МИНИСТЕРСТВО ОБРАЗОВАНИЯ НОВОСИБИРСКОЙ ОБЛАСТИ
ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ НОВОСИБИРСКОЙ ОБЛАСТИ
«НОВОСИБИРСКИЙ КОЛЛЕДЖ ЭЛЕКТРОНИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ»
(ГБПОУ НСО «НКЭиВТ»)

ОТЧЕТ

по учебной практике по профессиональному модулю
ПМ.01. Разработка модулей программного обеспечения для компьютерных систем

По специальности:
09.02.07 «Информационные системы и программирование»
В объеме 72 ч.

Выполнил(а): студент группы_____
_____
Проверил: преподаватель спец. Дисциплин
Усольцев А.А._____
Оценка:_____

Новосибирск 2023г.

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

В рамках практики было разработано приложение для заметок для платформы iOS. Целью практики было изучение основных принципов разработки приложений для iOS, а также применение полученных знаний на практике. В отчете будут представлены основные этапы разработки приложения, архитектура и дизайн приложения, используемые технологии, а также проблемы, с которыми столкнулись в процессе разработки и способы их решения. Также будет проведен обзор основных возможностей разработанного приложения, его функциональности и интерфейса. Кроме того, в отчете будет рассмотрены результаты тестирования. В заключение будут подведены основные итоги практики по разработке приложения для iOS и сформулированы принципиальные выводы по всем аспектам разработки.

1.1 Описание проекта

Проект мобильного приложения для заметок на iOS с взаимодействием с облачным хранилищем supabase включает в себя удобную организацию и хранение заметок для пользователей.

Первый экран приложения представляет собой список папок, а также кнопку для добавления новой папки. При выборе папки осуществляется переход на экран со списком заметок в данной папке. На этом экране также имеется кнопка для добавления новой заметки. При выборе заметки открывается экран с содержимым заметки, которое можно редактировать.

Основные функции приложения включают:

- Создание и управление папками для организации заметок.
- Создание, редактирование и удаление заметок внутри конкретной папки.
- Интуитивный и привлекательный интерфейс для удобства использования

Это приложение предлагает пользователям надежное и удобное средство для организации и хранения их заметок, обеспечивая безопасный доступ к данным в любое время и в любом месте.

2. ТЕХНИЧЕСКИЙ СТЕК РАЗРАБОТКИ

2.1 Среда разработки Xcode

Xcode - это интегрированная среда разработки (IDE), разработанная компанией Apple, предназначенная для создания приложений для операционных систем iOS, macOS, watchOS и tvOS. Xcode предоставляет разработчикам всю необходимую функциональность для написания, отладки и тестирования приложений для устройств Apple.

В Xcode доступны широкие возможности, такие как создание пользовательского интерфейса, написание кода на различных языках программирования (таких как Swift, Objective-C, C++), отладка, анализ производительности, тестирование и оптимизация приложений. В Xcode также доступен инструментарий для создания и управления локализацией, управления версиями и сборкой приложений.

Кроме того, Xcode предоставляет разработчикам доступ к различным инструментам разработки, таким как симуляторы устройств, интегрированная система сборки и непрерывной интеграции (CI/CD), автоматическое завершение кода, интеграция с графическими редакторами и другие инструменты для создания качественных приложений.

Таким образом, Xcode является основным инструментом для разработки приложений для платформ Apple, обеспечивая разработчикам все необходимые средства для создания инновационных и высококачественных программных продуктов.

2.1 Язык программирования swift

В данном проекте использовался язык программирования Swift. Swift - это мощный и интуитивно понятный язык программирования, разработанный компанией Apple специально для создания приложений под iOS, macOS, watchOS и tvOS. Он заменил Objective-C, стал более современным и удобным в использовании.

Swift обладает чистым синтаксисом, который делает его легко читаемым и понятным, что облегчает разработку и поддержание кода. Он также предоставляет

возможности безопасности типов, что помогает предотвратить ошибки во время выполнения.

Этот язык обеспечивает высокую производительность и включает в себя множество современных функций, таких как функциональное программирование, замыкания, обработка ошибок, расширения и многое другое. Благодаря этим возможностям, Swift позволяет разработчикам писать более чистый и эффективный код.

Swift также активно развивается, поэтому разработчики могут быть уверены, что он будет обновляться и развиваться, чтобы соответствовать современным требованиям и стандартам программирования. Этот язык стал основным выбором для многих разработчиков Apple и имеет широкую поддержку сообщества разработчиков.

2.3 Swift Concurrency

Swift Concurrency - это новый подход к работе с параллелизмом и асинхронным программированием в Swift. Он включает в себя новые инструменты, такие как async/await, structured concurrency и actors, которые позволяют разработчикам легче писать и поддерживать асинхронный код. Swift Concurrency упрощает управление параллелизмом и обеспечивает безопасность работы с данными в многопоточных средах, что делает разработку более эффективной и удобной.

2.4 Фреймворк swiftui

SwiftUI - это фреймворк для создания пользовательского интерфейса на платформах Apple, таких как iOS, macOS, watchOS и tvOS. Он предлагает новый декларативный подход к созданию интерфейса, позволяя разработчикам описывать структуру и внешний вид пользовательского интерфейса с помощью простых и интуитивно понятных кодовых конструкций. SwiftUI также поддерживает динамическое обновление интерфейса при изменении данных, что делает его более эффективным и удобным для использования.

2.5 Архитектура mvvm

Архитектура MVVM (Model-View-ViewModel) в iOS разработке представляет собой концепцию, которая помогает разделить приложение на три основных компонента - Model (модель данных), View (представление) и ViewModel (представление модели).

Модель отвечает за представление данных, бизнес-логику или работу с сетью. Представление обычно отображает данные и реагирует на действия пользователя. ViewModel выступает в качестве посредника между моделью и представлением, он предоставляет данные из модели для отображения на представлении, а также обрабатывает действия пользователя и взаимодействует с моделью для получения необходимых данных.

MVVM помогает разделить бизнес-логику и отображение данных, что делает код более чистым, модульным и удобным для тестирования. Кроме того, это позволяет лучше управлять состоянием представлений и уменьшить связность между различными частями кода.

2.6 Supabase

Supabase - это open-source платформа для создания и масштабирования приложений, которая предлагает полный стек инструментов для разработки с использованием PostgreSQL. Supabase предоставляет возможности для работы с базами данных, аутентификации, хранилищем файлов, в реальном времени и другими функциями, которые помогают разработчикам быстро создавать веб-приложения и мобильные приложения.

Supabase позиционируется как открытая платформа с гибкой и расширяемой архитектурой, которая позволяет разработчикам ускорить процесс разработки и развертывания приложений. Также она предлагает удобный интерфейс управления, дружелюбное API и поддержку различных технологий и фреймворков, что делает ее привлекательным выбором для создания современных приложений.

3. ПРИНЦИП РАБОТЫ ПРИЛОЖЕНИЯ

3.1 Список папок (ListFoldersView)

На экране отображается список папок (см. Рис. 1). Если пользователь сделает свайп влево с правого края по определенной папке, то появится кнопка "Удалить", которая позволит удалить эту папку, и также появится кнопка «Редактировать" (см. Рис. 2), с помощью которой можно будет изменить название папки (см. Рис. 7). В правом верхнем углу экрана расположена кнопка для добавления новой папки. При нажатии на эту кнопку появится возможность ввести название новой папки и сохранить ее (см. Рис. 3).

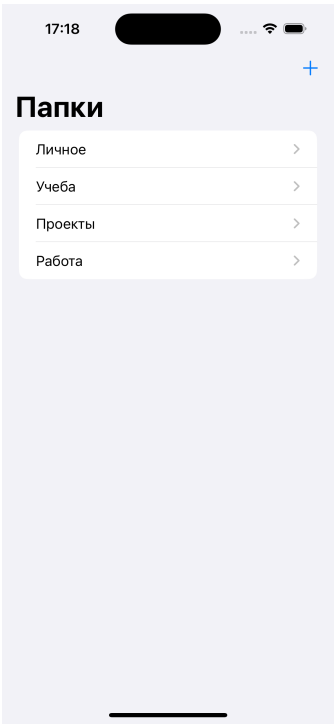


Рис. 1. Экран списка папок (ListFoldersView)

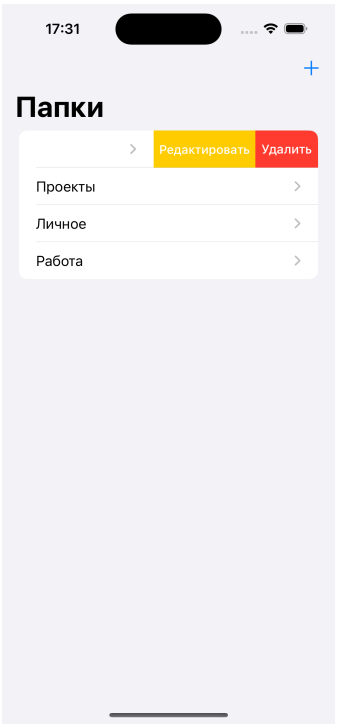


Рис. 2. Кнопки редактировать и удалить на экране ListFoldersView

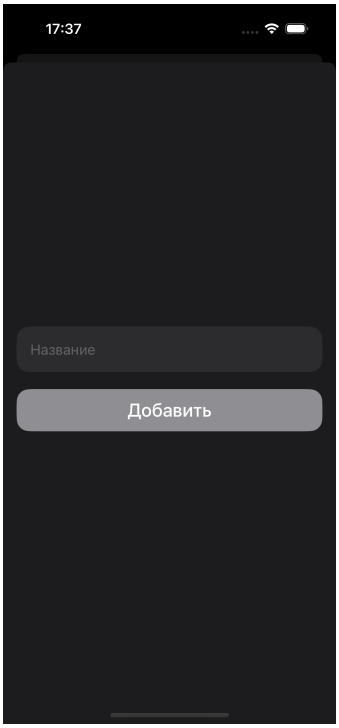


Рис. 3. Экран создания папки или заметки (Темная тема)а

Код представляет собой интерфейс приложения, в котором отображаются папки с заметками. Условие проверяет, загружаются ли данные в данный момент. Если загрузка идет, то отображается индикатор ProgressView. Если данные загружены и список папок не пустой, отображается список List, в котором отображаются папки, иначе отображается сообщение о том, что пользователь еще не создал ни одной папки. В List мы показываем NavigationLink для каждой папки.

NavigationLink: Создает ссылку для навигации внутри приложения на FolderView для конкретной папки. Ниже определяюся действия с помощью swipeActions, которые будут отображаться при свайпе по папке. При свайпе влево по папке, появятся кнопки для удаления или редактирования папки. Sheet отвечает за отображение модального окна для редактирования названия папки. Refreshable создает эффект pull-to-refresh, который позволяет пользователю обновить список папок. Toolbar добавляет кнопку в верхний правый угол экрана, которая добавляет новую папку. Метод onAppear вызывается при появление экрана, в котором мы загружаем информации о папках из облачного хранилища (см. Рис. 4).

```
NavigationStack {
    ZStack {
        if viewModel.isLoading {
           ProgressView()
        } else if !viewModel.folders.isEmpty {
            List {
                ForEach(viewModel.sortedFolders) { folder in
                    NavigationLink(folder.title) {
                        FolderView(
                            folder: folder,
                            service: service
                        )
                    }
                    .swipeActions(edge: .trailing, allowsFullSwipe: true) {
                        Button("Удалить") {
                            viewModel.deleteFolder(id: folder.id)
                        }
                        .tint(Color.red)

                        Button("Редактировать") {
                            self.showEditFolderSheet.toggle()
                        }
                        .tint(Color.yellow)
                    }
                    .sheet(isPresented: $showEditFolderSheet, content: {
                        EditEntitySheet(text: folder.title) { text in
                            viewModel.updateTitleFolder(newTitle: text, id: folder.id)
                        }
                    })
                }
            }
            .refreshable {
                await viewModel.asyncFetchFolders()
            }
        } else {
            noDataMessageView
        }
    }
    .onAppear {
        viewModel.fetchFolders()
    }
    .toolbar {
        ToolbarItem(placement: .topBarTrailing) {
            Button(action: {
                self.showAddFolderSheet.toggle()
            }, label: {
                Image(systemName: "plus")
            })
        }
    }
```

Рис. 4. Код экрана ListFoldersView

3.1 Экран папки (FolderView)

На экране отображается список заметок в папке (см. Рис. 5). Если пользователь сделает свайп влево с правого края по определенной заметке, то появится кнопка "Удалить", которая позволит удалить эту заметку, и также появится кнопка «Редактировать» (см. Рис. 6), с помощью которой можно будет изменить название заметки или ее содержимое (см. Рис. 7). В правом верхнем углу экрана расположена кнопка для добавления новой заметки. При нажатии на эту кнопку появится возможность ввести название новой заметки и сохранить ее (см. Рис. 3).

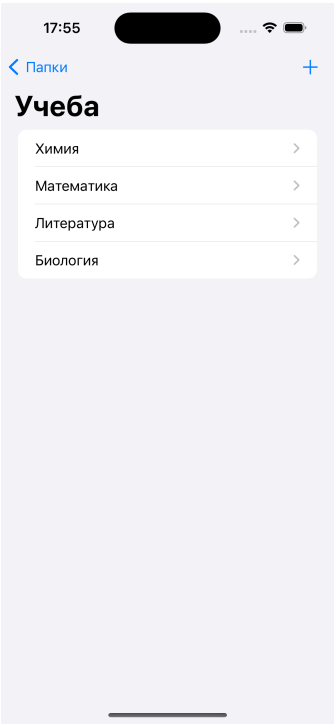


Рис. 5. Экран папки (FolderView)

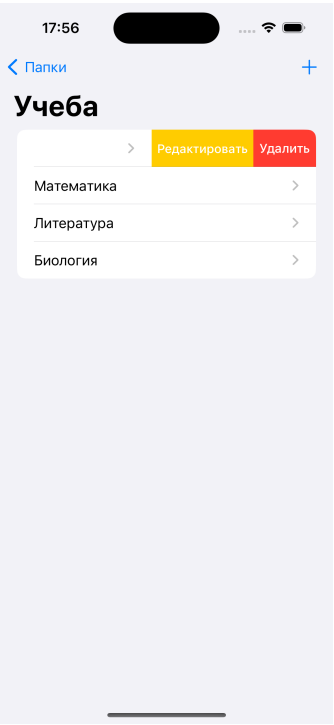


Рис. 6. Кнопки редактировать и удалить на экране FolderView

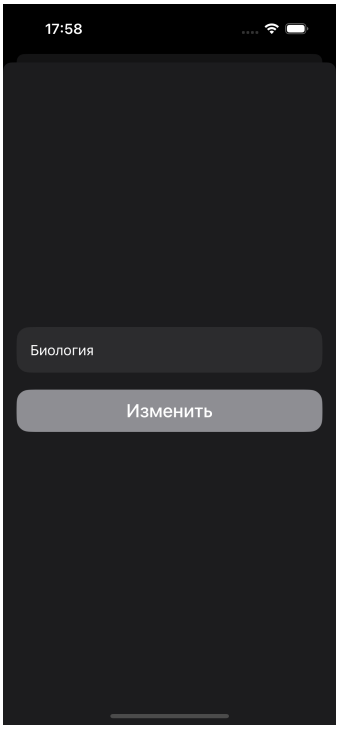


Рис. 7. Экран редактирования заметки или папки (Темная тема)

Код представляет собой интерфейс приложения, в котором отображаются заметки в папке. Условие проверяет, загружаются ли данные в данный момент. Если загрузка идет, то отображается индикатор ProgressView. Если данные загружены и список заметок не пустой, отображается список List, в котором отображаются заметки, иначе отображается сообщение о том, что пользователь еще не создал ни одной заметки. В List мы показываем NavigationLink для каждой заметки. NavigationLink: Создает ссылку для навигации внутри приложения на NoteView для

конкретной заметки. Ниже определяюся действия с помощью swipeActions, которые будут отображаться при свайпе по заметки. При свайпе влево по заметки, появятся кнопки для удаления или редактирования заметки. Sheet отвечает за отображение модального окна для редактирования названия заметки. Refreshable создает эффект pull-to-refresh, который позволяет пользователю обновить список заметок. Toolbar добавляет кнопку в верхний правый угол экрана, которая добавляет новую заметку. Метод onAppear вызывается при появление экрана, в котором мы загружаем информации о заметок из облачного хранилища (см. Рис. 8).

```
ZStack {
    if viewModel.isLoading {
        ProgressView()
    } else if !viewModel.notes.isEmpty {
        List {
            ForEach(viewModel.sortedNotes) { note in
                NavigationLink(note.title) {
                    NoteView(note: note, noteViewModel: viewModel) { text, id in
                        viewModel.updateNote(text: text, id: id)
                    }
                }
                .swipeActions(edge: .trailing, allowsFullSwipe: true) {
                    Button("Удалить") {
                        viewModel.deleteNote(id: note.id)
                    }
                    .tint(Color.red)

                    Button("Редактировать") {
                        self.showEditNoteSheet.toggle()
                    }
                    .tint(Color.yellow)
                }
                .sheet(isPresented: $showEditNoteSheet, content: {
                    EditEntitySheet(text: note.title) { newTitle in
                        viewModel.updateTitleNote(newTitle: newTitle, id: note.id)
                    }
                })
            }
        }
    } else {
        noDataView
    }
}
.onAppear {
    if viewModel.notes.isEmpty {
        viewModel.fetchNotes()
    }
}
.popover(isPresented: $showAddNoteSheet, content: {
    CreateEntitySheet { title in
        viewModel.addNote(title: title)
    }
    .presentationCompactAdaptation(.sheet)
})
.toolbar {
    ToolbarItem(placement: .topBarTrailing) {
        Button(action: {
            self.showAddNoteSheet.toggle()
        }, label: {
            Image(systemName: "plus")
        })
    }
}
```

Рис. 8. Код экрана FolderView

3.3 Экран заметки (NoteView)

На экране вверху посередине отображается название заметки. Ниже расположено большое поле для редактирования заметки, с возможностью добавления текста. Это поле позволяет пользователям свободно создавать, редактировать и организовывать свои заметки, делая их более удобными для использования.

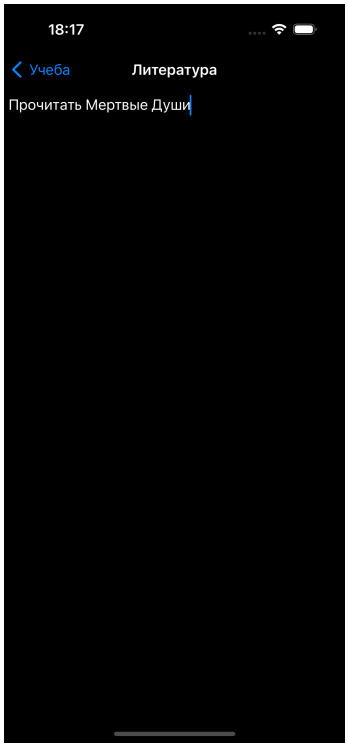


Рис. 9. Экран заметки (NoteView)

```
TextEditor(text: $viewModel.text)
    .textInputAutocapitalization(.never)
    .autocorrectionDisabled()
    .navigationTitle(note.title)
    .navigationBarTitleDisplayMode(.inline)
    .toolbar {
        ToolbarItem(placement: .topBarTrailing) {
            if noteViewModel.isLoading {
                ProgressView()
            }
        }
    }
    .focused($textEditorFocus)
    .onAppear {
        textEditorFocus = true
    }
```

Рис. 10. Код экрана заметки (NoteView)

TextEditor - это компонент, который отображает поле для редактирования текста. Он привязан к viewModel.text, что позволяет отображать и изменять текст заметки. С помощью textInputAutocapitalization(.never) мы отключаем автоматическое использование заглавных букв, а с помощью autocorrectionDisabled() отключаем автокоррекцию. NavigationTitle(note.title) - устанавливает заголовок экрана, используя название заметки из note.title. В onAppear устанавливаем фокус на TextEditor (см. Рис. 10).

4. ТЕСТИРОВАНИЕ

4.1 Тестирование DataService

В проекте тестирование было проведено с помощью unit тестирования, которое представляет собой тестирование отдельных компонентов программного обеспечения. Каждая единица кода, такая как функции, методы и классы, была протестирована отдельно от остальных частей программы.

Для проведения unit тестов были написаны специальные тестовые случаи, которые проверяли корректность работы каждого компонента. Например, для функции, которая меняла заголовок у папки, были написаны тестовые случаи для проверки корректного заголовка, пустого заголовка.

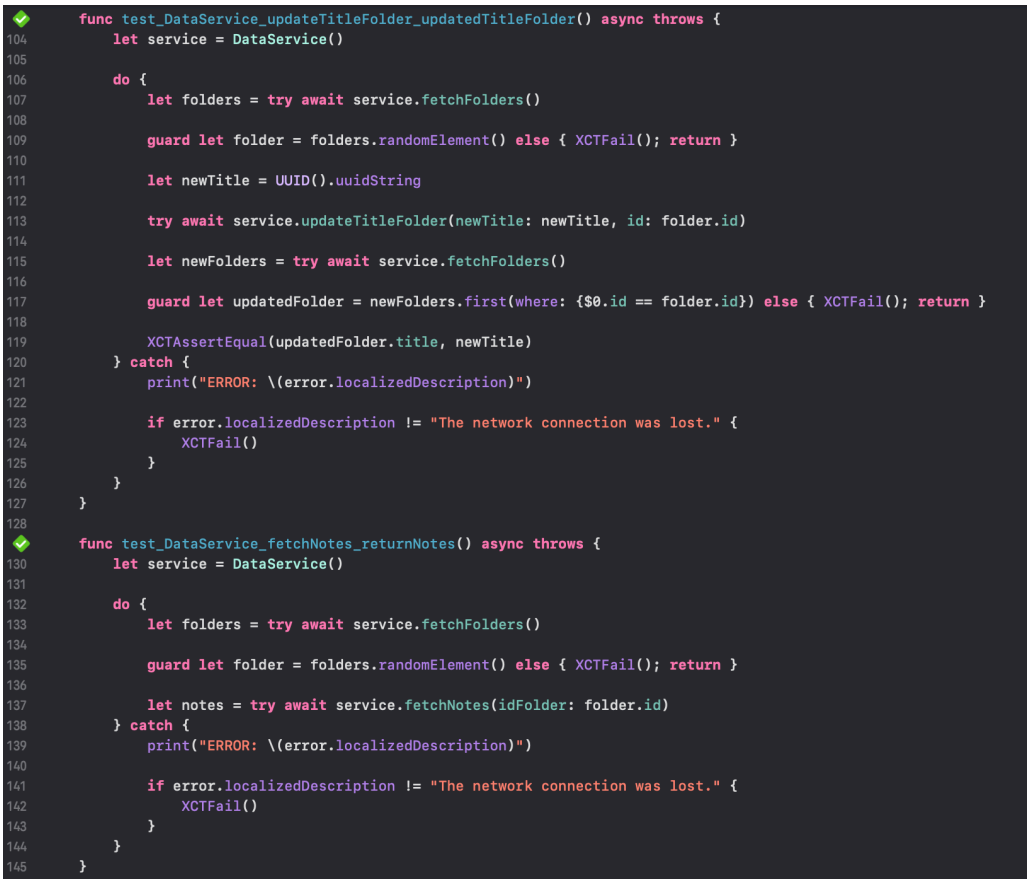
В результате проведения unit тестирования было обнаружено несколько ошибок и недочетов, которые были исправлены до выпуска программы. Таким образом, использование unit тестов позволило повысить качество программного обеспечения и обеспечить его надежную работу.

```
37      func test_DataService_fetchFolders_returnFolders() async throws {
38          let service = DataService()
39
40          let folders = try await service.fetchFolders()
41
42          XCTAssertFalse(folders.isEmpty)
43      }
45
46      func test_DataService_addFolder_addedFolder() async throws {
47          let service = DataService()
48
49          let name = "test folder"
50
51          let folder = FolderModel(title: name)
52
53          do {
54              try await service.addFolder(folder: folder)
55
56              XCTAssertTrue(true)
57          } catch  {
58              if error.localizedDescription != "The network connection was lost." {
59                  print(error.localizedDescription)
60                  XCTFail()
61              }
62          }
      }
```

Рис. 11. Тесты для функций fetchFolders и addFolder в DataService

```swift
    func test_DataService_addFolder_notAddEmptyString() async throws {
        let service = DataService()

        let emptyString = ""

        let folder = FolderModel(title: emptyString)

        do {
            try await service.addFolder(folder: folder)

            XCTFail()
        } catch {

        }
    }

    func test_DataService_deleteFolder_deletedFolder() async throws {
        let service = DataService()

        do {
            let folders = try await service.fetchFolders()

            guard let folder = folders.randomElement() else { XCTFail(); return }

            try await service.deleteFolder(id: folder.id)

            print("Deleted: \(folder.id)")

            let newFolders = try await service.fetchFolders()

            XCTAssertGreaterThan(folders.count, newFolders.count)
        } catch {
            print("ERROR: \(error.localizedDescription)")

            if error.localizedDescription != "The network connection was lost." {
                XCTFail()
            }
        }
    }
```

Рис. 12. Тесты для функций addFolder и deleteFolder в DataService



```
func test_DataService_updateTitleFolder_updatedTitleFolder() async throws {
    let service = DataService()

    do {
        let folders = try await service.fetchFolders()

        guard let folder = folders.randomElement() else { XCTFail(); return }

        let newTitle = UUID().uuidString

        try await service.updateTitleFolder(newTitle: newTitle, id: folder.id)

        let newFolders = try await service.fetchFolders()

        guard let updatedFolder = newFolders.first(where: {$0.id == folder.id}) else { XCTFail(); return }

        XCTAssertEqual(updatedFolder.title, newTitle)
    } catch {
        print("ERROR: \(error.localizedDescription)")

        if error.localizedDescription != "The network connection was lost." {
            XCTFail()
        }
    }
}

func test_DataService_fetchNotes_returnNotes() async throws {
    let service = DataService()

    do {
        let folders = try await service.fetchFolders()

        guard let folder = folders.randomElement() else { XCTFail(); return }

        let notes = try await service.fetchNotes(idFolder: folder.id)
    } catch {
        print("ERROR: \(error.localizedDescription)")

        if error.localizedDescription != "The network connection was lost." {
            XCTFail()
        }
    }
}
```

Рис. 13. Тесты для функций updateTitleFolder и fetchNotes в DataService

```
func test_DataService_addNote_addedNote() async throws {
    let service = DataService()

    do {
        let folders = try await service.fetchFolders()

        guard let folder = folders.randomElement() else { XCTFail(); return }

        let title = UUID().uuidString

        let note =  NoteModel(title: title, idFolder: folder.id)

        try await service.addNote(note: note)
    } catch {
        print("ERROR: \(error.localizedDescription)")

        if error.localizedDescription != "The network connection was lost." {
            XCTFail()
        }
    }
}
```

Рис. 14. Тесты для функции addNote в DataService

```swift
func test_DataService_deleteNote_deletedNote() async throws {
    let service = DataService()

    do {
        let folders = try await service.fetchFolders()

        guard let folder = folders.randomElement() else { XCTFail(); return }

        let notes = try await service.fetchNotes(idFolder: folder.id)

        guard let note = notes.randomElement() else { XCTFail(); return }

        try await service.deleteNote(id: note.id)

        print("Deleted note: title - \(note.title), idFolder - \(folder.id)")
    } catch {
        print("ERROR: \(error.localizedDescription)")

        if error.localizedDescription != "The network connection was lost." {
            XCTFail()
        }
    }
}
```

Рис. 15. Тесты для функции deleteNote в DataService

```swift
func test_DataService_updateTitleNote_updateTitleNote() async throws {
    let service = DataService()

    do {
        let folders = try await service.fetchFolders()

        guard let folder = folders.randomElement() else { XCTFail(); return }

        let note = NoteModel(title: "old title", idFolder: folder.id)

        try await service.addNote(note: note)

        let notes = try await service.fetchNotes(idFolder: folder.id)

        let newTitle = UUID().uuidString

        try await service.deleteNote(id: note.id)

        print("Updated note: new title - \(newTitle), idFolder - \(folder.id)")

        try await service.updateTitleNote(newTitle: newTitle, id: note.id)
    } catch {
        print("ERROR: \(error.localizedDescription)")

        if error.localizedDescription != "The network connection was lost." {
            XCTFail()
        }
    }
}
```

Рис. 16. Тесты для функции updateTitleNote в DataService

```swift
func test_DataService_updateNote_updatedNote() async throws {
    let service = DataService()

    do {
        let folders = try await service.fetchFolders()

        guard let folder = folders.randomElement() else { XCTFail(); return }

        let note = NoteModel(title: "test title", text: "old text", idFolder: folder.id)

        try await service.addNote(note: note)

        let notes = try await service.fetchNotes(idFolder: folder.id)

        let newText = UUID().uuidString

        try await service.updateNote(text: newText, id: note.id)

        print("Updated note: new text - \(newText), idFolder - \(folder.id)")

        let newNotes = try await service.fetchNotes(idFolder: folder.id)

        guard let updatedNote = newNotes.randomElement() else { XCTFail(); return }

        XCTAssertEqual(updatedNote.text, newText)
    } catch {
        print("ERROR: \(error.localizedDescription)")

        if error.localizedDescription != "The network connection was lost." {
            XCTFail()
        }
    }
}
```

Рис. 17. Тесты для функции updateNote в DataService

5. ЗАКЛЮЧЕНИЕ

В ходе практики было разработано приложение для заметок на языке Swift с использованием фреймворка SwiftUI. Процесс разработки приложения был успешно завершен, и были достигнуты следующие результаты

В приложении была реализована основная функциональность, включая создание, редактирование и удаление заметок.

Использование SwiftUI позволило создать современный и интуитивно понятный пользовательский интерфейс, что сделало приложение удобным в использовании.

Были применены базовые принципы разработки на Swift, такие как использование классов, структур, протоколов и расширений, что способствует более чистому и модульному коду.

Кроме того, в ходе работы над проектом экспериментировали с различными техниками тестирования, включая unit тестирование, что позволило повысить качество кода и обнаружить и исправить возможные ошибки до релиза приложения.

В целом, опыт работы над проектом по разработке приложения для заметок на Swift с использованием SwiftUI был полезным и позволил приобрести ценный опыт в области мобильной разработки.

6. ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ

1. Neil Smyth. SwiftUI Essentials: iOS 14 Edition: Learn to Develop iOS Apps Using SwiftUI, Swift 4 and Xcode 12: практическое пособие по разработке мобильных приложений для iOS с использованием SwiftUI, Swift 4 и Xcode 12 / Neil Smyth. - Chicago, May 2021. - 526 с.
2. Kodeco Inc. SwiftUI by Tutorials (Fifth Edition): Declarative App Development on the Apple Ecosystem: практические пособие / Kodeco Inc. - Los Angeles, June 13, 2023. - 748с.
3. Jayant Varma. SwiftUI for Absolute Beginners - Program Controls and Views for iPhone, iPad, and Mac Apps: практическое пособие / Jayant Varma. - Melbourne, Australia, April 25, 2021. - 183с.
4. Официальная документация по Swift - URL: htttp///