Geekbrains

Дипломный проект по теме: «Разработка мобильного приложения на языке Swift»

IT-специалист:

Разработка мобильных приложений

Павлов А.Д.

Ульяновск

2024г.

Содержание.

1. Введение
2. Основная часть
3. Заключение
4. Список Литературы
5. Приложение

**Введение.**

Описание проекта.

Мобильное приложение DailyPlanner("Ежедневник") на языке программирования Swift - это инновационное и удобное приложение, предназначенное для учета и планирования ежедневных задач и дел.

Оно разработано с учетом современных требований пользователей, которые ищут способы эффективного управления своим временем и задачами. Приложение предлагает широкий набор функций, которые помогут пользователям организовать свой день и быть более продуктивными.

Одной из ключевых особенностей приложения является возможность создания персонального профиля. Пользователи могут добавлять информацию о себе, такую как имя, фотография и контактные данные, чтобы делать записи более персонализированными.

Приложение позволяет создавать задачи и дела, устанавливать им приоритеты и сроки выполнения. Кроме того, пользователи могут добавлять важные моменты, заметки и даже прикреплять фотографии к своим записям. Отмечая выполненные задачи, пользователи могут легко отслеживать свой прогресс и достигать поставленных целей.

Дополнительные функции приложения включают возможность установки напоминаний о ближайших событиях и встречах, чтобы ничего не забыть, а также функцию синхронизации с другими устройствами, что позволяет переносить свои записи на другие платформы и быть в курсе своих дел всегда и везде.

**Обоснование темы.**

В современном мире все больше людей сталкивается с проблемой управления своим временем и задачами. В связи с этим, разработка мобильных приложений, которые помогают организовать повседневные дела, становится все более актуальной. В данный момент, iOS платформа занимает одно из лидирующих положений в мире мобильных устройств, и программирование на языке Swift становится все более популярным.

Целью данного дипломного проекта является разработка мобильного приложения, которое позволит пользователям эффективно планировать свои задачи и управлять своим временем.

В процессе разработки приложения будет осуществлена реализация базовых функций, таких как добавление задач, установка и отслеживание сроков, приоритетов, напоминания о задачах и других функций, которые позволят пользователю легко организовать свою работу и повседневные дела.

Проект будет реализован на языке программирования Swift, который является основным языком разработки для IOS-приложений. Использование Swift позволит создать надежное и эффективное приложение с современным и интуитивно понятным интерфейсом.

Анализ существующих решений в данной области позволит выявить возможности улучшения и уникальных функций, которые могут сделать это приложение особенным.

В итоге, разработанное мобильное приложение позволит пользователям более эффективно управлять своим временем и повседневными задачами, обеспечивая им удобство и уверенность в их выполнении.

Мобильное приложение "Ежедневник" на Swift - это незаменимый помощник для тех, кто стремится организовать свою жизнь, повысить эффективность своего времени и управлять своими задачами с легкостью и удобством.

**Цель проекта.**

Целью дипломного проекта является разработка функционального и удобного в использовании мобильного приложения для организации и планирования повседневных задач и событий. Основная задача проекта заключается в создании удобного интерфейса, позволяющего пользователям создавать, редактировать и просматривать задачи и события на различные даты.

Целью разработки такого приложения является упрощение планирования и организации повседневной жизни пользователей, позволяя им эффективно распределять свое время и не пропускать важные задачи или события. Приложение будет иметь возможность добавлять напоминания, устанавливать приоритеты, добавлять заметки и просматривать расписание на день, неделю или месяц.

В результате работы над дипломным проектом ожидается создание полноценного и легко настраиваемого мобильного приложения, которое будет значительно облегчать организацию и планирование повседневных задач. Пользователям приложения будет предоставлена возможность контролировать свое время и эффективно использовать его для выполнения задач и достижения поставленных целей.

Специализация "Разработчик мобильных устройств" подразумевает работу над созданием программного и аппаратного обеспечения для мобильных устройств, таких как смартфоны, планшеты, носимые устройства и другие портативные устройства.

Основные обязанности разработчика мобильных устройств включают:

1. Проектирование архитектуры и разработка программного обеспечения для мобильных устройств. Это может включать создание приложений, операционных систем, драйверов и других компонентов системы.

2. Разработка системного программного обеспечения, такого как ядро операционной системы или драйверы, для обеспечения безопасности, производительности и стабильной работы мобильных устройств.

3. Оптимизация производительности и энергоэффективности мобильных устройств, чтобы улучшить пользовательский опыт при работе с приложениями и увеличить время автономной работы устройств.

4. Тестирование и отладка разработанного программного обеспечения для обнаружения и устранения ошибок, а также обеспечение соответствия функциональным и качественным требованиям.

5. Исследование и следование последним тенденциям и технологиям в области мобильных устройств для разработки инновационных функций и улучшения производительности.

6. Сотрудничество с командами дизайнеров и инженеров для создания удобного и эргономичного дизайна мобильных устройств.

7. Участие в процессе развертывания и поддержки разработанного программного обеспечения на мобильных устройствах, включая взаимодействие с пользователем и решение проблем.

В целом, специализация разработчика мобильных устройств требует обширных знаний в области программирования, архитектуры систем, тестирования и оптимизации, а также способности следовать современным трендам и быстро адаптироваться к изменениям в технологической среде.

На платформе GeekBrains я обучался языкам программирования C#, Python, Java, Swift, Objetive-C.

Для данного проекта я выбрал язык Swift, так как он является профильным для моего направления, плюс как я описал выше у языка много достоинств, мне показался интересным, буду изучать его дальше.

Мое приложение решает проблему организации и планирования повседневных задач и активностей. Оно помогает пользователям структурировать свой день, записывать важные события, задачи, встречи, напоминания и другую информацию. Преимущества такого приложения включают возможность создания и отслеживания списков дел, установку оповещений и приоритетов для задач, хранение записей и информации в удобном формате, а также доступность и удобство использования на мобильных устройствах.

Инструменты которые я использовал это : приложение Xcode Version 14.0 (14A309), в нем использовал Main StoryBoard для расположения объектов на на экранах ViewController плюс код на языке Swift для связки объектов и правильного алгоритма работы. Библиотеку использовал UIKit. Для баз данных использовал CoreData.

Дизайн UX/UI рисовал в Figma.com, ERD диаграммы в erd.dbdesigner.net, UML и Use Case диаграммы online.visual-paradigm.com.

В проекте выполнял роль разработчика, в какой то степени и

роль архитектора проекта.

**Реализация приложения DailyPlanner(Ежедневник).**

Изначально идея состояла в том чтобы реализовать удобное приложение для решения проблемы организации и планирования повседневных задач и активностей.

В нем должно быть несколько экранов с возможность ведения записей, заметок, с возможностью сохранения их в памяти устройства. Так же должна быть какая то отчетность для пользователя приложения.

В итоге вышло приложение для удобства ведения домашнего хозяйства, а именно приложение для ведения покупок. В нем есть возможность заполнить на одном экране рецепт, допустим пирога с яблоками, на втором экране составить список продуктов для магазина, а третий экран калькулятор расходов.

Тем самым мы получаем приложение 3 в одном, в которые пользователь заходит и решает сразу три задачи – тем самым экономит время.

**Анализ пользовательского спроса на приложение.**

# Согласно статье в журнале <https://www.verywellmind.com> «Best Time Management Apps» :

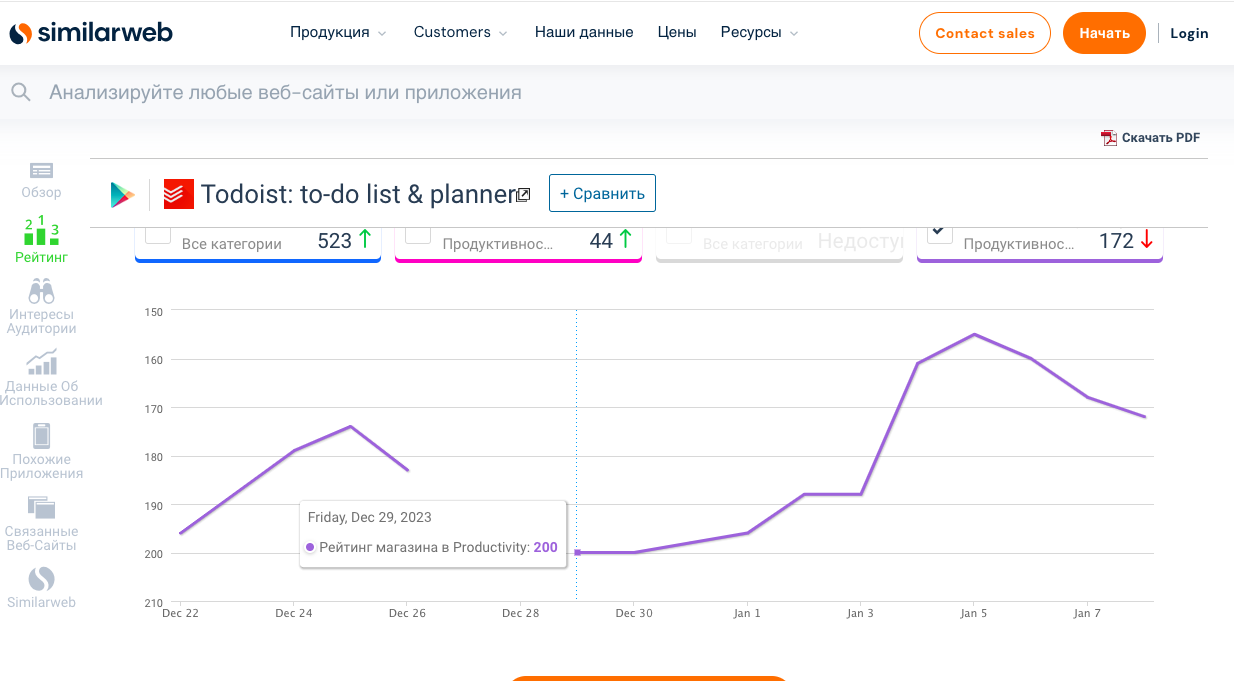
### Полезны ли приложения для тайм-менеджмента?

*Приложения для управления временем могут быть полезны, если вам нужна помощь со списками дел, тайм-трекерами, напоминаниями, календарями и / или их сочетанием. Дополнительные функции часто включают совместное использование календаря или задач, отслеживание проектов и аналитику, которые помогут вам лучше видеть, как тратится ваше время.*

### Эффективны ли приложения для тайм-менеджмента?

*Приложения для управления временем могут быть эффективными, однако это зависит от конкретного человека и его предпочтений. Исследования показывают, что в целом тайм-менеджмент связан с более высоким уровнем удовлетворенности жизнью и общим благополучием, что может положительно повлиять на работу, дом и учебу. Это также связано с снижением уровня стресса.*

Статистика рейтинга самого популярного на данный момент приложения- трекера *Todoist: to-do list & planner* по Google play показывает уверенный спрос у пользователей приложений тайм менеджмента.



Согласно статье ТАСС [https://tass.ru/obschestvo/9508331:](https://tass.ru/obschestvo/9508331Ж)

* В мае-июле интернетом в России хотя бы раз в месяц пользовались 95 млн человек, или 78% населения страны старше 12 лет. Смартфон остается главным устройством для выхода в интернет - им пользуются 67% россиян. С помощью компьютеров и ноутбуков в интернет выходит 51% населения, планшетов - 17%, Смарт ТВ - 15%.

По данным аналитиков, россияне пользуются гаджетами в среднем по пять часов в день по работе и три часа по личным делам. Гаджеты нужны для работы 68% трудоустроенных россиян, для личных нужд – 77%.

Согласно приведенным статьям и размышлениям - основная цель приложения-трекера для пользователя в 2024 году - помочь ему эффективно организовать свою повседневную жизнь и достигать поставленных целей. Приложение должно предоставлять удобный и интуитивно понятный интерфейс, чтобы пользователь мог легко планировать свои дела, встречи, задачи, а также вести записи о важных событиях или мыслях.

Дополнительные функции приложения могут включать оповещения о предстоящих событиях и задачах, возможность создания ежедневных, еженедельных и ежемесячных планов, отслеживание прогресса выполнения задач, аналитику использования времени, возможность добавления приоритетов к задачам, интеграцию с календарем и другими приложениями, а также защиту личных данных пользователя.

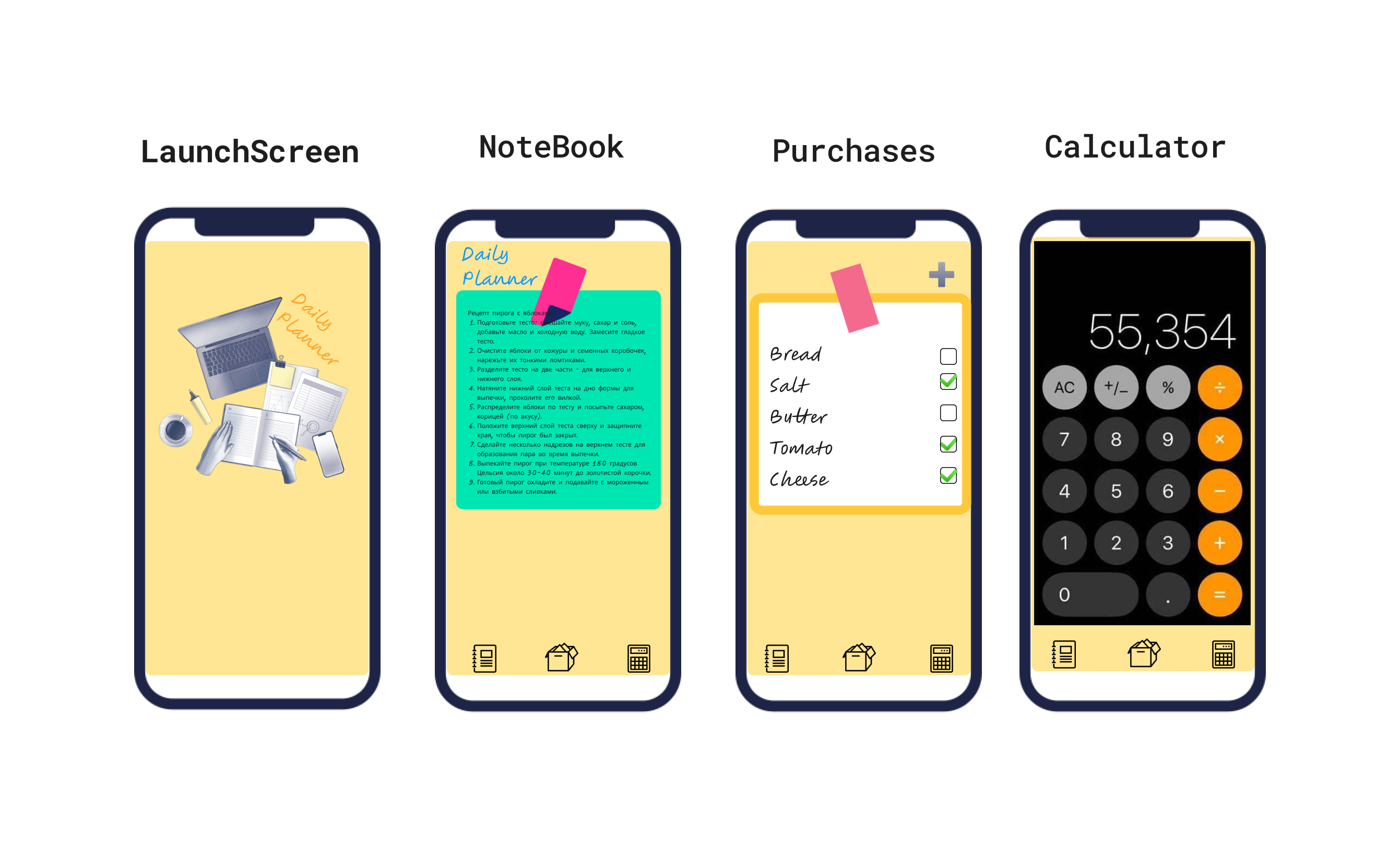
Главная цель приложения для пользователя в 2024 году - сделать его жизнь более организованной, продуктивной и сбалансированной через эффективное планирование и управление его временем.

**Архитектурная часть.**

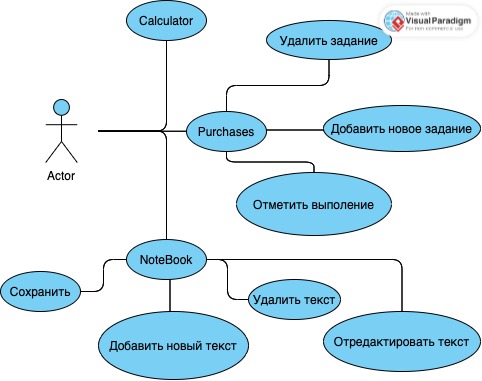
* **Ux/Ui дизайн**

Проектирование приложения я начал с Ux/Ui дизайн модели. Изначально продумал какие будут экраны согласно структуры Xcode.

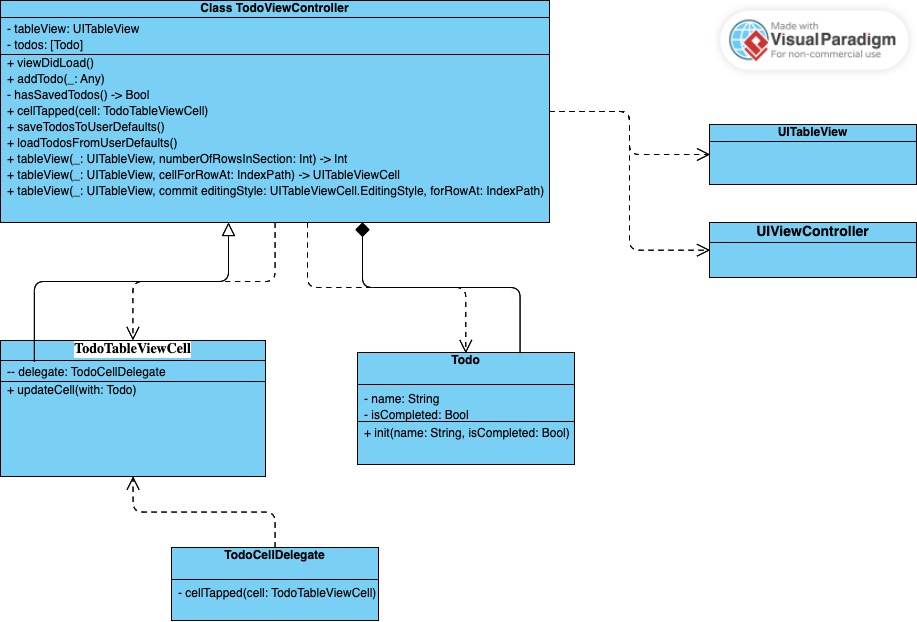
* LaunchScreen – экран загрузки приложения, на нем разместил главное Лого приложения.
* NoteBook – экран одной заметки, здесь может быть рецепт, вообще любой нужный текст, этот экран как один стикер на столе либо холодильнике.
* Purchases – экран списка покупок,может быть и список дел. Отмечаются галочкой о выполениии. Так же сверху есть кнопка добавить «+» для добавления в список нового дела.
* Calculator – калькулятор, если вдруг нужно посчитать сумму покупки, в дальнейшем возможна замена на калькулятор колорий.



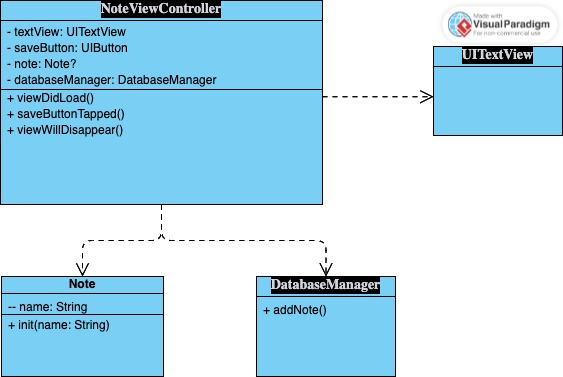
* **Use Case диаграмма.**



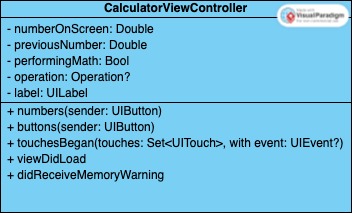
* **UML диаграмма TodoViewController**



* **UML диаграмма NoteViewController**

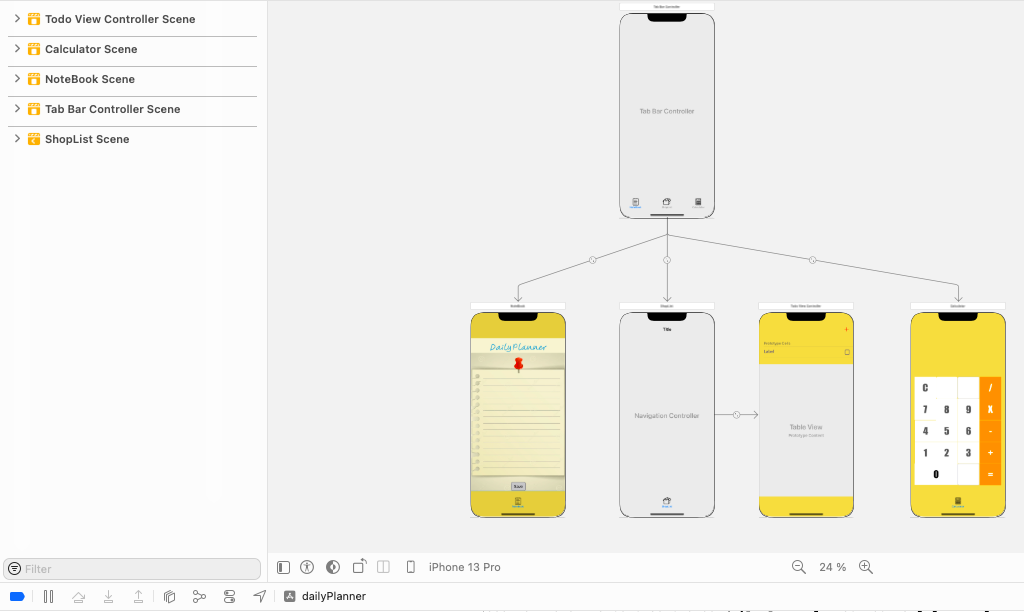
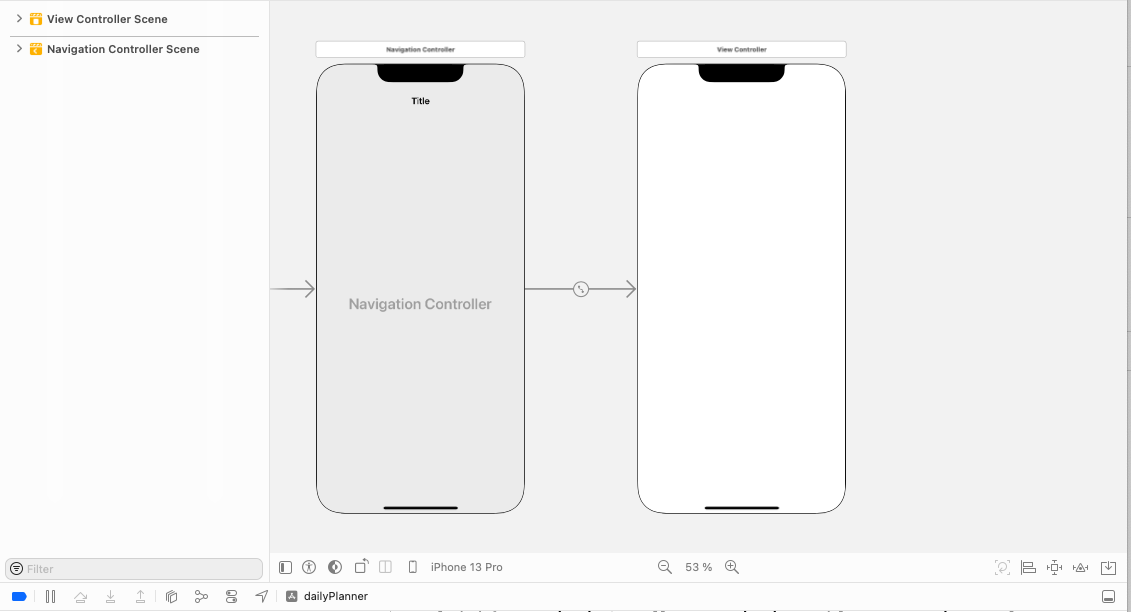


* **UML диаграмма CalculatorViewController**



**Реализация приложения в Xcode.**

Реализация приложения начал с MainStoryBoard, а именно создания NavigationController для перехода на TabBar.stroryboard, управляемый TabBarController.



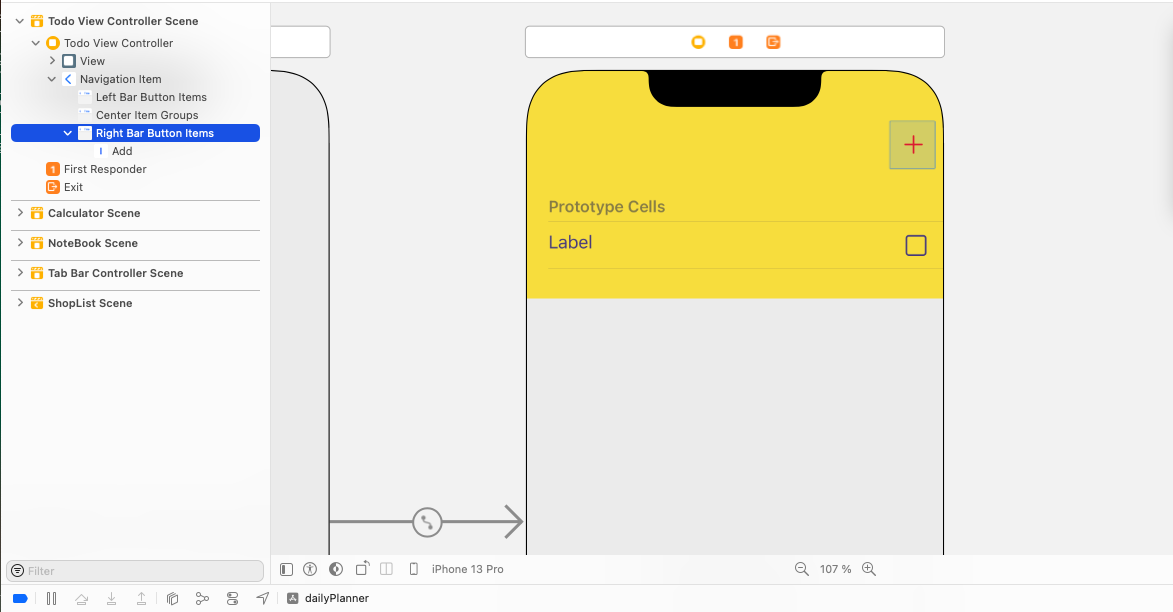
Реализовал экрана на TabBar.stroryboard - все 3 экрана NoteBook Scene, ShopList Scene, Calculator Scene.

**Реализация ShopList.**

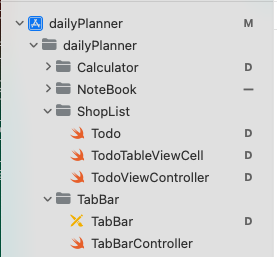
В основе ShopList.  лежит  ToDoList, представляет собой реализацию контроллера представления для приложения, которое отображает список задач и позволяет пользователю добавлять, удалять и отмечать задачи как выполненные.

Изначально добавил на экран tableView для создания таблицы, на него ячейку tableViewCell. На ячейку добавил Label, для текста заметки, и Button для рамки с галочкой – отчетность о выполнений.

Для создания копки добавления заметки Bar Button Items добавил на View NavigationController – и после добавил Bar Button Items на экран.



Создал папку для экрана ShopList, в нем файлы: модель Todo, ячейка TodoTableViewCell, и контроллер TodoViewController.



Сначала реализовал код файла модели Todo с протоколами Codable и Equatable.с свойствами name (название задачи) и isCompleted (флаг завершенности задачи). и конструктор класса принимает два аргумента: name и isCompleted. Он используется для инициализации объекта класса Todo с заданными значениями свойств.

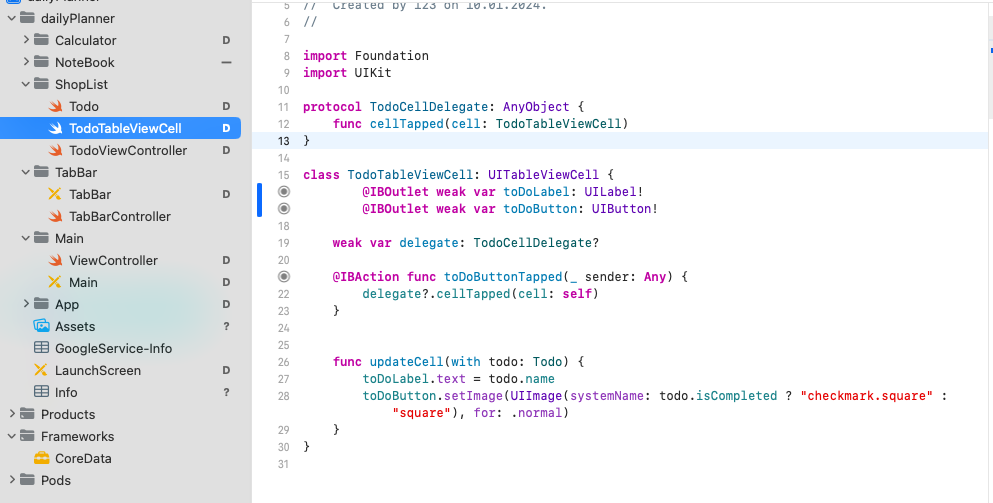
В классе также реализован статический метод ==, который позволяет сравнивать два объекта класса Todo на равенство. Метод сравнивает значения свойств name и isCompleted у двух объектов Todo.

Объявление класса Todo с протоколами Codable и Equatable позволяет использовать его для сериализации и десериализации объектов Todo, а также для сравнения двух объектов на равенство. Это особенно полезно при работе с коллекциями задач (например, массивом) и операциях сохранения/загрузки данных.

В классе ячейки TodoTableViewCell закинул со storyboard outlet’ы

* @IBOutlet weak var toDoLabel: UILabel! – label
* @IBOutlet weak var toDoButton: UIButton! – button

для связи TodoViewController с TabBar.storyboard, так же добавил в Identity Inspector в строку class – TodoViewController, чтобы storyboard понимал каким контроллером он управляется и наоборот. Так же добавил в Identity Inspector для TableViewCell добавил TodoTableViewCell, иначе outlet’ы не связались бы с TodoTableViewCell.



Добавил ссылку IBOutlet на таблицу UITableView, которая отображает задачи.

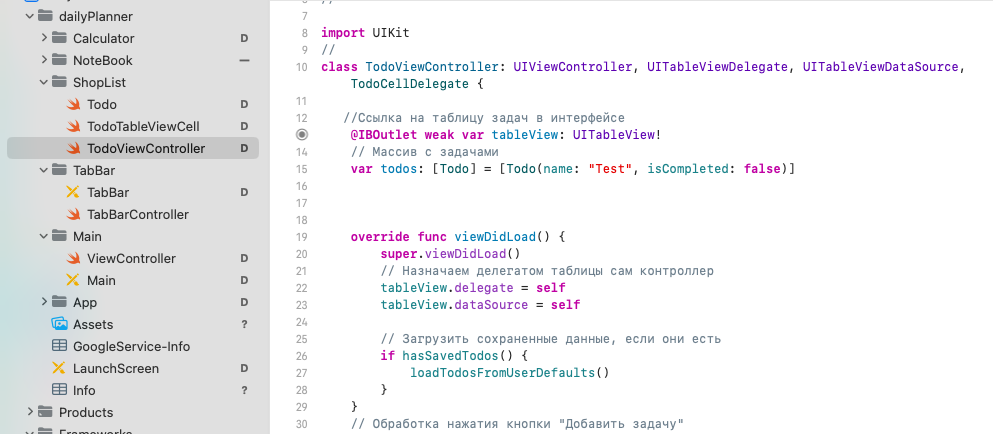
Action  @IBAction func toDoButtonTapped нужен для действия по нажатию на окошко справа от заметки. По нему срабатывает функция cellTapped.

Функция updateCell дает возможность по нажатию на окошко ставить галочку либо убирать ее.

В TodoViewController’е создаем массив todos, и закидывем туда пару значений.Переменная todos представляет массив объектов Todo, где каждый объект Todo представляет одну задачу с именем и флагом выполнения.

Подписываем наш ViewController на UITableViewDelegate, UITableViewDataSource. UITableViewDelegate и UITableViewDataSource используются для обработки событий, связанных с таблицей задач, таких как количество строк, настройка ячеек и обработка редактирования задач.

TodoCellDelegate применяется для обработки событий нажатия на ячейку задачи.



Настраиваем ячейки в нашей таблице tableView через numberOfRowsInSection и cellForRowAt .В numberOfRowsInSection задаем по количеству элементов в нашем массиве. В cellForRowAt получаем ячейку из таблицы и получаем задачу для текущей ячейки и устанавливаем делегата ячейке и обновляем ячейку данными задачи cell.delegate = **self** , cell.updateCell(with: todoCell).

Метод tableView(\_:commit:forRowAt:) вызывается при свайпе ячейки задачи для ее удаления. Он удаляет задачу из массива todos, а затем обновляет таблицу и сохраняет список задач в UserDefaults.

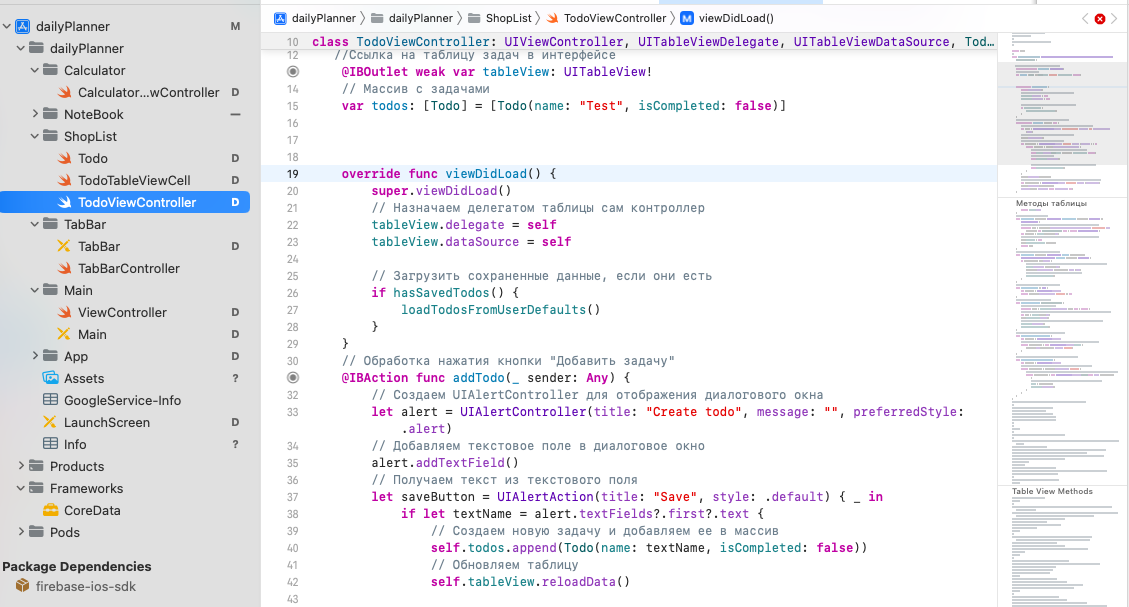
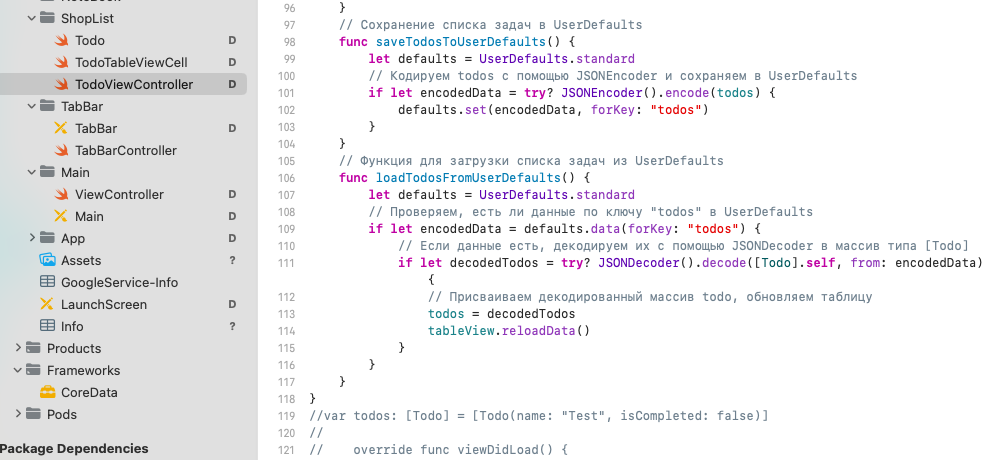
Метод cellTapped(cell:) вызывается при нажатии на ячейку задачи. Он находит индекс нажатой ячейки, изменяет флаг выполнения задачи и обновляет таблицу и сохраняет список задач в UserDefaults.



Action @IBAction func addTodo вызывается при нажатии на кнопку "Добавить задачу". Он отображает UIAlertController с полем для ввода имени новой задачи. При сохранении задачи, если введено имя, она добавляется в массив todos, таблица обновляется, и список задач сохраняется в UserDefaults, чтобы он был доступен в следующий раз.

Хранить решил все заметки в памяти устройства через UserDefaults.

Методы saveTodosToUserDefaults() и loadTodosFromUserDefaults() отвечают за сохранение списка задач в UserDefaults и загрузку списка задач из UserDefaults соответственно. Они используют JSONEncoder и JSONDecoder для кодирования и декодирования объектов Todo в данные, сохраняемые в UserDefaults.

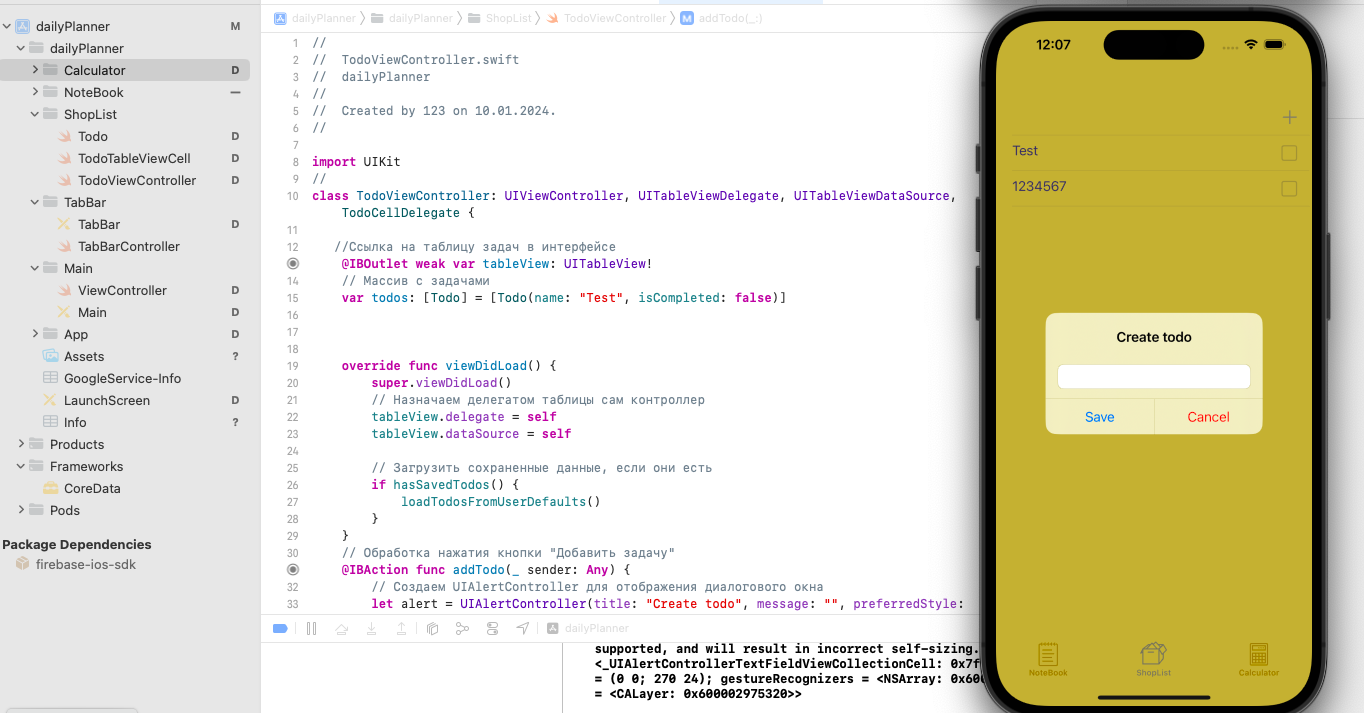
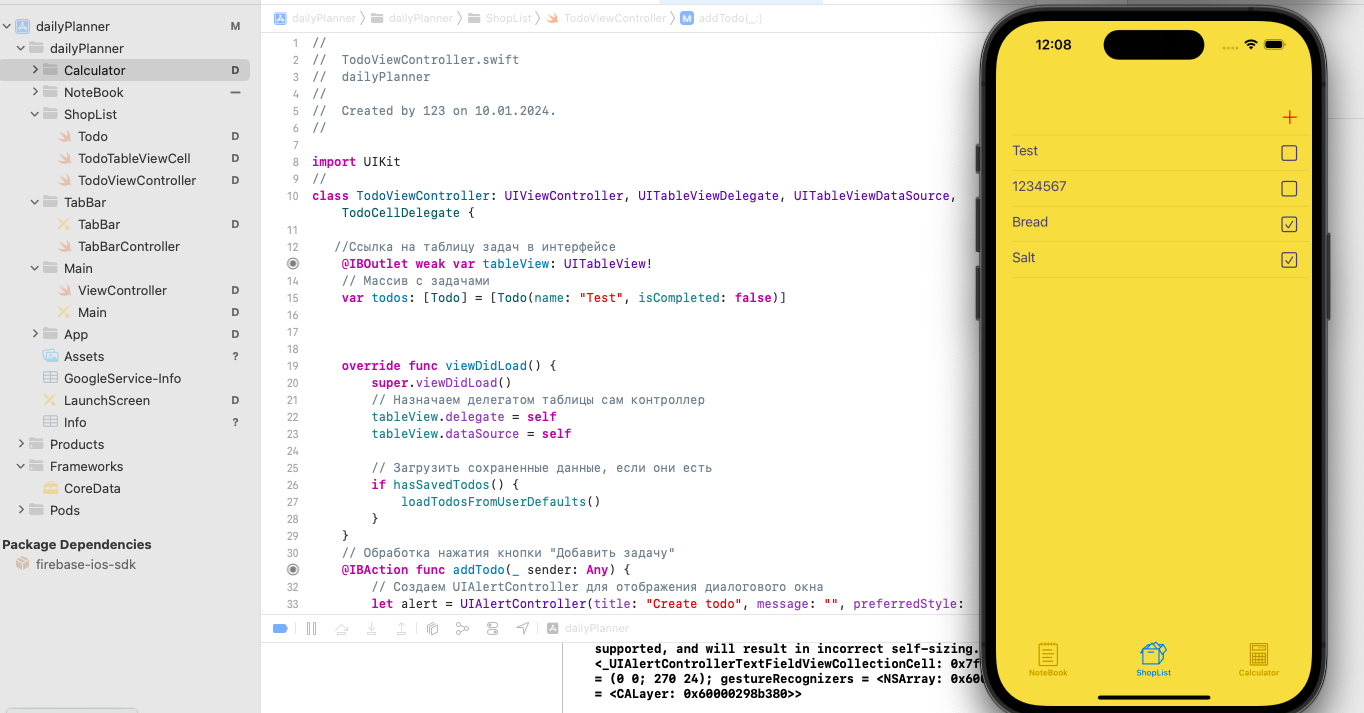


 Добавил во viewDidLoad проверку загрузки сохраненных данных, если они есть

**if** hasSavedTodos() {

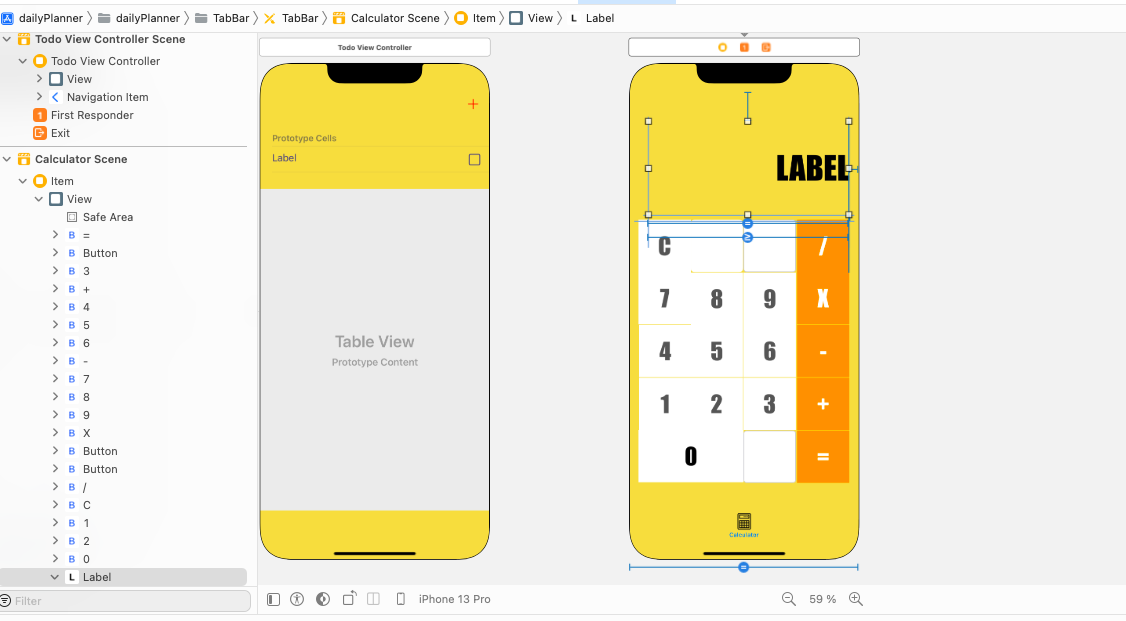
            loadTodosFromUserDefaults()

ScreenShot’ы работы ShopList экрана приложения.



**Реализация Calculator.**

Так как View для Calculator уже создан, я добавил папку Calculator в проект. В ней создал CalculatorViewController и подписал на него View в Tabbar.storyboard.



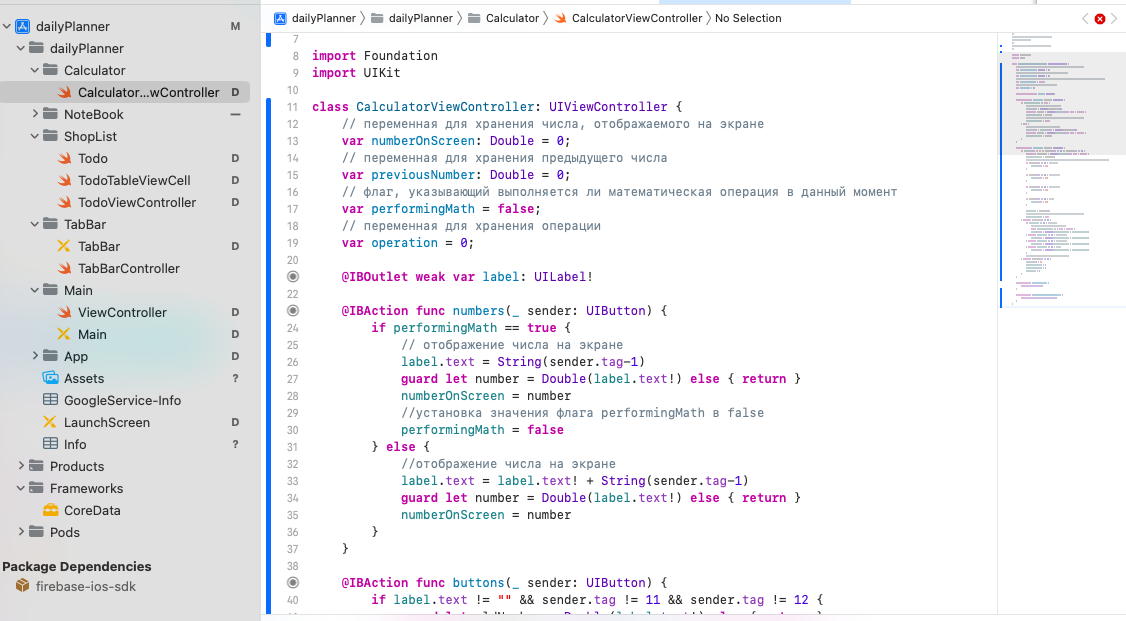
В классе CalculatorViewController объявлены следующие переменные:

- numberOnScreen: переменная для хранения числа, отображаемого на экране;

- previousNumber: переменная для хранения предыдущего числа;

- performingMath: флаг, указывающий, выполняется ли математическая операция в данный момент;

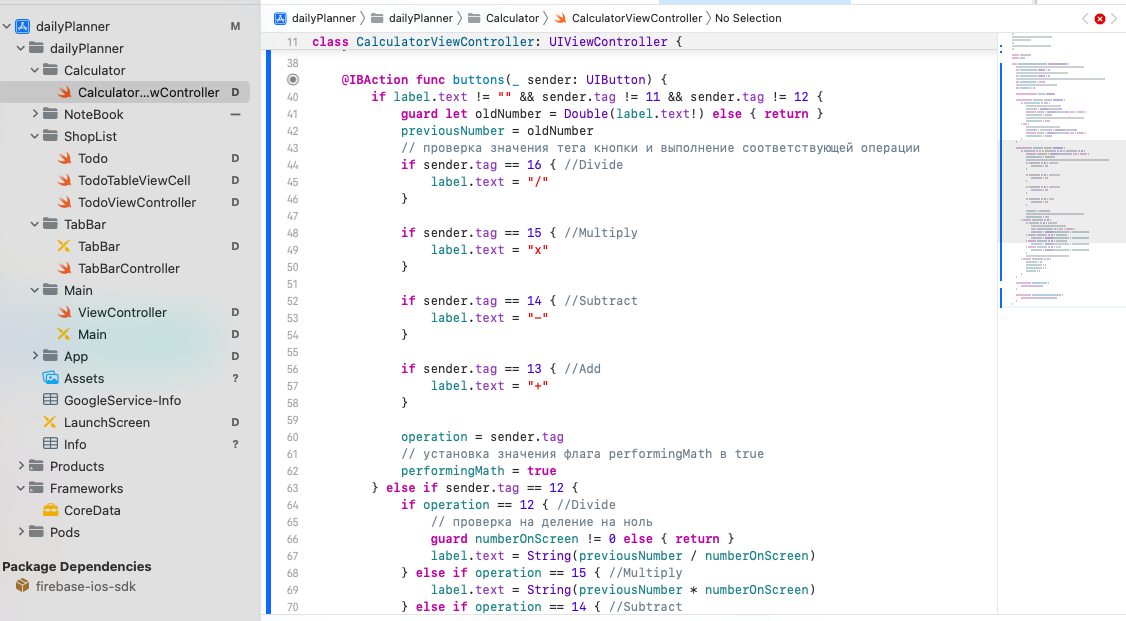
- operation: переменная для хранения операции.



В коде также объявлено IBOutlet для связи с UILabel, в котором отображается результат вычислений.

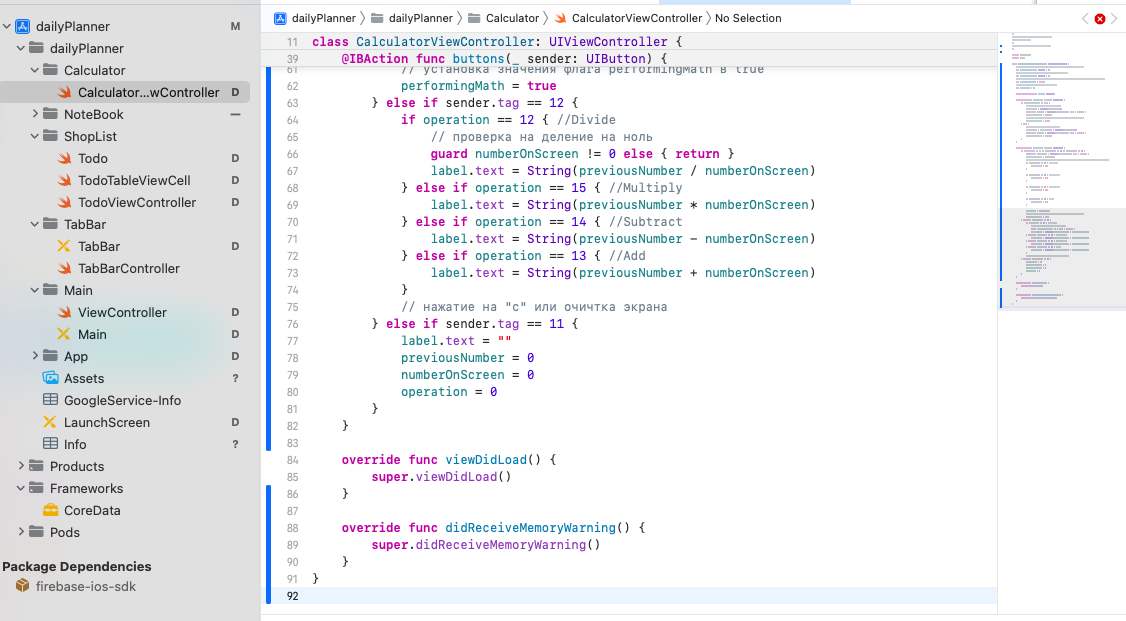
Метод numbers(\_: UIButton) отвечает за обработку нажатий на кнопки с числами. Если performingMath равно true, то выводится введенное число, иначе число добавляется к уже введенному на экране. Значение numberOnScreen обновляется с помощью номера кнопки (тега).

Метод buttons(\_: UIButton) отвечает за обработку нажатий на кнопки с действиями (операциями). Если на экране уже есть число и нажатая кнопка не является кнопкой "с" или очистки экрана, то предыдущее число сохраняется в переменной previousNumber. Затем, в зависимости от тега нажатой кнопки, в UILabel отображается соответствующая операция. Значение переменной operation также обновляется в соответствии с тегом нажатой кнопки. Флаг performingMath устанавливается в true.

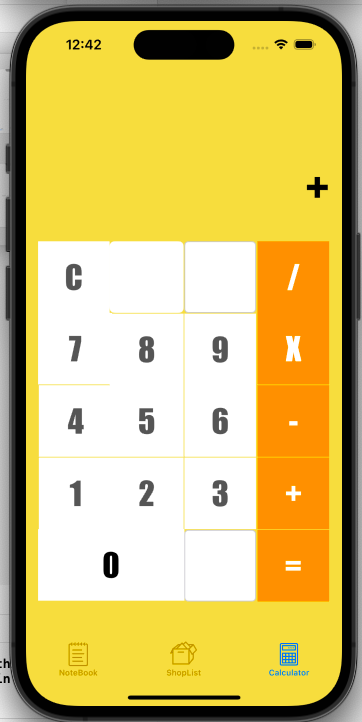


Если нажата кнопка "с" или очистки экрана (тег равен 11), то все переменные и UILabel обнуляются и очищаются.

В методе viewDidLoad() и didReceiveMemoryWarning() нет особых действий.

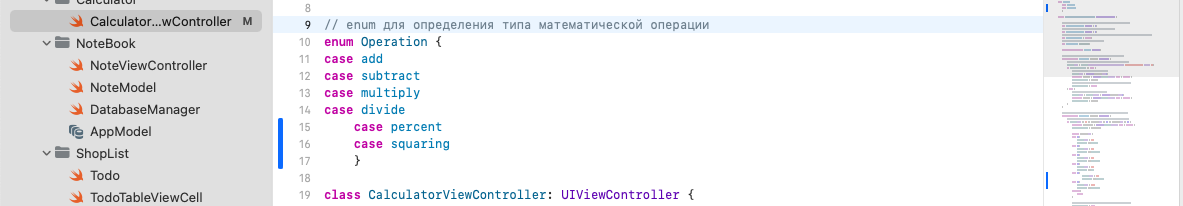


Таким образом, данный код реализует простой калькулятор с возможностью выполнения основных математических операций

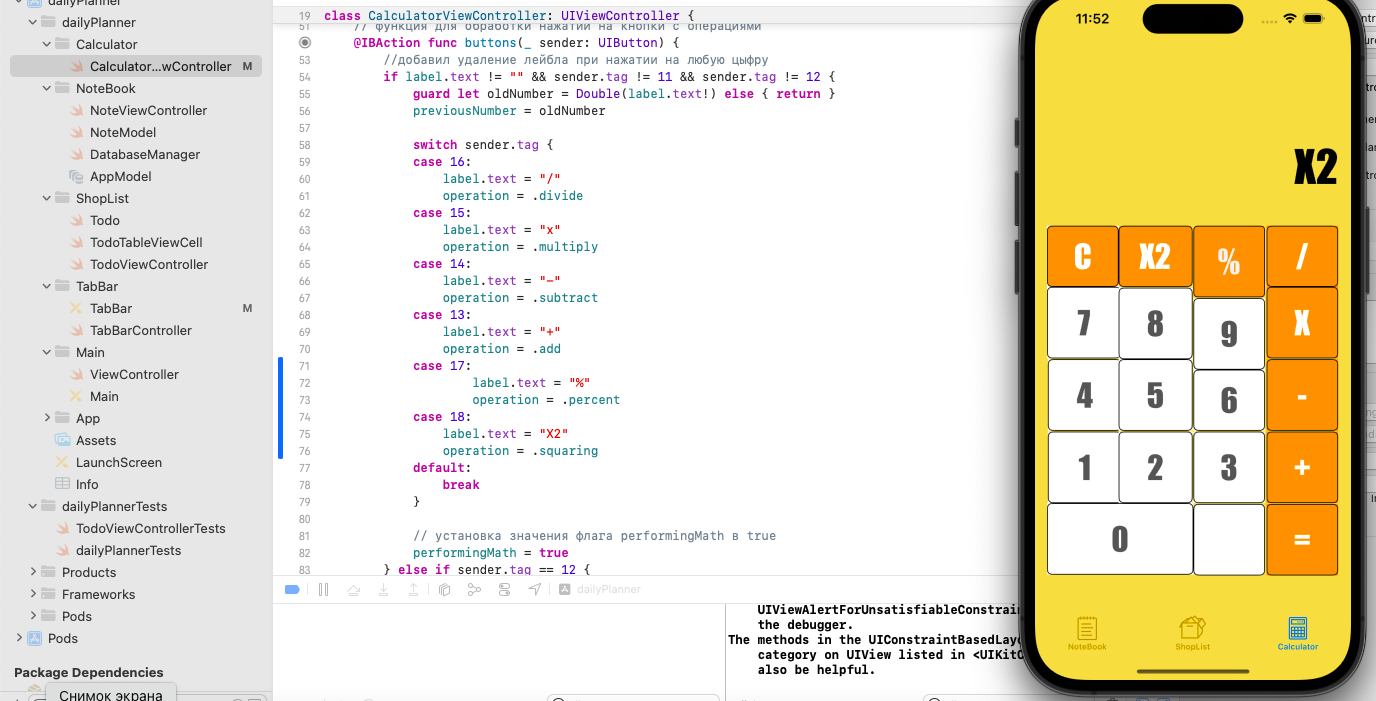


**Update CalculatorViewController**

Добавил перечисления Enum вместо тегов 12,13 и тд.



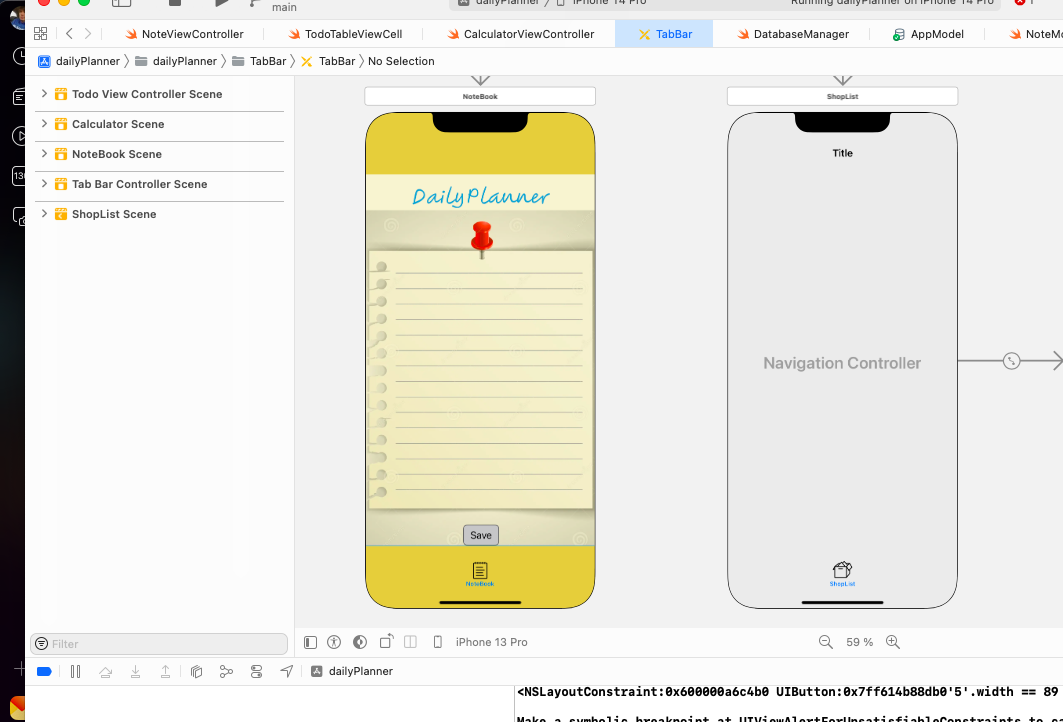
Добавил операцию вычисления процента от числа на экране и возведение в квадрат.



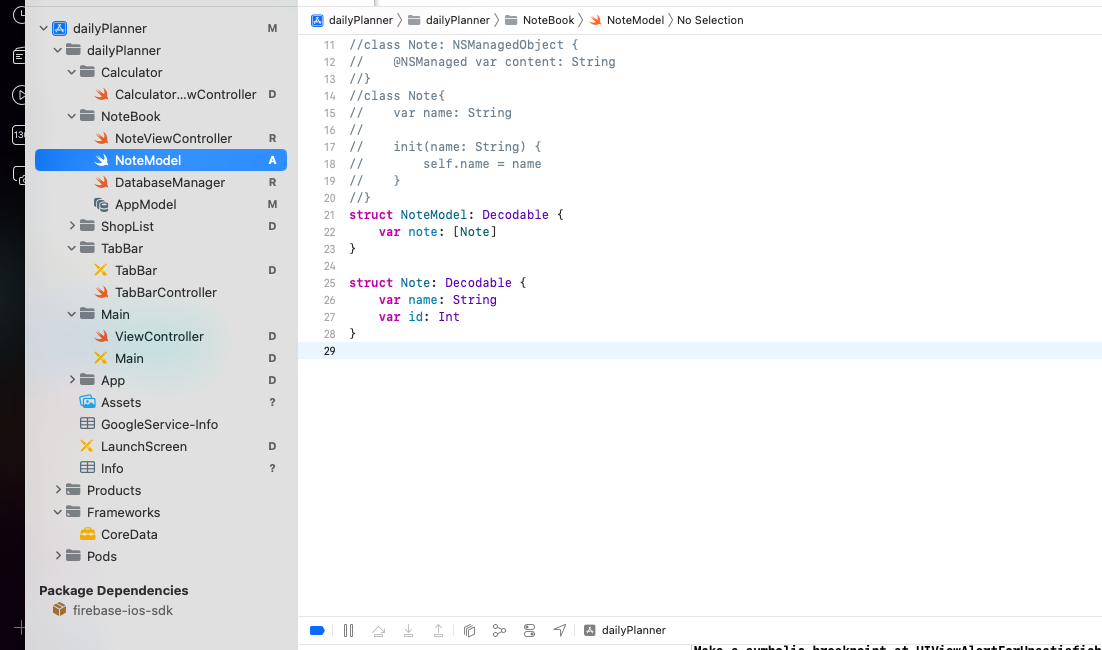
**Реализация NoteBook.**

Идея была в том чтобы реализовать эффект стикера, как клеят на холодильник или монитор компьютера. Пользователь записывает свои мысли либо рецепт, стих, песню.

Добавил на View простой textView почти на весь экран по форме фона- листка из блокнота. В нем можно написать объемный текст. Хранение решил сделать в отличие от ShopList в базе данных. Выбрал CoreData, так как посчитал этот путь более простым.

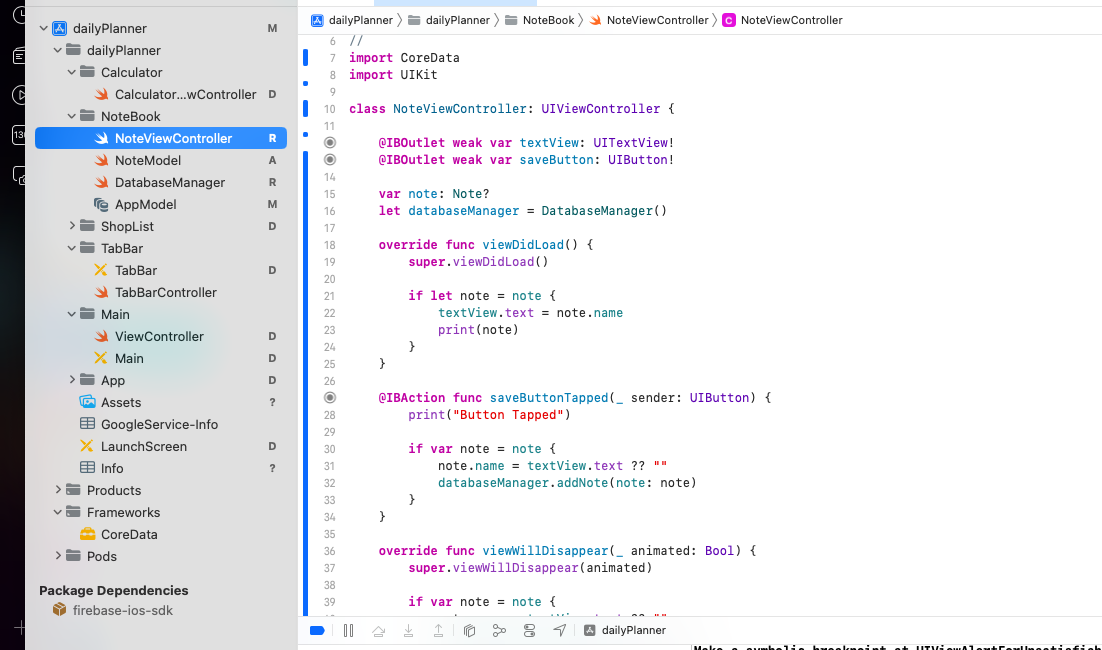
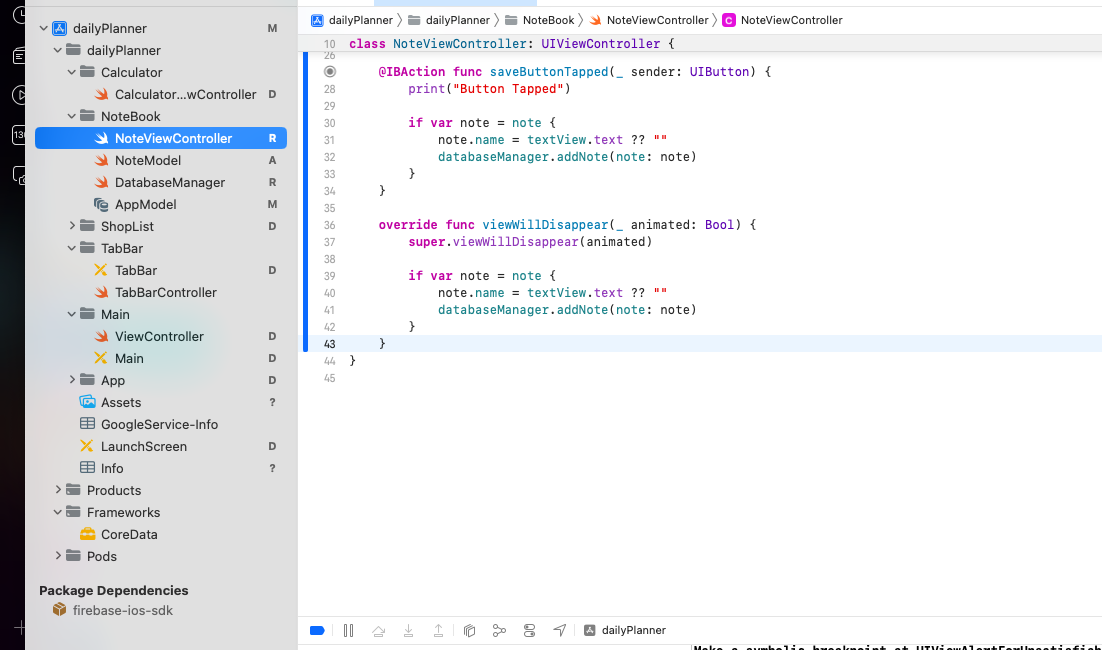


Изначально создал модель NoteModel и заполнил ее свойствами id и name.

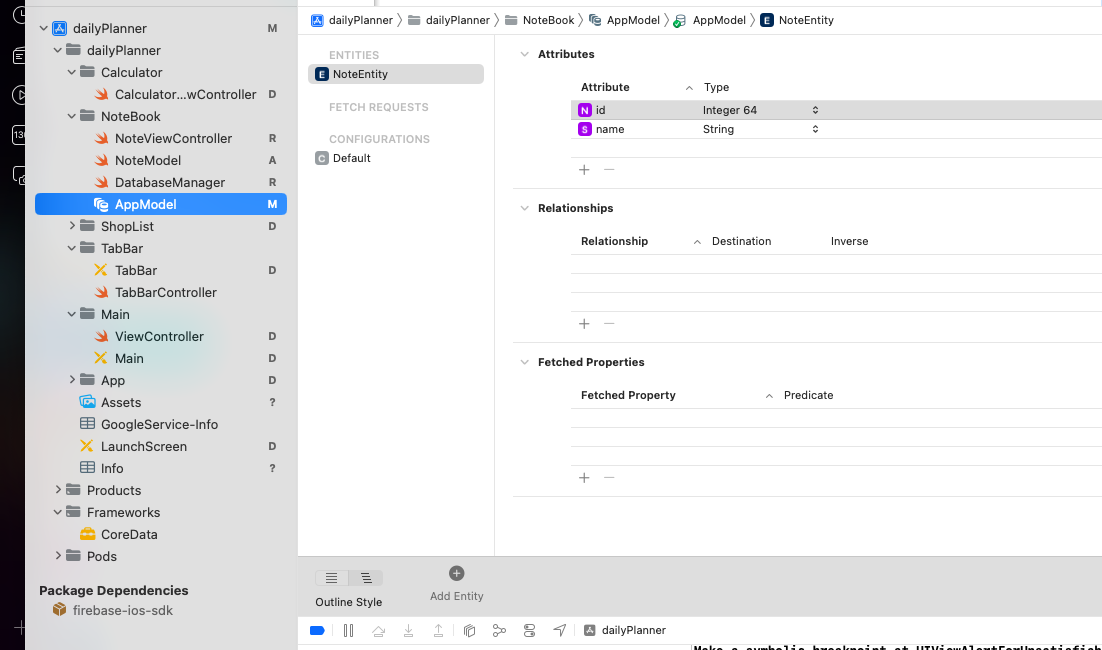


В NoteViewController добавил Outlet textView со storyboard. Контроллер NoteViewController отображает заметку и позволяет редактировать ее. Два Outlet’а - textView (для отображения и редактирования текста заметки) и saveButton (кнопка сохранения изменений).

Создал переменную note типа Note, которая представляет собой модель заметки.

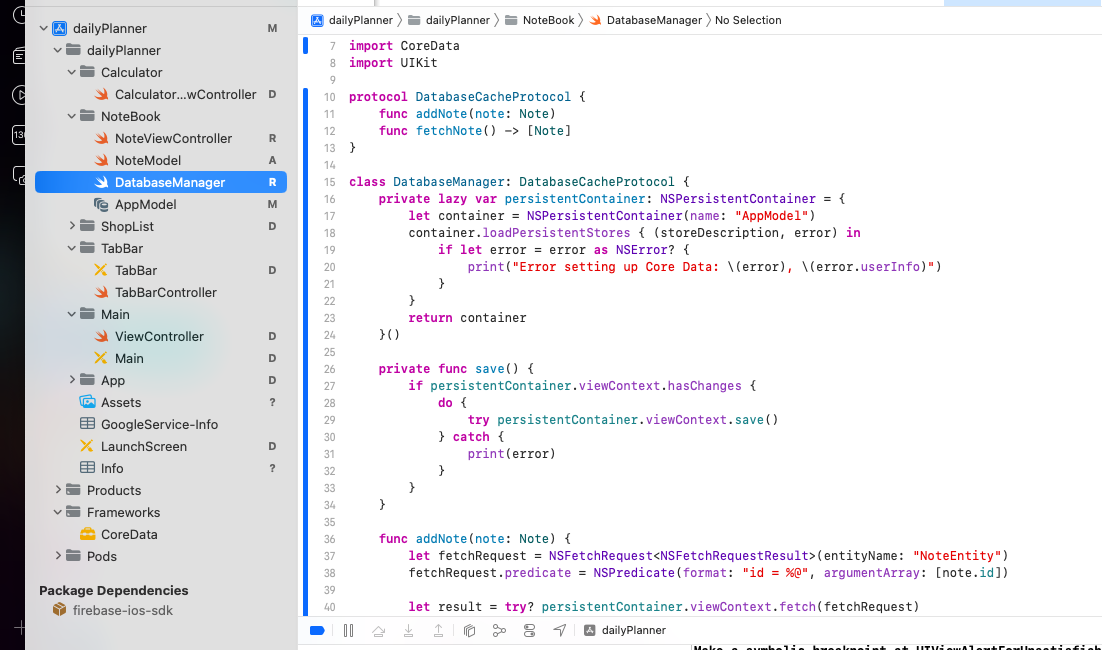


В настройках вашего проекта добавил CoreData framework. Создал файл"AppModel.xcdatamodeld"., который будет определять структуру базы данных.



В созданном файле добавил сущность "Note" с атрибутом "name" типа "String" и "id" типа "Int64".Создал класс для работы с CoreData DatabaseManage, который представляет собой реализацию протокола DatabaseCacheProtocol.

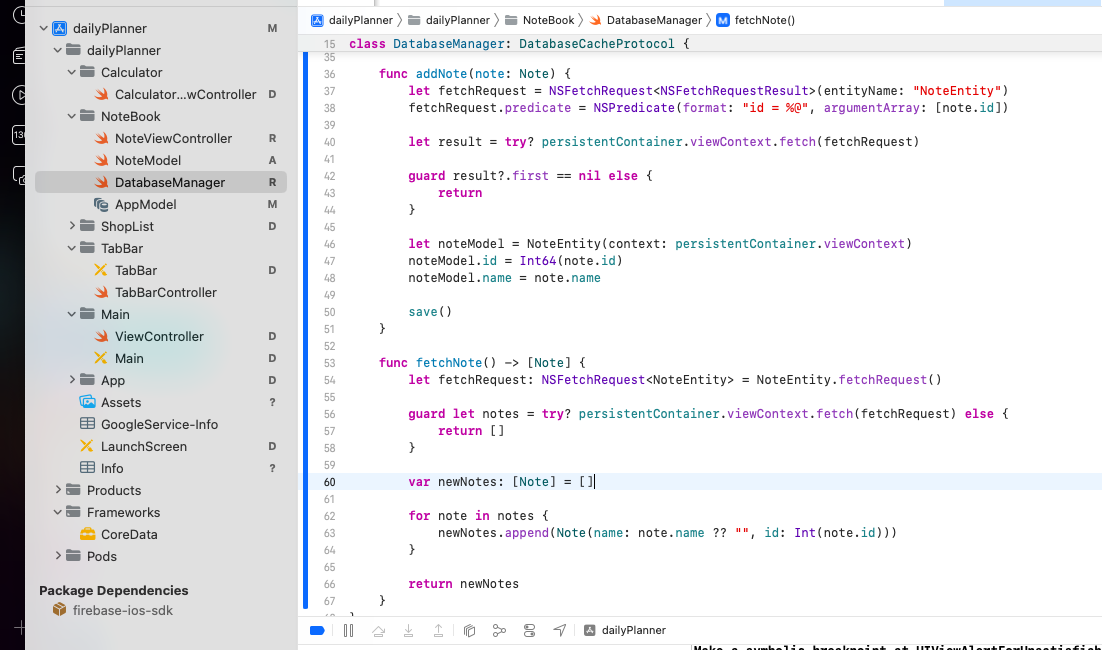
Класс DatabaseManager включает в себя lazy свойство persistentContainer, которое представляет NSPersistentContainer (класс, который предоставляет доступ к базе данных Core Data). В данном свойстве происходит инициализация NSPersistentContainer с указанием названия модели данных "AppModel" и загрузка постоянного хранилища данных.



Функция save() сохраняет все несохраненные изменения в контексте NSPersistentContainer.viewContext.

Функция addNote(note: Note) добавляет новую заметку в базу данных. Сначала выполняется запрос NSFetchRequest для проверки наличия заметки с таким же id в базе данных. Если заметка уже существует, то функция просто возвращает управление. В противном случае, создается новая модель заметки NoteEntity в контексте NSPersistentContainer.viewContext и сохраняются изменения при помощи функции save().

Функция fetchNote() извлекает все заметки из базы данных. Создается запрос NSFetchRequest для извлечения всех объектов типа NoteEntity. Результат запроса сохраняется в массив notes типа [NoteEntity]. Затем массив notes преобразуется в массив новых заметок типа [Note] и возвращается. Каждая новая заметка состоит из имени и id из NoteEntity.

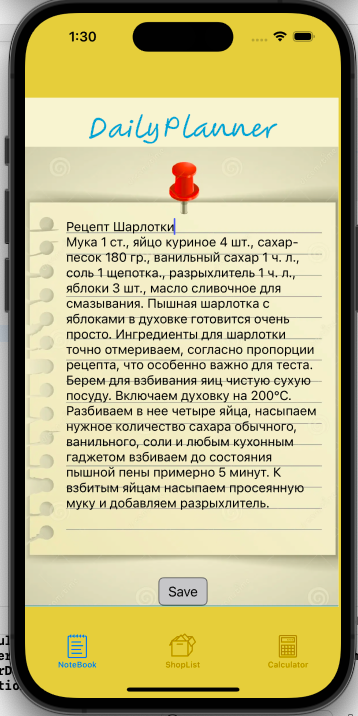


В класс NoteViewController добавил экземпляр класса DatabaseManager для работы с базой данных.

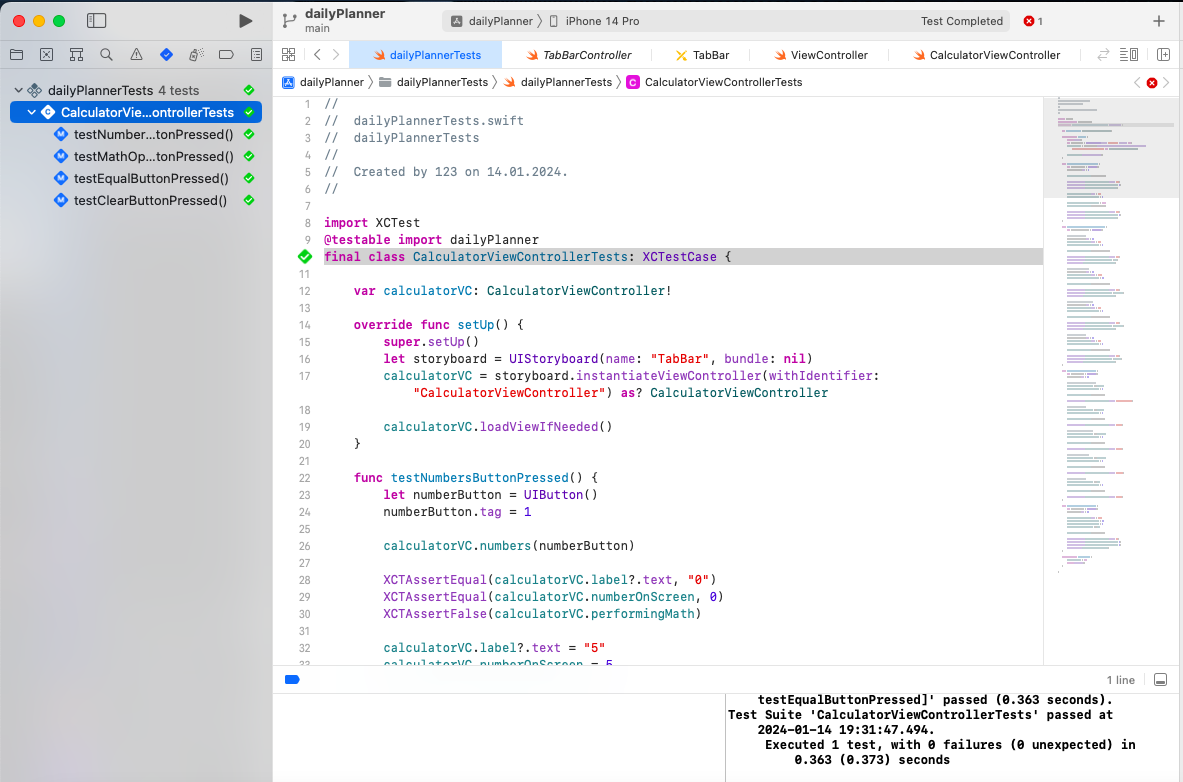
В методе viewDidLoad происходит инициализация текстового поля textView. Если переменная note не равна nil, то текстовое поле заполняется текстом заметки и выводится в консоль.

При нажатии на кнопку saveButton вызывается метод saveButtonTapped. В этом методе, если переменная note не равна nil, обновляется текст заметки и вызывается метод addNote из класса databaseManager для сохранения изменений в базе данных.

Метод viewWillDisappear вызывается при скрытии контроллера. В нем происходит аналогичный процесс сохранения заметки в базу данных.



**Unit тесты.**



Unit тесты добавил для класса CalculatorViewController.

В функции setUp() происходит настройка тестов. Создается экземпляр CalculatorViewController с помощью UIStoryboard и устанавливается его view, чтобы можно было работать со значениями элементов интерфейса.

Функция testNumbersButtonPressed() тестирует поведение при нажатии кнопки с цифрой. Создается экземпляр UIButton и ему присваивается тег 1. Затем метод numbers() вызывается для calculatorVC с передачей numberButton в качестве аргумента. Проверяется, что значение текстового поля label равно "0", значение numberOnScreen равно 0 и performingMath равно false. Затем значения текстового поля и numberOnScreen меняются на 5, performingMath устанавливается в true и снова вызывается numbers(). Проверяется, что значения изменяются обратно на "0", 0 и false соответственно.

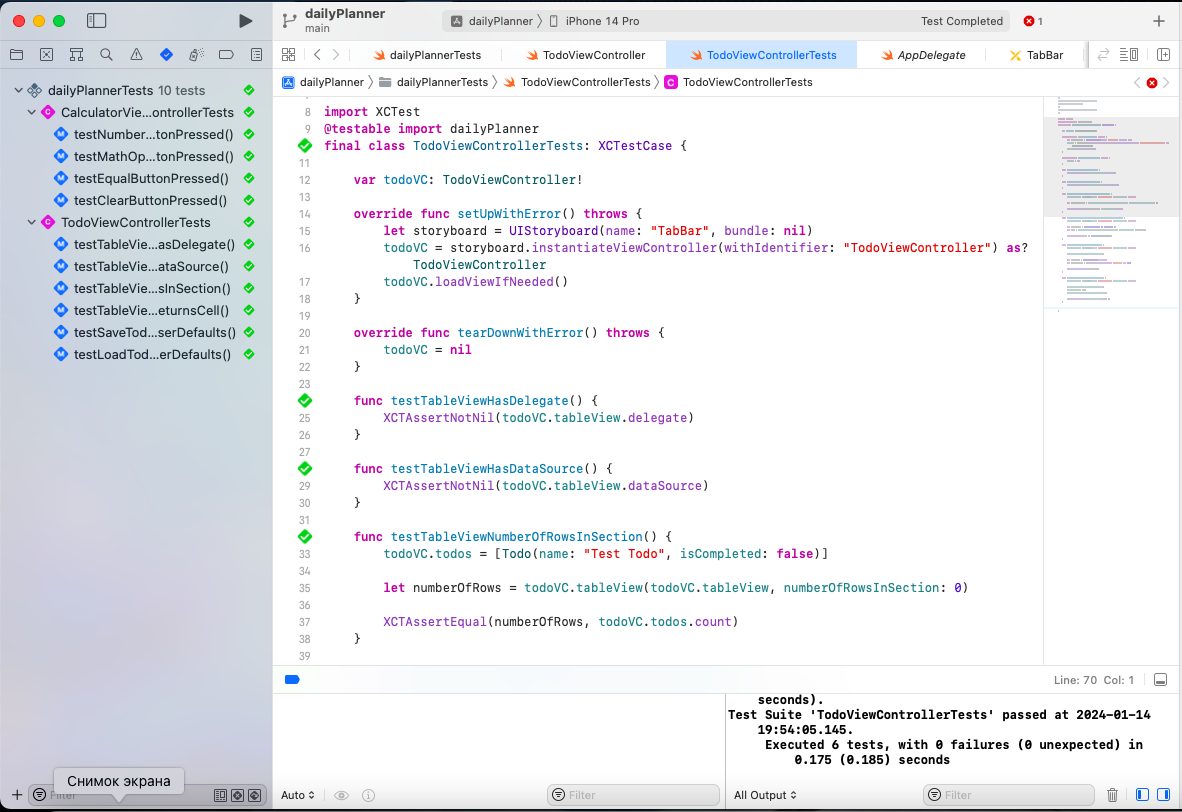
Функция testMathOperationButtonPressed() тестирует поведение при нажатии кнопки с математической операцией. Создается экземпляр UIButton и ему последовательно присваиваются теги 13, 14, 15 и 16, соответствующие операциям сложения, вычитания, умножения и деления. Затем для каждой операции меняются значения текстового поля и previousNumber, вызывается метод buttons() для calculatorVC с передачей operationButton в качестве аргумента и проверяется, что значения изменяются соответствующим образом.

Функция testEqualButtonPressed() тестирует поведение при нажатии кнопки равно. Создается экземпляр UIButton и ему присваивается тег 12. Затем устанавливается операция деления и numberOnScreen равное 0, вызывается метод buttons() для calculatorVC с передачей equalButton в качестве аргумента и проверяется, что значение текстового поля становится "Enter number". Затем устанавливаются другие значения для операций и numberOnScreen, снова вызывается метод buttons() и проверяется, что значение текстового поля изменяется соответствующим образом.

Функция testClearButtonPressed() тестирует поведение при нажатии кнопки очистки. Создается экземпляр UIButton и ему присваивается тег 11. Затем устанавливаются значения для текстового поля, previousNumber, numberOnScreen и operation, вызывается метод buttons() для calculatorVC с передачей clearButton в качестве аргумента и проверяется, что значения сбрасываются в начальное состояние.

Функция tearDown() выполняет очистку после каждого теста. Обнуляется значение calculatorVC и вызывается суперклассовый метод.

Так же добавил Unit тесты для класса TodoViewController.



В методе setUpWithError происходит настройка тестируемого экрана, инициализируется экземпляр TodoViewController, и загружается его view.

Методы testTableViewHasDelegate и testTableViewHasDataSource проверяют, что tableView у экземпляра TodoViewController имеет установленных делегата и источник данных соответственно.

Метод testTableViewNumberOfRowsInSection проверяет, что количество ячеек в таблице соответствует количеству элементов массива todos, который был установлен в экземпляре TodoViewController.

Метод testTableViewCellForRowAtIndexPathReturnsCell проверяет, что метод cellForRowAt возвращает экземпляр класса TodoTableViewCell для заданного indexPath.

Метод testSaveTodosToUserDefaults проверяет сохранение массива todos в UserDefaults и проверяет, что сохраненные данные не являются nil.

Метод testLoadTodosFromUserDefaults проверяет загрузку данных из UserDefaults и проверяет, что количество элементов массива todos после загрузки равно 1.

**Итог.**

Целью моей дипломного проекта была задача спроектировать простое, понятное приложение для сохранения и отслеживания задач пользователя, так же добавил поле с возможностью записывать текст как в заметке, и калькулятор чтобы подсчитать расходы.

Приложение умеет хранить заметки в памяти устройства и хранить большую заметку Notebook в CoreData.

При работе использовал StoryBoard и библиотеки UIKit и CoreData.

Провел тестирование с помощью встроенной библиотеки XCtest.

**Заключение.**

В результате исследования и разработки мобильного приложения DailyPlanner(Ежедневник) на языке Swift были достигнуты следующие цели:

- Разработано приложение с функциональностью создания, изменения и удаления задач и событий.

- Разработан пользовательский интерфейс, который интуитивно понятен и удобен в использовании.

- Данные сохраняются в базе данных и памяти устройства и корректно отображаются на различных экранах приложения.

Все это позволяет пользователям удобно планировать свои задачи и события, а также быть в курсе предстоящих событий благодаря функции напоминаний.

**Список Источников:**

1. <https://www.verywellmind.com/best-time-management-apps-5116817#toc-faqs-2a93ae98-5f04-4431-bfdc-003ec0af9632>
2. <https://www.vedomosti.ru/society/news/2021/06/08/873329-vremya-pered-ekranami-gadzhetov>
3. <https://www.similarweb.com/ru/app/google-play/com.todoist/statistics/#overview>
4. статья ТАСС [https://tass.ru/obschestvo/9508331:](https://tass.ru/obschestvo/9508331Ж)
5. Время на вес золота: Тайм-трекинг для удаленной работы <https://habr.com/ru/companies/roi4cio/articles/500766/>
6. Как программировать под Swift. Архитектура и модульный подход. <https://www.iphones.ru/iNotes/swift-guide-architecture>
7. Core Data: Часть 1: SwiftBook <https://swiftbook.ru/post/tutorials/core-data-chast-1/>
8. Архивы и документация Swift <https://swiftbook.ru/contents/doc/>
9. [Ui/Ux :https://www.figma.com/file/K5ceEExQgCgfu5V82DWhQz/dailyPlannerApp?type=whiteboard&node-id=0-1&t=O8kdXiARXcU2xd0R-0](https://www.figma.com/file/K5ceEExQgCgfu5V82DWhQz/dailyPlannerApp?type=whiteboard&node-id=0-1&t=O8kdXiARXcU2xd0R-0)
10. Use case diagram: <https://online.visual-paradigm.com/w/zvpmkdzn/diagrams/#diagram:proj=0&type=UseCaseDiagram&width=11&height=8.5&unit=inch>
11. Swift открытый объектно-ориентированный язык <https://habr.com/ru/hubs/swift/articles/>
12. Русифицированный и немного измененный гайдлайн от raywenderlich.com: <https://github.com/ILYA2606/SwiftGuideline>
13. Unit testing ViewController and UiView in Swift <https://www.vadimbulavin.com/unit-testing-view-controller-uiviewcontroller-and-uiview-in-swift/>
14. Туториал по Unit и Ui-тестированию в IOS https://swiftbook.ru/post/tutorials/ios-unit-testing-and-ui-testing-tutorial/