Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерного проектирования
Кафедра проектирования информационно-компьютерных систем
Дисциплина «Программное обеспечение мобильных систем»

«К ЗАЩИТЕ ДОПУСТИТЬ»

Руководитель курсового проекта
ассистент каф. ПИКС
_____ К.С. Крез

___.____.2023

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему:
«ПРОГРАММНОЕ СРЕДСТВО ПО ОБЕСПЕЧЕНИЮ АВИАБАЗЫ»

БГУИР КР 1-39 03 02 010 ПЗ

Выполнил студент группы 113802
МАКЕЙ Павел Игоревич

_____
(подпись студента)

Курсовой проект представлен на проверку ___.____.2023

_____
(подпись студента)

Минск 2023

РЕФЕРАТ

БГУИР КР 1-39 03 02 010 ПЗ

Макей, П.И. Программное средство по обеспечению авиабазы: пояснительная записка к курсовой работе/ П.И. Макей. – Минск: БГУИР, 2023. – 52 с.

Пояснительная записка 52 с., 15 рис., 29 источника, 3 приложений.

ПРОГРАММНОЕ СРЕДСТВО, УЧЕТ СНАРЯЖЕНИЯ НА СКЛАДЕ, УЧЕТ ПИЛОТОВ, УЧЕТ САМОЛЕТОВ, УСЛОВИЯ ЭКСПЛУАТАЦИИ, ПРОЕКТНАЯ ЧАСТЬ.

Цель проектирования: разработка программного средства по обеспечению авиабазы, предназначенного для упрощения и автоматизации процесса учета снаряжения на складе.

Методология проведения работы: в процессе решения поставленных задач использованы принципы системного подхода, был проведен анализ потребностей служащих авиабазы, был реализован необходимый функционал.

Результаты работы: выполнен анализ потребностей и требований служащих авиабазы, рассмотрено общетехническое обоснование разработки программного средства, осуществлена проверка исключительных ситуаций, уделено внимание вопросам корректной работы программы, разработана графическая часть проекта.

Программное средство обеспечивает учет данных о самолетах, пилотах и снаряжении.

Область применения результатов: могут быть использованы при проектировании аналогичных программных средств для смежных областей с повышенной надежностью работы.

ВВЕДЕНИЕ

Как известно, в авиационной индустрии ключевым элементом эффективной работы является учет и контроль запасов. Отслеживание наличия и распределение различных компонентов, запасных частей и оборудования является неотъемлемой частью обеспечения работы авиабазы. В связи с этим, было разработано специализированное приложение, предназначенное для автоматизации учета и управления складскими операциями в авиабазе.

Данное программное средство было разработано на языке программирования Swift, что позволит беспрепятственно запускать разработанную программу на одной из самых популярных мобильных операционных систем: iOS.

Целью данного проекта является разработка программного решения, способного оптимизировать операционную деятельность авиабазы и сократить время, затрачиваемое на учет и управление складскими запасами. Программа предоставляет разносторонний функционал, позволяющий не только отслеживать продажи и товары, но и осуществлять учет и анализ данных о сотрудниках, обеспечивая начальство всесторонней информацией для принятия решений.

При выполнении курсового проекта были применены государственные стандарты и нормативные документы, а также СТП БГУИР 01-2017. В пояснительной записке будут представлены три раздела: анализ исходных данных на курсовое проектирование, проектирование и разработка программного средства, эксплуатация программного средства.

Перечень графического материала данной курсовой работы будет включать в себя четыре диаграммы: схема алгоритма, UML диаграмма классов, диаграмма состояний и структура графического пользовательского интерфейса.

Написанный материал пояснительной записки является оригинальным на XX процентов. С отчетом о проверке на плагиат можно ознакомиться в приложении А на рисунке А.1.

1.АНАЛИЗ ИСХОДНЫХ ДАННЫХ НА КУРСОВОЕ ПРОЕКТИРОВАНИЕ

1.1 Анализ исходных данных к курсовой работе

В современном мире трудно представить себе функционирование авиабазы без использования технических средств, которые явно облегчают ежедневную работу. Однако даже в таких условиях невозможно обойтись без бумажной документации, инструкций, предписаний и других разнообразных документов. Это приводит к повышенному спросу на специализированное оборудование для оснащения авиабаз. И это вполне оправданно, поскольку никто не желает тратить время на многократное ручное переписывание одних и тех же документов. В результате возникает большой спрос на различные технические устройства, такие как системы хранения, обработки и передачи информации, автоматизирующие рабочие процессы в авиабазах [1]. Разработанное приложение направлено на решение данной проблемы.

Данная программа предоставляет возможность вести компьютизированный учет всей амуниции, снаряжения и боеприпасов. В приложении реализована возможность зарегестрироваться или же авторизоваться. Также программа позволяет вносить изменения в базу данных непосредственно через само оконное приложение, добавлять либо же удалять необходимые данные из выбранной пользователем таблицы.

Программа имеет интуитивно понятный интерфейс [2], что значительно облегчит освоение данного программного средства потенциальными пользователями.

1.2 Обоснование и описание выбора языка программирования, средств разработки, используемых технологий и сторонних библиотек

Свифт (Swift) – это мощный и современный язык программирования, разработанный компанией Apple для создания приложений под операционные системы iOS, macOS, watchOS и tvOS. За короткое время своего существования, Свифт превратился в один из наиболее популярных языков программирования, используемых разработчиками по всему миру [3]. В этом тексте мы рассмотрим историю Свифта, его возможности и объясним, почему выбор Свифта может быть отличным решением для написания вашей курсовой работы.

История Свифта началась в 2010 году, когда компания Apple решила разработать новый язык программирования, который мог бы предложить более современный и эффективный подход к разработке приложений для их

платформ. Целью было создание языка, который был бы безопасным, быстрым, выразительным и легким в использовании.

В 2014 году Apple представила Свифт публике, и он стал доступным для разработчиков вместе с выпуском Xcode 6 [4]. С тех пор Свифт претерпел множество изменений и улучшений, и на данный момент актуальная версия языка – Swift 5.5 (на момент моего знания, в сентябре 2021 года).

Свифт обладает рядом преимуществ, которые делают его привлекательным для разработчиков. Вот некоторые из них:

Простота использования: Свифт был разработан с учетом простоты и интуитивности. Он имеет чистый и лаконичный синтаксис, который позволяет разработчикам писать код быстрее и эффективнее[5].

Безопасность: Свифт предлагает множество функций, которые помогают предотвратить ошибки программирования. Он обладает системой типов данных, которая помогает выявить ошибки на этапе компиляции, а не во время выполнения программы. Это позволяет снизить количество ошибок и упрощает отладку кода.

Высокая производительность: Свифт создан с учетом высокой производительности. Он использует современные оптимизации, которые позволяют ему выполняться быстрее и эффективнее, чем некоторые другие языки программирования.

Кросс-платформенность: Хотя Свифт изначально был разработан для создания приложений под платформы Apple, Apple открыла исходный код Свифта, и теперь он доступен для разработки на других платформах [6]. Это означает, что вы можете использовать Свифт для создания приложений под Linux и другие платформы, расширяя свою аудиторию и возможности разработки.

Активное сообщество: Свифт имеет активное и дружественное сообщество разработчиков [7]. Существует множество онлайн-ресурсов, форумов и инструментов, которые помогут вам в процессе разработки на Свифте. Вы всегда сможете найти поддержку и решить свои вопросы, если возникнут сложности [8].

Выбор Свифта является отличным решением по нескольким причинам. Во-первых, Свифт – это язык программирования, который широко используется для разработки iOS-приложений. Использование Свифта позволит лучше понять особенности разработки под iOS и создать более качественное приложение.

Во-вторых, Свифт предлагает множество современных функций и возможностей, которые упрощают процесс разработки. Например, функциональное программирование, опциональные типы данных, обработка ошибок, удобная работа с коллекциями данных и многое другое [9]. Эти

возможности позволяют писать более чистый, эффективный и поддерживаемый код.

В-третьих, Свифт обладает богатой экосистемой инструментов и библиотек, которые помогут вам ускорить разработку и добавить функциональность в ваше приложение [10] Существует множество готовых решений для работы с пользовательским интерфейсом, базами данных, сетевым взаимодействием и другими задачами разработки [11]. Это позволит сосредоточиться на решении основных задач и создать более полноценное приложение.

Наконец, выбор Свифта для написания курсовой работы также имеет практическое значение. Знание и опыт работы с Свифтом являются ценными навыками на рынке труда [12].

В целом, Свифт – это мощный, современный и популярный язык программирования, который предлагает множество преимуществ и возможностей для разработки приложений под платформы Apple.

Свифт является одним из наиболее востребованных языков программирования на рынке труда, особенно среди разработчиков, специализирующихся на создании приложений для платформ Apple. Благодаря своим мощным возможностям и современному дизайну, Свифт позволяет разработчикам создавать высококачественные и инновационные приложения, которые могут быть оптимизированы для работы на различных устройствах Apple, включая iPhone, iPad, Mac и Apple Watch [13].

Одним из ключевых преимуществ Свифта является его безопасность и надежность. Язык имеет строгую систему типов, которая помогает выявлять ошибки на ранних стадиях разработки и предотвращать множество распространенных ошибок программирования. Это позволяет разработчикам создавать приложения с меньшим количеством ошибок и повышать их общую стабильность и надежность.

2.ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1 Проектирование объектной модели и описание состояний программного средства

Одним из важных этапов в создании программного обеспечения является проектирование объектной модели и описание состояний программного средства. Объектная модель представляет собой абстракцию системы, которая включает в себя объекты, их свойства и взаимодействия между ними. Описание состояний программного средства позволяет определить, как система может находиться в различных состояниях в зависимости от внешних воздействий и внутренних процессов[15]. Это помогает разработчикам предусмотреть различные сценарии использования, обрабатывать ошибки и обеспечивать стабильную работу приложения.

В данном разделе будет представлена диаграмма состояний программного средства для организации деятельности авиабазы. Диаграмма состояний отображает все возможные состояния системы и переходы между ними. Она помогает визуализировать логику работы программы и понять, как система реагирует на различные события и воздействия.

Кроме того, в этом разделе будет рассмотрен фрагмент диаграммы классов. Диаграмма классов представляет собой структурное представление классов и их отношений в программе. Она помогает понять, какие классы существуют в системе и как они взаимодействуют друг с другом.

Для лучшего понимания организации логики в разработанном приложении также будет представлена структура программного средства. Структура программного средства определяет, как компоненты и модули системы организованы и взаимодействуют между собой. Она также позволяет определить, каким образом осуществляется хранение файлов и какие файлы отвечают за различные аспекты программы.

Все вышеуказанные моменты будут подробно рассмотрены с помощью иллюстраций, которые помогут визуализировать концепции и улучшить понимание описанных аспектов разработки программного обеспечения.

На рисунке 2.1 будет представлена диаграмма состояний программного средства.

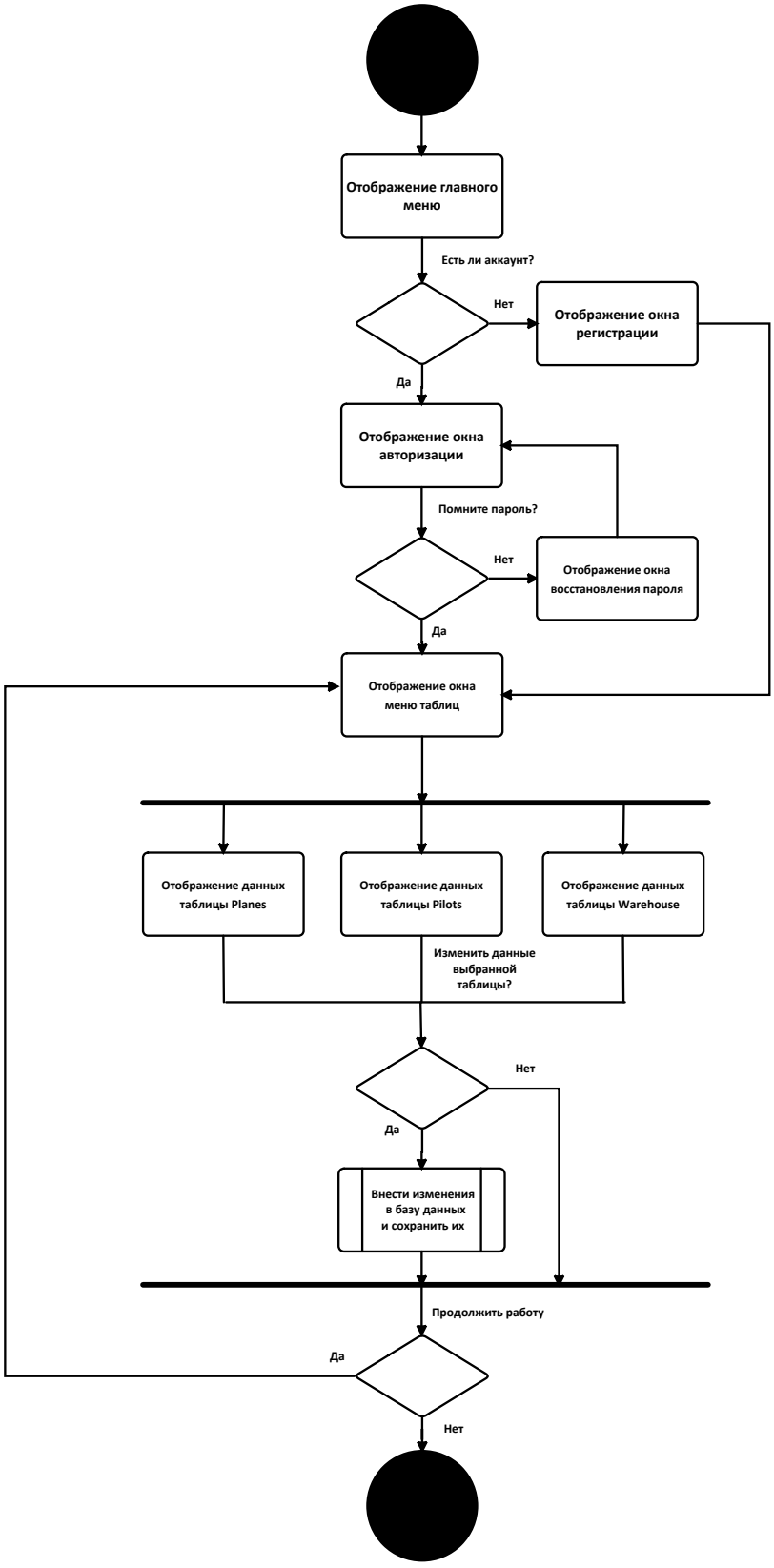


Рисунок 2.1 – Диаграмма состояний программного средства

Данная диаграмма показывает принцип работы разработанного программного средства. Программа при запуске открывает окно регистрации, где от пользователя потребуется ввести адрес электронной почты, к которой он хочет привязать свой аккаунт, придумать логин и пароль. При выборе варианта «авторизация» пользователю предоставится окно, где необходимо будет ввести логин и пароль от желаемой учетной записи. После входа в программу пользователю будет предоставлена возможность просматривать и редактировать данные в таблицах.

Рассмотрим более детально состояние программы на этапе авторизации. Фрагмент диаграммы, отображающий этот этап, будет продемонстрирован на рисунке 2.2.

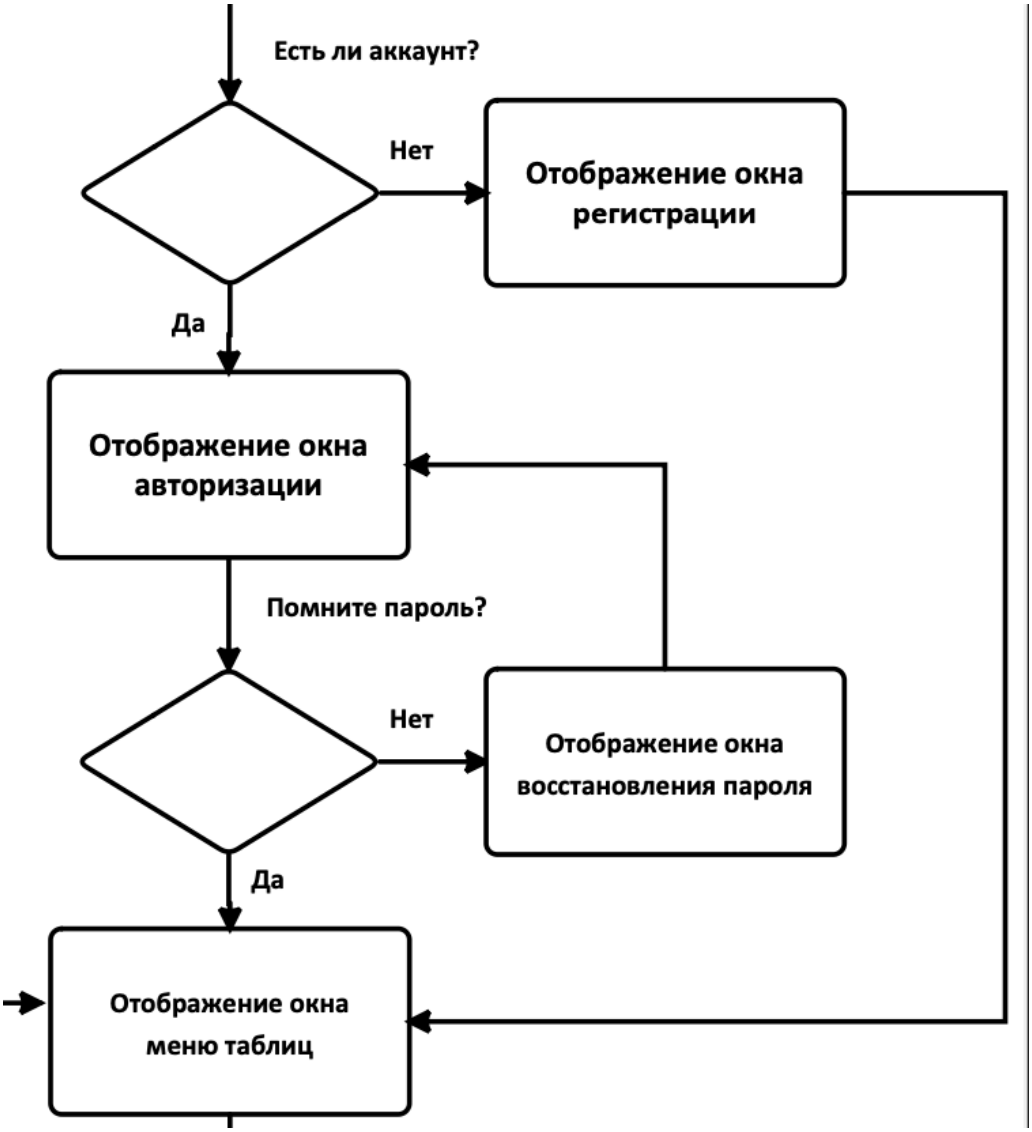


Рисунок 2.2 – Фрагмент диаграммы для состояния программы на этапе авторизации

При входе в аккаунт пользователю потребуется ввести свои логин и пароль. Так же этом месте предусмотрены функции восстановления пароля на случай, если пользователь не может вспомнить свои данные для входа.

Также для данного программного средства была продумана своя определенная структура проекта, которая будет приведена на рисунке 2.3.

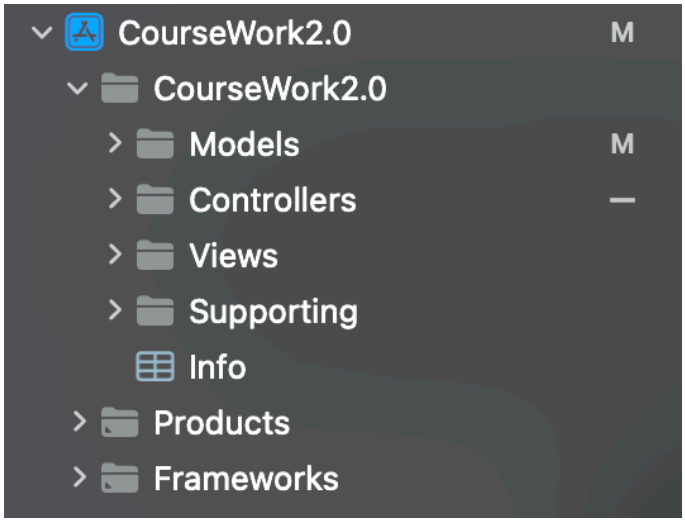


Рисунок 2.3 – Структура проекта

В папке «CourseWork2.0» находятся все классы, отвечающие за логику проекта, а также за правильное выполнение всех функций и корректное подключение программного средства к базе данных.

Изучение и оптимизация процессов проектирования объектной модели и описания состояний программного средства существенно влияют на качество программ, обеспечивая их устойчивость, гибкость и эффективность.

2.2 Проектирование и разработка графического интерфейса

Для проектирования и разработки графического интерфейса программного средства для оптимизации деятельности авиабазы было принято решение воспользоваться библиотекой для графики UIKit языка программирования Swift. Библиотека UIKit предоставляет различные инструменты и функции для создания пользовательского интерфейса (UI) и оптимизации пользовательского опыта (UX).

UI (User Interface) и UX (User Experience) - это два термина, связанных с проектированием пользовательского интерфейса приложений или веб-сайтов [17].

UI (User Interface) описывает то, как пользователь взаимодействует с интерфейсом приложения или веб-сайта. Это включает в себя дизайн элемен-

тов интерфейса, таких как кнопки, поля ввода, меню и т.д., а также расположение и организацию этих элементов. Цель UI-дизайна – сделать интерфейс интуитивно понятным, удобным в использовании и привлекательным для пользователя [18].

UX (User Experience) описывает общий опыт пользователя при использовании приложения или веб-сайта. Это включает в себя все аспекты взаимодействия пользователя с продуктом – от первичного впечатления до навигации, процесса взаимодействия и выполнения задач. Цель UX-дизайна – создать удовлетворительный и приятный пользовательский опыт, учитывая потребности и ожидания пользователей.

UI и UX тесно связаны и важны для создания успешных и эффективных пользовательских интерфейсов. Хороший дизайн UI и UX может повысить удобство использования, улучшить вовлеченность пользователей и повысить общую стоятельность приложений и веб-сайтов.

Все окна данного программного средства будут оформлены в едином стиле для поддержания единообразия всей программы, чтобы у пользователей не возникало проблем со взаимодействием с ними и перемещением между окнами.

Как уже упоминалось ранее, разработанное программное средство будет начинаться с окна регистрации. Вид данного окна во время работы с программой будет представлен на рисунке 2.4.

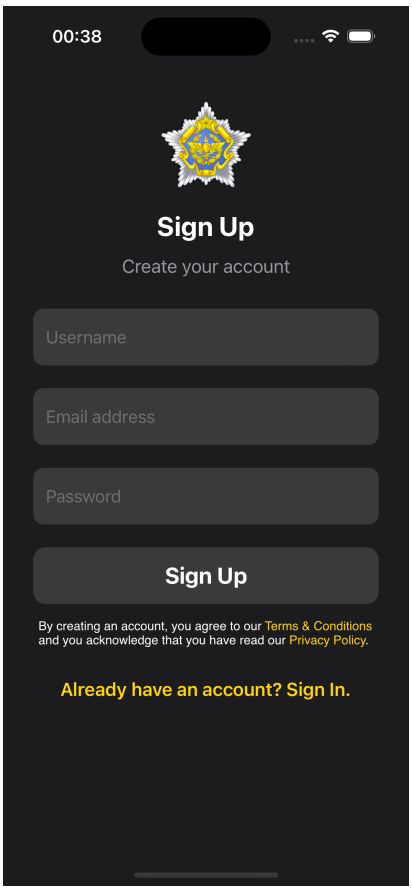


Рисунок 2.4 – Вид окна регистрации

Ниже будет приведен фрагмент кода, отвечающий за с регистрацию нового пользователя.

```
    public func registerUser(with userRequest: RegisterUser-
Request, completion: @escaping (Bool, Error?)->Void) {
            let username = userRequest.username
            let email = userRequest.email
            let password = userRequest.password

              Auth.auth().createUser(withEmail: email, password:
password) { result, error in
                if let error = error {
                    completion(false, error)
                    return
                }

                guard let resultUser = result?.user else {
                    completion(false, nil)
                    return
                }

                let db = Firestore.firestore()
                db.collection("users")
                    .document(resultUser.uid)
```

```
                .setData([
                    "username": username,
                    "email": email,
                    "password": password
                ]) { error in
                    if let error = error {
                        completion(false, error)
                        return
                    }

                    completion(true, nil)
                }
            }
        }
```

Рассмотрим детальнее окна авторизации и регистрации.

Все поля для заполнения должны быть подписаны, чтобы пользователь знал, какую информацию необходимо туда внести.

На рисунке 2.5 представлено окно авторизации пользователя.

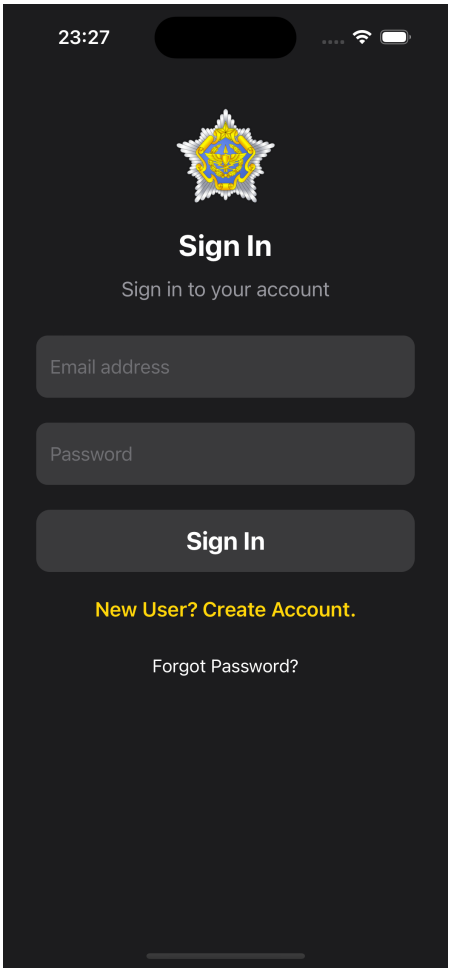


Рисунок 2.5 – Вид окна авторизации

На рисунке 2.6 представлено окно регистрации пользователя.

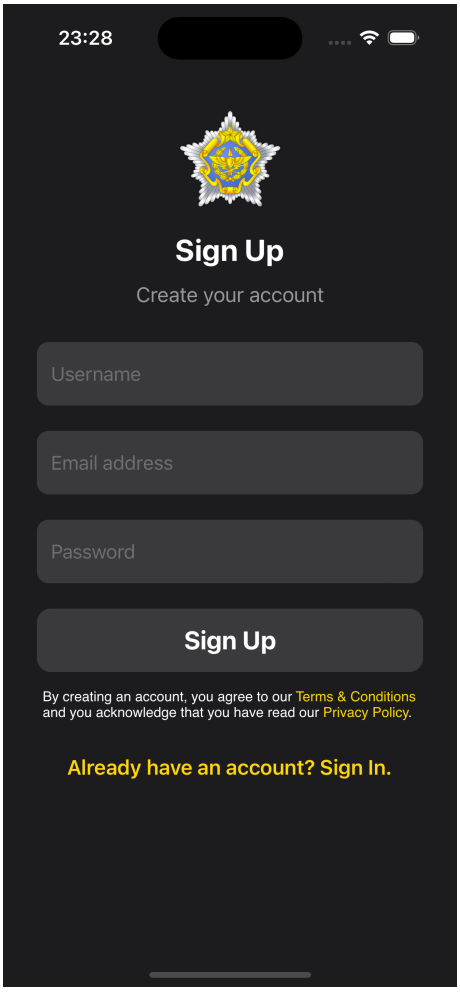


Рисунок 2.6 – Вид окна регистрации

После авторизации пользователя в программе запускается окно, содержащее все таблицы, имеющиеся в системе. В каждом окне должна быть навигация, то есть с любого окна пользователь может вернуться в главное меню. На рисунке 2.7 изображено окно, содержащее все таблицы, имеющиеся в системе. При нажатии на любую из них отобразится окно с информацией по данной таблице. При нажатии на кнопку «Back» пользователь перейдёт на страницу с выбором таблиц.

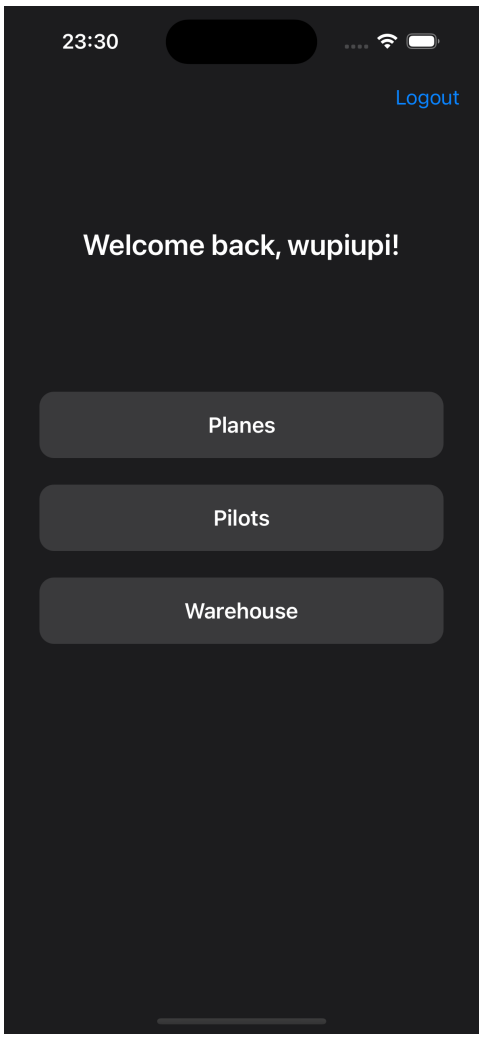


Рисунок 2.7 – Вид окна со списком таблиц

На рисунке 2.8 изображено окно информации о конкретной таблице. На ней пользователь может добавлять либо же удалять данные.

На данном экране со списком всех таблиц, пользователь видит меню, содержащее все таблицы, имеющиеся в системе. На экране представлено приветствие пользователя, кнопка выхода, а также 3 кнопки: Planes, Pilots и Warehouse.

Каждая кнопка ведет на соответствующий список, где пользователь может выполнять дальнейшие действия.

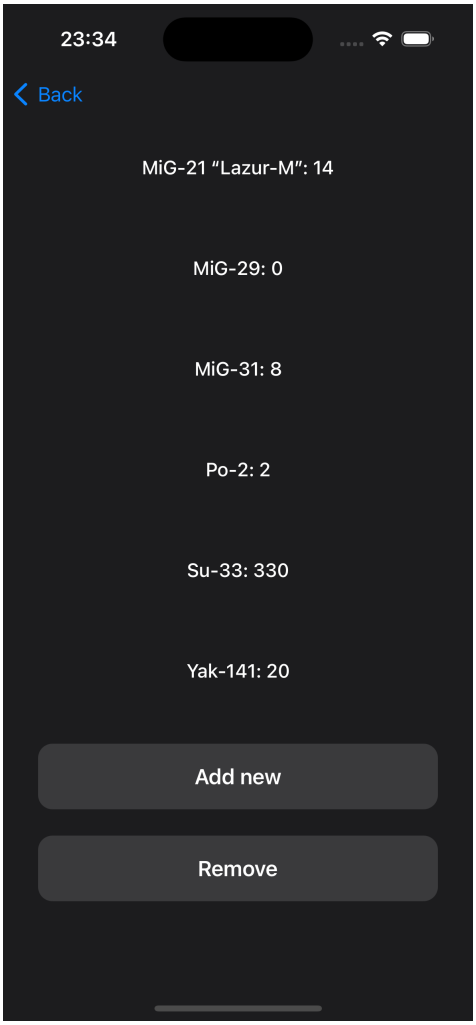


Рисунок 2.8 – Окно информации о таблице

Таким образом, были спроектированы прототипы пользовательского интерфейса разрабатываемого программного средства и внедрены в проект.

2.3 Описание и реализация используемых в программном средстве алгоритмов

В мире программирования алгоритмы играют решающую роль, определяя эффективность и функциональность создаваемых программ. Алгоритм – это набор инструкций, предписывающих последовательность шагов для достижения конкретной цели. Он подобен рецепту, по которому готовится блю-

до, где каждый шаг определен и направлен на получение определенного результата [19].

Алгоритмы широко используются в программировании для решения разнообразных задач, начиная от сортировки данных и поиска оптимальных путей до шифрования информации. Важной характеристикой алгоритма является его инструктивность – способность быть понятым и легко интерпретированным человеком.

Взаимодействие алгоритмов с различными структурами данных, такими как массивы, списки и деревья, играет ключевую роль в эффективной обработке и хранении информации. Выбор подходящей структуры данных в сочетании с оптимальным алгоритмом обеспечивает оптимальную производительность программы.

Известные алгоритмы, такие как сортировка QuickSort, поиск в ширину (BFS), шифрование RSA, являются краеугольными камнями мира программирования. Их анализ и понимание обеспечивают необходимый фундамент для разработчиков.

Существуют различные типы алгоритмов [20]:

- Линейный алгоритм − последовательность команд (инструкций), выполняемых последовательно во времени.

- Разветвляющийся алгоритм − алгоритм, содержащий в качестве результата управления хотя бы одно условие, которое можно разделить на несколько альтернативных ветвей алгоритма.

- Циклический алгоритм − алгоритм, в котором одно и то же действие (один и тот же процесс) повторяется несколько раз. Большинство вычислительных методов сводятся к круговым алгоритмам. Цикл программы − последовательность инструкций (серия, тело цикла), которая может выполняться многократно.

Основным способом визуализации последовательности алгоритмов является создание схем алгоритмов. Схема алгоритма представляет собой визуальное представление шагов, которые необходимо выполнить для решения определенной задачи или достижения конкретной цели. С помощью схемы алгоритма можно проиллюстрировать последовательность действий, операторы и управляющие конструкции, используемые в процессе выполнения алгоритма.

Схема алгоритма позволяет легко визуализировать и понять последовательность выполнения операций, структуру алгоритма и поток управления программы. Она может быть использована для разработки, исправления и отладки алгоритмов перед переходом к написанию программного кода. Благодаря графическому представлению, схема алгоритма удобна для коммуника-

ции и обмена информацией между разработчиками или командами, работающими над одной задачей.

Далее будут представлены примеры используемых в проекте алгоритмов каждого типа с их схемами для лучшего понимания работы и наглядности.

Первым следует представить пример линейного алгоритма в связи с его простотой. Таким алгоритмом является алгоритм очистки полей почты и пароля. Его код представлен ниже:

```
private func clear-
     emailField.text =
    passwordField.text
}
```

```
TextFields() {
nil
= nil
```

Схема данного алгоритма будет приведена на рисунке 2.9.

Рисунок 2.9 – Схема линейного алгоритма

Далее представлен пример разветвлящегося алгоритма. Данным алгоритмом является алгоритм авторизации . Его код представлен ниже:

```
public func signIn(with userRequest: LoginUserRequest, completion:
@escaping (Error?)->Void) {
            Auth.auth().signIn(
                withEmail: userRequest.email,
                password: userRequest.password
            ) { result, error in
                if let error = error {
                    completion(error)
```

```
                return
            } else {
                completion(nil)
            }
        }
    }
```

Схема данного алгоритма будет приведена на рисунке 2.10.

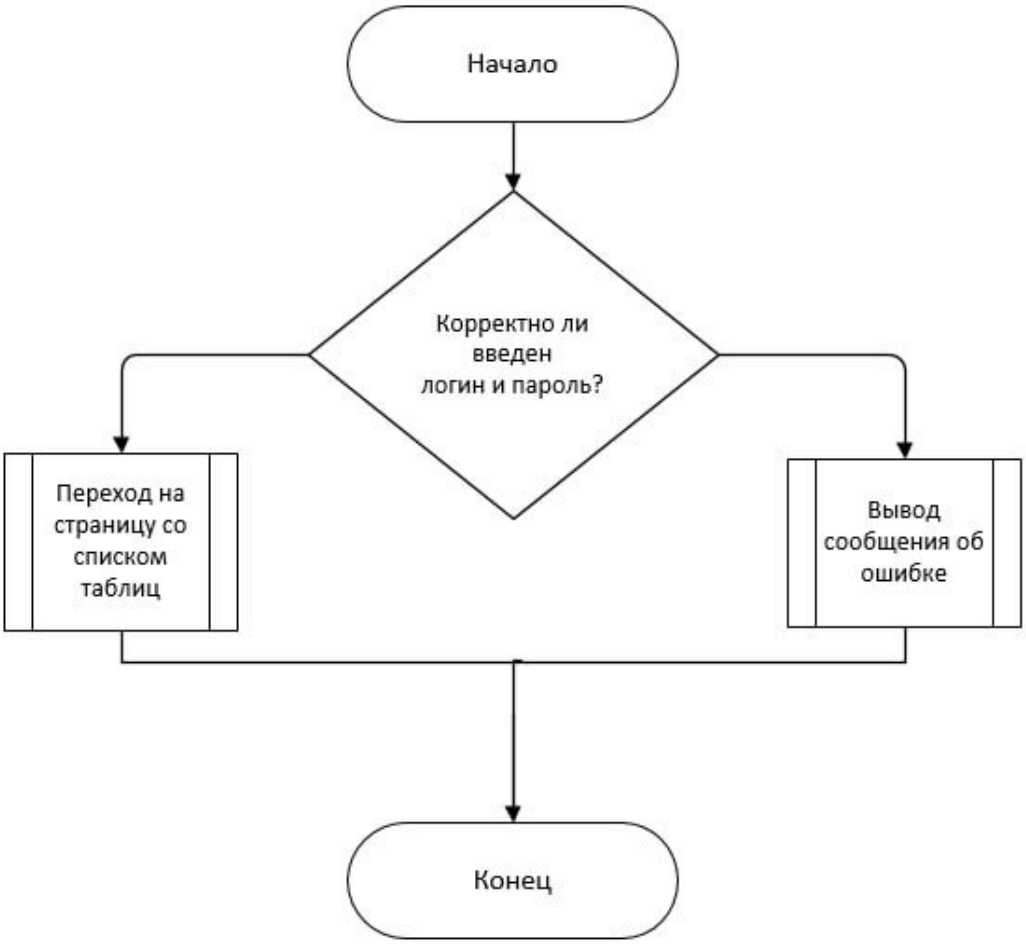


Рисунок 2.10 – Схема разветвляющегося алгоритма

Последним представлен циклический алгоритм. Чаще всего алгоритмы данного типа применяются при необходимости работы с большим массивом связанных данных. Одним из таких алгоритмов в проекте является алгоритм присвоения текстовым полям значений, а также их добавления в стек. Его код представлен ниже:

```
for (key, value) in sortedDictionary {
                    let label = self.getLabel()
                    label.text = "\(key): \(value)"
                    self.stackView.addArrangedSubview(label)
                }
```

Схема данного алгоритма будет приведена на рисунке 2.11.

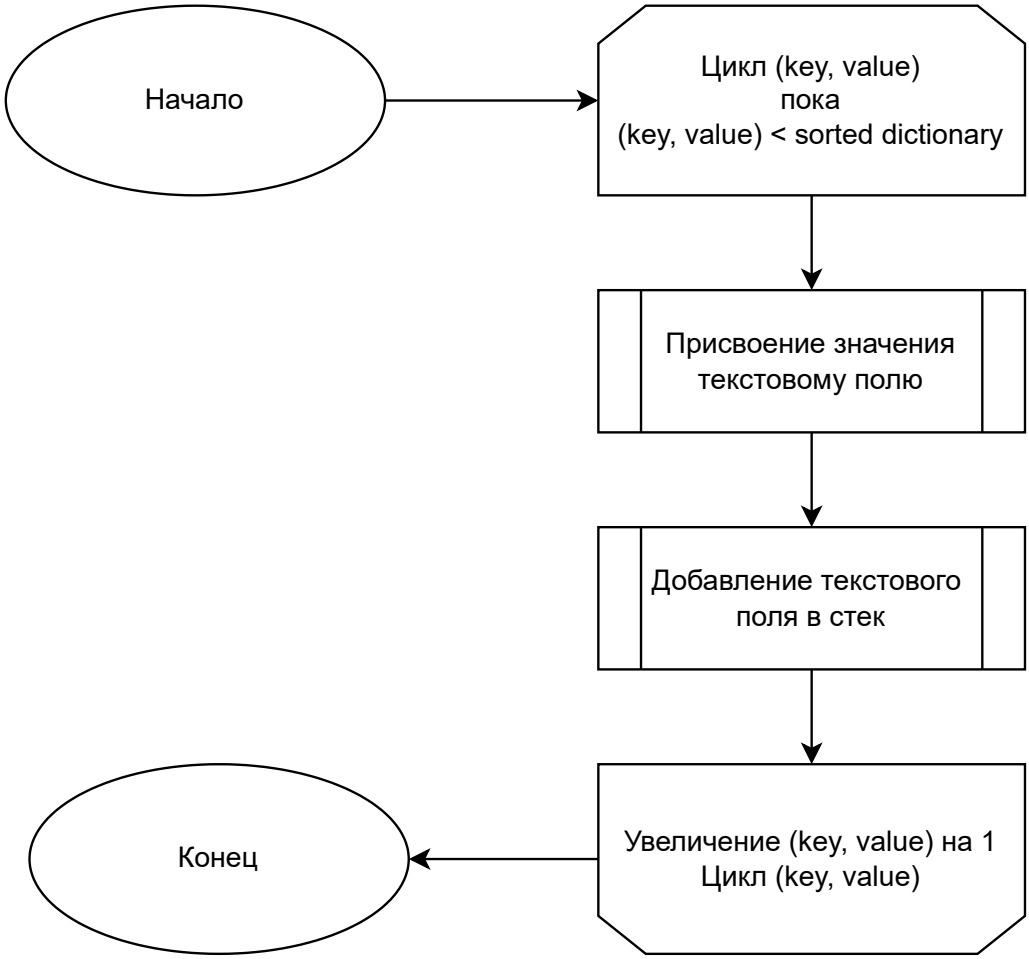


Рисунок 2.11 – Схема циклического алгоритма

Также в перечне графического материала будет представлена схема алгоритма всей программы. Эта схема позволит лучше отобразить все внутренние процессы разработанного программного средства.

3.ЭКСПЛУАТАЦИЯ ПРОГРАММНОГО СРЕДСТВА

3.1 Ввод в эксплуатацию и обоснование минимальных технических требований к оборудованию

Ввод в эксплуатацию программного средства на платформе Xcode представляет собой критически важный этап в жизненном цикле разработки программного обеспечения [21]. Этот этап включает в себя развертывание приложения, подготовку к работе конечных пользователей и обеспечение необходимых технических ресурсов для эффективной работы приложения. В данном разделе будет рассмотрен процесс ввода в эксплуатацию программного средства на платформе AppStore и обоснование минимальных технических требований к оборудованию.

Ввод в эксплуатацию предполагает выполнение ряда шагов, направленных на успешное развертывание программного продукта и готовность его к использованию конечными пользователями. Основные этапы ввода в эксплуатацию в AppStore включают в себя:

- Регистрацию в программе разработчиков на официальном сайте Apple для разработчиков.

– Тестирование на совместимость: проведение тестирования на различных устройствах и операционных системах, чтобы обеспечить совместимость приложения с различными устройствами.

– Развертывание на сервере или хосте: если приложение требует централизованного доступа, необходимо убедиться, что оно развернуто на сервере или хосте, готовом обрабатывать запросы от пользователей.

– Загрузка приложения в AppStore.

– Обеспечение безопасности: разработчику следует разработать и реализовать меры безопасности для защиты приложения от несанкционированного доступа и вредоносных атак.

– Обучение пользователей: предоставление руководства пользователя и обучение для конечных пользователей, чтобы обеспечить эффективное использование приложения.

Обоснование минимальных технических требований к оборудованию включает в себя определение минимальных характеристик, которые должны быть у конечных пользователей для нормального функционирования программного средства. Эти требования могут включать в себя:

– Процессор и память: минимальные требования к процессору и оперативной памяти для стабильного выполнения приложения.

Разрешение экрана: рекомендуемое разрешение экрана для оптимального визуального восприятия приложения.

Версия iOS: указание минимальной версии iOS, которую должно поддерживать устройство пользователя.

В рамках данной курсовой работы проект был реализован при помощи Xcode, что накладывает ограничение на версию языка Swift, то есть Swift должна быть 4 версии и старше [22].

Рассмотрим вариант для ввода в эксплуатацию и дальнейшего распространения с созданием исполняемого файла и последующей загрузкой в AppStore, откуда его сможет скачать любой желающий. Чтобы создать такой файл сперва необходимо заархивировать проект. Это можно сделать в самой среде разработки Xcode, перейдя в окно Product -> Archive. После успешного архивирования проекта, необходимо выбрать опцию «Distribute App» или «Export» в Xcode, далее, следуя по инструкциям мастера экспорта, необходимо создать исполняемый файл для вашего проекта.

Затем, созданный исполняемый файл можно загрузить в AppStore, чтобы любой желающий мог скачать его. Для обеспечения работоспособности программы на корпоративном уровне, убедитесь, что на сервере установлена подходящая версия Swift и необходимые зависимости для выполнения вашего приложения [23].

Также для обеспечения работоспособности программы в условиях корпоративного использования необходимо убедиться, что на сервере установлена подходящая версия Swift, необходимая для выполнения разработанного приложения. Затем необходимо перенести собранный исполняемый файл и все необходимые ресурсы на сервер, в том числе и разработанная база данных [24]. Это можно сделать с использованием scp (Secure Copy Protocol) или других инструментов передачи файлов.

3.2.Руководство по эксплуатации программного средства

Представим руководство пользователя служащего авиабазы, который пользуется программным средством для работы с информацией о летном составе пилотов, наличии самолетов или же снаряжения на складе.

При запуске программы открывается главная страница, на которой есть кнопки «Sign Up», «Already have an account? Sign In», «Terms & Conditions» и «Privacy Policy» (см. рисунок 3.1).

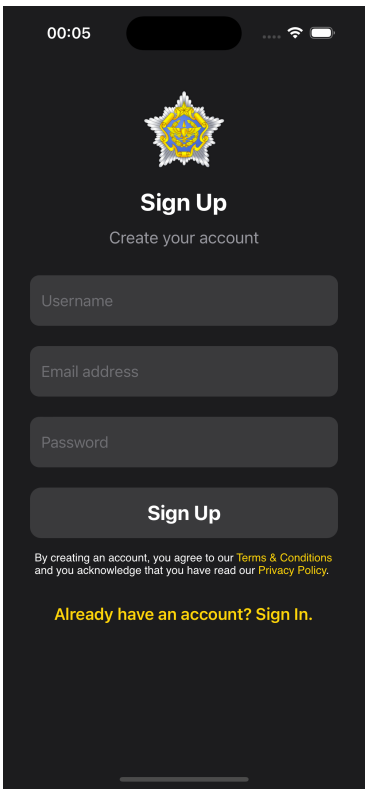


Рисунок 3.1 – Начальная страница

В случае, если пользователь уже зарегистрирован в системе, необходимо нажать на кнопку «Already have an account? Sign In.», тогда пользователь попадает на страницу авторизации в программном средстве (см. рисунок 3.2).

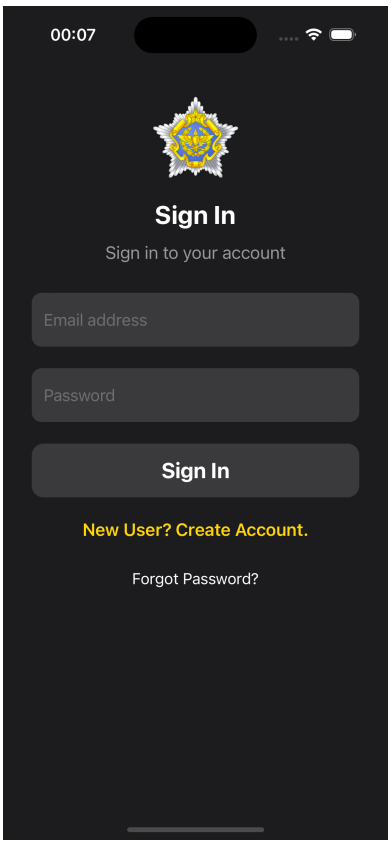


Рисунок 3.2 – Страница авторизации

После прохождения авторизации пользователь будет перенаправлен на страницу с выбором таблиц.

В случае, если пользователь забыл свои учетные данные, в программе предусмотрена возможность восстановления пароля. Для восстановления пароля пользователю необходимо нажать на кнопку «Forgot Password?», после чего программа перейдет к окну восстановления пароля (см. рисунок 3.3).

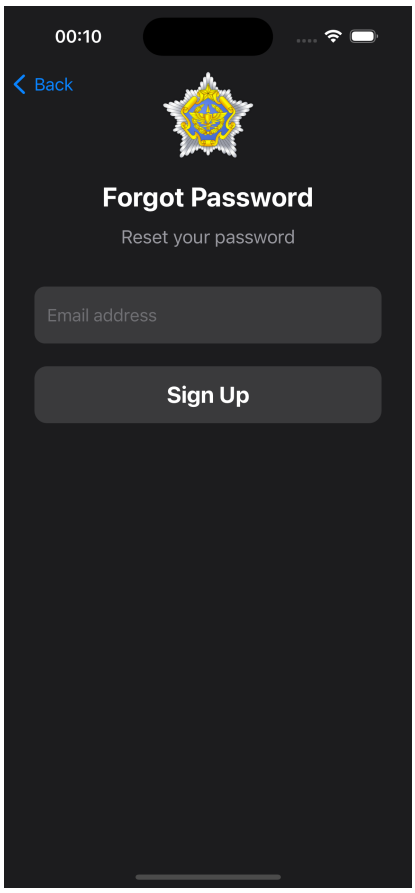


Рисунок 3.3 – Страница восстановления пароля

Сделав выбор необходимой таблицы, пользователь перейдет к странице, которая содержит все интересующие его данные. На этой странице пользователь сможет просматривать данные, а также удалять и добавлять данные в таблицу (см. рисунок 3.4).

Данный экран содержит оглавление, напоминающее пользователю, что он находится на экране восстановления пароля, поле для ввода почты, на которую придет форма со сбросом пароля и две кнопки: "назад" и "восстановить пароль".

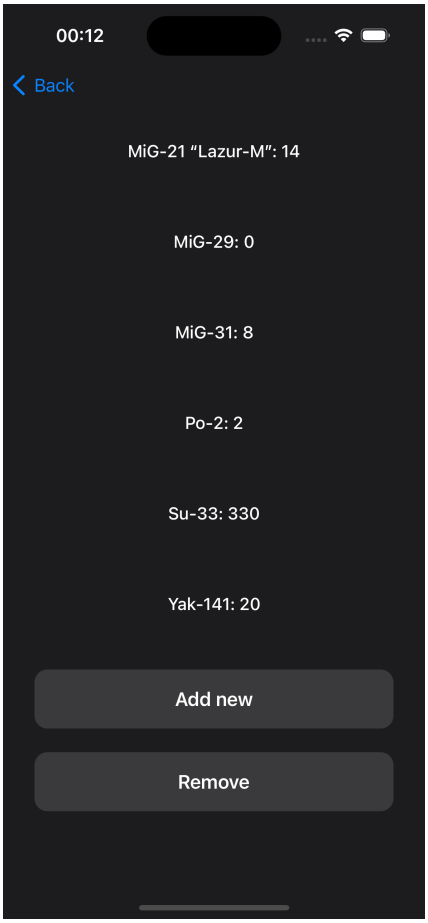


Рисунок 3.4 – Страница с данными выбранной таблицы

Для выхода из своего аккаунта пользователю необходимо перейти на страницу главного меню и нажать на кнопку «Logout».

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была достигнута цель – разработано программное средство для обеспечения авиабазы. В ходе разработки были решены следующие задачи:

Был проведен анализ информационных потребностей пользователей и сформулирована задача, реализация которой позволила удовлетворить их потребности. Также был проведен анализ исходных данных, была спроектирована объектная модель программного средства, составлено описание его состояний, был спроектирован и разработан графический интерфейс программы. Вместе с этим было составлено описание используемых в программном средстве алгоритмов, руководство по эксплуатации, обоснование минимальных технических требований к оборудованию, а также описана среда и язык реализации с подробной аргументацией выбора используемых технологий.

Были разработаны инфологическая и физическая модели БД, определены типы данных и ограничения. Были освоены некоторые приемы написания запросов и представлений на языке Firebase.

В данной курсовой работе была спроектирована (до 3НФ) и реализована реляционная база данных (СУБД Firebase).

В разработанном программном средстве также созданы ограничения на ввод некорректных символов, проверка на правильность введенных логинов и паролей, а также были предусмотрены ограничения для ненадежных паролей.

В дальнейшем разработанную систему можно усовершенствовать, добавив дополнительный функционал программного средства.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Оргтехника – что к ней относится? [Электронный ресурс]. – Режим доступа: https://flex-system.ru/novosti-i-stati/articles/3576000/ – Дата доступа: 19.09.2023.

[1] "The Swift Programming Language" - официальная документация по языку Swift от Apple. [Электронный ресурс]. – Режим доступа: https://docs.swift.org/swift-book/ – Дата доступа: 23.11.2023.

[2] Интуитивно понятный интерфейс: 4 простых принципа [Электронный ресурс]. – Режим доступа: https://ru.experrto.io/blog/2019/06/28/intuitivno-ponyatnyj-interfejs-4-prostyh-principa/ – Дата доступа: 24.09.2023.

[3] "Why All Aspiring Coders Should Learn How to Use Swift" - блог и подкаст, посвященные разработке на Swift. [Электронный ресурс]. – Режим доступа: https://emeritus.org/ – Дата доступа: 01.12.2023.

[4] "Hacking with Swift" - обучающие ресурсы и учебники по Swift. [Электронный ресурс]. – Режим доступа: https://www.hackingwithswift.com/ – Дата доступа: 25.11.2023.

[5] "Ray Wenderlich" - онлайн-платформа с множеством руководств и статей по Swift и iOS-разработке. [Электронный ресурс]. – Режим доступа: https://www.raywenderlich.com/ – Дата доступа: 27.11.2023.

[6] "SwiftVersion" - архив всех версий языка Swift. [Электронный ресурс]. – Режим доступа: https://swiftversion.net/ – Дата доступа: 26.11.2023.

[7] "Swift News" - новости и статьи о Swift. [Электронный ресурс]. – Режим доступа: https://swiftnews.curated.co/ – Дата доступа: 24.11.2023.

[8] "Swift.org YouTube Channel" - официальный YouTube-канал сообщества Swift с видеоуроками и докладами. [Электронный ресурс]. – Режим доступа: https://www.youtube.com/c/swiftorg – Дата доступа: 27.11.2023.

[9] "Swift Developments" - подкаст о разработке на Swift. [Электронный ресурс]. – Режим доступа: https://www.swiftdevelopments.net/ – Дата доступа: 23.11.2023.

[10] "Swift Weekly Brief" - еженедельный обзор новостей и событий в мире Swift. Режим доступа: https://swiftweekly.github.io/ – Дата доступа: 28.11.2023.

[11] "Swift Forums" - официальные форумы Swift, где можно задавать вопросы и обсуждать темы с сообществом. [Электронный ресурс]. – Режим доступа: https://forums.swift.org/ – Дата доступа: 24.11.2023.

[12] "SwiftLint" - инструмент статического анализа и форматирования кода на Swift. [Электронный ресурс]. – Режим доступа: https://github.com/realm/SwiftLint – Дата доступа: 23.11.2023.

[13] "Awesome-Swift" - подборка полезных ресурсов, библиотек и инструментов для разработки на Swift. [Электронный ресурс]. – Режим доступа: https://github.com/matteocrippa/awesome-swift – Дата доступа: 26.11.2023.

[14] "Swift Playgrounds" - приложение от Apple для изучения и экспериментирования с языком Swift. Режим доступа: https://www.apple.com/swift/playgrounds/ – Дата доступа: 25.11.2023.

[15] "Swift News App" - приложение для получения новостей и статей о Swift. [Электронный ресурс]. – Режим доступа: https://apps.apple.com/app/swift-news/id1217823876 – Дата доступа: 22.11.2023.

[16] "SwiftHub" - приложение для iOS, предоставляющее доступ к репозиториям и вопросам Stack Overflow по Swift. [Электронный ресурс]. – Режим доступа: https://github.com/khoren93/SwiftHub – Дата доступа: 24.11.2023.

[17] "The Swift Programming Language" - документация по языку Swift от Apple. Режим доступа: https://docs.swift.org/swift-book/ – Дата доступа: 19.09.2023.

[18] "Swift.org" - веб-сайт сообщества Swift. [Электронный ресурс]. – Режим доступа: https://swift.org/ – Дата доступа: 21.09.2023.

[19] "Swift by Sundell" - блог и подкаст, посвященные разработке на Swift. Режим доступа: https://www.swiftbysundell.com/ – Дата доступа: 22.09.2023.

[20] "Hacking with Swift" - обучающие ресурсы по Swift. [Электронный ресурс]. – Режим доступа: https://www.hackingwithswift.com/ – Дата доступа: 20.09.2023.

[21] "Ray Wenderlich" - онлайн-платформа с множеством руководств и статей по Swift. [Электронный ресурс]. – Режим доступа: https://www.raywenderlich.com/ – Дата доступа: 23.09.2023.

[22] "SwiftLee" - блог с практическими советами и руководствами по Swift. [Электронный ресурс]. – Режим доступа: https://www.avanderlee.com/ – Дата доступа: 24.09.2023.

[23] "Swift News" - новости и статьи о Swift. [Электронный ресурс]. – Режим доступа: https://swiftnews.curated.co/ – Дата доступа: 25.09.2023.

[24] "Swift.org YouTube Channel" - официальный YouTube-канал сообщества Swift с видеоуроками и докладами. [Электронный ресурс]. – Режим доступа: https://www.youtube.com/c/swiftorg – Дата доступа: 26.09.2023.

[25] "Swift Developments" - подкаст о разработке на Swift. [Электронный ресурс]. – Режим доступа: https://www.swiftdevelopments.net/ – Дата доступа: 27.09.2023.

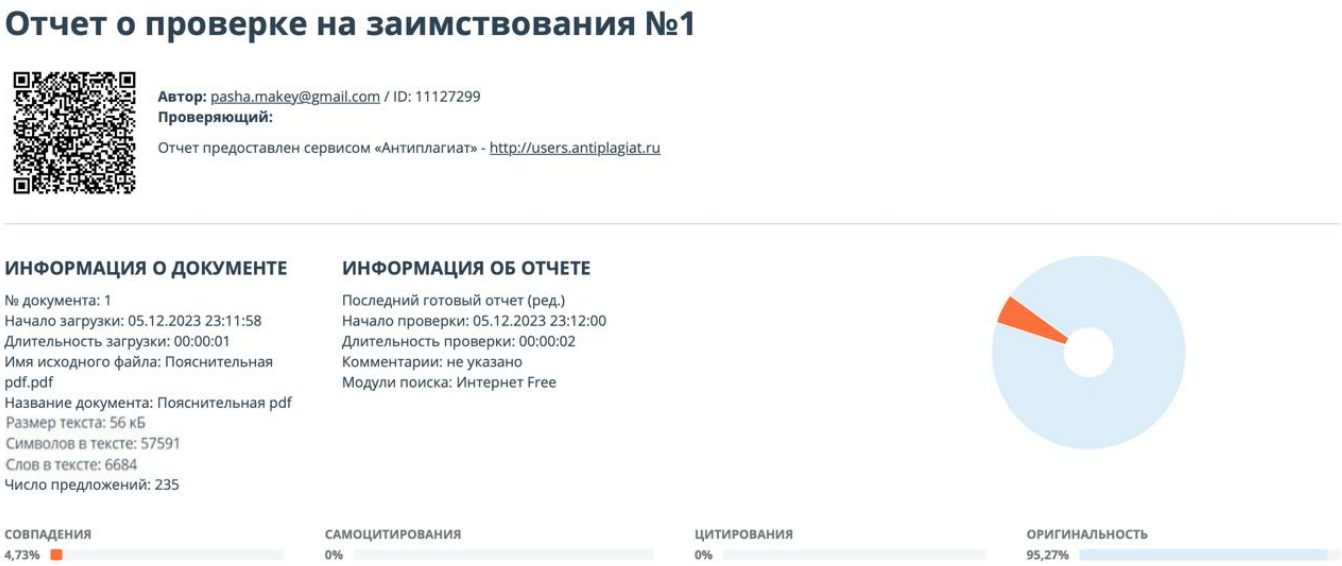
[26] "Swift Weekly Brief" - обзор новостей и событий в мире Swift. Режим доступа: https://swiftweekly.github.io/ – Дата доступа: 28.09.2023.

[27] "Swift Forums" - официальные форумы Swift, где можно задавать вопросы и обсуждать темы с сообществом. [Электронный ресурс]. – Режим доступа: https://forums.swift.org/ – Дата доступа: 20.09.2023.

[28] "SwiftLint" - инструмент статического анализа и форматирования кода на Swift. Режим доступа: https://github.com/realm/SwiftLint – Дата доступа: 21.09.2023.

[29] "Awesome-Swift" - подборка полезных ресурсов для Swift. Режим доступа: https://github.com/matteocrippa/awesome-swift – Дата доступа: 22.09.2023.

ПРИЛОЖЕНИЕ А
(обязательное)
Отчет о проверке на заимствование
в системе «Антиплагиат»



Отчет о проверке на заимствования №1

Автор: pasha.makey@gmail.com / ID: 11127299
Проверяющий:

Отчет предоставлен сервисом «Антиплагиат» - http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 1
Начало загрузки: 05.12.2023 23:11:58
Длительность загрузки: 00:00:01
Имя исходного файла: Пояснительная pdf.pdf
Название документа: Пояснительная pdf
Размер текста: 56 кБ
Символов в тексте: 57591
Слов в тексте: 6684
Число предложений: 235

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 05.12.2023 23:12:00
Длительность проверки: 00:00:02
Комментарии: не указано
Модули поиска: Интернет Free

СОВПАДЕНИЯ 4,73%
САМОЦИТИРОВАНИЯ 0%
ЦИТИРОВАНИЯ 0%
ОРИГИНАЛЬНОСТЬ 95,27%

Рисунок А.1 – Отчет о проверке на заимствование
в системе «Антиплагиат»

ПРИЛОЖЕНИЕ Б
(обязательное)
Листинг кода с комментариями

Файл AuthService.swift

```
import Foundation
import FirebaseAuth
import FirebaseFirestore

final class AuthService {

    public static let shared = AuthService()
    private init() {}

    public func registerUser(with userRequest: RegisterUserRequest,
completion: @escaping (Bool, Error?)->Void) {
        let username = userRequest.username
        let email = userRequest.email
        let password = userRequest.password

        Auth.auth().createUser(withEmail: email, password: password)
{ result, error in
            if let error = error {
                completion(false, error)
                return
            }

            guard let resultUser = result?.user else {
                completion(false, nil)
                return
            }

            let db = Firestore.firestore()
            db.collection("users")
                .document(resultUser.uid)
                .setData([
                    "username": username,
                    "email": email,
```

```
                    "password": password
                ]) { error in
                    if let error = error {
                        completion(false, error)
                        return
                    }

                    completion(true, nil)
                }
            }
        }

        public func signIn(with userRequest: LoginUserRequest, completion:
@escaping (Error?)->Void) {
            Auth.auth().signIn(
                withEmail: userRequest.email,
                password: userRequest.password
            ) { result, error in
                if let error = error {
                    completion(error)
                    return
                } else {
                    completion(nil)
                }
            }
        }

        public func signOut(completion: @escaping (Error?)->Void) {
            do {
                try Auth.auth().signOut()
                completion(nil)
            } catch let error {
                completion(error)
            }
        }

        public func forgotPassword(with email: String, completion: @escap-
ing (Error?) -> Void) {
            Auth.auth().sendPasswordReset(withEmail: email) { error in
                completion(error)
```

```
            }
        }

        public func fetchUser(completion: @escaping (User?, Error?) ->
Void) {

            guard let userUID = Auth.auth().currentUser?.uid else
{ return }

            let db = Firestore.firestore()
            db.collection("users")
                .document(userUID)
                .getDocument { snapshot, error in
                    if let error {
                        completion(nil, error)
                        return
                    }

                    if let snapshot,
                       let snapshotData = snapshot.data(),
                       let username = snapshotData["username"] as? String,
                       let email = snapshotData["email"] as? String {
                        let user = User(username: username, email: email,
userUID: userUID)
                        completion(user, nil)
                    }
                }
        }
    }
```

Файл Validator.swift

```
    import Foundation

    final class Validator {

        static func isValidEmail(for email: String) -> Bool {
            let email = email.trimmingCharacters(in: .whitespacesAndNew-
lines)
```

```
            let emailRegEx = "[A-Z0-9a-z._%+-]+@[A-Za-z0-9.-]+\\.{1}[A-Za-
z]{2,64}"
            let emailPred = NSPredicate(format: "SELF MATCHES %@", email-
RegEx)
            return emailPred.evaluate(with: email)
        }

        static func isValidUsername(for username: String) -> Bool {
            let username = username.trimmingCharacters(in: .whitespacesAnd-
Newlines)
            let usernameRegEx = "\\w{4,24}"
            let usernamePred = NSPredicate(format: "SELF MATCHES %@", user-
nameRegEx)
            return usernamePred.evaluate(with: username)
        }

        static func isPasswordValid(for password: String) -> Bool {
            let password = password.trimmingCharacters(in: .whitespacesAnd-
Newlines)
            let passwordRegEx = "^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?
=.*[$@$#!%*?&]).{6,32}$"
            let passwordPred = NSPredicate(format: "SELF MATCHES %@", pass-
wordRegEx)
            return passwordPred.evaluate(with: password)
        }
    }
```

Файл HomeController.swift

```
    import UIKit
    final class HomeController: UIViewController {

        // MARK: - UI Components
        let planesButton = CustomButton(
            title: "Planes",
            hasBackground: true,
            fontSize: .medium,
            textColor: .label
        )
```

```
private let pilotsButton = CustomButton(
    title: "Pilots",
    hasBackground: true,
    fontSize: .medium,
    textColor: .label
)

private let warehousePasswordButton = CustomButton(
    title: "Warehouse",
    hasBackground: true,
    fontSize: .medium,
    textColor: .label
)

private let greetingLabel: UILabel = {
    let label = UILabel()
    label.textColor = .label
    label.textAlignment = .center
    label.font = .systemFont(ofSize: 24, weight: .semibold)
    label.text = "Loading..."
    label.numberOfLines = 2
    return label
}()

// MARK: - LifeCycle
override func viewDidLoad() {
    super.viewDidLoad()
    self.setupUI()

    self.planesButton.addTarget(
        self,
        action: #selector(didTapPlanes),
        for: .touchUpInside
    )

    self.pilotsButton.addTarget(
        self,
        action: #selector(didTapPilots),
        for: .touchUpInside
    )
```

```
            self.warehousePasswordButton.addTarget(
                self,
                action: #selector(didTapWarehouse),
                for: .touchUpInside
            )


            AuthService.shared.fetchUser { [weak self] user, error in
                guard let self = self else { return }
                if let error = error {
                    AlertManager.showFetchingUserError(on: self, with: er-
ror)
                    return
                }

                if let user {
                    self.greetingLabel.text = "Welcome back, \(user.user-
name)!"
                }
            }
        }


        // MARK: - UI Setup
        private func setupUI() {
            view.backgroundColor = .systemGray6
            navigationItem.rightBarButtonItem = UIBarButtonItem(
                title: "Logout",
                style: .plain,
                target: self,
                action: #selector(didTapLogout)
            )

            view.addSubview(greetingLabel)
            view.addSubview(planesButton)
            view.addSubview(pilotsButton)
            view.addSubview(warehousePasswordButton)

            greetingLabel.translatesAutoresizingMaskIntoConstraints = false
```

```
            planesButton.translatesAutoresizingMaskIntoConstraints = false
            pilotsButton.translatesAutoresizingMaskIntoConstraints = false
            warehousePasswordButton.translatesAutoresizingMaskIntoCon-
straints = false

            NSLayoutConstraint.activate([
                greetingLabel.topAnchor.constraint(equalTo:view.layoutMar-
ginsGuide.topAnchor),
                greetingLabel.leadingAnchor.constraint(equalTo: self-
.view.leadingAnchor),
                greetingLabel.trailingAnchor.constraint(equalTo: self-
.view.trailingAnchor),
                greetingLabel.heightAnchor.constraint(equalToConstant:
200),

                planesButton.topAnchor.constraint(equalTo: greeting-
Label.bottomAnchor, constant: 22),
                planesButton.centerXAnchor.constraint(equalTo: greeting-
Label.centerXAnchor),
                planesButton.heightAnchor.constraint(equalToConstant: 55),
                planesButton.widthAnchor.constraint(equalTo: view.width-
Anchor, multiplier: 0.85),

                pilotsButton.topAnchor.constraint(equalTo: planesBut-
ton.bottomAnchor, constant: 22),
                pilotsButton.centerXAnchor.constraint(equalTo: greeting-
Label.centerXAnchor),
                pilotsButton.heightAnchor.constraint(equalToConstant: 55),
                pilotsButton.widthAnchor.constraint(equalTo: view.width-
Anchor, multiplier: 0.85),

                warehousePasswordButton.topAnchor.constraint(equalTo: pi-
lotsButton.bottomAnchor, constant: 22),
                warehousePasswordButton.centerXAnchor.constraint(equalTo:
greetingLabel.centerXAnchor),
                warehousePasswordButton.heightAnchor.constraint(equalToCon-
stant: 55),
                warehousePasswordButton.widthAnchor.constraint(equalTo:
view.widthAnchor, multiplier: 0.85)
            ])
```

```
        }

        // MARK: - Selectors
        @objc private func didTapLogout() {
            AuthService.shared.signOut { [weak self] error in
                guard let self = self else { return }
                if let error = error {
                    AlertManager.showLogoutError(on: self, with: error)
                    return
                }

                if let sceneDelegate = self.view.window?.windowScene?.dele-
gate as? SceneDelegate {
                    sceneDelegate.checkAuthentication()
                }
            }
        }

        @objc private func didTapPlanes() {
            let vc = PlanesViewController()
            self.navigationController?.pushViewController(vc, animated:
true)
        }

        @objc private func didTapPilots() {
            let vc = PilotsViewController()
            self.navigationController?.pushViewController(vc, animated:
true)
        }

        @objc private func didTapWarehouse() {
            let vc = WarehouseViewController()
            self.navigationController?.pushViewController(vc, animated:
true)
        }
    }
```

Файл AlertManager.swift

```
    import UIKit
```

```
final class AlertManager {

    private static func showBasicAlert(
        on vc: UIViewController,
        title: String,
        message: String?
    ) {
        DispatchQueue.main.async {
            let alert = UIAlertController(
                title: title,
                message: message,
                preferredStyle: .alert
            )
            alert.addAction(
                UIAlertAction(
                    title: "Dismiss",
                    style: .default,
                    handler: nil
                )
            )
            vc.present(alert, animated: true)
        }
    }
}

// MARK: - Show Validation Alerts
extension AlertManager {

    public static func showEmptyFieldAlert(on vc: UIViewController) {
        self.showBasicAlert(
            on: vc,
            title: "Empty field",
            message: "Please, fill the field(s)"
        )
    }

    public static func showInvalidEmailAlert(on vc: UIViewController) {
        self.showBasicAlert(on: vc, title: "Invalid Email", message:
"Please enter a valid email.")
```

```
        }

        public static func showInvalidPasswordAlert(on vc: UIViewCon-
troller) {
            self.showBasicAlert(on: vc,
                                title: "Invalid Password",
                                message: """
                                        Please enter a valid password.
                                        Your password must be have at
least:
                                        6 characters long
                                        1 uppercase & 1 lowercase character
                                        1 number
                                        1 special symbol ($O$#!%*?&)
                                        """)
        }

        public static func showInvalidUsernameAlert(on vc: UIViewCon-
troller) {
            self.showBasicAlert(on: vc, title: "Invalid Username", message:
"Please enter a valid username.")
        }
    }


    // MARK: - Registration Errors
    extension AlertManager {

        public static func showRegistrationErrorAlert(on vc: UIViewCon-
troller) {
            self.showBasicAlert(on: vc, title: "Unknown Registration
Error", message: nil)
        }

        public static func showRegistrationErrorAlert(on vc: UIViewCon-
troller, with error: Error) {
            self.showBasicAlert(on: vc, title: "Unknown Registration
Error", message: "\(error.localizedDescription)")
        }
    }
```

```
    // MARK: - Log In Errors
    extension AlertManager {

        public static func showSignInErrorAlert(on vc: UIViewController) {
            self.showBasicAlert(on: vc, title: "Unknown Error Signing In",
message: nil)
        }

        public static func showSignInErrorAlert(on vc: UIViewController,
with error: Error) {
            self.showBasicAlert(on: vc, title: "Error Signing In", message:
"\(error.localizedDescription)")
        }
    }


    // MARK: - Logout Errors
    extension AlertManager {

        public static func showLogoutError(on vc: UIViewController, with
error: Error) {
            self.showBasicAlert(on: vc, title: "Log Out Error", message: "\
(error.localizedDescription)")
        }
    }


    // MARK: - Forgot Password
    extension AlertManager {

        public static func showPasswordResetSent(on vc: UIViewController) {
            self.showBasicAlert(on: vc, title: "Password Reset Sent", mes-
sage: nil)
        }

        public static func showErrorSendingPasswordReset(on vc: UIViewCon-
troller, with error: Error) {
```

```
            self.showBasicAlert(on: vc, title: "Error Sending Password Re-
set", message: "\(error.localizedDescription)")
        }
    }


    // MARK: - Fetching User Errors
    extension AlertManager {

        public static func showFetchingUserError(on vc: UIViewController,
with error: Error) {
            self.showBasicAlert(on: vc, title: "Error Fetching User", mes-
sage: "\(error.localizedDescription)")
        }

        public static func showUnknownFetchingUserError(on vc: UIViewCon-
troller) {
            self.showBasicAlert(on: vc, title: "Unknown Error Fetching
User", message: nil)
        }
    }

    // MARK: - Already has data error
    extension AlertManager {

        public static func showRepeatingDataError(
            on vc: UIViewController,
            with error: Error
        ) {
            self.showBasicAlert(
                on: vc,
                title: "Already here",
                message: "\(error.localizedDescription)"
            )
        }
    }
```

Файл RegisterController.swift

```
    import UIKit
```

```
final class RegisterController: UIViewController {

    // MARK: - UI Components
    private let headerView = AuthHeaderView(title: "Sign Up", subtitle: "Create your account")

    private let usernameField = CustomTextField(fieldType: .username)
    private let emailField = CustomTextField(fieldType: .email)
    private let passwordField = CustomTextField(fieldType: .password)

    private let signUpButton = CustomButton(title: "Sign Up", hasBackground: true, fontSize: .big, textColor: .label)
    private let signInButton = CustomButton(title: "Already have an account? Sign In.", hasBackground: false, fontSize: .medium, textColor: .systemYellow)

    private let termsTextView: UITextView = {
        let attributedString = NSMutableAttributedString(string: "By creating an account, you agree to our Terms & Conditions and you acknowledge that you have read our Privacy Policy.")

        attributedString.addAttribute(.link, value: "terms://termsAndConditions", range: (attributedString.string as NSString).range(of: "Terms & Conditions"))

        attributedString.addAttribute(.link, value: "privacy://privacyPolicy", range: (attributedString.string as NSString).range(of: "Privacy Policy"))

        let tv = UITextView()
        tv.linkTextAttributes = [.foregroundColor: UIColor.systemYellow]
        tv.backgroundColor = .clear
        tv.attributedText = attributedString
        tv.textColor = .label
        tv.isSelectable = true
        tv.isEditable = false
        tv.delaysContentTouches = false
        tv.isScrollEnabled = false
```

```
            return tv
        }()

        // MARK: - LifeCycle
        override func viewDidLoad() {
            super.viewDidLoad()
            setupUI()

            termsTextView.delegate = self

            signUpButton.addTarget(self, action: #selector(didTapSignUp),
for: .touchUpInside)
            signInButton.addTarget(self, action: #selector(didTapSignIn),
for: .touchUpInside)
        }

        override func viewWillAppear(_ animated: Bool) {
            super.viewWillAppear(animated)
            self.navigationController?.navigationBar.isHidden = true
        }

        // MARK: - UI Setup
        private func setupUI() {
            view.backgroundColor = .systemGray6

            view.addSubview(headerView)
            view.addSubview(usernameField)
            view.addSubview(emailField)
            view.addSubview(passwordField)
            view.addSubview(signUpButton)
            view.addSubview(termsTextView)
            view.addSubview(signInButton)

            headerView.translatesAutoresizingMaskIntoConstraints = false
            usernameField.translatesAutoresizingMaskIntoConstraints = false
            emailField.translatesAutoresizingMaskIntoConstraints = false
            passwordField.translatesAutoresizingMaskIntoConstraints = false
            signUpButton.translatesAutoresizingMaskIntoConstraints = false
            termsTextView.translatesAutoresizingMaskIntoConstraints = false
            signInButton.translatesAutoresizingMaskIntoConstraints = false
```

```
            NSLayoutConstraint.activate([
                self.headerView.topAnchor.constraint(equalTo: self-
.view.layoutMarginsGuide.topAnchor),
                headerView.leadingAnchor.constraint(equalTo: self-
.view.leadingAnchor),
                headerView.trailingAnchor.constraint(equalTo: self.view.-
trailingAnchor),
                headerView.heightAnchor.constraint(equalToConstant: 222),

                usernameField.topAnchor.constraint(equalTo: headerView.bot-
tomAnchor, constant: 12),
                usernameField.centerXAnchor.constraint(equalTo: header-
View.centerXAnchor),
                usernameField.heightAnchor.constraint(equalToConstant: 55),
                usernameField.widthAnchor.constraint(equalTo: view.width-
Anchor, multiplier: 0.85),

                emailField.topAnchor.constraint(equalTo: usernameField.bot-
tomAnchor, constant: 22),
                emailField.centerXAnchor.constraint(equalTo: headerView.-
centerXAnchor),
                emailField.heightAnchor.constraint(equalToConstant: 55),
                emailField.widthAnchor.constraint(equalTo: view.width-
Anchor, multiplier: 0.85),

                passwordField.topAnchor.constraint(equalTo: emailField.bot-
tomAnchor, constant: 22),
                passwordField.centerXAnchor.constraint(equalTo: header-
View.centerXAnchor),
                passwordField.heightAnchor.constraint(equalToConstant: 55),
                passwordField.widthAnchor.constraint(equalTo: view.width-
Anchor, multiplier: 0.85),

                signUpButton.topAnchor.constraint(equalTo: passwordField-
.bottomAnchor, constant: 22),
                signUpButton.centerXAnchor.constraint(equalTo: headerView.-
centerXAnchor),
                signUpButton.heightAnchor.constraint(equalToConstant: 55),
```

```
                signUpButton.widthAnchor.constraint(equalTo: view.width-
Anchor, multiplier: 0.85),

                termsTextView.topAnchor.constraint(equalTo: signUpBut-
ton.bottomAnchor, constant: 6),
                termsTextView.centerXAnchor.constraint(equalTo: header-
View.centerXAnchor),
                termsTextView.widthAnchor.constraint(equalTo: view.width-
Anchor, multiplier: 0.85),

                signInButton.topAnchor.constraint(equalTo: termsText-
View.bottomAnchor, constant: 11),
                signInButton.centerXAnchor.constraint(equalTo: headerView.-
centerXAnchor),
                signInButton.heightAnchor.constraint(equalToConstant: 44),
                signInButton.widthAnchor.constraint(equalTo: view.width-
Anchor, multiplier: 0.85),
            ])
        }

        // MARK: - Selectors
        @objc func didTapSignUp() {
            let registerUserRequest = RegisterUserRequest(
                username: self.usernameField.text ?? "",
                email: self.emailField.text ?? "",
                password: self.passwordField.text ?? ""
            )

            // Username check
            if !Validator.isValidUsername(for: registerUserRequest.user-
name) {
                AlertManager.showInvalidUsernameAlert(on: self)
                return
            }

            // Email check
            if !Validator.isValidEmail(for: registerUserRequest.email) {
                AlertManager.showInvalidEmailAlert(on: self)
                return
            }
```

```
            // Password check
            if !Validator.isPasswordValid(for: registerUserRequest.pass-
word) {
                AlertManager.showInvalidPasswordAlert(on: self)
                return
            }

            AuthService.shared.registerUser(with: registerUserRequest) {
                [weak self] wasRegistered,
                error in

                guard let self = self else { return }

                    if let error = error {
                    AlertManager.showRegistrationErrorAlert(on: self, with:
error)
                    return
                }

                if wasRegistered {
                    if let sceneDelegate =
self.view.window?.windowScene?.delegate as? SceneDelegate {
                        sceneDelegate.checkAuthentication()
                    }
                } else {
                    AlertManager.showRegistrationErrorAlert(on: self)
                }
            }
        }

        @objc private func didTapSignIn() {
            self.navigationController?.popToRootViewController(animated:
true)
        }

    }

    extension RegisterController: UITextViewDelegate {
```

```
        func textView(_ textView: UITextView, shouldInteractWith URL: URL,
in characterRange: NSRange, interaction: UITextItemInteraction) -> Bool {

            if URL.scheme == "terms" {
                self.showWebViewerController(with: "https://policies.-
google.com/terms?hl=en")
            } else if URL.scheme == "privacy" {
                self.showWebViewerController(with: "https://policies.-
google.com/privacy?hl=en")
            }

            return true
        }

        private func showWebViewerController(with urlString: String) {
            let vc = WebViewerController(with: urlString)
            let nav = UINavigationController(rootViewController: vc)
            self.present(nav, animated: true, completion: nil)
        }

        func textViewDidChangeSelection(_ textView: UITextView) {
            textView.delegate = nil
            textView.selectedTextRange = nil
            textView.delegate = self
        }
    }

    extension RegisterController: UITextFieldDelegate {

        override func touchesBegan(_ touches: Set<UITouch>, with event:
UIEvent?) {
            self.view.endEditing(true)
        }
    }
```

ПРИЛОЖЕНИЕ В
(обязательное)
Ведомость курсовой работы