МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное   
учреждение высшего образования

**«Южно-Уральский государственный университет**

**(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

**Тема работы**

КУРСОВАЯ РАБОТА

по дисциплине «Программная инженерия»

ЮУрГУ – 09.03.02.2024. 308-077.КР

|  |  |
| --- | --- |
| Нормоконтролер,  к.ф.-м.н., доцент кафедры системного программирования  \_\_\_\_\_\_\_\_\_\_\_\_\_\_Т.Ю. Маковецкая  “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | Научный руководитель:  к.ф.-м.н., доцент кафедры системного программирования  \_\_\_\_\_\_\_\_\_\_Т.Ю. Маковецкая  Автор работы:  студент группы КЭ-304  \_\_\_\_\_\_\_\_\_\_Н.С. Кожухов  Работа защищена  с оценкой: \_\_\_\_\_\_\_\_\_\_\_  “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |

Челябинск 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Южно-Уральский государственный университет**

**(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

УТВЕРЖДАЮ

Зав. кафедрой СП

\_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский

10.02.2024

**ЗАДАНИЕ**

**на выполнение курсовой работы**

по дисциплине «Программная инженерия»

студенту группы КЭ-304 Кожухову Никите Сергеевичу,

обучающемуся по направлению 09.03.04 «Программная инженерия»

1. **Тема работы**Разработка приложения-ежедневника для iOS.
2. **Срок сдачи студентом законченной работы:** 31.05.2024 г.
3. **Исходные данные к работе**
4. iOS & Swift Books. [Электронный ресурс] URL: https://www.kodeco.com/ios/books (дата обращения: 12.02.2023 г.).
5. Apple Developer Documentation. [Электронный ресурс] URL: https://devel- oper.apple.com/documentation (дата обращения: 15.02.2023 г.).
6. **Перечень подлежащих разработке вопросов**
7. Провести анализ предметной области и обзор аналогов.
8. Спроектировать мобильное приложение.
9. Реализовать мобильное приложение.
10. Протестировать мобильное приложение.
11. **Дата выдачи задания:** 9 февраля 2024 г.

Научный руководитель Т.Ю. Маковецкая

К.ф.-м.н., доцент кафедры СП,

Задание принял к исполнению Н.С. Кожухов

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc169110530)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc169110531)

[1.1. Обзор аналогичных проектов 6](#_Toc169110532)

[2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОЙ СИСТЕМЕ 11](#_Toc169110533)

[2.1. Функциональные требования к проектируемой системе 11](#_Toc169110534)

[2.2. Нефункциональные требования к проектируемой системе 11](#_Toc169110535)

[2.3. Диаграмма вариантов использования 12](#_Toc169110536)

[3. ПРОЕКТИРОВАНИЕ СИСТЕМЫ 14](#_Toc169110537)

[3.1. Архитектура системы 14](#_Toc169110538)

[3.2. Описание компонентов 15](#_Toc169110539)

[3.3. Модель базы данных 16](#_Toc169110540)

[3.4. Процесс работы с системой 17](#_Toc169110541)

[3.5. Проектирование интерфейса 18](#_Toc169110542)

[4. РЕАЛИЗАЦИЯ СИСТЕМЫ 21](#_Toc169110543)

[4.1 Анализ существующих решений для реализации проекта 21](#_Toc169110544)

[4.2 Реализация компонента ReminderStore 22](#_Toc169110545)

[4.3 Реализация интерфейса системы 24](#_Toc169110546)

[5. ТЕСТИРОВАНИЕ 28](#_Toc169110547)

[ЗАКЛЮЧЕНИЕ 29](#_Toc169110548)

[ЛИТЕРАТУРА 30](#_Toc169110549)

[ПРИЛОЖЕНИЯ 32](#_Toc169110550)

[Приложение А. Спецификация вариантов использования 32](#_Toc169110551)

# ВВЕДЕНИЕ

**Актуальность**

Мобильные приложения занимают особое место в нашей современной жизни, преобразуя ее и сделав более удобной и приспособленной к быстрому темпу современности. Они не просто инструменты для доступа к информации, развлечений или закупок, они способны значительно упростить рутинные задачи и повысить продуктивность.

Одной из главных ролей мобильных приложений является обеспечение удобства и мобильности пользователей. Благодаря ним, мы можем легко оставаться на связи, управлять задачами, составлять планы и даже управлять своими финансами в любое время и в любом месте. Они становятся нашим надежным проводником в мире информации и сервисов.

Важность мобильных приложений заключается также в их влиянии на современный бизнес. Для компаний мобильное приложение — это не просто инструмент для привлечения клиентов, но и возможность создать более глубокий контакт с ними, улучшить качество обслуживания и повысить лояльность. Благодаря мобильным приложениям, бизнесы могут эффективно просматривать и анализировать данные, оптимизировать процессы и предлагать персонализированные решения. Также нельзя не отметить их важность в образовании и здравоохранении.

Таким образом, мобильные приложения становятся незаменимым инструментом во всех сферах жизни. Их важность и влияние продолжают расти, открывая новые возможности для улучшения качества жизни и работы в целом.

**Цель и задачи работы**

Целью данной работы является приложения-ежедневника для iOS Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ предметной области и существующих аналогичных проектов.
2. Провести анализ требований к приложению.
3. Спроектировать и реализовать мобильное приложение для операционной системы iOS.
4. Протестировать приложение.

**Структура и содержание работы**

Работа состоит из введения, пяти глав, заключения и списка литературы. Объем работы составляет 37 страниц, объем списка литературы – 15 источников.

В первой главе описывается предметная область разработки мобильных приложений. Также были проанализированы мобильные приложения для создания заметок, представленные на платформе AppStore, и выделен основной стек функций в подобных приложениях.

Вторая глава посвящена проектированию мобильного приложения. В ходе работы над данной главой были определены варианты использования, функциональные и нефункциональные требования, спроектированы спецификация основных вариантов использования и диаграмма вариантов использования мобильного приложения.

В третьей главе описывается реализация архитектуры мобильного приложения, описания компонентов, разработка базы данных и процесса работы с системой. Кроме того, был описан алгоритм использования приложения, модели базы данных, а так же архитектура приложения.

В четвертой главе описывается реализация управления данными о напоминаниях пользователя, а так же реализация интерфейса пользователя.

Пятая глава содержит описание функционального тестирования приложения.

В приложении А содержится спецификация вариантов использования мобильного приложения.

В приложении Б содержатся листинг класса отвечающей за обновление данных.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Анализ предметной области является важным этапом в начале проекта, направленным на полное понимание текущего состояния и основных аспектов предметной области. В процессе анализа собирается и изучается информация о текущих процессах, системах и технологиях, используемых в данной области. Выявление основных проблем и вызовов позволяет точно определить те аспекты, которые требуют внимания и улучшения в рамках проекта.

Моё мобильное приложение для управления задачами и напоминаниями создаётся с учётом всех выявленных потребностей и проблем, чтобы предложить пользователям удобный и эффективный инструмент для организации их времени. Оно предоставляет функции для создания, редактирования и отслеживания задач, а также интеграцию с напоминаниями и возможность синхронизации данных между устройствами. Таким образом, приложение поможет пользователям повысить свою продуктивность и лучше управлять своим временем.

1.1. Обзор аналогичных проектов

В ходе проведенного обзора аналогов было выявлено большое количество приложений, которые схожи с разрабатываемым приложением.

К примеру, мобильное приложение «Sticky Notes+Reminders events» в App Store. Данное приложение пользуется популярностью и имеет высокий рейтинг. Данный сервис является одним из самых популярных в вышеупомянутом магазине.

Скриншоты данного приложения представлены на рисунке 1.

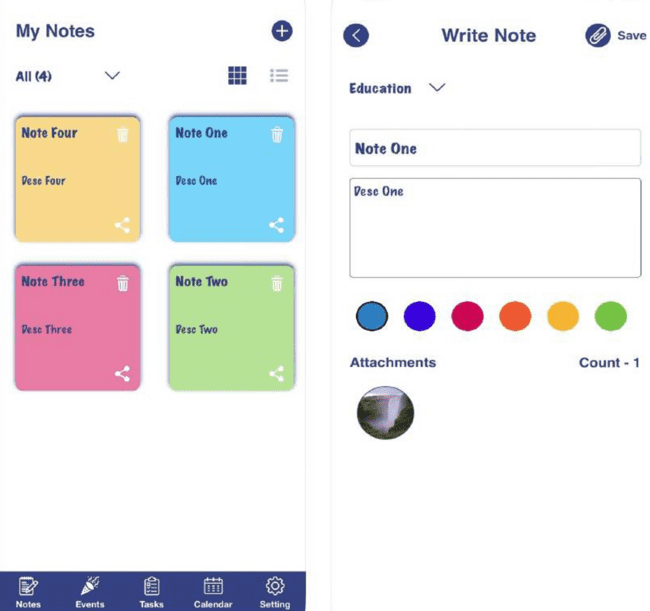


Рисунок 1 — Скриншоты приложения Sticky Notes+Reminders events.

Основные выявленные достоинства и недостатки приложения приведены в таблице 1.

Таблица 1 — Преимущества и недостатки Sticky Notes+Reminders events.

|  |  |
| --- | --- |
| **Преимущества** | **Недостатки** |
| Минималистичный интерфейс. | Работа только с устройством iOS. |
| Вывод всех необходимых данных | Несовременный дизайн приложения |

Следующий пример - приложение для iOS, обладающее немного большим функционалом и улучшенным дизайном.

Скриншоты данного приложения представлены на рисунке 2.

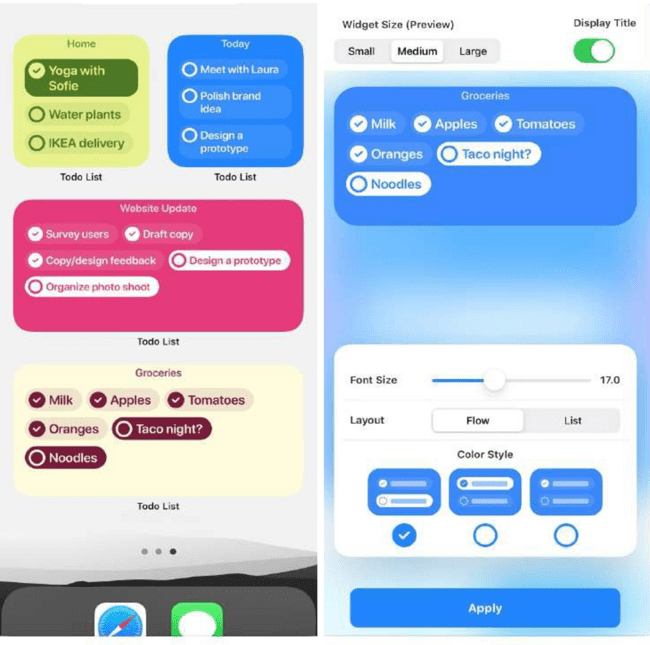


Рисунок 2 — Скриншоты приложения To Do List Widget.

Основные выявленные достоинства и недостатки приложения приведены в таблице 2.

Таблица 2 — Преимущества и недостатки To Do List Widget.

|  |  |
| --- | --- |
| **Преимущества** | **Недостатки** |
| Наличие виджетов приложения | Ограниченная совместимость с устройствами Apple |
| Большое количество изменяемых параметров | Невозможность создать объемную заметку |

Последний пример — iOS приложение Things 3. Данное приложение почти полностью повторяет функционал прошлых, но также имеет несколько уникальных функций.

Скриншоты данного приложения представлены на рисунке 3.

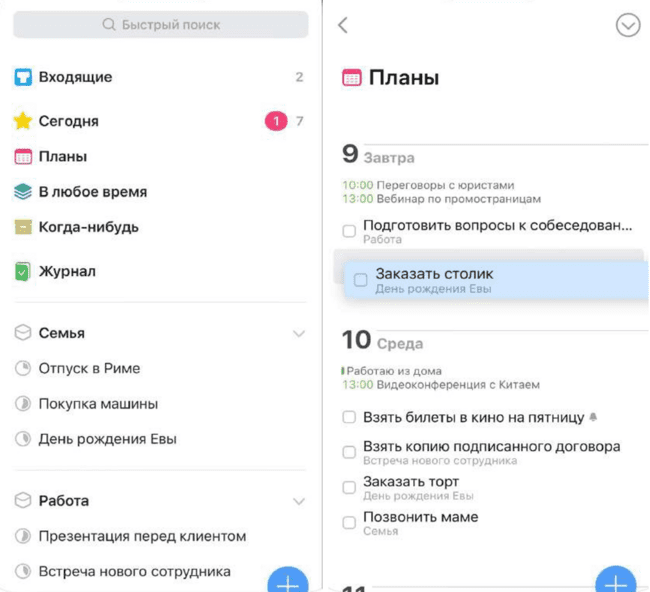


Рисунок 3 — Скриншоты приложения Things 3

Основные выявленные достоинства и недостатки приложения приведены в таблице 3.

Таблица 3 Преимущества и недостатки Things 3

|  |  |
| --- | --- |
| **Преимущества** | **Недостатки** |
| Поддержка большого количества  разнообразного ввода | Визуально перегруженный интерфейс |
| Работа с разными устройствами Apple. | Нет возможности наследования из других приложения |

Обзор перечисленных аналогов показал, что все рассмотренные мобильные приложения имеют как общие, так и уникальные функции. Все приложения предназначены для использования на устройствах компании Apple, без которых они не могут функционировать. Среди преимуществ рассмотренных решений можно отметить разнообразные реализации пользовательского интерфейса. К недостаткам можно отнести невозможность интеграции со всеми устройствами.

**Вывод главе**

Анализ аналогичных приложений для iOS, выявил их общие и уникальные функции. Эти приложения предлагают различные подходы к управлению задачами, но имеют ограничения, связанные с интеграцией и дизайном.

Рассмотрев преимущества и недостатки данных приложений, было решено разработать мобильное приложение для iOS, используя Swift, Xcode и UIKit. Такой выбор технологий обеспечит гибкость и функциональность, позволяя эффективно управлять задачами и отслеживать их выполнение.

# 2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОЙ СИСТЕМЕ

Определение требований к проекту включает в себя выяснение функциональных и нефункциональных характеристик, необходимых для достижения поставленных целей. Анализ технологической составляющей оценивает текущие технологии и системы, используемые в предметной области, и их пригодность для реализации поставленных задач. На основе анализа формулируются рекомендации и стратегии для улучшения процессов и решения проблем, что способствует более эффективному достижению целей проекта и повышению его успешности.

Для дальнейшего проектирования приложения были составлены два типа требований:

– функциональные требования – определяют, что приложение должно делать;

­ нефункциональные требования – определяют, как приложение должно работать.

Оба типа требований играют важную роль в процессе проектирования приложения, поскольку помогают разработчикам понять, что от них ожидается и какие ограничения имеются.

2.1. Функциональные требования к проектируемой системе

Функциональные требования описывают, как продукт должен вести себя в различных ситуациях. Они определяют, какие функции и возможности должны быть реализованы разработчиками, чтобы пользователи могли успешно выполнять свои задачи в рамках бизнес – требований. Это важное соотношение между требованиями пользователя, бизнес – потребностями и функциональностью продукта является ключевым для успешного завершения проекта. Функциональные требования обычно формулируются в виде утверждений, использующих слова «должен» или «должна», и описывают конкретные функции или поведение продукта, необходимые для удовлетворения потребностей пользователей.

В рамках вышеописанной задачи были выявлены следующие функциональные требования:

1. Создание и удаление задач;
2. Просмотр и редактирование содержимого задач;
3. Отслеживания прогресса выполнения задач;
4. Просмотр статистики выполнения задач.

2.2. Нефункциональные требования к проектируемой системе

1. Удобный и интуитивно понятный дизайн интерфейса;
2. Приложение должно иметь возможность просмотра задач с различных устройств;
3. Приложение должно быть доступным для пользователей с ограниченными возможностями зрения и слуха;
4. Интерфейс приложения должен адаптироваться к различным языковым настройкам пользователя.

2.3. Диаграмма вариантов использования

Для проектирования описанных выше функциональных требований с помощью языка объектного моделирования UML была создана диаграмма вариантов использования, показывающая отношения между пользователем и прецедентами (рисунок 4).

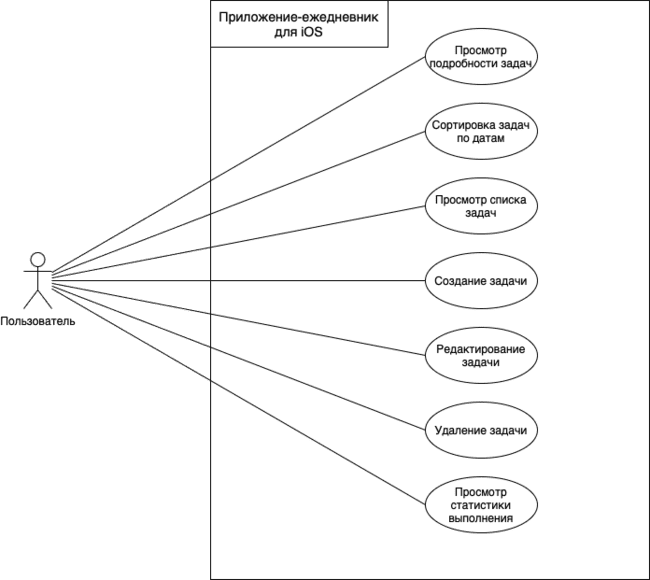


Рисунок 4 – Диаграмма вариантов использования мобильного   
приложения

**Основные актеры, взаимодействующие с системой**

Основным актером в контексте описанного проекта мобильного приложения для управления задачами является **– пользователь приложения**. Он взаимодействует с приложением для создания новых задач, их редактирования, удаления и отслеживания прогресса выполнения.

Спецификация основных вариантов использования приведена в Приложении А.

# 3. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

3.1. Архитектура системы

Для корректной работы функций пользователь должен предоставить разрешение на использование хранилища стандартного приложения iOS «Напоминания». С помощью фреймворка EventKit можно получить доступ к календарным данным устройства. Путем использования API EventKit приложения могут взаимодействовать с календарями и событиями, а также создавать, изменять и удалять события и напоминания.

EventKit предоставляет удобный интерфейс для работы с событиями и напоминаниями, что делает его отличным инструментом для создания приложений, связанных с организацией времени и расписанием. Кроме того, EventKit позволяет синхронизировать задачи с другими устройствами через iCloud, обеспечивая единое и согласованное управление событиями и напоминаниями на разных платформах.

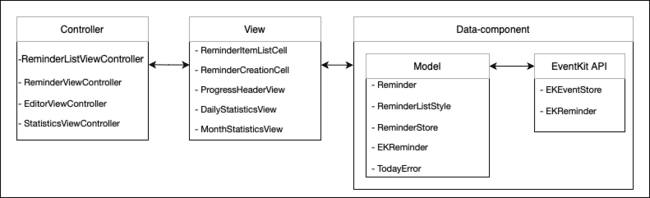


Рисунок 5 – Диаграмма компонентов

Для логического разделения кода приложения и его абстрагирования будет использоваться паттерн проектирования Model-View-Controller (MVC). Основная идея данного паттерна заключается в разделении модели данных приложения, пользовательского интерфейса и взаимодействия с пользователем на три отдельных компонента, таким образом, чтобы изменения в одном из компонентов минимально влияли на остальные. Это делает данный подход оптимальным для небольших приложений, так как сокращается время разработки. В случае, если в будущем проектом будут заниматься несколько человек, каждый член команды сможет быстро разобраться в коде и внести изменения без значительных трудностей. Этот паттерн предполагает наличие трех основных типов классов.

1. Model – классы, отвечающие за хранение данных приложения и методы работы с этими данными.

2. View – классы, отвечающие за отображение интерфейса приложения. Они показывают данные, полученные от контроллеров, и реагируют на действия пользователя (например, нажатие кнопки), передавая управление соответствующему контроллеру.

3. Controller – классы, которые обеспечивают связь между Model и View. Они обрабатывают изменения в Model и реагируют на действия пользователя, переданные от View.

Эта архитектура обеспечивает четкое разделение ответственности между компонентами, что улучшает тестируемость и упрощает поддержку и расширение приложения.

3.2. **Описание компонентов**

Компонент Controller содержит следующие классы:

1. **ReminderListViewController**: Входная точка приложения, главный экран, на котором пользователи могут видеть список всех своих напоминаний. Определяет навигацию между страницами;
2. **ReminderViewController**: Экран для просмотра деталей конкретного напоминания;
3. **EditorViewController**: Экран для создания новых напоминаний или редактирования существующих;
4. **StatisticsViewController**: Экран, предоставляющий пользователям статистику выполнения их напоминаний.

Компонент Model содержит классы описывающие модель приложения:

1. **Reminder**: Класс, отвечающий за хранение данных модели напоминания;
2. **ReminderListStyle**: Класс, отвечающий за изменения содержимого таблицы напоминаний;
3. **ReminderStore**: Класс, отвечающий за синхронизацию напоминаний с базой данных;
4. **EKReminder**: Класс, обновляющий содержимое модели напоминания;
5. **TodayError**: Класс, отвечающий за обработку ошибок путем выбрасывания исключений в случае проблем;

Компонент View содержит следующие классы:

1. **ReminderItemListCell**: Ячейка таблицы, отображающая содержимое напоминания;
2. **ReminderCreationCell**: Ячейка таблицы, позволяющая пользователям перейти на экран создания нового напоминания;
3. **ProgressHeaderView**: Круговая диаграмма, показывающая количество выполненных задач за короткий промежуток времени;
4. **DailyStatisticsView**: Вид, отображающий график выполнения задач за 7 или 10 дней;
5. **MonthStatisticsView**: Вид, отображающий график выполнения задач за год.

Компонент EventKit:

1. EKEventStore: Класс для работы с хранилищем событий и напоминаний;
2. EKReminder: Класс для работы с напоминаниями.

3.3. Модель базы данных

Для данного приложения была использована существующая в iOS база данных напоминаний, интегрированная с помощью фреймворка EventKit, обеспечивающего доступ к данным из приложения заметок. Из этой БД будут использованы следующие поля:

1. **id**: Это уникальный идентификатор каждого напоминания;
2. **title**: Название напоминания;
3. **dueDate**: Дата и время, к которым нужно выполнить напоминание;
4. **notes**: Дополнительные заметки к напоминанию;
5. **isComplete**: Флаг, указывающий, выполнено ли напоминание.

Таблица 4. Таблица напоминаний.

|  |  |
| --- | --- |
| **Название поля** | **Тип поля** |
| **id** | String = UUID().uuidString |
| **title** | String |
| dueDate | Date |
| notes | String? = nil |
| **isComplete** | Bool = false |

3.4. Процесс работы с системой

В качестве иллюстрации процесса работы EventKit API была создана диаграмма деятельности, приведенная на рисунке 6.

Диаграмма описывает процесс работы с EventKit API для управления календарными событиями и напоминаниями. Сначала приложение запрашивает доступ (Request Access) к календарю, затем проверяет статус доступа (Check Status). Если доступ предоставлен, приложение продолжает чтение, создание, изменение и удаление событий (Read/Save/Remove). В случае отказа в доступе, оно обрабатывает ошибку и уведомляет пользователя (Proceed or Error).

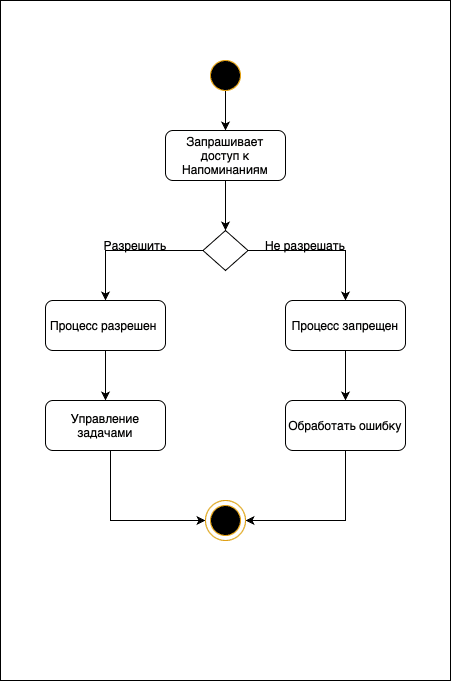


Рисунок 6 –Процесс работы EventKit

3.5. Проектирование интерфейса

В процессе проектирования интерфейса приложения были определены основные экраны и элементы управления, которые обеспечат пользователю удобный доступ к функциональности приложения.

Главный экран

Главный экран схематично показан на рисунке 7, содержит в себе следующие элементы:

* Таблица с заголовками – представляющая список задач.
* **График выполнения задач** – визуализация данных, отображающая текущий прогресс.
* **NavigationItem** – навигационный элемент, для выбора временного диапазона задачи.
* **CreateButton** – кнопка для создания новой задачи.
* **ProgressButton –** кнопка для отображения прогресса.

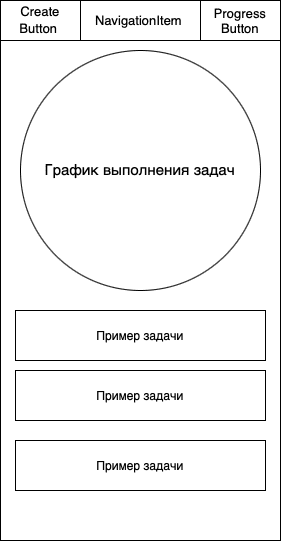


Рисунок 7 – Макет главного экрана

При нажатии на задачу открывается экран с подробной информацией о задаче, схематично показан на рисунке 8, который представлен в виде модального окна показан на рисунке 8:

1. Title – название текущего экрана;
2. **Заголовок задачи** – текстовое поле с названием задачи;
3. **Время выполнения задачи** – поле для указания времени;
4. **Пояснение к задаче** – расширенное текстовое поле для подробного описания задачи;
5. **EditButton** – **кнопка редактирования** задачи, при нажатии открывается экран редактирования задачи.

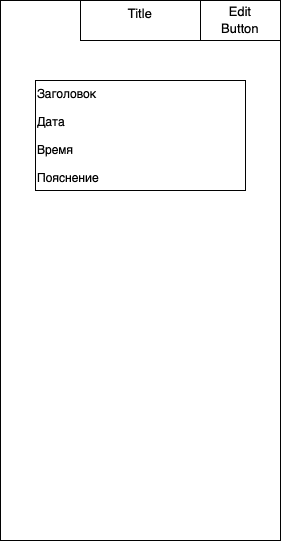


Рисунок 8 – Макет экрана просмотра задачи

Экран редактирования, он же и экран создания заметки, схематично показан на рисунке 9, отображает следующие элементы:

1. **Поле для ввода заголовка** –текстовое поле для изменения названия задачи;
2. **Большой календарь и время** – компоненты для выбора даты и времени выполнения задачи;
3. **Расширенное пояснение к заметке** – текстовое поле для детального описания задачи;
4. SaveButton – сохранение заметки.

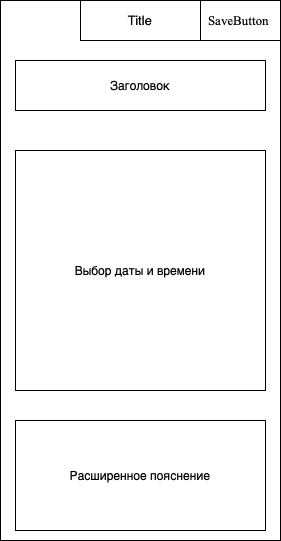


Рисунок 9 – Макет экрана редактирования

С главного экрана, который показан на рисунке 10 можно перейти на экран статистики, который содержит два графика:

* График статистикиза 7 дней — визуализация данных за последнюю неделю, с возможностью изменения диапазона на 10 дней.
* График статистики за 1 год — отображает данные за последний год, предоставляя пользователю возможность анализа долгосрочных тенденций.

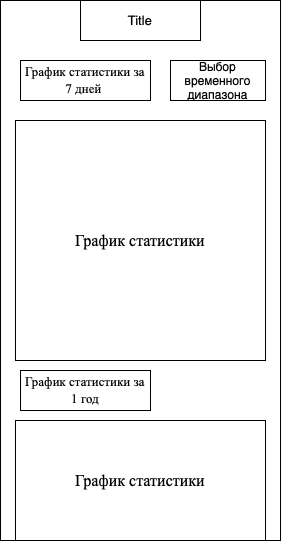


Рисунок 10 – Макет экрана статистики выполнения

Каждый из экранов и элементов интерфейса был продуман для обеспечения интуитивного и эффективного взаимодействия пользователя с приложением.

# 4. РЕАЛИЗАЦИЯ СИСТЕМЫ

* 1. Анализ существующих решений для реализации проекта

Был выбран метод нативной разработки, который включает использование среды разработки Xcode и фреймворка UIKit на языке программирования Swift для создания мобильного приложения. При таком подходе используется язык программирования, предоставляемый платформой разработчика.

По сравнению с кроссплатформенной разработкой, нативный подход обеспечивает более высокую производительность и гибкость в реализации за счет полного использования возможностей мобильной операционной системы. Однако выбор нативной разработки ограничивает использование приложения только на одной платформе.

Для создания пользовательского интерфейса Apple предлагает два варианта: UIKit и SwiftUI. В данном проекте был выбран фреймворк UIKit из-за его большей гибкости в программировании и удобства.

После проведения анализа инструментов и подходов к разработке было принято решение в пользу нативной разработки, учитывая такие критерии как простота тестирования, скорость работы приложения и гибкость в реализации.

Для разработки всего проекта было решено использовать язык программирования Swift с использованием UIKit на iOS. Swift - это мощный и интуитивно понятный язык, разработанный Apple для создания приложений под iOS, macOS, watchOS и tvOS. UIKit - это фреймворк, предоставляемый Apple для создания графического интерфейса пользовательского приложения на iOS.

Для работы с базой данных в приложении было необходимо добавить соответствующие разрешения. Для доступа к заметкам и работы с заметками в приложении было добавлено разрешение в Info.plist (листинг 2).

Листинг 2 - Разрешение на доступ к заметкам в Info.plist

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">

<plist version="1.0">

<string>Your reminders can help you stay on track</string>

</plist>

* 1. Реализация компонента ReminderStore

Компонент ReminderStore является ключевой частью приложения, обеспечивающей управление данными о напоминаниях пользователя. Он представляет собой набор методов для запроса доступа к календарю, чтения всех существующих напоминаний, сохранения новых или обновления существующих, а также удаления напоминаний.

* requestAccess: метод для запроса доступа к календарю пользователя. В зависимости от текущего статуса доступа происходит либо автоматическое разрешение доступа, либо выбрасывается исключение. Реализация показана в листинге 1

Листинг 1.

func requestAccess() async throws {

let status = EKEventStore.authorizationStatus(for: .reminder)

switch status {

case .authorized:

return

case .restricted:

throw TodayError.accessRestricted

case .notDetermined:

let accessGranted = try await ekStore.requestAccess(to: .reminder)

guard accessGranted else {

throw TodayError.accessDenied

}

case .denied:

throw TodayError.accessDenied

@unknown default:

throw TodayError.unknown

}

}

* readAll: метод для асинхронного чтения всех существующих напоминаний пользователя из календаря. Реализация показана в листинге 2.

Листинг 2.

**func** readAll() **async** **throws** -> [Reminder] {

**guard** isAvailable **else** {

**throw** TodayError.accessDenied

}

**let** predicate = ekStore.predicateForReminders(in: **nil**)

**let** ekReminders = **try** **await** ekStore.reminders(matching: predicate)

**let** reminders: [Reminder] = **try** ekReminders.compactMap { ekReminder **in**

**do** {

**return** **try** Reminder(with: ekReminder)

} **catch** TodayError.reminderHasNoDueDate {

**return** **nil**

}

}

**return** reminders

}

* save: метод для сохранения нового или обновления существующего напоминания в календаре. Реализация показана в листинге 3.

Листинг 3.

@discardableResult

func save(\_ reminder: Reminder) throws -> Reminder.ID {

guard isAvailable else {

throw TodayError.accessDenied

}

let ekReminder: EKReminder

do {

ekReminder = try read(with: reminder.id)

} catch {

ekReminder = EKReminder(eventStore: ekStore)

}

ekReminder.update(using: reminder, in: ekStore)

try ekStore.save(ekReminder, commit: true)

return ekReminder.calendarItemIdentifier

}

* remove: метод для удаления напоминания по его идентификатору. Реализация показана в листинге 4.

Листинг 4.

**`**func remove(with id: Reminder.ID) throws {

guard isAvailable else {

throw TodayError.accessDenied

}

let ekReminder = try read(with: id)

try ekStore.remove(ekReminder, commit: true)

}

* 1. Реализация интерфейса системы

Главный экран приложения отображает список задач с возможностью просмотра текущего прогресса выполнения и навигации по временному диапазону задач.

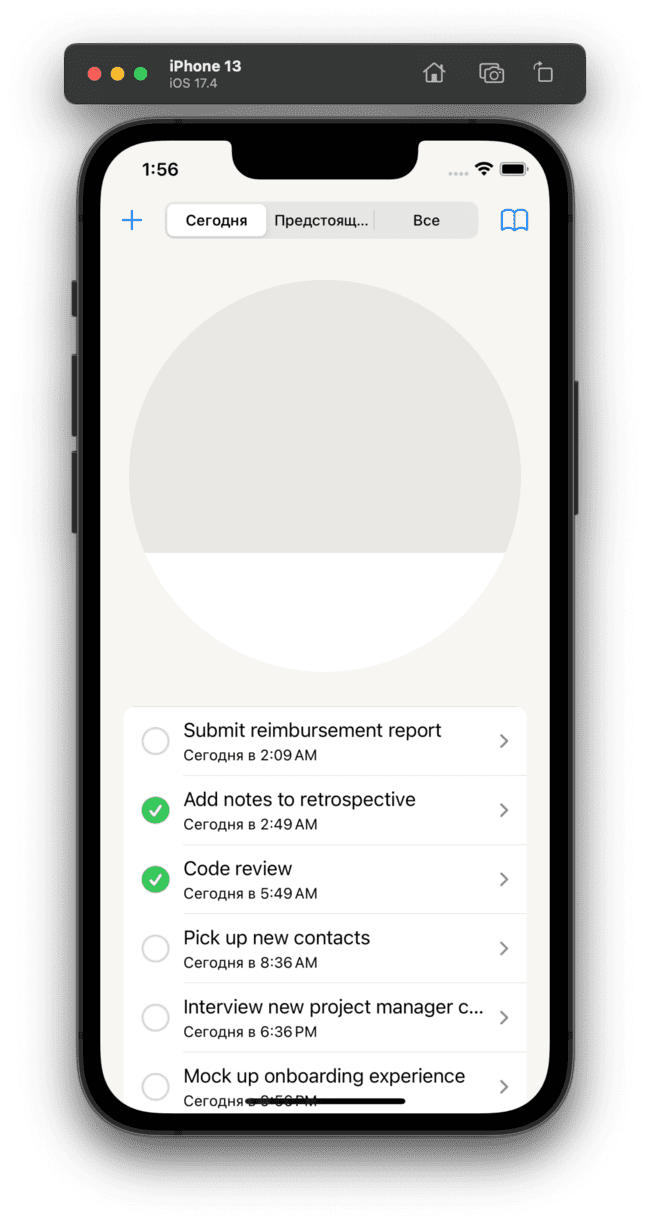
****

Рисунок 11 – главный экран.

Экран просмотра задачи предоставляет детальную информацию о выбранной задаче, включая её название, время выполнения и расширенное описание.

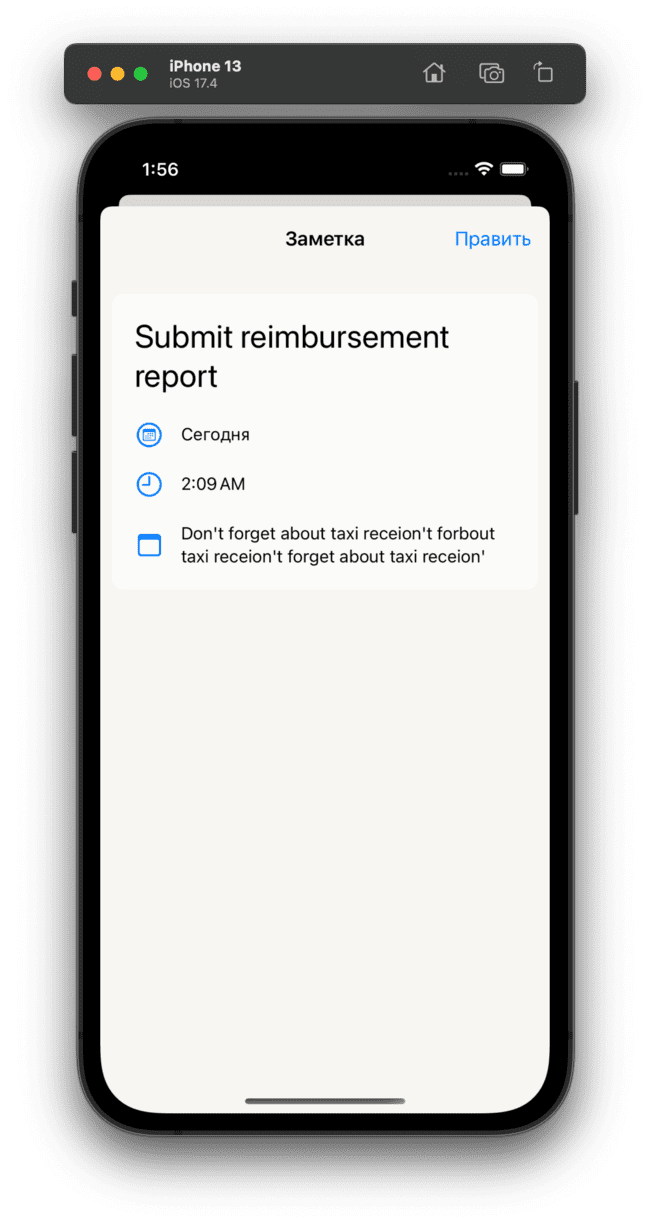


Рисунок 12 – Экран просмотра задачи.

Экран редактирования задачи позволяет пользователю изменять заголовок, время выполнения и описание задачи, а также сохранять внесённые изменения.

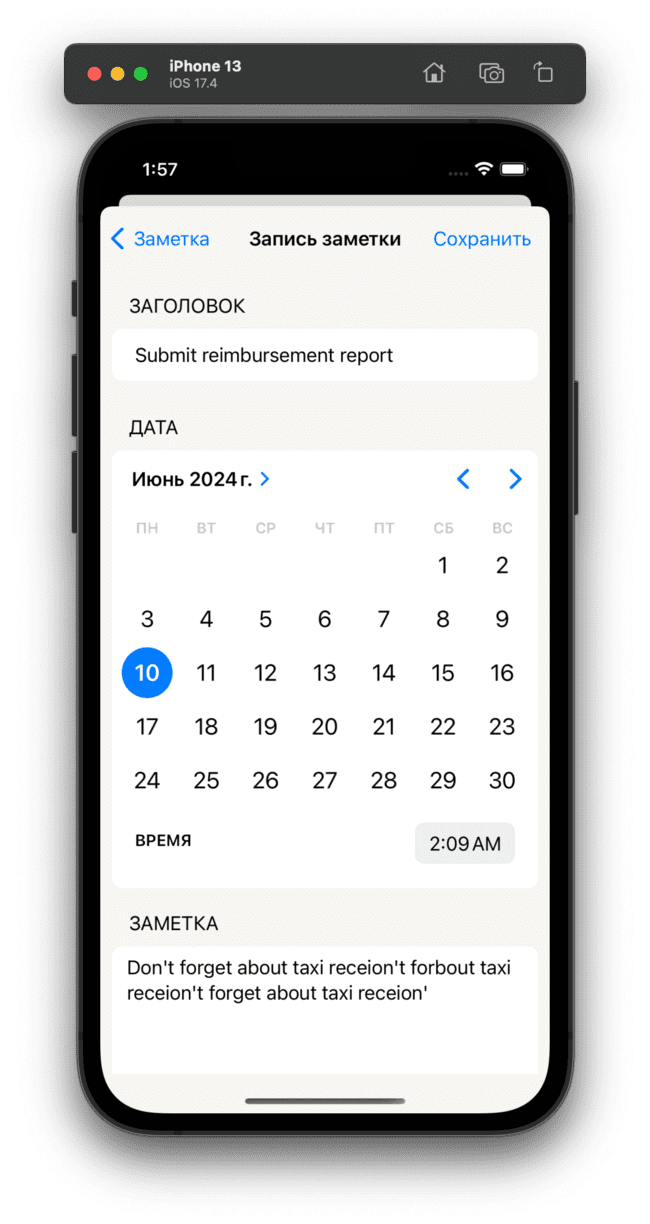


Рисунок 13 – Экран редактирования задачи.

Экран анализа выполнения задач содержит графики статистики за последние 7 дней и за последний год для анализа эффективности и долгосрочных тенденций работы пользователя.

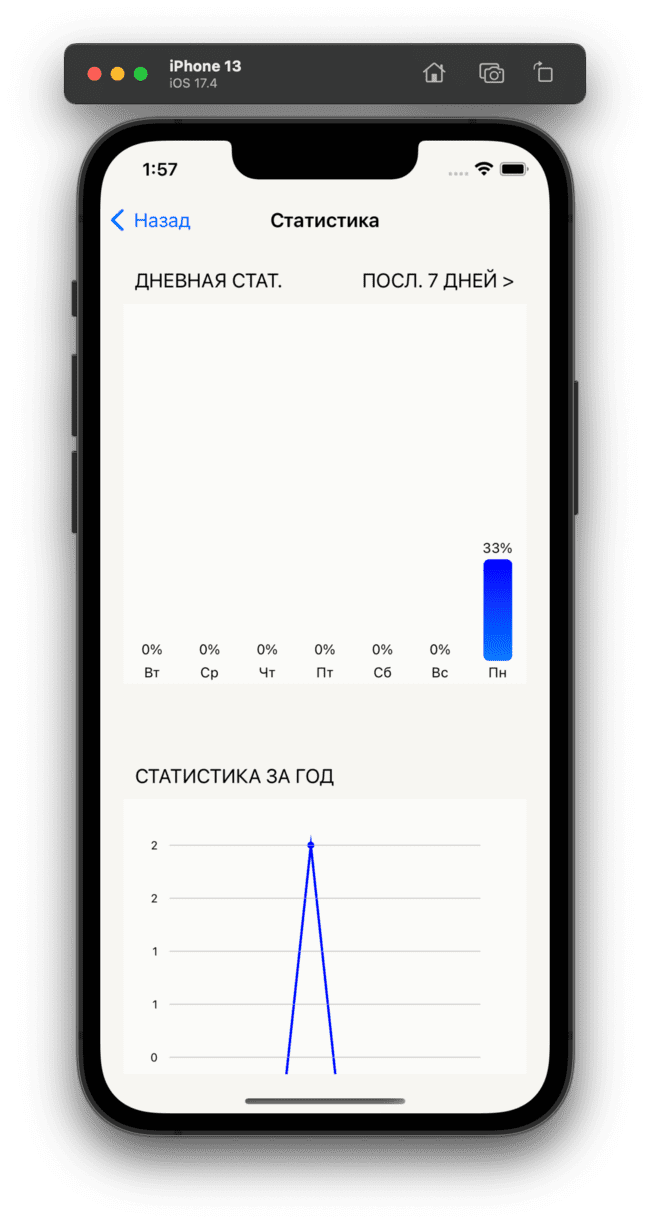


Рисунок 14 – Экран статистики.

Каждый экран реализован в соответствии с предоставленными макетами, обеспечивая интуитивно понятное взаимодействие пользователя с приложением и эффективное управление задачами.

# 5. ТЕСТИРОВАНИЕ

Для функционального тестирования и тестирования пользовательского интерфейса получившегося приложения, было использовано устройство iPhone XR и эмулятор iPhone 13 с версиями iOS 17.4. Набор тестов и их результаты представлены в таблице 5.

Таблица 5 – Протоколы тестирования системы

| № | **Название  теста** | **Шаги** | **Ожидаемый  результат** | **Тест пройден?** |
| --- | --- | --- | --- | --- |
| 1 | Разрешение доступа к данным приложения заметки | При первом использовании  приложения, система осуществляет запрос на разрешение доступа  к данным пользователя. | Без разрешения доступа к  данным недоступно полноценное использо-  вание приложения, после  разрешения приложение  снимает ограничения. | Да |
| 2 | Добавление  задачи | При добавление задачи учитываются все возможные варианты ее создания. | Сохраняется в базе данных. | Да |
| 3 | Сохранение  отредактиро-  ванной задачи | При внесении изменений в задачу учитываются все возможные способы ее редактирования. | Сохраняется и изменяется в базе данных. | Да |
| 4 | Удаление  задачи | При удаление задачи учитываются все возможные варианты ее удаления. | Удаление задачи из базы данных | Да |
| 5 | Отображение списка задач | При открытии главного экрана приложения отображается список всех задач пользователя | Все созданные задачи отображаются корректно и доступны для просмотра. | Да |
| 6 | Фильтрация задач по категориям | Пользователь может выбрать определенную категорию задач для отображения | Только задачи выбранной категории отображаются, остальные скрываются | Да |
| 7 | Отображение статистики выполнения задач | На экране статистики отображаются графики выполнения задач за разные периоды времени | Графики корректно отображают данные о выполненных задачах | Да |
| 8 | Мультиязычность интерфейса | Приложение поддерживает выбор языка интерфейса пользователя | Интерфейс приложения отображается на выбранном языке без ошибок и согласно выбору пользователя | Да |

# ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано iOS–приложение для отслеживания задач.

* + - 1. Был проведен анализ аналогичных решений – были выявлены их преимущества и недостатки.
      2. Были проведены анализ функциональных и нефункциональных требований и постановка задачи.
      3. Было спроектировано и разработано iOS–приложение.
      4. Было проведено тестирование приложения.

Разработанное приложение успешно протестировано и реализует все поставленные функциональные и нефункциональные требования.

# ЛИТЕРАТУРА

1.Усов В. У76 Swift. Основы разработки приложений под iOS,

iPadOS и macOS. 6-е изд. дополненное и переработанное. – СПб.: Питер,

2021. – 544 с.: ил. (Серия «Библиотека программиста») (дата обращения:

10.06.2024г.).

2. Харазян А. Язык Swift. Самоучитель. – СПб.: БХВ-Петербург,

2016. – 176 с. (дата обращения: 10.06.2024г.).

3. Грей Э. Swift. Карманный справочник. Программирование в

среде iOS и ОS X. – М.: Вильямс, 2016. – 288 с. (дата обращения:

10.06.2024г.).

4. Марк Д. Swift. Разработка приложений в среде Xcode для iPhone

и iPad с использованием iOS SDK. – М.: Вильямс, 2016. – 816 с. (дата

обращения: 10.06.2024г.).

5. Пайлон Т. Программируем для iPhone и iPad. – СПб.: Питер,

2014. – 336 с. (дата обращения: 10.06.2024г.).

6. Swift: Компилируемый язык программирования общего

назначения: [Электронный ресурс] URL: https://developer.apple.com/swift/

(дата обращения: 10.06.2024г.).

7. Bessarabova E. MVP vs. MVC vs. MVVM vs.VIPER. What is Better

For iOS Development? [Электронный ресурс] URL:

https://themindstudios.com/blog/mvp-vs-mvc-vs-mvvm-vs-viper/ (дата

обращения: 10.06.2024г.).

8. Wals D. Mastering iOS 10 Programming. – Birmingham: Packt

Publishing, 2016. – 543 с. (дата обращения: 10.06.2024г.).

9. Документация EventKit. [Электронный ресурс] URL:

https://developer.apple.com/documentation/eventkit (дата обращения: 10.06.2024г.)

10. Human Interface Guidelines: Рекомендации по интерфейсу для

iOS: [Электронный ресурс] URL: https://developer.apple.com/design/humani

nterface-guidelines/ (дата обращения: 10.06.2024г.).

11. UIKit: Набор элементов, из которых состоит интерфейс

приложения: [Электронный ресурс] URL: https://developer.apple.com/docum

entation/uikit. (дата обращения: 10.06.2024г.).

12. App Store: магазин приложений для iOS устройств.

[Электронный ресурс] URL: https://www.apple.com/app-store/ (дата

обращения: 10.06.2024г.).

13. Приложение «Sticky Notes & To do list» [Электронный ресурс] URL: https://apps.apple.com/ru/app/sticky-notes-to-do-list/id1534095481?l=en-GB (дата обращения: 10.06.2024г.).

14. Приложение «To Do List Widget» [Электронный ресурс] URL:

https://apps.apple.com/ru/app/to-do-list-widget/id1532165568?l=en-GB (дата обращения: 10.06.2024г.).

15. Приложение «Things 3» [Электронный ресурс] URL

https://apps.apple.com/ru/app/things3/id6475672390?l=en-GB (дата обращения: 10.06.2024г.).

# ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

**Таблица 1**. Спецификация вариантов использования «Просмотр задач»

|  |
| --- |
| Прецедент**:** Просмотр задач |
| ID: 1 |
| Аннотация: Пользователь просматривает ленту задач. |
| Главные актеры: Пользователь. |
| Второстепенные актеры: Нет. |
| Предусловия: Приложение запущено. Данные задачах. |
| Основной поток:   1. Пользователь открывает приложение и просматривает экран задач. |
| Постусловия: Нет. |
| Альтернативные потоки: Нет. |

**Таблица 2.** Спецификация вариантов использования «Просмотр списка задач»

|  |
| --- |
| Прецедент**:** Просмотр списка задач |
| ID: 2 |
| Аннотация: Пользователь просматривает список задач. |
| Главные актеры: Пользователь. |
| Второстепенные актеры: Нет. |
| Предусловия: Приложение запущено. Данные задачах. |
| Основной поток:   1. Выбор временного диапазона. |
| Постусловия: Нет. |
| Альтернативные потоки: Нет. |

**Таблица 3.** Спецификация вариантов использования «Создание/Изменение задач»

|  |
| --- |
| Прецедент**:** Создание/Изменение задачи |
| ID: 3 |
| Аннотация: Пользователь создает/изменяет задачу. |
| Главные актеры: Пользователь. |
| Второстепенные актеры: Нет. |
| Предусловия: Приложение запущено. Данные о задачах. |
| Основной поток:   1. Наименование заметки 2. Выбор даты и время 3. Добавление описания |
| Постусловия: Нет. |
| Альтернативные потоки: Нет. |

**Таблица 4.** Спецификация вариантов использования «Посмотреть статистику выполнения задач»

|  |
| --- |
| Прецедент**:** Посмотреть статистику выполнения задач. |
| ID: 4 |
| Аннотация: Пользователь просматривает статистику выполнения. |
| Главные актеры: Пользователь. |
| Второстепенные актеры: Нет. |
| Предусловия: Приложение запущено. Данные задачах. |
| Основной поток:   1. Выбор временного диапазона |
| Постусловия: Нет. |
| Альтернативные потоки: Нет. |

**Приложение Б.**

Листинг 5 – Пример класса EKReminder

import EventKit

import Foundation

extension EKReminder {

func update(using reminder: Reminder, in store: EKEventStore) {

title = reminder.title

notes = reminder.notes

isCompleted = reminder.isComplete

calendar = store.defaultCalendarForNewReminders()

alarms?.forEach { alarm in

guard let absoluteDate = alarm.absoluteDate else { return }

let comparison = Locale.current.calendar.compare(

reminder.dueDate, to: absoluteDate, toGranularity: .minute)

if comparison != .orderedSame {

removeAlarm(alarm)

}

}

if !hasAlarms {

addAlarm(EKAlarm(absoluteDate: reminder.dueDate))

}

}

}

extension EKEventStore {

func reminders(matching predicate: NSPredicate) async throws -> [EKReminder] {

try await withCheckedThrowingContinuation { continuation in

fetchReminders(matching: predicate) { reminders in

if let reminders {

continuation.resume(returning: reminders)

} else {

continuation.resume(throwing: TodayError.failedReadingReminders)

}

}

}

}

}