Fuctivity

Архитектурный документ

Раздел регистрации изменений

Версия документа	Дата	Описание изменения	Автор изменения
01	17.03.2023	Создание первой версии архитектурного документа [П1]	Сосин В. М. Акст Е.Е. Дерезовский И. Д. Якунюшкин Н. О.
02	10.04.2023	п 2.3 Исправлен пункт “Ключевые ограничения, добавлены новые ключевые ограничения. Так, что теперь ограничения относятся к разрабатываемой программе, а не к техническому оборудованию; п 3 Удалена избыточная информация используемых фреймворков и библиотек; п 5.1 Изменены факторы и мотивировка технического решения №1; п 5.2 Приведено техническое описание решения новой проблемы, которое влияет на архитектуру приложения - способ хранения данных.	Дерезовский И. Д.
03	11.04.2023	Распределениие разделов документа по их авторам; п 2.1 Исключение из ключевых лиц технического писателя, так как он не взаимодействует с системой; п 2.2 Внесены пояснения в обозначение нефункциональных требований. п 4.9, 4.10 Требования к временным ограничениям реагирования приложения перенесены из Представления производительности в раздел Атрибутов качества	Якунюшкин Н. О.
04	22.04.2023	п 2.2 Добавлены требования к демонстрации статистики деятельности пользователя в приложении; п 3 Архитектура проекта исправлена с MVC на MVVM;	Дерезовский И. Д.

		4.2 Внесены изменения в диаграмму классов и диаграмму пакетов ; п 4.3 Исправлено представление архитектуры процессов; п 4.5 Добавлен способ установки библиотеки SwifUI-Charts для построения графиков, используемых в статистике пользователя; п 4.7 Добавлен и описан метод хранения и хеширования пароля пользователя; п 5.1 Исправлено техническое описание решения проблемы выбора архитектуры программы (с MVC на MVVM); п 5.4 Добавлено техническое описание решения проблемы выбора способа отображения графиков в разделе статистики отдыха пользователя. п 5.5 Добавлено техническое описание решения проблемы расположения классов и файлов в проекте	
05	12.05.2023	Добавлен пункт 6 Проверка результативности внесенных в проект изменений	Сосин В. М. Дерезовский И. Д. Акст Е.Е. Якунюшкин Н.О.
06	21.05.2023	п 4.2 Добавлено описание диаграммы классов, исправлена диаграмма пакетов	Дерезовский И. Д.
07	22.05.2023	п 6.2 Обновление информации о разделении файлов в проекте п 6.3 Обновление описания тестирования навигации Добавлен пункт 7 Дальнейшие улучшения системы	Сосин В. М. Якунюшкин Н.О. Акст Е.Е

1. Введение

1.1. Название проекта

Приложение для планирования отдыха «Fuctivity»

1.2. Задействованные архитектурные представления

Представление прецедентов предоставлено и описывает требуемую функциональность системы, системное окружение и связи между ними. Оно необходимо для общего понимания того, кто и для чего будет использовать систему и какие минимальные сценарии использования она должна покрывать.

Логическое представление предоставляется для верхнеуровневого ознакомления с тем, как работает система, по какому принципу построена и функционирует. Это является очень важной частью при исследовании и использовании проекта.

Представление архитектуры процессов предоставлено и содержит наиболее полное представление о поведении системы, об особенностях реализации процессов в ней. Это является очень важной частью при исследовании и использовании проекта.

Физическое представление архитектуры предоставлено и содержит информацию о требованиях к аппаратным средствам, на которых возможна работа проекта. Оно необходимо для того, чтобы физический запуск и выполнение приложения были возможны и происходили корректно.

Представление развертывания предоставлено и необходимо, так как описывает путь создания установочного файла, а также способ его запуска, чтобы любой потенциальный пользователь смог начать работу с приложением.

Представление архитектуры данных важно для понимания имеющихся в системе сущностей. Кроме того, важно выделить используемый инструментарий для хранения данных, так как количество доступных вариантов крайне велико, и тот или иной выбор может определить дальнейшее развитие проекта.

Представление архитектуры безопасности рассказывает о том, что в рамках данного проекта возникает естественная необходимость в аутентификации пользователя для обращения и взаимодействия с ним. На текущем уровне реализации необходимости в авторизации нет, однако используемая внешняя система позволит реализовать это при необходимости. В соответствующем разделе выбранная система описана более подробно.

Представление реализации и разработки описание инструментов, использованных при реализации, и описание того, как и где хранится исходный код проекта. Раздел необходим для возможности ознакомления с кодом.

Представление производительности представлено, так как, ввиду особенностей мобильной разработки, производительность системы оказывает большое влияние на охват аудитории и опыт пользователей. Однако, не приведено описание алгоритмов, так как они не используются при реализации системы.

Атрибуты качества системы - раздел представлен и необходим, так как даёт наиболее полное представление о нефункциональных требованиях к системе.

1.3. Контекст задачи и среда функционирования системы

На сегодня создано множество программ для ПК и мобильных приложений, выполняющих функции планировщиков. Культура продуктивности и переработок создаёт майндсет, в котором отдых или “ничегонеделание” - это что-то непродуктивное, а значит плохое.

Люди часто планируют на день огромное количество дел, но не учитывают, что надо успеть не только поработать, но и отдохнуть.

Данное мобильное приложение для планирования отдыха с простым и интуитивно понятным интерфейсом позволяет легко вносить в своё планирование дня время на отдых и любимые занятия.

1.4. Рамки и цели проекта

Fuctivity — это мобильное приложение, которое специально разработано и адаптировано под мобильные устройства с операционной системой iOS не ниже версии 13.0, целью которого является планирование отдыха. Благодаря систематизации всех необходимых задач и действий, приложение помогает контролировать отдых и улучшить жизнь за счёт повышения качества отдыха. Приложение представляет собой простой список ежедневных задач для отдыха, который можно редактировать, разделять на этапы и получать уведомления с напоминаниями.

2. Архитектурные факторы (цели и ограничения)

2.1. Ключевые заинтересованные лица

Действующее лицо	Заинтересованность в системе
Разработчик	Поддержка существующего функционала и добавление нового, написание кода. Документирование исходного кода
Пользователь	Быстрое и удобное составление списка задач для отдыха, наглядное представление задач, напоминание о предстоящих задачах

2.2. Ключевые требования к системе

Функциональные требования, которые программный продукт должен выполнять:

- зарегистрировать новый профиль пользователя: для регистрации вводится имя, почта и пароль;
- осуществить вход в существующий профиль пользователя: для входа в аккаунт используются те же данные, что и при регистрации (имя, почта, пароль);
- указывать рабочие дни недели: пользователем выбираются его рабочие дни недели;
- указывать время длительности рабочего дня: пользователем вводится примерная длительность его рабочего времени за выбранные дни недели;
- указывать время, которое планируется потратить для отдыха на день: пользователем выбирается среднее время в часах, которое он готов потратить на отдых;
- добавлять новую активность отдыха: пользователем выбирается длительность отдыха, его описание, а также его время начала и окончания;
- указывать время отправки уведомления: время, в которое пользователю приходит уведомление о выбранном отдыхе;
- просматривать список добавленных задач: показывается календарь, в котором графически показываются добавленные пользователем активности;
- показывать статистику по времени, отведённому на отдых: демонстрация столбчатых и линейных графиков по количеству часов, отведённых на отдых, на каждый день недели;
- получение уведомлений: пользователю приходит уведомление.

Нефункциональные требования, предъявляемые системе:

- Скорость - скорость загрузки приложения, реагирования на действия пользователя, отображения элементов интерфейса, восстановления после технических сбоев;

- Надежность - способность системы корректно реагировать на технические сбои, противостоять им и сохранять работоспособность;
- Безопасность - способность системы корректно реагировать на внешние угрозы проникновения в систему, надёжность хранения личных данных пользователя;
- Публичная и полная документация - соответствие общепринятым стандартам кодстайла для языка Swift.

2.3. Ключевые ограничения

Ключевые ограничения, накладываемые на разрабатываемый продукт:

- приложение должно работать на устройствах с установленной операционной системой iOS версии не ниже 13.0;
- приложение должно работать на устройствах с объёмом свободной встроенной памяти не меньше 100 МБ;
- приложение должно работать на устройствах с объёмом оперативной памяти не меньше 1 ГБ;
- исходный код приложения должен быть структурирован и стандартизирован, соблюдать принятые нормы кодстайла;
- приложение должно сопровождаться системой контроля версий;
- приложение должно работать на устройствах с сенсорным экраном диагональю не меньше 4,7 дюйма.

Открытый и задокументированный исходный код на языке программирования Swift версии 5.6.1 и выше.

3. Общее архитектурное решение

- **Пользовательский интерфейс**. Приложение написано на UIKit - фреймворке, позволяющем создавать пользовательские интерфейсы (UI), которые могут обрабатывать события касания и входные данные, управляя взаимодействиями между пользователем, системой и приложением. Для отображения графиков в статистике пользователя использовалась среда для создания интерфейсов SwiftUI, которая предоставляет различные библиотеки (SwifUI-Charts) для быстрого и удобного построения графиков, отображения постоянных и динамических данных

- **Общие использованные архитектурные решения.** Шаблон проектирования - MVVM (Model-View-ViewModel). В проекте использованы CalendarKit - библиотека пользовательского интерфейса Swift Calendar для iOS, которая позволяет настраивать календарь по необходимым параметрам с учетом локализации приложения, для презентабельного календаря с событиями, и для отправки локальных уведомлений используется фреймворк UserNotifications - набор API, который предоставляет унифицированный, объектно-ориентированный способ работы с локальными и удаленными уведомлениями.

- **Хранение данных**. В приложении Fuctivity используется локальное хранение данных пользователя, организованное с помощью UserDefaults - способ локального хранения небольших объемов данных, таких как настройки приложения, предпочтения пользователя и т.д. Данные хранятся в виде пар "ключ-значение", где ключ - это строковый идентификатор, а значение может быть любого типа, поддерживаемого plist.

3.1. Принципы проектирования

В основе архитектуры приложения лежит шаблон проектирования **MVVM**. Основная идея паттерна MVVM заключается в том, что свойства View совпадают со свойствами ViewModel/Model. При этом ViewModel не имеет ссылки на интерфейс представления. Изменение состояния ViewModel автоматически изменяет View, и наоборот. Для этого используется механизм связывания данных. Также характерная черта MVVM — двусторонняя коммуникация с View (рис. 1).

- Код модели Model хранит данные и связанную с ними логику, а также закрепляет структуру приложения. Код модели по шаблону определяет основные компоненты приложения: что такое «задача» и что такое «список».
- Код внешнего вида приложения, View, состоит из функций, которые отвечают за интерфейс и способы взаимодействия пользователя с ним. Представления создают на основе данных, собранных из модели. View определяет, как выглядят задачи.
- Код модели представления, ViewModel, которая служит прослойкой между View и Model. Он получает на вход пользовательский ввод, интерпретирует его и информирует о необходимых изменениях. Отправляет команды для обновления состояния, такие как сохранение изменений. Определяет, как пользователь добавляет новую задачу.

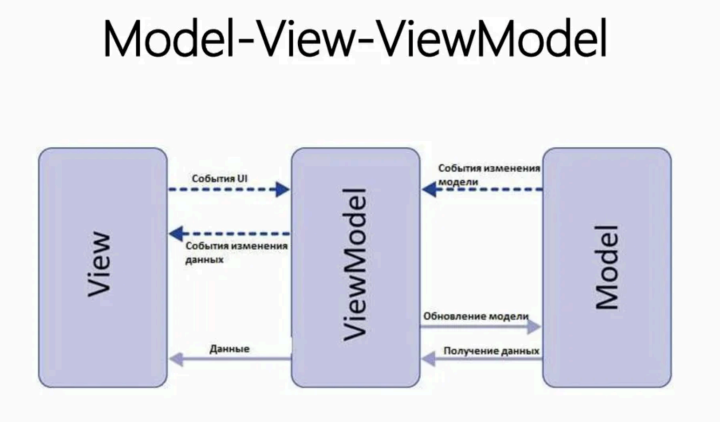


Рис 1. Схема модели MVVM

Система Fuctivity является наглядным примером использования **методологии объектно-ориентированного программирования**. Суть понятия объектно-ориентированного программирования в том, что все программы, написанные с применением этой парадигмы, состоят из объектов. Каждый объект — это определённая сущность со своими данными и набором доступных действий. В ходе анализа открытого исходного кода приложения несложно заметить активное применение таких принципов ООП, как инкапсуляция, наследование и полиморфизм.

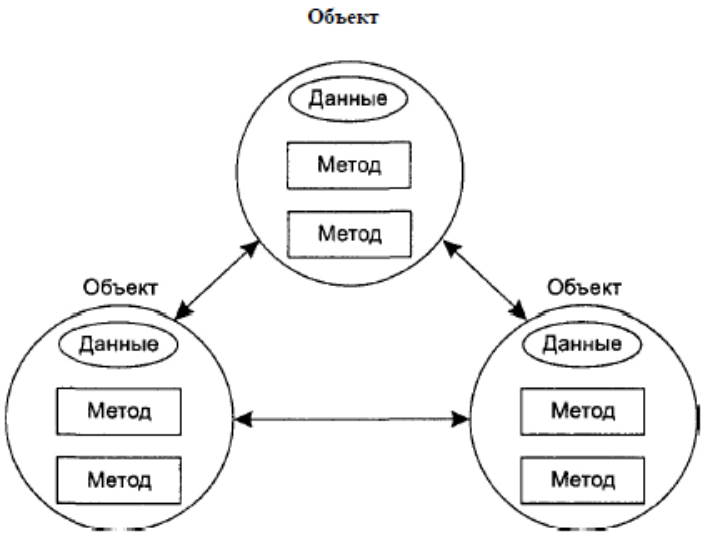


Рис 2. Схема представления модели ООП объектов

Вся информация, которая нужна для работы конкретного объекта, должна храниться внутри этого объекта. Если нужно вносить изменения, методы для этого тоже должны лежать в самом объекте — посторонние объекты и классы этого делать не могут. Для внешних объектов доступны только публичные атрибуты и методы. Такой принцип (инкапсуляция) обеспечивает безопасность и не даёт повредить данные внутри какого-то класса со стороны. Ещё он помогает избежать случайных зависимостей, когда из-за изменения одного объекта что-то ломается в другом.

При разработке приложения активно используется **шаблон событийно-ориентированной архитектуры** - подход к проектированию и разработке систем, в которых компоненты взаимодействуют между собой через асинхронную передачу сообщений, основанных на событиях.

Этот шаблон архитектуры ориентирована на обработку событий, которые происходят в системе, таких как изменение состояния объектов, действия пользователей и т.д. Компоненты системы могут отправлять сообщения, когда происходит определенное событие, и другие компоненты могут подписываться на эти сообщения для реагирования на эти события.

Шаблон событийно-ориентированной архитектуры также позволяет разделить приложение на множество более мелких, независимых компонентов, что упрощает разработку, тестирование и сопровождение системы.

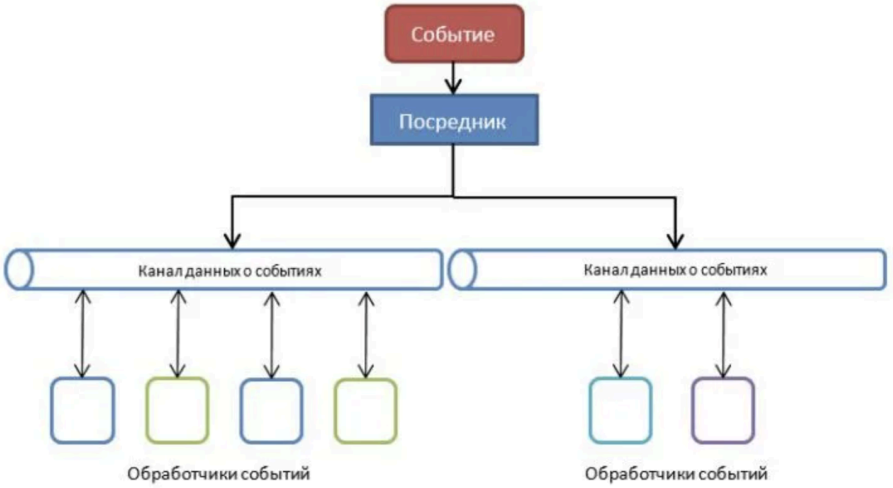
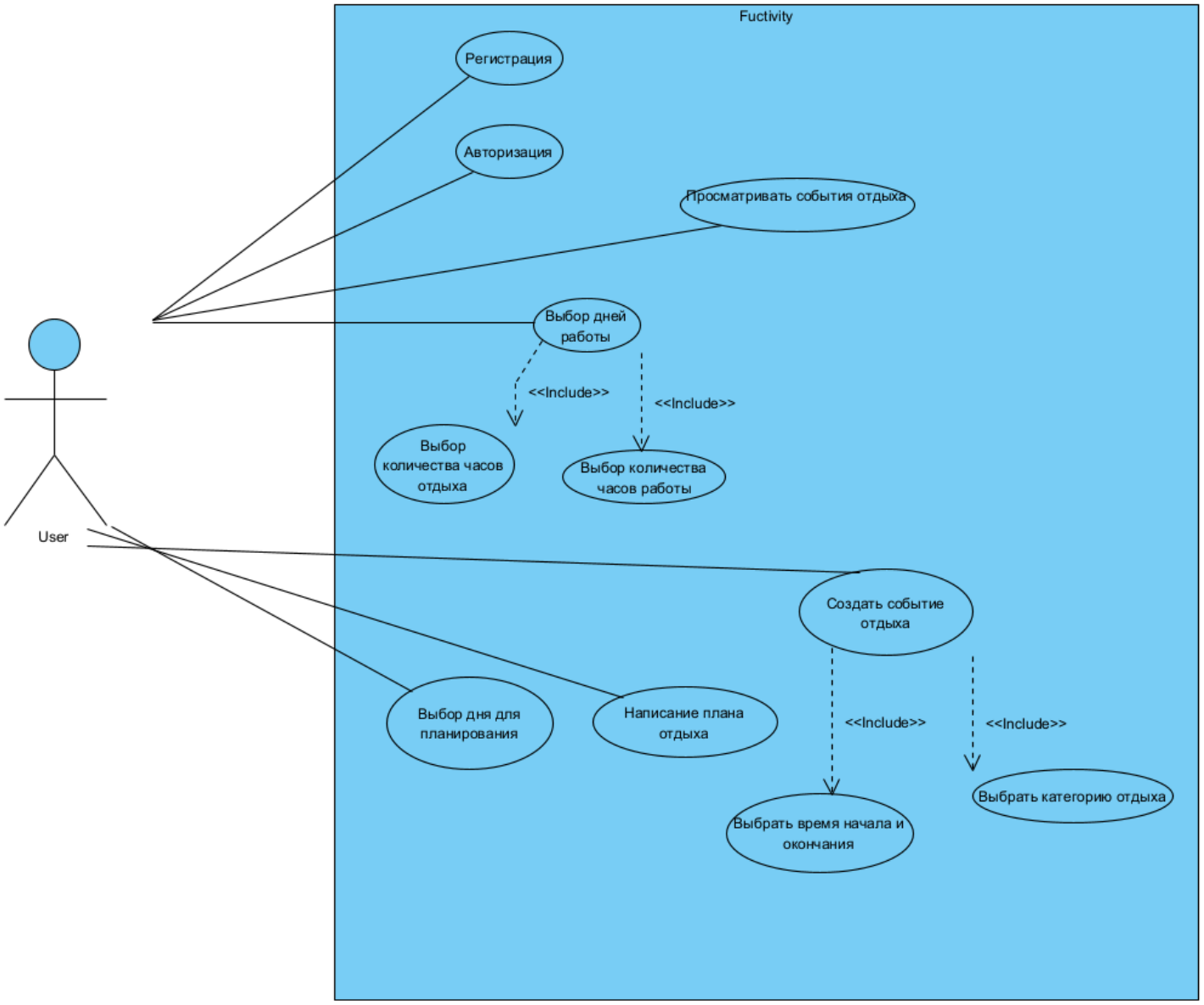


Рис 3. Схема событийно-ориентированной архитектуры

4. Архитектурные представления

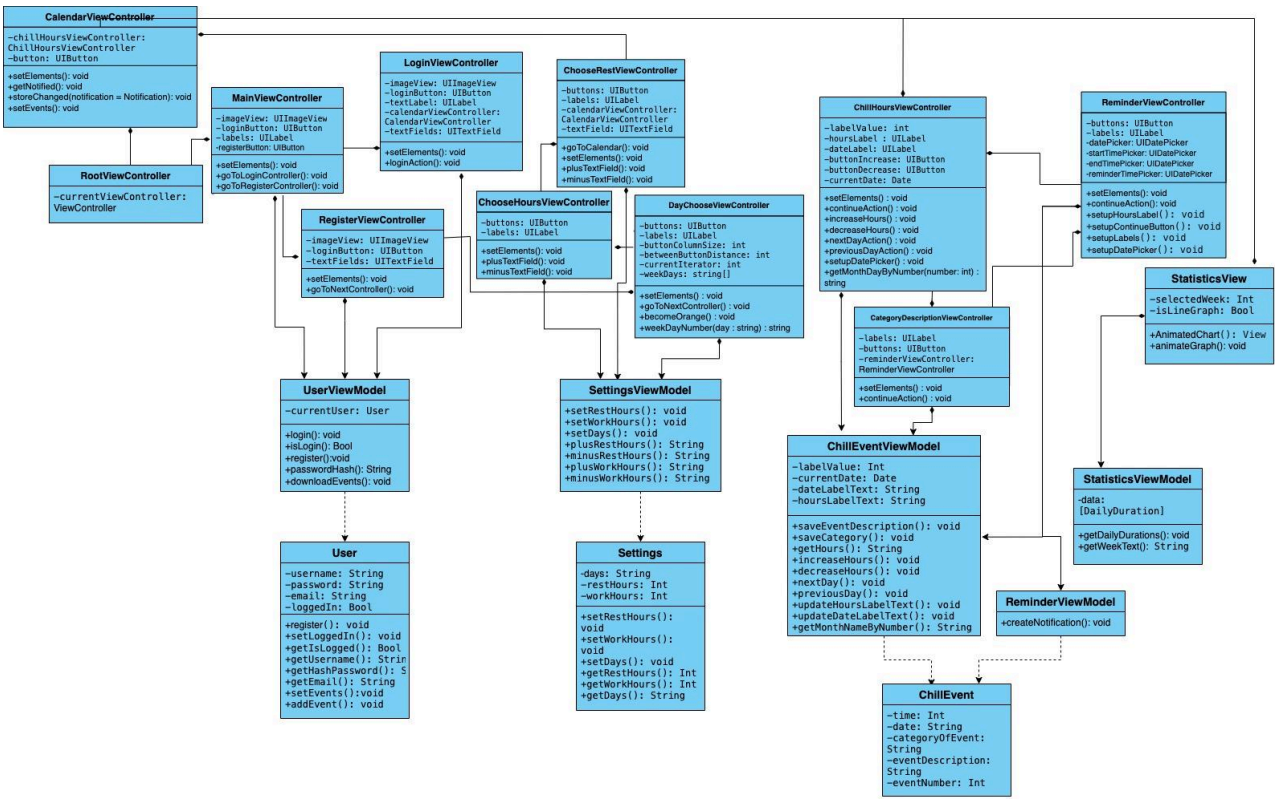
4.1. Представление прецедентов



4.2. Логическое представление

Использовались три вида разбиения - Model, View, ViewModel, которые помогают отделять логику приложения от его представления - теперь свойства View совпадают со свойствами ViewModel/Model. При этом ViewModel не имеет ссылки на интерфейс представления. Изменение состояния ViewModel автоматически изменяет View, и наоборот.

Ниже 4 представлено полученное взаимодействие классов:



Контроллер NavigationController запускается первым и во внутренней логике определяет, был ли уже ранее совершён успешный вход в приложение. Если был, то изначально будет

открываться контроллер календаря, иначе - контроллер регистрации и входа:

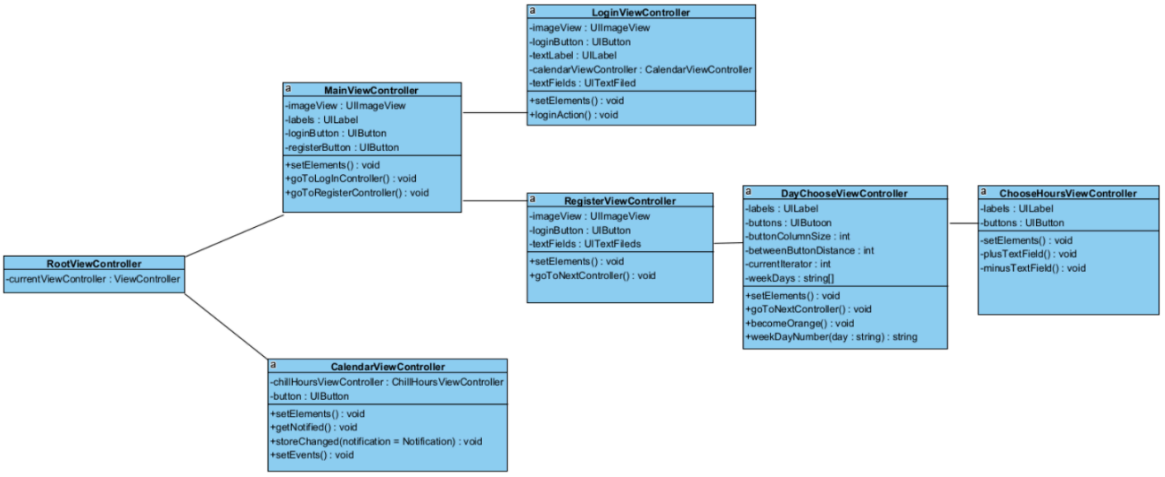
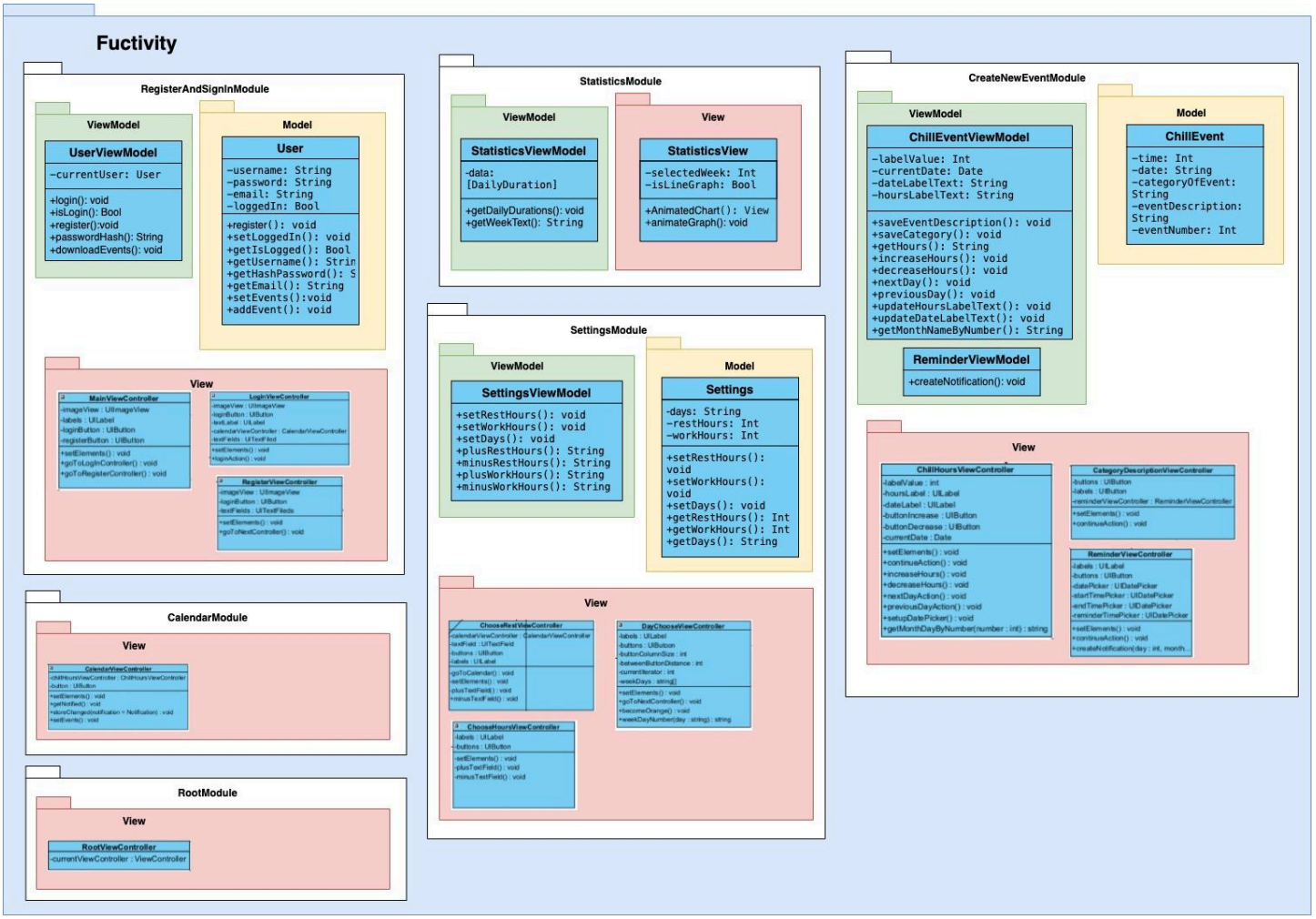


Диаграмма пакетов:



4.3. Представление архитектуры процессов

Во время запуска приложения запускается главный - корневой - контроллер RootController. Он же определяет во внутренней логике, был ли уже успешный вход в приложение. Если был, то изначально будет открываться контроллер календаря, иначе - контроллер регистрации и входа. Другие контроллеры создаются при их открытии.

Система поддерживает реагирования на события, связанные с действиями пользователя. Например, нажатия, удерживания и скроллинг. Это позволяют сделать viewController'ы, которые связывают элементы интерфейса с методами логики при взаимодействии с ними.

События привязываются к конкретным сущностям - представлениям, которые уже отвечают за логику обработки.

4.4. Физическое представление архитектуры

Для бесперебойной работы программного продукта требуется мобильное устройство с:

- сенсорным экраном диагональю не меньше 4,7 дюйма;
- операционной системой iOS не ниже версии 13.0;
- объемом свободной встроенной памяти не меньше 100 МБ;
- объемом оперативной памяти не меньше 1 ГБ;

Мобильное устройство также должно быть предназначено предназначено для эксплуатации в помещениях с искусственно-регулируемыми климатическими условиями, например, в отапливаемых и вентилируемых помещениях категории 4.1 согласно ГОСТ 15150-69 .

4.5. Представление развертывания

Программа предполагает развёртывание на операционной системе macOS (рекомендуется использовать последний пакет обновлений).

Загрузка системы осуществляется посредством клонирования репозитория github или скачивания и развертывания .zip файла. Для запуска программы пользователю необходимо открыть .xcodeproj файл в заранее установленной интегрированной среде разработки (IDE) Xcode версии 13.4 или новее, поддерживающей язык программирования Swift версии 5.6.1.

Дополнительно необходимо установить библиотеку CalendarKit:

- Открыть проект в Xcode, перейти File → Swift Packages → Add Package Dependency…
- Ввести ссылку на репозиторий Git (https://github.com/richardtop/CalendarKit.git) и нажать Next;
- Выбрать версию "Up to Next Major" и нажать Next;
- Нажать Finish.

Аналогично добавить пакет SwifUI-Charts (ссылка на репозиторий Git: https://github.com/spacenation/swiftui-charts).

Обновление производится непосредственно скачиванием новой версии системы из репозитория github.

Ссылка на репозиторий: https://github.com/VladVelik/Fuctivity .

4.6. Представление архитектуры данных

Для успешного использования приложения пользователю сначала необходимо пройти систему регистрации/авторизации. Все данные пользователей хранятся локально на устройстве с помощью UserDefaults, что не дает возможность пользователю перенести свои задачи с одного устройства на другое, так как в локальном хранилище другого устройства не будет ранее зарегистрированного аккаунта пользователя.

Для хранения всех данных пользователя (включая как данные для авторизации, так и данные о календарных событиях, обозначающих время отдыха) используется класс ChillEvent, включающий следующие поля:

- time: Int
- date: String
- categoryOfEvent: String
- eventDescription: String
- eventStorage: [Event]
- days: String
- username: String
- password: String
- email: String
- restHours: Int
- eventNumber: Int

Также используется функция getEvents(), которая возвращает список событий для календаря.

4.7. Представление архитектуры безопасности

Для входа в свой аккаунт необходимо ввести имя, почту и пароль.

Для безопасного хранения пароля пользователя и передачи его между экранами приложения и моделями представления было принято решение хранить в приложении хешированный пароль пользователя. Для этого использовался алгоритм CryptoKit — это одна из самых популярных коллекций многих стандартных криптографических алгоритмов, написанных на Swift. Криптография сложна, надёжна и проверена.

Работа метода авторизации пользователя реализована следующим образом: принимает электронный адрес и пароль пользователя, и возвращает хешированную строку. Константа salt это уникальная строка, которая делает из обычного пароля редкий SHA256() — это метод с фреймворка CryptoKit, который хеширует введенную строку по алгоритму SHA-2. Использование salt увеличивает сложность подбора пароля. Кроме того, дополнительно комбинируется электронная почта и пароль пользователя с salt для создания хэша.

Таким образом, для аутентификации пользователя, UserViewModel и LogInViewController используют одну и ту же salt. Это позволяет им строить хэши однообразным способом и сравнивать два хэша для проверки личности.

При первом открытии приложение должно получить от пользователя разрешение на отправку уведомлений.

4.8. Представление реализации и разработки

Приложение написано на языке программирования Swift версии 5.6.1, используемые библиотеки также написаны на Swift.

Программно-аппаратная часть приложения, отвечающая за функционирование его внутренней части, написана на языке программирования Swift. Используемые библиотеки также написаны на Swift.

В качестве интегрированной среды разработки (IDE) для работы с платформой iOS используется Xcode, интегрированная среда разработки программного обеспечения для платформ macOS, iOS, watchOS и tvOS, разработанная корпорацией Apple.

Проект с исходным кодом и его версии хранится в репозитории на Github: https://github.com/VladVelik/Fuctivity .

4.9. Представление производительности

В системе используются алгоритмы шифрования, хэширования.

Продвинутые структуры данных в системе не используются.

Производительность системы ключевой роли не играет.

4.10. Атрибуты качества системы

К атрибутам качества относятся:

- Удобство пользования - приоритет отдается готовым элементам интерфейса, при этом они используются по своему прямому назначению, снабжаются поясняющими надписями.
- Скорость - анимации проигрываются нормально, новые предложения отображаются сразу при прокрутке:

    - Скорость загрузки приложения должна быть не более 2 секунд на любом устройстве;
    - Добавление нового элемента в список задач должно занимать не более 1 секунды;
    - Обновление статуса задачи (выполнено) должно занимать не более 0,5 секунды.
- Надежность - система должна быть доступна всегда.
- Безопасность - система не должна подвергать угрозам личные данные пользователя.
- Качество исходного кода и документации - соответствие общепринятым стандартам кодстайла для языка Swift.

4.10.1. Объем данных и производительность системы

Время ответа на запрос не должно превышать 300 мс. Система должна выдерживать нагрузки до 3 тысяч запросов в минуту. Приложение работает бесперебойно и не выдает ошибок.

4.10.2. Гарантии качества работы системы

- Постоянная доступность данных обеспечивается за счёт их локального хранения на устройстве пользователя.
- Удобство пользования - гарантируется за счет соблюдения лучших практик Material Design - проверяется, например, рейтингом приложения в App Store и отзывами там же.
- Скорость - гарантируется использованием эффективных компонентов UI и алгоритмов (см. раздел Представление производительности) - проверяется нагрузочным тестированием.
- Надежность - стабильность работы на устройствах гарантируется качеством ОС iOS и низким требованиям приложения к ресурсам устройства. Проверяется тестированием на различных версиях ОС iOS.
- Качественность исходного кода и документации - гарантируется опытностью разработчиков и проверяется линтерами, а также числом issues и звезд на github.

4.11. Другие представления

Другие представления не предусмотрены.

5. Технические описания отдельных ключевых архитектурных решений

5.1. Техническое описание решения №1

5.1.1 Проблема

Какую архитектуру проекта лучше использовать?

5.1.2 Идея решения

Стоит использовать архитектуру MVVM.

5.1.3 Факторы

Так как приложение является небольшим, представленным несколькими контроллерами, для дальнейшего масштабирования и добавления нового функционала необходимо правильно выбрать архитектуру приложения.

5.1.4 Решение

Использовать три вида разбиения - Model, View, ViewModel, которые помогают отделять логику приложения от его представления - свойства View совпадают со свойствами ViewModel/Model. При этом ViewModel не имеет ссылки на интерфейс представления. Изменение состояния ViewModel автоматически изменяет View, и наоборот.

5.1.5 Мотивировка

В приложении задействовано “связывание данных” - то есть есть взаимосвязь между пользовательским интерфейсом (View) и бизнес-логикой разрабатываемого продукта (Model). Данный механизм связывания данных лежит в основе архитектуры MVVM, поэтому стоит использовать именно его.

5.1.6 Неразрешенные вопросы

Нет.

5.1.7 Альтернативы

MVC - весьма гибкая архитектура и дает возможность изменять интерфейс и логику, разбивая код на модели и отображающие интерфейс контроллеры. Однако MVC не поддерживает явное использование возможностей связывания данных.

5.2. Техническое описание решения №2

5.2.1 Проблема

Какой способ хранения данных выбрать для реализации проекта?

5.2.2 Идея решения

Стоит использовать UserDefaults - один из способов локального хранения и загрузки настроек приложения или других данных, которые нужно хранить между сеансами работы приложения.

5.2.3 Факторы

Приложение требует хранения небольшого объёма простых данных о пользователе и его задачах

5.2.4 Решение

Способом хранения данных в приложении был выбран механизм UserDefaults

5.2.5 Мотивировка

UserDefaults - это простой способ хранения небольших объемов данных, таких как настройки приложения, предпочтения пользователя и т.д. Данные хранятся в виде пар "ключ-значение", такого формата хранения данных будет достаточно для реализации проекта.

5.2.6 Неразрешенные вопросы

Нет

5.2.7 Альтернативы

Альтернативой является CoreData - более мощный механизм хранения данных, который используется для сохранения и управления более сложными структурами данных, такими как базы данных. Разрабатываемый проект не предполагает хранения таких сложных данных.

5.3. Техническое описание решения №3

5.3.1 Проблема

С помощью чего представлять элементы фронтенда?

5.3.2 Идея решения

Использовать UIKit

5.3.3 Факторы

Приложение имеет небольшое количество интерфейсов и сложных элементов.

5.3.4 Решение

Использовать фронтенд библиотеку UIKit для отображения элементов приложения.

5.3.5 Мотивировка

UIKit прост в использовании стандартных элементов, а также позволяет сильно изменять их под конкретное приложение и интерфейс

5.3.6 Неразрешенные вопросы

Нет

5.3.7 Альтернативы

SwiftUI

Плюсы: более новый, поддерживает большее количество элементов, в том числе и сложных
Минусы: поддерживает только iOS 13 и Xcode 11

- Для отображения графиков в статистике пользователя использовалась среда для создания интерфейсов SwiftUI, которая предоставляет различные библиотеки (SwifUI-Charts) для быстрого и удобного построения графиков, отображения постоянных и динамических данных

5.4. Техническое описание решения №4

5.4.1 Проблема

С помощью чего отображать графики в разделе статистики отдыха пользователя?

5.4.2 Идея решения

Использовать специализированную библиотеку SwifUI-Charts.

5.4.3 Факторы

Приложение должно отображать наглядные, понятные и эстетичные графики в разделе статистики отдыха пользователя.

5.4.4 Решение

Использовать фронтенд библиотеку SwifUI-Charts - библиотеку для построения и отображения графиков в SwiftUI.

5.4.5 Мотивировка

Библиотека SwifUI-Charts проста в использовании стандартных элементов, а также позволяет быстро и наглядно строить графики различной сложности, отображать постоянные и динамические данных. И обеспечивает простую и удобную интеграцию библиотеки (различных типов графиков) с приложением (данными об отдыхе пользователя).

5.4.6 Неразрешенные вопросы

Нет

5.4.7 Альтернативы

Фреймворк Core Graphics - сложности в интеграции с данными в приложении, очень мало возможностей кастомизации отображения данных, отсутствие готовых элементов пользовательского интерфейса.

5.5. Техническое описание решения №5

5.5.1 Проблема

Как стоит располагать классы и файлы в проекте?

5.5.2 Идея решения

Стоит группировать модели и их представления вместе

5.5.3 Факторы

Приложение может часто обновляться, изменяя как интерфейсы, так и внутреннюю логику.

5.5.4 Решение

Группировать каждый Model, View и Viewmodel, относящиеся к одному решению вместе.

5.5.5 Мотивировка

При разбиение файлов на группы изменение какого-то одного компонента может быть осуществлено намного быстрее, благодаря смежности как кода, отвечающего за представление интерфейса, так и за внутреннюю логику приложения. Таким образом, разработчику будет проще найти и изменить нужные ему аспекты

5.5.6 Неразрешенные вопросы

Нет

5.5.7 Альтернативы

Хранить файлы представления интерфейса и модели логики раздельно, чтобы у разработчика была возможность быстро просматривать остальные решения проекта. Не очень подходит для проектов, где много классов, так как дальнейшая их группировка может быть не очевидной и сложной для понимания

6. Проверка результативности внесенных в проект изменений

Проблемы, описанные в документе “Предложение по переработке системы” (https://docs.google.com/document/d/12seQudATr-uBTuJMHczzCBw8SJBHeXrQN4rK3xufpRY/edit?usp=sharing) будут считаться решенными, если в результате разработки:

1. Пользователю не придётся при каждом открытии приложения авторизоваться в приложении;
2. Архитектура приложения будет изменена с MVC на MVVM: будут добавлены новые классы - View Model, которые будут связывать бизнес-логику, то есть Model, и View, то есть UI, а контроллеры будут отвечать только за отображение интерфейса и реагирование на намерения пользователя;
3. Все файлы будут объединены в папки, в соответствии с логикой MVVM;
4. Система будет разделена на компоненты в соответствии с понятием разделения ответственности.

В качестве доказательства выполнения критериев было проведено ручное тестирование полученного приложения с внесенными изменениями:

6.1. Шаблон проектирования

Использовались три вида разбиения - Model, View, ViewModel, которые помогают отделять логику приложения от его представления - свойства View совпадают со свойствами ViewModel/Model. При этом ViewModel не имеет ссылки на интерфейс представления. Изменение состояния ViewModel автоматически изменяет View, и наоборот.

На рисунке 4 представлено полученное взаимодействие классов и разделение классов по видам разбиения (Model, View, ViewModel) в соответствии с выполняемой классом функции.

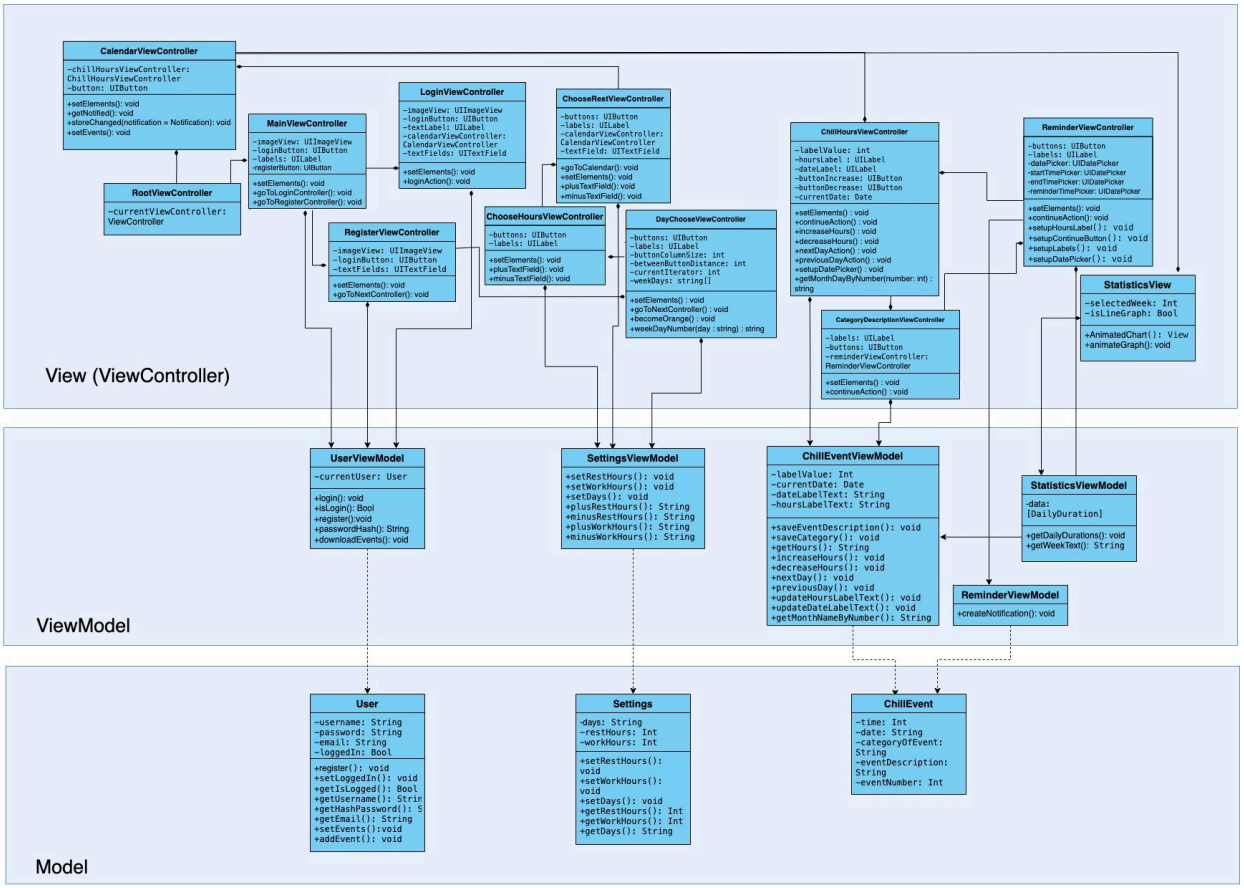


Рис 4. Диаграмма классов

6.2. Разделение файлов в проекте

Все файлы были объединены в папки, в соответствии с логикой MVVM (рис. 5). Таким образом, классы будут объединены согласно логике - Model, ViewModel, View, соответствующие одному решению. Это позволяет облегчить ориентирование, доработку и обновление проекта, так как связанные объекты будут находиться рядом. Помимо этого улучшится и поиск, ведь найти надо будет не конкретный файл класса, а папку с нужной логикой.

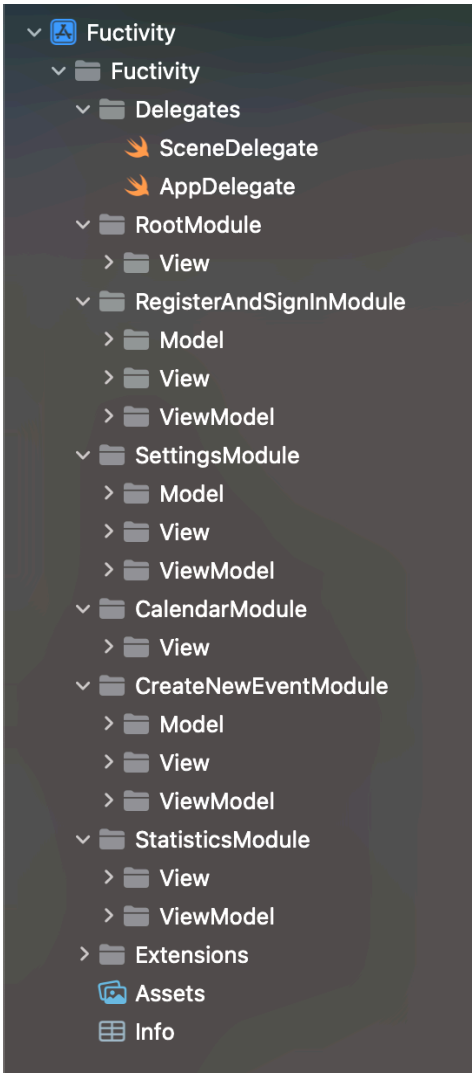
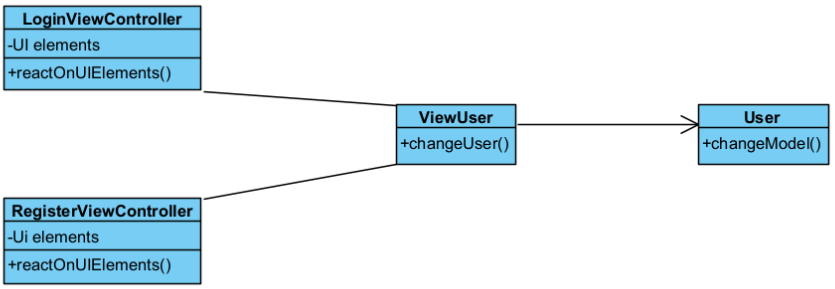


Рис 5. Объединены классов в папки, в соответствии с логикой MVVM

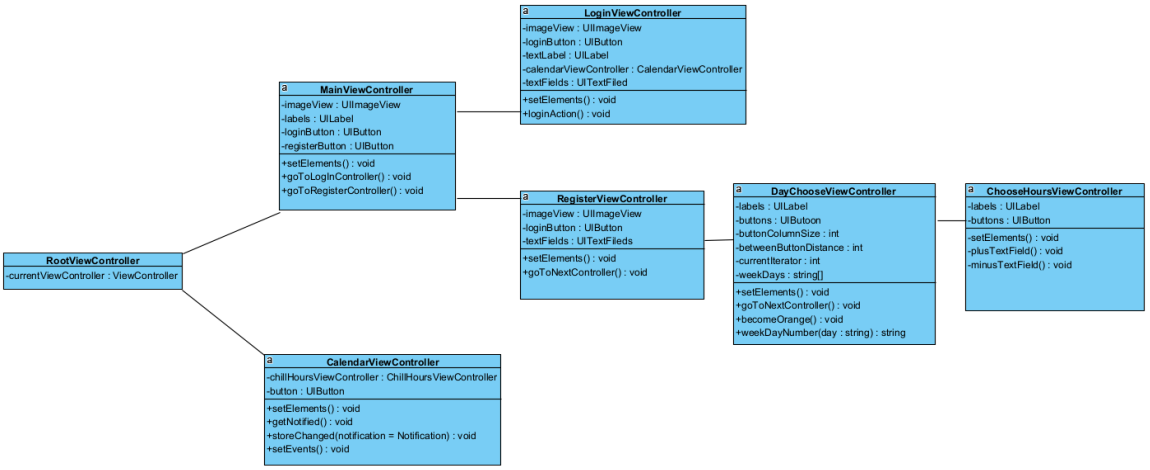
Пример переработки модуля регистрации/авторизации:



Как видно из диаграммы, теперь отображение UI и работа с данным полностью разделены по отдельным блокам. При этом изменение данных User будет влечь за собой изменение интерфейса из-за связывания делегатами

6.3. Управление навигацией

Создан контроллер NavigationController, который запускается первым и во внутренней логике определяет, был ли уже ранее совершен успешный вход в приложение. Если был, то изначально будет открываться контроллер календаря, иначе - контроллер регистрации и входа.



6.3.1 Тестирование - Проверка корректности обновлённой навигации проекта при регистрации

Для тестирования был выбран следующий порядок действий:

1. Вход в приложение;
2. Регистрация пользователя;
3. Заполнение информации о пользователе;
4. Создание новой задачи;
5. Закрытие приложения;
6. Вход в приложение;
7. Авторизация пользователя
8. Проверка наличия в календаре созданной ранее задачи
9. Закрытие приложения
10. Вход в приложение
11. Проверка наличия в календаре созданной ранее задачи

Результат эксперимента: Успешное изменение корневого контроллера при совершенной ранее регистрации и авторизации. https://disk.yandex.ru/i/2OC-xnwHsPcviQ

Можно заметить, что после одной успешной авторизации пользователю не приходится при каждом повторном входе в приложение проходить через экран авторизации и регистрации.

6.4. Разделение ответственности

После смены архитектуры проекта на MVVM система приобрела разделение ответственности по ее компонентам (рис. 4), что значительно упрощает ее доработку и совершенствование в будущем.

- Теперь код модели Model хранит данные и связанную с их обновлениями логику, а также закрепляет структуру приложения:
    - User - хранит данные о пользователе (имя, email, хешированный пароль);
    - Settings - хранит данные о введенных пользователем настройках приложения (количество рабочих дней, количество часов, отведённых на отдых и работу);
    - ChillEvent - хранит данные о запланированном мероприятии / задаче (дата, длительность, описание и категория события);
- Код внешнего вида приложения, View (ViewController), состоит из функций, которые отвечают за интерфейс и способы взаимодействия пользователя с ним. Представления создаются на основе данных, собранных из модели.
    - CalendarViewController - основной экран приложения с календарём и созданными мероприятиями;
    - ChillHoursViewController - экран выбора количество часов отдыха в день;
    - RootViewController - экран определения, был ли уже совершён успешный вход в приложение;
    - MainViewController - первоначальный экран приложения для дальнейшей авторизации/регистрации пользователя;
    - RegisterViewController - экран регистрации пользователя;
    - LoginViewController - экран авторизации пользователя;
    - ChooseHoursViewController - экран выбора длительности рабочего дня;
    - ChooseRestViewController - экран создания нового мероприятия;
    - DayChooseViewController - экран для распределения рабочих и выходных дней;
    - ReminderViewController - экран создания нового уведомления о предстоящем мероприятии;
    - StatisticsView - экран просмотра статистики пользователя;
- Код модели представления, ViewModel, которая служит прослойкой между View и Model. Он получает на вход пользовательский ввод, интерпретирует его и информирует о необходимых изменениях. Отправляет команды для обновления

состояния, такие как сохранение изменений. Определяет, как пользователь добавляет новую задачу. То есть исполняет всю бизнес-логику приложения.

- UserViewModel;
- SettingsViewModel;
- Chill EventViewModel;
- StatisticsViewModel;
- ReminderViewModel.

В результате изменений удалось избавиться от антипаттерна Massive View Controller и разделить всю логику приложения от пользовательского интерфейса

7. Дальнейшие улучшения системы

7.1. Переработка регистрации/авторизации пользователя

На данный момент используется локальная регистрация и авторизация пользователя. Чтобы приложение могло быть перенесено на другое устройство без потери данных, а также для безопасности данных пользователя предлагается использовать Vapor для написания серверной части приложения

Изменения системы:

- Повышается безопасность данных
- Появляется переносимость между устройствами
- Не используются UserDefaults для хранения важных данных

7.2. Использования базы данных

Предлагается использовать PostgreSQL для создания базы данных событий отдыха привязанных к конкретному пользователю.
Такое изменение будет проще поддерживать, так как PostgreSQL один из основных вариантов использования с Vapor. Уменьшится объем хранимой памяти для приложения на устройстве, так как все данные будут запрашиваться с серверной части

7.3. Мультиплатформенность

Для расширения распространения приложения предлагается сделать его мультиплатформенным и сделать его доступным на macOS и iPad.
Это изменение позволит пользователю повысить удобство пользования календарем и сделает его доступным на рабочем месте. Помимо этого подобное изменение повысит эффективность напоминаний о необходимости отдыха, так как уведомления будут появляться и на рабочем месте пользователя

8. Приложения

8.1. Словарь терминов

Баг - означает ошибку в программе или в системе, из-за которой программа выдает неожиданное поведение

Кодстайл - набор правил и рекомендаций, определяющих стиль написания кода в конкретном языке программирования или в рамках определенного проекта.

Линтер - программа, которая проверяет код на соответствие стандартам в соответствии с определенным набором правил

ОС - операционная система

ПК - персональный компьютер

iOS - мобильная операционная система для смартфонов, электронных планшетов, носимых проигрывателей, разрабатываемая и выпускаемая американской компанией Apple

MVC, MVP, MVVM - архитектурные паттерны

Swift - открытый мультипарадигмальный компилируемый язык программирования общего назначения, разработанный и поддерживаемый компанией Apple

UI - User Interface

UIKit - модульный front-end фреймворк

Xcode - интегрированная среда разработки (IDE) программного обеспечения для платформ macOS, iOS, watchOS и tvOS, разработанная корпорацией Apple

8.2. Ссылки на используемые документы

https://github.com/VladVelik/Fuctivity - ссылка на проект