**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформаційних систем та технологій**

До захисту допущено:

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_ Олександр РОЛІК

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2023 р.

**Дипломний проєкт**

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Інформаційне забезпечення робототехнічних систем»**

**спеціальності 126 «Інформаційні системи та технології»**

**на тему: «Інтерактивно-компонентний планувальник організації часу людини»**

Виконав:

студент IV курсу, групи ІК-91

Остапченко Дмитро Олексійович \_\_\_\_\_\_\_\_\_\_

Керівник:

Доцент кафедри ІСТ, канд.техн.наук, доцент

Солдатова М. О. \_\_\_\_\_\_\_\_\_\_

Рецензент:

Зав. Кафедри ІПІ, доктор техн.наук, доцент

Жаріков Е. В. \_\_\_\_\_\_\_\_\_\_

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_

Київ – 2023 року

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформаційних систем та технологій**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційне забезпечення робототехнічних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_\_\_ Олександр РОЛІК

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2023 р.

**ЗАВДАННЯ**

**на дипломний проєкт студенту**

**Остапченку Дмитру Олексійовичу**

1. Тема проєкту «Інтерактивно-компонентний планувальник організації часу людини», керівник проєкту Солдатова Марія Олександрівна, кандидат технічних наук, доцент, затверджені наказом по університету від «11» травня 2021 р. № 1139-с

2. Термін подання студентом проєкту: 12.06.2023р

3. Вихідні дані до проєкту: потік завдань виконуваних колективом людей, вимоги та терміни виконання завдань, технічні засоби планування організації часу, інструментальні програмні засоби розробки на платформах iOS, Telegram Bot, MySQL

4. Зміст пояснювальної записки: назва розділ 1, розділ 2, розділ 3, розділ 4

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо)

6. Консультанти розділів проєкту\*

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада  консультанта | Підпис, дата | |
| завдання  видав | завдання прийняв |
|  |  |  |  |

7. Дата видачі завдання

Календарний план

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів виконання  дипломного проєкту | Термін виконання  етапів проєкту | Примітка |
| 1 | Аналіз проблем створення програмних систем організації часу людини та існуючих рішень | 24.04.2023р |  |
| 2 | Формулювання завдання на проектування та вимог до системи | 01.05.2023р |  |
| 3 | Розробка архітектури та алгоритмів роботи системи | 5.05.2023р |  |
| 4 | Аналіз та вибір інструментальних засобів проектування | 15.05.2023р |  |
| 5 | Розробка програмних модулів системи | 29.05.2023р |  |
| 6 | Тестування програмних модулів системи | 05.06.2023р |  |
| 7 | Розробка технічною документації | 12.06.2023р |  |

Студент Дмитро ОСТАПЧЕНКО

Керівник Марія СОЛДАТОВА

**АНОТАЦІЯ**

Остапченко Д.О. Інтерактивно-компонентний планувальник організації часу людини. КПІ ім. Ігоря Сікорського, Київ, 2023.

Проєкт містить 63 с. тексту, 7 рисунків, 3 таблиці, посилання на 22 літературні джерела, додатки та 5 конструкторських документів.

Ключові слова: організація часу, планувальник завдань, інформаційне нагадування, пакет програм, застосунок обміну повідомленнями, база даних

Об’єктом розробки є система планування та організації часу.

Мета розробки – підвищення якості і оперативності організації взаємодії колективу людей в інтерактивному середовищі за рахунок автоматизації процесів організації часу і створення програмного інтерактивно-компонентного планувальника завдань.

У дипломному проєкті розроблено комплекс програмних продуктів, які пов’язані між собою і утворюють повну систему, а саме: iOS застосунок, що дозволяє отримувати пакети завдань користувачів та створювати власні; Telegram Bot, що призначений для надсилання повідомлень та інформацію про завдання користувачу; компонентну систему, що дає змогу гнучку конструювати завдання будь-якої, складності може бути розширена і впроваджена в інші програмні засоби; сервіс обслуговування з використанням бази даних, що реалізує авторизацію користувача та зберігання даних пакетів завдань на сервері.

Отримані результати розробки програмного комплексу можуть використовуватись при плануванні робочих завдань як для окремої людини (користувача) так і для груп (колективів) робітників.

**SUMMARY**

Ostapchenko D.O. Interactive component planner for organizing a person's time. KPI named after Igor Sikorskyi, Kyiv, 2023.

The project contains 63 pages. text, 7 figures, 3 tables, references to 22 literary sources, appendices and 5 design documents.

Keywords: time management, task scheduler, information reminder, software package, messaging application, database

The object of development is a system of planning and organizing time.

The purpose of the development is to improve the quality and efficiency of organizing the interaction of a group of people in an interactive environment due to the automation of time management processes and the creation of a software interactive and component task planner.

The diploma project developed a set of software products that are interconnected and form a complete system, namely: an iOS application that allows you to receive packages of user tasks and create your own; Telegram Bot, designed to send messages and task information to the user; the component system, which enables flexible construction of tasks of any complexity, can be expanded and implemented in other software tools; maintenance service using a database that implements user authorization and data storage of task packages on the server.

The obtained results of the development of the software complex can be used in the planning of work tasks both for an individual person (user) and for groups (teams) of workers.

Зм.

Аркуш

№ докум.

Підпис

Дата

Аркуш

1

###### І К91.010БАК.003 ТП

Розроб.

Іванченко І.І.

Керівн.

Петренко П.П.

Затв.

###### Назва проєкту.

Відомість проєкту

Літ.

Аркушів

1

КПІ ім. Ігоря Сікорського

Група ІК-91

Номер рядка

Формат

Позначення

Найменування

Кільк. аркушів

Номер екзем.

Примітка

1

Документація загальна

2

3

Знову розроблена

4

5

А4

ІК91.010БАК.003 ПЗ

Пояснювальна записка

65

6

А3

ІК91.010БАК.003 Э1

Назва проєкту. Схема

1

7

структурна

8

А3

ІК91.010БАК.003 Э2

Назва проєкту. Схема

9

функціональна

1

10

Назва проєкту. Діаграма

11

використання

12

Назва проєкту. Діаграма

13

А3

ІК91.010БАК.003 Д1

діяльності

1

14

15

16

17

А3

ІК91.010БАК.003 Д2

1

18

19

20

21

22

23

24

25

26

27

28

т

**Пояснювальна записка**

**до дипломного проєкту**

**на тему: «** **Інтерактивно-компонентний планувальник організації часу людини»**

Київ – 2023 року

ЗМІСТ

[ВСТУП 11](#_Toc137417447)

[1 АНАЛІЗ ПРОБЛЕМ СТВОРЕННЯ ПРОГРАМНИХ СИСТЕМ ОРГАНІЗАЦІЇ ЛЮДИНИ 13](#_Toc137417448)

[1.1 Огляд предметної області автоматизації процесів організації часу людини 13](#_Toc137417449)

[1.1.1 Сфера 13](#_Toc137417450)

[1.1.2 Ідеологія проекта 14](#_Toc137417451)

[1.1.3 Які проблеми вирішує проект 14](#_Toc137417452)

[1.2 Аналіз існуючих рішень з організації програмних систем 15](#_Toc137417453)

[1.2.1 Огляд Structured 16](#_Toc137417454)

[1.2.2 Огляд Routinery 18](#_Toc137417455)

[1.3 Проблеми створення і організації програмних систем 20](#_Toc137417456)

[1.4 Постановка задачі створення інтерактивного-компонентного планувальника завдань 21](#_Toc137417457)

[Висновок до розділу 21](#_Toc137417458)

[2 АРХІТЕКТУРА СИСТЕМИ ІНТЕРАКТИВНОГО-КОМПОНЕНТНОГО ПЛАНУВАННЯ ЗАВДАНЬ 22](#_Toc137417459)

[2.1 Формалізоване подання процесу планування завдань 22](#_Toc137417460)

[2.1.1 Система компонентів 22](#_Toc137417461)

[2.1.2 Алгоритм роботи з пакетами завдань 22](#_Toc137417462)

[2.1.3 Реалізації компонентів 23](#_Toc137417463)

[2.1.4 Обґрунтування методу розв’язування 24](#_Toc137417464)

[2.2 Архітектура 24](#_Toc137417465)

[2.3 Постановка задачі і функцій при створенні інтерактивного-компонентного планувальника завдань 25](#_Toc137417466)

[2.3.1 Аналіз вимог 25](#_Toc137417467)

[2.3.2 Функціональні вимоги 30](#_Toc137417468)

[2.4 Архітектура 32](#_Toc137417469)

[2.5 Вимоги до розробки компонентного планувальника завдань 32](#_Toc137417470)

[2.5.1 Вимоги до коду 32](#_Toc137417471)

[2.5.2 Системні вимоги 33](#_Toc137417472)

[3 РЕАЛІЗАЦІЯ ІНТЕРАКТИВНОГО-КОМПОНЕНТНОГО ПЛАНУВАННЯ ЗАВДАНЬ 34](#_Toc137417473)

[3.1 Засоби розробки 34](#_Toc137417474)

[3.1.1 Технології iOS застосуноку 35](#_Toc137417475)

[3.1.2 Система компонентів 41](#_Toc137417476)

[3.1.3 Серверна програма і БД 42](#_Toc137417477)

[3.1.4 Система повідомлень 43](#_Toc137417478)

[*3.2* *Реалізація iOS застосунку* 46](#_Toc137417479)

[3.2.1 Організація окремого модуля 46](#_Toc137417480)

[3.2.2 Шари програми 47](#_Toc137417481)

[**3.2.3** Навігація 48](#_Toc137417482)

[3.2.4 Впровадження залежностей у модулі 50](#_Toc137417483)

[3.2.5 Інші частини системи 51](#_Toc137417484)

[*3.3* Аналіз допоміжних програмних засобів та засобів розробки 52](#_Toc137417485)

[Висновок до розділу 52](#_Toc137417486)

[4 ВПРОВАДЖЕННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНТЕРАКТИВНО-КОМПОНЕНТНОГО ПЛАНУВАЛЬНИКА ЗАВДАНЬ 53](#_Toc137417487)

[4.1 Тестування 53](#_Toc137417488)

[4.2 Розгортання програмного забезпечення 59](#_Toc137417489)

[4.2.1 iOS застосунок 59](#_Toc137417490)

[4.2.2 Telegram Bot 61](#_Toc137417491)

[4.2.3 Back-end програма 62](#_Toc137417492)

[4.3 Підтримка програмного забезпечення 63](#_Toc137417493)

[4.3.1 iOS застосунок 63](#_Toc137417494)

[4.3.2 Telegram Bot 64](#_Toc137417495)

[4.3.3 Back-end програма 64](#_Toc137417496)

[4.4 Висновок до розділу 64](#_Toc137417497)

[ВИСНОВОК 65](#_Toc137417498)

[ПЕРЕЛІК ПОСИЛАНЬ 66](#_Toc137417499)

# ****ВСТУП****

**Актуальність проблеми**. В сучасному світі, де ритм життя набуває все більшої швидкості, організація часу стає критично важливим фактором для досягнення успіху та збереження рівноваги між роботою, особистими справами та відпочинком. Ефективне управління часом дозволяє максимально використовувати потенціал і досягати поставлених цілей, як для окремої людини, так і для робочих груп (або колективів).

Кожен день людина має два типи завдань для виконання – які надходять протягом доби та ті, які заплановані або повторюються. Якщо перші вирішуються по мірі надходження, то другий тип завдань потрібно виконати за певний час.

Одна з великих проблем людини – це обмежені можливості при утриманні актуального стану (інформації) для кількох складних завдань та часу їх виконання в пам’яті. Протягом робочого дня утримати всю необхідну інформацію про завдання в пам’яті, коли виконати та завершити виконання - достатньо складно, як для окремої людини, так і для колективу робітників (груп розробників), тощо.

Обмеженість пам'яті створює перешкоди в ефективному управлінні робітниками та утриманні необхідних завдань й інформації про завдання у свідомості окремого виконавця (або персонально для людини). Робітники (як окрема людина) та колективи стикаються з ситуаціями, коли частина завдань не виконується або виконується частково та не вчасно. Не ефективне зосередження на другорядних задачах, використання часу більше на окремі завдання призводить до неефективної роботи та невиконання окремих завдань, як частково, так і повністю.

**Об’єктом дослідження** є процеси планування і організації часу людини або колективу при виконанні комплексу взаємопов’язаних завдань або задач проектів. Неправильне розподілення задач робітника або колективу негативно впливає на ~~нашу~~ продуктивність праці та результати роботи в цілому.

**Предмет дослідження**. Для вирішення проблем з розподіленням робочого часу групи (колективу) або часу окремої людини (робітника), створюється проект з організації часу, який спрямований на надання необхідних інструментів та підходів для ефективного управління завданнями, які можуть повторюватись та виконуватись в певний час або з певним проміжком часу (за розкладом).

Важливо підкреслити, розробка проекту не має на меті надати універсальний підхід до організації часу, але ставить за завданням створити інструменти, які дозволять кожному користувачу (або колективам) індивідуалізувати свій підхід до виконання задач залежно від особистих потреб, пріоритетів та стилю роботи (або життя).

**Мета і завдання дослідження**. Підставою для створення проекту стала актуальна проблема відсутності гнучких інструментів організації часу людини на ринку програмних засобів. Головною метою проекту є створення простого та гнучкого програмного засобу для організації роботи (та відпочинку) так, як потребує людина (або управлінець колективом), накопичувати персональний досвід та мати різні можливості як передавати інформацію іншим - в якості завдань або типовим розпорядком дня (шаблоном).

Для досягнення цієї мети, в проекті мають бути виконані наступні завдання:

1. Проведення аналізу предметної області процесів організації часу людини і створення спеціалізованих програмних систем з визначенням проблем у їх застосуванні і напрямків розвитку.
2. Створення концепції інтерактивно-компонентної системи з інтуїтивно зрозумілим інтерфейсом користувача, який дозволить легко планувати свій робочий час та розподіляти завдання.
3. Розробка архітектури та створення функціоналу програмного середовища інтерактивно-компонентної системи, який дозволить користувачам зберігати свій досвід (розпорядок дня) та накопичувати влучні поради, завдання та рекомендації в особистому профілі.
4. Розробка механізму зворотного зв'язку та підтримки, щоб користувачі (або групи) могли обмінюватись інформацією, результатами роботи та досягнутими успіхами.
5. Проведення тестування і експериментального випробування функціоналу програмного середовища інтерактивно-компонентної системи для визначення ефективності використання часу та успішності виконання завдань користувача.

**Практичне значення одержаних результатів**. Проект призначений для вирішення проблеми неефективного використання часу і надання користувачам інструмента, який допоможе раціонально розподіляти час на виконання завдань, отримувати нагадування, повідомлення та іншу необхідну інформацію про роботу або відпочинок. Обмін інформації між користувачами, як керівниками робочих груп або окремими робітниками має призвести до більш швидкого та якісного виконання поставлених задач та розподілення часу між роботою та відпочинком.

Дипломний проєкт складається з наступних розділів: вступ, основні розділи, висновки, список використаних джерел із ... найменувань, ... додатків. Графічна частина включає … креслеників формату … . Загальний обсяг … сторінок

# АНАЛІЗ ПРОБЛЕМ СТВОРЕННЯ ПРОГРАМНИХ СИСТЕМ ОРГАНІЗАЦІЇ ЧАСУ ЛЮДИНИ

## Огляд предметної області автоматизації процесів організації часу людини

Висока інтенсивність процесів організації і проведення різних видів діяльності людини або в виробничій або в побутовій сфері вимагає контролю за відтворенням точної послідовності подій та обробки значного обсягу інформації. Саме тому знаходження ефективних стратегій для керування і обробки інформації стає насущною задачею в області автоматизації процесів організації часу людини.

На даний час існує багато напрямків та програмних систем з організації і планування часу людини, починаючи від простих “списків до виконання" (To-do lists) й закінчуючи складними планувальними системами, такими як Microsoft Project. Основна різниця між ними полягає у масштабі та цілях, які системи намагаються досягти, орієнтуючись на різні фокусні групи людей або колективи. Отже, розвивається дві моделі створення проектів з організації часу людини: спрямовані на бізнес - модель «бізнес-до-бізнесу» (БТБ) і на кінцевого користувача - модель «бізнес-до-споживача» (БТС), тобто "від продукту до клієнта" (Project to Client)).

Проекти, спрямовані на бізнес і організацію бізнес-процесів, створюють свої продукти у вигляді кількох застосунків чи веб-сервісів з навантаженим інтерфейсом користувача та великою кількістю функцій для планування, аналізу ефективності, тощо. Існують окремі випадки програмного забезпечення, з універсальною вимогою для настільних або веб-планувальників з моделлю БТБ (бізнес-до-бізнесу) з великою кількістю функцій, які теж важко використовувати на малих екранах.

Проекти, спрямовані на користувачів, що мають на меті організацію часу для окремої особи (робітника), мають інші підходи. Інтерфейс повинен бути мінімізованим і зрозумілим, оскільки для моделі БТС (бізнес-до-споживача) кожен клієнт є унікальним. Такі системи використовуються "на ходу", тому мобільний телефон чи смарт-годинник мають велике значення для їхнього успішного функціонування.

Прикладами означених проектів є Atlassian Jira та Microsoft Project спрямовані на організацію бізнес-процесів у компаніях, в той час як Microsoft Anydo та Structured націлені на сегмент "від продукту до клієнта" (Product to Customer) - продаж програмних засобів для клієнтів, які зацікавлені у кращому управлінні своїм часом.

В будь якому випадку, таке організаційне програмне забезпечення виконує алгоритмічну роботу з даними та створює умови для ефективного сприйняття завдань та управління часом. Незалежно від цілей і масштабів, системи орієнтуються на найменший елемент обробки подій - завдання.

## Аналіз існуючих рішень з організації програмних систем

1.2.1 Аналіз підходів до організації програмних систем планування і організації часу людини

На ринку офісних застосувань і пакетів програм (в офіційних магазинах розробників) існує велика кількість програмного забезпечення для організації часу. Це означає, що конкуренція серед розробників є дуже високою, і для того, щоб отримати перевагу у користувачів, необхідно створити та запропонувати нові ефективні методи взаємодії та інтерфейси для користувачів.

Деякі засоби програмного забезпечення пропонують інтуїтивно зрозумілий та простий інтерфейс користувача, що дозволяє швидко оволодіти програмою і почати організовувати свій час без складнощів. Інші пропонують інноваційні методи планування, такі як графіки Ганта або “канбан-дошки", які дають змогу візуально відстежувати прогрес виконання та керувати завданнями.

Деякі програми акцентуються на синхронізації та доступності даних. та пропонують синхронізацію розкладів та завдань між різними пристроями та платформами, такими як комп'ютери, смартфони та планшети, щоб мати постійний доступ до оновленої інформації.

Деякі інші програми спеціалізуються на окремих групах користувачів або на виконання специфічних завдань. Наприклад, є програмні засоби, призначені для менеджменту проектів, нагадувань про медичні препарати, отримання часу роботи та багато інших.

Окрім цього, оновлення та підтримка такого програмного забезпечення є одним із важливих аспектів успішності. Розробники надають регулярні оновлення з новими функціями, виправленням помилок та забезпечують гарантовану підтримку користувачів, щоб забезпечити якість та сучасність програмного продукту.

В цілому на даний час утворилося два найпопулярніших підходи (пакети) створення додатків з організації часу на ринку програмного забезпечення, порівняльне оцінювання яких доцільно провести на підставі аналізу таких основних характеристик, як підтримувані пристрої, цілі та ідеології побудови, простоту керування процесами.

1.2.2. Огляд та оцінювання пакету Structured

Пакет Structured - це програмне забезпечення щоденного візуального планувальника, який поєднує календар і список справ для надання можливість планування завдань, відстеження прогресу їх виконання протягом дня та позначення їх як виконані, коли завдання завершені.

Сервіс пакету доступний лише на пристроях від компанії Apple. Може використовуватись на iPhone, iPad та на пристроях, що працюють під управлінням операційної системи macOS. Це означає, що користувачі, які володіють пристроями Apple, можуть використовувати функціональність, яку надає пакет Structured для планування та відстеження своїх завдань та розкладу.

Пакет Structured - це застосунок, який пропонує ряд корисних функцій для організації часу та завдань. Основний функціонал додатку Structured включає:

1. Планування завдань: Застосунок дозволяє вам створювати та планувати завдання на календарі. Надається можливість встановлювати терміни виконання, пріоритети та тривалість кожного завдання.
2. Список справ: Structured надає можливість створювати список справ, який включає окремі завдання або другорядні завдання. Ви можете легко організовувати свої завдання за категоріями або проектами.
3. Візуальне планування: За допомогою Structured є можливість використовувати візуальний планувальник для відстеження розкладу та завдань. Завдання представляються на календарі з можливістю переміщати за допомогою перетягування, змінюючи дату та час.
4. Нагадування та сповіщення: Structured дозволяє встановлювати нагадування про наближення термінів виконання завдань або про важливі події. Отримування сповіщення про важливе завдання.
5. Відстеження прогресу: Застосунок надає можливість відстежувати прогрес виконання завдань. Надається можливість позначати завдання як виконані, встановлювати статуси або використовувати вбудовані мітки для класифікації завдань.
6. Синхронізація даних: Structured забезпечує синхронізацію даних між пристроями, що дозволяє отримувати доступ до завдань та розкладу з різних пристроїв, таких як смартфони або планшети.
7. Функції програмного забезпечення дозволяють більш ефективно організувати час, планувати завдання та відстежувати виконання, забезпечуючи більшу продуктивність та організованість роботи.

Сервіс пакету Structured, на жаль, не має базового функціоналу для організації складних (наприклад, повторюваних) завдань людини, працююча версія програмного забезпечення призначена для планування виконання простих задач. Бізнес схема програми та сервісу в цілому, не може використовуватись для організації робочих процесів для груп користувачів або робочих колективів.

Сервіс програми зосереджений переважно на плануванні завдань та відстеженні прогресу. Це може бути обмеженням для тих користувачів, хто шукає інструмент, який допоможе організувати та регулярно повторювати рутинні дії і ділові звички.

1.2.3. Огляд та оцінювання пакету Routinery

Пакет Routinery - це додаток для організації завдань, який надає широкий функціонал для планування, відстеження та керування складними задачами. Основні функції Routinery включають:

1. Створення задачі: користувач створює власні задачі залежно від потреб і пріоритетів. Наприклад, задачі для ранкових нарад, щоденного тренування, тестування, тощо.
2. Гнучке планування: програмне забезпечення Routinery дозволяє гнучко планувати свої задачі на різні дні тижня або на конкретні дати. Надається можливість встановити повторюваність задачі щоденно, щотижня або вибрати приватний графік.
3. Нагадування і сповіщення: додаток Routinery надсилає нагадування та сповіщення, щоб нагадати користувачеві про важливі завдання. Користувач отримує повідомлення на смартфоні або інших пристроях.
4. Відстеження прогресу: програмне забезпечення Routinery дозволяє відстежувати прогрес у виконанні завдань. Надається можливість позначати завдання як виконані або відмічати їх у разі затримки. Функціонал надає можливість спостерігати за продуктивністю та дотримання графіку виконання задач.
5. Спільний доступ: додаток Routinery дозволяє спільний доступ до задач між користувачами. Надається можливість створювати групи завдань та додавати учасників, що спільно виконують їх. Функціонал надає можливість обмежено використовуватись як програмне забезпечення для робочих груп (колективів).
6. Статистика і звіти: програмне забезпечення Routinery надає статистику та звіти щодо задачі. Надається можливість: переглядати час (проведений на кожному завданні), аналіз продуктивності та отримувати дані виконання задачі.
7. Інтеграція: додаток Routinery надає можливість налаштовувати і персоналізувати завдання згідно з вимогами користувача. Надається можливість додавати зображення, опис, кольори та інші елементи, що допомагають налагодити пакет завдання. Routinery має можливість інтегруватись з іншими додатками або сервісами, такими як календарі та інші (для ефективності і зручності).

Routinery має розширений функціонал, але незадовільний на фундаментальному рівні - складний інтерфейс, відсутність функціоналу обміну завдань, відсутність компонентної системи і інтерактивності, відсутність спільного користування завданнями.

Незважаючи на окремі функціональні можливості, програмне забезпечення Routinery складне в використанні, зі складним дизайном та набором компонентів, які не використовуються при роботі з програмою.

## Проблеми створення і організації програмних систем

Створення і організація програмних систем планування і організації часу людини має проблеми зі складною структурою компонентної системи, залученням новітніх підходів та засобів розробки, в тому числі нових мов програмування.

Переліком основних проблем, з якими доводиться зіткнутися розробниками програмних систем є:

1. Визначення необхідних вимог. Необхідне визначення точних вимог до системи, які можуть змінюватися під час процесу розробки або бути неповними. Неправильне визначення вимог може призвести до некоректної роботи програмної системи.
2. Управління змінами до вимог. Програмні системи потребують постійних змін і оновлень в роботі. Некоректні або недостатні зміни можуть призвести до невірною роботи системи або втрати даних.
3. Розробка та тестування. Розробка програмних систем вимагає достатнього часу, ресурсів та ефективного тестування, щоб виявити помилки та дефекти програми. Недостатнє тестування може призвести до появи помилок у системі.
4. Управління проектом. Управління проектом є ключовим фактором для розробки програмних систем. Недостатнє планування або погане керування призводять до затримок у розробці системи, перевищення бюджету і невиконання необхідних вимог.
5. Безпека. Програмні системи вразливі до різного роду загрозам безпеці: “хакерські” атаки, витоки даних та інше. Забезпечення безпеки програмної системи є важливою вимогою, яке потребує змін та тестування на кожному етапі розробки.
6. Сумісність та інтеграція. Необхідно забезпечити сумісність та інтеграцію програмної системи з вже наявними програмними комплексами та компонентами. Несумісність приводить до проблем з обміном даних та невірною роботою системи.

Отже, можна сформулювати наступні шляхи розв’язання проблем створення програмних систем організації часу людини для підвищення їх ефективного використання і рівня зацікавленості користувачів:

1)удосконалення підходів щодо опису і поданні організаційних завдань за рахунок створення компонентного конструктора для користувачів.

2)удосконалення методів покращення якості виконання завдань за рахунок створення інструментів з поширення напрацювань за окремими видами завдань.

3)удосконалення управлінням робочими групами та окремими робітниками за рахунок створення системи заохочення найкращих рішень при виконанні завдань.

Аналіз показує, що доцільно звернути увагу на подальший розвиток і вдосконалення напряму пакету Structured і розвивати його функціональні можливості за наступними задачами у новому проекті системи організації часу людини:

1. Пакети завдань: проект повинен дозволяти організовувати завдання в пакети або групи, що допомагає структурувати роботу та керувати комплексними проектами.
2. Групи користувачів: проект повинен дозволяти додавати пакети до груп користувачів, що необхідно для спільної роботи над проектами або для делегування завдань між учасниками групи.
3. Відстеження прогресу: проект повинен надавати не тільки можливість відстежувати прогрес виконання завдань, але й інструмент для аналітичної і інтерактивної роботи з даними. Утворити систему інтерактивних компонентів, що надасть можливість нескінченно розширювати можливості для користувачів взаємодіяти з інформацією (завданнями).
4. Компонентна система: проект повинен надати можливість гнучкого середовища для настройки пакетів завдань, оновлення поточного набору компонентів надають набір інструментів для встановлення і налагодження завдань. Проект системи повинен представляти завдання як налаштовані компоненти, що налагоджуються за допомогою простого інтерфейсу користувача.
5. Робота зі списками даних: проект повинен надати можливість працювати зі списками даних та алгоритмічно їх прикріпляти до завдань. Цей функціонал корисний для обробки та аналізу інформації, пов'язаної з конкретним завданням.
6. Запис даних до завдань: проект повинен надати можливість користувачеві зберігати дані до завдань, включати додаткову інформацію, коментарі, примітки або відповіді, які можуть бути оброблені компонентами, пов'язаними з завданням.
7. Інтерфейс користувача: проекті повинен надати користувачеві можливість легко виконувати функції та операції з завданнями та даними.

## Постановка задачі створення інтерактивного-компонентного планувальника завдань

Основна задача проектування інтерактивного-компонентного планувальника завдань полягає в розробці гнучкого інструменту для планування завдань, які повторюються, оптимізації рутинних завдань, інтеграції з групами робітників (колективів) або виконавців. Проект повинен розроблятися за моделлю "від продукту до клієнта" (Project to Client) і бути спрямований на покупців (користувачів).

Основними вимогами до функціоналу інтерактивного-компонентного планувальника завдань є:

* Ефективне розподілення часу людини або робітника при плануванні робочого дня або при плануванні денних подій (які мають вимогу повторення)
* Отримання своєчасних інформаційних повідомлень про розподілення денного часу поточних подій
* Створення нових персональних пакетів завдань. Завантаження пакетів завдань до системи для обміну між всіма користувачами.
* Створення умов для використання пакетів завдань групами користувачів або груп за спільними завданнями.
* Інтуїтивно зрозумілий, гнучкий та простий інтерфейс користувача при роботі з додатками системи

Проект має розроблятися на системах гнучких повідомлень і обробки інформації у платформі iOS, Telegram та враховувати наступні вимоги:

* Застосунок використовує інтерфейс згідно з Apple Guidelines. Кожен елемент повинен мати функціональну роль, дизайн система і ідеологія візуальних компонент Apple повинна використовуватись застосунку. Інтерфейс користувача повинен бути простим та легким в використанні. Інтерфейс користувача адаптований під всі типи екранів смартфонів Apple (iPhone).
* Застосунок повинен заощадливо використовувати ресурси пристрою: швидка та безпечна робота з оперативною пам'яттю, оптимальна робота з потоками системи.
* Інтерфейс взаємодії з ботом повідомлень надається через спеціальні команди.

## Висновок до розділу

Проаналізовано предметну область з розробки програмної системи для автоматизації часу людини, визначені основні проблеми при створені програмних систем, та виконана постановка задачі на створення інтерактивного-компонентного планувальника завдань. За умови ефективного вирішення проблем зі створення програмної системи та за умови виконання поставлених задач, очікується програмне рішення, яке надасть користувачам зручний інструмент для планування денних завдань та подій. Користувачі об'єднані в робочі групи та групи за спільними завданнями отримають гнучке програмне рішення для поліпшення умов праці та результатів роботи.

# АРХІТЕКТУРА СИСТЕМИ ІНТЕРАКТИВНОГО-КОМПОНЕНТНОГО ПЛАНУВАННЯ ЗАВДАНЬ

## Формалізоване подання процесу планування завдань

### Система компонентів

##### Типи компонентів та їх обробники

Необхідно мати n типів компонентів + n1 типів обробників + n2 реалізацій типів обробників. Де n = n1 = n2

##### Компоненти

Необхідно мати m компонентів + m1 вхідних даних до них + m2 реалізацій цих компонентів. Де m = m1 = m2

### Алгоритм роботи з пакетами завдань

##### Завдання (Task)

Необхідно щоб кожен task[n] мав можливість мати m компонентів, де компонент може мати під собою k типів комонентів, які у свою чергу мають k1….kN обробників, які в свою чергу мають l…lN обробників типів компонентів

##### Пакет

Необхідно мати можливість створювати і зберігати n пакетів для m користувачів. Кожен пакет має мати k завдань

##### Магазин пакетів

Магазин повинен зберігати n кількість пакетів. Користувач має мати змогу загружати пакет до сховища – n.push(newPackage), та видаляти n.deleteAt(packageIndex)

#### Обробляюча програма

Програма має мати можливість загружати для кожного окремого користувача n[userIndex], k кількість пакетів.

##### Обробник «Appear» компонентів

Кожну секунду потрібно робити перевірку чи є завдання y1…yN, які треба показати для пакету q[packageIndex] для користвувача n[userIndex]. Кожен компонент повертає Bool кожну секунди чи потрібно його показувати

##### Обробник «Data» компонентів

Для кожного компоненту k, за умови що він “Data” компонент в завдані z, за умови що він повинен з’явитись у час T отримати строку данних. Відобразити массив [k] у завданні

##### Обробник «Interactive» компонентів

Для кожного «Interactive» компоненту згенерувати тег у форматів «буква» «цифра» - a1, a2, a3 ….a50…aN. Для кожного компоненту k, за умови що він “ Interactive” компонент в завдані z, за умови що він повинен з’явитись у час T отримати строку данних. Відобразити массив [k] у завданні. Для кожного компоненти k надати можливість дати відповідь. Повернути Bool чи відповідть була правильна.

### Реалізації компонентів

##### Interval

Кожен компонент k перевіряє який тип – WeekDays чи Days. WeekDays – чи сьогодні той день d, який зазначено у масиві днів, в які потрібно з’являтись завданню, WeekDays[dayToAppearIndex1… dayToAppearIndexN]. І перевіряє час Interval, чи пройшов Interval і підохидть час

##### Description

Для кожного компоненту k повертає строку с описом

### Обґрунтування методу розв’язування

Мова Swift є високорівневою мовою програмування, розробленою з урахуванням простоти та ефективності розробки програм. У порівнянні з мовами C або C++, Swift не надає програмісту прямого доступу до структур даних, подібних до "масиву" з С/С++, які забезпечують швидкий пошук елементів завдяки простій операції додавання i+index, а також не вимагають алокації пам'яті у купі, що може бути часомоємним процесом для операційної системи.

Однак, масив у Swift оптимізований та має схожу реалізацію до "вектора у С++". Це означає, що масив у Swift може динамічно збільшуватись, що є критично важливим у випадку, коли не можна передбачити кількість користувачів або пакетів. Гнучкість є критичною властивістю в таких випадках. Для досягнення більшої оптимізації, рекомендується належним чином встановити властивість "capacity" масиву, щоб мінімізувати шанс алокації нової пам'яті, коли розмір масиву наближається до межі, оскільки Swift Array автоматично подвоює обсяг пам'яті.

У випадках, коли послідовність даних не має значення, можна використовувати множину Swift Set для збереження даних в пам'яті. Множина Swift Set є швидшою, ніж Swift Array, і може бути використана, наприклад, для зберігання пакетів у боті.

Крім того, корисним може бути використання хеш-мапи, яка у Swift називається словник (Dictionary). Наприклад у боті, ключем буде ідентифікатор користувача у Telegram, а значенням - сам пакет. Використання словника є оптимальним варіантом у такому сценарії.

## Архітектура

## Постановка задачі і функцій при створенні інтерактивного-компонентного планувальника завдань

### Аналіз вимог

#### iOS застосунок

A picture containing text, diagram, screenshot, line

Description automatically generated

Рисунок 1 iOS Діаграма варіантів використання

В таблицях далі наведено варіанти використання

|  |  |
| --- | --- |
| Use case name | Відкриття допомоги |
| Goals | Надати користувачу доступ до допомоги в застосунку |
| Actors | Користувач |
| Trigger | Користувач натискає кнопку обирає з меню налаштувань Onboarding |
| Pre-conditions | Зайти до налаштувань |
| Flow of Events | Користувач натискає кнопку налаштування і обирає з меню налаштувань Onboarding |
| Extension | - |
| Post-Condition | Запустився Onboarding з поясненнями |

Варіант використання UC- 1

|  |  |
| --- | --- |
| Use case name | Створити пакет |
| Goals | Надати можливість створювати нові «болванки» пустих пакетів |
| Actors | Користувач |
| Trigger | Користувач натискає кнопку додати пакет |
| Pre-conditions | - |
| Flow of Events | Користувач натискає мнопку налаштування, вводить назву |
| Extension | - |
| Post-Condition | Створюеться пустий пакет |

Варіант використання UC- 2

|  |  |
| --- | --- |
| Use case name | Створити задвання і запонити пакет |
| Goals | Надати можливість наповнювати пакети завданнями |
| Actors | Користувач |
| Trigger | Користувач натискає додати задання |
| Pre-conditions | Бути у пакеті |
| Flow of Events | Користувач заходить до пакету, натискає додати задання, додає компоненти до завдання |
| Extension | - |
| Post-Condition | Створюеться завдання в пакеті |

Варіант використання UC- 3

|  |  |
| --- | --- |
| Use case name | Створити задвання і запонити пакет |
| Goals | Надати можливість наповнювати пакети завданнями |
| Actors | Користувач |
| Trigger | Користувач натискає додати задання |
| Pre-conditions | Бути у пакеті |
| Flow of Events | Користувач заходить до пакету, натискає додати задання, додає компоненти до завдання |
| Extension | - |
| Post-Condition | Створюеться завдання в пакеті |

Варіант використання UC- 4

|  |  |
| --- | --- |
| Use case name | Видаляти пакети |
| Goals | Надати можливість видаляти пакети завданнями |
| Actors | Користувач |
| Trigger | Свайп вліво, натиснути кнопку видалити |
| Pre-conditions | - |
| Flow of Events | Свайп вліво, натиснути кнопку видалити |
| Extension | - |
| Post-Condition | Пакет видаляється і зникає |

Варіант використання UC- 5

|  |  |
| --- | --- |
| Use case name | Видаляти завдання |
| Goals | Надати можливість видаляти завдання |
| Actors | Користувач |
| Trigger | Свайп вліво, натиснути кнопку видалити |
| Pre-conditions | Бути у пакеті |
| Flow of Events | Свайп вліво, натиснути кнопку видалити |
| Extension | - |
| Post-Condition | Завдання видаляється і зникає |

Варіант використання UC- 6

|  |  |
| --- | --- |
| Use case name | Видаляти копомнент |
| Goals | Надати можливість видаляти завдання |
| Actors | Користувач |
| Trigger | Свайп вліво, натиснути кнопку видалити |
| Pre-conditions | Бути у задавнні |
| Flow of Events | Свайп вліво, натиснути кнопку видалити |
| Extension | - |
| Post-Condition | Завдання видаляється і зникає |

Варіант використання UC- 7

|  |  |
| --- | --- |
| Use case name | Авторизація |
| Goals | Надати можливість бути авторизованим і дати доступ до загрузки своїх пакетів до магазину |
| Actors | Користувач |
| Trigger | Викласти пакет |
| Pre-conditions | - |
| Flow of Events | Користувач заходить до магазину і пробує викласти свії пакет до магазину, вилазить авторизація, користувач авторизується |
| Extension | - |
| Post-Condition | Користувач авторизувався, авторизація зникає |

Варіант використання UC- 8

|  |  |
| --- | --- |
| Use case name | Копіювання пакету |
| Goals | Надати можливість вибрати пакет і скопіювати його собі до аккаунту |
| Actors | Користувач |
| Trigger | Свайп вліво, кнопка скопіювати |
| Pre-conditions | - |
| Flow of Events | Користувач заходить до магазину, вибирає пакет, свайп вліво, кнопка скопіювати |
| Extension | - |
| Post-Condition | Користувач скопіював пакет, він з'явився у головному меню, мін може його редагувати |

Варіант використання UC- 9

|  |  |
| --- | --- |
| Use case name | Викладка пакету |
| Goals | Надати можливість викладати пакету до магазину |
| Actors | Користувач |
| Trigger | Нажати + у магазині |
| Pre-conditions | Бути авторизованим |
| Flow of Events | Зайти до магазину, натиснути кнопку +, вибрати пакет |
| Extension | - |
| Post-Condition | Пакет загружається до серверу, кожен інший користувач тепер може його побачити і скопіювати |

Варіант використання UC- 10

#### Система нотифікацій

A diagram of a person

Description automatically generated with low confidence

Рисунок 2 Telegram Bot діаграма варіантів використання

|  |  |
| --- | --- |
| Use case name | Додати пакет |
| Goals | Надати можливість додавати пакет |
| Actors | Користувач |
| Trigger | Команда add package |
| Pre-conditions | Бути у боті |
| Flow of Events | Закинути в бота пакет, реплаем зробити команду add |
| Extension | - |
| Post-Condition | Пакет додається до боту |

Варіант використання UC- 11

|  |  |
| --- | --- |
| Use case name | Видалити пакет |
| Goals | Надати можливість видаляти пакет |
| Actors | Користувач |
| Trigger | Команда rm package\_index |
| Pre-conditions | Бути у боті |
| Flow of Events | Команда rm package\_index |
| Extension | - |
| Post-Condition | Пакет видаляється з боту |

Варіант використання UC- 12

|  |  |
| --- | --- |
| Use case name | Список пакетів |
| Goals | Надати можливість дивитися список пакетів |
| Actors | Користувач |
| Trigger | Команда pls |
| Pre-conditions | Бути у боті |
| Flow of Events | Команда pls |
| Extension | - |
| Post-Condition | Показує все пакети в боті |

Варіант використання UC- 13

|  |  |
| --- | --- |
| Use case name | Надсилання повідомлення |
| Goals | Надати можливість надсилати повідомлення |
| Actors | Користувач |
| Trigger | Один з «Appear» компонентів повернув True |
| Pre-conditions | Додати до бота пакет |
| Flow of Events | Час приходить для показу задвання, приходить повідомлення на телефон з завданням |
| Extension | - |
| Post-Condition | Повідомлення прийшло |

Варіант використання UC- 14

### Функціональні вимоги

#### iOS Застосунок

|  |  |
| --- | --- |
| Назва | Надати можливість надавати допомогу |
| Опис | Система повинна надати можливість надавати допомогу |

Функціональна вимога FR- 1

|  |  |
| --- | --- |
| Назва | Надати можливість створювати пакети завдань |
| Опис | Система повинна надати можливість створювати пакети завдань |

Функціональна вимога FR- 2

|  |  |
| --- | --- |
| Назва | Надати можливість редагувати пакети |
| Опис | Система повинна надати можливість редагувати пакети, а саме: змінювати назву пакету, додавати нові завдання, видаляти, редагувати завдання |

Функціональна вимога FR- 3

|  |  |
| --- | --- |
| Назва | Надати можливість створювати завдання |
| Опис | Система повинна надати можливість створювати завдання, а саме: давати назву завданню, додавати компоненти, видаляти, редагувати компоненти |

Функціональна вимога FR- 4

|  |  |
| --- | --- |
| Назва | Надати можливість редагувати завдання |
| Опис | Система повинна надати можливість редагувати завдання, а саме: давати назву завданню, додавати компоненти, видаляти, редагувати компоненти |

Функціональна вимога FR- 5

|  |  |
| --- | --- |
| Назва | Надати можливість видаляти окремий екземпляр компонента |
| Опис | Система повинна надати можливість видаляти окремий екземпляр компонента, його налаштування |

Функціональна вимога FR- 6

|  |  |
| --- | --- |
| Назва | Надати можливість редагувати окремий екземпляр компонента |
| Опис | Система повинна надати можливість редагувати окремий екземпляр компонента, його налаштування |

Функціональна вимога FR- 7

|  |  |
| --- | --- |
| Назва | Надати можливість переглядати пакети спільноти |
| Опис | Система повинна надати можливість показувати всі пакети спільноти |

Функціональна вимога FR- 8

|  |  |
| --- | --- |
| Назва | Надати можливість копіювати пакети спільноти |
| Опис | Система повинна надати можливість скопіювати будь-який пакет з магазину |

Функціональна вимога FR- 9

#### Система нотифікацій

|  |  |
| --- | --- |
| Назва | Надати можливість маніпулювати з пакетами в боті |
| Опис | Система повинна надати можливість маніпулювати з пакетами в боті, а саме: додавати і видаляти |

Функціональна вимога FR- 10

|  |  |
| --- | --- |
| Назва | Надати можливість дивитися активні пакети в боті |
| Опис | Система повинна надати можливість дивитися активні пакети в боті |

Функціональна вимога FR- 11

|  |  |
| --- | --- |
| Назва | Надати можливість отримувати повідомлення |
| Опис | Система повинна надати можливість надсилати повідомлення |

Функціональна вимога FR- 12

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | FR-1 | FR-2 | FR-3 | FR-4 | FR-5 | FR-6 | FR-7 | FR-8 | FR-9 | FR-10 | FR-11 | FR-12 |
| UC-1 | + |  |  |  |  |  |  |  |  |  |  |  |
| UC-2 |  | + |  |  |  |  |  |  |  |  |  |  |
| UC-3 |  |  | + |  |  |  |  |  |  |  |  |  |
| UC-4 |  |  |  | + |  |  |  |  |  |  |  |  |
| UC-5 |  |  | + |  |  |  |  |  |  |  |  |  |
| UC-6 |  |  |  |  | + |  |  |  |  |  |  |  |
| UC-7 |  |  |  |  |  | + | + |  |  |  |  |  |
| UC-8 |  |  |  |  |  |  |  | + |  |  |  |  |
| UC-9 |  |  |  |  |  |  |  |  | + |  |  |  |
| UC-10 |  |  |  |  |  |  |  |  | + |  |  |  |
| UC-11 |  |  |  |  |  |  |  |  |  | + |  |  |
| UC-12 |  |  |  |  |  |  |  |  |  | + |  |  |
| UC-13 |  |  |  |  |  |  |  |  |  |  | + |  |
| UC-14 |  |  |  |  |  |  |  |  |  |  |  | + |

Таблиця 1 - Матриця трасування вимог

## Архітектура

## Вимоги до розробки компонентного планувальника завдань

### Вимоги до коду

Не ставити абстрактні цілі писати код «чисто», «гарно» «добре» чи дотримання принципів SOLID, DRY, бо вони дають тільки абстрактні вимоги до розробника і не показують як саме потрібно зробити. Є деякі цілі в коді, які допоможуть підтримувати систему на далі

* Складні системи як застосунок iOS, чи будь-які надалі повинні мати генеральну архітектуру, набір правил за якими буде розвиватися проект. При невдалому проектуванні цієї частини – виявити проблему і знайти час переписати. Такі зміни позначати у першому числі версії. <нова архітектура>.Х.Х (1.0.0; 2.2.3)
* Не зв’язувати код куски між собою дуже сильно. Для того, щоб можна було просто видалити невдалі зміни в проекті і замінити їх. Для щоб можна було код перевикористовувати.
* Кожна зміна в коді повинна відповідати на питання «навіщо це робити?»
* Не зациклюватись на коді. Так як системи пишеться модульно, щось неідеальне можна покращити чи зовсім видалити і замінити

### Системні вимоги

Для повноцінного функціювання системи потрібно розробити наступні підсистем:

* Система повідомлень. Надсилає повідомлення і збирає всю інформацію і відправляє разом із завданням.
* Місце створення пакетів завдань. Надати в системі інструменти для створення гнучких завдань за допомогою компонентної системи. Надати можливість редагувати компоненти, зберігати і видаляти
* Місце відображення пакетів спільноти. Нади можливість копіювати пакети
* Гнучка розширювана система компонентів. Надати можливість додавати нові компоненти, розробити типи компонентів і їх обробники.
* Розробити сховище пакетів завдань користувачів і авторизацію.

# РЕАЛІЗАЦІЯ ІНТЕРАКТИВНОГО-КОМПОНЕНТНОГО ПЛАНУВАННЯ ЗАВДАНЬ

## Засоби розробки

У виборі технологій для розробки застосунку для мобільних пристроїв я керуюся двома головними критеріями: моїми особистими уподобаннями та вимогами моєї системи. Важливо зауважити, що ці вибори базуються на моєму власному досвіду і не можуть бути розглянуті як загальноприйнята істина.

Мій особистий досвід та уподобання включають такі фактори, як:

* Знання та досвід технологій: Якщо я вже маю досвід з певною технологією, це може стати перевагою при виборі, оскільки це дозволить мені швидше та ефективніше розробляти застосунок.
* Комфорт у використанні інструментів
* Уподобання щодо стилю програмування
* Функціональність: Я оцінюю, чи задовольняють обрані технології всі функціональні вимоги мого застосунку.
* Продуктивність: В залежності від потреб моєї системи, я оцінюю продуктивність і швидкодію технологій, щоб забезпечити оптимальну роботу застосунку.

При розробці даної інформаційної системи використовувались багато  
технічних засобів. Технології наведені у таблиці

|  |  |
| --- | --- |
| Тип | Назва |
| Мова програмування | Swift, PHP, SQL, |
| Фреймворки | SwiftUI, UIKit, TelegramSDK, Foundation |
| IDE | Xcode, DBvaer |
| RDBMS | MySQL |
| Засіб проектування | Figma |
| Система контроля версій | Git |
| Репозиторій | GitHub |

Таблиця 2 – технології

Git є системою керування версіями, яка забезпечує можливість відстежувати та керувати змінами в файловій системі під час розробки програмного забезпечення. Вона дозволяє кільком розробникам спільно працювати над проектом одночасно, зберігаючи та контролюючи різні версії файлів зі змінами.

### Технології iOS застосуноку

Для того, щоб розробити застосунок можна використовувати різні інструменти. Їх умовно можна поділити на два типи

Перший це інструменти, які дозволяють розробити застосунок для iOS і Android одночасно, а саме:

React Native: React Native дозволяє використовувати JavaScript для розробки мобільних додатків для iOS та Android.

Flutter: Flutter використовує мову програмування Dart і дозволяє створювати нативно виглядаючі додатки для iOS та Android зі спільним кодом. Він має власний візуальний двигун, що забезпечує більшу швидкість та високу продуктивність.

Xamarin: Xamarin дозволяє розробляти мобільні додатки для iOS, Android і Windows, використовуючи мову програмування C#. Він надає можливість спільного використання коду та бібліотек, але також дозволяє використовувати платформено-специфічні функції.

Другий тип розробки, це використання офіційних інструментів платформ

iOS. UIKit і SwiftUI це два головних інструмента для розробки під iOS. Вони використовують мови Swift і Objective-C. Вони надають повний доступ до всіх функцій і можливостей, пропонованих платформою Apple.

Android.Android SDK є головним інструментом для розробки Android застосунку. Він використовує мови програмування Java або Kotlin. Забезпечується повний доступ до функцій та можливостей платформи Android.

Нативна розробка надає найвищу продуктивність та повний контроль над функціоналом пристроїв, але потребує окремого розробки для кожної платформи і більше зусиль.

Переваги кросплатформеного підходу включають:

* Загальний код для різних платформ, що зберігає час і зусилля розробки.
* Швидка розробка та здатність оновлювати застосунки одночасно для всіх платформ.
* Більш широкий вибір розробних інструментів та фреймворків.

Нативний підхід має наступні переваги:

* Максимальна продуктивність та ефективність роботи на платформі.
* Повний доступ до платформено-специфічних функцій і можливостей.
* Краща інтеграція з екосистемою платформи.

Недоліки кросплатформеного підходу можуть включати:

* Менша продуктивність порівняно з нативними додатками.
* Обмеження в доступі до деяких платформено-специфічних функцій.
* Потенційні проблеми з роздрібними платформами оновлень та сумісності.

Написання високоякісного застосунку можливе на будь-якій технології. Однак, як користувач пристроїв Apple, я вирішив обрати нативну розробку під iOS. Технології Apple дозволяють швидко створити нативний інтерфейс, який, на мою думку, виглядає найкраще на ринку. Це дозволяє забезпечити оптимальний досвід користувача та максимальну сумісність з функціоналом і стандартами iOS.

**Мова програмування.** Swift і Objective-C є двома основними мовами програмування, які використовуються для розробки застосунків під платформу iOS. Ось порівняння між ними:

1. Синтаксис і зручність використання:
   * Swift має сучасний, експресивний синтаксис, який робить код більш зрозумілим та читабельним. Вона підтримує безпечну типізацію, опціонали, функціональне програмування та інші сучасні парадигми програмування.
   * Objective-C використовує класичний синтаксис C з додатковими розширеннями для об'єктно-орієнтованого програмування. Він може виглядати більш складним для новачків, оскільки використовує квадратні дужки та специфічний спосіб виклику методів.
2. Сумісність та наявність кодової бази:
   * Swift є мовою, яка активно розвивається та підтримується Apple. Вона стає все більш популярною, і багато нових проектів розробляються виключно з використанням Swift. Однак, є велика кількість існуючої кодової бази на Objective-C, яку можна легко інтегрувати з Swift.
   * Objective-C була основною мовою розробки для платформи iOS протягом багатьох років. Багато існуючих бібліотек, фреймворків та додатків написані на Objective-C, що робить його важливим для збереження сумісності та розширення старих проектів.
3. Продуктивність та швидкодія:
   * Swift відома своєю високою швидкодією та продуктивністю. Вона використовує вбудовану оптимізацію та має простий доступ до низькорівневих операцій, що дозволяє писати швидкодіючі застосунки.
   * Objective-C також може бути швидким, але у порівнянні з Swift, його виконання може бути трохи повільніше. Це особливо помітно в великих проектах з великою кількістю об'єктів та викликів методів.
4. Підтримка та майбутнє розвитку:
   * Swift активно розвивається Apple і має велику підтримку спільноти розробників. Вона постійно отримує оновлення та нові функції, що полегшують розробку та покращують продуктивність.
   * Objective-C все ще підтримується Apple, але більшість нових функцій та поліпшень доступні виключно для Swift. Тим не менш, існуючі проекти на Objective-C продовжують працювати і можуть бути підтримані протягом тривалого часу.

Загалом Objective-C вже не є стандартом для розробки. Хоча я маю зауважити, що це потужна мова, яка навіть зараз має деякий вплив на iOS розробку. Через те, що Swift сумісний з Objective-C можна зробити цікаві функціональні речі. Для свого застосунку буду використовувати Swift, через зручність і швидкідсть

Swift - це мова програмування, розроблена компанією Apple. Вона була представлена в 2014 році і призначена для розробки програмного забезпечення для платформ Apple, таких як iOS, macOS, watchOS та tvOS. Swift поєднує в собі потужність традиційних мов програмування з простотою та безпекою сучасних мов.

Основні особливості Swift включають:

* Синтаксис, легкий для читання та написання: Swift має чистий та зрозумілий синтаксис, що сприяє простоті розробки та зрозумінню коду. Він був створений з урахуванням зручності програміста.
* Безпека типів: Swift є типо-безпечною мовою програмування, що дозволяє виявляти та уникати багів, пов'язаних з неправильним використанням типів даних. Вона має механізми перевірки типів, які допомагають уникати помилок під час компіляції.
* Швидкодія: Swift була оптимізована для максимальної продуктивності. Вона використовує передові техніки компіляції та оптимізації, що дозволяють програмам, написаним на Swift, працювати швидко та ефективно.
* Широкий функціонал: Swift надає розширений функціонал для розробки різноманітних програмних застосунків. Вона підтримує роботу зі звичайними типами даних, об'єктно-орієнтований підхід, функціональне програмування, обробку помилок, паралельне програмування та багато іншого.

Swift став популярним серед розробників завдяки своїм перевагам і простоті використання. Він продовжує активно розвиватись та оновлюватись, сприяючи швидкому та надійному розробці програмного забезпечення для пристроїв Apple.

**SwiftUI чи UIKit.** UIKit і SwiftUI - це два основні фреймворки для розробки інтерфейсу користувача (UI) для пристроїв Apple, таких як iPhone, iPad та Mac. Вони мають деякі суттєві відмінності і призначені для різних сценаріїв розробки. Давайте порівняємо їх:

1. UIKit:

* UIKit є традиційним фреймворком для розробки UI під iOS
* Використовує мову програмування Objective-C або Swift.
* Має широкий набір готових елементів інтерфейсу, таких як кнопки, тексти, таблиці, колекції тощо, що дає розробникам велику гнучкість у створенні власних UI-елементів.
* Вимагає більше коду для створення та налаштування інтерфейсу.
* Забезпечує високу ступінь кастомізації та контролю над виглядом та поведінкою UI-елементів.
* Має можливість глибоко завезти у реалізації

1. SwiftUI:

SwiftUI - це новий декларативний фреймворк, представлений Apple у 2019 році.

* Використовує мову програмування Swift.
* Основна ідея SwiftUI полягає у тому, що ви описуєте, як має виглядати UI, а не як його створити, за допомогою декларативного підходу. Це спрощує процес розробки та забезпечує більш чистий та зрозумілий код.
* Має широкий набір вбудованих компонентів та модифікаторів, що дозволяють швидко створювати складні інтерфейси.
* SwiftUI є кросплатформеним і може бути використаний для розробки UI для iOS, macOS, watchOS та tvOS.
* Надає функціональність, таку як автоматичне оновлення UI при зміні даних (reactive programming) та підтримку Dunk Mode та інших нових функцій.

SwiftUI - це захоплива інновація, яка мені дуже подобається, проте він має свої нюанси. Додатки, розроблені за допомогою SwiftUI, вимагають iOS 13 або новішої версії, і в ньому є критичні баги в навігаційному шарі, а також обмеження в реалізації деяких UI-компонентів. З іншого боку, UIKit не має таких обмежень.

Моє рішення - поєднати обидва інструмента, використовуючи найкращі аспекти кожного з них для досягнення найкращого результату. З UIKit я беру навігацію, реалізацію певних UI-компонентів та управління вікнами, оскільки він надійно працює в цих аспектах. SwiftUI же відповідає за реалізацію решти UI-елементів і вікон, оскільки він пропонує зручний і декларативний підхід до розробки інтерфейсу.

Таке поєднання фреймворків забезпечує потужність та гнучкість UIKit разом зі зручністю та швидкістю SwiftUI. Такий підхід дозволяє створювати сучасні та привабливі додатки для пристроїв Apple, використовуючи оптимальні можливості обох фреймворків.

SwiftUI – це набір інструментів і бібліотек для розробки інтерфейсів користувача програм на мові програмування Swift, яка використовується для розробки додатків на платформах Apple, таких як iOS, macOS, watchOS та tvOS. SwiftUI надає простий спосіб створювати і візуально налаштовувати різноманітні елементи інтерфейсу користувача, такі як кнопки, текстові поля, списки та багато іншого. Завдяки своїй декларативній природі, SwiftUI дозволяє описувати, як має виглядати ваш інтерфейс, а інструмент самостійно забезпечує його відображення та оновлення. Це спрощує процес розробки і дозволяє швидше створювати багатофункціональні додатки для платформ Apple

UIKit - це фреймворк, розроблений компанією Apple для побудови графічного інтерфейсу користувача на пристроях з операційною системою iOS, таких як iPhone, iPad і iPod Touch. Він надає набір інструментів , які дозволяють розробникам створювати високоякісні інтерактивні додатки з використанням різноманітних елементів управління, таких як кнопки, тексти, таблиці, зображення і багато іншого. UIKit забезпечує такі можливості, як обробка подій користувача, анімація, мультимедіа, робота зі зображеннями та графікою, а також навігація між різними екранами додатка. UIKit є одним з основних компонентів для розробки iOS-додатків і дозволяє розробникам створювати привабливі та функціональні програми для платформи iOS.

### Система компонентів

Для розробки бібліотеки з логікою, яка буде взаємодіяти зі Swift, є різні варіанти вибору мови програмування. Swift пропонує можливість взаємодії з багатьма іншими мовами, особливо з мовами, які підтримують міжмовний інтерфейс (англ. "interoperability").

Ось кілька можливих варіантів мов програмування для розробки бібліотеки з логікою, яка інтероперує зі Swift:

1. Swift: Розробка бібліотеки з логікою прямо на Swift є найбільш природним варіантом, оскільки вона гармонійно взаємодіє з мовою Swift і надає доступ до всіх його функціональних можливостей. Це також сприяє зручності управління кодом та його обслуговування.
2. C/C++: Якщо потрібна максимальна продуктивність або вже є існуючий код на мовах C або C++, можна використовувати ці мови для розробки бібліотеки. Swift має потужні можливості інтероперабельності з мовами C та C++, що дозволяє легко використовувати функції та типи цих мов у Swift-коді.
3. Інші мови: Залежно від конкретного випадку використання, можна також використовувати інші мови програмування, які підтримують міжмовний інтерфейс зі Swift, наприклад Objective-C, Python, Ruby та інші. За допомогою спеціальних інтерфейсів, які надає Swift, можна інтегрувати функціональність цих мов у бібліотеку.

Так як я роблю додаток і бота на Swift, робити бібліотеку також на цій мові мені задалось гарним варіантом, її можна буду легко імпортувати, такий же синтаксис, доступні всі функції мови, чого може не бути з іншою мовою програмування.

### Серверна програма і БД

Backend (також відомий як бекенд або бек) можна реалізувати за допомогою різних мов програмування. Однак я вирішив обрати PHP як найшвидший спосіб написання бекенду, оскільки в мене обмежений термін для розробки системи та написання записок - всього лише місяць. Фізично мені не вистачає часу, щоб розгортати та налаштовувати Java або Kotlin, які, можливо, більше мені до вподоби.

Крім того, мій бекенд є дуже простим, і вибір мови тут не є вирішальним фактором. Кодування та обробка JSON даних однаково прості для будь-якої мови програмування. PHP також має достатню потужність і добре справляється з поставленими задачами. В майбутньому, я можу замінити PHP, якщо він суттєво відстає від інших серверних мов (хоча я сумніваюся в цьому під час розробки дипломної роботи).

PHP (Hypertext Preprocessor) - це скриптова мова програмування, яка призначена для розробки веб-додатків та динамічних веб-сторінок. PHP використовується для обробки даних на стороні сервера, тобто виконується на веб-сервері перед тим, як сторінка надсилається до клієнта.

Основні особливості та можливості PHP включають:

1. Вбудований HTML: PHP може бути вбудований безпосередньо в HTML-код, що дозволяє використовувати PHP-скрипти для генерації динамічного вмісту веб-сторінок.
2. Розширена підтримка баз даних: PHP має вбудовану підтримку для різних типів баз даних, таких як MySQL, PostgreSQL, SQLite та інші. Це дозволяє легко взаємодіяти з базами даних та виконувати операції збереження, отримання та оновлення даних.
3. Обробка форм: PHP надає зручні засоби для обробки даних, що надсилаються через форми на веб-сторінках. Він дозволяє отримувати та обробляти введені користувачем дані, валідувати їх та здійснювати необхідні дії на основі цих даних.
4. Робота з файлами: PHP надає можливості для роботи з файлами на сервері. Це охоплює читання, запис та редагування файлів, створення нових файлів, керування директоріями та інші операції, пов'язані з файловою системою.
5. Розширюваність: PHP має велику кількість розширень та бібліотек, які дозволяють розширити його функціональність. Це дозволяє використовувати готові рішення та інструменти для швидкого розроблення веб-додатків.

PHP є однією з найпопулярніших мов програмування для веб-розробки і використовується для створення різноманітних веб-додатків, від простих сайтів до складних веб-порталів та електронних комерційних систем.

MySQL - це система керування базами даних (СКБД), яка надає засоби для зберігання, організації та управління великими обсягами даних. Вона використовує мову запитів SQL (Structured Query Language) для взаємодії з базою даних. MySQL є однією з найпопулярніших відкритих систем у світі та широко використовується як веб-серверами, так і у самостійних додатках.

### Система повідомлень

Системи повідомлень можна реалізувати декільками способами, а саме: в iOS застосунку, через стороній сервіс, наприклад Telegram Bot

Реалізація в iOS застосунку має певинй перелік недоліків: тежє саме потрібно буде робити для Android застосунку, користувачі не зможуть ніяк доторкнутися до системи без iPhone, потрібно додатково розробити функціонал груп, це дорожче по грошам, ніж хостити бота. Однак рішення зробити нагадування у вигляді бота для месенджера вирішує багато з цих проблем. Я обрав Telegram як платформу для розробки, оскільки сам використовую його і маю багато друзів. У майбутньому, можливо, знадобиться розробка бота для інших платформ або перенесення функціоналу на додатки. Проте, це залежатиме від популярності системи. Наразі аудиторія Telegram буде достатньою для моїх потреб.

**Мова програмування.** Телеграм-ботів можна створювати на різних мовах програмування. Основні мови, які широко використовуються для розробки телеграм-ботів, включають наступні:

1. **Python:** Python є однією з найпопулярніших мов програмування для створення телеграм-ботів. Ви можете використовувати Python-бібліотеки, такі як python-telegram-bot або pyTelegramBotAPI, для взаємодії з Telegram Bot API та обробки повідомлень.
2. **Node.js:** Node.js є платформою для розробки серверних додатків на JavaScript. Ви можете використовувати бібліотеки, такі як Telegraf або node-telegram-bot-api, для створення телеграм-бота на Node.js.
3. **Java:** Java є широко використовуваною мовою програмування, яка також може бути використана для створення телеграм-ботів. Ви можете використовувати бібліотеку, наприклад, TelegramBots для взаємодії з Telegram Bot API.
4. **Ruby**: Ruby є простою та елегантною мовою програмування, яка також підтримує створення телеграм-ботів. Ви можете використовувати бібліотеку, таку як telegram-bot-ruby, для роботи з Telegram Bot API та створення функціонального бота.
5. **PHP**: PHP є широко використовуваною мовою для веб-розробки і також підтримує створення телеграм-ботів. Ви можете використовувати бібліотеки, такі як Telegram Bot SDK або php-telegram-bot, для взаємодії з Telegram Bot API.

Це найпопулярніші варіанти для створення телеграм-ботів. Я обрав мову програмування Swift для розробки бота, оскільки це надає зручну можливість імпортувати бібліотеку компонентів до нього. Таким чином, логіка може бути перевикористана як у iOS додатку, так і в боті.

**API для створення бота.** Для створення телеграм-бота на Swift ви можете використати такі інструменти:

1. **Telegram Bot API:** Це офіційний API, наданий Telegram, який дозволяє взаємодіяти з ботами через HTTP-запити. Можна використовувати бібліотеку Swift для роботи з цим API, наприклад, таку як telegram-bot-swift
2. **Vapor:** Vapor є фреймворком на мові Swift для створення веб-додатків та API. Ви можете використати Vapor для створення серверної частини вашого телеграм-бота та взаємодії з Telegram Bot API. Vapor надає можливість легко обробляти вхідні повідомлення, відповідати на них та виконувати потрібні дії.
3. **Perfect:** Perfect є ще одним фреймворком на мові Swift для створення серверних додатків та API. Можна можете використовувати Perfect для створення телеграм-бота. Він надає зручний інтерфейс для роботи з Telegram Bot API та обробки вхідних повідомлень.

Моїм рішенням є використання "telegram-bot-swift", який є обгорткою для мови Swift. Це дозволяє швидко досягати результату та сконцентруватися на виконанні поставлених завдань.

TelegramSDK - це набір інструментів для розробки додатків, які взаємодіють з месенджером Telegram, використовуючи мову програмування Swift. Цей SDK надає розробникам доступ до різноманітних функцій Telegram API, таких як надсилання повідомлень, отримання оновлень, робота з контактами і групами, управління ботами та багато іншого.

Swift TelegramSDK дозволяє зручно взаємодіяти з Telegram, надаючи простий і зрозумілий інтерфейс для використання функцій месенджера у власних додатках.

## Реалізація iOS застосунку

Для застосунку необхідно обрати кілька важливих архітектурних рішень:

* **Організація окремих модулів (вікон)**
* **Взаємодія модулів та передача даних між ними**
* **Навігація**: Потрібно вирішити, як буде організована навігація між модулями. Хто це робить, логіка навігації
* **Впровадження залежностей у модулі**

### Організація окремого модуля

Стандартна практика в індустрії - це паттерн MVC (Model-View-Controller). У компанії Apple вони розробили свій варіант паттерну, який називається Apple MVC. Хоча він є прийнятним рішенням, в деяких випадках може не відповідати потребам.

У світі розробки для iOS існує розмаїття архітектурних паттернів, які відомі як "архітектурні шаблони". Ці шаблони базуються на традиційній архітектурі MVC, але розширюють їх, розбиваючи на більші компоненти, використовуючи реактивний підхід для передачі даних між компонентами або включаючи логіку навігації в модуль вікна.

Варто зазначити, що логіка навігації є окремою схемою та окремою складовою програми, а модулі не залежать від неї і не мають жодного знання про неї. Таким чином, цей шар знаходиться нижче логіки навігації, що дозволяє підтримувати відокремленість між компонентами і спрощує розширення та підтримку системи в цілому.

У світі iOS розрізняють такі архітектурні паттерни:

1. MVP (Model-View-Presenter): У цьому архітектурному паттерні бізнес-логіка витягується у окрему сутність P (Presenter). Цей паттерн поліпшує стандартну MVC-архітектуру від Apple і не перевантажує систему окремим модулем.
2. MVVM (Model-View-ViewModel): У цьому архітектурному паттерні бізнес-логіка витягується у сутність VM (ViewModel), а передача даних та подій між екраном і ViewModel здійснюється за допомогою реактивного підходу. У стандартному iOS-додатку це передбачає використання RxSwift або Combine. Це ефективний підхід, особливо для прихильників реактивного програмування. Однак, на даний момент цей паттерн може бути складним вибором, оскільки Apple надає нову бібліотеку для цього, яка ще нестабільна, а RxSwift, хоч і популярний, може вийти з використання. Це питання особистих вподобань і наразі не відповідає моїм потребам, адже я не є прихильником реактивного підходу.
3. VIPER: Цей архітектурний паттерн включає навігаційну логіку в сутність R (Router), інтерактор виконує запити до сервера та передає дані презентеру, а презентер оновлює View, відповідальну за користувацький інтерфейс. VIPER є досить складним паттерном, що утрудняє написання коду, оскільки він включає в себе логіку навігації і не є належним чином перевикористовуваним, а також не відповідає принципам DRY (Don't Repeat Yourself). Крім того, інтерактор не є обов'язковою складовою, його можна передавати через контрактний інтерфейс до бізнес-логіки презентера або ViewModel. Загалом, на мою думку, це не найкращий варіант для використання.

Я буду використовувати MVP, бо він простий і виконує всі мої потреби.

A diagram of a presentation

Description automatically generated with low confidence

Рисунок 3.1 — Схема MVP

### Шари програми

Програма буде поділена на шари, для того, щоб завжди знати де що знаходиться

A screenshot of a phone

Description automatically generated with low confidence

Рисунок 3.2 — iOS шари програми

На вищому рівні в розробці для iOS існує навігація, яка базується на модулях і формує дерево програмного потоку. Одним з ключових елементів цієї навігаційної логіки є координатори, які відповідають за керування модулями та визначають їх послідовність. Використання координаторів є стандартною практикою в iOS.

Головним принципом цієї архітектури є те, що бізнес-логіка залежить лише від інтерфейсів користувача, що означає, що її інтерфейс можна замінити не торкаючись логіки.

Щодо мережевої взаємодії, вона відокремлена в окремий шар, який використовує лише бібліотеки Apple для цієї цілі.

### **Навігація**

Координатори (Coordinators) є популярним архітектурним підходом в iOS-розробці, який допомагає керувати навігацією та координацією функціональних модулів додатку. Вони дозволяють розділити відповідальність за навігацію між різними екранами та модулями додатку.

Основна ідея полягає в тому, що кожна логічна частина програми має свій власний координатор, який відповідає за керуванням навігацією та роботою з цим конкретним модулем. Координатори спрощують створення, ініціалізацію та взаємодію модулів.

Основні переваги використання координаторів включають:

1. **Відокремлення відповідальностей**: Координатори дозволяють відокремити відповідальність за навігацію та координацію між модулями. Це полегшує розширення та тестування окремих модулів.
2. **Перевикористання та модульність**: Координатори можуть бути легко перевикористані в інших частинах додатку або навіть в інших проектах. Вони покращують модульність та розширюваність додатку.
3. **Спрощення тестування**: Координатори полегшують тестування окремих модулів, оскільки їх можна відокремити від решти додатку та незалежно тестувати їх функціональність.

A picture containing text, line, font, diagram

Description automatically generated

Рисунок 3.3 — Схематичне зображення навігаційного дерева iOS застосунку

У рамках архітектури навігації, всі елементи навігаційної структури утворюють дерево, в якому кожен вузол може мати зв'язки з іншими вузлами, що дозволяє переходити до наступного вузла або повертатися назад. Координатори визначають логіку цієї навігаційної структури.

A picture containing text, diagram, plan, line

Description automatically generated

Рисунок 7 - Генеральна схема навігації

На схемі можно помітити два головні типи координторів: ті, що об’єднують тілька вікна і ті, що об’єднують інші координатори, забезпечують повну незалежність і перевикористовуємість окремих координторів. Перший тип у свою чергу забеспечую повну незалежність окремих вікон.

### **Впровадження залежностей у модулі**

Для забезпечення внедрення залежностей в iOS-розробці існує три основних підходи: Service Locator, Dependency Injection (DI) і фабричний метод.

Dependency Injection (DI) є процесом передачі залежностей об'єкту ззовні, замість внутрішнього створення цих залежностей всередині об'єкта. Цей підхід забезпечує більшу гнучкість, оскільки об'єкт не залежить від конкретної реалізації своїх залежностей і може легко замінювати їх іншими реалізаціями. В проекті також створюється клас з фабричними методами, який збирає окремі модулі, а модулі передаються через інтерфейс збирача, що реалізує DI. Коли створюється модуль, він передає себе як реалізатор контракту.

Service Locator (локатор служб) - це патерн проектування, який використовується в iOS-розробці для централізованого доступу до служб (сервісів) у додатку. Він надає механізм, який дозволяє об'єктам отримувати доступ до потрібних служб без прямого знання про їх реалізацію або деталі їх створення. Service Locator в iOS може бути реалізований за допомогою спеціального об'єкта-локатора, який містить посилання на реалізації служб і надає методи для отримання цих служб.

Фабричний метод (Factory Method) - це метод, який збирає сутність і повертає її екземпляр. Він активно використовується в поєднанні з DI та SL.

У моєму проекті, який має невеликий обсяг, я буду використовувати іноді фабричний метод, DI і Service Locator, який можна розглядати як варіант фабричного методу. Зараз фабричний метод для створення окремих сутностей виглядає досить доцільно.

На майбутнє, DI-клас з усіма сконфігурованими сутностями виглядає як дуже ефективне рішення, оскільки воно дозволяє швидко змінювати реалізації окремих частин коду в модулях і надає можливість для тестування коду в рамках модульних тестів. DI-система пронизує всю програму, і всі сутності стають частково залежними від нього, але це не є проблемою. Системи, такі як DI, рідко змінюються у проекті після їх впровадження.

### Інші частини системи

Інші компоненти системи на даному етапі не виявляють складних структур або рішень, оскільки ці окремі програми не є дуже масштабними в процесі створення першої версії системи.

## Аналіз допоміжних програмних засобів та засобів розробки

Xcode - це IDE (інтегроване середовище розробки), яке, можна сказати, є єдиною опцією для розробки під пристрої Apple. Хоча воно не є найкращим IDE у світі, останній конкурент, AppCode від JetBrains, недавно припинив своє існування. Крім того, лише Xcode дозволяє розміщувати додатки в App Store.

DBeaver - це простий безкоштовний редактор для тестування запитів до MySQL та інших баз даних. Він дуже зручний у використанні.

Simulator - це інструмент, що дозволяє симулювати роботу iPhone на ноутбуку. Він розроблений компанією Apple і використовується для тестування програм на різних моделях пристроїв.

## Висновок до розділу

У розділі, я детально пояснив правила та принципи написання коду. Пояснив, які технології і чому буду використовувати. Порівняв їх, один з одним. Розповів про архітектуру та методології в застосунку. Пояснив, де я буду писати системи і чому саме там

# ВПРОВАДЖЕННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНТЕРАКТИВНО-КОМПОНЕНТНОГО ПЛАНУВАЛЬНИКА ЗАВДАНЬ

## Тестування

Мета тестування якості програмного забезпечення

Виявлення помилок: Головною метою тестування є виявлення помилок, дефектів або некоректного функціонування програмного забезпечення. Це допомагає забезпечити належну якість та надійність продукту перед його випуском на ринок.

Перевірка відповідності вимогам: Тестування допомагає перевірити, чи відповідає програмне забезпечення вимогам і специфікаціям, встановленим для нього. Це дозволяє забезпечити, що програма виконує заплановані функції та задовольняє потребам користувачів.

Підтримка надійності: Тестування допомагає переконатися, що програмне забезпечення працює безперебійно та надійно в різних умовах та сценаріях. Це включає перевірку стабільності, витривалості та відмовостійкості системи.

Підвищення ефективності: Тестування може допомогти виявити та усунути недоліки в роботі програмного забезпечення, що можуть призводити до його низької продуктивності або швидкодії. Це дозволяє поліпшити ефективність роботи системи та забезпечити задоволення користувачів.

Забезпечення високої якості: Тестування сприяє підвищенню загальної якості програмного забезпечення шляхом ідентифікації та усунення проблем. Це включає перевірку функціональності, правильності та зручності використання програми.

Для перевірки якості програмонго забеспечення буде використовуватися Функціональне тестування, Unit tests

Функціональне тестування - це процес перевірки програмного забезпечення з метою визначення, чи працює воно відповідно до очікувань і вимог щодо функціональності. Цей тип тестування спрямований на перевірку основних функцій програми, таких як введення, обробка даних, виведення результатів та взаємодія з користувачем.

Unit tests (одиничні тести) - це процес тестування окремих "одиниць" програмного забезпечення для перевірки, чи працюють вони правильно і відповідають очікуваному результату. Одиниця може бути функцією, методом, класом або іншим незалежним компонентом програми.

Опис процесів тестування

Таблиця 4.1

|  |  |
| --- | --- |
| Мета тесту | Відкриття допомоги |
| Початковий стан системи | Головний єкран |
| Схема проведення тесту | Натискається кнопка налаштування і обираєтсья з меню налаштувань Onboarding |
| Очкуванний результат | Onboarding з поясненнями з’явився |
| Стан системи після тесту | Запустився Onboarding з поясненнями |

Таблиця 4.2

|  |  |
| --- | --- |
| Мета тесту | Створити пакет |
| Початковий стан системи | Пакету немає |
| Схема проведення тесту | Користувач натискає мнопку налаштування, вводить назву |
| Очкуванний результат | Створюеться пустий пакет |
| Стан системи після тесту | Створився пустий пакет |

Таблиця 4.3

|  |  |
| --- | --- |
| Мета тесту | Створити задвання і запонити пакет |
| Початковий стан системи | Пустий пакет |
| Схема проведення тесту | Тестер заходить до пакету, натискає додати задання, додає компоненти до завдання |
| Очкуванний результат | Створюеться завдання в пакеті |
| Стан системи після тесту | Створюеться завдання в пакеті |

Таблиця 4.4

|  |  |
| --- | --- |
| Мета тесту | Створити задвання і запонити пакет |
| Початковий стан системи | Пустий пакет |
| Схема проведення тесту | Тестер заходить до пакету, натискає додати задання, додає компоненти до завдання |
| Очкуванний результат | Створюеться завдання в пакеті |
| Стан системи після тесту | Створюеться завдання в пакеті |

Таблиця 4.5

|  |  |
| --- | --- |
| Мета тесту | Видаляти пакети |
| Початковий стан системи | Пустий пакет |
| Схема проведення тесту | Свайп вліво, натиснути кнопку видалити |
| Очкуванний результат | Пакет зник |
| Стан системи після тесту | Пакет зник |

Таблиця 4.6

|  |  |
| --- | --- |
| Мета тесту | Видаляти завдання |
| Початковий стан системи | Не пустий пакет |
| Схема проведення тесту | Свайп вліво, натиснути кнопку видалити |
| Очкуванний результат | Завдання видаляється і зникає |
| Стан системи після тесту | Завдання видаляється і зникає |

Таблиця 4.6

|  |  |
| --- | --- |
| Мета тесту | Видаляти копомнент |
| Початковий стан системи | Не пусте завдання |
| Схема проведення тесту | Свайп вліво, натиснути кнопку видалити |
| Очкуванний результат | копомнент видаляється і зникає |
| Стан системи після тесту | копомнент видаляється і зникає |

Таблиця 4.7

|  |  |
| --- | --- |
| Мета тесту | Авторизація |
| Початковий стан системи | Неавторизована система |
| Схема проведення тесту | Тестер заходить до магазину і пробує викласти свії пакет до магазину, вилазить авторизація, користувач авторизується |
| Очкуванний результат | Авторизація з’являється і тестер авторизується, стан системи стає авторизованим |
| Стан системи після тесту | Авторизація з’являється і тестер авторизується, стан системи стає авторизованим |

Таблиця 4.8

|  |  |
| --- | --- |
| Мета тесту | Копіювання пакету |
| Початковий стан системи | Авторизована система, тестер у магазині пакетів |
| Схема проведення тесту | Тестер заходить до магазину, вибирає пакет, свайп вліво, кнопка скопіювати |
| Очкуванний результат | Користувач скопіював пакет, він з'явився у головному меню, мін може його редагувати |
| Стан системи після тесту | Користувач скопіював пакет, він з'явився у головному меню, мін може його редагувати |

Таблиця 4.9

|  |  |
| --- | --- |
| Мета тесту | Викладка пакету |
| Початковий стан системи | Авторизована система, тестер у магазині пакетів |
| Схема проведення тесту | Нажати “Share” у магазині |
| Очкуванний результат | Пакет загружається до серверу, кожен інший користувач тепер може його побачити і скопіювати |
| Стан системи після тесту | Пакет загружається до серверу, кожен інший користувач тепер може його побачити і скопіювати |

Таблиця 4.11

|  |  |
| --- | --- |
| Мета тесту | Додати пакет у боті |
| Початковий стан системи | Пакету немає в боті |
| Схема проведення тесту | Закинути в бота пакет, реплаем зробити команду add |
| Очкуванний результат | Пакет додається до боту |
| Стан системи після тесту | Пакет додається до боту |

Таблиця 4.12

|  |  |
| --- | --- |
| Мета тесту | Видалити пакет у боті |
| Початковий стан системи | Пакету є в боті |
| Схема проведення тесту | Команда rm package\_index |
| Очкуванний результат | Пакет видаляється з боту |
| Стан системи після тесту | Пакет видаляється з боту |

Таблиця 4.13

|  |  |
| --- | --- |
| Мета тесту | Паказати список активних пакетів у боті |
| Початковий стан системи | Бот має якісь пакети |
| Схема проведення тесту | Команда pls |
| Очкуванний результат | Показує всі пакети в боті |
| Стан системи після тесту | Показує всі пакети в боті |

Таблиця 4.14

|  |  |
| --- | --- |
| Мета тесту | Надсилання повідомлення у боті |
| Початковий стан системи | Бот має якісь пакети |
| Схема проведення тесту | Дочекатись на повідомлення, Один з «Appear» компонентів повернув True |
| Очкуванний результат | Повідомлення прийшло до користувача |
| Стан системи після тесту | Повідомлення прийшло до користувача |

Таблиця 4.15

|  |  |
| --- | --- |
| Use case name | Надсилання повідомлення |
| Goals | Надати можливість надсилати повідомлення |
| Actors | Користувач |
| Trigger | Один з «Appear» компонентів повернув True |
| Pre-conditions | Додати до бота пакет |
| Flow of Events | Час приходить для показу задвання, приходить повідомлення на телефон з завданням |
| Extension | - |
| Post-Condition | Повідомлення прийшло |

## Розгортання програмного забезпечення

### iOS застосунок

Для того, щоб викласти iOS застосунок потрібно використовувати Xcode і App Store Connect.

App Store Connect є онлайн-платформою, розробленою компанією Apple, яка дозволяє розробникам програм для iOS, iPadOS, macOS, watchOS та tvOS керувати процесом розміщення та поширення своїх програм через офіційний App Store.

App Store Connect надає розробникам можливість вивантажувати свої програми, проводити тестування, встановлювати ціни, керувати версіями програм та здійснювати різні аспекти маркетингу. На цій платформі розробники також можуть відстежувати метрики продажів, аналізувати дані про користувачів та звітувати про доходи.

App Store Connect є важливим інструментом для розробників додатків Apple, оскільки вона надає доступ до широкої аудиторії користувачів Apple та допомагає в управлінні та просуванні їхніх програм у магазині.

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 4.1 — Архіви iOS застосунку

Загальні кроки, які я виконав, щоб розмістити застосунку в App Store.

* Зареєструвався в Apple Developer Program: Я зареєструвався в Apple Developer Program, щоб мати можливість розміщувати мої застосунки в App Store. Це дозволило мені отримати доступ до необхідних інструментів розробки і можливостей розміщення в App Store.
* Підготував застосунок: Я підготував свій застосунок для розміщення в App Store. Це включало в себе переконання, що мій застосунок відповідає всім вимогам та стандартам, встановленим Apple, і що він працює на платформах iOS, iPadOS, macOS, watchOS або tvOS.
* Зареєстрував застосунок в App Store Connect: Я зареєстрував свій застосунок в App Store Connect, платформі для керування застосунками в App Store. Тут я заповнив всі необхідні метадані, такі як опис, скріншоти, іконки та інші важливі дані про мій застосунок.
* Провів тестування: Я провів тестування мого застосунку, щоб переконатися, що він працює належним чином і не має помилок або проблем. Це включало в себе тестування функціональності, сумісності, безпеки та продуктивності застосунку.
* Встановив ціни та план розповсюдження: Я встановив ціни для свого застосунку, якщо він є платним, або обрав безкоштовний варіант. Також я обрав країни, в яких бажав розповсюджувати свій застосунок.
* Надіслав застосунок на перевірку: Після підготовки і налаштування всіх необхідних даних, я надіслав свій застосунок на перевірку Apple. Apple проводить ретельний технічний та контентний огляд, щоб переконатися, що застосунок відповідає їхнім вимогам та стандартам.
* Отримав схвалення та розмістив застосунок: Після успішного проходження перевірки, мій застосунок отримав схвалення від Apple, і я зміг розмістити його в App Store. Застосунок став доступним для користувачів для завантаження та встановлення.

A screenshot of a phone

Description automatically generated with low confidence

Рисунок 4.2 — вигляд iOS застосунку в AppStore

### Telegram Bot

Для того, щоб захостити бота я використовув Google Cloud

Google Cloud є хмарною платформою, розробленою компанією Google, яка надає широкий спектр послуг інфраструктури, обчислень, зберігання даних, аналітики, штучного інтелекту та інших послуг, що допомагають підприємствам розробляти, розгортати та керувати своїми додатками та сервісами в хмарному середовищі.

Google Cloud пропонує гнучкі та масштабовані рішення для хмарних обчислень, включаючи віртуальні машини, контейнери, сервери без обслуговування (serverless) та інші сервіси. Він також надає можливості зберігання даних, такі як об'єктне сховище, блокове сховище, бази даних та інструменти для аналітики даних.

Google Cloud також має широкий набір інструментів для розробників, включаючи сервіси управління API, інструменти для розробки мобільних додатків, машинного навчання та інтеграції з іншими сервісами Google, такими як Google Maps, Google Analytics та інші.

Ця платформа дозволяє організаціям будувати потужні, масштабовані та безпечні додатки та сервіси, використовуючи потужність хмарних ресурсів та інструменти, надані Google.

Я виконав такі кроки для того, щоб захостити бота на Swift

* 1. Взяв безкоштовних 200 доларів в місяць
  2. Створив server instance з Ubuntu
  3. Настроїв сервер, а саме скачав компілятор Swift, скачав код бота, скопіював
  4. Запустив бота

### Back-end програма

Як встановити серверну частину (Database + API) Встановлюються файли php та запускаються файли sql для створення таблиць та даних схеми бази даних.

Використовую вхід через адміністративну панель cPanel хостингу, підключитись до сервера хостингу https://server148.web-hosting.com/cpanel (або https://proq.cc/cpanel) username: x\*\*\*\*akr password: \*\*\*\*\*\*\*\*

У панелі керування cPanel запустити утиліту phpAdmin для створення схеми бази даних MySQL (найменування схеми: внутрішнє\_ім'я\_хостингу\_proq\_cc)

Створити схему бази даних та користувача бази даних з правами адміністратора командами CREATE DATABASE та CREATE USER, як це наводиться у скрипті /sql/proq-user.sql (або утилітою MySQL Database Wizard)

Запустити скрипт створення таблиць бази даних /sql/proq-schema.sql в утиліті phpAdmin (закладка SQL) або за допомогою програмного забезпечення Dbeaver

Запустити скрипт створення тестових даних для налагодження роботи бази даних /sql/proq-data.sql в утиліті phpAdmin (закладка SQL) або за допомогою програмного забезпечення Dbeaver

Виконати перенесення файлів /php/\*.\* у папку /HOME/service/api/ та зберегти на сервері (хостингу) за допомогою утиліти FileManager з адміністративної панелі cPanel

Виконати налаштування файлу /php/index.php (основний файл для запуску серверної програми для роботи API)

Редагування рядків змінних для підключення до бази даних (схема базхи даних, ім'я користувача та пароль) – вказати налаштування підключення. Для налагодження роботи увімкнути запис у файл протоколювання service.log (вказати значення true для параметра запису у файл лог, у перевійному кроці файлу index.php)

При необхідності налагодити роботу за допомогою запуску API запитів із рядка браузера, наведених у тестовому файлі /doc/readme\_api.txt

## Підтримка програмного забезпечення

### iOS застосунок

Будь-який користвуч, що встановив застосунок з AppStore має можливіть встановлювати оновлення. Оновлення викладуються за такою ж схемою як і просто викладка застосунку.

A screenshot of a phone

Description automatically generated with medium confidence

Рисунок 4.3 — Вигляд оновлення в AppStore

### Telegram Bot

Telegram Bot отримує оновлення не залежно від дій користувача, для цього потрібно просто загрузити нову версію програми до Google Cloud користувачі навіть нічого не помітять.

### Back-end програма

Back-end програма тримує оновлення не залежно від дій користувача, для цього потрібно просто загрузити нову версію програми до Google Cloud користувачі навіть нічого не помітять.

## Інструкція користувача

A screenshot of a phone

Description automatically generated with medium confidence

Рисунок 4.4 — Спливаюче вікно вводу назви для нового пакету

Натиснути New Package. Відкриється вікно для вводи назви пакету. Введіть назву, натисніть Save.

Screens screenshot of a phone

Description automatically generated with low confidence

Рисунок 4.5 — Вікна для налаштування пакету завдань

Натисніть на пакет, налаштуйте його так як потрібно. Додайте завдання, змініть назву, якщо потрібно.

Screens screenshot of a phone

Description automatically generated with medium confidence

Рисунок 4.6 — Вікна налаштування компонентів

Налаштуйте завдання так як Вам потрібно. Додайте потрібну кількість компонентів, налаштовуйте кожний окремий компонент

Screens screenshots of a phone

Description automatically generated with medium confidence

Рисунок 4.7 — Всі вікна налаштувань

В налаштуваннях можна подивитись інформацію про застосунок. Подивитись як підключити пакет до бота, вийти з профілю.

Screens screenshot of a phone

Description automatically generated with medium confidence

Рисунок 4.8 — Магазин і авторизація

Якщо Ви в додатку вперше, заходячі до магазина потрібно зареєструватися. Введіть дані і натисніть Done. Ви отримаєте доступ до користувацьких пакетів, скопіюйте той, який сподобається за допомогою жесту перетаскування пальцем в ліво. Якщо є бажання поділитися пакетом, натисніть кнопку Share і виберіть потрібний.

Screens screenshots of a phone

Description automatically generated with medium confidence

Рисунок 4.9 — Процес підключення пакету до бота

Відправте до бота пакет. Для цього довго натисніть на потрібний пакет і виберіть Connect з контекстного меню. Відправте пакет в телеграм боту. Якщо є бажання використовувати пакет разом з кимось, відправте пакет до групи, де є бот. У боті виберіть пакет і напишіть команду add.

Screens screenshots of a chat

Description automatically generated with medium confidence

Рисунок 4.10 — Команди бота

Додавайте пакетів до бота скільки потрібно. Видаляйте за допомогою команди rm<Index>. Отримуйте повідомлення від боту.

## Висновок до розділу

У розділі було розкрито процес завантаження та оновлення застосунків в App Store - офіційному магазині додатків для пристроїв. Цей магазин є єдиним законним місцем, де користувачі можуть завантажувати та оновлювати застосунки на своїх пристроях. Описав використання системи для користувачів

# ВИСНОВОК

Зроблено огляд предметної області автоматизації процесів часу людини. Проаналізовано існуючі рішення, виявлено їх проблеми і те як їх вирішити. Виявлено проблем при створенні подібних систем і як їх уникнути. Поставлено конкретні вимоги

Поставлена формалізована задача. Описані алгоритми роботи в програмі. Створено архітектуру системи. Поставлені задачі до реалізації систґми.

Описано засоби розробки. Проаналізовані технічні засоби і обґрунтовано їх використання. Описані методолгії, які вискористовуються в системах

Протестовано систему, виявлено та виправлено всі помилки. Розгорнуто всі системні частини, описано їх оновлення для користувача.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Apple Develper Documentation. URL: <https://developer.apple.com/develop/> (дата звернення 15.04.2023р)
2. Jonfir, IoC Article. URL: <https://jonfir.github.io/posts/ioc-ios/> (дата звернення 17.05.2023р)
3. Type-Driven Design in Swift. URL: <https://www.youtube.com/watch?v=pbVjkY9fS8c> (дата звернення 14.04.2023р)
4. PHP Documentation. URL: <https://www.php.net/docs.php> (дата звернення 14.04.2023р)
5. Apple WWDC23. URL: <https://developer.apple.com/wwdc23/> (дата звернення 12.05.2023р)
6. WWDC Notes. URL: <https://www.wwdcnotes.com/> (дата звернення 11.05.2023р)
7. Apple Swift Package Manager Documentation. URL: <https://docs.swift.org/package-manager/> (дата звернення 14.05.2023р)
8. Telegram Bot Documentation. URL: <https://core.telegram.org/bots/api> (дата звернення 05.05.2023р)
9. RxSwift -Как это работает?. URL: <https://www.youtube.com/watch?v=gV22Yy1Hxzw&t=1540s> (Дата звернення: 05.05.2023)
10. YouTube video: "iOS App Development Tutorial for Beginners". URL: <https://www.youtube.com/watch?v=3ReoqZCxg08> (Дата звернення: 29.05.2023)
11. GitHub repository: "Swift Design Patterns". URL: <https://github.com/eleev/swift-design-patterns> (Дата звернення: 05.05.2023)
12. YouTube video: "Introduction to SwiftUI". URL: <https://www.youtube.com/watch?v=RWrDahv8m0I&t=404s> (Дата звернення: 19.03.2023)
13. YouTube video: "Mastering SwiftUI". URL: <https://www.youtube.com/watch?v=iWYpFWQpvkA&t=2398s> (Дата звернення: 01.06.2023)
14. YouTube video: "Advanced iOS Development with SwiftUI". URL: <https://www.youtube.com/watch?v=vJikZvTfrdg&t=1950s> (Дата звернення: 15.05.2023)
15. YouTube video: "Building Interactive iOS Apps with SwiftUI". URL: <https://www.youtube.com/watch?v=AqrMPlzB8iI&t=1493s> (Дата звернення: 07.04.2023)
16. YouTube video: "SwiftUI Masterclass". URL: <https://www.youtube.com/watch?v=OD1uGsM1shg&t=2454s> (Дата звернення: 02.06.2023)
17. YouTube video: "Introduction to iOS App Development". URL: <https://www.youtube.com/watch?v=ewZ85qQIuYQ> (Дата звернення: 01.06.2023)
18. YouTube video: "iOS App Design and Development Tutorial". URL: <https://www.youtube.com/watch?v=B_T8o1vP4H8> (Дата звернення: 28.05.2023)
19. YouTube video: "SwiftUI Crash Course". URL: <https://www.youtube.com/watch?v=aSoWDKonXew&list=PLJfJvphK-POx-wz_e9vcMnY3IhHi9MSwH&index=2> (Дата звернення: 25.04.2023)
20. Article: "iOS Navigation Best Practices". URL: <https://frankrausch.com/ios-navigation> (Дата звернення: 15.03.2023)
21. Website: "Refactoring Guru - Design Patterns". URL: <https://refactoring.guru/design-patterns/> (Дата звернення: 30.05.2023)
22. GitHub repository: "DesignPattern" by artkirillov. URL: <https://github.com/artkirillov/DesignPattern/> (Дата звернення: 17.04.2023)