с профилем пользователя

Таблица 5.3 Тестирование взаимодействия с профилем пользователя

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последователь-ность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Подписка на пользователя | Нажатие на «подписаться» | Имя кнопки меняется на «отписаться», в другом клиенте, а также на экране подписчики данного пользователя можно заметить аккаунт пользователя программного средства | Тест пройден |
| 2 | Отписка от пользователя | Нажатие на «отписаться» | Имя кнопки меняется на «подписаться», в другом клиенте, а также на экране подписчики данного пользователя можно заметить отсутсвие пользователя программного средства | Тест пройден |
| 3 | Просмотр подписчиков | Нажать на кнопку “подписчики” | Загружается экран с таблицей пользователей | Тест пройден |
| 4 | Просмотр подписок | Нажать на кнопку “подписки” | Загружается экран с таблицей пользователей | Тест пройден |
| 5 | Обновление ленты | Создать новую запись, обновить ленту (свою) | Появляется новая запись | Тест пройден |

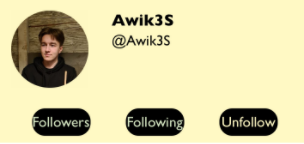


Рисунок 5.4 – верхняя панель пользователя, на которого подписан пользователь программного средства. Кнопка “Unfollow” (отписаться)

Подводя итог, отмечу, что программа отвечает заданным функциональным требованиям, наблюдается стабильность в работе. Вопросов к эстетической части не имеется.

1. Руководство по использованию

Программное средство “TweetMe”, использующий также служебное название “TweetMeCourse”, предназначено для работы с TwitterAPI: с записями пользователей, а также аккаунтами пользователей.

Данное программное средство разработано для использования на мобильных устройствах Apple под управлением операционной системы iOS 13 и новее.

При запуске программы отображается экран авторизации (рис 6.1).

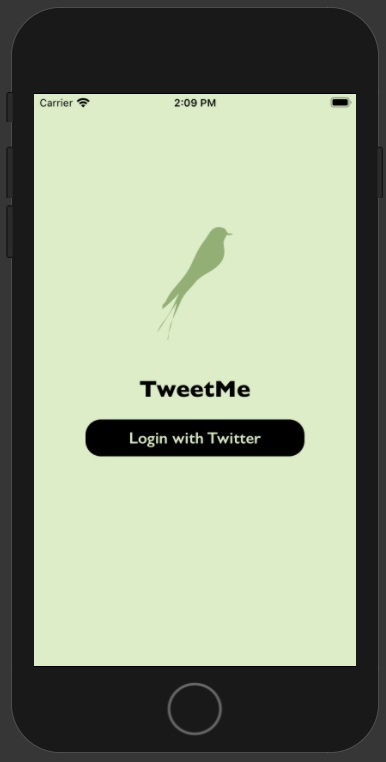


Рисунок 6.1 – экран авторизации

При нажатии на кнопку «Войти при помощи Twiiter» загружается экран браузера, где можно ввести свои логин и пароль. Они не храняться на устройстве, и программное средство напрямую с ними не взаимодействует.

После авторизации загружается главное меню, состоящее из экранов новостной ленты и экрана пользователя. Внизу экрана находится панелья для переключения между ними, а в правом нижнем углу кнопка для открытия экрана создания записи (рис 6.2).



Рисунок 6.2 – кнопка создания записи (справа), панель переключения между экранами (снизу)

Запись в общем случае представляет ячейку (рис. 6.3). Внизу доступны кнопки «отметить понравившуюся запись» и «сделать ретвит».

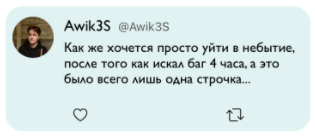


Рисунок 6.3 – запись

При нажатии на иконку пользователся, открывается экран с профилем данного пользователя (рис 6.4). Там доступны его записи, а также список подписок и подписчиков. Там же можно подписаться на пользователя или отписаться. Если вы находитесь в своем профиле, то вы можете выйти из него, вернувшись на экран авторизации. При этом ваши ключи доступа к вашему аккаунту будут удалены.

Если аккаунт пренадлежит известной личности, то рядом с его именем будет стоять «галочка» (рис. 6.4).

Лента поддерживает возможность обновления, она может загружать записи до тех пор, пока не закончаться новости за последние 7 дней (ограничение со стороны Twitter API [1]).

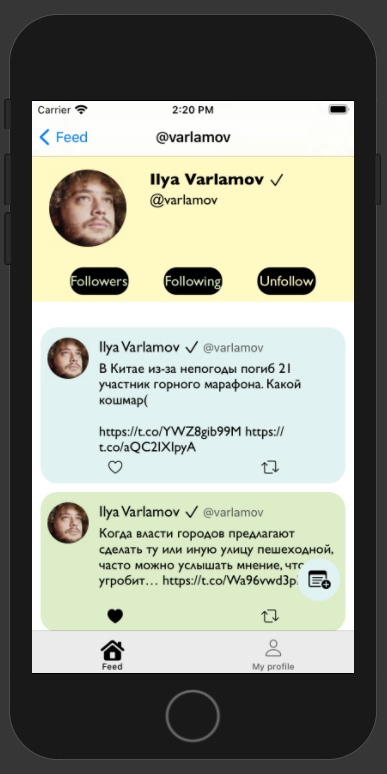


Рисунок 6.4 – профиль пользователя @varlamov

Заключение

В ходе данной работы создано программное средство «Twitter-клиент для платформы iOS», которое предоставляет пользователю возможность взаимодействовать со своим аккаунтом Twitter, создавать, отмечать записи, делать их ретвит, а также подписываться и отписываться на других пользователей.

При этом в ходе работы мною получен опыт работы с парадигмой ООП, для архитектуры программного средства я использовал продвиную архитектуру MVP для клиентского приложения,которая позволила разнести логику приложения и графический интерфейс в два независимых слоя.

Я также изучил такие методы работы с многопоточностью в iOS как OperationQueue и Grand Central Dispatch и использовал их для организации взаимодействия сети и клиентского приложения.

Мною были изучены системные классы URLSession и URLSessionDataTask, класс NSCache.

Также при работе я использовал стороннюю библиотеку Swifter, подключенную с помощью менеджера зависимостей Swift Package Manager.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

Список литературы

[1] Twitter Developer [Электронный ресурс] – Twitter API– Режим доступа: <https://developer.twitter.com/en/products/twitter-api>

[2] Wikipedia [Электронный ресурс] – OAuth – Режим доступа: <https://ru.wikipedia.org/wiki/OAuth>

[3] AppStore [Электронный ресурс] – Twitter – Режим доступа: <https://apps.apple.com/us/app/twitter/id333903271>

[4] AppStore [Электронный ресурс] – UberSocial – Режим доступа: <https://apps.apple.com/us/app/ubersocial/id396050119>

[5] AppStore [Электронный ресурс] – Twitterific – Режим доступа: <https://apps.apple.com/us/app/twitterrific-tweet-your-way/id580311103>

[6] John Codeos [Электронный ресурс] – How to add Twitter Login button to your iOS app using Swift– Режим доступа: [https://johncodeos.com/how-to-add-twitter-login-button-to-your-ios-app-using-swift/](https://johncodeos.com/how-to-add-twitter-login-button-to-your-ios-app-using-swift/%20)

[7] Habr [Электронный ресурс] – Архитектурные паттерны в iOS – Режим доступа: <https://habr.com/ru/company/badoo/blog/281162/>

[8] Apple Developer [Электронный ресурс] – Apple Documentation – Режим доступа: <https://developer.apple.com/documentation/>

[9] GitHub [Электронный ресурс] – Swifter – Режим доступа: <https://github.com/mattdonnelly/Swifter>

Приложение A

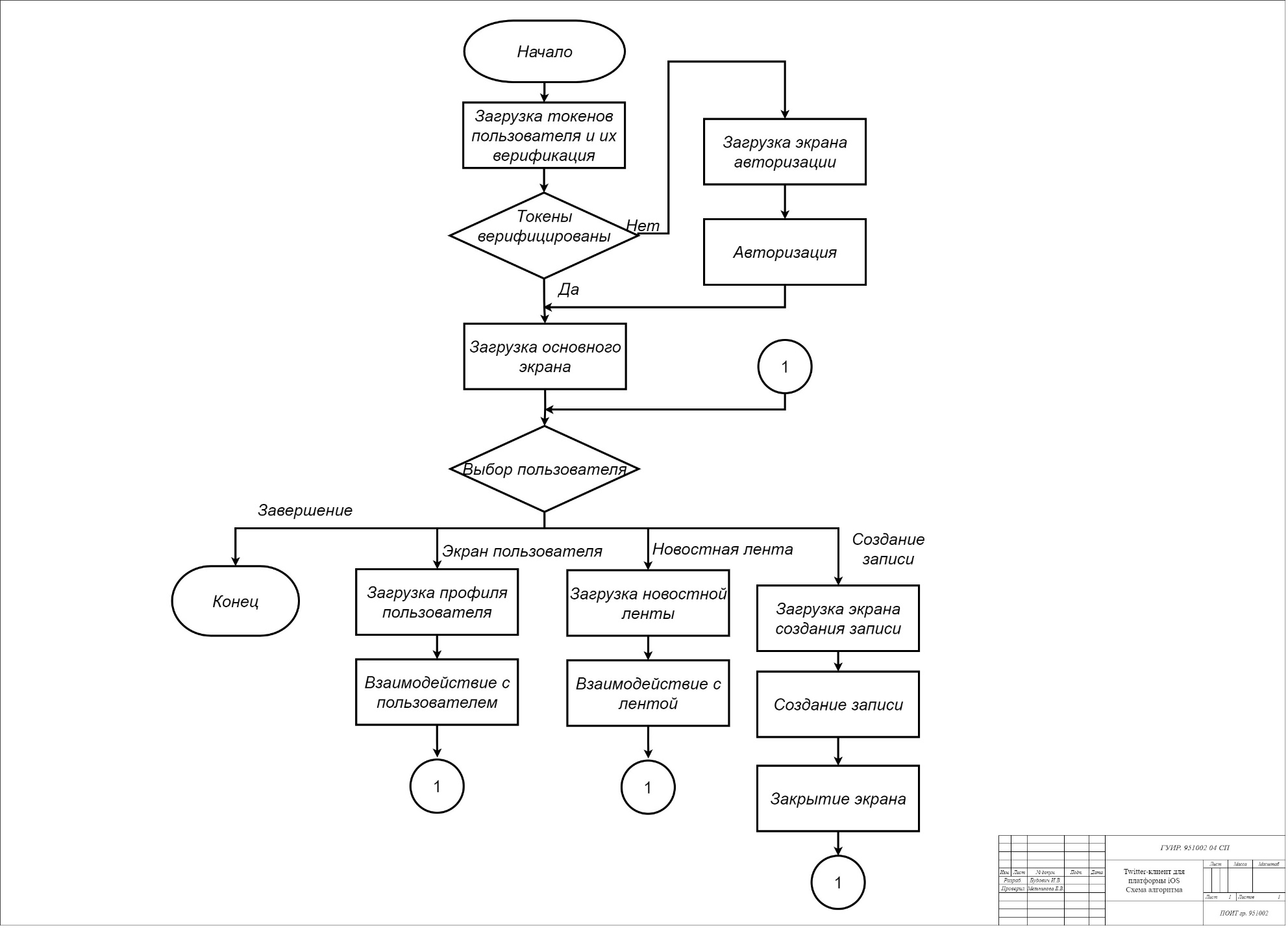


Рисунок А.1 – Схема алгоритма программы

Приложение Б

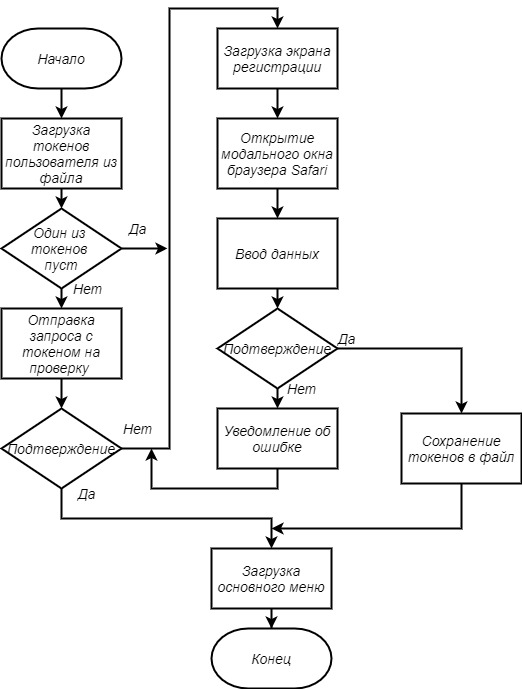


Рисунок Б.1 – Схема алгоритма авторизации

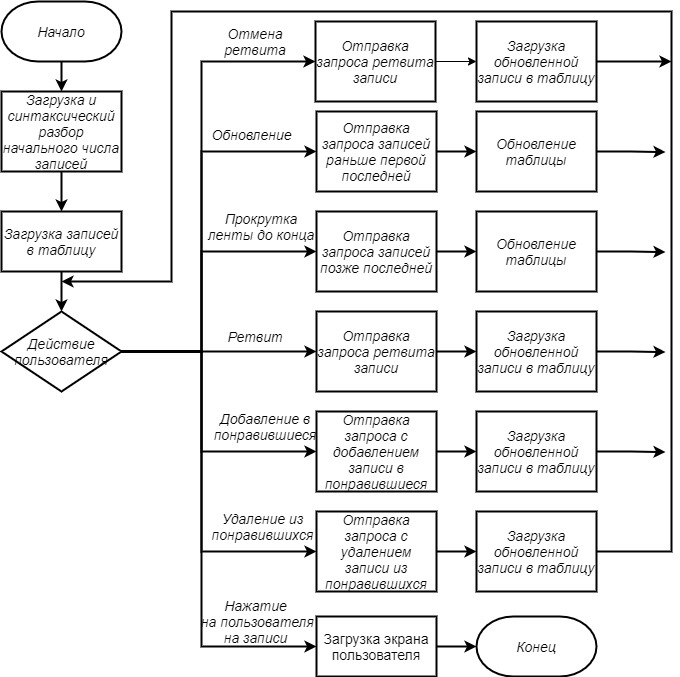


Рисунок Б.2 – Схема взаимодействия с новостной лентой

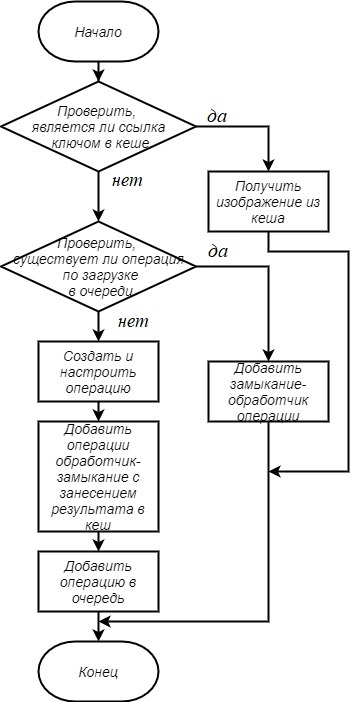


Рисунок Б.3 – Схема алгоритма загрузки изображения

Приложение В

Листинг файла AppDelegate.swift

import UIKit

import Swifter

@main

class AppDelegate: UIResponder, UIApplicationDelegate {

func application(\_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

UITabBar.appearance().unselectedItemTintColor = .darkGray

UITabBar.appearance().tintColor = .black

return true

}

// MARK: UISceneSession Lifecycle

func application(\_ application: UIApplication, configurationForConnecting connectingSceneSession: UISceneSession, options: UIScene.ConnectionOptions) -> UISceneConfiguration {

return UISceneConfiguration(name: "Default Configuration", sessionRole: connectingSceneSession.role)

}

func application(\_ app: UIApplication, open url: URL, options: [UIApplication.OpenURLOptionsKey : Any] = [:]) -> Bool {

let callbackUrl = URL(string: TwitterConstants.callbackURL)!

Swifter.handleOpenURL(url, callbackURL: callbackUrl)

return true

}

}

Листинг файла SceneDelegate.swift

import UIKit

import Swifter

class SceneDelegate: UIResponder, UIWindowSceneDelegate {

var window: UIWindow?

func scene(\_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions: UIScene.ConnectionOptions) {

guard let scene = (scene as? UIWindowScene) else { return }

let theWindow = UIWindow(windowScene: scene)

isLoggedIn{ [weak self] loggedIn in

if loggedIn {

let rootVC = MainTabBarController()

theWindow.rootViewController = rootVC

}

else

{

let rootVC = LoginController(nibName: "LoginController", bundle: nil)

let loginPresenter = LoginPresenter(controller: rootVC)

rootVC.presenter = loginPresenter

TwitterService.swifter = nil

theWindow.rootViewController = rootVC

}

self?.window = theWindow

self?.window?.makeKeyAndVisible()

}

}

func scene(\_ scene: UIScene, openURLContexts URLContexts: Set<UIOpenURLContext>) {

guard let context = URLContexts.first else {return}

let callbackUrl = URL(string: TwitterConstants.callbackURL)!

Swifter.handleOpenURL(context.url, callbackURL: callbackUrl)

}

func isLoggedIn(completion: @escaping(Bool) -> ()) {

let userDefaults = UserDefaults.standard

guard let accessToken = userDefaults.string(forKey: "oauth\_token"), let accessSecretToken = userDefaults.string(forKey: "oauth\_token\_secret") else

{

completion(false)

return

}

let swifter = Swifter(consumerKey: TwitterConstants.consumerKey, consumerSecret: TwitterConstants.consumerSecret, oauthToken: accessToken, oauthTokenSecret: accessSecretToken)

TwitterService.userID = userDefaults.string(forKey: "user\_id")

TwitterService.swifter = swifter

swifter.verifyAccountCredentials(includeEntities: false, skipStatus: true, includeEmail: false, success: {\_ in completion(true)}, failure: {\_ in completion(false)})

}

}

Листинг файла ImageDownloadOperation.swift

import Foundation

import UIKit

class ImageDownloadOperation: Operation {

private var url: String

var dataTask: URLSessionDataTask?

var completion: [((\_ image: UIImage) -> Void)] = []

var image: UIImage?

init(url: String) {

self.url = url

}

override func main() {

if self.isCancelled {

return

}

guard let url = URL(string: self.url) else {return}

self.dataTask = URLSession.shared.dataTask(with: url) {

[weak self] data, response, error in

guard let self = self else

{return}

if data == nil {

return

}

if self.isCancelled

{

return

}

if let data = data, let image = UIImage(data: data) {

self.image = image

for block in self.completion{

block(image)

}

}

}

if self.isCancelled {

return

}

self.dataTask?.resume()

}

override func cancel() {

super.cancel()

self.dataTask?.cancel()

}

}

Листинг файла ImageDownloader.swift

import Foundation

import UIKit

class ImageDownloader {

private let queue = OperationQueue()

private var operations = Dictionary<String,ImageDownloadOperation>()

private var imageCache = NSCache<NSString, UIImage>()

func loadImage(for url: String, completion: @escaping ((UIImage) -> Void)) {

let nsstring = NSString(string: url)

if let image = imageCache.object(forKey: nsstring) {

completion(image)

return

}

if let operation = operations[url] {

operation.completion.append({image in completion(image)})

return

}

let operation = ImageDownloadOperation(url: url)

operations[url] = operation

operation.completion.append({

[weak self]

image in completion(image)

self?.imageCache.setObject(image, forKey: nsstring)

self?.operations.removeValue(forKey: url)

})

self.queue.addOperation(operation)

}

}

Листинг файла TwitterService.swift

import Foundation

import Swifter

struct TwitterConstants {

static let consumerKey = "xOCrn1Ptf4lTxyAPHnMgyuzdr"

static let consumerSecret = "YGOlwRNaHXm7dq1jBQ7YkgqJ5KAjWvsu2eXE3IYFNTIAjwgwWq"

static let callbackURL = "tweetmecourse://"

}

class TwitterService {

static var swifter: Swifter?

static var userID: String?

static var imageDownloader: ImageDownloader? = ImageDownloader()

}

Листинг файла Tweet.swift

import Foundation

import Swifter

struct Tweet {

let text: String

let tweetID: String

var favorited: Bool

var retweeted: Bool

let user: User

init?(json: JSON) {

guard let text = json["text"].string,

let user = User(json: json["user"]),

let tweetID = json["id\_str"].string,

let favorited = json["favorited"].bool,

let retweeted = json["retweeted"].bool

else {return nil}

self.text = text

self.user = user

self.tweetID = tweetID

self.favorited = favorited

self.retweeted = retweeted

}

static func array(of jsonTweets: [JSON]) -> [Tweet] {

var tweets = [Tweet]()

for jsonTweet in jsonTweets {

if let tweet = Tweet(json:jsonTweet) {

tweets.append(tweet)

}

}

return tweets

}

Листинг файла User.swift

import Foundation

import Swifter

struct User {

let screenName: String

let userPhotoLink: String

let id: String

let name: String

let verified: Bool

let following: Bool

init?(json: JSON){

guard let screenName = json["screen\_name"].string,

let userPhotoLink = json["profile\_image\_url"].string,

let id = json["id\_str"].string,

let name = json["name"].string,

let verified = json["verified"].bool,

let following = json["following"].bool

else {return nil}

self.screenName = screenName

self.userPhotoLink = userPhotoLink

self.id = id

self.name = name

self.verified = verified

self.following = following

}

static func array(of jsonUsers: [JSON]) -> [User] {

var users = [User]()

for jsonUser in jsonUsers {

if let user = User(json:jsonUser) {

users.append(user)

}

}

return users

}

}

Листинг файла LoginPresenter.swift

import Foundation

import Swifter

class LoginPresenter{

weak private var controller: LoginViewProtocol?

init(controller: LoginViewProtocol) {

self.controller = controller

}

func login()

{

guard let controller = controller else {return}

let swifter = Swifter(consumerKey: TwitterConstants.consumerKey, consumerSecret: TwitterConstants.consumerSecret)

swifter.authorize(withProvider: controller, callbackURL: URL(string: TwitterConstants.callbackURL)!, success:{

accessToken, \_ in

guard let accessToken = accessToken else {return}

self.save(credentials: accessToken)

TwitterService.swifter = Swifter(consumerKey: TwitterConstants.consumerKey, consumerSecret: TwitterConstants.consumerSecret, oauthToken: accessToken.key, oauthTokenSecret: accessToken.secret)

TwitterService.userID = accessToken.userID

self.controller?.loadMainView()

}, failure: {

\_ in

self.controller?.failure()

})

}

func save(credentials: Credential.OAuthAccessToken)

{

let userDefaults = UserDefaults.standard

userDefaults.set(credentials.key, forKey: "oauth\_token")

userDefaults.set(credentials.secret, forKey: "oauth\_token\_secret")

userDefaults.set(credentials.userID, forKey: "user\_id")

}

}

Листинг файла LoginViewProtocol.swift

import Foundation

import AuthenticationServices

protocol LoginViewProtocol: NSObject, ASWebAuthenticationPresentationContextProviding{

func loadMainView()

func failure()

}

Листинг файла LoginController.swift

import UIKit

import Swifter

import SafariServices

import AuthenticationServices

class LoginController: UIViewController {

@IBOutlet weak var loginButton: UIButton!

var presenter: LoginPresenter!

override func viewDidLoad() {

super.viewDidLoad()

loginButton.layer.cornerRadius = 18

}

@IBAction private func loginAction(\_ sender: UIButton)

{

presenter.login()

}

}

extension LoginController: LoginViewProtocol{

func loadMainView() {

self.view.window?.rootViewController = MainTabBarController()

}

func failure() {

let allertVC = UIAlertController(title: "Authorization Error", message: "An authorization error happened, please try later.", preferredStyle: .actionSheet)

allertVC.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))

self.present(allertVC, animated: true, completion: nil)

}

}

extension LoginController: SFSafariViewControllerDelegate{

func safariViewControllerDidFinish(\_ controller: SFSafariViewController) {

controller.dismiss(animated: true, completion: nil)

}

}

extension LoginController: ASWebAuthenticationPresentationContextProviding{

func presentationAnchor(for session: ASWebAuthenticationSession) -> ASPresentationAnchor {

return view.window!

}

}

Листинг файла FeedPresenter.swift

import Foundation

import Swifter

import UIKit

class FeedPresenter{

weak private var controller: FeedViewProtocol?

private var tweets = [Tweet]()

private var requestSend = false

init(controller: FeedViewProtocol) {

self.controller = controller

}

func loadNewTweets(){

guard let swifter = TwitterService.swifter, !requestSend else {return}

if let first = tweets.first

{

swifter.getHomeTimeline(count: nil, sinceID: first.tweetID, success: {json in

let newTweets = Tweet.array(of: json.array!)

self.tweets.insert(contentsOf: newTweets, at: 0)

self.controller?.stopUpdating()

self.controller?.update()

self.requestSend = false

}, failure: {\_ in

self.controller?.stopUpdating()

self.requestSend = false

})

}

else {

loadInitialTweets(action: {

self.controller?.stopUpdating()

})

}

}

func loadMoreTweets(){

guard let swifter = TwitterService.swifter, tweets.count > 0, !requestSend else

{return}

requestSend = true

swifter.getHomeTimeline(count: 10, maxID: tweets.last?.tweetID, success: {json in

if var array = json.array {

array.remove(at: 0)

self.tweets.append(contentsOf: Tweet.array(of: array))

self.controller?.update()

self.requestSend = false

}

}, failure: {\_ in self.requestSend = false})

}

func loadInitialTweets(action: (() -> Void)?){

guard let swifter = TwitterService.swifter else {return}

requestSend = true

swifter.getHomeTimeline(count: 10, success: {json in

guard let array = json.array else {return}

self.tweets = Tweet.array(of: array)

guard let controller = self.controller else {return}

if let action = action {

action()

}

controller.update()

self.requestSend = false

}, failure: { \_ in

self.requestSend = false

if let action = action {

action()

}

}

)

}

func numberOfTweets() -> Int{

return tweets.count

}

func textForTweet(at index: Int) -> String{

return tweets[index].text

}

func nameForTweet(at index: Int) -> String {

return tweets[index].user.name

}

func screenNameForTweet(at index: Int) -> String {

return tweets[index].user.screenName

}

func isVerifiedUser(at index: Int) -> Bool {

return tweets[index].user.verified

}

func isLikedTweet(at index: Int) -> Bool {

return tweets[index].favorited

}

func isRetweetedTweet(at index: Int) -> Bool {

return tweets[index].retweeted

}

func imageForTweet(at index: Int, action: @escaping ((UIImage) -> Void)){

TwitterService.imageDownloader?.loadImage(for: tweets[index].user.userPhotoLink, completion: action)

}

func idForUser(at index: Int) -> String{

return tweets[index].user.id

}

func likeTweet(at index: Int){

guard let swifter = TwitterService.swifter, !requestSend else { return }

requestSend = true

if tweets[index].favorited {

swifter.unfavoriteTweet(forID: tweets[index].tweetID, includeEntities: false, tweetMode: .compat, success: { \_ in

self.tweets[index].favorited = false

self.controller?.update()

self.requestSend = false

}, failure: { \_ in

self.requestSend = false

})

}

else {

swifter.favoriteTweet(forID: tweets[index].tweetID, includeEntities: false, tweetMode: .compat, success: { \_ in

self.tweets[index].favorited = true

self.controller?.update()

self.requestSend = false

}, failure: {

\_ in

self.requestSend = false

})

}

}

func retweetTweet(at index: Int){

guard let swifter = TwitterService.swifter, !requestSend else { return }

requestSend = true

if tweets[index].retweeted {

swifter.unretweetTweet(forID: tweets[index].tweetID, trimUser: true, tweetMode: .compat, success:

{ \_ in

self.tweets[index].retweeted = false

self.controller?.update()

self.requestSend = false

}, failure: { \_ in

self.requestSend = false

})

}

else {

swifter.retweetTweet(forID: tweets[index].tweetID, trimUser: true, tweetMode: .compat, success: { \_ in

self.tweets[index].retweeted = true

self.controller?.update()

self.requestSend = false

}, failure: {

\_ in

self.requestSend = false

})

}

}

}

Листинг файла FeedViewProtocol.swift

import Foundation

protocol FeedViewProtocol: NSObject{

func update()

func stopUpdating()

}

Листинг файла FeedController.swift

import UIKit

private let reuseIdentifier = "TweetCellID"

class FeedController: UICollectionViewController, FeedViewProtocol {

private var tweets = [Tweet]()

private var requestSend: Bool = false

var presenter: FeedPresenter!

override func viewDidLoad() {

super.viewDidLoad()

self.collectionView!.register(TweetCell.self, forCellWithReuseIdentifier: reuseIdentifier)

self.clearsSelectionOnViewWillAppear = false

view.backgroundColor = UIColor(named: "BackgroundColor")

collectionView?.backgroundColor = UIColor(named: "BackgroundColor")

let refresher = UIRefreshControl()

self.collectionView.alwaysBounceVertical = true

refresher.addTarget(self, action: #selector(refresh), for: .valueChanged)

self.collectionView.refreshControl = refresher

presenter.loadInitialTweets(action: nil)

}

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(animated)

navigationController?.navigationBar.largeTitleTextAttributes = [.foregroundColor: UIColor.black, .font: UIFont(name: "Gill Sans Bold", size: 30) as Any]

navigationItem.title = "Feed"

navigationController?.navigationBar.prefersLargeTitles = true

navigationItem.largeTitleDisplayMode = .always

}

func update() {

self.collectionView.reloadData()

}

func stopUpdating() {

collectionView.refreshControl?.endRefreshing()

}

@objc private func refresh() {

collectionView.refreshControl?.beginRefreshing()

presenter.loadNewTweets()

}

}

// MARK: UICollectionViewDataSource

extension FeedController

{

override func numberOfSections(in collectionView: UICollectionView) -> Int {

return 1

}

override func collectionView(\_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {

return presenter.numberOfTweets()

}

override func collectionView(\_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {

let cell = collectionView.dequeueReusableCell(withReuseIdentifier: reuseIdentifier, for: indexPath) as! TweetCell

let index = indexPath.item

var backgroundColor: UIColor

switch index % 3 {

case 0:

backgroundColor = UIColor(named: "SpecialBlue")!

case 1:

backgroundColor = UIColor(named: "SpecialGreen")!

default:

backgroundColor = UIColor(named: "SpecialYellow")!

}

cell.backgroundColor = backgroundColor

cell.setInfoLabel(name: presenter.nameForTweet(at: index), screenName: presenter.screenNameForTweet(at: index), verified: presenter.isVerifiedUser(at: index))

cell.setTweetText(text: presenter.textForTweet(at: index))

cell.setLike(presenter.isLikedTweet(at: index))

cell.setRetweeted(presenter.isRetweetedTweet(at: index))

cell.index = index

cell.delegate = self

presenter.imageForTweet(at: index, action: {

image in

DispatchQueue.main.async {

cell.profileImageView.image = image

}

})

return cell

}

override func scrollViewDidScroll(\_ scrollView: UIScrollView) {

if (scrollView.contentOffset.y + scrollView.frame.size.height) > scrollView.contentSize.height {

presenter.loadMoreTweets()

}

}

}

extension FeedController: UICollectionViewDelegateFlowLayout {

func collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {

let width = UIScreen.main.bounds.width - 20

let measureLabel = UILabel()

measureLabel.text = presenter.textForTweet(at: indexPath.item)

measureLabel.numberOfLines = 0

measureLabel.font = UIFont(name: "Gill Sans", size: 16)

measureLabel.lineBreakMode = .byCharWrapping

measureLabel.translatesAutoresizingMaskIntoConstraints = false

measureLabel.widthAnchor.constraint(equalToConstant: (width - 80)).isActive = true

let height = measureLabel.systemLayoutSizeFitting(UIView.layoutFittingCompressedSize).height + 90

return CGSize(width: width, height: height)

}

func collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, insetForSectionAt section: Int) -> UIEdgeInsets {

return UIEdgeInsets(top: 30, left: 10, bottom: 30, right: 10)

}

}

extension FeedController: TweetCellDelegate{

func userImageTapped(index: Int?) {

guard let index = index else {

return

}

let userID = presenter.idForUser(at: index)

let userVC = UserController(nibName: "UserController", bundle: nil)

let userPresenter = UserPresenter(controller: userVC, userID: userID)

userVC.presenter = userPresenter

navigationController?.pushViewController(userVC, animated: true)

}

func retweetButtonTapped(index: Int?) {

guard let index = index else {

return

}

presenter.retweetTweet(at: index)

}

func likeButtonTapped(index: Int?) {

guard let index = index else {

return

}

presenter.likeTweet(at: index)

}

}}

Листинг файла MainTabBarController.swift

import UIKit

import Swifter

class MainTabBarController: UITabBarController {

private let actionButton: UIButton = {

let button = UIButton(type: .system)

button.tintColor = .black

button.backgroundColor = UIColor(named: "SpecialBlue")

button.setImage(UIImage(systemName: "note.text.badge.plus"), for: .normal)

button.layer.cornerRadius = 25

button.addTarget(self, action: #selector(actionButtonTapped), for: .touchUpInside)

return button

}()

override func viewDidLoad() {

super.viewDidLoad()

tabBar.barTintColor = .white

let unselectedImageConfiguration =

UIImage.SymbolConfiguration(weight: .light)

let selectedImageConfiguration = UIImage.SymbolConfiguration(weight: .heavy)

let feedVC = FeedController(collectionViewLayout: UICollectionViewFlowLayout())

let feedPresenter = FeedPresenter(controller: feedVC)

feedVC.presenter = feedPresenter

let feedNav = UINavigationController(rootViewController: feedVC)

feedNav.tabBarItem.title = "Feed"

feedNav.tabBarItem.image = UIImage(systemName: "house", withConfiguration: unselectedImageConfiguration)

feedNav.tabBarItem.selectedImage = UIImage(systemName: "house.fill", withConfiguration: selectedImageConfiguration)

let userVC = UserController(nibName: "UserController", bundle: nil)

let userPresenter = UserPresenter(controller: userVC, userID: TwitterService.userID!)

userVC.presenter = userPresenter

let userNav = UINavigationController(rootViewController:userVC)

userNav.tabBarItem.title = "My profile"

userNav.tabBarItem.image = UIImage(systemName: "person", withConfiguration: unselectedImageConfiguration)

userNav.tabBarItem.selectedImage = UIImage(systemName: "person.fill", withConfiguration: selectedImageConfiguration)

configureActionButton()

viewControllers = [feedNav, userNav]

}

@objc func actionButtonTapped(){

let tweetVC = TweetCreationController(nibName: "TweetCreationController", bundle: nil)

let tweetPresenter = TweetCreationPresenter(controller: tweetVC)

tweetVC.presenter = tweetPresenter

present(tweetVC, animated: true, completion: nil)

}

func configureActionButton(){

view.addSubview(actionButton)

actionButton.translatesAutoresizingMaskIntoConstraints = false

NSLayoutConstraint.activate([actionButton.heightAnchor.constraint(equalToConstant: 50), actionButton.widthAnchor.constraint(equalToConstant: 50), actionButton.bottomAnchor.constraint(equalTo: view.safeAreaLayoutGuide.bottomAnchor, constant: -84), actionButton.rightAnchor.constraint(equalTo: view.rightAnchor, constant: -16)])

}

}

Листинг файла UserListPresenter.swift

import Foundation

import Swifter

import UIKit

class UserListPresenter{

weak private var controller: UserListController?

private var users = [User]()

private let option: UserListOption

private let userTag: UserTag

init(controller: UserListController, userID: String, option: UserListOption){

self.controller = controller

self.userTag = .id(userID)

self.option = option

}

func numberOfUsers() -> Int{

return users.count

}

func nameForUser(at index: Int) -> String {

return users[index].name

}

func screenNameForUser(at index: Int) -> String {

return users[index].screenName

}

func isVerifiedUser(at index: Int) -> Bool {

return users[index].verified

}

func imageForTweet(at index: Int, action: @escaping ((UIImage) -> Void)){

TwitterService.imageDownloader?.loadImage(for: users[index].userPhotoLink, completion: action)

}

func idForUser(at index: Int) -> String{

return users[index].id

}

func title() ->String {

switch option {

case .followers:

return "Followers"

case .following:

return "Following"

}

}

func loadUsers() {

guard let swifter = TwitterService.swifter else {

return

}

switch option {

case .followers:

swifter.getUserFollowers(for: userTag, count: 50, skipStatus: true, includeUserEntities: false, success: {

json, response, \_ in

self.users.append(contentsOf: User.array(of: json.array!))

self.controller?.update()

})

case .following:

swifter.getUserFollowing(for: userTag, count: 50, skipStatus: true, includeUserEntities: false, success: {

json, \_, \_ in

self.users.append(contentsOf: User.array(of: json.array!))

self.controller?.update()

})

}

}

enum UserListOption {

case followers

case following

}

}

Листинг файла UserListViewProtocol.swift

import Foundation

protocol UserListViewProtocol: NSObject{

func update()

}

Листинг файла UserListController.swift

class UserListController: UITableViewController, UserListViewProtocol {

var presenter: UserListPresenter!

override func viewDidLoad() {

super.viewDidLoad()

navigationItem.title = presenter.title()

self.clearsSelectionOnViewWillAppear = true

tableView.register(UINib(nibName: "UserCell", bundle: nil), forCellReuseIdentifier: "UserCellID")

presenter.loadUsers()

}

func update() {

tableView.reloadData()

}

// MARK: - Table view data source

override func numberOfSections(in tableView: UITableView) -> Int {

return 1

}

override func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

return presenter.numberOfUsers()

}

override func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

let index = indexPath.row

let cell = tableView.dequeueReusableCell(withIdentifier: "UserCellID", for: indexPath) as! UserCell

var backgroundColor: UIColor

switch index % 3 {

case 0:

backgroundColor = UIColor(named: "SpecialBlue")!

case 1:

backgroundColor = UIColor(named: "SpecialGreen")!

default:

backgroundColor = UIColor(named: "SpecialYellow")!

}

cell.contentView.backgroundColor = backgroundColor

cell.set(name: presenter.nameForUser(at: index), screenName: presenter.screenNameForUser(at: index), verified: presenter.isVerifiedUser(at: index))

presenter.imageForTweet(at: index, action: {

image in

DispatchQueue.main.async {

cell.userProfileImageView.image = image

}

})

return cell

}

override func tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {

let index = indexPath.row

let userVC = UserController(nibName: "UserController", bundle: nil)

let userPresenter = UserPresenter(controller: userVC, userID: presenter.idForUser(at: index))

userVC.presenter = userPresenter

navigationController?.pushViewController(userVC, animated: true)

}

override func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {

return 70

}

}

Листинг файла TwitterCreationPresenter.swift

import Foundation

import Swifter

class TweetCreationPresenter{

private weak var controller: TweetCreationViewProtocol?

init(controller: TweetCreationViewProtocol) {

self.controller = controller

}

func createTweet(text: String) {

guard let swifter = TwitterService.swifter else {return}

swifter.postTweet(status: text, success: {

\_ in

self.controller?.dismiss()

})

}

}

Листинг файла TwitterCreationViewProtocol.swift

import Foundation

protocol TweetCreationViewProtocol: NSObject {

func dismiss()

}

Листинг файла TwitterCreationController.swift

import UIKit

class TweetCreationController: UIViewController{

@IBOutlet private weak var textViewHeight: NSLayoutConstraint!

@IBOutlet private weak var tweetTextView: UITextView!

@IBOutlet private weak var postButton: UIButton!

var presenter: TweetCreationPresenter!

override func viewDidLoad() {

super.viewDidLoad()

postButton.layer.cornerRadius = 18

tweetTextView.layer.cornerRadius = 5

textViewHeight.constant = UIScreen.main.bounds.height \* 0.55

}

@IBAction private func postAction(\_ sender: UIButton) {

guard let text = tweetTextView.text else {return}

presenter.createTweet(text: text)

dismiss(animated: true, completion: nil)

}

override func viewDidAppear(\_ animated: Bool) {

super.viewDidAppear(animated)

NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillShow(notification:)), name: UIWindow.keyboardWillShowNotification, object: nil)

NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillHide(notification:)), name: UIWindow.keyboardWillHideNotification, object: nil)

}

@objc private func keyboardWillShow(notification: Notification) {

textViewHeight.constant = UIScreen.main.bounds.height \* 0.3

view.setNeedsLayout()

}

@objc private func keyboardWillHide(notification: Notification) {

textViewHeight.constant = UIScreen.main.bounds.height \* 0.55

view.setNeedsLayout()

}

override func viewWillDisappear(\_ animated: Bool) {

NotificationCenter.default.removeObserver(self)

}

}

extension TweetCreationController: TweetCreationViewProtocol{

func dismiss() {

dismiss(animated: true, completion: nil)

}

}

Листинг файла UserPresenter.swift

import Foundation

import Swifter

import UIKit

class UserPresenter{

weak private var controller: UserController?

private var tweets = [Tweet]()

private var requestSend = false

private var userTag: UserTag

private var user: User!

init(controller: UserController, userID: String) {

self.controller = controller

self.userTag = .id(userID)

}

func loadNewTweets(){

guard let swifter = TwitterService.swifter else {return}

if let first = tweets.first, !requestSend

{

swifter.getTimeline(for: userTag, count: nil, sinceID: first.tweetID, success: {json in

let newTweets = Tweet.array(of: json.array!)

self.tweets.insert(contentsOf: newTweets, at: 0)

self.controller?.stopUpdating()

self.controller?.update()

self.requestSend = false

}, failure: {\_ in

self.controller?.stopUpdating()

self.requestSend = false

})

}

else {

loadInitialTweets(action: {

self.controller?.stopUpdating()

})

}

}

func loadUser(){

guard let swifter = TwitterService.swifter else {return}

requestSend = true

swifter.showUser(userTag, success: {

json in

self.user = User(json: json)

self.controller?.setUserName(self.user.name, screenName: self.user.screenName, verified: self.user.verified)

let largePhotoLink = self.user.userPhotoLink.replacingOccurrences(of: "\_normal", with: "")

TwitterService.imageDownloader?.loadImage(for: largePhotoLink, completion:

{image in

DispatchQueue.main.async {

self.controller?.setUserImage(image)

}

})

if self.user.id == TwitterService.userID {

self.controller?.setFollowText(text: "Logout")

}

else

{

if self.user.following {

self.controller?.setFollowText(text: "Unfollow")

}

else

{

self.controller?.setFollowText(text: "Follow")

}

}

self.requestSend = false

self.loadInitialTweets(action: nil)

})

}

func loadMoreTweets(){

guard let swifter = TwitterService.swifter, tweets.count > 0, !requestSend else

{return}

requestSend = true

swifter.getTimeline(for: userTag, count: 10, sinceID: tweets.last?.tweetID, success: {json in

if var array = json.array {

array.remove(at: 0)

self.tweets.append(contentsOf: Tweet.array(of: array))

self.controller?.update()

self.requestSend = false

}

}, failure: {\_ in self.requestSend = false})

}

func numberOfTweets() -> Int{

return tweets.count

}

func textForTweet(at index: Int) -> String{

return tweets[index].text

}

func nameForTweet(at index: Int) -> String {

return user.name

}

func screenNameForTweet(at index: Int) -> String {

return user.screenName

}

func isVerifiedUser(at index: Int) -> Bool {

return user.verified

}

func isLikedTweet(at index: Int) -> Bool {

return tweets[index].favorited

}

func isRetweetedTweet(at index: Int) -> Bool {

return tweets[index].retweeted

}

func imageForTweet(at index: Int, action: @escaping ((UIImage) -> Void)){

TwitterService.imageDownloader?.loadImage(for: user.userPhotoLink, completion: action)

}

func idForUser(at index: Int) -> String{

return tweets[index].user.id

}

func loadInitialTweets(action: (() -> Void)?){

guard let swifter = TwitterService.swifter else {return}

requestSend = true

swifter.getTimeline(for: userTag, count: 10, excludeReplies: true, includeRetweets: true, success: {json in

guard let array = json.array else {return}

self.tweets = Tweet.array(of: array)

guard let controller = self.controller else {return}

if let action = action {

action()

}

controller.update()

self.requestSend = false

}, failure: { \_ in

if let action = action {

action()

}

self.requestSend = false

})

}

func likeTweet(at index: Int){

guard let swifter = TwitterService.swifter, !requestSend else { return }

requestSend = true

if tweets[index].favorited {

swifter.unfavoriteTweet(forID: tweets[index].tweetID, includeEntities: false, tweetMode: .compat, success: { \_ in

self.tweets[index].favorited = false

self.controller?.update()

self.requestSend = false

}, failure: { \_ in

self.requestSend = false

})

}

else {

swifter.favoriteTweet(forID: tweets[index].tweetID, includeEntities: false, tweetMode: .compat, success: { \_ in

self.tweets[index].favorited = true

self.controller?.update()

self.requestSend = false

}, failure: {

\_ in

self.requestSend = false

})

}

}

func retweetTweet(at index: Int){

guard let swifter = TwitterService.swifter, !requestSend else { return }

requestSend = true

if tweets[index].retweeted {

swifter.unretweetTweet(forID: tweets[index].tweetID, trimUser: true, tweetMode: .compat, success:

{ \_ in

self.tweets[index].retweeted = false

self.controller?.update()

self.requestSend = false

}, failure: { \_ in

self.requestSend = false

})

}

else {

swifter.retweetTweet(forID: tweets[index].tweetID, trimUser: true, tweetMode: .compat, success: { \_ in

self.tweets[index].retweeted = true

self.controller?.update()

self.requestSend = false

}, failure: {

\_ in

self.requestSend = false

})

}

}

func followAction() {

guard let swifter = TwitterService.swifter, let user = user else {return}

if user.id == TwitterService.userID {

let userDefaults = UserDefaults.standard

userDefaults.removeObject(forKey: "oauth\_token")

userDefaults.removeObject(forKey: "oauth\_token\_secret")

userDefaults.removeObject(forKey: "user\_id")

TwitterService.imageDownloader = ImageDownloader()

return

}

if user.following {

swifter.unfollowUser(userTag, success: {

json in

self.controller?.setFollowText(text: "Follow")

self.user.following = false

}, failure: nil)

return

}

if !user.following {

swifter.followUser(userTag, success: {

json in

self.controller?.setFollowText(text: "Unfollow")

self.user.following = true

})

}

}

func getUserID() -> String {

return user.id

}

}

Листинг файла UserViewProtocol.swift

import Foundation

import UIKit

protocol UserViewProtocol: NSObject{

func refresh()

func stopUpdating()

func backToLogin()

func setUserImage(\_ image: UIImage)

func setUserName(\_ name: String, screenName: String, verified: Bool)

func setFollowText(text: String)

}

Листинг файла UserController.swift

import UIKit

class UserController: UIViewController, UserViewProtocol {

var presenter: UserPresenter!

@IBOutlet weak var userProfileImageView: UIImageView!

@IBOutlet weak var userNameLabel: UILabel!

@IBOutlet weak var screenNameLabel: UILabel!

@IBOutlet weak var followersButton: UIButton!

@IBOutlet weak var followsButton: UIButton!

@IBOutlet weak var followButton: UIButton!

@IBOutlet weak var collectionView: UICollectionView!

override func viewDidLoad() {

super.viewDidLoad()

collectionView.delegate = self

collectionView.dataSource = self

configureUI()

collectionView?.register(TweetCell.self, forCellWithReuseIdentifier: "TweetCellID")

let refresher = UIRefreshControl()

self.collectionView.alwaysBounceVertical = true

refresher.addTarget(self, action: #selector(refresh), for: .valueChanged)

self.collectionView.refreshControl = refresher

presenter.loadUser()

//presenter.loadInitialTweets(action: nil)

}

override func viewWillAppear(\_ animated: Bool)

{

super.viewWillAppear(animated)

navigationController?.navigationBar.prefersLargeTitles = false

navigationItem.largeTitleDisplayMode = .never

}

private func configureUI(){

collectionView?.backgroundColor = UIColor(named: "BackgroundColor")

followsButton.layer.cornerRadius = 30/2

followersButton.layer.cornerRadius = 30/2

followButton.layer.cornerRadius = 30/2

userProfileImageView.contentMode = .scaleAspectFit

userProfileImageView.clipsToBounds = true

userProfileImageView.layer.cornerRadius = 90/2

userProfileImageView.backgroundColor = .white

}

func update() {

self.collectionView.reloadData()

}

func stopUpdating() {

collectionView.refreshControl?.endRefreshing()

}

func setUserImage(\_ image: UIImage) {

userProfileImageView.image = image

}

func setUserName(\_ name: String, screenName: String, verified: Bool) {

set(name: name, verified: verified)

screenNameLabel.text = "@\(screenName)"

navigationItem.title = "@\(screenName)"

}

func setFollowText(text: String) {

followButton.setTitle(text, for: .normal)

}

func backToLogin() {

let loginController = LoginController(nibName: "LoginController", bundle: nil)

let loginPresenter = LoginPresenter(controller: loginController)

loginController.presenter = loginPresenter

view.window?.rootViewController = loginController

}

func set(name: String, verified: Bool) {

if verified {

let fullString = NSMutableAttributedString(string:"\(name) ", attributes: [.font : UIFont(name: "Gill Sans Bold", size: 19)!, .foregroundColor: UIColor.black])

let imageAttachment = NSTextAttachment()

imageAttachment.image = UIImage(systemName: "checkmark")

fullString.append(NSAttributedString(attachment: imageAttachment))

userNameLabel.attributedText = fullString

}

else {

userNameLabel.text = "\(name)"

}

}

@IBAction func showFollowersAction(\_ sender: Any) {

let userListVC = UserListController(style: .plain)

let id = presenter.getUserID()

let presenter = UserListPresenter(controller: userListVC, userID: id, option: .followers)

userListVC.presenter = presenter

navigationController?.pushViewController(userListVC, animated: true)

}

@IBAction func showFollowingAction(\_ sender: Any) {

let userListVC = UserListController(style: .plain)

let id = presenter.getUserID()

let presenter = UserListPresenter(controller: userListVC, userID: id, option: .following)

userListVC.presenter = presenter

navigationController?.pushViewController(userListVC, animated: true)

}

@IBAction func followAction(\_ sender: Any) {

presenter.followAction()

}

@objc func refresh() {

self.collectionView.refreshControl?.beginRefreshing()

presenter.loadNewTweets()

}

}

extension UserController: UICollectionViewDelegate, UICollectionViewDataSource, UICollectionViewDelegateFlowLayout {

func numberOfSections(in collectionView: UICollectionView) -> Int {

return 1

}

func collectionView(\_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {

return presenter.numberOfTweets()

}

func collectionView(\_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {

let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "TweetCellID", for: indexPath) as! TweetCell

let index = indexPath.item

var backgroundColor: UIColor

switch index % 3 {

case 0:

backgroundColor = UIColor(named: "SpecialBlue")!

case 1:

backgroundColor = UIColor(named: "SpecialGreen")!

default:

backgroundColor = UIColor(named: "SpecialYellow")!

}

cell.backgroundColor = backgroundColor

cell.setInfoLabel(name: presenter.nameForTweet(at: index), screenName: presenter.screenNameForTweet(at: index), verified: presenter.isVerifiedUser(at: index))

cell.setTweetText(text: presenter.textForTweet(at: index))

cell.setLike(presenter.isLikedTweet(at: index))

cell.setRetweeted(presenter.isRetweetedTweet(at: index))

cell.delegate = self

cell.index = index

presenter.imageForTweet(at: index, action: {

image in

DispatchQueue.main.async {

cell.profileImageView.image = image

}

})

return cell

}

func scrollViewDidScroll(\_ scrollView: UIScrollView) {

if (scrollView.contentOffset.y + scrollView.frame.size.height) > scrollView.contentSize.height {

presenter.loadMoreTweets()

}

}

func collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {

let width = UIScreen.main.bounds.width - 20

let measureLabel = UILabel()

measureLabel.text = presenter.textForTweet(at: indexPath.item)

measureLabel.numberOfLines = 0

measureLabel.font = UIFont(name: "Gill Sans", size: 16)

measureLabel.lineBreakMode = .byCharWrapping

measureLabel.translatesAutoresizingMaskIntoConstraints = false

measureLabel.widthAnchor.constraint(equalToConstant: (width - 80)).isActive = true

let height = measureLabel.systemLayoutSizeFitting(UIView.layoutFittingCompressedSize).height + 90

return CGSize(width: width, height: height)

}

func collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, insetForSectionAt section: Int) -> UIEdgeInsets {

return UIEdgeInsets(top: 30, left: 10, bottom: 30, right: 10)

}

}

extension UserController: TweetCellDelegate{

func userImageTapped(index: Int?) {

}

func retweetButtonTapped(index: Int?) {

guard let index = index else {return}

presenter.retweetTweet(at: index)

}

func likeButtonTapped(index: Int?) {

guard let index = index else {return}

presenter.likeTweet(at: index)

}

}

Листинг файла TweetCell.swift

import UIKit

class TweetCell: UICollectionViewCell {

weak var delegate: TweetCellDelegate?

var index: Int?

let profileImageView: UIImageView = {

let imageView = UIImageView()

imageView.contentMode = .scaleAspectFit

imageView.clipsToBounds = true

imageView.layer.cornerRadius = 48/2

imageView.backgroundColor = .white

imageView.isUserInteractionEnabled = true

return imageView

}()

let textLabel: UILabel = {

let label = UILabel()

label.font = UIFont.systemFont(ofSize: 14)

label.numberOfLines = 0

return label

}()

let infoLabel: UILabel = UILabel()

var likeButton = UIButton()

var retweetButton = UIButton()

override init(frame: CGRect) {

super.init(frame: frame)

addSubview(profileImageView)

let tap = UITapGestureRecognizer(target: self, action: #selector(userImageTapped))

tap.numberOfTapsRequired = 1

profileImageView.addGestureRecognizer(tap)

profileImageView.translatesAutoresizingMaskIntoConstraints = false

NSLayoutConstraint.activate([profileImageView.topAnchor.constraint(equalTo: topAnchor, constant: 12),

profileImageView.leadingAnchor.constraint(equalTo: leadingAnchor, constant: 8),

profileImageView.heightAnchor.constraint(equalToConstant: 48),

profileImageView.widthAnchor.constraint(equalToConstant: 48)])

let stack = UIStackView(arrangedSubviews: [infoLabel, textLabel])

addSubview(stack)

stack.axis = .vertical

stack.distribution = .fillProportionally

stack.spacing = 5

stack.translatesAutoresizingMaskIntoConstraints = false

NSLayoutConstraint.activate([stack.topAnchor.constraint(equalTo: profileImageView.topAnchor),

stack.leadingAnchor.constraint(equalTo: profileImageView.trailingAnchor, constant: 12),

stack.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -12)])

self.layer.cornerRadius = 20

textLabel.textColor = .black

textLabel.font = UIFont(name: "Gill Sans", size: 16)

let actionStack = UIStackView(arrangedSubviews: [likeButton, retweetButton])

actionStack.distribution = .fillEqually

actionStack.axis = .horizontal

actionStack.spacing = 30

addSubview(actionStack)

actionStack.translatesAutoresizingMaskIntoConstraints = false

NSLayoutConstraint.activate([actionStack.bottomAnchor.constraint(equalTo: bottomAnchor, constant: -8),

actionStack.centerXAnchor.constraint(equalTo: centerXAnchor),

actionStack.leadingAnchor.constraint(equalTo: leadingAnchor, constant: 12),

actionStack.centerXAnchor.constraint(equalTo: centerXAnchor)])

let likeTap = UITapGestureRecognizer(target: self, action: #selector(likeButtonTapped))

likeButton.addGestureRecognizer(likeTap)

likeButton.tintColor = .black

let retweetTap = UITapGestureRecognizer(target: self, action: #selector(retweetButtonTapped))

retweetButton.addGestureRecognizer(retweetTap)

retweetButton.tintColor = .black

}

func setInfoLabel(name: String, screenName: String, verified: Bool = false) {

let fullString = NSMutableAttributedString(string:"\(name) ", attributes: [.font : UIFont(name: "Gill Sans", size: 18)!, .foregroundColor: UIColor.black])

if (verified) {

let imageAttachment = NSTextAttachment()

imageAttachment.image = UIImage(systemName: "checkmark")

fullString.append(NSAttributedString(attachment: imageAttachment))

}

fullString.append(NSAttributedString(string: " @\(screenName)", attributes: [.font: UIFont.systemFont(ofSize: 14),

.foregroundColor: UIColor.darkGray]))

infoLabel.attributedText = fullString

}

func setTweetText(text: String) {

textLabel.text = text

}

func setLike(\_ isLiked: Bool) {

if isLiked {

let imageConfiguration = UIImage.SymbolConfiguration(weight: .heavy)

let image = UIImage(systemName: "heart.fill", withConfiguration: imageConfiguration)

likeButton.setImage(image, for: .normal)

}

else {

let imageConfiguration = UIImage.SymbolConfiguration(weight: .light)

let image = UIImage(systemName: "heart", withConfiguration: imageConfiguration)

likeButton.setImage(image, for: .normal)

}

}

func setRetweeted(\_ isRetweeted: Bool) {

if !isRetweeted {

let imageConfiguration = UIImage.SymbolConfiguration(weight: .light)

let image = UIImage(systemName: "arrow.2.squarepath", withConfiguration: imageConfiguration)

retweetButton.setImage(image, for: .normal)

}

else {

let imageConfiguration = UIImage.SymbolConfiguration(weight: .heavy)

let image = UIImage(systemName: "arrow.2.squarepath", withConfiguration: imageConfiguration)

retweetButton.setImage(image, for: .normal)

}

}

@objc func likeButtonTapped() {

delegate?.likeButtonTapped(index: index)

}

@objc func retweetButtonTapped(){

delegate?.retweetButtonTapped(index: index)

}

@objc func userImageTapped(){

delegate?.userImageTapped(index: index)

}

required init?(coder: NSCoder) {

fatalError("init(coder:) has not been implemented")

}

}

protocol TweetCellDelegate: AnyObject {

func userImageTapped(index: Int?)

func retweetButtonTapped(index: Int?)

func likeButtonTapped(index: Int?)

}

Листинг файла UserCell.swift

import UIKit

class UserCell: UITableViewCell {

@IBOutlet weak var userProfileImageView: UIImageView!

@IBOutlet weak var usernameLabel: UILabel!

@IBOutlet weak var screenNameLabel: UILabel!

@IBOutlet weak var descriptionLabel: UILabel!

override func awakeFromNib() {

super.awakeFromNib()

userProfileImageView.contentMode = .scaleAspectFit

userProfileImageView.clipsToBounds = true

userProfileImageView.layer.cornerRadius = 48/2

userProfileImageView.backgroundColor = .white

}

override func setSelected(\_ selected: Bool, animated: Bool) {

super.setSelected(selected, animated: animated)

}

func set(name: String, screenName: String, verified: Bool = false) {

let string = NSMutableAttributedString(string: "\(name)", attributes: [.font : UIFont.boldSystemFont(ofSize: 14), .foregroundColor: UIColor.black])

if (verified) {

string.append(NSAttributedString(attachment: NSTextAttachment(image: UIImage(systemName: "checkmark")!)))

}

usernameLabel.attributedText = string

screenNameLabel.text = "@\(screenName)"

}

}

ВЕДОМОСТЬ

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КР 1–40 01 01 204 ПЗ | | | | Пояснительная записка | | | | 85 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 951002 204 СП | | | | Схема программы «Twitter-клиент для платформы iOS» | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КР 1-40 01 01 204 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Twitter-клиент для платформы iOS  Ведомость курсовой  работы |  | | | | Лист | Листов |
| Разраб. | | Будович И.В. |  | 25.05.21 | Т |  | |  | 85 | 85 |
| Пров. | | Мельникова Е.В. |  | 25.05.21 | Кафедра ПОИТ  гр. 951002 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |