КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ВЫСШАЯ ШКОЛА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Кафедра программной инженерии

И.С. ШАХОВА, И.Р. ЗАЛЯЛОВ

ПРАКТИКУМ ПО КУРСУ “ВВЕДЕНИЕ В РАЗРАБОТКУ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ”

iOS

Казань – 2019

УДК 004.42

Принято на заседании учебно-методической комиссии Высшей школы информационных технологий и интеллектуальных систем
Протокол № 2 от 03 октября 2019 года

Рецензент:
кандидат технических наук,
заведующий кафедрой интеллектуальных технологий поиска Высшей школы информационных технологий и интеллектуальных систем Казанского федерального университета **Д.С. Зуев**

Шахова И.С.
Практикум по курсу “Введение в разработку мобильных приложений”. iOS / И.С. Шахова, И.Р. Залялов. – Казань: Казан. ун-т, 2019. – 23 с.

Практические задания предназначены для студентов второго курса Высшей школы информационных технологий и интеллектуальных систем, обучающихся по направлению 09.03.04 “Программная инженерия” и изучающих курс “Введение в разработку мобильных приложений” по направлению iOS.

В данной разработке представлены ключевые практические задания по курсу “Введение в разработку мобильных приложений”, базовая информация по работе с системой контроля версий Git, а также даны детальные инструкции по отправке результатов выполнения заданий для проверки.

СОДЕРЖАНИЕ

1 ОРГАНИЗАЦИЯ РАБОТЫ С GIT

Получение детализированной удаленной обратной связи от преподавателя в контексте решения настоящих заданий происходит посредством работы с системой контроля версий Git и сервиса GitHub.

Система контроля версий – это система, регистрирующая изменения в одном или нескольких файлах с той целью, чтобы в дальнейшем была возможность вернуться к определенным версиям этих файлов [1].

Git – это распределенная система контроля версий с открытым исходным кодом [2]. Распределенная система контроля версий позволяет обеспечивать оперативный обмен файлами между несколькими разработчиками в рамках одного проекта, поддерживая актуальную версию проекта у каждого из разработчиков. Всякий раз, когда пользователь распределенной системы контроля версий загружает актуальную версию файлов, он создает на своем локальном компьютере полную копию всех данных (рис. 1).

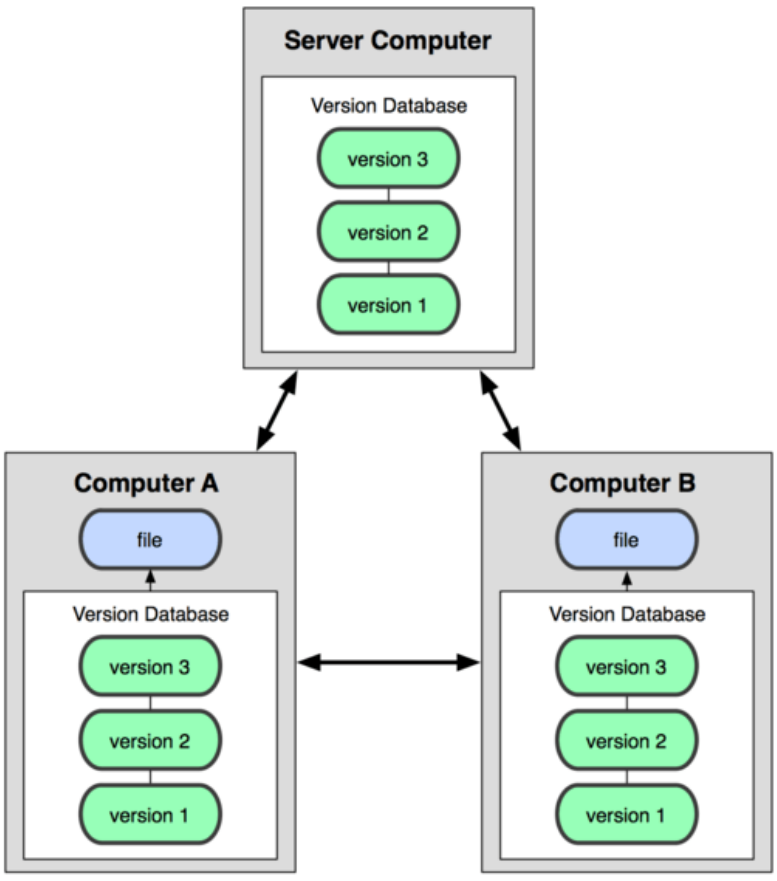

Рис. 1. Схема распределенной системы контроля версий [1]

Всякий раз, когда разработчик фиксирует текущую версию проекта, Git сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Если файл не был изменен, Git делает ссылку на ранее сохраненный файл. То, как Git подходит к хранению данных, изображено на рисунке 2.

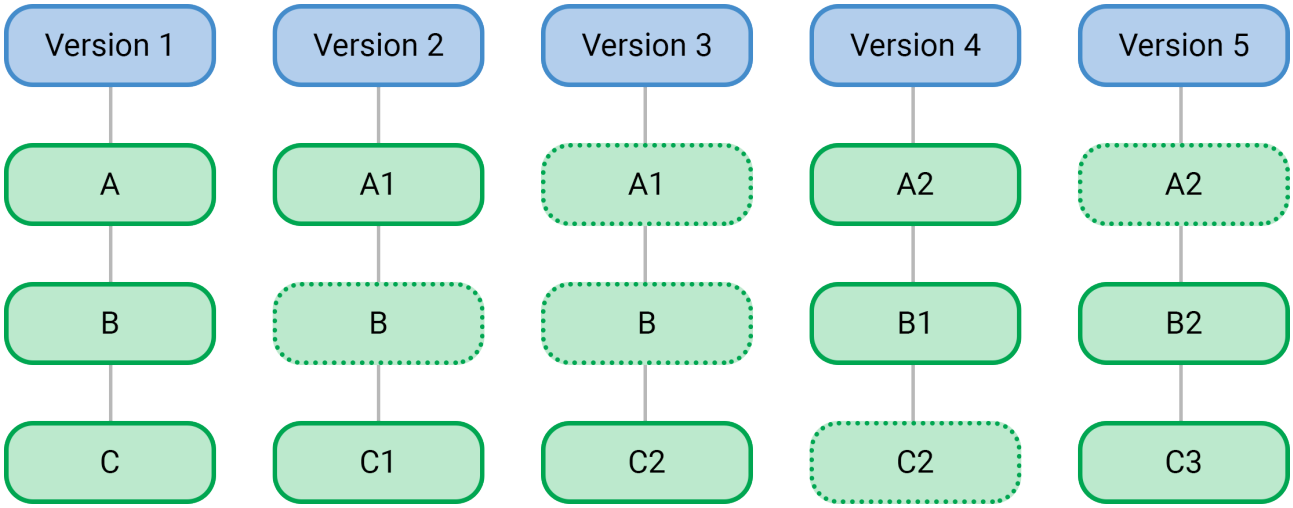

Рис. 2. Подход Git к хранению данных

Файлы в Git могут находиться в одном из трёх состояний: *зафиксированном*, *измененном* и *подготовленном*. “Зафиксированный” означает, что файл сохранен в локальном репозитории. К измененным относятся файлы, которые были изменены, но не были зафиксированы. Подготовленные файлы – это измененные файлы, отмеченные для включения в следующее фиксирование.

Проекты, использующие систему контроля версий Git включают в себя следующие области: git-репозиторий, рабочий каталог (working directory) и область подготовленных файлов (staging area) [3].

Базовый процесс использования системы контроля версий Git представляет собой следующий набор шагов (рис. 3):

- внесение изменений в файлы;
- подготовка файлов – добавление их слепков в область подготовленных файлов;
- фиксирование (commit) – помещение подготовленных файлов из staging area в git-репозиторий на постоянное хранение.

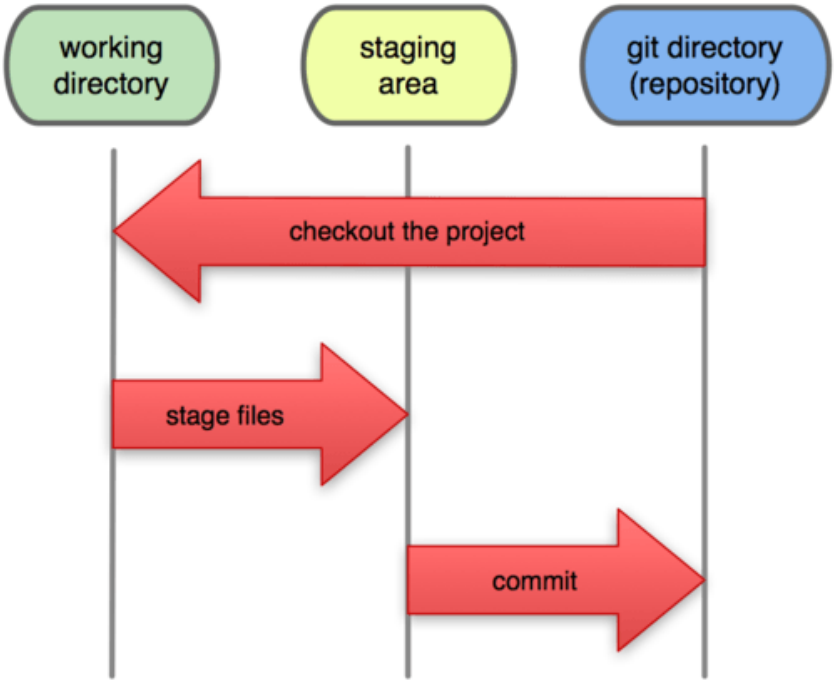

Рис. 3. Процесс фиксирования изменений [3]

Команды системы контроля версий Git, знание которых необходимо для успешного решения заданий в рамках курса “Введение в разработку мобильных приложений” описаны ниже.

git clone

Данная команда необходима для получения копии существующего git-репозитория проекта (например, шаблона задания в рамках данного курса). Командой *git clone* на локальный компьютер будет загружена копия всех данных проекта, которые есть на сервере в удаленном репозитории (например, на Git-Hub). Клонирование репозитория осуществляется командой:

```
git clone ссылка_на_репозиторий
```

Например, для клонирования данных репозитория с исходным кодом системы контроля версий Git, в программе “Терминал” (для ОС macOS) необходимо выполнить команду:

```
git clone https://github.com/git/git.git
```

git status

Команда *git status* – основной инструмент для определения состояния файлов проекта. В числе прочего, в контексте выполнения заданий по курсу, выполнение данной команды в репозитории проекта позволит контролировать отслеживание изменений в файлах, чтобы убедиться, что все файлы, в которые были внесены изменения, были отправлены в репозиторий и будут доступны для проверки преподавателем.

git add

Команда *git add* отвечает за добавление новых файлов под версионный контроль, а также за индексацию изменений файлов, уже находящихся под версионным контролем. Таким образом, для того, чтобы сделанные изменения попали под контроль версий и были зафиксированы при следующем коммите, необходимо выполнить команду *git add* для необходимых файлов. Добавление всех файлов проекта под контроль версий осуществляется командой:

```
git add -A
```

git commit

Следующим этапом после выполнения команды *git add* и добавления всех необходимых файлов под контроль версий является этап фиксации изменений. Фиксация изменений в репозитории на локальном компьютере происходит посредством выполнения команды:

```
git commit -m ”комментарий_к_коммиту”
```

Добавление комментария к коммиту необходимо для понимания того, что конкретно было изменено в рамках данного коммита. Таким образом, комментарий к коммиту должен быть емким, но содержательно описывать изменения, внесенные в рамках данного коммита.

Проверить текущее состояние файлов проекта и убедиться, что все изменения были зафиксированы, можно при помощи вышеописанной команды *git status*.

git push

После того, как все изменения были зафиксированы на локальном компьютере, необходимо отправить их в удаленный репозиторий (в рамках данного курса – на GitHub). Для отправки данных необходимо выполнить команду:

```
git push
```

После выполнения команды *git push* все данные будут отправлены в удаленный репозиторий.

2 СОЗДАНИЕ PULL REQUEST

Отправка изменений в удаленный репозиторий в рамках выполнения заданий по данному курсу происходит путем внесения изменений в созданный преподавателем шаблон проекта посредством создания запросов на принятие изменений (pull request).

Далее будет рассмотрена последовательность действий, которые необходимо совершить для успешной отправки результатов задания на проверку.

1. ***Авторизоваться на GitHub*** по ссылке https://github.com/login, введя Username или email и пароль от аккаунта. При отсутствии аккаунта на GitHub необходимо зарегистрироваться в системе по ссылке https://github.com (форма регистрации доступна на главной странице веб-сервиса). Для регистрации необходимо заполнить поля: Username, Email, Password.

2. ***Перейти по ссылке на GitHub с заданием***. После перехода по ссылке с заданием на экране отобразится страница репозитория с шаблоном задания, подготовленного преподавателем. Для выполнения задания будет необходимо внести требуемые в задании изменения (п.8) в данный шаблон. Для внесения

изменений необходимо в первую очередь создать собственное ответвление (собственную копию проекта).

3. ***Создать ответвление (fork) проекта.*** Для этого необходимо нажать на кнопку *"Fork"* на странице репозитория (рис. 4). После выполнения данного действия будет автоматически совершено перенаправление к репозиторию с собственной копией проекта и правами на запись (рис. 5).

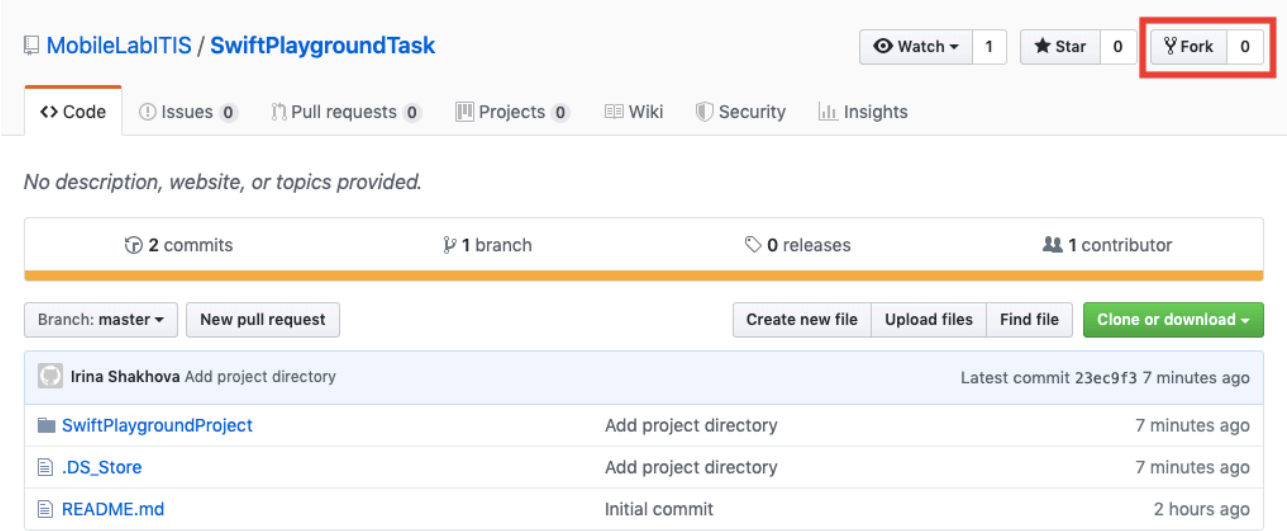

Рис. 4. Страница репозитория с шаблоном проекта

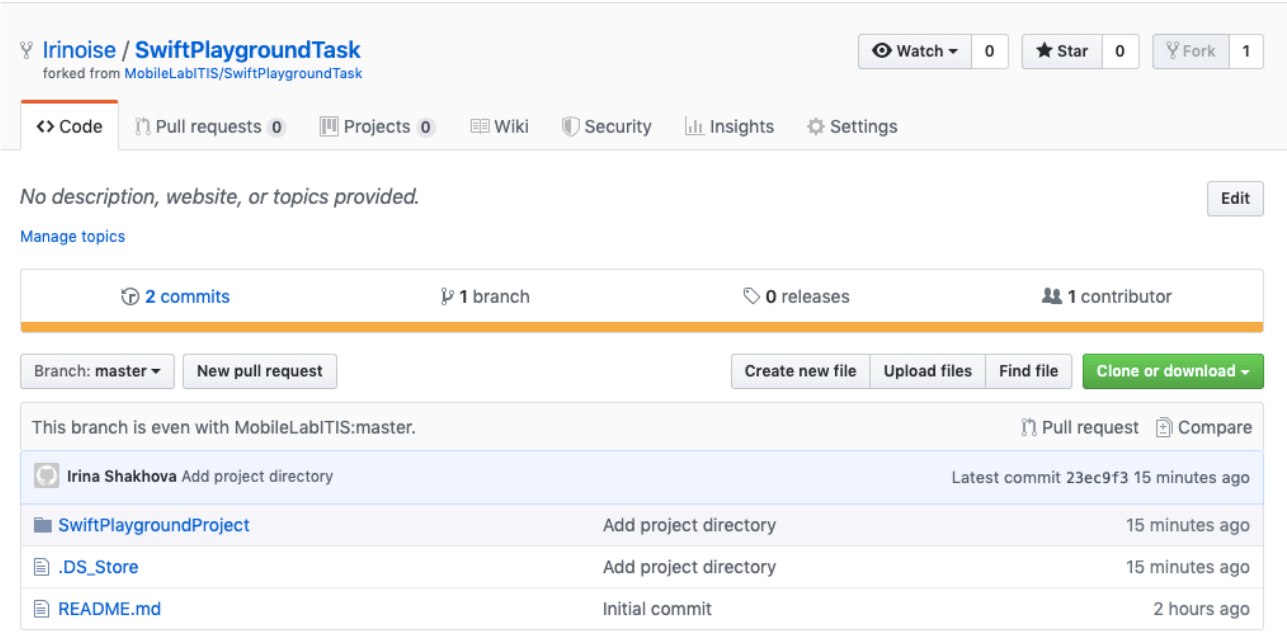

Рис. 5. Страница копии проекта в репозитории студента

4. ***Склонировать репозиторий*** на локальный компьютер. Для этого необходимо нажать на кнопку *“Clone or download”* и в появившемся элементе скопировать ссылку на репозиторий (рис. 6).

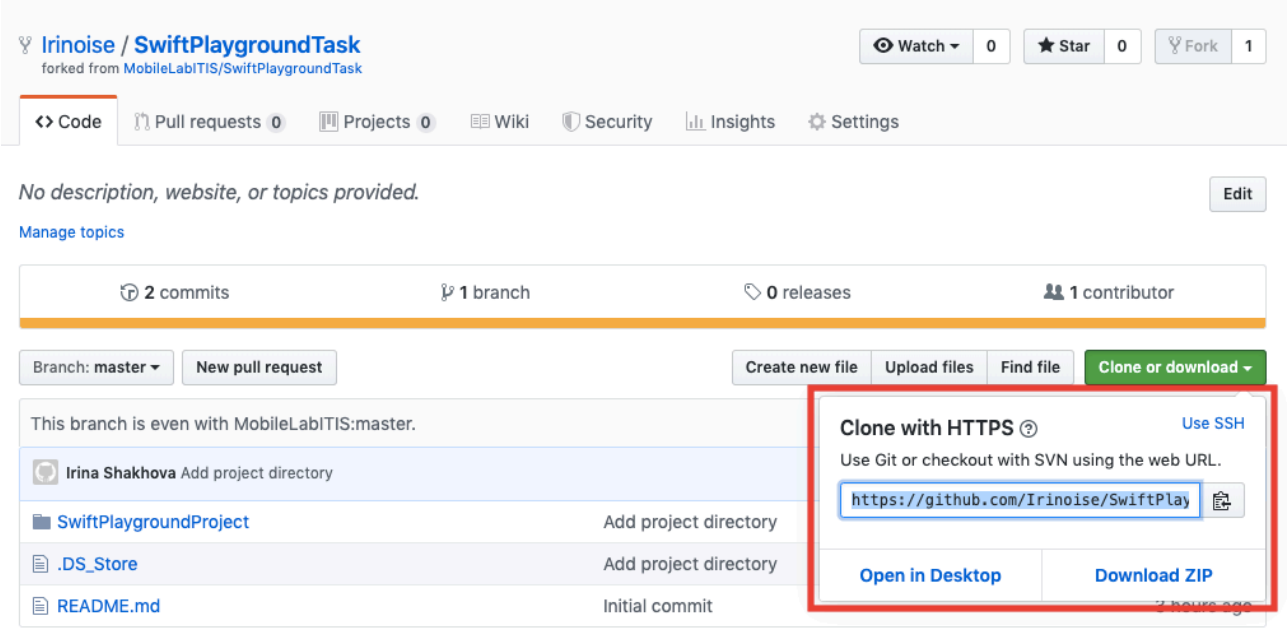

Рис. 6. Копирование ссылки на репозиторий

Далее необходимо открыть на компьютере (пример показан для компьютера с ОС macOS) программу *“Терминал”*, перейти в директорию, куда необходимо склонировать проект, и ввести команду:

```
git clone скопированная_ссылка_на_репозиторий
```

Вид окна программы “Терминал” для текущего примера приведен на рисунке 7.

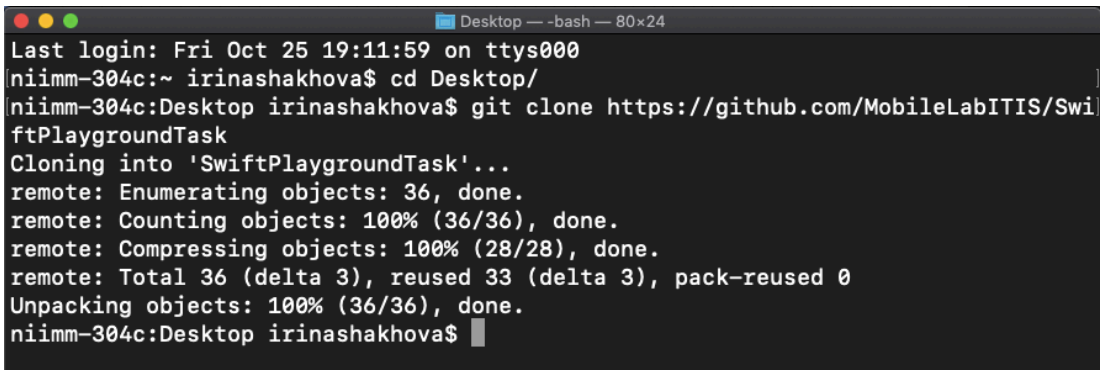

Рис. 7. Вид окна программы “Терминал” после команды *git clone*

5. ***Внести изменения, отвечающие требованиям задания.*** После выполнения операции *git clone* в соответствующей директории на локальном компьютере появится папка с шаблоном проекта, который был склонирован из удаленного репозитория. Необходимо открыть папку с проектом, открыть проект в IDE Xcode и выполнить соответствующее практическое задание. Краткие комментарии по заданию можно найти в файле README склонированного проекта.

Примечание 1

После внесения всех необходимых изменений можно в программе "Терминал" перейти в локальный репозиторий проекта выполнить команду *git status*, чтобы убедиться, что все внесенные изменения отображаются корректно. Пример вывода результата команды *git status* изображен на рисунке 8.

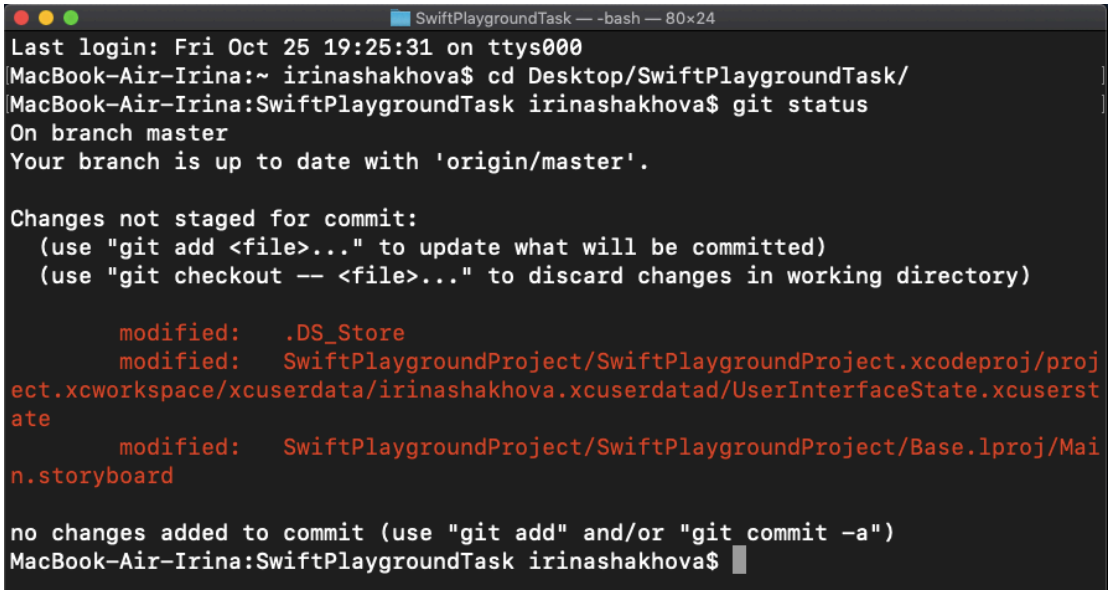

Рис. 8. Вид окна программы "Терминал" после команды *git status*

6. ***Зафиксировать изменения в локальном репозитории.*** Для выполнения данного этапа необходимо добавить новые и измененные файлы проекта под контроль версии и осуществить коммит внесенных изменений. Для этого необходимо выполнить команды (результат выполнения команд представлен на рисунке 9):

```
git add -A
git commit -m ”комментарий_к_коммиту”
```

Примечание 2

После выполнения команд можно повторно выполнить *git status*, чтобы убедиться, что в локальном репозитории не осталось незафиксированных изменений.

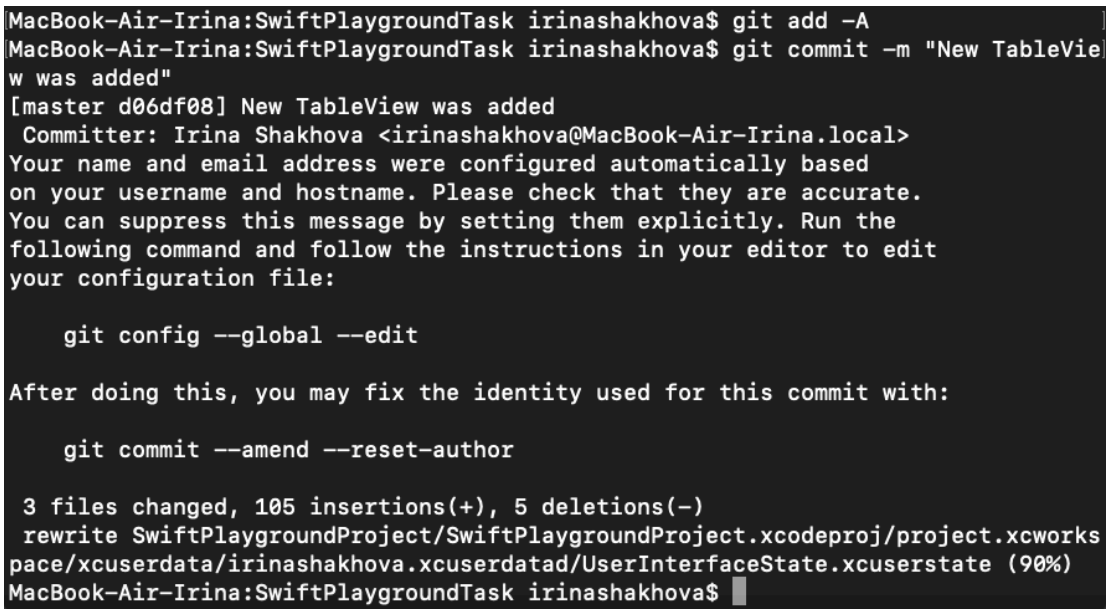

Рис. 9. Результат выполнения команд *git add* и *git commit*

7. ***Отправить изменения в удаленный репозиторий на GitHub.*** Для этого необходимо выполнить команду *git push.* Пример результата выполнения команды представлен на рисунке 10.

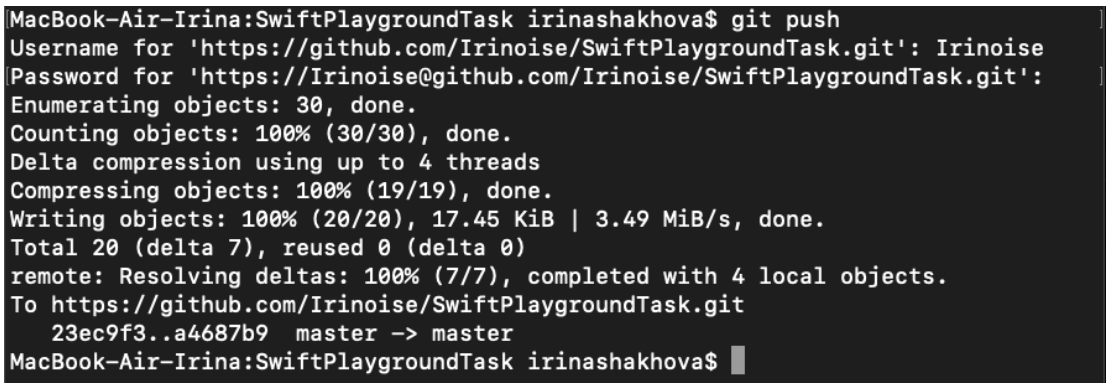

Рис. 10. Результат выполнения команды *git push*

После выполнения команды *git push* изменения должны отобразиться на странице репозитория студента на GitHub. Необходимо обратить внимание, что изменения на текущий момент времени попали только в удаленный репозиторий студента.

Для того, чтобы изменения попали на проверку преподавателю, необходимо выполнить *pull request*, что будет описано ниже. Однако, в первую очередь, необходимо проверить, все ли коммиты учтены и готово ли задание для сдачи. Для этого необходимо перейти в свой репозиторий на GitHub и просмотреть коммиты на вкладке с количеством коммитов (рис. 11).

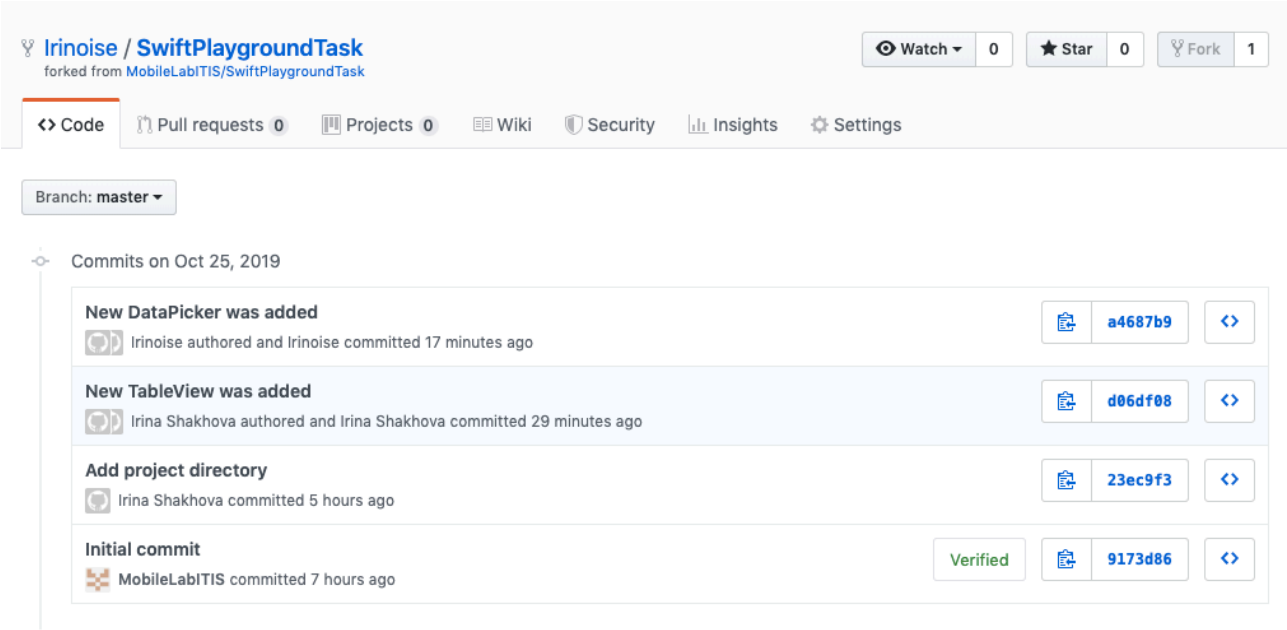

Рис. 11. Список коммитов пользователя

Если все коммиты учтены и задание готово для сдачи, то необходимо перейти к созданию *pull request* для отправки результата на проверку преподавателю.

1. По нажатию на название проекта перейти на экран проекта и нажать на кнопку ***“New pull request”*** (рис. 12).

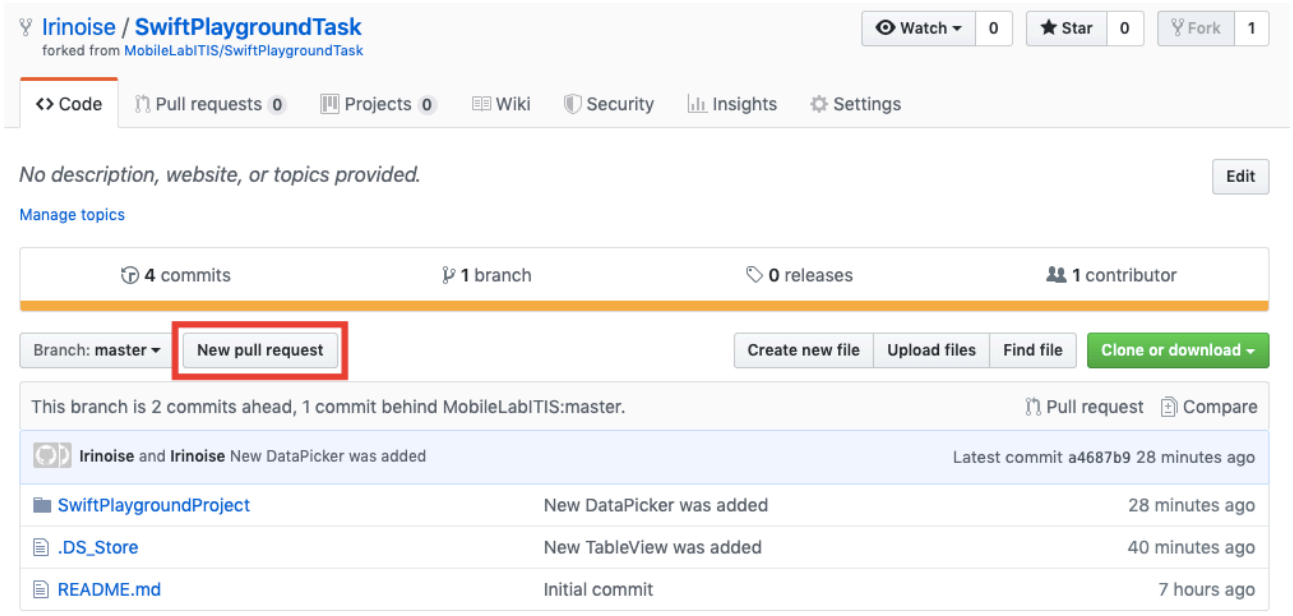

Рис. 12. Создание *pull request.* Шаг первый

2. В зависимости от типа внесенных изменений может отображаться “Able to merge” или “Can’t automatically merge”. Первое означает, что в случае слияния изменений, слияние будет проведено без конфликтов, второе говорит о том, что при слиянии изменений возникнут конфликты. В рамках решения практических задач на данном курсе обращать внимание на это не стоит. При любом раскладе необходимо нажать на кнопку “Create pull request” (рис. 13).

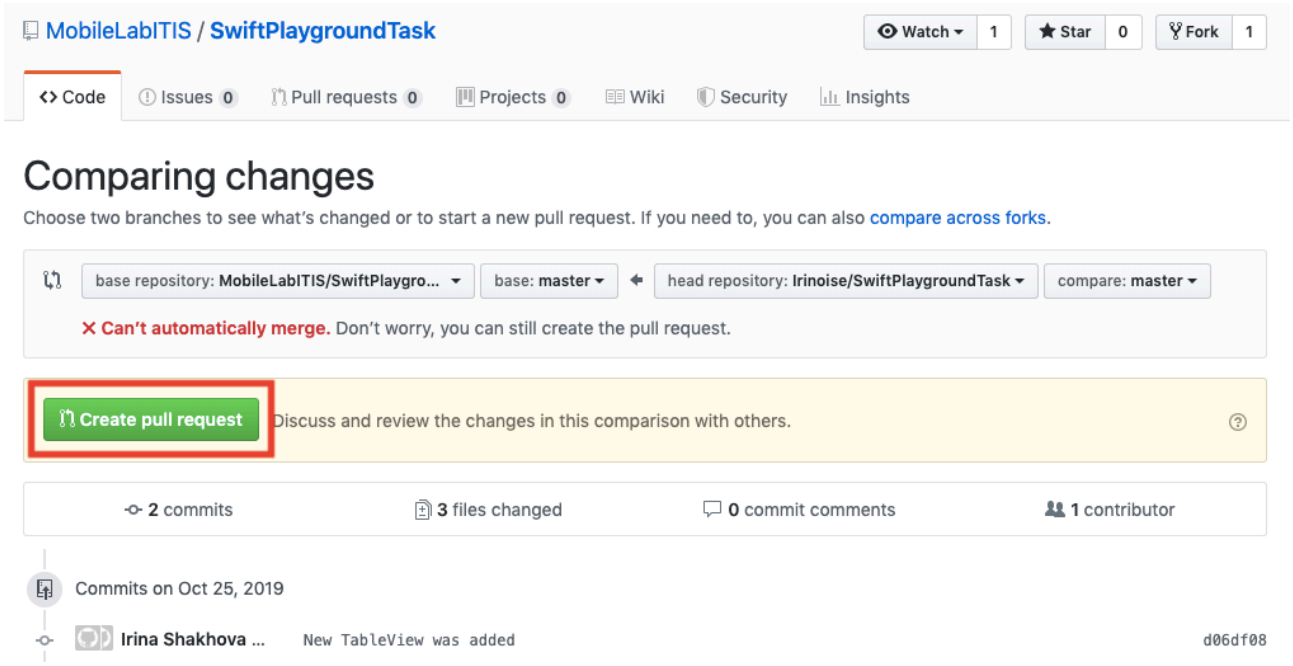

Рис. 13. Создание *pull request.* Шаг второй

3. Заполнить поля и нажать “*Create pull request”* (рис. 14).

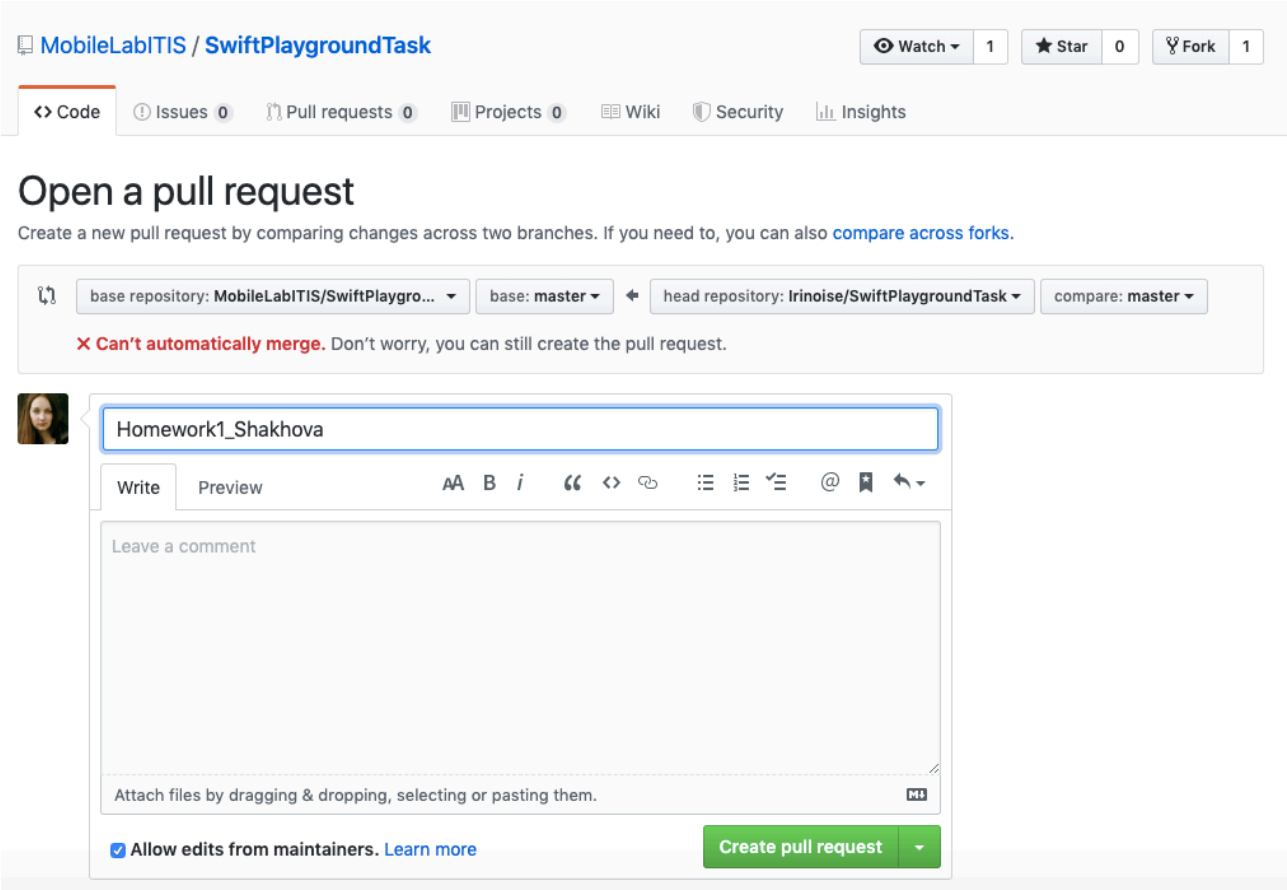

Рис. 14. Создание *pull request.* Шаг третий

4. Проверить наличие *pull request* в базовом репозитории: перейти в проект (кликнуть по названию проекта), перейти на вкладку *“Pull requests”* (рис. 15).

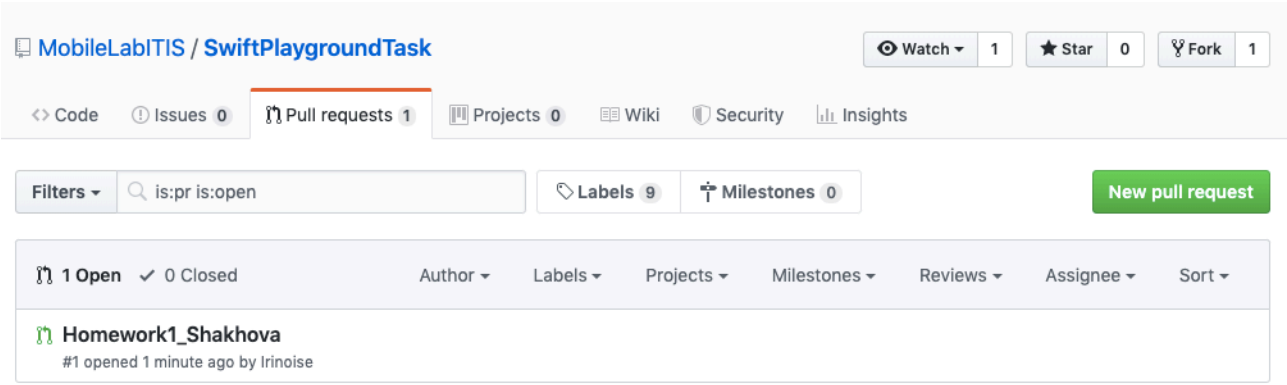

Рис. 15. Вкладка *“Pull requests”*

3 ПРАКТИЧЕСКИЕ ЗАДАНИЯ

3.1 Storyboard

Storyboard – это визуальное представление пользовательского интерфейса мобильных приложений для операционной системы iOS, отображающее экраны с их содержимым, а также соединения между экранами. Storyboard состоит из последовательности сцен, каждая из которых представлена посредством View Controller и View; сцены соединены между собой связями, которые отображают переходы между связанными экранами [4].

Interface Builder – визуальный инструмент проектирования пользовательских интерфейсов iOS-приложений, встроенный в IDE Xcode. Данный инструмент предоставляет возможность визуального проектирования UI мобильных приложений для ОС iOS независимо от их реализации.

Autoresizing Mask – компонент, позволяющий задавать правила изменения размеров элементов интерфейса в зависимости от размеров экрана.

В задании, посвященном развитию навыков работы со сторибордом, необходимо при помощи инструментов Interface Builder сверстать экран профиля на основе социальной сети ВКонтакте.

- Все элементы интерфейса должны быть соединены с кодом и отображать заполненные данные.
- Необходимо обеспечить корректное расположение элементов интерфейса на различных размерах экранов, воспользовавшись возможностями Autoresizing Mask.
- Страница должна прокручиваться для предоставления информации, которая не поместилась на экране.

На экране должны быть отображены:

- аватар пользователя;
- ФИО;
- возраст;
- место жительства;
- редактируемый статус;

- контакты (адрес электронной почты, ссылки на другие соцсети);
- карьера (места работы);
- образование (вуз, школа);
- подарки (отображение 3-5 подарков).

Типы элементов, которые необходимо использовать для отображения соответствующих данных, студентам предлагается определить самостоятельно.

3.2 Table View Controller

Table View Controller – класс, используемый при отображении данных в табличном виде.

UIAlertController – класс, используемый для конфигурирования диалоговых окон с сообщениями (*alert*) или списком действий (*action sheet*).

Для развития навыков работы с данными в табличном виде необходимо реализовать мобильное приложение для ОС iOS, состоящее из двух экранов: список публикаций и детальное отображение выбранной публикации. Данные (текст, картинки), отображаемые в приложении, хранятся локально.

Список публикаций. Каждая публикация может включать в себя содержимое двух типов: текст и изображение. Размер ячеек должен соответствовать количеству содержимого. Текст большого размера отображается в сокращенном виде в количестве 5-6 строк. По нажатию на публикацию осуществляется переход на экран с детальным отображением.

Детальное отображение публикации. На данном экране в полном размере отображается всё содержимое публикации. В правом верхнем углу Navigation Bar на данном экране отображается кнопка, по нажатию на которую отображается диалоговое окно с действиями: “редактировать”, “удалить”, “отмена”.

По нажатию на “Редактировать” отображается экран с текстом публикации с возможностью его редактирования. После редактирования текста изменения отображаются на экране детального отображения и на экране списка публикаций.

По нажатию на “Удалить” отображается диалоговое окно с подтверждением удаления публикации. После удаления публикации осуществляется переход на экран списка публикаций.

По нажатию на “Отмена” диалоговое окно скрывается.

3.3 Auto Layout

Auto Layout динамически вычисляет размер и расположение всех элементов пользовательского интерфейса в иерархии элементов, основываясь на ограничениях (*constraint*), ассоциированных с данными элементами. Например, если на кнопку наложены ограничения в виде центрирования по горизонтали относительно Image View и верхний край кнопки имеет отступ относительно нижнего края Image View, то в случае изменения положения Image View положение кнопки изменится в соответствии с ограничениями [5].

Для практики навыков работы с Auto Layout необходимо реализовать мобильное приложение для ОС iOS со следующими функциональными возможностями:

1) авторизация в приложении;
2) экран со списком публикаций из задания 3.2;
3) экран профиля из задания 3.1.

Авторизация. Экран содержит поля “логин” и “пароль”. Значение, введенное в поле “логин”, должно проверяться на соответствие формату email. Значение, введенное в поле “пароль” должно проверяться на соответствие следующим условиям: а) пароль должен состоять не менее чем из 6 символов; б) пароль должен включать в себя исключительно латиницу; в) пароль должен содержать хотя бы одну цифру. В приложении должна быть задана информация по 3 пользователям. Для каждой из трех пар “логин”-”пароль” должен быть определен свой набор данных (публикации, информация профиля). Все данные, отображаемые в приложении, хранятся локально.

При успешной авторизации осуществляется переход к *экрану со списком публикаций* из задания 3.2. Помимо функционала, реализованного в рамках

прошлого задания, на данном экране должна отображаться краткая информация о пользователе (аватар, статус, место работы/учебы). Также на данный экран должна быть добавлена кнопка “i” по нажатию на которую будет осуществляться переход к экрану с подробной информацией профиля из задания 3.1. По нажатию на ячейку с публикацией должен осуществляться переход к детальной версии публикации (реализованной в рамках задания 3.2).

Весь пользовательский интерфейс приложения должен корректно отображаться на устройствах с разными размерами экранов и быть реализован с использованием Auto Layout.

3.4 Паттерн MVC. Блоки кода

MVC (Model-View-Controller) – шаблон проектирования, согласно которому объекты приложения могут представлять одну из трех ролей: модель (*Model*), представление (*View*) и контроллер (*Controller*). Каждый из трех типов объектов, согласно данному шаблону, не зависит от остальных. Объекты *Model* включают в себя данные приложения, а также определяют логику и вычисления, которые обрабатывают эти данные. Пользовательские действия на уровне представления, которые создают или изменяют данные, передаются через объект контроллера, который взаимодействует с моделью, изменяя или создавая объекты модели. С другой стороны, когда изменяется объект модели (например, новые данные получены из сети), он уведомляет объект контроллера, который, в свою очередь, обновляет соответствующие объекты представления. Объекты *View* – это объекты пользовательского интерфейса. Основной задачей объектов представления является отображение данных из объектов модели приложения и предоставления возможности редактирования этих данных. Объекты представления узнают об изменениях в объектах модели через объекты контроллера и передают инициированные пользователем изменения через объекты контроллера в объекты модели. Объекты *Controller* выступает в качестве посредника между объектами модели и объектами представления [6].

UICollectionView – класс, используемый для управления упорядоченными коллекциями объектов данных и отображения их посредством настраиваемых макетов [7].

Блоки кода (Замыкания, Closures) – самодостаточные блоки кода с определенным функционалом, которые могут быть использованы в различных частях программы, в том числе в виде параметров.

Для формирования навыков работы с MVC и замыканиями, необходимо реализовать прототип профиля пользователя социальной сети Instagram. Функциональные возможности, которые должны быть реализованы в рамках прототипа:

1) информация о пользователе (аватар, количество публикаций/подписок/подписчиков);
2) отображение изображений пользователя в виде сетки с использованием UICollectionView;
3) переход к полноразмерному изображению по нажатию на превью изображения.

Приложение должно быть спроектировано на основе шаблона MVC и реализовывать все его принципы.

Информация о пользователе, изображения и данные для одной ячейки UICollectionView должны возвращаться с помощью блоков кода.

3.5 GCD. Operation Queue

GCD (Grand Central Dispatch) содержит языковые функции, runtime-библиотеки и системные усовершенствования, которые обеспечивают системные комплексные улучшения поддержки параллельного выполнения кода [8].

Operation Queue выполняет поставленные в очередь объекты Operation, исходя из приоритета и готовности. После добавления в очередь операция остается там до тех пор, пока не сообщит о завершении своей задачи.

Задание для практики навыков работы с GCD состоит из набора подзадач, описанных ниже.

1) Реализовать протокол по работе с данными, включающий методы:
   - сохранения моделей;
   - получения моделей;
   - поиска моделей по названию или id.
2) Добавить класс Manager, который будет реализовывать протокол по работе с данными.
3) Каждая операция должна иметь синхронную и асинхронную версии.
4) Асинхронные методы должны иметь completionBlock, в котором будет возвращаться результат сохранения, получения, поиска.
5) Для каждой асинхронной операции (сохранение, получение, поиск) должны быть реализованы OperationQueue.
6) Используя реализацию Manager, вывести массив данных в Table View.
7) При добавлении новой записи сохранение должно происходить асинхронно, затем должен быть выполнен асинхронный запрос данных из Manager-а и обновление данных Table View.

3.6 Клиент-серверное взаимодействие

Core Data – фреймворк, используемый для сохранения данных приложения для автономного использования или кэширования временных данных.

Для развития навыков организации клиент-серверного взаимодействия в приложениях для ОС iOS необходимо реализовать подгрузку данных для прототипа, реализованного в задачах 3.3 или 3.4, из интернета. В качестве источника данных можно использовать VK API [9], Twitter API [10] или Instagram API [11].

Необходимо реализовать следующие функциональные возможности:

1) авторизация;
2) подгрузка информации о текущем пользователе;
3) подгрузка публикаций пользователя;
4) создание новой публикации;
5) отметка “Мне нравится”.

Полученная информация должна кэшироваться с использованием Core Data. Для решения поставленной задачи необходимо использовать исключительно встроенные компоненты iOS SDK.

ЛИТЕРАТУРА

1. Git – о контроле версий. URL: https://git-scm.com/book/ru/v1/Введение-О-контроле-версий (дата обращения: 25.10.2019).
2. Git Source Code Mirror. URL: https://github.com/git/git (дата обращения: 25.10.2019).
3. Git – основы Git. URL: https://git-scm.com/book/ru/v1/Введение-Основы-Git (дата обращения: 25.10.2019).
4. Storyboard. URL: https://developer.apple.com/library/archive/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html (дата обращения: 26.10.2019).
5. Auto Layout Guide: Understanding Auto Layout. URL: https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutolayoutPG/index.html (дата обращения: 26.10.2019).
6. Model-View-Controller. URL: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html (дата обращения: 27.10.2019).
7. UICollectionView – UIKit // Apple Developer Documentation. URL: https://developer.apple.com/documentation/uikit/uicollectionview (дата обращения: 27.10.2019).
8. Dispatch // Apple Developer Documentation. URL: https://developer.apple.com/documentation/DISPATCH (дата обращения: 27.10.2019).
9. Запросы к API // Разработчикам. URL: https://vk.com/dev/api_requests (дата обращения: 27.10.2019).
10. Docs // Twitter Developers. URL: https://developer.twitter.com/en/docs (дата обращения: 27.10.2019).

11. API Instagram Basic Display // Платформа Instagram. URL: https://developers.facebook.com/docs/instagram-basic-display-api/ (дата обращения: 27.10.2019).