МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет

імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних

інформаційних систем, д-р техн. наук, проф.

_____ Ю.П. Кондратенко

«____»_____ 2022р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА ТАЙМ-МЕНЕДЖМЕНТУ ПРАЦІВНИКА

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 402.21810222

Виконав студент 4-го курса, групи 402

_____ **С. О. Резніченко**

«__» червня 2022 р.

Керівник: канд. техн. наук,

_____ **І. В. Кулаковська**

«__» червня 2022 р.

Миколаїв – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет

імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**

Спеціальність **122 «Комп'ютерні науки»**

(шифр і назва)

Галузь знань **12 «Інформаційні технології»**

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.

_____ Ю.П. Кондратенко

«____» _____ 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

Видано студенту групи 402 факультету комп'ютерних наук

Резніченко Сергію Олександровичу

1. Тема кваліфікаційної роботи

Інтелектуальна система тайм-менеджменту працівника

Керівник роботи Кулаковська Інесса Василіївна, кандидат фізично-математичних наук, доцент

Затв. наказом Ректора ЧНУ ім. Петра Могили від «07» 12 2021р. № 318

2. Строк представлення кваліфікаційної роботи студентом «__» _____ 2022р.

3. Вхідні(початкові) дані роботи: інтелектуальна система тайм-менеджменту працівника, елементи інтелектуальної системи, середовище для розробки програмного забезпечення

Очікуваний результат роботи: мобільний застосунок для інтелектуальної системи тайм-менеджменту працівника

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- огляд елементів розробки інтелектуальної системи
- створення та аналіз структури інтелектуальної системи
- вибір програмного забезпечення для інтелектуальної системи
- програмна реалізація застосунка для інтелектуальної системи

5. Перелік графічних матеріалів: сторінок – 89, таблиць – 4, рисунків – 20, посилань – 22, додатків – 1, презентація

6. Завдання до спеціальної частини

Охорона праці на робочому місці у кабінеті у ЧНУ ім. Петра Могили

7. Консультанти:

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охороною праці	Макарова О.В., старший викладач	

Керівник роботи кандидат фізично-математичних наук, доцент Кулаковська І.В.

_____
(підпис)

Завдання прийнято до виконання Резніченко Сергій Олександрович
(прізвище, ім’я, по батькові студента)

_____
(підпис)

Дата видачі завдання «4» жовтня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

на виконання кваліфікаційної роботи

Тема: «Інтелектуальна система тайм-менеджменту працівника»

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників БКР	04.10.2022	04.10.2022	виконано
2	Отримання завдання на виконання БКР	05.10.2022	10.10.2022	виконано
3	Складання календарного плану роботи на весь період виконання БКР	11.10.2022	11.10.2022	виконано
4	Отримання завдання на переддипломну практику	06.04.2022	06.04.2022	виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до БКР	20.04.2022	09.05.2022	виконано
6	Розробка звіту з переддипломної практики	10.05.2022	12.05.2022	виконано
7	Виконання БКР: аналіз сфери систем для тайм-менеджменту, вибір підходів для створення застосунок, створення застосунку, тестування	13.05.2022	29.05.2022	виконано
8	Попередній захист БКР на засіданні комісії кафедри	31.05.2022	31.05.2022	виконано
9	Доробка та остаточно оформлення БКР	01.06.2022	19.06.2022	виконано
10	Подання БКР рецензенту	20.06.2022	20.06.2022	виконано
11	Подання БКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	24.06.2022	24.06.2022	
12	Захист БКР перед екзаменаційною комісією (ЕК)	27.06.2022	27.06.2022	

Розробив студент ___Резніченко С. О._____ _____
(прізвище, ім’я, по батькові студента) (підпис)

Керівник роботи _канд. техн. наук, Кулаковська І. В.__ _____
(*наук. ступінь, вчене звання, прізвище та ініціали*) (підпис)

«____» _____ 202__ р.

АНОТАЦІЯ

бакалаврської кваліфікаційної роботи студента групи 402 ЧНУ ім. Петра Могили

Резніченка Сергія Олександровича

Тема: «Інтелектуальна система тайм-менеджменту працівника»

Об’єкт дослідження: процес створення застосунку під операційну систему iOS використовуючи мову програмування Swift

Предмет дослідження: засоби для створення програмного застосунку тайм-менеджементу працівника

Мета роботи: створення застосунку для тайм менеджменту працівника на операційній системі iOS.

У першому розділі аналізується предметна сфера тайм менеджменту та розглядаються аналоги. У другому розділі – опис можливих технологій та методів при розробці застосунку. У третьому розділі представлено моделювання застосунку, які необхідні для подальшої реалізації. У останьому розділі було запрограмовано застосунок використовуючі матеріали, використані у попередніх розділах.

Сторінок – 89, таблиць – 4, рисунків – 20, посилань – 22, додатків – 1

Ключові слова: шаблони проєктування, тайм-менеджмент, iOS, UIKit, база даних

ABSTRACT

for bachelor's qualification work of a student of 402 group at Petro Mohyla Black Sea National University

Reznichenko Serhii Oleksandrovich

Topic: «Intelligent employee time management system»

The object of the research is the process of creating an application for the iOS operating system using the Swift programming language

The subject of the research is tools for creating a software application for employee time management

The purpose of the research is to create an application for employee time management for the iOS operating system.

In the first section, the area of time management is analozied and analogues are considered. In the second, the possible technologies and methods for application development are describe. In the third section, the application's structure is modeled, which is necessary for further development. In the last section, the application was created using the models created in the previous sections.

Pages – 89, tables – 4, pictures – 20, references – 22, appendices – 1

Keywords: design pattern, time management, iOS, UIKit, database

ЗМІСТ

Пояснювальна записка

до кваліфікаційної роботи

на тему:

«ІНТЕЛЕКТУАЛЬНА СИСТЕМА ТАЙМ-МЕНЕДЖМЕНТУ ПРАЦІВНИКА»

Спеціальність 122 «Комп’ютерні науки»

122 – БКР – 402.21810222

Виконав студент 4-го курса, групи 402

_____ С. О. Резніченко

«__» червня 2022 р.

Керівник: канд. техн. наук,

_____ І. В. Кулаковська

«__» червня 2022 р.

Миколаїв - 2022

ВСТУП

З розвитком технологій та прогресу людства все більше задач постає перед працівниками та бізнесом. Зазвичай працівники у компаніях мають багато завдань, з різними часовими проміжками, різними пріоритетами, які потребують швидкого ознайомлення в різні моменти часу.

Саме для таких проблем і розробляються застосунки для тайм менеджменту, які дозволяють розставляти пріоритети, обирати дедлайни завдань. Більш модернізовані системи, можуть позволити сумісну роботу з іншими працівникам, та мати сумісні завдання з іншими працівниками.

Для цією кваліфікаційної роботи було розроблено такий застосунок, для менеджменту завдань працівника, на операційній системі iOS через популярність операційної системи.

Мета кваліфікаційної роботи – створення застосунку для тайм менеджменту працівника на операційній системі iOS

Об’єкт дослідження – процес створення застосунку під операційну систему iOS використовуючи мову програмування Swift

Предмет дослідження – засоби для створення програмного застосунку тайм-менеджементу працівника

Для досягнення мети необхідно розв’язати наступні задачі:

- аналіз предметної сфери створення програмного застосунку для тайм менеджементу;
- огляд методів створення програмного застосунку;
- вибір найкращих методів для створення застосунку;
- програмна реалізація застосунку обраних методів для створення застосунку.

Для розв’язання задач має бути використана мова програмування Swift, та фреймворк UIKit для опису інтерфейсу.

Актуальність цієї теми полягає у необхідності рішень для тайм менеджменту та обмеженістю наявних рішень для тайм менеджементу, які не дозволяють працівникам продуктивно працювати

Метою спеціального розділу про охорону праці для опису вимог до робочого місця для збереження здоров’я. Ці умові установлені у законодавстві України, які зобов’язують роботодавця надати працівникам необхідні умови праці.

1. АНАЛІЗ СФЕРИ ЗАСТОСУНКІВ ДЛЯ ТАЙМ-МЕНЕДЖМЕНТУ

1.1 Вимоги до мобільних застосунків, які виконують роль тайм-менеджера

До вимог до застосунку, які можна задати для тайм-менеджменту для працівників, можна віднести такі функції:

Застосунок повинен відображати усі завдання, які є запланованими у робітника:

- можливість створення власного списку завдань;
- можливість видалення завдань;
- можливість створення нагадувань для окремих завдань.

Застосунок повинен бути зручним в використані, та не мати явних помилок при використані. Застосунок повинен бути гнучким, та не мати суворих обмежень в тому, як задавати завданням працівникам. Це слід враховувати через те, що кожна людина планує свій час у свій, власний, унікальний спосіб. Тому, якщо обмежити працівники у виборі завдання задач, то це буде тільки заважати працівникам використовувати свій час раціонально.

Проте, слід забезпечити працівника деяким набором функцій, таких як відображення завдань у зручний спосіб, у вигляді календаря завдань чи розподілення завдань за окремими категоріями та групами. Це дозволить працівникам економити час, надаючи зручні функції для працівників.

Більшість мобільних застосунків можуть мати схожі функції, але головна особливість кожного з застосунків, це представлення цих функцій користувачеві в особливий, більш зручний формат. Ця ціль також переслідується у цій роботі.

1.2 Огляд та аналіз наявних аналогів

Серед наявних аналогів серед застосунків для тайм-менеджменту можна виділити такі найпопулярніші застосунки:

- Todoist;
- TickTick;
- MinimaList.

Застосунок Todoist є безкоштовним додатком з можливістю придбати повноцінну версію з повним набором функцій. Вид екрану створення завдання у застосунку Todoist зображен на рисунку 1.1

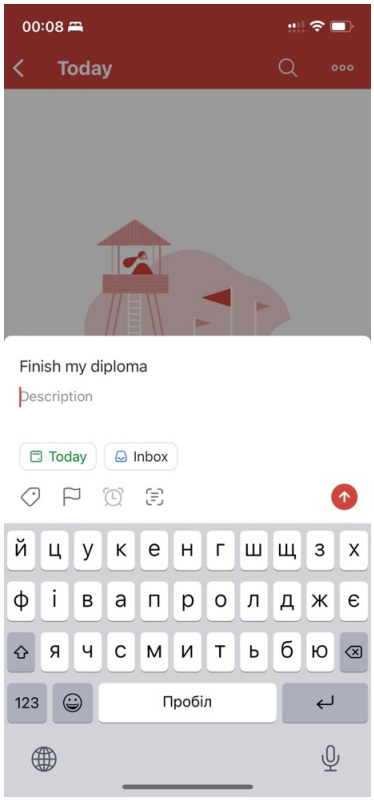


Рисунок 1.1 – Екран створення завдання застосунку Todoist

Слід зазначити, що усі представлені застосунки використовують цей самий метод заробітку. Ця модель заробітку називається Freemium [7].

Застосунок Todoist має синхронізацію між різними пристроями та дозволяє зайти у застосунок з різних аккаунтів. Застосунок дозволяє додати завдання та прикріпити дату, пріоритет, теги, категорію.

Вид екрану при створених завданнях зображений на рисунку 1.2

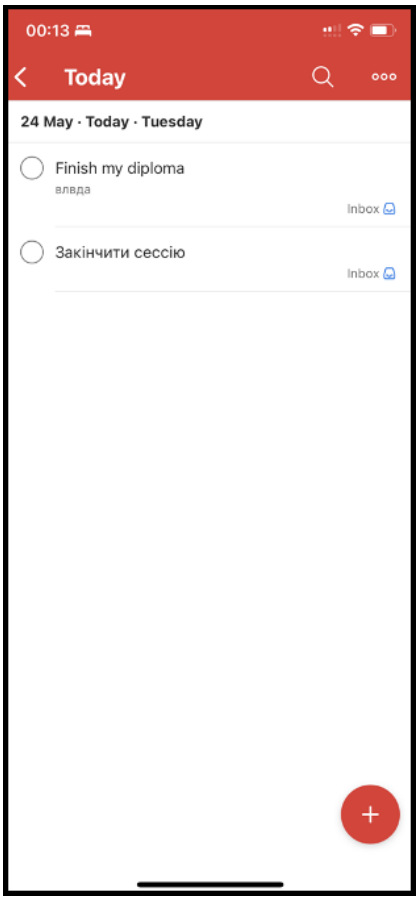


Рисунок 1.2 – Екран зі створеними завданнями

Інший популярний застосунок TickTick є також одним з найпопулярніших застосунків для тайм менеджменту. Головною відмінністю застосунку TickTick є наявність великої кількості функцій.

Кількість наявних функцій можна побачити на екрані налаштувань застосунку, який зображений на рисунку 1.3

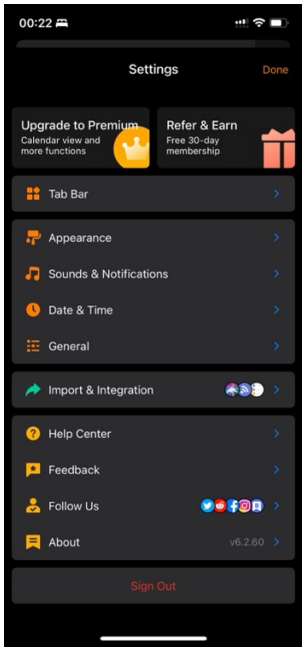


Рисунок 1.3 – Налаштування застосунку TickTick

Відміність застосунку MinimaList від інших в мінімалістичному дізайні, що і відображує назва данного застосунку. Основні функції та спосіб монетизації має той же, що й інші два застосунки

Вид застосунку зображений на рисунку 1.4

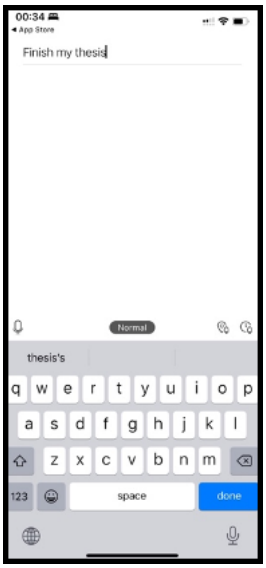


Рисунок 1.4 – Вид застосунку MinimaList

1.3 Постановка задачі

Основна мета виконання кваліфікаційної роботи це вивчення методів та отримання навичок при створенні застосунків використовуючі мову програмування Swift розроблюючи застосунок для операційної системи iOS, яка розроблюється Apple. Також внаслідок виконання роботи необхідно отримати навички при роботи з базами даними, створенням інтерфейсів та використання необхідних шаблонів проектування.

Досить важливим є використання правильних підходів для створення застосунку, тому вибір бази даних, методу створення інтерфейсів та необхідні шаблони потрібні бути ретельно підібрані та необхідно провести порівняння між різними методами розробки.

Окрім цього, внаслідок виконання роботи будуть отримані такі навички як вміння працювати з інтегрованим середовищем розробки, та у випадку цієї роботи це буде програма Xcode.

Предметом дослідження є технологія створення iOS застосунку.

Для того, щоб написати застосунок необхідно виконати наступні завдання:

- розгляд технології для створення програмного забезпечення;
- вибрати необхідні технології: мову програмування, фреймворк та базу даних для виконання роботи;
- проектування моделі бази даних;
- програмна реалізація застосунку.

Висновки до розділу 1

Було розглянуто основні вимоги до застосунків для тайм-менеджменту працівника. Також, були розглянуті приклади готових робіт, та виявлено особливості кожного з рішень. Кожен з застосунків було завантажено, та перевірено працездатність застосунків.

Внаслідок розгляду готових рішень та основним вимог для застосунків, було сформовано список шагів, для створення власного застосунку для тайм менеджменту працівника використовуючи знання про недоліки та переваги кожного з підходів для готових застосунків. Цей план і буде використано в подальшому для створення власного застосунку та буде продемонстроване у кваліфікаційній роботі.

2 ЗАГАЛЬНИЙ ОПИС ТЕХНОЛОГІЙ РОЗРОБКИ СИСТЕМИ

2.1 Вибір бази даних

Проблеми зберігання та обробки інформації завжди виникали у зв'язку з потребами промисловості, бізнесу та наукових досліджень. Бази даних були створені з метою забезпечення організованого доступу до даних. Вони почали з навігаційних баз даних зі зв'язаними списками, потім розробили реляційну модель даних, а разом з нею реляційні бази даних, потім об'єктно-орієнтовані бази даних і, нарешті, метод NoSQL.

Реляційні бази даних і бази даних NoSQL на даний момент є найпопулярнішими типами баз даних. Незважаючи на те, що бази даних NoSQL новіші за інші, вони зросли в популярності завдяки їх здатності швидко обробляти неструктуровані дані. Велика різноманітність баз даних ускладнює для розробників вибір найкращого рішення. Важливо порівняти ці два базових типи даних; це дозволить розкрити їх сильні та недоліки, а також їх актуальність у вирішенні конкретної проблеми. Бази даних мають вирішальне значення в застосунку на смартфоні, і неправильний вибір може бути дорогим у довгостроковій перспективі, оскільки важко змінити базу даних, коли система запущена, особливо на різних платформах.

Існують численні дослідження, які порівнюють бази даних NoSQL з реляційними базами даних. Способи підтримки паралельності керування, зберігання даних, реплікації та механізмів транзакцій у різних СУБД і NoSQL обговорюються в [2].

Автори роботи [5] представляють різні категорії систем NoSQL і порівнюють реалізацію та продуктивність кількох баз даних NoSQL з SQL рішеням. У роботі [4] порівнюються бази даних NoSQL з СУБД і

виділяються загальні особливості та недоліки NoSQL. Результати тестів продуктивності різних баз даних NoSQL відображені в [1].

Реляційні бази даних

Бази даних описуються як сукупність даних, які зберігаються та доступні програмним забезпеченням [3, с. 11]. Хоча ми часто використовуємо термін «база даних» для позначення всієї системи, сам термін відноситься лише до колекцій і даних. Ця система керує даними, транзакціями та іншим використанням.

Система управління даними — це назва бази даних, тобто Система управління базами даних(СУБД).

Усі дані представлені у вигляді n-арного відношення, яке потім представляється як підмножина n-арного декартового добутку n множин у реляційних базах даних, які засновані на реляційній моделі та теорії множин. Відношення (таблиця) складається з набору кортежів (записів) з тими ж властивостями, що й стовпці. Ця модель даних надзвичайно точна і добре організована.

Завдяки повній підтримці чотирьох властивостей ACID реляційна база даних забезпечує високу надійність транзакцій. Властивості реляційних баз даних:

- атомарність: якщо будь-який компонент транзакції виходить з ладу, повна транзакція також;
- узгодженість: якщо база даних була в узгодженому стані до транзакції, вона буде в узгодженому стані після транзакції;
- ізоляція: кілька транзакцій, що виконуються одночасно, не заважають одна одній. Іншими словами, паралельні транзакції повинні бути серіалізованими;
- довговічність: незважаючи на будь-які збої, зміни, внесені зафіксованою транзакцією, будуть зареєстровані.

Нереляційні бази даних

Останніми роками було створено більше нових систем, які забезпечують гарне горизонтальне масштабування для основних операцій читання/запису над базами даних та розподіленим навантаженням.

Традиційні бази даних, з іншого боку, мають менше параметрів масштабування.

Фраза сховища даних «NoSQL» використовується для опису багатьох інноваційних технологій.

Термін «Not Only SQL» або «Not Relational», що означає «Not Only SQL» або «Not Relational», не визначено повністю. Як правило, такі системи відповідають наступним вимогам:

1. базовий протокол виклику для операцій (порівняно з SQL);

2. транзакції ACID мають більш слабку модель паралельності;

3. ефективне зберігання даних з використанням розподілених індексів і оперативної пам'яті;

4. немає встановленої схеми даних;

5. більш легка інтеграція з серверами для клієнтів.

Властивості транзакцій ACID часто не відповідають системам NoSQL: остаточна угода не дозволяється. На відміну від ACID, був наданий підхід «BAS» (Basically Available, Soft State, Eventually Consistent, Account). Ідея полягає в тому, що, знявши обмеження ACID, можна значно покращити продуктивність і масштабованість. Ступінь, до якої більшість систем відмовляється від ACID, різниться.

Прихильники NoSQL часто посилаються на теорему CAP, яка стверджує, що система може задовольняти лише двом із трьох властивостей:

- доступність – дані завжди доступні для користувача;
- узгодженість – дані завжди однакові для всіх реплік;

- допуск до розділів – система бази даних продовжує працювати правильно, навіть якщо мережа або вузли виходять з ладу.

У системах NoSQL послідовність часто ігнорується, але зробити припущення складніше.

Моделі даних в системах NoSQL часто поділяють на такі групи [9]:

- сховища ключ-значення: зберігається значення плюс ідентифікатор, щоб ви можна було шукати їх за допомогою ключа;
- сховища документів: система зберігає дані у вигляді індексованих документів;
- записи можна розподіляти по вертикалі та горизонталі між вузлами в розширених сховищах записів.

Беручи до уваги переваги нереляційних баз даних, було обрано використовувати саме такий тип бази. У iOS розробці зазвичай використовується база даних Realm для опису нереляційних даних.

2.2 Шаблони проектування

Шаблони проектування – це рішення розробки програмного забезпечення для вирішення поширених проблем. Шаблони проектування — це не готові рішення, які можна перетворити в код, а скоріше широкий опис вирішення проблеми, який можна застосувати до різноманітних сценаріїв.

Матеріали, наведені в списку літератури, активно використовувалися при побудові запропонованого тут каталогу шаблонів дизайну. В якості відправної точки використовується класифікація шаблонів GoF за видами та типами. У той же час вміст, наведений у каталозі GoF, прямо не повторюється. Також включені шаблони з інших джерел.

Існують різні типи шаблонів дизайну, кожен зі своїм набором труднощів для вирішення:

- створення шаблонів, які призведуть до появи нових елементів у системі;
- структурні шаблони, які вирішують проблему компонування системи на основі класів і об'єктів;
- шаблони поведінки, які використовуються для розподілу обов'язків між системними об'єктами.

Породжуючі патерни

Мабуть, найпоширенішою задачею, з якою стикаються інженери-програмисти, є створення нових об’єктів. Шаблони творчого дизайну створюються для створення елементів, що дозволяє системі бути незалежною як від процесу створення, так і від видів створених об’єктів. Перш ніж ми розглянемо характеристики кожного з генеруючих шаблонів, давайте розглянемо деякі поширені проблеми, з якими стикаються розробники під час створення нових видів об’єктів у системі.

Давайте створимо «Пунічні війни», потенційну стратегічну гру, яка зображує величезний військовий конфлікт між Римською Республікою та Карфагеном (264-146 рр. до н.е.). У грі можна знайти воїнів трьох категорій: піхоту, кінноту та лучників. Кожен з цих типів має різні особливості, такі як зовнішній вигляд, бойова сила, швидкість пересування та рівень захисту. Незважаючи на різноманітність, всі бойові одиниці мають ряд характеристик. Наприклад, всі вони можуть подорожувати по ігровому полю в різних напрямках, однак вершники є найшвидшими. Крім того, кожен бойовий підрозділ має власну шкалу здоров’я, і якщо вона досягає нуля, воїн гине. У той же час лучника значно легше вбити, ніж інші типи воїнів.

Якщо гра виявиться успішною, ми продовжимо її розвивати. Наприклад, розділивши солдатів на легку та важку екіпіровану піхоту, ми

можемо створити нові типи бійців, наприклад бойових слонів, або покращити існуючі. Тепер ми повинні прагнути зробити гру максимально незалежною від певних типів персонажів, щоб вносити ці модифікації без зміни існуючого коду. Здається, що для цього достатньо наступної ієрархії класів.

```
class Warrior {
    func info() {
        print(“Warrior”)
    }
}

class Archer: Warrior {
    override func info() {
        print(“Archer”)
    }
}

class Horseman: Warrior {
    override func info() {
        print(“Horseman”)
    }
}

class Infantryman: Warrior {
    override func info() {
        print(“Infantryman”)
    }
}

class CrossbowMan: Warrior {
    override func info() {
        print(“Crossbowman”)
    }
}
```

Рисунок 2.1 – Код ієрархіє класів

Warrior — це поліморфний базовий клас, який визначає загальний інтерфейс, тоді як Infantryman, Archer, Horseman та Crossbowman — це похідні класи, які реалізують особливості кожного типу воїнів.

Проблема виникає з того факту, що, хоча системний код працює з готовими об’єктами через загальні інтерфейси, нові персонажі повинні

створюватися, безпосередньо вказуючи їх точні види під час гри. Буде важко ввести нові типи символів або замінити наявні, якщо код для їх створення буде розподілений по програмі.

У таких ситуаціях на допомогу приходить фабрика об’єктів, яка централізує генерацію об’єктів. Фабрика об’єктів працює подібно до віртуального конструктора, оскільки дозволяє нам створювати об’єкти потрібних класів без явного оголошення їх типів. У найпростішому сценарії використовуються ідентифікатори типів. Найпростіша версія фабрики об’єктів, функція фабрики, демонструється в наступному прикладі.

```
enum WarriorId {
    case infantryman
    case archer
    case horseman
    case crossbowman
}

func createWarrior(_ id: WarriorId) -> Warrior {
    switch id {
    case .infantry:
        return Infantryman()
    case .archer:
        return Archer()
    case .horseman:
        return Horseman()
    case .crossbowman:
        return CrossbowMan()
    }
}
```

Рисунок 2.2 – Код фабрики об’єктів

Код для створення об’єктів різних типів ігрових персонажів тепер об’єднаний в одному місці, зокрема, у фабричному методі createWarrior(), який приховує тонкощі. Ця функція приймає тип об’єкта, який потрібно створити, як вхідні дані, конструює його та повертає вказівник на базовий

клас. Незважаючи на очевидні переваги, цей заводський варіант має недоліки. Наприклад, щоб додати новий тип бойової одиниці, ви повинні спочатку створити новий ідентифікатор типу, а потім змінити код заводського методу createWarrior().

Структурні патерни

Компонування системи, заснованої на класах і об'єктах, вирішується структурними шаблонами. Це можна зробити за допомогою таких механізмів:

Коли базовий клас визначає інтерфейс, а підкласи описують реалізацію, це називається наслідування. Структури на основі наслідування статичні.

Коли структури будуються шляхом об’єднання об’єктів з різних класів, це називається композицією. Ви можете використовувати композицію для створення структур, які можна змінити під час виконання.

Розглянемо на мить характеристики структурних шаблонів (шаблони).

Шаблон адаптера — це програмна оболонка, яка використовується для перекладу інтерфейсів існуючих класів у формат, який можна використовувати в новому програмному проекті.

Шаблон Bridge ізолює абстракцію від реалізації, дозволяючи змінювати обидва окремо.

Шаблон Composite створює деревоподібні структури з подібних компонентів. Лікування як простих, так і складних предметів є послідовним.

Шаблон Decorator використовується, щоб зробити речі більш функціональними. Шаблон Decorator динамічно додає нові обов’язки до об’єкта як гнучку альтернативу висновку класу.

Щоб зробити набір інтерфейсів підсистеми простішим у використанні, шаблон Facade забезпечує високорівневий об’єднуючий інтерфейс.

Щоб правильно підтримувати багато предметів, дизайн Flyweight розділяє їх.

Щоб керувати доступом до іншого об’єкта, шаблон проксі замінює його.

Шаблони поведінки

Питання про взаємодію між об’єктами та розподіл відповідальності між ними вирішуються за допомогою моделей поведінки. Для цього можна використовувати механізми, засновані на наслідування та композиції.

Розглянемо на мить характеристики моделей поведінки (шаблони).

Декілька об’єктів-одержувачів можуть обробляти запит за допомогою шаблону ланцюга відповідальності. Запит передається по ланцюжку, поки його не обробить якийсь об'єкт, а одержувачі не будуть з'єднані в ланцюжок. Сувора залежність між відправником запиту та його одержувачами також уникає за допомогою шаблону ланцюга відповідальності.

Запит на виконання дії перетворюється на окремий командний об’єкт за допомогою шаблону Command. Це забезпечує гнучкість системи, дозволяючи динамічну заміну команд, використання складних складених команд і можливість переривати дії.

Шаблон Iterator дозволяє переміщатися по елементах складених об'єктів (колекції) без розкриття їх внутрішнього представлення.

Шаблон Interpreter призначений для вирішення повторюваних дій, які можна вказати мовою. Шаблон Interpreter досягає цього, описуючи проблему, яку потрібно вирішити, словами цієї мови, а потім інтерпретуючи їх.

Шаблон Медіатор інкапсулює групу взаємодій об'єктів в один об'єкт-посередник. Зменшує ступінь зв’язку між взаємодіючими елементами, оскільки їм не потрібно стежити за посиланнями один одного.

Шаблон Memento витягує та зберігає внутрішній стан об’єкта поза об’єктом, щоб його можна було відновити пізніше. Конструкція Observer встановлює зв’язок «один до багатьох» між об’єктами, гарантуючи, що коли стан одного об’єкта змінюється, всі залежні об’єкти автоматично сповіщаються та оновлюються.

Шаблон стану дозволяє змінювати поведінку об'єкта залежно від його внутрішнього стану. Схоже, що клас об’єкта змінився. Шаблон стану — це об'єктно-орієнтована реалізація кінцевого автомата.

Якщо для налаштування поведінки системи використовується сімейство алгоритмів, шаблон стратегії використовується для переміщення сімейства алгоритмів до окремої ієрархії класів, що дозволяє під час виконання програми замінювати один алгоритм іншим. Крім того, таку систему менш складно розширювати та підтримувати.

Шаблон Template Method визначає основу алгоритму і дозволяє підкласам змінювати деякі його фази, не впливаючи на загальну структуру методу.

Шаблон Visitor визначає операцію, яка повинна виконуватися над кожним елементом структури, не впливаючи на їхні класи. Можливість додавати нові операції до існуючих структур об’єкта без зміни структур є практичним ефектом такого поділу.

Шаблони проектування наведені у таблиці 2.1

Таблиця 2.1 – Види шаблонів проектування

Вид шаблонів	**Породжувальні**	**Структурні**	**Поведінкові**

Закінчення таблиці 2.2

Приклад шаблонів	Фабричний метод Абстрактна фабрика Будівельник Прототип Одинак	Адаптер Міст Компонувальник Декоратор Фасад Легковаговик Замісник	Ланцюжок обов’язків Команда Ітератор Посередник Знімок Спостерігач Стан Стратегія Шаблонний метод Відвідувач

В процесі роботи над проектом буде використано декілька видів шаблонів.

2.3. Мова програмування при написанні застосунку

Для написання iOS застосунків зазвичай використовуються дві мови програмування: Swift або Objective-C. Кожна мова має свої переваги та недоліки, які продемонстровані у таблиці 2.2

Таблиця 2.2 – Переваги мов програмування Objective-C та Swift

Характеристика	Objective-C	Swift

Продовження таблиці 2.2

Управління пам’яттю	Недоліки: Використовує автоматичний підрахунок посилань, які використовуються тільки у Cocoa програмного інтерфейсу програми	Переваги: Використовує автоматичний підрахунок посилань у всіх програмних інтерфейсах програми
Підтримка динамічних бібліотек	Недоліки: Немає підтримки динамічних бібліотек	Переваги: Динамічні бібліотеки підтримані
Перспективи на використання в майбутньому	Переваги: Продовжує підтримуватися Apple, але не так активно, як Swift	Переваги: Активно підтримується компанією Apple
Швидкодія	Переваги: Функції у мові програмуванні C працюють швидше ніж у Swift Недоліки: Виконується повільніше через наявність компіляції в момент виконання програми	Переваги: Мова програмування працює набагато швидше Objective-C

Закінчення таблиці 2.2

Безпека	Недоліки: Використовує нульові показчики, які призводять до крашів програми	Переваги: Використовує технології які дозволяють програмістам швидко знаходити помилки та виправляти їх
Синтаксис	Недоліки: Використовує дуже багато службових символів, які призводять то нечитабельності коду	Переваги: Мова програмування дуже легко читається навіть людьми, які не мають досвід з мовою програмування Swift
Підтримка спільнотою	Переваги: Мова програмування розвивалась досить довгий період часу	Переваги: Мова програмування отримала велику популярність, яка переважає популярність Objective-C зараз
Підтримка	Недоліки: Розробникам необхідно створювати багато файлів, а саме файли імплементації та інтерфейсу	Переваги: Необхіден усього один файл, в якому використовуються модифікатори доступу

За результатами з опублікованої наукової статті «SPEED PERFORMANCE BETWEEN SWIFT AND OBJECTIVE-C» [6] швидкість роботи мови програмування Swift в деяких операціях переважала швидкості

Objective-C. Наукова стаття була опублікована у 2016, за який час змінилось багато версій мови програмування Swift та швидкість збільшилась як і зазнається автором статті, як можливий розвиток дій.

Хоча, були операції, в яких ці дві мови виконали операції приблизно з однаковою швидкістю. Метод кожної мови використовувався для сортування списку з 1 000 000 елементів у порядку зростання. Об’єкти були відсортовані по порядку за допомогою числової змінної екземпляра, яка була створена випадковим чином і була в діапазоні від 1000 до 1000. Замість об’єкта використовувалася структура, оскільки C не підтримує об’єкти. Було побудовано дані після запуску кожної програми 25 разів. Swift і Objective-C працювали приблизно з тою самою швидкістю як зазначається автором статті.

Беручи до уваги швидкість мови програмування Swift, її новина, та відносна простота в використані

2.4 Використання UIKit та SwiftUI для мобільної розробки

SwiftUI та UIKit мають свої переваги та недоліки, та є досить важливим рішенням при початку розробки застосунку. Два підходи мають свої переваги та недоліки.

До особливостей SwiftUI можна віднести:

- простий синтаксис, який легко вивчити;
- важно опанувати SwiftUI. Важко використовувати SwiftUI для створення більш складних речей, і вивчення деяких нових речей, таких як Bindings, ObservableObjects, змінні стану тощо, метод роботи більш складний та незвичний;

- є незрілим і незавершеним. Багато простих операцій, які можна легко зробити у UIKit вимагають складних обхідних шляхів і модифікацій у SwiftUI;
- з ним працювати швидко — його можна використовувати для швидкого створення простих сторінок;
- через новизну фреймворку існує не так багато інтернет-ресурсів, щоб допомогти у можливих труднощах, з якими можна зіткнутися. Ця проблема погіршується тим, що SwiftUI постійно зростає, через що корисні веб-матеріали швидко застарівають;
- у багатьох сторонніх бібліотеках, таких як Firebase, бракує документації, щоб налаштувати їхню інфраструктуру у проекті SwiftUI;
- SwiftUI має можливість з'єднуватися з UIKit, що дозволяє отримати доступ до можливостей UIKit у поданні SwiftUI через об'єкт UIViewRepresentable;
- деякі режими перегляду, наприклад, навігаційні, досить просто налаштувати;
- повідомлення про помилки є великою проблемою, і насправді вони не допомагають у налагодженні. Швидше за все, це пов’язано з молодістю SwiftUI;
- деякі операції, наприклад центрування мітки або кнопки, відносно прості та займають набагато менше часу, ніж у UIKit.

Відносно UIKit, то він має такі особливості:

- UIKit має набагато більше можливостей і функцій;
- UIKI є зрілим — можна легко знайти безліч корисної інформації, яка допоможе майже з будь-яким завданням;
- написання шаблонного коду або аналогічного повторюваного коду для простих завдань може бути незручним і часом неприємним;
- контролери хостингу можуть підключати SwiftUI до UIKit;

- можна використовувати Storyboard;
- це може зайняти багато часу, щоб створити складні уявлення, через трудомісткість використання фреймворку.

Внаслідок, Обмеження SwiftUI здебільшого пов’язані з його недосвідченістю. SwiftUI має великий потенціал, але він, потребує більше часу. UIKit не зникне найближчим часом, оскільки SwiftUI побудований на його основі і навіть має власний перехідний API, що вказує на те, що Apple добре знала про недоліки SwiftUI. У багатьох відношеннях UIKit є полярною протилежністю: він має численні переваги завдяки своїй зрілості, але йому бракує ефективного та швидкого стилю розробки SwiftUI. Однак один ключовий принцип залишається незмінним: все, що можна робити в SwiftUI, можна зробити в UIKit. Однак те, що можна досягти в UIKit, не обов’язково легко виконати в SwiftUI.

Висновки до розділу 2

У даному розділі було розглянуто основні проблеми, які постають перед розробкою застосунку. Було розглянуто вибір бази даних, яка є невід’ємною частини кожного застосунку. Окрім цього було розглянуто мову програмування, фреймворк, та існуючі шаблони проектування для написання застосунку.

3. МОДЕЛЮВАННЯ ЗАСТОСУНКУ

3.1 Розробка інтерфейсу системи

Для розробки інтерфейсу було обрано програмний застосунок Figma. Figma дозволяє швидко робити дизайн складних систем, та має багато стандартних елементів, які саме і прискорюють роботу над програмою. Було вирішено зробити дизайн усіх екранів спочатку у Figma, а потім скопіювати дизайн екранів, вже у застосунку.

Загальний дизайн системи зображений на рисунку 3.1

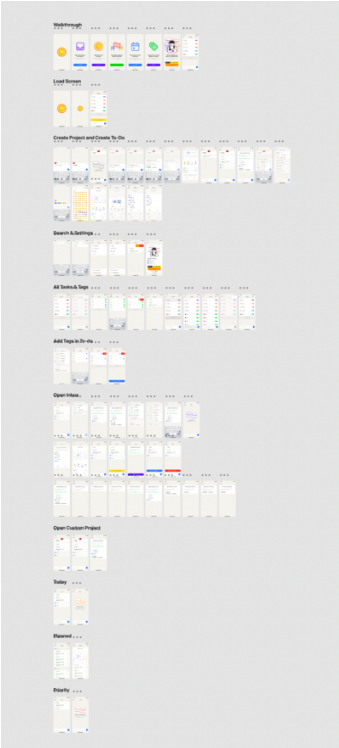


Рисунок 3.1 – Дизайн усіх екранів застосунку

Також, є досить важливим показати інші екрани, їх вигляд, щоб порівняти з остаточним результатом внаслідок розробки застосунку. Дизайн інших екранів розташовані на рисунках 3.2, 3.3 та 3.4

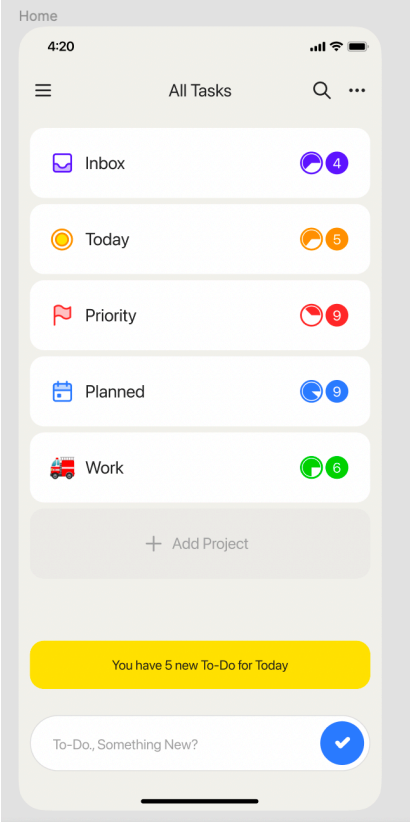


Рисунок 3.2 – Вигляд головного екрану

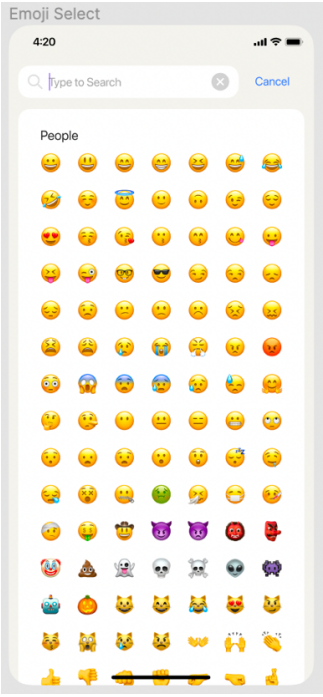


Рисунок 3.3 – Вигляд екрану вибору іконки проекту

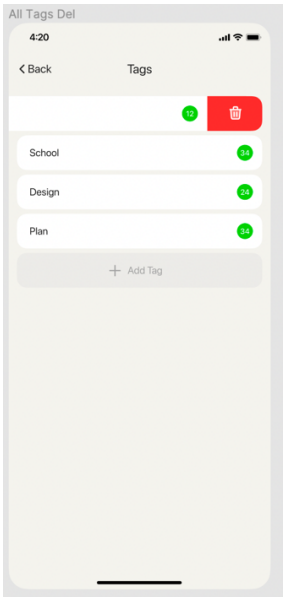


Рисунок 3.4 – Вигляд екрану з тегами

3.2 Вибір функцій, які потрібно реалізувати у рамках проекта

Функції та можливості які були обрані, включають у себе: створення власні проекти, які виконують функцію папки та допомагають об’єднати однорідні завдання в одну категорію або розташувати їх у будь-який спосіб, який користувач вважає доцільним; змінювати різні атрибути користувальницьких проектів, наприклад назву, колір та піктограму; створювати завдання безпосередньо всередині проекту.

- за замовчуванням програма має 4 спеціальні папки:
  - папка «Вхідні», яка діє як індивідуальний проект, але не може бути видалена ні в якому разі;
  - папка «Сьогодні», яка складається із завдань, запланованих на сьогодні, як випливає з назви;
  - папка «Пріоритет», яка складається з усіх завдань із зазначеним пріоритетом (ідея пріоритетів буде пояснена нижче);

  - папка «Запланований», яка складається з усіх завдань із зазначеною датою (ідея дат буде пояснена нижче).
- створення завдань поза проектом; при цьому завдання поміщаються в папку «Вхідні» за замовчуванням:
- створення завдань з певними атрибутами, такими як «Назва», «Опис», «Теги», «Пріоритет» і «Дата»; установка та зняття статусу виконаного завдання; перевірення деталей завдання з усіма атрибутами, такими як «Назва», «Опис», «Теги», «Пріоритет» і «Дата».
- редагування будь-якого з атрибутів завдання на екрані деталей завдання та можливість видалити завдання.
- Наявна можливість змінення сортувань завдань на екрані деталей проекту за датою створення, пріоритетом, назвою та датою (вказана в завданні, вона відрізняється від дати створення). Доступне додання підзавдання до будь-якого завдання на екрані деталей завдання: можна написати скільки завгодно підзавдань; можна видалити будь-яке підзавдання зі списку за допомогою проведенням пальцями; можна перевіряти та знімати галочку з виконаного статусу завдання.
- Можливість відкрити екран вибору тегів, натиснувши на будь-який із наявних тегів або тег із трьома крапками, що вказує на можливість додати більше тегів до завдання: можна виконати всі завдання в проекті, натиснувши кнопку «Завершити все» в додатковому меню на екрані проекту;
- Видалення всіх виконаних завдань в проектах, натиснувши кнопку «Видалити завершені» в додатковому меню на екрані проекту; видалення користувацький проект на екрані проектів, натиснувши кнопку «Видалити проект» у додатковому меню на екрані проекту
- Пошук конкретних завдань за назвою та мати можливість встановлювати та знімати прапорці окремих завдань: відкриття всіх

тегів натиснувши на кнопку «Відкрити теги» в додатковому меню на екрані всіх проектів можна переглянути всі теги створені користувачем протягом усього використання програми; Також можна перевірити кількість завдань, які мають однакове завдання, і перевірити, яке завдання має певні теги, натиснувши назву тега.

- Відкриття екрану налаштувань торкнувшись значка трьох ліній на екрані всіх проектів. Екран налаштувань допомагає отримати доступ до таких функцій:
- архів, який допомагає перевіряти завдання, які були видалені раніше, і, можливо, відновити їх при потребі. Функція доступна тільки в преміум-версії.
- підключити преміальні функції, які доступні за фіксовану суму грошей. Преміум включає:
  - необмежена кількість тегів, за замовчуванням максимальна кількість тегів обмежена
  - необмежена кількість нагадувань, які також називаються датами.
  - Розблокування архіву
  - Необмежена кількість атрибутів пріоритету
- змінити мову програми;
- рекомендувати програму друзям
- залишти відгуки та пропозиці, кнопка веде на сторінку в Twitter.

3.3 Архітектурний підхід до коду системи застосунку

Одним із популярних методів проектування є MVVM. MVVM спочатку з’явився на початку 2000-х в Microsoft, щоб дати простий метод для проектування та розробки платформ XAML-застосунків, таких як

Silverlight [8]. Пізніше він став популярним у спільноті iOS для створення адаптивних застосунків [9].

MVVM складається з трьох частин: Model; View; ViewModel. Модель у MVVM, як і інші шаблони Model-View-*, містить бізнес-логіку програми, яка включає збереження даних, маніпулювання даними, мережеву логіку тощо [10]. Представлення все ще відповідає за відтворення елементів інтерфейсу користувача та показ даних користувачеві. Новий компонент з ім’ям ViewModel використовується для зв’язку між представленням і моделлю. Він також служить посередником, забезпечуючи безперебійну передачу даних. Коли відбуваються нові зміни в даних, ViewModel може оновлювати модель і слухати події зміни стану.

Багато архітектурних підходів схожі між собою. Наприклад, навіть назва шаблонів часто має ті самі компоненти, а саме Model и View у Model-View-Controller та Model-View-ViewModel.

Використовуючи архітектурний підхід MVVM, структура проекту буду виглядати як на рисунку 3.5

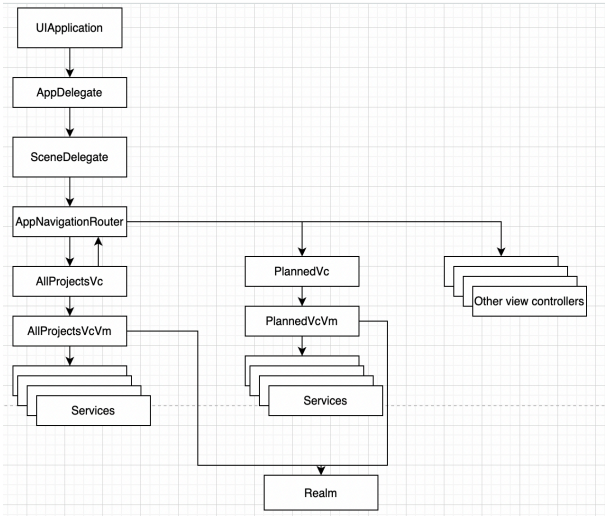


Рисунок 3.5 – Структура застосунку

Розберемо частини діаграми, зображеної на рисунку 3.5:

- UIApplication - Основний центр управління та координації для всіх програм iOS;
- AppDelegate – Делегат програми, який слугує для керування поведінки програми;
- UISceneDelegate - Основні методи, які використовуються для реагування на події життєвого циклу, що відбуваються в сцені;
- AppNavigationRouter – Процес переходу з екрану на екран передбачає створення даних для нових представлень, та створення цих представлень належним чином. Для того, щоб розвантажити класи представлень, і створюється такий об’єкт;
- AllProjectsVc –ViewController об’єкт у архітектурному підході MVVM для екрану з демонстрацією усіх проектів;
- AllProjectsVcVm – ViewModel об’єкт у архітектурному підході MVVM для екрану з демонстрацією усіх проектів;
- Services – Можливі сервісні об’єкти, які можуть бути використані при розробці застосунку. Прикладами може бути робота з мережею, відслідковування подій, аналітика;
- PlannedVc – ViewController об’єкт у архітектурному підході MVVM для екрану з демонстрацією запланованих завдань, у вигляди календаря;
- PlannedVcVm – ViewModel об’єкт у архітектурному підході MVVM для екрану з демонстрацією запланованих завдань, у вигляди календаря;
- Realm – база даних, яка використовується для збереження даних. Робота з базою даних відбувається без використання абстракцій для отримання повний доступ можливих функцій, які NoSQL база даних може надати. В цьому підході є свої недоліки, такі як складність зміни бази даних, але враховуючи плюси у вигляді унікальних

функцій, які інші бази даних не надають, було вирішено використовувати саме цю базу даних.

зВисновки до розділу 3

В данному розділі було розглянуто дизайн застосунку для інтелектуальної системи тайм-менеджменту працівника. Далі було обрано необхідний функціонал, який потрібно реалізувати при створенні застосунку.

Останнім кроком було обрання архітектурного підходу до розробку застосунку, та створення діаграми відношень компонентів системи.

4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

4.1 Огляд засобів для розробки застосунку

Для розробки iOS застосунків використовують зазвичай два застосунки: AppCode та Xcode. Зазвичай, у розробці iOS застосунків використовується Xcode так як він був розроблений компанією Apple і включає усі необхідні функції для розробника в себе. Xcode це IDE, яке використовується для створення програмного забезпечення для macOS, iOS, watchOS і tvOS. Це єдиний офіційно підтримуваний інструмент для розробки та розповсюдження програм у магазині затосунків Apple, і він підходить як для початківців, так і для професійних розробників.

Зображення інтерфейсу IDE зображений на рисунку 1

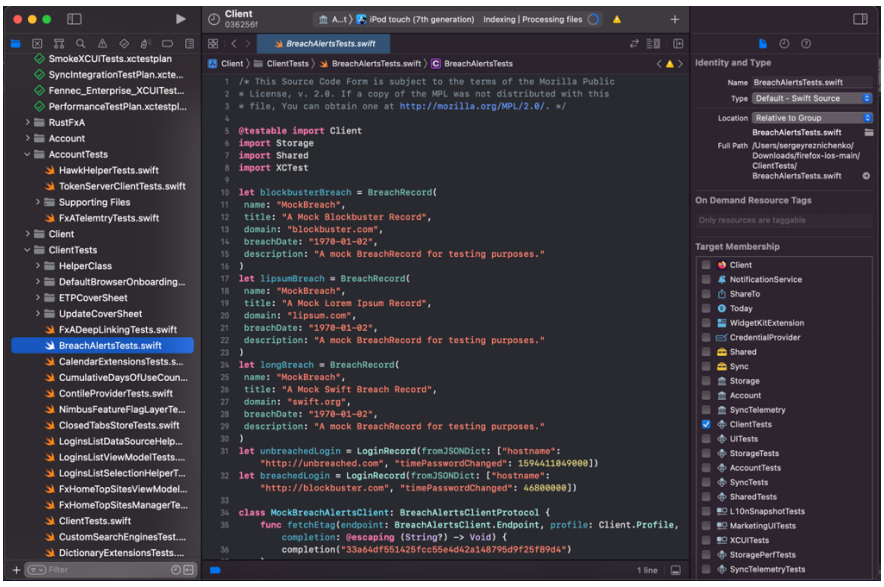


Рисунок 4.1 – Інтерфейс Xcode

Xcode являється пакетом програм, який містить усі інструменти, необхідні для створення програми, включаючи текстовий редактор,

компілятор і систему збірки. Можна розробляти, компілювати та налагоджувати програму за допомогою Xcode, а потім надсилати її до Apple App Store після її завершення. Він має різноманітні інструменти для прискорення процесу розробки, дозволяючи досвідченим розробникам створювати програми в рекордно короткий термін, а новачки відчувають менше складності та перешкод.

При створенні проекту, необхідно обрати бажану мову програмування та тип інтерфейсу. Для створення проекту потрібно для початку обрати шаблон проекту. Процес обрання шаблону зображений на рисунку 4.2.

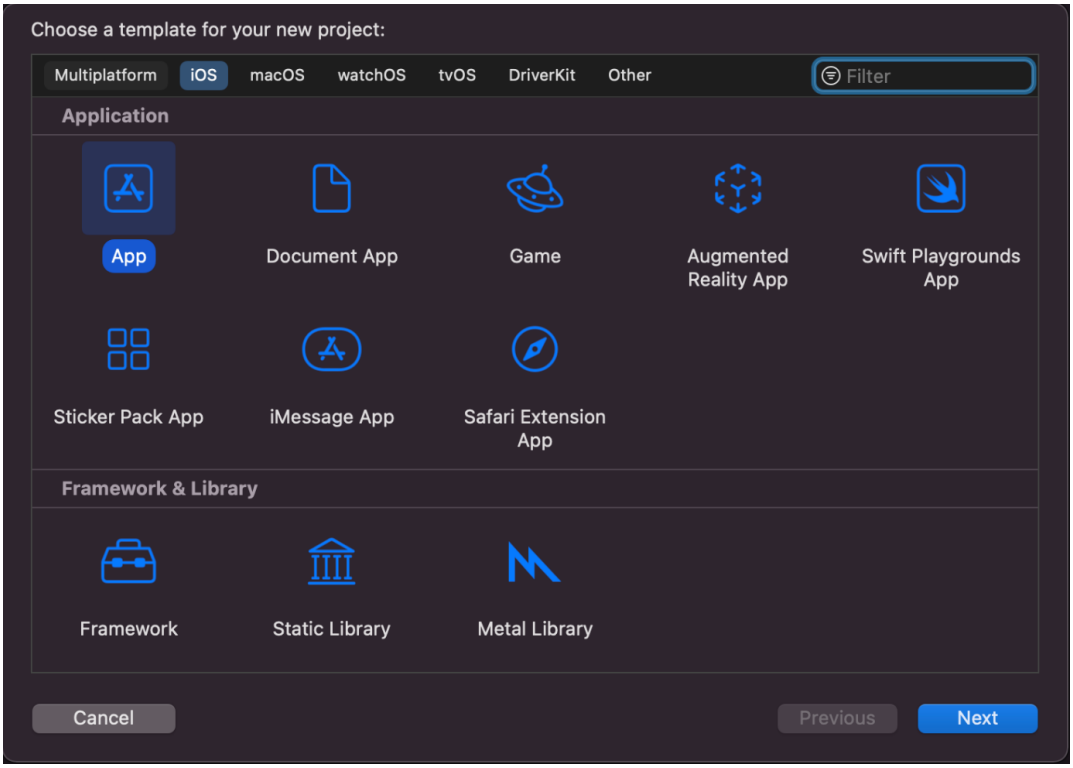


Рисунок 4.2 – Вибір шаблону для створення проекту

Потім необхідно заповнити конфігураційні дані для проету, такі як назва проекта, мова програмування, використання бази даних, та включення

тестів до проекту. Процес заповнення конфігурації зображений на рисунку 4.3

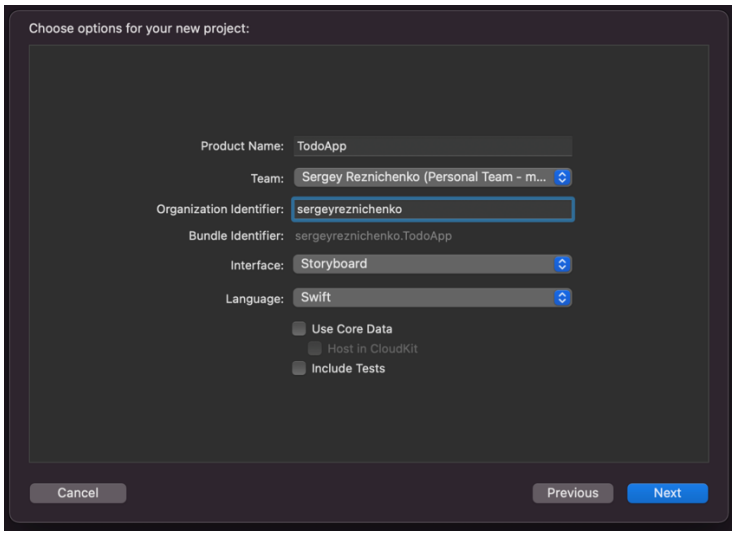


Рисунок 4.3 – Вибір конфігурацій проекту

4.2 Опис програмної реалізації застосунку

Для початку слід розробити подбати про конфігурацію проекта. Для конфігурації, такі як назва проекту, використовувані шрифти, номер версії та зборки використовується файл Info.plist. Зміст файлу Info.plist зображений на рисунку 4.4

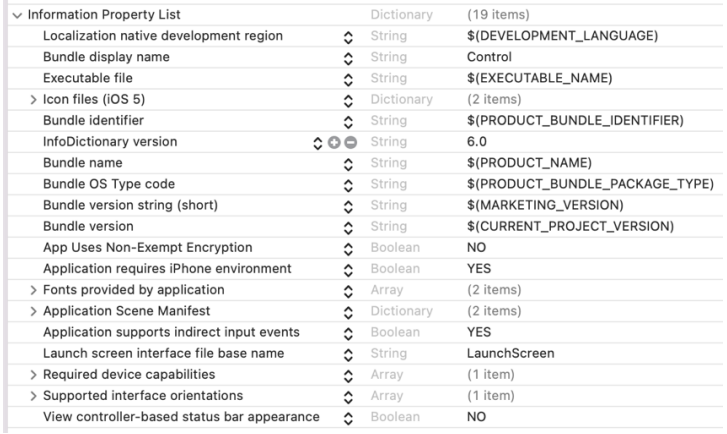


Рисунок 4.4 – Конфігураційний файл

Для відображення проекту у AppStore необхідно загрузити зображення іконки застосунку та додати у Bundle проекту. Для цього потрібно зробити зображення іконки для таких розмірів:

- 20pt x 20pt; 2x та 3x;
- 29pt x 29pt; 2x та 3x;
- 40pt x 40pt; 2x та 3x;
- 60pt x 60pt; 2x та 3x;
- 20pt x 20pt; 2x та 1x;
- 29pt x 29pt; 1x та 2x;
- 40pt x 40pt; 1x та 2x;
- 76pt x 76pt; 1x та 2x;
- 83.5pt x 83.5pt; 2x;
- 1024pt x 1024pt; 1x.

В нашому випадку, ми будемо використовувати одне і то же зображення, тому необхідно використовувати програму, яка може згенерувати зображення, необхідних нам розмірів. Для цього, можна використати програму ImageMagick [11], яка дозволяє нам конвертувати зображення одного розміру у інший. Тому, ми використаємо скрипт, який дозволить нам згенерувати потрібні зображення.

```
#!/bin/sh

imagePath=$1

convert "$imagePath" -resize '40x40'     -unsharp 1x4 "Icon-Small-40.png"
convert "$imagePath" -resize '76x76'     -unsharp 1x4 "Icon-76.png"
convert "$imagePath" -resize '58x58'     -unsharp 1x4 "Icon-Small@2x.png"
convert "$imagePath" -resize '114x114'   -unsharp 1x4 "Icon@2x.png"
convert "$imagePath" -resize '60x60'     -unsharp 1x4 "Icon-60.png"
convert "$imagePath" -resize '72x72'     -unsharp 1x4 "Icon-72.png"
convert "$imagePath" -resize '512x512'   -unsharp 1x4 "iTunesArtwork"
convert "$imagePath" -resize '192x192'   -unsharp 1x4 "Icon-xxxhdpi.png"
convert "$imagePath" -resize '80x80'     -unsharp 1x4 "Icon-Small-40@2x.png"
convert "$imagePath" -resize '120x120'   -unsharp 1x4 "Icon-60@2x.png"
convert "$imagePath" -resize '144x144'   -unsharp 1x4 "Icon-72@2x.png"
convert "$imagePath" -resize '50x50'     -unsharp 1x4 "Icon-Small-50.png"
convert "$imagePath" -resize '57x57'     -unsharp 1x4 "Icon.png"
convert "$imagePath" -resize '152x152'   -unsharp 1x4 "Icon-76@2x.png"
convert "$imagePath" -resize '100x100'   -unsharp 1x4 "Icon-Small-50@2x.png"
convert "$imagePath" -resize '29x29'     -unsharp 1x4 "Icon-Small.png"
convert "$imagePath" -resize '180x180'   -unsharp 1x4 "Icon-60@3x.png"
convert "$imagePath" -resize '1024x1024' -unsharp 1x4 "iTunesArtwork@2x"
convert "$imagePath" -resize '144x144'   -unsharp 1x4 "Icon-xxhdpi.png"
convert "$imagePath" -resize '36x36'     -unsharp 1x4 "Icon-ldpi.png"
convert "$imagePath" -resize '48x48'     -unsharp 1x4 "Icon-mdpi.png"
convert "$imagePath" -resize '72x72'     -unsharp 1x4 "Icon-hdpi.png"
convert "$imagePath" -resize '96x96'     -unsharp 1x4 "Icon-xhdpi.png"
```

Рисунок 4.5 – Скрипт для генерування іконок

Після встановлення іконки та конфігурації, необхідно подбати про встановлення головного екрану та зробити початкові налаштування. Для цього, потрібно встановити всі ці значення після того як фреймворк від UIKit зробить усі налаштування на своїй стороні.

Метод scene(_ scene:, willConnectTo:, options:) може частиною коду, яка може допомогти зробити початкові конфігурування для проекту. Розглянемо використання коду у цієї частині

Код цієї частини розташований на рисунку 4.6

```
14      var window: UIWindow?
15      var realmObserverToken: Any?
16
17      func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions:
            UIScene.ConnectionOptions) {
18          guard let windowScene = scene as? UIWindowScene else { return }
19          SwiftDate.defaultRegion = .current
20          _ = InAppManager.shared
21          let window = UIWindow(windowScene: windowScene)
22          realmObserverToken = RealmProvider.main.realm.observe { (_, _) in
23              WidgetCenter.shared.reloadAllTimelines()
24          }
25          if !(RealmProvider.main.configuration.fileURL.flatMap { DirPath.fileExists($0) } ?? false) {
26              let languagePrefix = Locale.preferredLanguages.first
27              if languagePrefix == "en" {
28                  try! FileManager.default.copyItem(at: RealmProvider.bundled_en.configuration.fileURL!, to:
                        RealmProvider.main.configuration.fileURL!)
29              } else {
30                  try! FileManager.default.copyItem(at: RealmProvider.bundled_any.configuration.fileURL!, to:
                        RealmProvider.main.configuration.fileURL!)
31              }
32          }
33          let viewController = AllProjectsVc()
34
35          let navigationVc = TANavigationController(rootViewController: viewController)
36          AppNavigationRouter.shared.navigationController = navigationVc
37
38          window.rootViewController = navigationVc
39
40          self.window = window
41          window.makeKeyAndVisible()
42      }
```

Рисунок 4.6 – Код, який виконується на початку програму

Розглянемо кожну зі основних рядків, та їх призначення

- рядок 19 – встановлення формату дати згідно з регіоном. Встановлюємо регіон, в якому знаходиться користувач;

- рядок 20 – використовується шаблон Singleton. При початку програми потрібен для виконання коду ініціалізатору;
- рядки 22 – 24 – При змінні даних у базі даних треба обновити дані у віджеті;
- рядки 25 – 32 – При відсутності створеної бази даних, перенесемо стандартну базу даних, з первісними даними;
- рядок 33 – Створимо ViewController за архітектурним підходом MVVM, який буде відображати список усіх проектів;
- рядок 36 – Використаємо підхід Router, який дозволить робити переходи з одного екрану до іншого;
- рядок 38 – встановимо екран з відображенням усіх проектів як початковий, тобто перший, який потрібно відобразити.

Розглянемо методи створення екранів для нашого застосунку. Так як в проекті використовується фреймворк UIKit, то для створення представлень з відображенням списків необхідно використати клас цього фреймворку UITableView. Цей клас дозволяє створювати колекції з великої кількістю параметрів. У рамках цієї роботи, ми розглянемо лише параметри, які були використані нами.

Були використані такі параметри:

- встановити власний колір заднього фону. В нашому випадку колір прозорий;
- обрати власний вид роздільнику. В нашому випадку – відсутній;
- обрати висоту кожної з комірки. В нашому випадку – 62 pt чи 80 pt в залежності від розмірів смартфону.

Також, потрібно подбати про використання жестів при використанні застосунку. Наприклад, у нашому випадку є задача відкриття користувальницьких налаштувань при проведеннях пальцями з лівої частини екрану до правої. Екран налаштувань зображений на рисунке 4.7

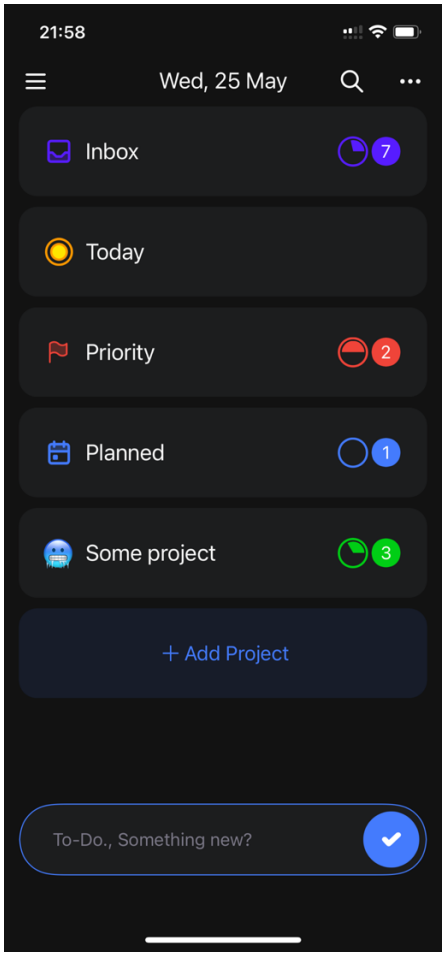


Рисунок 4.7 – Вид екрану з усіма проектами

Для додавання жестів необхідно використати клас UIGestureRecognizer. У нашому випадку, для додавання жестів проведення пальців з лівої частини екрана до правої необхідно використати підклас UIScreenEdgePanGestureRecognizer, та у властивість edges цього класу присвоїти значення UIRectEdge.left та додати жест до екрану. Оброблювати використання цього жесту буде метод didPan(gesture:), який в залежності від того, в якому стані знаходиться від стану жесту знаходиться код, відобразить новий контролер на екрані. Код класу AllProjectsVc розташований у додатку А.

Розглянемо роботу ViewModel класу. Цей клас має працювати з базою даних: мати доступ к даним, та надавати дані представленням. Для цього, спостерігаються зміни у базі даних та поновлюється представлення при змінні даних у базі даних. База даних Realm може оповіщати користувача при можливих змінах у вигляді трьох подіях:

- Update – зміна об’єктів у базі даних. Для цього разом з подією передаються змінені об’єкті, видалені, змінені масиви об’єктів;
- Initial – створення нового об’єкту у базі даних;
- Error – помилка при роботі з базою даних.

Також при роботі з базою даних, при запитах від AllProjectsVc відбуваються запити до бази даних, та повернуті дані об’єктів в залежності від запитів. Також, повернуті об’єкті мають додаткову інформацію, такі як окремі зображення, необхідні для кожного з проектів.

Інші екрани та частини проекту побудовані за схожим принципом, використовуючи архітектуру MVVM, та в залежності від потреб, які виникають в певних екранах, використовують ту чи іншу логіку. Окрім цього, проект використовує чисельну кількість різних бібліотек, опис функціональностей яких не входить у задачу цієї кваліфікаційної роботи. Використовуються бібліотеки для роботи з інтерфейсом користувача, та службові.

Для роботи з інтерфейсом користувача використовуються такі бібліотеки:

- Material – набір UI компонентів, стилів, зображень, які прискорюються швидкість розробки застосунку;
- NewPopMenu – UI компонент, який допомогає відобразити popup елементи;
- SnapKit – допоміжні функції для створення Layout відображення;

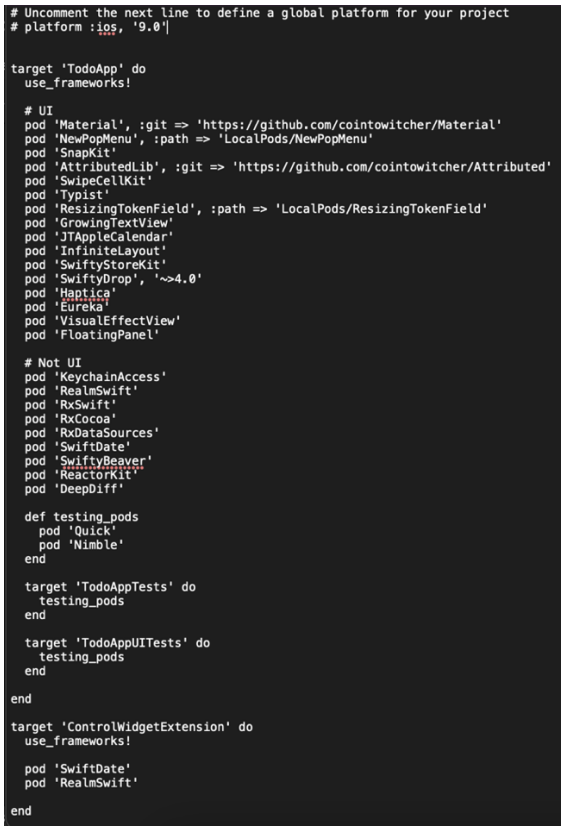
- AttributedLib – дозволяє створювати рядки з атрибутами та форматуванням у простий спосіб;
- SwipeCellKit – дозволяє додати функцію проведення пальцями для відкриття додаткових дій у клітинках;
- Typist – помічник при роботі з клавіатурою;
- ResizingTokenField – колекція, для роботи з тегами;
- GrowingTextView – робота з текстовими полями, які розширюються униз;
- JTAppleCalendar – готовий вид календаря;
- InfiniteLayout – відображення нескінченної кількості елементів у колекції;
- SwiftyStoreKit – зручна робота з платежами Apple;
- SwiftyDrop – відображення додаткових UI елементів;
- Haptica – помічник при роботі з вібраціями;
- Eureka – допомагає створювати форми у простий спосіб. Використовується окремо, у меню для розробника;
- VisualEffectView – помічник при роботі з візуальними ефектами;
- FloatingPanel – будівельник форм для компонентів, які схожі до тих, що представлені у Apple Maps, Shortcuts та Stocks застосунків.

До службових бібліотек відносяться:

- KeychainAccess – забезпечення безпеки – необхідність у сучасних пристроях. Ця бібліотека дозволяє працювати з Keychain механізмом від Apple в більш простий спосіб;
- RealmSwift – обгортка над базою даних Realm для простого використання;
- RxSwift – використання реактивного програмування у застосунку;
- RxCocoa – додавання реактивного стилю програмування до взаємодії з UI компонентами ;

- RxDataSources – додавання реактивного стилю програмування до взаємодії з таблицями та колекціями;
- SwiftDate – робота з датами;
- SwiftyBeaver – більш зручна робота з логуванням.

Для додавання усіх залежних пакетів до проекту, у цієї роботи використовується менеджер пакетів CocoaPods. Для цього створюється конфігураційний файл Podfile. Зміст цього файлу зображений на рисунку 4.8



```
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

target 'TodoApp' do
  use_frameworks!

  # UI
  pod 'Material', :git => 'https://github.com/cointowitcher/Material'
  pod 'NewPopMenu', :path => 'LocalPods/NewPopMenu'
  pod 'SnapKit'
  pod 'AttributedLib', :git => 'https://github.com/cointowitcher/Attributed'
  pod 'SwipeCellKit'
  pod 'Typist'
  pod 'ResizingTokenField', :path => 'LocalPods/ResizingTokenField'
  pod 'GrowingTextView'
  pod 'JTAppleCalendar'
  pod 'InfiniteLayout'
  pod 'SwiftyStoreKit'
  pod 'SwiftyDrop', '~>4.0'
  pod 'Haptica'
  pod 'Eureka'
  pod 'VisualEffectView'
  pod 'FloatingPanel'

  # Not UI
  pod 'KeychainAccess'
  pod 'RealmSwift'
  pod 'RxSwift'
  pod 'RxCocoa'
  pod 'RxDataSources'
  pod 'SwiftDate'
  pod 'SwiftyBeaver'
  pod 'ReactorKit'
  pod 'DeepDiff'

  def testing_pods
    pod 'Quick'
    pod 'Nimble'
  end

  target 'TodoAppTests' do
    testing_pods
  end

  target 'TodoAppUITests' do
    testing_pods
  end

end

target 'ControlWidgetExtension' do
  use_frameworks!

  pod 'SwiftDate'
  pod 'RealmSwift'

end
```

Рисунок 4.8 – зміст конфігураційного файлу Podfile

Після обрання необхідних бібліотек, виконується команда pod init, яка починає завантаження код усіх необхідних бібліотек та інтегрує у проект.

Висновки до розділу 4

В останньому розділі кваліфікаційної роботи був описаний процес створення застосунку використовуючі сучасні інструменти розробки. Також були описані процеси, які входять до конфігурації проекту та налаштування його розробки, додавання бібліотек сторонніх розробників.

Спеціальний розділ

ОХОРОНА ПРАЦІ

до кваліфікаційної роботи

на тему:

«ІНТЕЛЕКТУАЛЬНА СИСТЕМА ТАЙМ-МЕНЕДЖМЕНТУ ПРАЦІВНИКА»

Спеціальність 122 «Комп’ютерні науки»

122 – БКР – 402.21810222

Виконав студент 4-го курса, групи 402

_____ ***С.О. Резніченко***

«__» _____ 2022 р.

Консультант, **старший викладач**

_____ Макарова О. В.

«__» _____ 2022 р.

Миколаїв – 2022

5. ОХОРОНА ПРАЦІ

ВСТУП

Досить відомим фактом є те, що використання комп'ютерних технологій може значно вплинути на здоров'я людей, які працюють за ним.

Зазвичай ці проблеми можуть виникають через неправильне використання комп'ютера та експлуатування технологій неналежним чином. Наприклад, досить велику роль відіграє те, в якій позі сидить людина, як часто вона робить вправи на м'язи очей та її фізичну підготовку. Проте, сьогодні кожен бізнес використовує інформаційні технології або працює з різним технічним обладнанням. Дуже важливим є надання робочого місця працівникам, які будуть більш сприятливих для здоров'я, а також профілактичних заходів для пом'якшення шкідливого впливу персональних комп'ютерів на здоров'я.

Наприклад, у проблеми можна віднести зображення на екрані, яке динамічно оновлюється та мерехтить через низьку частоту оновлення. Надмірне навантаження також зношує очі та внутрішньоочні м'язи, які концентрують погляд. Короткозорість виникає внаслідок зорової втоми. Тривала робота за комп'ютером вимагає підвищеної концентрації, що може призвести до головного болю, нетерпіння, нервового напруження та стресу.

Мета: створення безпечних і здорових умов праці на робочих місцях для офісних приміщень.

Завдання: опис санітарних умов виробничого шуму, ультразвуку, інфразвуку та вимог щодо безпеки на робочому місці. Розрахунок наявного природнього освітлення у робочому приміщенні та порівняння з нормативними даними.

1. Опис робочих місць, приміщення та обладнання. Складання вихідних даних для оцінки умов праці

Приміщення робочого кабінету, в якому виконувалась робота над дипломної роботою проходив у 4 корпусі Чорноморського Національного Університету розташованого на другому поверсі п'ятиповерхової будівлі, що розташована у м. Миколаєві на адресу вулиця 68 Десантників, Миколаїв, Миколаївська область, 54000. Розміри приміщення складають $a \times b \times H = 9{,}75 \times 4{,}55 \times 3{,}5$ м. Це приміщення має три металопластикових вікна, які мають розміри $c \times d = 1{,}9 \times 2{,}1$ м.

Для вимірювання розмірів приміщення та розміри вікон використано рулетку.

Приміщення обладнано сучасними меблями. Використовуються нові технологічні рішення, такі як світлодіодні лампи. Стіни виконані у білому цвіті. Підлога має покриття з лінолеуму.

Приміщення може вмістити 15 осіб, які можуть працювати за робочим місцем, яке буде обладнано комп'ютером, власним столом, та кожен з робочих місць має своє відділення для речей у вигляді шафи. Окрім цього, доступні сучасні персональні комп'ютери з усіма необхідними аксесуарами, два лазерні принтери та лазерний ксерокс. Робоча документація та науково-технічні матеріали зберігаються у власному кабінеті. Побутова техніка (холодильник, електричний чайник, кавоварка, мікрохвильова піч) використовується на постійній основі за згодою адміністрації.

Рисунок 5.1 – Креслення виробничого приміщення

Напруга джерела живлення електрообладнання 220 В. Трипровідна мережа побудована відповідно до вимог нормативних документів. Приміщення, з точки зору ризику ураження електричним струмом, відноситься до місця, де немає підвищеного ризику ураження електричним струмом.

Відповідність нормам НПАОП 0,00-1,28-10 [12] забезпечує пожежну безпеку на зазначеному виробничому підприємстві.

Також, у кабінеті є у доступності персональні комп’ютери з усіма необхідними аксесуарами, також принтери та ксерокс, які необхідні працівникам. Робоча документація та науково-технічні матеріали зберігаються у власному кабінеті.

Система кондиціонування забезпечує мікрокліматичні умови влітку, але її потужності не завжди вистачає для комфортних умов роботи. Опалення взимку забезпечується централізованою системою, яка не завжди забезпечує необхідний тепловий режим.

Оцінка умов праці (основних аспектів виробничого середовища та трудового процесу) у обраному виробничому приміщенні здійснюється на основі контрольних обстежень, виконання відповідних вимірювань та експертних оцінок (табл. 1). Зміст цих розрахунків використовуються у подальшому обрахунку для спеціального розділу бакалаврської роботи.

В приміщенні використовуються дуже багато можливих пристроїв, які потребують електроенергію, та не мало важливо, що мають вплив на очі працівників, які працюють у приміщенні. Не можна легковажно відноситись до цього факту, тому необхідно в спеціальному розділі розглянути цей факт детально.

2. Оцінка природнього освітлення у процесі виробництво

Освітлення місця денним світлом, що проникає через світлові прорізи (вікна) у зовнішніх захисних конструкціях приміщення, називається природним освітленням. Природне освітлення має широкий діапазон залежностей, наприклад від часу доби, сезону та ряду інших факторів. Природнє освітлення було розраховано відповідно до методичних вказівок, наданих у документі ДБН В.2.5-28:2018. Природне і штучне освітлення [22].

Проведемо розрахунок освітлення для кімнати, описаної у пункті 1.

Нормативне значення коефіцієнта природного освітлення для III поясу світлового клімату $e_{\text{Н}}^{\text{Ш}}$, %.

Визначається відповідно до ДБН В.2.5-28:2018 [22]. Для зорових робіт середньої точності при найменшому розмірі об'єкта розпізнавання 0,5-1 мм при боковому освітленні (так званий IV розряд зорової роботи):

$$e_{\text{Н}}^{\text{Ш}} = 1\%$$

Знаходження коефіцієнту світлового клімату m

Миколаївська область відноситься до IV поясу [22] світлового клімату, де

$$m = 0{,}9$$

Знаходження коефіцієнту сонячного клімату c.

Визначається відповідно до ДБН В.2.5-28:2018 [22]. Для світлових отворів (вікон) в зовнішніх стінах будівель, розташованих у IV поясі світлового клімату та зорієнтованих по азимуту в діапазоні 136... 225 градусів

$$c = 0{,}7$$

Розрахування нормованого значення коефіцієнта природного освітлення для розрахункових умов $e_{\text{н}}$, %:

$$e_{\text{н}} = e_{\text{Н}}^{\text{Ш}} * m * c = 1 * 0{,}9 * 0{,}7 = 0{,}63\%$$

Розрахунок відношення довжини приміщення a до його ширини b, a/b:

$$\frac{a}{b} = \frac{9{,}75}{6{,}175} = 1.57$$

Розрахунок відношення ширини приміщення b до відстані від верхньої кромки вікна до робочої поверхні h, b/h

$$\frac{b}{h} = \frac{6{,}175}{2{,}45} = 2{,}52$$

Розрахунок світлової характеристики вікна ηв

Визначається за формулою

$$\eta_{\text{в}} = f\left(\frac{a}{b}, \frac{b}{h}\right) = f(1{,}57, 2{,}52) = 13$$

Визначення коефіцієнту світлопропускання матеріалу τ_1.

Визначається відповідно до ДБН В.2.5-28:2018 [22]. Світлопрозорий матеріал(стекла) завтовшки 4мм

$$\tau_1 = 0{,}87$$

Визначення коефіцієнта τ_2

Визначається відповідно до ДБН В.2.5-28:2018 [22]. Віконні рами використовуються металеві, одинарні, які відкриваються

$$\tau_2 = 0{,}75$$

Визначення коефіцієнта τ_3

Визначається відповідно до ДБН В.2.5-28:2018 [22]. При боковому освітленні згідно рекомендацій

$$\tau_3 = 1$$

Визначення коефіцієнта τ_4

Визначається відповідно до ДБН В.2.5-28:2018 [22]. Жалюзі не використовуються. Використовується горизонтальний козирок з захисним кутом не більше 30

$$\tau_4 = 0{,}8$$

Розрахунок загального коефіцієнту світлопропускання

$$\tau_{\text{заг}} = \tau_1 * \tau_2 * \tau_3 * \tau_4 = 0{,}87 * 0{,}75 * 1 * 0{,}8 = 0{,}522$$

Визначення коефіцієнту відбиття внутрішніх поверхонь приміщення.

Для підлоги:

$$\rho_{\text{підлоги}} = 40\%$$

Для стін:

$$\rho_{\text{стін}} = 40\%$$

Для поверхні стелі:

$$\rho_{\text{стелі}} = 82\%$$

Розрахунок площ внутрішніх поверхонь

$$S_{\text{підлоги}} = a * b - a_1 * b_1$$

$$= 9{,}75 * 6{,}175 - 1{,}41 * 1{,}41 =$$

$$58{,}2 \text{ м}^2$$

$$S_{\text{стін}} = a_1 * H + a_2 * H + a_3 * H + a_4 * H + a_5 * H$$

$$= (a_1 + a_2 + a_3 + a_4 + a_5) * \text{H}$$

$$= (6{,}175 + 9{,}75 + 4{,}55 + 7 + 2) * 3{,}5 = 103 \text{ м}^2$$

$$S_{\text{стелі}} = a * b - a_1 * b_1 = 58{,}2 \text{ м}^2$$

Розрахунок середнього значення коефіцієнту відбиття

$$\rho_{\text{сер}} = \frac{\rho_{\text{стелі}} * S_{\text{стелі}} + \rho_{\text{стін}} * S_{\text{стін}} + \rho_{\text{підлоги}} * S_{\text{підлоги}}}{\left(S_{\text{стелі}} + S_{\text{стін}} + S_{\text{підлоги}}\right)} =$$

$$\frac{0{,}82 * 58{,}2 + 0{,}4 * 103 + 0{,}4 * 58{,}2}{(58{,}2 + 103 + 58{,}2)} = 0{,}511413$$

Розрахунок співвідношення геометрії приміщення

$$\frac{a}{b} = 1{,}57$$

$$\frac{b}{h} = 2{,}52$$

$$\frac{1}{b} = 0{,}161$$

Визначення коефіцієнту підсилення природного освітлення

$$r_1 = f\left(\rho_{\text{сер}}, \frac{a}{b}, \frac{b}{h}, \frac{1}{b}\right) = f(0.511, 1.57, 2.52, 0.161) = 2$$

Розрахунок відношення відстані до протилежної будівлі до висоту карнизу протилежної будівлі над підвіконням

$$\frac{\text{D}}{\text{H}} = \frac{21}{8} = 2{,}63$$

Розрахунок площі вікон, необхідної для забезпечення нормованого коефіцієнту освітлення.

$$S_{\text{в}} = \frac{e_{\text{н}} * \text{к}_{\text{з}} * \eta_{\text{в}} * S_{\text{підлоги}} * \text{К}_{\text{буд}}}{\tau_{\text{заг}} * r_1 * 100} = \frac{0{,}63 * 2{,}0 * 13 * 58{,}2 * 1}{0{,}522 * 2} = 9{,}13 \text{ м}^2$$

У приміщенні знаходяться 3 вікна розміром $c * d = 1{,}9 * 2{,}1$ м. Значення загальної площі вікон дорівнює $S = c * d * 3 = 1{,}9 * 2{,}1 * 3$ м $= 11{,}97$ м^2. Для вимірювання розмірів вікон використано рулетку.

Внаслідок обрахувань та порівняння з нормативними даними, можна визначити, що приміщення має необхідну кількість вікон, та достатню кількість освітлення для світлої пори доби.

3. Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями

Відповідно до статті 28 Закону України «Про охорону праці» пунктів 8, 10 Положення про Міністерство соціальної політики України, затверджено документ «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» чинним на даний момент [13, 14].

Цей документ містить в собі правила, які потрібно дотримуватися роботодавцям та повідомити працівників про засоби безпеки.

Ці Вимоги застосовуються до всіх підприємств, незалежно від форм власності, юридичної структури чи операцій, і забезпечують основні стандарти безпеки та охорони здоров’я для роботи, яка вимагає використання екранних пристроїв, незалежно від типу чи моделі.

Окрім цього ці вимоги не виключають роботодавця від встановлення жорсткіших та/або унікальних вимог щодо безпеки та здоров’я працівників

під час використання екранних пристроїв, якщо це не заборонено відповідним законодавством.

Працівники повинні бути проінформовані про робочі умови та наявність небезпечних і шкідливих виробничих елементів (фізичних, хімічних, біологічних та психофізіологічних), які виникають при використанні екранних пристроїв і ще не ліквідовані, а також про можливі наслідки їх впливу. відповідно до здоров'я працівників [25, 19].

Роботодавець зобов’язаний за необхідності проводити лабораторні дослідження умов праці працівників з метою виявлення шкідливих і небезпечних факторів виробничого середовища, важкості та напруженості трудового процесу (зокрема щодо виявлення ризиків, пов’язаних із погіршенням зору, порушенням фізичного стану, стресом) та вживати заходів щодо усунення виявлених ризиків [20].

4. Санітарні норми виробничого шуму, ультразвуку та інфразвуку

Санітарні норми поширюються на шум, інфрачервоний та ультразвук, які передаються повітрям (газоподібним середовищем), рідким або твердим середовищем і впливають на працюючу людину [15, 18].

Доведено, що шум завдає шкоди слуховому аналізатору, а також іншим органам і системам людського організму. Обсяг шуму, його частотний склад, тривалість добового впливу, індивідуальні ознаки та деталі виробничих операцій – усе це відіграє роль у такій діяльності [21].

Параметри постійного шуму на робочих місцях, що нормуються, є рівнями звукових тисків у октавних смугах з середньогеометричними частотами 31,5; 63; 125; 500; 1000; 2000; 4000; 8000 Гц в децибелах, які визначаються за формулою:

$$|\_ = 20 \text{ Lg } P/Po,$$

де Р - середньоквадратичне значення звукового тиску у кожній октавній смузі, Па;

Ро - вихідне значення звукового тиску у повітрі, що дорівнює 2 х 10 в ступ. -5 Па.

Вимірювання шуму в октавних смугах або рівня шуму проводиться за допомогою шумоміра, який відповідає діючим вимогам Держстандарту України і має посвідчення про перевірку.

Вимірювання еквівалентних рівнів шуму слід проводити інтегруючими шумомірами та шумоінтеграторами.

Допускається використовувати індивідуальні дозиметри шуму з параметром еквівалентності q = 3 - число децибел, що додаються до рівня шуму, при зменшенні часу його дії у 2 рази для збереження тієї ж дози шуму. Прилади повинні бути перевірені в органах Держстандарту.

При проведенні вимірювань мікрофон слід розташовувати на висоті 1,5 м над рівнем підлоги чи робочого майданчика (якщо робота виконується стоячи) чи на висоті і відстані 15 см від вуха людини, на яку діє шум (якщо робота виконується сидячи чи лежачи).

Мікрофон повинен бути зорієнтований у напрямку максимального рівня шуму та віддалений не менш ніж на 0,5 м від оператора, який проводить вимірювання.

Рівні звукового тиску в октавних смугах частот, рівні звуку та еквівалентні рівні звуку на робочих місцях, обладнаних ВДТ і ПК визначені ДСанПіН 3.3.2-007-98 [16].

Час спостереження при вимірюванні октавних рівнів звукового тиску повинен відповідати зазначеним величинам рівнів звукового тиску, вказаним у таблиці 5.1.

Таблиця 5.1 – Октавні рівні звукового тиску

Час вимірювання	Помилка оцінки рівнів, дБ	Час вимірювання (с) в октавних полосах середньогеометричних частот, Гц			
		2	4	8	16
Мінімальний	+3	30	15	8	4
Рекомендований	+1	300	150	80	40

Допустимі рівні звукового тиску у октавних смугах частот, еквівалентні рівні звуку на робочому місці для висококваліфікованої роботи, що вимагає зосередження, адміністративно-керівна діяльність, вимірювальні та аналітичні роботи лабораторії: робочі місця в приміщеннях цехового керівного апарату, контор, лабораторій наведені у таблиці 5.2 [15].

Таблиця 5.2 – Рівні звукового тиску

Рівні звукового тиску в дБ в октавних смугах з середньогеометричними частотами, Гц									Рівні шуму та еквівалентні рівні шуму, ДБА, дБаекв.
31,5	63	125	250	500	1000	2000	4000	8000	
93	79	70	63	58	55	52	50	49	60

Інтенсивність потоків інфрачервоного випромінювання має не перевищувати допустимих значень, які зазначені у ДСН 3.3.6.042-99 [17, 21].

ВИСНОВКИ
до спеціального розділу

Внаслідок виконання роботи, описано робоче приміщення. Визначено вихідні дані для приміщення, такі як ширина та довжина.

Розраховано природнє освітлення у приміщені, описаному у першому пункті. Внаслідок обрахувань та порівняння з нормативними даними, визначено, що приміщення має необхідну кількість вікон, та достатню кількість освітлення для світлої пори доби.

Розглянуто вимоги щодо безпеки та захисту здоров’я під час роботи з екранними пристроями, запроваджені у Законі України.

Досліджено санітарні норми виробничого шуму, ультразвуку та інфразвуку. Наведено допустимі рівні шуму.

ВИСНОВКИ

В процесі виконання кваліфікаційної роботи було створено застосунок для тайм менеджменту працівника на операційній системі iOS.

Для досягнення мети було розв’язано наступні задачі:

- проаналізовано предметну сферу створення програмного застосунку для тайм менеджменту працівника;
- оглянуто методи створення програмного застосунку;
- обрано найкращі методи для створення застосунку;
- програмно реалізовано обрані методи у застосунку.

Перевагою розробленого застосунку є те, що він предоставляє унікальні функції працівникам, які не можна знайти у інших застосунках, які преставлені на ринку. Окрім цього, застосунок є суттєво доступнішим, ніж використання інших застосунків, та потенціал додавання нових функцій також надає переваги серед інших представленних застосунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Abramova Veronika, Bernardino Jorge, Furtado Pedro. Which NoSQL Database? A Performance Overview. — 2014
2. Cattell Rick. Scalable SQL and NoSQL data stores // ACM SIGMOD Record. — 2011. — Vol. 39, no. 4. — P. 12–27.
3. Data CJ. An introduction to database systems, 8/E. — Addison-Wesley publ., 2004.
4. Nayak Ameya, Poriya Anil, Poojary Dikshay. Type of NOSQL Databases and its Comparison with Relational Databases // International Journal of Applied Information Systems. — 2013. — Vol. 5, no. 4. — P. 16–19.
5. Tudorica Bogdan George, Bucur Cristian. A comparison between several NoSQL databases with comments and notes // Roedunet International Conference (RoEduNet), 2011 10th / IEEE. — 2011. — P. 1–5.
6. Harwinder Singh. Speed performance between Swift and Objective-C // International Journal of Engineering Applied Sciences and Technology, 2016 Vol. 1, Issue 10 / ISSN No. 2455-2143, Pages 185-189
7. Erlend Vihovde Reime Exploring the Freemium Business Model. University of Oslo.
8. Microsoft. The MVVM Pattern, 2017 [Електроний ресурс] – Режим доступу: https://msdn.microsoft.com/en-us/library/hh848246.aspx [Дата звернення 26 травня]
9. Cacheaux R and Berlin J. Advanced iOS App Architecture – Real-world app architecture in Swift 4.2. Ray Wenderlich. 2018.
10. Orlov B. iOS Architecture Patterns [Електроний ресурс] – Режим доступу: https://medium.com/iosos-x-development/ios-architecture-patterns-ecba4c38de52 [Дата звернення 26 травня]

11. ImageMagick Command-line Tools Description URL: https://imagemagick.org/script/command-line-tools.php [Дата звернення 26 травня]
12. НПАОП 0.00-1.28-10. Правила охорони праці під час експлуатації електронно-обчислювальних машин.
13. Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями [Електронний ресурс] – Режим доступу http://zakon3.rada.gov.ua/laws/show/z0508-18 - Загол. з екрану.
14. Закон України про охорону праці [Електроний ресурс] – Режим доступу https://zakon.rada.gov.ua/laws/show/2694-12#n36 [Дата звернення 26 травня]
15. Санітарні норми виробничого шуму, ультразвуку та інфразвуку ДСН 3.3.6.037-99 [Електроний ресурс] – Режим доступу https://zakon.rada.gov.ua/rada/show/va037282-99#Text [Дата звернення 26 травня]
16. Державні санітарні правила і норми роботи з ВДТ ЕОМ ДСанПІН 3.3.2.007-98 [Електронний ресурс] – Режим доступу: http://mozdocs.kiev.ua/view.php?id=2445 [Дата звернення 26 травня]
17. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень [Електронний ресурс] – Режим доступу: https://zakon.rada.gov.ua/rada/show/va042282-99#Text [Дата звернення 26 травня]
18. ГОСТ 12.1.005-88.ССБП [Електронний ресурс] – Режим доступу: http://docs.cntd.ru/document/1200003608 [Дата звернення 26 травня]
19. Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електроно-обчислювальних машин – Режим доступу:

https://zakon.rada.gov.ua/rada/show/v0007282-98#Text [Дата звернення 26 травня]

20. Наказ про затвердження положення про порядок проведення медичних оглядів працівників певних категорій – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0136-94#Text [Дата звернення 26 травня]
21. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень [Електронний ресурс] – Режим доступу: https://dnaop.com/html/34094/doc -ДСН_3.3.6.042-99 [Дата звернення 26 травня]
22. ДБН В.2.5-28:2018. Природне і штучне освітлення [Електронний ресурс] – Режим доступу: https://ledeffect.com.ua/images/__branding/dbn2018.pdf [Дата звернення 26 травня]

ДОДАТОК А

Код класів застосунку

```
import UIKit
import Motion
import Material
import RxSwift
import RxCocoa
import PopMenu
import Typist
import SwiftDate

class AllProjectsVc: UIViewController {
    let viewModel: AllProjectsVcVM = AllProjectsVcVM()
    let tableView = UITableView()
    var pushTransition = SlidePushTransition()
    private var searchVcScreenOpened: Bool = false
    private var didAppear = false

    lazy var tasksToolbar: AllTasksToolbar = {
        let view = AllTasksToolbar(frame: .zero)
        view.onClick = { [weak self] in
            let project = RealmProvider.main.realm.objects(RlmProject.self).first(where: { $0.id ==
Constants.inboxId })!
            self?.router.openProjectDetails(project: project, state: .startAddTask, isInbox: true)
        }
        return view
    }()
    private let keyboard = Typist()
    private lazy var menuButton: UIBarButtonItem = {
        let button = UIBarButtonItem(image: UIImage(named: "menu"), style: .done, target: self, action:
#selector(menuButtonClicked))
        button.tintColor = UIColor(named: "TAHeading")!
        return button
    }()

    private lazy var searchButton: UIBarButtonItem = {
```

```
        let button = UIBarButtonItem(image: UIImage(named: "search"), style: .done, target: self, action:
#selector(searchButtonClicked))
        button.tintColor = UIColor(named: "TAHeading")!
        return button
    }()

    private lazy var actionsButton: UIBarButtonItem = {
        let button = UIBarButtonItem(image: UIImage(named: "dots"), style: .done, target: self, action:
#selector(actionsButtonClicked))
        button.tintColor = UIColor(named: "TAHeading")!
        return button
    }()

    override func viewDidLoad() {
        super.viewDidLoad()
        setupViews()
        setupSettingsPushVc()
        if !UserDefaultsWrapper.shared.didOnboard {
            let onboardingVc = OnboardingVc.getOnboardingNavigation { [weak self] in
                self?.dismiss(animated: true, completion: nil)
                UserDefaultsWrapper.shared.didOnboard = true
            } onPremiumVc: { [weak self] in
                self?.dismiss(animated: true, completion: nil)
                UserDefaultsWrapper.shared.didOnboard = true
            }
            onboardingVc.modalPresentationStyle = .overFullScreen

            present(onboardingVc, animated: false, completion: nil)
        }
        // In case something bad happened. Maybe manually violated etc.
        if !(RealmProvider.main.realm.objects(RlmProject.self).contains { $0.id == Constants.inboxId }) {
            let inboxProject = RlmProject(name: "Inbox", icon: .assetImage(name: "inboximg", tintHex: "#571cff"),
notes: "", color: .hex("#571cff"), date: Date())
            inboxProject.id = Constants.inboxId
            RealmProvider.main.safeWrite {
                RealmProvider.main.realm.add(inboxProject)
            }
        }
        DispatchQueue.main.asyncAfter(deadline: .now() + 3) {
```

```
        if let windowScene = self.view.window?.windowScene {
            StoreKitHelper.maybeDisplayStoreKit(windowScene: windowScene)
        }
    }

}

private func setupViews() {
    view.backgroundColor = UIColor(named: "TABackground")
    setupNavigationBar()
    setupTableView()
    view.layout(tasksToolbar).leadingSafe(13).trailingSafe(13).bottom(-AllTasksToolbar.estimatedHeight)
}

private func setupSettingsPushVc() {
    let edgeGesture = UIScreenEdgePanGestureRecognizer(target: self, action: #selector(didPan))
    edgeGesture.edges = .left
    view.addGestureRecognizer(edgeGesture)
}

@objc func didPan(gesture: UIScreenEdgePanGestureRecognizer) {
    switch gesture.state {
    case .began:
        pushTransition.isInteractive = true
        navigationController?.pushViewController(SettingsVc(), animated: true)
    case .ended, .cancelled:
        pushTransition.isInteractive = false
    default: break
    }
    pushTransition.handlePan(gesture)
}

func setupTableView() {
    view.layout(tableView).topSafe().bottom().leadingSafe(13).trailingSafe(13)
    tableView.backgroundColor = .clear
    tableView.separatorStyle = .none
    tableView.rowHeight = UITableView.automaticDimension
    tableView.register(ProjectViewCell.self, forCellReuseIdentifier: ProjectViewCell.reuseIdentifier)
    tableView.register(AddProjectCell.self, forCellReuseIdentifier: AddProjectCell.reuseIdentifier)
```

```
        viewModel.initialValues = { [weak self] in
            self?.tableView.reloadData()
        }
        viewModel.tableUpdates = { [weak self] in
            self?.tableView.reloadData()
        }
        tableView.delegate = self
        tableView.dataSource = self
        tableView.contentInset = .init(top: 0, left: 0, bottom: 110, right: 0)
        tableView.showsVerticalScrollIndicator = false
        view.layout(gradientView).bottom().leading().trailing().height(216)
    }
    private let gradientView = GradientView()

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        view.layout(tasksToolbar).bottomSafe(Constants.vcMinBottomPadding)
        UIView.animate(withDuration: 1, delay: 0, usingSpringWithDamping: 0.5, initialSpringVelocity: 0.3) {
[weak self] in
            self?.view.layoutSubviews()
        }
        searchVcScreenOpened = false
        didAppear = true
    }

    private func setupNavigationBar() {
        applySharedNavigationBarAppearance(addBackButton: false)
        title = Date().toFormat("E") + ", \(Date().toFormat("d MMMM"))"

        navigationItem.leftBarButtonItem = menuButton
        navigationItem.rightBarButtonItems = [actionsButton, searchButton]
    }

    @objc private func searchButtonClicked() {
        router.openSearch()
    }

    @objc private func menuButtonClicked() {
        router.openSettings()
```

```
    }

    // MARK: - POPUP
    @objc private func actionsButtonClicked() {
        let actions: [PopuptodoAction] = [
            PopuptodoAction(title: "Open Tags".localizable(), image: UIImage(named: "tag"), didSelect: { [weak
self] action in
                self?.openTags(action: action)
            })
        ]
        PopMenuAppearance.appCustomizeActions(actions: actions)
        let popMenu = PopMenuViewController(sourceView: actionsButton, actions: actions)
        popMenu.appearance = .appAppearance

        present(popMenu, animated: true)
    }

    func openTags(action: PopMenuAction) {
        dismiss(animated: true, completion: nil)
        router.openAllTags(mode: .show)
    }
}

extension AllProjectsVc: UITableViewDataSource {

    func vmIndex(for indexPath: IndexPath) -> Int {
        indexPath.section
    }

    func tableView(_ tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat {
        return section == 0 ? 0 : 0.011160714285714 * UIScreen.main.bounds.height
    }

    func tableView(_ tableView: UITableView, viewForHeaderInSection section: Int) -> UIView? {
        return UIView()
    }

    func numberOfSections(in tableView: UITableView) -> Int {
        viewModel.models.count
```

```
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        1
    }

    func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        Constants.displayVersion2 ? 62 : 80
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let index = vmIndex(for: indexPath)
        let model = viewModel.models[index]
        switch model {
        case .addProject:
            let addCell = tableView.dequeueReusableCell(withIdentifier: AddProjectCell.reuseIdentifier, for:
indexPath) as! AddProjectCell
            addCell.selectionStyle = .none
            return addCell
        case let .project(project), let .inboxProject(project):
            let projectCell = tableView.dequeueReusableCell(withIdentifier: ProjectViewCell.reuseIdentifier, for:
indexPath) as! ProjectViewCell
            let progress = viewModel.getProgress(for: project)
            let isInbox = project.id == Constants.inboxId
            projectCell.configure(icon: project.icon, name: !isInbox ? project.name : "Inbox".localizable(), progress:
CGFloat(progress), tasksCount: project.tasks.count, color: project.color, iconFontSize: isInbox ? 22 : nil)
            projectCell.selectionStyle = .none
            return projectCell
        case let .planned(project), let .priority(project), let .today(project):
            let projectCell = tableView.dequeueReusableCell(withIdentifier: ProjectViewCell.reuseIdentifier, for:
indexPath) as! ProjectViewCell
            projectCell.configure(icon: project.icon, name: project.name, progress: CGFloat(project.progress),
tasksCount: project.tasksCount, color: project.color, iconFontSize: project.iconFontSize)
            projectCell.selectionStyle = .none
            return projectCell
        }
    }
}
```

```
extension AllProjectsVc: UITableViewDelegate {
    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        let index = vmIndex(for: indexPath)
        let model = viewModel.models[index]
        switch model {
        case .addProject:
            router.openAddProject()
        case let .project(project):
            router.openProjectDetails(project: project, state: .emptyOrList)
        case let .inboxProject(project):
            router.openProjectDetails(project: project, state: .emptyOrList, isInbox: true)
        case .planned(_):
            router.openPlanned()
        case .priority(_):
            router.openPredefinedProject(mode: .priority)
        case .today(_):
            router.openPredefinedProject(mode: .today)
        }
    }
}

extension AllProjectsVc: TATransitionProvider {
    func pushTransitioning(to vc: UIViewController) -> UIViewControllerAnimatedTransitioning? {
        if vc is SettingsVc {
            return pushTransition
        }
        if vc is SearchVc {
            return FadePushTransition(duration: TimeInterval(UINavigationController.hideShowBarDuration))
        }

        return nil
    }

    func popTransitioning(from vc: UIViewController) -> UIViewControllerAnimatedTransitioning? {
        return nil
    }
    func interactionController(for animationController: UIViewControllerAnimatedTransitioning) ->
UIViewControllerInteractiveTransitioning? {
        return pushTransition.isInteractive ? pushTransition : nil
```

```
    }
}

import Foundation
import UIKit
import Material
import PopMenu
import ResizingTokenField
import AttributedLib
import RxSwift
import RxDataSources
import SwipeCellKit
import Typist
import SnapKit

final class TaskDetailsVc: UIViewController {
    private let viewModel: TaskDetailsVcVm
    private let bag = DisposeBag()
    private lazy var subtasksTable: UITableView = {
        let collectionView = UITableView(frame: .zero)
        collectionView.isScrollEnabled = false
        collectionView.backgroundColor = .clear
        return collectionView
    }()
    private lazy var actionsButton: UIBarButtonItem = {
        let button = UIBarButtonItem(image: UIImage(named: "dots"), style: .done, target: self, action:
#selector(actionsButtonClicked))
        button.tintColor = UIColor(named: "TAHeading")!
        return button
    }()
    private let keyboard = Typist()
    private var isCurrentlyShown = false
    private var shouldUpdateTagsOnShown = false
    private var wasAlreadyShown: Bool = false
    private var containerStackLeadingTrailing: CGFloat = 26
    private var fpc = CustomFloatingPanel()

    init(viewModel: TaskDetailsVcVm) {
        self.viewModel = viewModel
```

```
    super.init(nibName: nil, bundle: nil)
}

required init?(coder: NSCoder) {
    fatalError("init(coder:) has not been implemented")
}

override func viewDidLoad() {
    super.viewDidLoad()
    setupViews()
    setupViewModelBinding()
    setupBindings()
    updateTags()
}
override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
    print("viewDidAppear")
    if shouldUpdateTagsOnShown {
        updateTags()
        shouldUpdateTagsOnShown = false
    }
    isCurrentlyShown = true
    wasAlreadyShown = true
}
override func viewDidDisappear(_ animated: Bool) {
    super.viewDidDisappear(animated)
    isCurrentlyShown = false
}

private func setupViews() {
    setupTokenField()
    view.backgroundColor = UIColor(named: "TABackground")
    setupNavigationBar()

    setupContainerView()
    setupTableView()
    setupKeyboard()
}
var testx: CGRect?
```

```
    var subtasksAddCellReference: SubtaskAddCell?
    private func setupKeyboard() {
        var previousHeight: CGFloat?
        keyboard
            //.toolbar(scrollView: collectionView)
            .on(event: .willChangeFrame) { [weak self] options in
                guard let self = self else { return }
                let height = options.endFrame.intersection(self.scrollView.convert(self.scrollView.bounds, to:
nil)).height
                self.testx = options.endFrame
                if previousHeight == height { return }
                previousHeight = height
                UIView.animate(withDuration: Constants.animationDefaultDuration) {
                    self.scrollView.contentInset = .init(top: 0, left: 0, bottom: height, right: 0)
                }
            }
            .on(event: .willHide) { [weak self] options in
                guard let self = self else { return }
                let height = options.endFrame.intersection(self.scrollView.convert(self.scrollView.bounds, to:
nil)).height
                self.testx = options.endFrame
                if previousHeight == height { return }
                previousHeight = height
                UIView.animate(withDuration: Constants.animationDefaultDuration) {
                    self.scrollView.contentInset = .init(top: 0, left: 0, bottom: height, right: 0)
                }
            }
            .start()
    }

    private func setupBindings() {
        checkboxh1.onSelected = { [weak self] in
            self?.viewModel.toggleDone()
        }
        tokenField.onPlusButtonClicked = { [weak self] in
            guard let self = self else { return }
            self.addTagsSelected(action: PopuptodoAction())
        }
    }
```

```
    private func setupTableView() {
        subtasksTable.register(SubtaskCell.self, forCellReuseIdentifier: SubtaskCell.reuseIdentifier)
        subtasksTable.register(SubtaskAddCell.self, forCellReuseIdentifier: SubtaskAddCell.reuseIdentifier)
        subtasksTable.delegate = self
        subtasksTable.separatorInset = .zero

        let dataSource = RxTableViewSectionedAnimatedDataSource<AnimSection<TaskDetailsVcVm.Model>>
{ [weak self] (data, tableView, indexPath, model) -> UITableViewCell in
            switch model {
            case .addSubtask:
                let cell = tableView.dequeueReusableCell(withIdentifier: SubtaskAddCell.reuseIdentifier, for:
indexPath) as! SubtaskAddCell
                cell.subtaskCreated = { name in
                    self?.viewModel.createSubtask(with: name)
                }
                if self?.isCurrentlyShown ?? false {

                    cell.becomeFirstResponder()
                }
                return cell
            case let .subtask(subtask):
                let cell = tableView.dequeueReusableCell(withIdentifier: SubtaskCell.reuseIdentifier, for: indexPath)
as! SubtaskCell
                cell.configure(name: subtask.name, isDone: subtask.isDone)
                cell.delegate = self
                cell.onSelected = {
                    self?.viewModel.toggleDoneSubtask(subtask: subtask, isDone: $0)
                }
                return cell
            }
        }
        var wasAlreadyLoaded = false
        viewModel.subtasksUpdate
            .do(onNext: { [weak self] itemsSections in
                guard let self = self else { return }
                let widthForLabel: CGFloat = UIScreen.main.bounds.width - (self.containerStackLeadingTrailing * 2
+ SubtaskCell.nameLabelLeadingTrailingSpace * 2)
                let label = UILabel()
```

```
            label.font = SubtaskCell.nameLabelFont
            label.numberOfLines = 0
            var totalHeight: CGFloat = 0
            for item in itemsSections[0].items {
               switch item {
               case .addSubtask:
                  totalHeight += SubtaskAddCell.height
               case let .subtask(subtask):
                  label.text = subtask.name
                  totalHeight += label.sizeThatFits(.init(width: widthForLabel, height: 1000)).height +
SubtaskCell.nameLabelTopBottomSpace * 2 + 0.5
               }
            }
            self.subtasksTable.snp.remakeConstraints { make in
               make.height.equalTo(totalHeight)
            }
            if wasAlreadyLoaded {
               self.layoutAnimate()
            }
            wasAlreadyLoaded = true
         })
         .bind(to: subtasksTable.rx.items(dataSource: dataSource))
         .disposed(by: bag)
   }

   private func subtaskCreated(with name: String) {
      self.viewModel.createSubtask(with: name)
   }

   private func setupViewModelBinding() {
      self.taskNameh1.text = viewModel.task.name
      self.setTaskDescription(viewModel.task.taskDescription)
      viewModel.shouldEnableTaskDescription = { [weak self] in
         self?.explicitlyEnableTaskDescription()
      }
      viewModel.taskObservable
         .subscribe(onNext: { [weak self] task in
            guard let self = self else { return }
            self.checkboxh1.configure(isChecked: task.isDone)
```

```
            self.checkboxh1.configure(priority: task.priority)
            self.updateLabels(taskDate: task.date)
        })
        .disposed(by: bag)
    viewModel.tagsObservable
        .subscribe(onNext: { [weak self] tags in
            guard let self = self else { return }
            if self.isCurrentlyShown || !self.wasAlreadyShown {
                self.updateTags()
            } else {
                self.shouldUpdateTagsOnShown = true
            }
        })
        .disposed(by: bag)
}
var __previousHeight: CGFloat?
func layoutAnimate() {
    print("layout subviews")
    setSpacings()
    if wasAlreadyShown {
        UIView.animate(withDuration: Constants.animationDefaultDuration) {
            self.view.layoutSubviews()
            self.scrollView.layoutSubviews()
            self.containerStack.layoutSubviews()
        }
    }
}

func setTaskDescription(_ taskDescription: String) {
    defer {
        layoutAnimate()
    }
    if !self.taskDescription.text.isEmpty {
        return
    }
    guard !taskDescription.isEmpty else {
        self.taskDescription.isHidden = true
        return
    }
```

```
        self.taskDescription.isHidden = false
        self.taskDescription.text = taskDescription
    }

    func explicitlyEnableTaskDescription() {
        self.taskDescription.isHidden = false
        layoutAnimate()
        taskDescription.becomeFirstResponder()
    }

    func updateTags() {
        defer {
            layoutAnimate()
        }
        guard !(viewModel.task.tags.isEmpty) else {
            self.tokenField.isHidden = true
            self.tokenField.removeAllTokens()
            return
        }
        self.tokenField.isHidden = false
        let old = tokenField.tokens as? [ResizingToken]
        let newTags = ModelFormatt.tagsSorted(tags: Array(viewModel.task.tags)).map { ResizingToken(title:
$0.name) }
        tokenField.deepdiff(old: old ?? [], new: newTags)
    }

    func updateLabels(taskDate: RlmTaskDate?) {
        if let date = taskDate?.date {
            dateDetailLabel.isHidden = false
            dateDetailLabel.configure(with: DateFormatter.str(from: date))
        } else {
            dateDetailLabel.isHidden = true
        }
        if let reminder = taskDate?.reminder {
            reminderDetailLabel.isHidden = false
            reminderDetailLabel.configure(with: reminder.description)
        } else {
            reminderDetailLabel.isHidden = true
        }
```

```
    if let repeatt = taskDate?.repeat {
        repeatDetailLabel.isHidden = false
        repeatDetailLabel.configure(with: repeatt.description)
    } else {
        repeatDetailLabel.isHidden = true
    }
    layoutAnimate()
}

let containerView: UIView = {
    let view = UIView()
    view.accessibilityIdentifier = "containerView"
    return view
}()

private let containerStack: UIStackView = {
    let stack = UIStackView(frame: .zero)
    stack.accessibilityIdentifier = "containerStack"
    stack.axis = .vertical
    return stack
}()

private let horizontal1: UIStackView = {
    let stack = UIStackView(frame: .zero)
    stack.axis = .horizontal
    stack.accessibilityIdentifier = "horizontal1"
    stack.spacing = 13
    stack.alignment = .top
    return stack
}()
private let checkboxh1: CheckboxView = {
    let view = CheckboxView()
    view.tint = .hex("#447BFE")
    view.accessibilityIdentifier = "checkboxh1"
    return view
}()

private lazy var checkboxh1Container: UIView = {
    let view = UIView()
```

```
        view.layout(checkboxh1).leading().trailing().centerY()
        view.snp.makeConstraints { make in
            make.height.equalTo(taskNameh1.textField.font?.lineHeight ?? 24)
        }
        return view
    }()
    private lazy var taskNameh1: MyGrowingTextView = {
        let taskLabel = MyGrowingTextView(placeholderText: FunnyTextProvider.shared.getFunText(),
scrollBehavior: .noScroll)
        taskLabel.textField.font = Fonts.heading2
        taskLabel.accessibilityIdentifier = "taskNameh1"
        taskLabel.placeholderAttrs = Attributes().font(Fonts.heading2).foreground(color: UIColor(named:
"TASubElement")!)
        taskLabel.textFieldAttrs = Attributes().font(Fonts.heading2).foreground(color: UIColor(named:
"TAHeading")!)
        taskLabel.growingTextFieldDelegate = self
        taskLabel.textField.inputAccessoryView = AccessoryView(onDone: { [weak taskLabel] in
            taskLabel?.endEditing(true)
        }, onHide: { [weak taskLabel] in
            taskLabel?.endEditing(true)
        })
        return taskLabel
    }()

    private lazy var taskDescription: MyGrowingTextView = {
        let description = MyGrowingTextView(placeholderText: "Enter description".localizable(), scrollBehavior:
.noScroll)
        description.accessibilityIdentifier = "taskDescription"
        let attributes: Attributes = Attributes().lineSpacing(5).foreground(color: UIColor(named:
"TASubElement")!).font(Fonts.text)
        description.placeholderAttrs = attributes
        description.textFieldAttrs = attributes
        description.growingTextFieldDelegate = self
        description.isNewSpaceAllowed = true
        description.onEnter = { }
        description.textField.inputAccessoryView = AccessoryView(onDone: { [weak description] in
            description?.endEditing(true)
        }, onHide: { [weak description] in
            description?.endEditing(true)
```

```
    })
    return description
}()

private let tokenField: ResizingTokenField = ResizingTokenField()

private let stackDateDetail: UIStackView = {
    let stack = UIStackView(frame: .zero)
    stack.accessibilityIdentifier = "stackDateDetail"
    stack.axis = .horizontal
    stack.spacing = 0
    stack.alignment = .center
    return stack
}()

private let dateDetailLabel: DateDetailLabel = {
    let view = DateDetailLabel()
    view.accessibilityIdentifier = "dateDetailLabel"
    view.setImage(image: UIImage(named: "alarm")?.resize(toWidth: 14))
    view.isHidden = true
    return view
}()

private let datesStackSeparator: UIView = {
    let view = UIView()
    view.accessibilityIdentifier = "datesStackSeparator"
    view.heightAnchor.constraint(equalToConstant: 6).isActive = true
    return view
}()
private let datesStackSeparator2: UIView = {
    let view = UIView()
    view.accessibilityIdentifier = "datesStackSeparator2"
    view.heightAnchor.constraint(equalToConstant: 6).isActive = true
    return view
}()

private let stackReminderRepeat: UIStackView = {
    let stack = UIStackView(frame: .zero)
    stack.axis = .horizontal
```

```
    stack.alignment = .leading
    stack.spacing = 6
    stack.layoutMargins = .init(top: 6, left: 0, bottom: 0, right: 0)
    stack.accessibilityIdentifier = "stackReminderRepeat"
    return stack
}()

private let reminderDetailLabel: DateDetailLabel = {
    let view = DateDetailLabel()
    view.setImage(image: UIImage(named: "bell")?.resize(toWidth: 16))
    view.accessibilityIdentifier = "reminderDetailLabel"
    view.isHidden = true
    return view
}()

private let repeatDetailLabel: DateDetailLabel = {
    let view = DateDetailLabel()
    view.setImage(image: UIImage(named: "repeat")?.resize(toWidth: 13))
    view.isHidden = true
    view.accessibilityIdentifier = "repeatDetailLabel"
    return view
}()
let scrollView: UIScrollView = {
    let view = UIScrollView()
    view.accessibilityIdentifier = "scrollView"
    view.scrollIndicatorInsets = .init(top: 8, left: 0, bottom: 8, right: 0)
    return view
}()

func setupTokenField() {
    tokenField.delegate = self
    tokenField.itemSpacing = 4
    tokenField.allowDeletionTags = false
    tokenField.hideLabel(animated: false)
    tokenField.font = Fonts.heading5
    tokenField.preferredTextFieldReturnKeyType = .done
    tokenField.heightConstraint?.isActive = false
    tokenField.contentInsets = .zero
    tokenField.isHidden = true
```

```
        tokenField.snp.makeConstraints { make in
            make.height.equalTo(tokenField.itemHeight)
        }
    }

    private func setupContainerView() {
        view.layout(containerView).leading(0).trailing(0).topSafe().bottom() { _, _ in .lessThanOrEqual }
        containerView.layout(scrollView).edges()

scrollView.layout(containerStack).leading(containerStackLeadingTrailing).trailing(containerStackLeadingTrailing).top(23)
        scrollView.contentLayoutGuide.heightAnchor.constraint(equalTo: containerStack.heightAnchor, constant: 23 + 23).isActive = true // 23 top + 23 bottom + ContentInset
        let containerHeight = containerView.heightAnchor.constraint(equalTo: scrollView.contentLayoutGuide.heightAnchor)
        containerHeight.priority = .init(749)
        containerHeight.isActive = true
        scrollView.contentLayoutGuide.widthAnchor.constraint(equalTo: scrollView.frameLayoutGuide.widthAnchor).isActive = true
        containerStack.isUserInteractionEnabled = true
        horizontal1.isUserInteractionEnabled = true
        horizontal1.addArrangedSubview(checkboxh1Container)
        horizontal1.addArrangedSubview(taskNameh1)
        containerStack.addArrangedSubview(horizontal1)
        let taskDescriptionHeight = self.taskDescription.heightAnchor.constraint(equalToConstant: self.taskDescription.textField.font?.lineHeight ?? 20)
        taskDescriptionHeight.isActive = true
        taskDescription.shouldSetHeight = { [weak self] in
            taskDescriptionHeight.constant = $0
            self?.layoutAnimate()
        }

        let taskNameHeight = taskNameh1.heightAnchor.constraint(equalToConstant: 20)
        taskNameHeight.isActive = true
        taskNameh1.shouldSetHeight = { [weak self] newHeight in
            taskNameHeight.constant = newHeight
            self?.layoutAnimate()
        }
```

```
        containerStack.addArrangedSubview(taskDescription)
        containerStack.addArrangedSubview(tokenField)
        containerStack.addArrangedSubview(subtasksTable)
        stackDateDetail.addArrangedSubview(dateDetailLabel)
        stackDateDetail.addArrangedSubview(UIView()) // empty view

        if !Constants.displayVersion2 {
            containerStack.addArrangedSubview(stackDateDetail)
            stackReminderRepeat.addArrangedSubview(reminderDetailLabel)
            stackReminderRepeat.addArrangedSubview(repeatDetailLabel)
            stackReminderRepeat.addArrangedSubview(UIView()) // empty view
            containerStack.addArrangedSubview(datesStackSeparator) // empty view
            containerStack.addArrangedSubview(stackReminderRepeat)
        } else {
            containerStack.addArrangedSubview(stackDateDetail)
            containerStack.addArrangedSubview(datesStackSeparator)
            let reminderDetailLabelStack = UIStackView(arrangedSubviews: [reminderDetailLabel, UIView()])
            reminderDetailLabelStack.axis = .horizontal
            reminderDetailLabelStack.alignment = .leading
            reminderDetailLabelStack.accessibilityIdentifier = "reminderDetailLabelStack"

            containerStack.addArrangedSubview(reminderDetailLabelStack)
            containerStack.addArrangedSubview(datesStackSeparator2)
            let repeatDetailLabelStack = UIStackView(arrangedSubviews: [repeatDetailLabel, UIView()])
            repeatDetailLabelStack.axis = .horizontal
            repeatDetailLabelStack.alignment = .leading
            repeatDetailLabelStack.accessibilityIdentifier = "reminderDetailLabelStack"
            containerStack.addArrangedSubview(repeatDetailLabelStack)

        }
        [dateDetailLabel, repeatDetailLabel, reminderDetailLabel].forEach { $0.addTarget(self, action:
#selector(dateDetailLabelsClicked), for: .touchUpInside) }
    }

    @objc private func dateDetailLabelsClicked() {
        openCalendarVc()
    }

    func setSpacings() {
```

```
        let task = viewModel.task
        let spacingAfterHorizontal1 = !taskDescription.isHidden || !task.tags.isEmpty || !task.subtask.isEmpty ||
task.date?.date != nil || task.date?.reminder != nil || task.date?.repeat != nil ||
viewModel.explicitAddSubtaskEnabled
        containerStack.setCustomSpacing(spacingAfterHorizontal1 ? 32 : 0, after: horizontal1)
        let spacingAfterTaskDescription = !task.tags.isEmpty || !task.subtask.isEmpty || task.date?.date != nil ||
task.date?.reminder != nil || task.date?.repeat != nil
        containerStack.setCustomSpacing(spacingAfterTaskDescription ? 32 : 0, after: taskDescription)
        let spacingAfterSubtasksTable = task.date?.date != nil || task.date?.reminder != nil || task.date?.repeat != nil
        containerStack.setCustomSpacing(spacingAfterSubtasksTable ? 21.5 : 0, after: subtasksTable)
        let datesSeparatorVisible = task.date?.date != nil && (task.date?.reminder != nil || task.date?.repeat != nil)
        datesStackSeparator.isHidden = !datesSeparatorVisible
        if Constants.displayVersion2 {
            let datesSeparatorVisible2 = task.date?.reminder != nil && task.date?.repeat != nil
            datesStackSeparator2.isHidden = !datesSeparatorVisible2
        }
    }

    private func setupNavigationBar() {
        applySharedNavigationBarAppearance()
        navigationItem.rightBarButtonItem = actionsButton
    }

    var popMenuVc: PopMenuViewController?
    // MARK: - POPUP
    @objc private func actionsButtonClicked() {
        var actions: [PopuptodoAction] = []
        if viewModel.subtasksModels[0].items.isEmpty {
            actions.append(PopuptodoAction(title: "Add Checklist".localizable(), image: UIImage(named: "list-
check"), didSelect: { [weak self] action in
                self?.addChecklistSelected(action: action)
            }))
        }
        if tokenField.isHidden {
            actions.append(PopuptodoAction(title: "Add Tags".localizable(), image: UIImage(named: "tag"),
didSelect: { [weak self] action in
                self?.addTagsSelected(action: action)
            }))
        }
```

```
        if taskDescription.isHidden {
            actions.append(PopuptodoAction(title: "Add Description".localizable(), image: UIImage(named:
"taskdescription"), didSelect: { [weak self] action in
                self?.addDescriptionSelected(action: action)
            }))
        }
        actions.append(contentsOf: [
            PopuptodoAction(title: "Select Priority".localizable(), image: UIImage(named:
"flag")?.withRenderingMode(.alwaysTemplate), didSelect: { [weak self] action in
                guard KeychainWrapper.shared.isPremium ||
RealmProvider.main.realm.objects(RlmTask.self).filter({ $0.priority != .none }).count <=
Constants.maximumPriorities || self?.viewModel.task.priority != Priority.none else {
                    let premiumFeaturesVc = PremiumFeaturesVc(notification: .prioritiesLimit)
                    self?.dismiss(animated: true, completion: {
                        self?.present(premiumFeaturesVc, animated: true, completion: nil)
                    })
                    return
                }
                self?.selectPrioritySelected(action: action)
            }),
            PopuptodoAction(title: self.viewModel.task.date?.date != nil ? "Edit Date".localizable() : "Add
Date".localizable(), image: UIImage(named: "calendar-plus"), didSelect: { [weak self] action in
                self?.addCalendarSelected(action: action)
            }),
            PopuptodoAction(title: "Delete To-Do".localizable(), image: UIImage(named: "trash"), didSelect: {
[weak self] action in
                self?.deleteTodoSelected(action: action)
            }),
        ])
        PopMenuAppearance.appCustomizeActions(actions: actions)
        actions.first(where: { $0.title == "Select Priority".localizable() })?.iconWidthHeight = 24
        actions.first(where: { $0.title == "Select Priority".localizable() })?.imageTintColor =
UIColor.hex("#00CE15")
        let popMenu = PopMenuViewController(sourceView: actionsButton, actions: actions)
        popMenu.shouldDismissOnSelection = false
        popMenu.appearance = .appAppearance
        popMenuVc = popMenu
        present(popMenu, animated: true)
    }
```

```
    func addDescriptionSelected(action: PopMenuAction) {
        dismiss(animated: true, completion: nil)
        viewModel.addEmptyDescription()
    }

    func addChecklistSelected(action: PopMenuAction) {
        dismiss(animated: true, completion: nil)
        viewModel.explicitlyEnableTableView()
    }
    func addTagsSelected(action: PopMenuAction) {
        dismiss(animated: true, completion: nil)
        router.openAllTags(mode: .selection(selected: Array(viewModel.task.tags), { [weak self] tags in
            self?.viewModel.addTags(tags)
        }))
    }

    func selectPrioritySelected(action: PopMenuAction) {
        let actions: [PopuptodoAction] = [
            PopuptodoAction(title: "High Priority".localizable(), image: UIImage(named:
"flag")?.withRenderingMode(.alwaysTemplate), didSelect: { [weak self] _ in
self?.viewModel.selectHighPriority() }),
            PopuptodoAction(title: "Medium Priority".localizable(), image: UIImage(named:
"flag")?.withRenderingMode(.alwaysTemplate), didSelect: { [weak self] _ in
self?.viewModel.selectMediumPriority() }),
            PopuptodoAction(title: "Low Priority".localizable(), image: UIImage(named:
"flag")?.withRenderingMode(.alwaysTemplate), didSelect: { [weak self] _ in
self?.viewModel.selectLowPriority() }),
            PopuptodoAction(title: "No Priority".localizable(), image: UIImage(named:
"flag")?.withRenderingMode(.alwaysTemplate), didSelect: { [weak self] _ in
self?.viewModel.selectNonePriority() })
        ]
        actions[0].imageTintColor = .hex("#EF4439")
        actions[1].imageTintColor = .hex("#FF9900")
        actions[2].imageTintColor = .hex("#447BFE")
        actions[3].imageTintColor = UIColor(named: "TASubElement")!
        PopMenuAppearance.appCustomizeActions(actions: actions, iconWidth: 24)
        let popMenu = PopMenuViewController(sourceView: action.view, actions: actions)
        popMenu.appearance = .appAppearance
```

```
        popMenuVc?.present(popMenu, animated: true)
    }
    func addCalendarSelected(action: PopMenuAction) {
        openCalendarVc()
    }

    func openCalendarVc() {
        let taskDate = viewModel.task.date?.freeze()
        guard KeychainWrapper.shared.isPremium || viewModel.task.date != nil ||
RealmProvider.main.realm.objects(RlmTask.self).filter({ $0.date?.date != nil }).count <=
Constants.maximumDatesToTask else {
            dismiss(animated: true) { [weak self] in
                let premiumFeaturesVc = PremiumFeaturesVc(notification: .dateToTaskLimit)
                self?.dismiss(animated: true, completion: {
                    self?.present(premiumFeaturesVc, animated: true, completion: nil)
                })
            }
            return
        }
        dismiss(animated: true) { [weak self] in
            Notifications.shared.requestAuthorization { authorization in
                DispatchQueue.main.async {
                    switch authorization {
                    case .authorized:
                        let calendarVc = CalendarVc(viewModel: .init(reminder: taskDate?.reminder, repeat:
taskDate?.repeat, date: taskDate?.date)) { (newDate, newReminder, newRepeat) in
                            self?.viewModel.newDate(date: newDate, reminder: newReminder, repeatt: newRepeat)
                        }
                        self?.fpc.configure(vc: calendarVc, scrollViews: [calendarVc.scrollView])
                        if let fpc = self?.fpc.fpc {
                            self?.present(fpc, animated: true)
                        }
                    case .denied:
                        print("Denied")
                    case .deniedPreviously:
                        self?.showAlertToOpenSettings()
                    }
                }
            }
```

```
    }
  }

  private func showAlertToOpenSettings() {
    let alertController = UIAlertController(title: "Notifications are disabled".localizable(), message: "You
disabled notification for this app, so we cannot set up notifications".localizable(), preferredStyle: .alert)
    if let settingsUrl = URL(string: UIApplication.openSettingsURLString) {
      guard UIApplication.shared.canOpenURL(settingsUrl) else { return }
      let settingsAction = UIAlertAction(title: "Settings".localizable(), style: .default) { _ -> Void in
        UIApplication.shared.open(settingsUrl, completionHandler: { (success) in
          print("Settings opened: \(success)") // Prints true
        })
      }
      let cancelAction = UIAlertAction(title: "Cancel".localizable(), style: .default, handler: nil)
      alertController.addAction(cancelAction)
      alertController.addAction(settingsAction)
    } else {
      let cancelAction = UIAlertAction(title: "Close".localizable(), style: .default, handler: nil)
      alertController.addAction(cancelAction)
    }

    present(alertController, animated: true, completion: nil)
  }
  func deleteTodoSelected(action: PopMenuAction) {
    dismiss(animated: true, completion: { [weak self] in
      guard let task = self?.viewModel.task,
          let projectId = task.project.first?.id,
          task.realm != nil else { return }
      self?.router.navigationController.popViewController(animated: true)
      self?.viewModel.deleteItselfInRealm()
    })
  }

  func handleSwipeActionDeletion(action: SwipeAction, indexPath: IndexPath) {
    let subtaskmodel = viewModel.subtasksModels[0].items[indexPath.row]
    switch subtaskmodel {
    case let .subtask(subtask):
      viewModel.deleteSubtask(subtask: subtask)
    default: return
```

```
        }
    }
}
```