содержание

ВВЕДЕНИЕ.

1 АНАЛИЗ ПОСТАНОВКИ ЗАДАЧИ 5

2 ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПРИЛОЖЕНИЯ 7

3 РАЗРАБОТКА АЛГОРИТМОВ ФУНКЦИОНИРОВАНИЯ И СТРУКТУР ДАННЫХ 9

3.1 Основные функции 9

3.2 Разработка алгоритмов приложения 9

4 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ И РЕЗУЛЬТАТЫ ИСПЫТАНИЙ 12

4.1 Определение требований к техническим средствам 12

4.2 Определение формы представления входных и выходных данных 12

4.3 Проектирование программного обеспечения 12

4.4 Спецификация программы 14

4.5 Тестирование 16

ЗАКЛЮЧЕНИЕ 20

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 21

ПРИЛОЖЕНИЕ А ТЕКСТ ПРОГРАММЫ

ПРИЛОЖЕНИЕ Б ГРАФИЧЕСКИЙ МАТЕРИАЛ

­­

# ВВЕДЕНИЕ.

Любая организация нуждается в своевременном доступе к информации. Ценность информации в современном мире очень высока. Роль распорядителей информации в современном мире чаще всего выполняют базы данных. Базы данных обеспечивают надежное хранение информации, структурированном виде и своевременный доступ к ней. Практически любая современная организация нуждается в базе данных, удовлетворяющей те или иные потребности по хранению, управлению и администрированию данных.

В ходе выполнения данного курсового проекта необходимо разработать информационную базу данных для университета, которая поможет любому пользователю легко найти нужную информацию о времени, кабинете и преподавателе текущей пары.

Мощность базы данных обусловлена возможностью ее постоянного пополнения новыми данными, причем в неограниченном количестве информации. Это является очень удобным для пользователя. Таким образом, создание базы данных, обладающей такими свойствами, задача достаточно актуальная и полезная.

Разработка мобильного приложения "Расписание" - программа, предназначенная для быстрого информирования студентов. Покажет расписание на текущий день и расписание на всю неделю, чтобы студенты всегда знали, где и в какое время они должны быть.

# АНАЛИЗ ПОСТАНОВКИ ЗАДАЧИ.

Для реализации этой задачи использовался язык программирования Swift и фреймворк Swift UI, среда разработки Xcode. В основном Swift используют для создания очень быстрых и высокопроизводительных приложений. Это мессенджеры, игры или сложные ресурсоемкие программы. Он популярен в сферах, имеющих повышенные требования к безопасности ПО.

Swift — это надёжный и интуитивно понятный язык программирования от Apple, при помощи которого можно создавать приложения для iOS, Mac, Apple TV и Apple Watch. Он предоставляет разработчикам небывалую свободу творчества. Благодаря этому простому и удобному языку с открытым кодом вам достаточно просто интересной идеи, чтобы создать нечто невероятное.

Потенциал Swift оценили учебные заведения по всему миру. Они используют Swift и Xcode на Mac для обучения, обеспечивая своих студентов лучшими инструментами для создания приложений. А с бесплатной учебной программой от Apple «Разработка на языке Swift» даже новички легко переходят от основ программирования к профессиональной разработке.

Анализ постановки задачи для разработки мобильного приложения:

1. Определить цели и задачи приложения:

- Упростить процесс просмотра расписания занятий для студентов

- Увеличить эффективность использования времени студентов

- Уменьшить количество пропущенных занятий

- Предоставить возможность получения уведомлений о предстоящих занятиях

2. Определить категории пользователей приложения:

- Студенты

3. Определить функциональные требования к приложению:

- Отображение расписания занятий в удобном формате

- Возможность выбора группы студентов или факультета/кафедры

- Возможность просмотра расписания занятий для каждого дня недели

- Возможность получения уведомлений о предстоящих занятиях за 10/30 минут до начала занятия

4. Определить нефункциональные требования к приложению:

- Простой и интуитивно понятный интерфейс

- Быстрый доступ к основным функциям приложения

- Стабильная работа приложения без ошибок

5. Определить технологии, которые будут использоваться для разработки приложения:

- Язык программирования Swift для iOS

- Фреймворк SwiftUI для создания пользовательского интерфейса

- База данных для хранения расписания

# 2 ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПРИЛОЖЕНИЯ

Проектирование структуры мобильного приложения "Расписание" предполагает разработку архитектуры и организацию взаимодействия компонентов приложения:

1. Компоненты приложения:

- Представление (View) - отвечает за визуальное отображение пользовательского интерфейса приложения. В данном случае использован SwiftUI.

- Контроллеры (Controllers) - отвечают за обработку действий пользователя и управление бизнес-логикой приложения. В коде представлен класс ScheduleApp.

- Модель (Model) - отвечает за хранение и обработку данных. В коде это структура Lesson.

- Сетевой слой (Networking) - отвечает за общение с удаленным сервером и получение/отправку данных. В коде это не представлено.

2. Архитектура приложения:

- MVC (Model-View-Controller) - паттерн проектирования, в котором приложение разбивается на три компонента: модель (Model), представление (View) и контроллеры (Controllers).

- В данном случае, класс ScheduleApp выполняет функции контроллера, структура Lesson - модели, а SwiftUI - представления.

- Приложение использует локальное хранилище данных, что позволяет быстро получать доступ к данным без использования сети.

- Для оповещения пользователя о ближайшем уроке используется система уведомлений, реализованная с помощью UserNotifications.

Для разработки мобильного приложения "Расписание" была выбрана архитектура Model-View-ViewModel (MVVM), которая разделяет компоненты приложения на три основных слоя: модель (Model), представление (View) и модель представления (ViewModel). Каждый слой выполняет определенную функцию:

- Модель представляет данные и бизнес-логику приложения.

- Представление отвечает за отображение данных и интерфейс пользователя.

- Модель представления связывает модель и представление, обеспечивая коммуникацию между ними.

В приложении "Расписание" модель содержит данные о расписании занятий, включая название предмета, номер аудитории и время начала. Представление реализовано в виде пользовательского интерфейса, позволяющего пользователю просматривать расписание занятий. Модель представления управляет взаимодействием между моделью и представлением, обеспечивая отображение данных на экране.

Структура приложения также включает в себя следующие компоненты:

- Файл "ScheduleApp.swift", содержащий запуск приложения.

- Файл ContentView.swift", содержащий выбор факультета, курса, группы и подгруппы.

- Файл "NextView.swift", содержащий определение представления для просмотра вкладок приложения.

- Файл "TodayView.swift", “ScheduleView.swift”, “NewsWebView.swift” содержат определение представления для отображения списка занятий на сегодняшний день, расписание занятий на всю неделю и новости университета.

Таким образом, структура приложения разделена на три слоя MVVM, а также включает в себя необходимые компоненты для отображения данных о занятиях.

# 3 РАЗРАБОТКА АЛГОРИТМОВ ФУНКЦИОНИРОВАНИЯ И СТРУКТУР ДАННЫХ

**3.1 Основные функции**

В данном курсовом проекте используются следующие функции:

1. Отображение расписания в удобном виде: приложение позволяет просматривать расписание занятий в разных форматах, включая список занятий на текущий день и список занятий на всю неделю.

2. Уведомления: приложение отправляет пользователю уведомления о предстоящих занятиях за определенное количество времени до начала занятия.

3. Отображение новостей: приложение позволяет просматривать новости университета, загружая их с официального сайта.

**3.2 Разработка алгоритмов приложения**

Алгоритм отображения расписания:

1. Загрузка расписания из базы данных, которая находится на сервере.

2. Проверка наличия необходимых полей для каждого вида отображения (например, дата и время начала, название предмета, аудитория и т.д.).

3. Выбор формата отображения в зависимости от настроек пользователя или от сценария использования.

4. Отображение расписания в виде списка.

5. Обновление расписания при необходимости.

6. Обработка действий пользователя (например, нажатие на элемент расписания для просмотра дополнительной информации, переход на другую вкладку и т.д.).

7. Обработка ошибок и исключительных ситуаций (например, отсутствие подключения к интернету, неверный формат данных или невозможность отобразить расписание в выбранном формате).

Алгоритм отправки уведомлений:

1. Запросить разрешение на отправку уведомлений пользователю, используя класс UNUserNotificationCenter.

2. Если разрешение было предоставлено, удалить все предыдущие запросы уведомлений с помощью метода removeAllPendingNotificationRequests() класса UNUserNotificationCenter.

3. Для каждого события в расписании создать объект уведомления с помощью класса UNMutableNotificationContent. Задать заголовок, текст сообщения и звук, который будет проигрываться при получении уведомления.

4. Определить время, когда нужно отправить уведомление, на основе времени начала события и времени, за которое нужно отправить уведомление.

5. Создать объект триггера с помощью класса UNCalendarNotificationTrigger, используя определенное время.

6. Создать объект запроса уведомления с помощью класса UNNotificationRequest, используя созданный объект уведомления и объект триггера.

7. Добавить объект запроса уведомления в центр уведомлений с помощью метода add() класса UNUserNotificationCenter.

Дополнительно, алгоритм может включать проверку правильности формата времени начала события и времени отправки уведомления, а также обработку возможных ошибок при отправке уведомлений.

Разработка структуры базы данных — это важный этап в создании приложения, поскольку хранение и управление данными является одной из его основных функций. Описание структуры базы данных должно включать в себя:

1. Название базы данных: каждая база данных должна иметь уникальное название, которое отражает ее назначение.

2. Таблицы: определение таблиц, которые будут использоваться для хранения данных. Каждая таблица должна иметь уникальное имя, которое отражает ее содержимое.

3. Поля: определение полей, которые будут храниться в каждой таблице. Каждое поле должно иметь уникальное имя и тип данных, который определяет, какие типы значений могут быть сохранены в поле.

4. Отношения: определение связей между таблицами. Отношения могут быть один-к-одному, один-ко-многим или многие-ко-многим. Каждая связь должна иметь уникальное имя и указывать на связанные поля в каждой таблице.

5. Индексы: создание индексов для ускорения поиска данных. Индексы создаются на основе одного или нескольких полей в таблицах.

6. Безопасность: определение прав доступа к базе данных и таблицам. Необходимо установить права доступа для каждого пользователя, чтобы обеспечить безопасность данных и предотвратить несанкционированный доступ.

# 4 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ И РЕЗУЛЬТАТЫ ИСПЫТАНИЙ

**4.1 Определение требований к техническим средствам**

Разрабатываемое приложение должно работать на всех ios - устройствах с версией ПО 15.0 и выше.

Программное обеспечение включает в себя комплекс системных, прикладных и диагностических программных средств, необходимых для разработки и эксплуатации проектируемой игры.

К программному обеспечению диагностики и контроля относятся вспомогательные программы (утилиты) для выполнения различных сервисных функций, таких как проверка системы на вирусы, дефрагментация дискового пространства и т.д.

**4.2 Определение формы представления входных и выходных данных**

От правильной реализации диалога пользователя с системой во многом зависит успех системы, а также эффективность ее использования. При разработке системы к диалогу предъявляются следующие требования: требование гибкости диалога. Гибкость диалога — это мера того, насколько хорошо он соответствует различным уровням подготовки и производительности труда пользователя требование естественности диалога. Естественность — это свойство диалога, при наличии которого пользователь не изменяет свои традиционные способы решения. Немаловажен также порядок диалога. Всегда следует придерживаться такого порядка, в котором пользователь обычно обрабатывает информацию. Требование к последовательности диалога. Последовательность диалога гарантирует, что пользователь, освоивший работу с одной частью системы, освоит работу с другими ее частями. Данное требование к диалогу состоит из последовательности использования форматов данных, последовательности в размещении данных на экране и последовательности при выделении информации.

**4.3 Проектирование программного обеспечения**

**Реализованные пользовательские классы:­­**

ScheduleApp.swift– запуск программы;

DatabaseHelper.swift – файл соединения с базой данных;

ContentView.swift – файл, содержащий выбор факультета, курса, группы и подгруппы;

NextView.swift – содержит все складки приложения;

TodayView.swift – вкладка с текущим расписанием;

ScheduleView.swift – вкладка с полным расписанием;

NewsWebView.swift – вкладка с новостями университета;

**Реализованные пользовательские процедуры:**

func getCurrentWeekName() -> String

func getAutoWeekType() -> String

func loadLessons()

func scheduleNotifications()

func typeText() -> String

func getLessonsForWeekday(weekday: Weekday) -> [Lesson]

func makeUIView(context: Context) -> WKWebView

func connectToDatabase() -> OpaquePointer?

func downloadDatabase()

**Реализованная пользовательская база данных:**

Была разработана база данных "university.db" с двумя таблицами: "groups" и "schedule". Таблица "groups" содержит два поля: "id" и "name". Поле "id" является первичным ключом таблицы, а поле "name" хранит название группы.

Таблица "schedule" содержит следующие поля:

- "id": первичный ключ таблицы, автоинкрементируемое целое число.

- "group\_name": название группы, связанное с таблицей "group".

- "time": время начала занятия.

- "sub\_group\_name": название подгруппы, если она есть.

- "subject\_name": название предмета.

- "day\_of\_week": день недели, на котором проходит занятие.

- "week\_type": тип недели

- "room\_number": номер аудитории.

- "subject\_type": тип занятия (лекция/практика/лабораторная).

- "teacher": преподаватель, ведущий занятие.

Таблица "schedule" связана с таблицей "group" по полю "group\_name". Связь между таблицами является один-к-многим: одной группе может соответствовать множество занятий.

Все поля таблицы "schedule" имеют соответствующие ограничения на типы данных и не допускают NULL-значений. Для таблицы "group" поле "id" также не допускает NULL-значений, а поле "name" имеет ограничение UNIQUE для предотвращения дубликатов.

Была выбрана SQLite в качестве СУБД, так как она легковесная и хорошо подходит для небольших проектов, таких как мобильные приложения. Требования к настройке и безопасности базы данных описаны в соответствующем разделе технического задания.

**4.4 Спецификация программы.**

Запуск приложения:

struct ScheduleApp: App {

var body: some Scene {

WindowGroup {

ContentView()

}

}

}

Экран с опциями приложения:

var body: some View {

NavigationView {

VStack {

Text("Расписание")

.font(.custom("Zaychik", size: 65))

.padding(EdgeInsets(top: 100, leading: 0, bottom: 40, trailing: 0))

Spacer()

// Dropdown for faculty selection

Picker("Faculty", selection: $selectedFaculty) {

ForEach(faculties, id: \.self) {

Text($0)

}

}

……

Все окна приложения:

var body: some View {

TabView(selection: $selectedTab) {

// Вкладка с контентом

TodayView(selectedGroup: selectedGroup, selectedSubGroup: selectedSubGroup)

.tabItem {

Image(systemName: "star")

Text("Текущий день")

}.tag(Tabs.Сегодня)

// Вкладка с настройками

ScheduleView(selectedGroup: selectedGroup, selectedSubGroup: selectedSubGroup)

.tabItem {

Image(systemName: "house.fill")

Text("Вся неделя")

}.tag(Tabs.Расписание)

//Вкладка с сайта

NewsWebView(urlString: "https://news.bstu.by")

.tabItem{

Image(systemName: "display.and.arrow.down")

Text("Новости")

}

}

.navigationBarHidden(true)

}

Вывод списка с расписанием на текущий день:

List(lessons, id: \.self) { lesson in

VStack(alignment: .leading) {

Text("\(lesson.subjectName)")

Text("\(lesson.time), \(lesson.roomNumber), \(lesson.sybjectType), \(lesson.teacher)")

.font(.caption)

.foregroundColor(.secondary)

}

}

Вывод списка всего расписания:

ForEach(Weekday.allCases, id: \.self) { weekday in

Section(header: Text(weekday.rawValue)) {

ForEach(getLessonsForWeekday(weekday: weekday), id: \.self) { lesson in

VStack(alignment: .leading) {

Text("\(lesson.subjectName)")

Text("\(lesson.time), \(lesson.roomNumber), \(lesson.sybjectType), \(lesson.teacher)")

.font(.caption)

.foregroundColor(.secondary)

}

}

}

}

Загрузка новостей с сайта:

func makeUIView(context: Context) -> WKWebView {

let webView = WKWebView()

if let url = URL(string: urlString) {

let request = URLRequest(url: url)

webView.load(request)

}

return webView

}

**4.5 Тестирование**

Тестирование программы проводилось на удобство пользования и функционирование программного продукта. При тестировании на удобство пользования пользователи, работая с приложением, выполняли различные операции. Примером вариантов заданий для выполнения были: переключение между окнами, вывод расписания для различных подгрупп, просмотр всех вкладок на правильность.

При проведении тестов пользователь успешно справился с предложенными заданиями, что позволяет сказать о том, что разработанное приложение обладает интуитивно понятным интерфейсом.

При функциональном тестировании приложения проводилась проверка каждой вкладки. Как выяснилось, каждая вкладка приложения вызывает соответствующее окно, отображающее информацию. Тестирование на скорость позволяет определить время выполнения задачи. Выборка необходимой информации проводилась достаточно быстро, что является довольно приемлемой скоростью для такого приложения.

**Тестирование программы:**

1. Запуск приложения и проверка работоспособности всех пикеров. (см. рисунок 4.1).
2. Проверка на правильность работы всех вкладок приложения:

Работоспособность первой вкладки с расписанием на сегодняшний день. (см. рисунок 4.2).

Работоспособность второй вкладки с расписанием на всю неделю. (см. рисунок 4.3).

Работоспособность вкладки с новостями. (см. рисунок 4.4).

Проверка получения уведомлений о начале пары. (см. рисунок 4.5).



Рисунок 4.1 – Меню приложения.

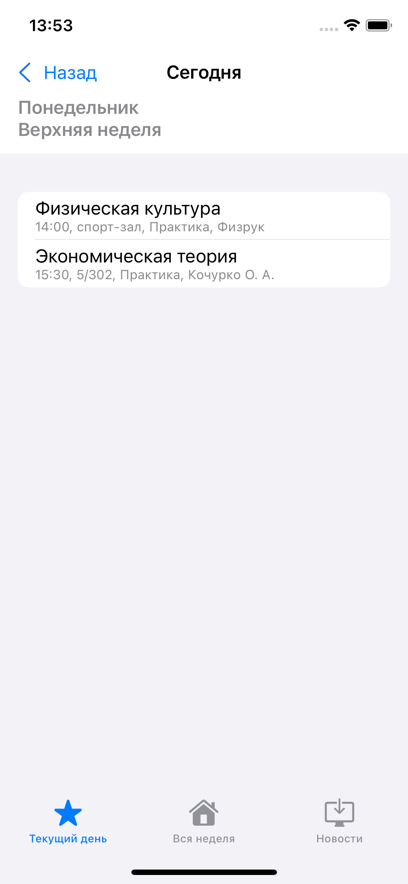


Рисунок 4.2 – Работоспособность первой вкладки.

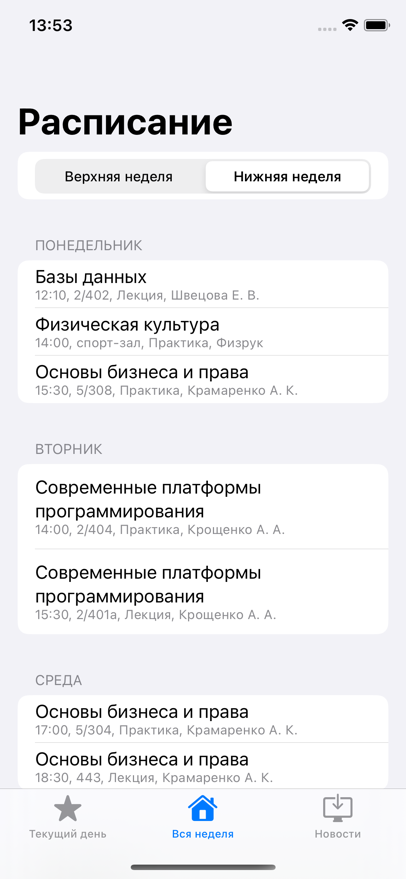
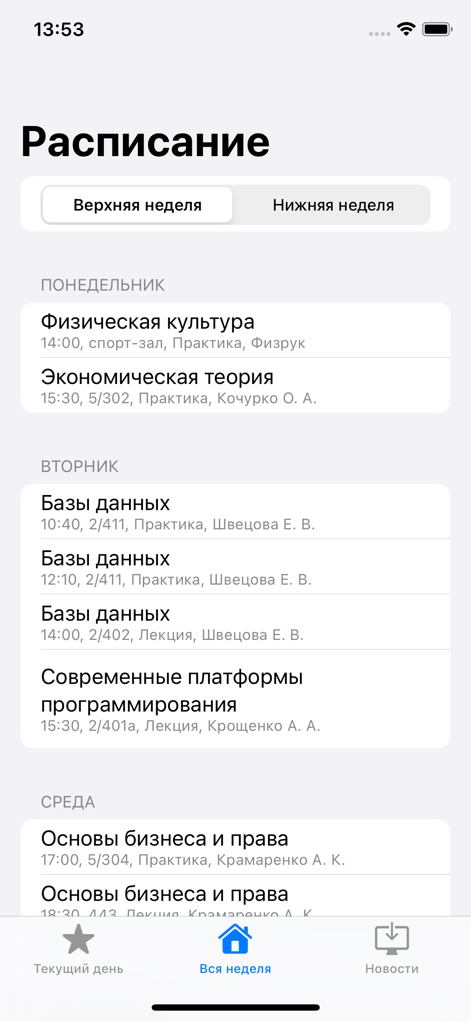


Рисунок 4.3 – Работоспособность второй вкладки.

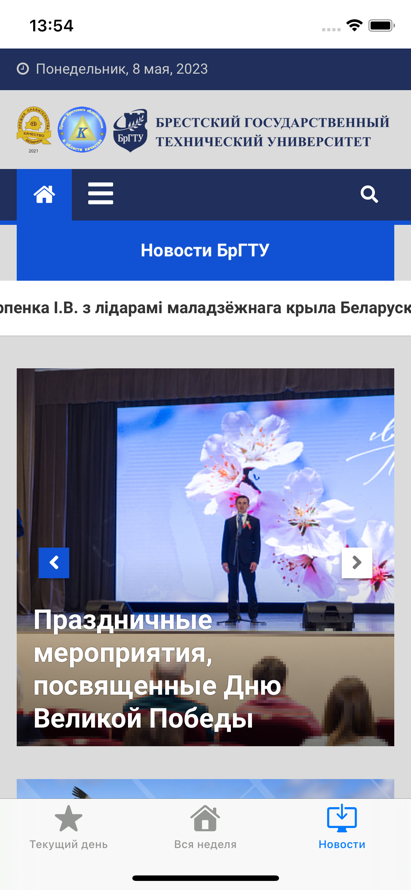


Рисунок 4.4 –Работоспособность третьей вкладки.

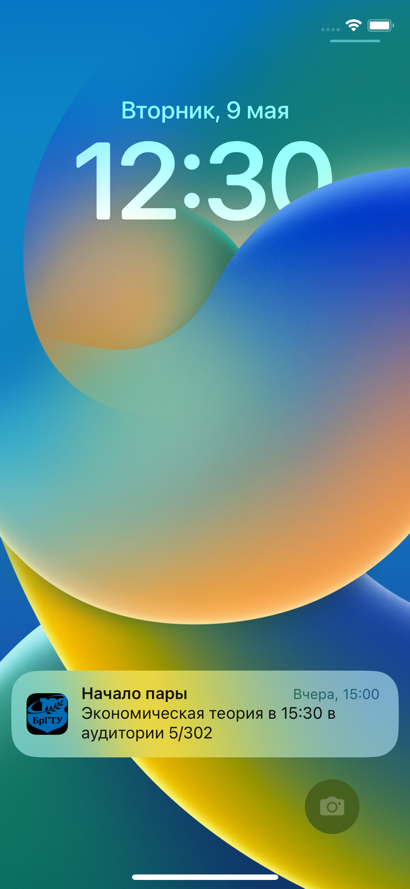


Рисунок 4.5 – Получение уведомления.

# Заключение

В ходе выполнения данной работы на основе различных источников были повышены знания в области ООП. И навыки программирования на языке Swift.

В результате выполнения проекта была полностью реализовано приложение «Расписание», используя объектно-ориентированное программирование.

В процессе тестирования приложения ошибок не обнаружено.

В ходе выполнения курсовой работы были более подробно изучены и изложены в пояснительной записке следующие вопросы:

- Основная информация по языку Swift.

- Ключевые алгоритмы и фрагменты приложения, отвечающие за корректную работу приложения.

- базовые элементы управления интерфейсом, их свойства и методы.

- Разработка приложения "Расписания" на языке Swift.

Результатом работы является приложение "Расписание", работающие на ios-устройство с ОС 15.0 и выше.

# Список используемых источников

1 Василий Усов. Swift. Основы разработки приложений под iOS, iPadOS и macOS, 20211.- 554с.

2 Айк Харазян. Язык Swift. Самоучитель - 2016. — 176с.

3 Basics of XCode. [Электронный ресурс] – 2023. – Режим доступа: https://swiftbook.ru/content/35-video-3/ - Дата доступа: 08.05.2023.

4 Обучение созданию приложений на Swift для всех! [Электронный ресурс] – 2023. – Режим доступа: <https://swiftbook.ru> - Дата доступа: 08.05.2023.

5 ГОСТ 7.12-77 Реферат и аннотация

6 ГОСТ 2.108-68 ЕСКД Спецификация

7 ГОСТ 2.113-75 ЕСКД Групповые и базовые конструкторские документы

8 ГОСТ 21.101-93 Основные требования к рабочей документации