



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Головной учебно-исследовательский и методический центр
профессиональной реабилитации лиц с ограниченными возможностями
здоровья (инвалидов)
КАФЕДРА «Информационная безопасность» (ИУ8)

ОТЧЁТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Тип практики: Технологическая практика

Название предприятия: ПАО «Сбербанк»

Студент: группа ИУ8Ц-101 л.д. 17Ц019

Кардаш Кирилл Григорьевич

(подпись, дата)

Руководитель от предприятия:

Руководитель направления Львов Константин Васильевич

(подпись, дата)

Руководитель от кафедры:

доцент кафедры ИУ8 Зайцева Анастасия Владленовна

(подпись, дата)

Оценка: _____

Москва, 2022



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Головной учебно-исследовательский и методический центр
профессиональной реабилитации лиц с ограниченными возможностями
здоровья (инвалидов)
КАФЕДРА «Информационная безопасность» (ИУ8)

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА ПРАКТИКУ

Название предприятия: ПАО «Сбербанк»

Сроки практики: с 30.06.2022 по 27.07.2022

Специальность / направление: 10.05.03_61 Информационная безопасность
автоматизированных систем

Специализация / профиль: Анализ безопасности информационных систем

За время прохождения практики студенту надлежит согласно программе практики:

Разработать аутентификацию с использованием Firebase на основе пользовательских и биометрических данных (Face ID или Touch ID), электронной почты и через Google в мобильном приложении iOS.

Руководитель от кафедры:

доцент кафедры ИУ8 Зайцева Анастасия Владленовна

(подпись, дата)

Руководитель от предприятия:

Руководитель направления Львов Константин Васильевич

(подпись, дата)

Студент: группа ИУ8Ц-101 л.д. 17Ц019

Кардаш Кирилл Григорьевич

(подпись, дата)

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	4
ОСНОВНАЯ ЧАСТЬ	6
1. Характеристика организации.....	6
2. Основные понятия и определения.....	6
3. Основные практики обеспечения безопасности iOS-приложений	6
3.1 Хранение данных.....	6
3.2 Touch ID и Face ID.....	7
3.3 Ввод данных.....	7
4. Разработка аутентификации в iOS-приложении.....	8
4.1 Начало работы.....	8
4.2 Интеграция с Firebase	10
4.3 Реализация аутентификации	13
4.4 Реализация входа с помощью Google	15
4.5 Реализация входа с помощью Touch ID и Face ID.....	17
ЗАКЛЮЧЕНИЕ.....	22
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	23
ПРИЛОЖЕНИЕ А.....	24
ПРИЛОЖЕНИЕ Б	29
ПРИЛОЖЕНИЕ В.....	30

ВВЕДЕНИЕ

В последние годы для аутентификации личности человека наиболее перспективным считается применение биометрических технологий, особенно в системах разграничения доступа при проведении финансовых операций, при запросах информации ограниченного доступа по телефону, при управлении различными устройствами, в криминалистике и т.д. Применение биометрических технологий обладает рядом существенных преимуществ перед традиционными средствами аутентификации, например более высокой подлинностью аутентификации и удобством использования [1].

Исследованиями проблем аутентификации пользователей по голосу занимаются во многих научно-исследовательских институтах, центрах разработки программного обеспечения, промышленные корпорациях, силовые структуры разных стран, что отражает актуальность исследуемой проблемы.

Наиболее распространёнными технологиями биометрической аутентификации человека является применение отпечатков пальцев и радужной оболочки глаза [2].

Биометрические системы аутентификации обладают существенными отличиями от традиционных систем контроля и управления доступом (СКУД), использующих в качестве идентификаторов личности символьные пароли и электронные ключи. Биометрическая аутентификация производится по действительно индивидуальным признакам личности, а не по ассоциированному с личностью материальному носителю или коду. Биометрия практически исключает возможность несанкционированных действий, связанных с потерей, кражей или передачей пароля третьим лицам.

Целью прохождения практики является разработка парольной и биометрической аутентификации в мобильном приложении под платформу iOS.

Для достижения поставленной цели необходимо выполнить следующие

задачи:

1. Ознакомиться с методами обеспечения безопасности iOS;
2. Изучить документацию по аутентификации Firebase для iOS;
3. Разработать мобильное приложение с аутентификацией на языке Swift на фреймворке UIKit;
4. Тестировать приложение для проверки корректной работы системы аутентификации.

ОСНОВНАЯ ЧАСТЬ

1. Характеристика организации

Публичное акционерное общество «Сбербанк» (Далее - ПАО «Сбербанк») — это универсальная организация, входящая в число крупнейших российских коммерческих банков России и стран СНГ [3]. «Сбербанк» теперь не только финансовая организация, но и является крупнейшей российской IT-компанией, занимающейся разработкой и внедрением программных решений. Структура внесена в реестр организаций в области IT, аккредитованных Минцифры.

«Сбербанк» имеет огромную филиальную сеть: 17 территориальных банков и более 18 400 подразделений. Он оказывает услуги во всех 83 субъектах Российской Федерации. Происходит развитие приложений «Сбербанк Онлайн» и «Мобильный банк» с широкой клиентской базой.

2. Основные понятия и определения

UIKit – это модульный front-end фреймворк для разработки графических пользовательских интерфейсов под iOS, iPadOS, Mac и tvOS.

Firebase — это полноценный платформенный сервис со множеством инструментов для разработки мобильных приложений [4].

3. Основные практики обеспечения безопасности iOS-приложений

3.1 Хранение данных

iOS всегда славится своей защищенностью и вниманием к информационной безопасности. Тем не менее за время существования этой ОС было выявлено несколько серьезных уязвимостей, из-за которых происходили утечки пользовательских данных [5]. Это лишний раз напоминает о том, что слишком много мер по обеспечению безопасности не бывает и нельзя во всем надеяться на систему. Чем меньше информации остается на диске после

использования приложения, тем лучше.

Поэтому рекомендуется хранить только те данные, без которых обойтись невозможно. Если среди них есть персональная информация пользователя — она хранится исключительно в Keychain. Если же необходимо использовать полноценную базу данных, такую как Core Data или Realm, она обязательно шифруется.

Не стоит забывать и про то, что после взаимодействия с приложением на диске будут оставаться те данные, которые разработчик не собирался сохранять специально, например логи. Поэтому в приложении на диск не должно логироваться либо ничего, либо только информация, в которой не фигурируют личные данные.

3.2 Touch ID и Face ID

Биометрическая аутентификация существенно упрощает вход в мобильные приложения. Apple приводит статистику, согласно которой шанс совпадения отпечатка пальца пользователя с отпечатком другого человека равен 1 к 50 000 [6], в то время как шанс совпадения сканов лиц — 1 к 1 000 000 [7]. Все связанные с этим вычисления производятся в сопроцессоре Secure Enclave, который полностью изолирован от операционной системы. Из-за этого получить доступ к биометрическим данным пользователя или воспользоваться ими невозможно.

3.3 Ввод данных

Важно обеспечить безопасность при вводе информации внутри приложения. В большинстве текстовых полей рекомендуется отключать возможность автозаполнения (свойство `UITextField autoCorrectionType`).

Если этого не сделать, вводимые данные, которые могут быть персональными, будут индексироваться операционной системой и станут появляться в качестве вариантов для автозаполнения в других приложениях. Все текстовые поля, в которых вводятся пароли, автоматически маскируются и не

поддерживают возможность копирования/вставки.

4. Разработка аутентификации в iOS-приложении

4.1 Начало работы

Был создан проект с названием «Auth-Firebase» в среде разработки Xcode с интерфейсом Storyboard и языком Swift, как показано на рисунке 1. Storyboard – это удобный механизм разработки интерфейса программы, который позволяет уменьшить количество кода, связанного с переходами между экранами и с настройкой ячеек в таблице.

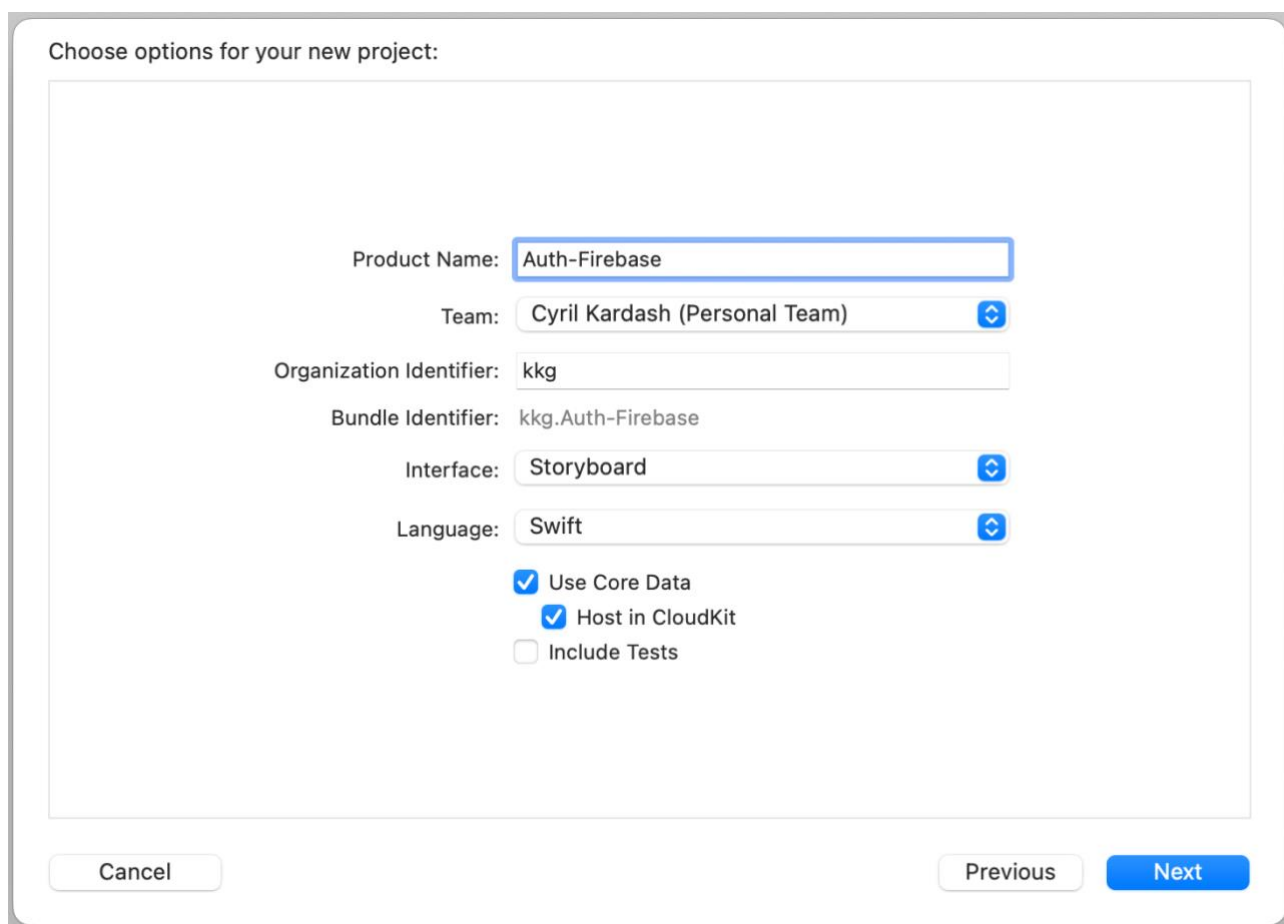


Рисунок 1 – Создание проекта в среде разработки Xcode

После добавления графических элементов в интерфейс с помощью Storyboard и

кода интерфейс аутентификации выглядит так, как показано на рисунке 2.

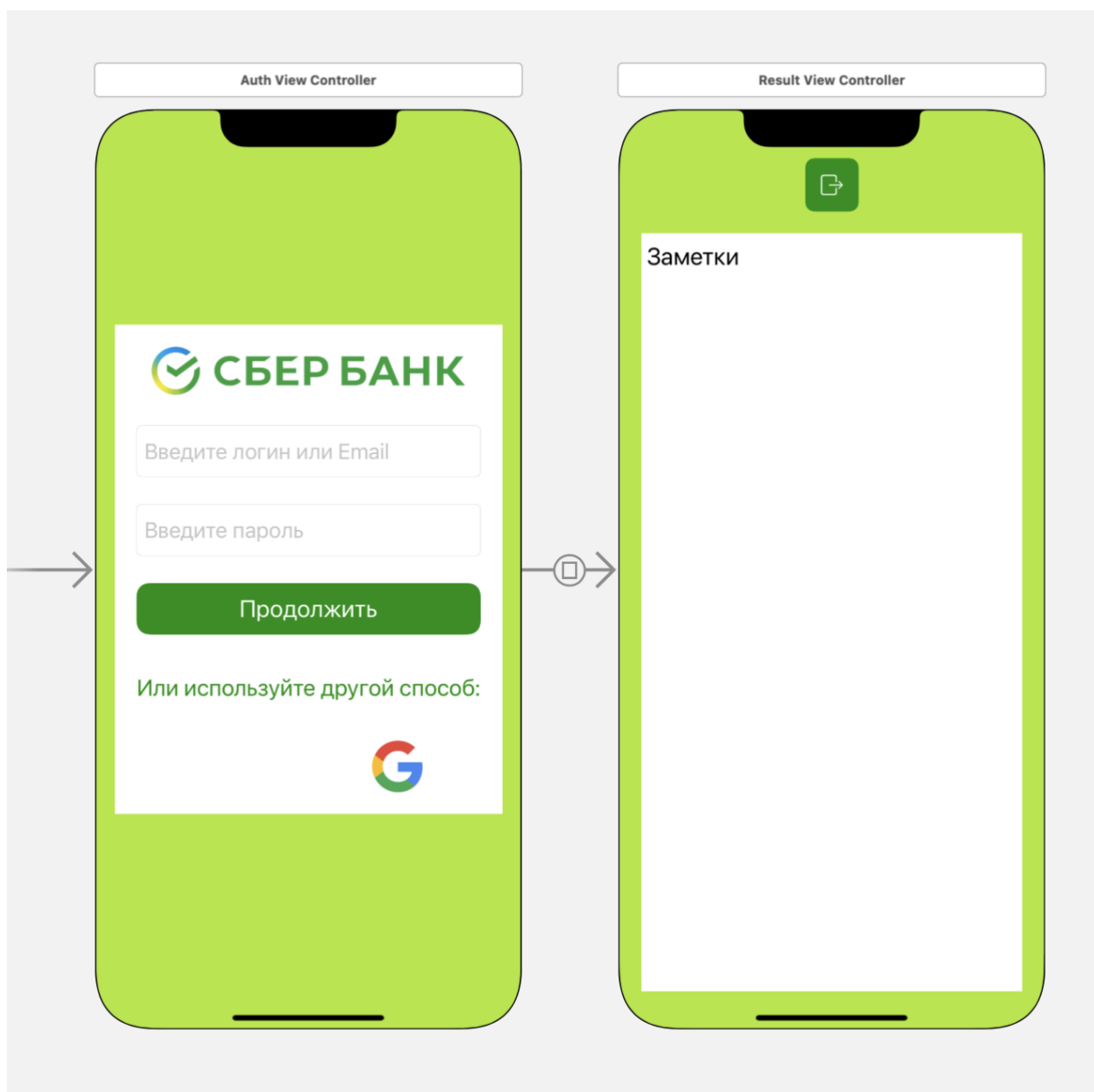


Рисунок 2 – Пользовательский интерфейс в форме входа в систему

Это интерфейс будущего приложения для ведения заметок имеет форму входа в систему, где пользователи могут ввести логин или адрес электронной почты и пароль, а также могут войти с помощью аккаунта Google. Два экрана приложения уже связаны друг с другом и готовы к использованию. На этом этапе нажатие кнопки "Продолжить" позволяет зарегистрироваться в случае отсутствия конкретного пользователя в базе или проверять предоставленные

пользователем учетные данные, в случае успешной проверки переходит к другому экрану и отображает заметки. Нажатие кнопки с изображением выхода, которая расположена вверху, возвращает пользователя к форме входа в систему.

4.2 Интеграция с Firebase

Разработка пользовательской аутентификации обычно требует больше времени, так как необходимо управлять взаимодействием на стороне сервера и тратить время на тестирование.

Для того, чтобы облегчить жизнь разработчиков, Firebase предоставляет систему аутентификации, которую можно интегрировать в приложение.

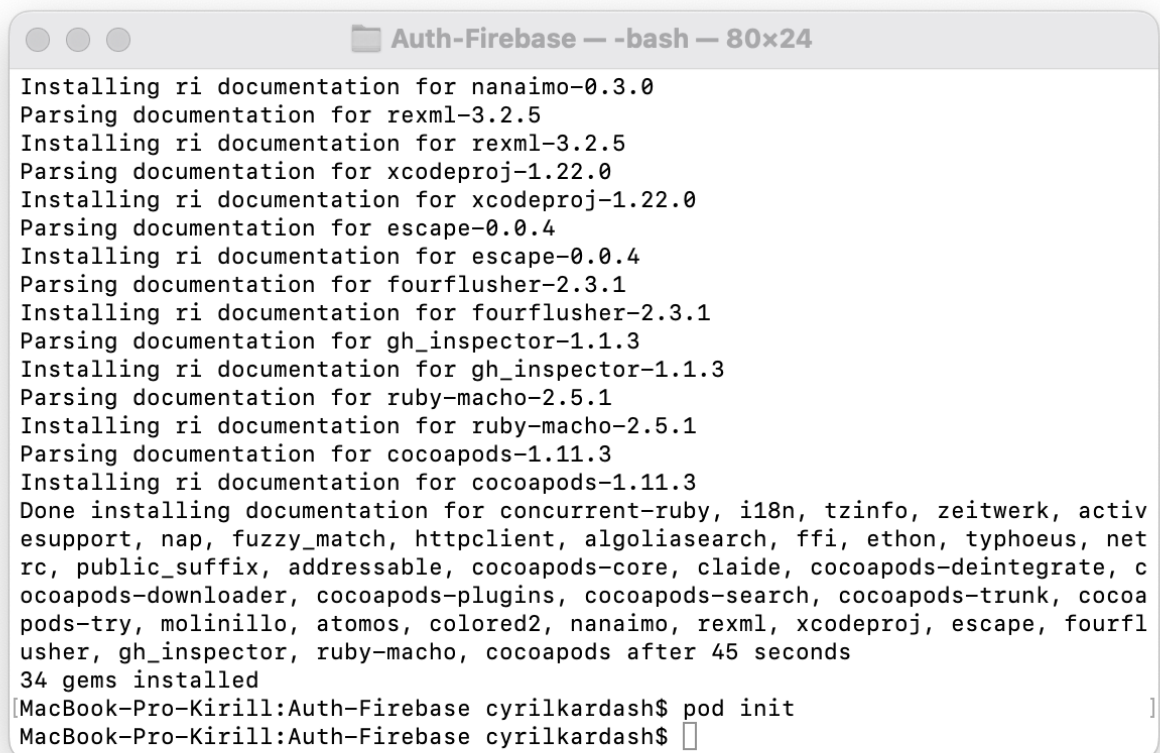
Firebase — это серверный поставщик услуг с интеграцией SDK и готовых к использованию элементов пользовательского интерфейса для аутентификации пользователей. Firebase поддерживает аутентификацию с использованием электронной почты, номеров телефонов и даже вход в социальные сети с помощью Facebook, Twitter и Google.

После входа в Firebase на сайте <https://console.firebase.google.com/> и создания проекта открывается панель инструментов, где можно выбрать нужные платформы для интеграции в разрабатываемое приложение. После выбора платформы iOS необходимо выполнить следующие действия добавления Firebase в приложение:

- 1) Зарегистрировать разрабатываемое приложение;
- 2) Загрузить файл `GoogleService-info.plist` в проект Xcode;
- 3) Добавить SDK Firebase в проект с помощью CocoaPods. Для этого нужно открыть командную строку и создать Podfile в текущем проекте путем команды `$pod init`, затем добавить ниже зависимости в Podfile, закрыть файл и установить с помощью команды `$pod install`. Эти действия показаны на рисунках 3, 4 и 5. Podfile — это спецификация, которая описывает зависимости для одного или нескольких проектов

Xcode. Файл должен просто называться Podfile. После установки модулей появится файл с расширением xcworkspace, и необходимо работать с ним вместо файла с расширением xcproj. Файл с расширением xcworkspace выглядит так на рисунке 6;

- 4) В проекте открыть файл AppDelegate.swift, который в приложении В, и импортировать SDK Firebase с помощью выражения `import Firebase`, чтобы использовать API Firebase.



```
Auth-Firebase — -bash — 80x24
Installing ri documentation for nanaimo-0.3.0
Parsing documentation for rexml-3.2.5
Installing ri documentation for rexml-3.2.5
Parsing documentation for xcproj-1.22.0
Installing ri documentation for xcproj-1.22.0
Parsing documentation for escape-0.0.4
Installing ri documentation for escape-0.0.4
Parsing documentation for fourflusher-2.3.1
Installing ri documentation for fourflusher-2.3.1
Parsing documentation for gh_inspector-1.1.3
Installing ri documentation for gh_inspector-1.1.3
Parsing documentation for ruby-macho-2.5.1
Installing ri documentation for ruby-macho-2.5.1
Parsing documentation for cocoapods-1.11.3
Installing ri documentation for cocoapods-1.11.3
Done installing documentation for concurrent-ruby, i18n, tzinfo, zeitwerk, activ
esupport, nap, fuzzy_match, httpclient, algoliasearch, ffi, ethon, typhoeus, net
rc, public_suffix, addressable, cocoapods-core, claide, cocoapods-deintegrate, c
ocoapods-downloader, cocoapods-plugins, cocoapods-search, cocoapods-trunk, cocoa
pods-try, molinillo, atomos, colored2, nanaimo, rexml, xcproj, escape, fourfl
usher, gh_inspector, ruby-macho, cocoapods after 45 seconds
34 gems installed
[MacBook-Pro-Kirill:Auth-Firebase cyrilkardash$ pod init
MacBook-Pro-Kirill:Auth-Firebase cyrilkardash$ ]
```

Рисунок 3 – Создание файла Podfile

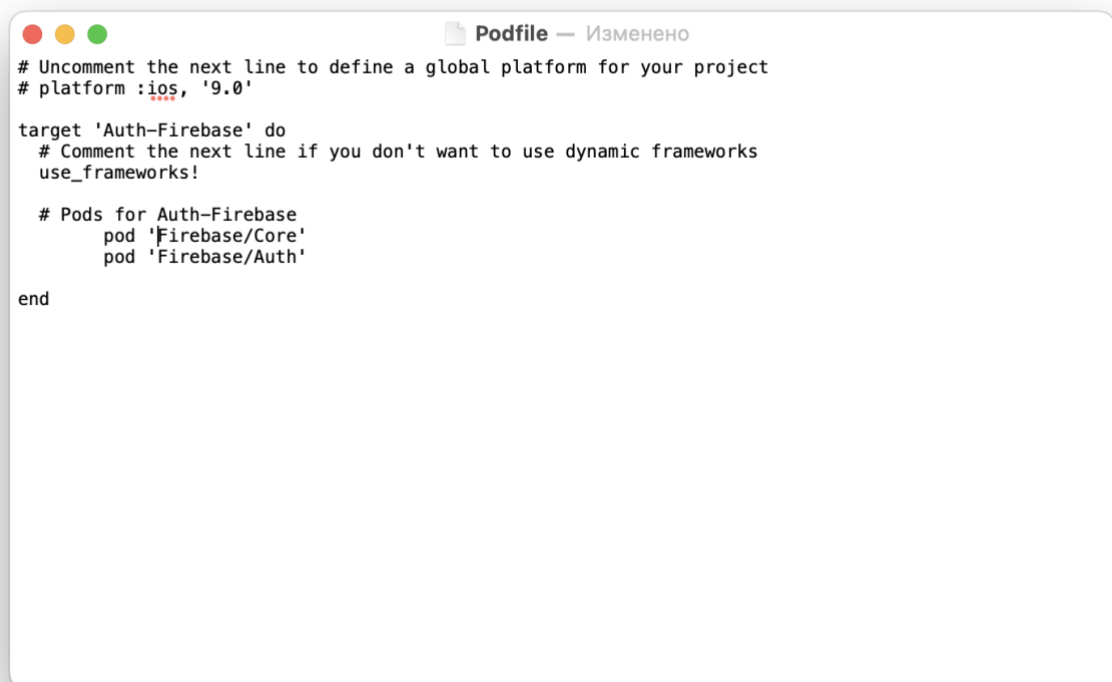


Рисунок 4 – Добавление зависимостей в файл Podfile

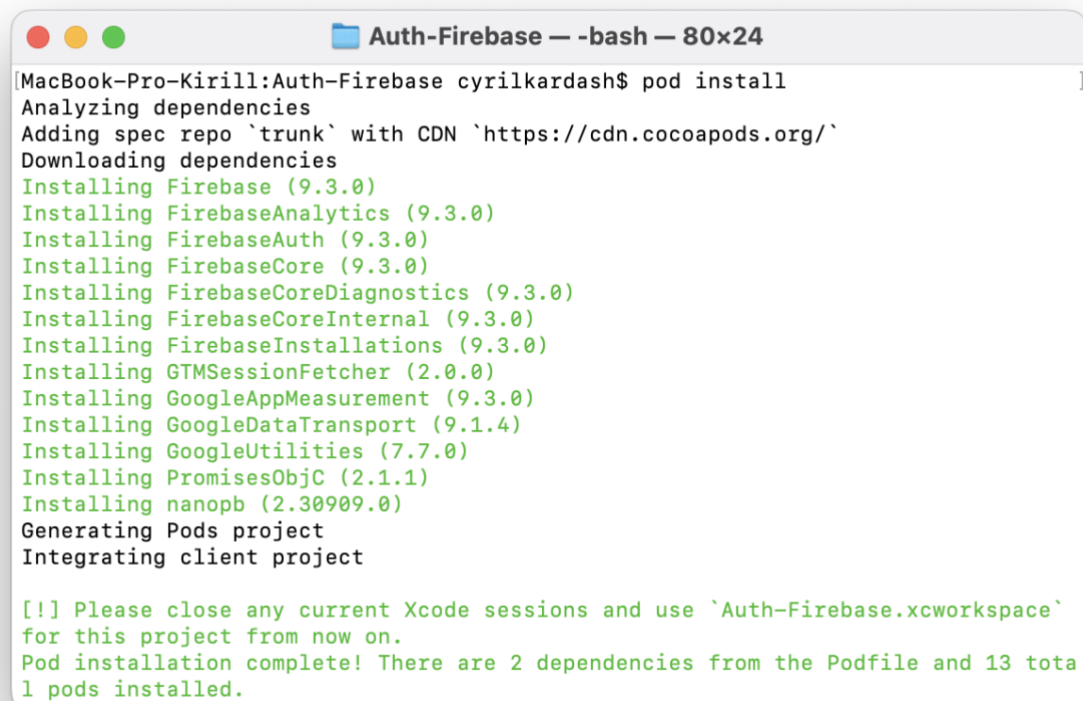


Рисунок 5 – Установка модулей

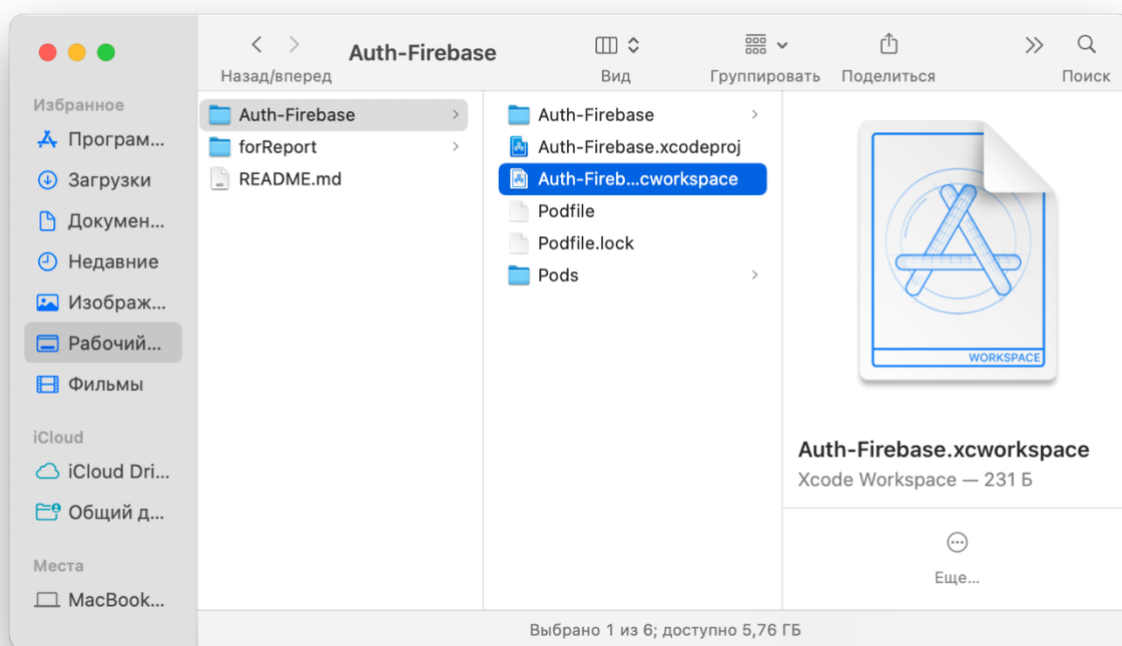


Рисунок 6 – Файл с расширением xcworkspace

4.3 Реализация аутентификации

Для реализации функции регистрации с Email и паролем следует выбрать «Email/Password», как показано на рисунке 7.

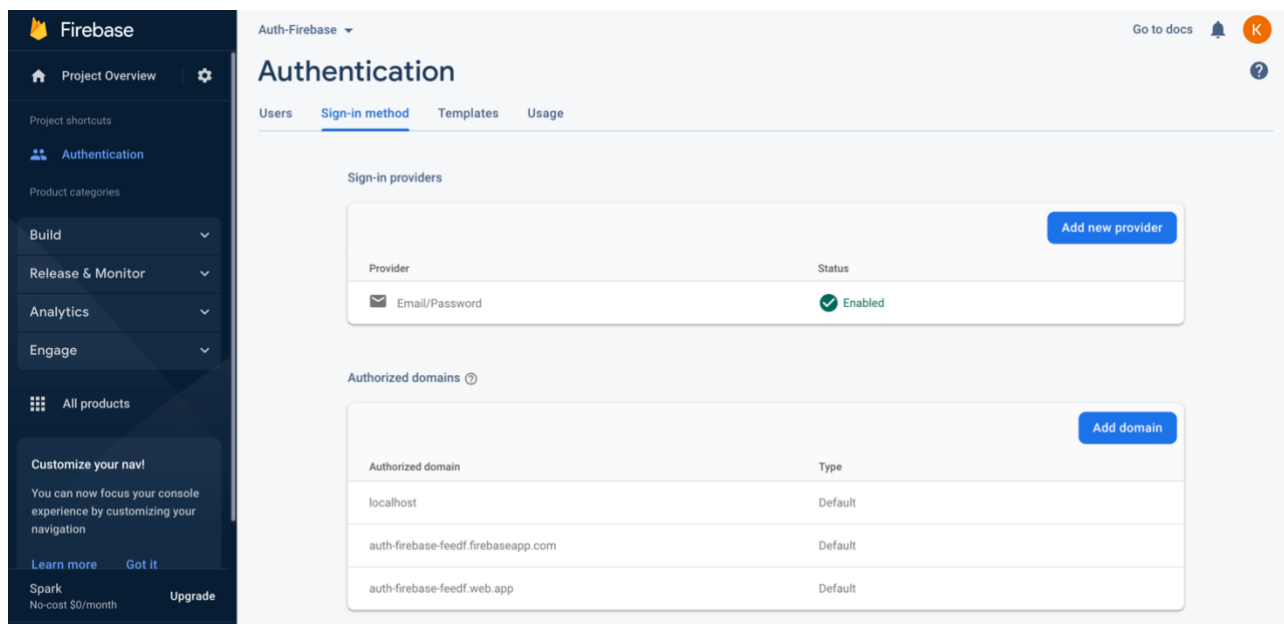


Рисунок 7 – Выбор опции Email/Password

В файле `AuthViewController.swift` в функции `showCreateAccount` был добавлен код для создания нового пользователя, он представлен в приложении А. В условии `if` идет проверка, заполнил ли пользователь оба текстовых поля, если пользователь введет некорректные данные, появится предупреждение, как представлено на рисунке 8.

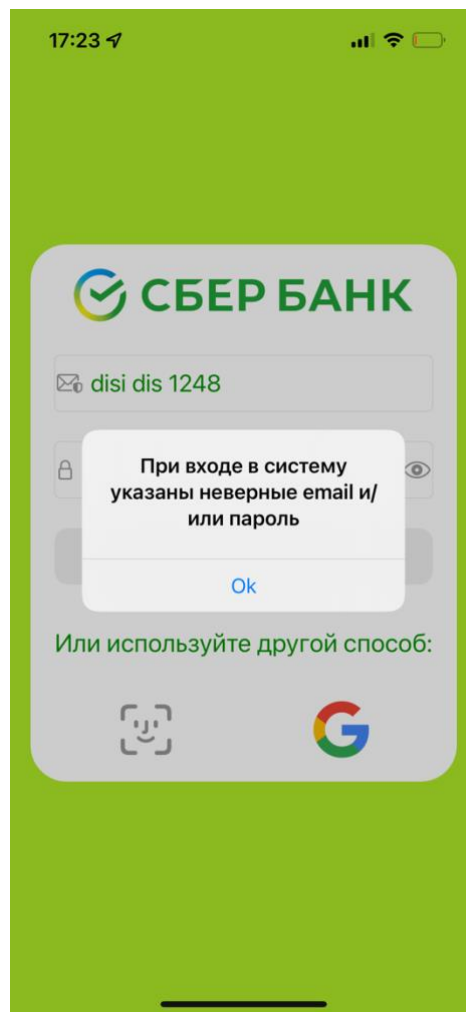









Рисунок 8 – Предупреждение после ввода некорректных данных

Нельзя зарегистрироваться снова с той же учетной записью, иначе Firebase вернет сообщение об ошибке, соответственно, необходимо реализовать код для входа в систему. Данный код был реализован в функции `textFieldShouldReturn`, которая представлена в приложении А. Если при входе возникнет какая-либо ошибка, данная функция вызовет обработчик ошибок, а если не найден, она вызывает метод `showCreateAccount` для регистрации.

alexkardash@mail.ru		Jul 23, 2022	Jul 23, 2022	8RNDQrGtMFVvN4ivGdTVco0yKD...
alexkard@mail.ru		Jul 23, 2022	Jul 23, 2022	VVW9AFzvAPRaN3W9PvpvYbZ...
kraft@mail.ru		Jul 23, 2022	Jul 23, 2022	uCWXm2DKepXHn0eU84PYazJbK...
okfh@mail.com		Jul 23, 2022	Jul 23, 2022	nWthXvfW4zdh202V0jqu0uA894I1
kgfyhb@mail.ru		Jul 23, 2022	Jul 23, 2022	HR8WYZldikUVJKur9rZHpK3d1vU2
fukhf@mail.ru		Jul 22, 2022	Jul 22, 2022	PwMkfGPZvLSVmPRLTL4THLUS8...
gmail@gmail.com		Jul 22, 2022	Jul 22, 2022	WrcY4YVQcDbRNdk2eKrbSltYgDB3

Rows per page: 50 1 – 28 of 28

Рисунок 9 – Список адресов эл. почты пользователей в Firebase

Если зайти в панель инструментов Firebase, можно увидеть список адресов электронной почты зарегистрировавшихся пользователей в разделе «Аутентификация», это показано выше на рисунке 9.

В файле `ResultViewController.swift`, отвечающем за отображение второго экрана после входа в систему, содержится функция `pressedBtnExit` с событием для выхода из системы.

4.4 Реализация входа с помощью Google

Для внедрения входа с помощью Google в разрабатываемое приложение необходимо выбрать флажок Google на вкладке «Метод входа» в разделе «Аутентификация», как показано на рисунке 10, и добавить Google SDK в проект путем CocoaPods, для этого достаточно вписать в существующий Podfile зависимость pod 'GoogleSignIn' и обновить этот файл. Затем в проекте инициализировать вход через Google в файле `AppDelegate.swift` с помощью `import GoogleSignIn`.

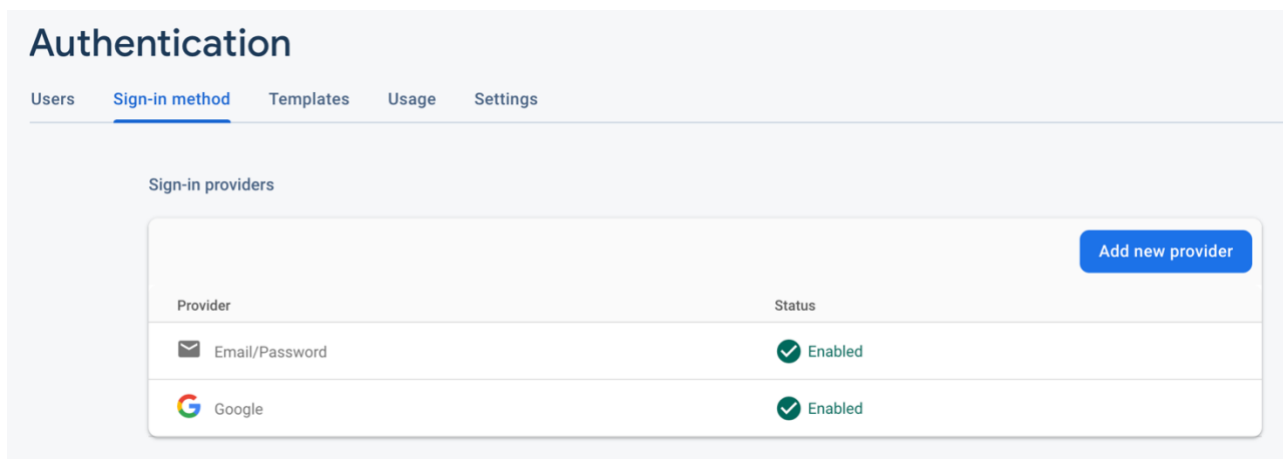


Рисунок 10 – Добавление опции Google

Событие кнопки в виде логотипа Google связана с файлом `AuthViewController.swift`, и туда же добавлена функция `pressedBtnGoogle`, отвечающая за событие нажатия данной кнопки. Данная функция приведена в приложении А. Когда пользователь собирается войти в свою учетную запись Google, всплывающее окно для продолжения процесса входа будет выглядеть так, как показано на рисунке 11.

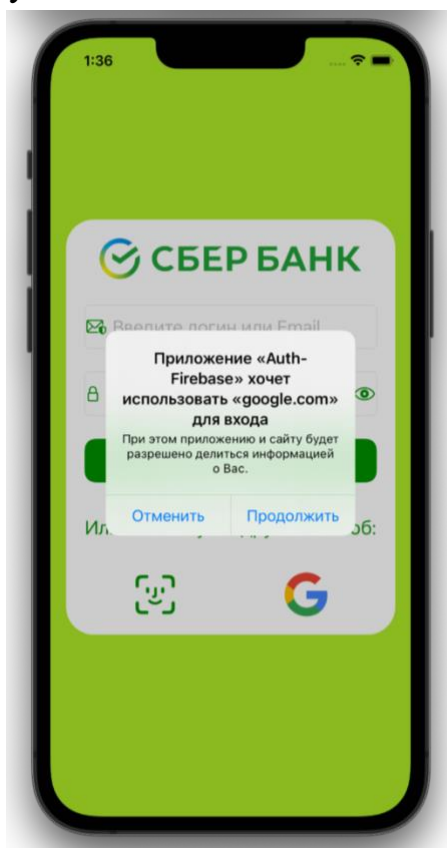


Рисунок 11 – Всплывающее окно после нажатия кнопки

Когда пользователь нажимает кнопку «Продолжить», ему будет предложено ввести телефон или адрес эл. Почты для входа в Google, как показано ниже.

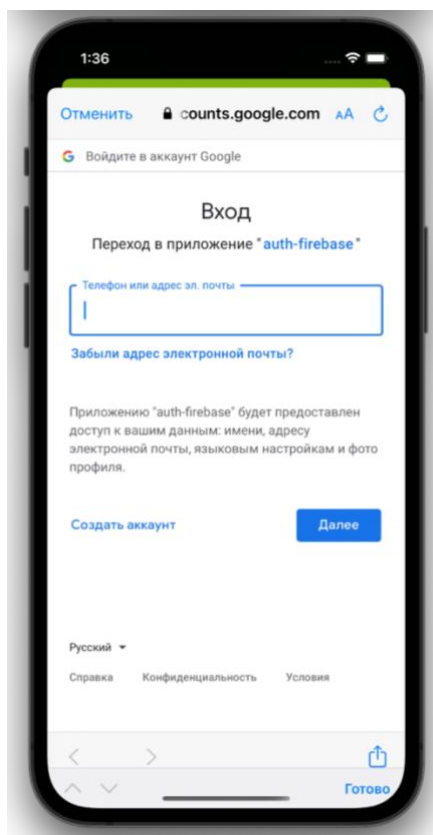


Рисунок 12 – Окно для ввода после нажатия кнопки

Аутентификация с помощью входа в Google успешно реализована. В файле `ResultViewController.swift` добавлен код для выхода из системы. Он приведен в приложении Б.

4.5 Реализация входа с помощью Touch ID и Face ID

Стоит отметить, что Face ID требует тестирования на физическом устройстве. Touch ID теперь можно эмулировать в Xcode в симуляторе. Вы можете протестировать биометрический идентификатор на любом устройстве с чипом A7 или новее и аппаратным обеспечением Face ID/Touch ID [6, 7].

Реализация биометрической аутентификации так же проста, как импорт фреймворка `LocalAuthentication` и вызов пары методов.

Новым в iOS 11 является поддержка Face ID. LocalAuthentication добавляет пару новых вещей: FaceIDUsageDescription и LABiometryType, чтобы определить, поддерживает ли устройство Face ID или Touch ID.

Чтобы добавить FaceIDUsageDescription, в навигаторе проектов Xcode следует выбрать проект и перейти на вкладку «Info», навести указатель мыши на правый край одного из ключей и нажать на значок +, затем ввести «Privacy». И во всплывающем списке выберите «Privacy – Face ID Usage Description» в качестве ключа. Это действие приведено на рисунке 13.

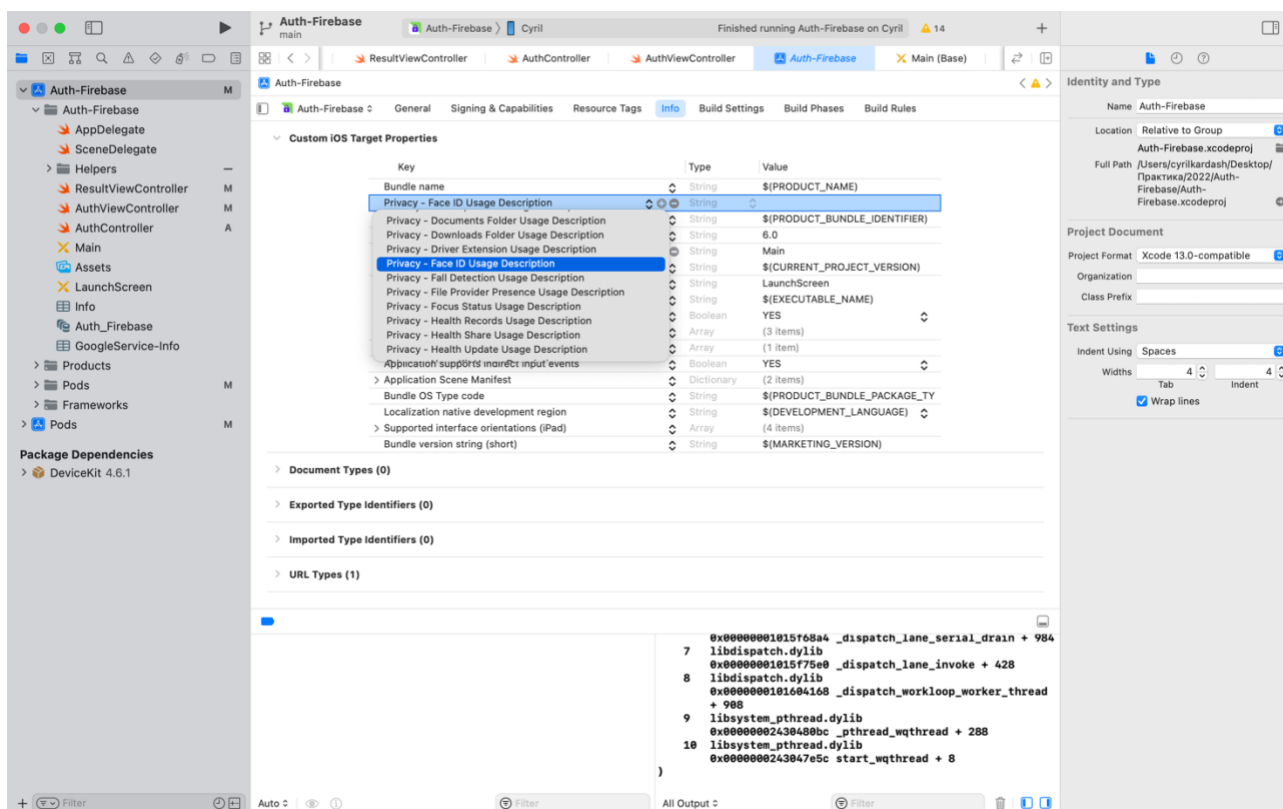


Рисунок 13 – Включение Face ID Usage Description

В файле AuthViewController.swift в методе pressedBtnBiometry понадобится импортировать с помощью `import LocalAuthentication`, создать экземпляр класса `LAContext` и вызвать функции, чтобы определить, доступен ли биометрический идентификатор на устройстве пользователя или в симуляторе. В приложении А представлен реализованный метод `pressedBtnBiometry`.

При запуске приложения отображение кнопки в виде изображения отпечатка или лица будет зависеть от модели устройства. Если запустить

приложение в iPhone 8, изображение будет выглядеть как отпечаток, как показано на рисунке 14. А на более современных устройствах изображение выглядит так, как на рисунках 8 и 11.

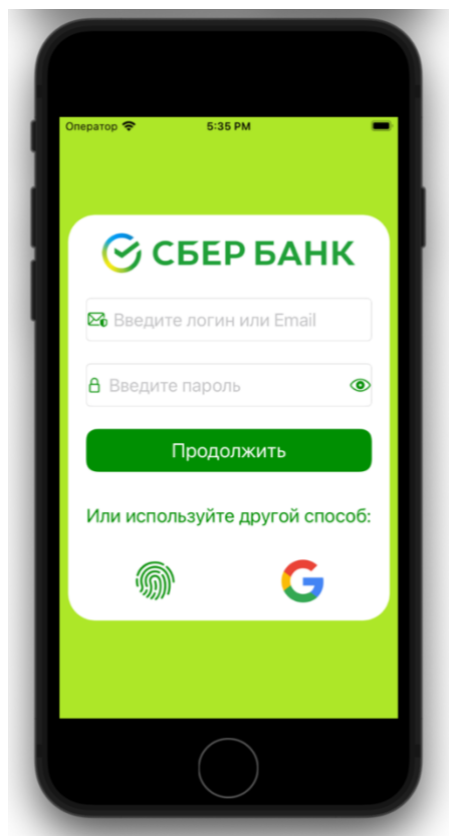


Рисунок 14 – Изображение кнопки в зависимости от модели устройства

Компания Apple предлагает еще один уровень защиты с Face ID и Touch ID – использовать Связку ключей, встроенную прямо в iOS, чтобы обеспечить безопасность их данных. Одним из важнейших элементов безопасности для разработчиков Apple является iOS Связка ключей, представляющая собой специализированную базу данных для хранения метаданных и конфиденциальной информации. Использование Связки ключей — лучший способ хранения небольших фрагментов данных, важных для приложения, таких как секреты и пароли.

Почему рекомендуется использовать Связку ключей, а не более простые решения? Будет ли достаточно хранения пароля пользователя в кодировке base-

64? Злоумышленнику несложно восстановить сохраненный таким образом пароль. Безопасность сложна, и пытаться реализовать собственное решение — не лучшая идея, даже если приложение предназначено не для финансовых операций.

Напрямую взаимодействовать со Связкой ключей сложно, особенно на Swift. Разработчики для взаимодействия должны использовать библиотеку, которая в основном написана на C. Однако можно избежать использования этих низкоуровневых API, для этого достаточно импортировать оболочку Swift из кода `GenericKeychain`[8] от Apple. Файл `KeychainPasswordItem.swift` предоставляет простой интерфейс Swift для Связки ключей и уже включен в проект.

Для взаимодействия со Связкой ключей необходимо объявить константу с именем проекта, это служебное имя, которое будет использоваться для идентификации приложения в Связке ключей, и создать метод `signIn`, код которого представлен в приложении А. Данный метод надежно сохраняет данные для входа пользователя в Связке ключей. Это сохранение в Связке ключей выглядит так, как на рисунке 15. Он создает `KeychainPasswordItem` с определяемым именем службы вместе с уникальным идентификатором. В проекте электронная почта пользователя или логин используется в качестве идентификатора для Связки ключей.

Однако эту реализацию нельзя считать завершенной, так как сохранение пароля пользователя без кодировки — не лучшая практика. Например, если злоумышленник скомпрометировал Связку ключей, он может получить доступ к паролям пользователей в виде простого текста. Лучшим решением является хранение хэша, созданного на основе идентификатора пользователя. Для этого достаточно импортировать библиотеку `CryptoSwift`.

`CryptoSwift` — одна из самых популярных коллекций множества стандартных криптографических алгоритмов, написанных на

Swift. Использование популярной библиотеки для обеспечения безопасности позволяет разработчикам сэкономить время на реализацию стандартных функций хеширования.

Фреймворк Apple CommonCrypto предоставляет множество полезных функций хеширования, но взаимодействовать с ним на Swift непросто, в связи с этим было решено выбрать библиотеку CryptoSwift.

Метод `signIn` принимает адрес электронной почты или логин и пароль, возвращает хешированную строку. В приведенном ранее примере злоумышленник, скомпрометировавший Связку ключей, находит этот хэш. Он может создать таблицу часто используемых паролей и их хэшей для сравнения с данным хешем. Если разработчики хешируете ввод данных без добавления соли, а пароль существует в хэш-таблице у злоумышленников, пароль будет скомпрометирован.

Включение соли увеличивает сложность атаки. Кроме того, в методе было реализовано объединение адреса электронной почты или логина пользователя и пароля с солью, чтобы создать хеш, который сложно будет взломать.

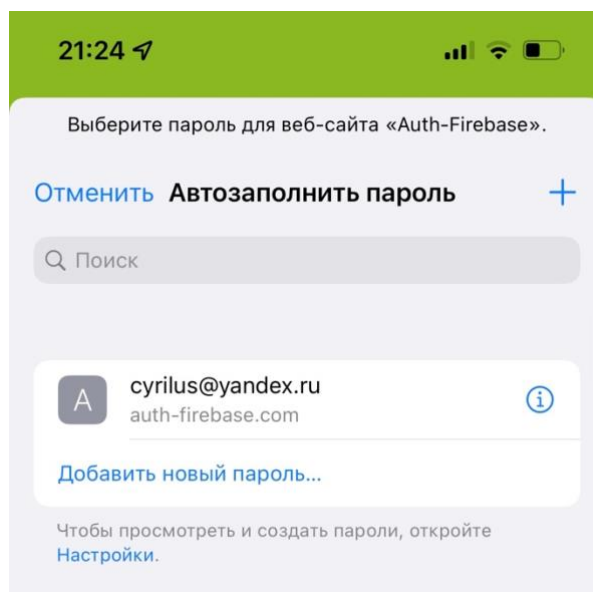


Рисунок 15 – Сохранение пользовательских данных в Связке ключей

ЗАКЛЮЧЕНИЕ

Цель практики, которая заключалась в разработке аутентификации в мобильном приложении под iOS, была достигнута. В процессе прохождения были изучены основные практики обеспечения безопасности iOS-приложений и протестировано разрабатываемое приложение.

Немаловажным является тот факт, что в процессе прохождения практики были получены практические навыки по разработке аутентификации Firebase с использованием электронной почты, Google и Touch ID или Face ID в зависимости модели устройства на языке программирования Swift. Важно отметить, что один из наиболее важных аспектов разработки программного обеспечения является безопасностью приложений и настоятельно рекомендуется ознакомиться с документацией по основным методам обеспечения безопасности iOS, включая доступ к Связке ключей и хеширование. Можно сделать выводы, что все задачи, поставленные в ходе производственной практики, успешно выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Биометрия [Электронный ресурс]. – Режим доступа:
<https://lib.itsec.ru/articles2/biometric/>
2. Плюсы и минусы биометрической идентификации [Электронный ресурс]. – Режим доступа: <https://lib.itsec.ru/articles2/biometric/plyusy-i-minusy-biometricheskoy-identifikatsii>
3. Сбербанк России [Электронный ресурс]. – Режим доступа:
https://ru.wikipedia.org/wiki/Сбербанк_России
4. Google Firebase [Электронный ресурс]. – Режим доступа:
https://codernet.ru/articles/drugoe/google_firebase_что_это_за_servis_i_dlya_chego_ego_mozhno_ispolzovat/
5. Основные практики обеспечения безопасности iOS-приложений [Электронный ресурс]. – Режим доступа:
<https://habr.com/ru/company/redmadrobot/blog/349272/>
6. Сведения о технологии безопасности Touch ID [Электронный ресурс]. – Режим доступа: <https://support.apple.com/ru-ru/HT204587>
7. Сведения о технологии Face ID [Электронный ресурс]. – Режим доступа:
<https://support.apple.com/ru-ru/HT208108>
8. GenericKeychain [Электронный ресурс]. – Режим доступа:
https://developer.apple.com/library/archive/samplecode/GenericKeychain/Introduction/Intro.html#//apple_ref/doc/uid/DTS40007797-Intro-DontLinkElementID_2

ПРИЛОЖЕНИЕ А

AuthViewController.swift

```
import UIKit
import FirebaseAuth
import GoogleSignIn
import Firebase
import LocalAuthentication
import CryptoSwift

class AuthViewController: UIViewController, UITextFieldDelegate {

    @IBOutlet weak var passwordField: UITextField!
    @IBOutlet weak var loginField: UITextField!
    @IBOutlet weak var btnBiometry: UIButton!

    var btnEye = UIButton(type: .custom)
    var btnMail = UIButton(type: .custom)
    var btnLock = UIButton(type: .custom)

    override func viewDidLoad() {
        super.viewDidLoad()
        setImageForBtnBiometry()
        loginField.delegate = self
        passwordField.delegate = self

        btnEye.setImage(UIImage(systemName: "eye"), for: .normal)
        btnEye.tintColor = UIColor(named: "colorForBtns")
        btnEye.addTarget(self, action: #selector(self.refresh),
for: .touchUpInside)
        passwordField.rightView = btnEye
        passwordField.rightViewMode = .always

        btnLock.setImage(UIImage(systemName: "lock"), for:
.normal)
        btnLock.tintColor = UIColor(named: "colorForBtns")
        passwordField.leftView = btnLock
        passwordField.leftViewMode = .always

        btnMail.setImage(UIImage(systemName:
"envelope.badge.shield.half.filled"), for: .normal)
        btnMail.tintColor = UIColor(named: "colorForBtns")
        loginField.leftView = btnMail
        loginField.leftViewMode = .always
    }
}
```



```

func setImageForBtnBiometry() -> Void {
    let context = LAContext()
    let _ =
context.canEvaluatePolicy(.deviceOwnerAuthenticationWithBiometrics
, error: nil)
    switch context.biometryType {
    case .faceID:
        btnBiometry.configuration?.background.image =
UIImage(systemName: "faceid")
        btnBiometry.configuration?.background.imageContentMode
= .scaleAspectFit
    case .touchID:
        btnBiometry.configuration?.background.image =
UIImage(systemName: "touchid")
        btnBiometry.configuration?.background.imageContentMode
= .scaleAspectFit
    default:
        break
    }
}

@IBAction func refresh(_ sender: UIButton) {
    if passwordField.isSecureTextEntry {
        btnEye.setImage(UIImage(systemName: "eye.slash"), for:
.normal)
        passwordField.isSecureTextEntry = false
    } else {
        btnEye.setImage(UIImage(systemName: "eye"), for:
.normal)
        passwordField.isSecureTextEntry = true
    }
}

func showAlert(message: String) {
    let alert = UIAlertController(title: message, message: "",
preferredStyle: .alert)
    alert.addAction(UIAlertAction(title: "Ok", style:
.default, handler: nil))
    self.present(alert, animated: true)
}

func passwordHash(from email: String, password: String) ->
String {
    let salt = "x567vV8bGgqfjkhggCoyXFQj+(o.nUNQhV67ND"
    return "\(password).\(email).\salt)".sha256()
}

func signIn(email: String, password: String) throws {
    let serviceName = "Auth-Firebase"

```

```

        let finalHash = passwordHash(from: email, password:
password)
        try KeychainPasswordItem(service: serviceName, account:
email).savePassword(finalHash)
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.endEditing(true)
        if passwordField.isTouchInside, let login =
loginField.text, !loginField.text!.isEmpty,
            let password = passwordField.text,
!passwordField.text!.isEmpty {
            do {
                try self.signIn(email: login, password: password)
            } catch {
                print("Error signing in:
\\(error.localizedDescription)")
            }
            Auth.auth().signIn(withEmail: login, password:
password, completion: {
                result, error in
                guard error == nil else {
                    if let err = AuthErrorCode.Code(rawValue:
error!._code) {
                        switch err {
                            case .userNotFound:
                                self.showCreateAccount(email: login,
password: password)
                            case .invalidEmail:
                                self.showAlert(message: "При входе в
систему указаны неверные email и/или пароль")
                            case .wrongPassword:
                                self.showAlert(message: "При входе в
систему указаны неверные email и/или пароль")
                            default:
                                self.showAlert(message: "Unexpected
error: \\(err.rawValue)")
                        }
                    }
                }
                return
            }
            self.performSegue(withIdentifier: "goToNotes",
sender: self)
            self.loginField.text?.removeAll()
            self.passwordField.text?.removeAll()
        })
    }
    else {
        print("Отсутствуют необходимые данные для входа")
    }
}

```

```

    }
    return true
}

func showCreateAccount(email: String, password: String) {
    let alert = UIAlertController(title: "Создать аккаунт",
message: "Хотели ли бы Вы создать аккаунт?", preferredStyle: .alert)
    alert.addAction(UIAlertAction(title: "Создать", style:
.default, handler: { _ in

        Auth.auth().createUser(withEmail: email, password:
password, completion: {
            result, error in
                guard error == nil else {
                    self.showAlert(message:
(error?.localizedDescription)!)
                    print("Не удалось создать аккаунт")
                    return
                }
                self.performSegue(withIdentifier: "goToNotes",
sender: self)

                self.loginField.text?.removeAll()
                self.passwordField.text?.removeAll()
            })
        })))
    alert.addAction(UIAlertAction(title: "Отмена", style:
.cancel, handler: { _ in
        })))
    present(alert, animated: true)
}

@IBAction func pressedBtnEnter(_ sender: UIButton) {
    textFieldShouldReturn(passwordField)
}

func textFieldDidEndEditing(_ textField: UITextField) {
    if textField.text!.isEmpty {
        textField.layer.borderColor = UIColor.red.cgColor
        textField.layer.borderWidth = 0.5
    }
    else {
        textField.layer.borderColor = UIColor.gray.cgColor
        textField.layer.borderWidth = 0.1
    }
}

@IBAction func pressedBtnBiometry(_ sender: UIButton) {
    let context = LAContext()

```

```

        if
context.canEvaluatePolicy(.deviceOwnerAuthenticationWithBiometrics
, error: nil) {

context.evaluatePolicy(.deviceOwnerAuthenticationWithBiometrics,
localizedReason: "Please authenticate to proceed.") {
    (succes, error) in
        if succes {
            DispatchQueue.main.async {
                self.performSegue(withIdentifier:
"goToNotes", sender: self)
            }
        }
        else {
            guard let error = error else {
                return
            }
            print(error.localizedDescription)
        }
    }
}

@IBAction func pressedBtnGoogle(_ sender: UIButton) {
    guard let clientId = FirebaseApp.app()?.options.clientID
else {
        return
    }
    let signInConfig = GIDConfiguration.init(clientID:
clientId)
    GIDSignIn.sharedInstance.signIn(with: signInConfig,
presenting: self) { user, error in
        guard user != nil else {
            print(error?.localizedDescription as Any)
            return
        }
        self.performSegue(withIdentifier: "goToNotes", sender:
self)
    }
}
}

```

ПРИЛОЖЕНИЕ Б

ResultViewController.swift

```
import UIKit
import FirebaseAuth
import GoogleSignIn

class ResultViewController: UIViewController, UITextViewDelegate {

    @IBOutlet weak var textView: UITextView!

    override func viewDidLoad() {
        super.viewDidLoad()
        textView.layer.borderColor = UIColor(named:
"colorForBtns")?.cgColor
        textView.layer.borderWidth = 1
    }

    @IBAction func pressedBtnExit(_ sender: UIButton) {
        do {
            try Auth.auth().signOut()
            self.dismiss(animated: true, completion: nil)
            print("log out")
        } catch let signOutError as NSError {
            self.showAlert(message: signOutError.localizedDescription)
        }
    }

    func showAlert(message: String) {
        let alert = UIAlertController(title: message, message: "",
preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "Ok", style:
.default, handler: nil))
        self.present(alert, animated: true)
    }
}
```

ПРИЛОЖЕНИЕ В

AppDelegate.swift

```
import UIKit
import CoreData
import Firebase
import GoogleSignIn

@main
class AppDelegate: UIResponder, UIApplicationDelegate {

    public var signInConfig: GIDConfiguration?

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

        FirebaseApp.configure()

        if let clientId = FirebaseApp.app()?.options.clientID {
            signInConfig = GIDConfiguration.init(clientID:
clientId)
        }

        return true
    }

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

        handled = GIDSignIn.sharedInstance.handle(url)
        if handled {
            return true
        }
        return false
    }

    // MARK: UISceneSession Lifecycle

    func application(_ application: UIApplication,
configurationForConnecting connectingSceneSession: UISceneSession,
options: UIScene.ConnectionOptions) -> UISceneConfiguration {
        // Called when a new scene session is being created.
```

```

        // Use this method to select a configuration to create the
new scene with.
        return UISceneConfiguration(name: "Default Configuration",
sessionRole: connectingSceneSession.role)
    }

    func application(_ application: UIApplication,
didDiscardSceneSessions sceneSessions: Set<UISceneSession>) {
        // Called when the user discards a scene session.
        // If any sessions were discarded while the application
was not running, this will be called shortly after
application:didFinishLaunchingWithOptions.
        // Use this method to release any resources that were
specific to the discarded scenes, as they will not return.
    }

    // MARK: - Core Data stack

    lazy var persistentContainer: NSPersistentCloudKitContainer =
{
    /*
        The persistent container for the application. This
implementation
        creates and returns a container, having loaded the store
for the
        application to it. This property is optional since there
are legitimate
        error conditions that could cause the creation of the
store to fail.
    */
    let container = NSPersistentCloudKitContainer(name:
"Auth_Firebase")
    container.loadPersistentStores(completionHandler: {
(storeDescription, error) in
        if let error = error as NSError? {
            // Replace this implementation with code to handle
the error appropriately.
            // fatalError() causes the application to generate
a crash log and terminate. You should not use this function in a
shipping application, although it may be useful during
development.

            /*
            Typical reasons for an error here include:
            * The parent directory does not exist, cannot be
created, or disallows writing.
            * The persistent store is not accessible, due to
permissions or data protection when the device is locked.
            * The device is out of space.

```

```

        * The store could not be migrated to the current
model version.
        Check the error message to determine what the
actual problem was.
        */
        fatalError("Unresolved error \(error),
\ (error.userInfo)")
    }
    })
    return container
}()

// MARK: - Core Data Saving support

func saveContext () {
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
        } catch {
            // Replace this implementation with code to handle
the error appropriately.
            // fatalError() causes the application to generate
a crash log and terminate. You should not use this function in a
shipping application, although it may be useful during
development.

            let nserror = error as NSError
            fatalError("Unresolved error \(nserror),
\ (nserror.userInfo)")
        }
    }
}
}

```