**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

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ р.

**Дипломний проєкт**

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Інформаційні управляючі системи та технології»**

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

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

Виконав (-ла):

студент (-ка) IV курсу, групи ІК-91

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

Керівник:

Доцент кафедри ІСТ, Солдатова Марія Олександрівна \_\_\_\_\_\_\_\_\_\_

Рецензент:

Жаріков Едуард В'ячеславович, д.т.н., доц. каф. ІПІ \_\_\_\_\_\_\_\_\_\_

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

Студент (-ка) \_\_\_\_\_\_\_\_\_\_\_\_\_

Київ – 2023 року

**Пояснювальна записка**

**до дипломного проєкту**

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

Київ – 2023 року

АНОТАЦІЯ

*Зм.*

*Арк.*

*Прізвище*

*Підпис*

*Дата*

*Лист*

2

*ДП 7101.00.000 ПЗ*

*Розроб.*

*Іванов І.І.*

*Перевірив.*

*.*

ПІБ керівникаДП

*Н. кон.*

ПІБ н/контр*.*

*Затв.*

ПІБ керівникаДП

Тема

*Літ.*

*Листів*

–

*КПІ ім. Ігоря Сікорського*

*Каф. АСОІУ*

*Гр. ІС-71*

**Структура та обсяг роботи.** Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить ?? рисунків, ?? таблиць, ?? додатків, ?? джерел.

Дипломний проєкт присвячений розробці комплексу задач … . Цілі та задачі розробки.

У розділі інформаційного забезпечення …

Розділ математичного забезпечення присвячений…

Програмне забезпечення …

У технологічному розділі …

|  |
| --- |
| ТУТ МАЄ БУТИ ПЕРЕЛІК КЛЮЧОВИХ СЛІВ: ВЕЛИКИМ ЛІТЕРАМИ, ЧЕРЕЗ КОМУ. |

ABSTRACT

*Той самий текст, що й в анотації, але іноземною мовою (англійська, німецька, НЕ російська)*

ЗМІСТ

[ВСТУП 7](#_Toc136375558)

[1 Предметна область і аналоги 10](#_Toc136375559)

[1.1 Огляд предметної області 10](#_Toc136375560)

[1.2 Аналіз існуючих рішень 11](#_Toc136375561)

[1.2.1 Огляд Structured 12](#_Toc136375562)

[1.2.2 Огляд Routinery 15](#_Toc136375563)

[2 ВИБІР ТА ОБҐРУНТУВАННЯ КОМПОНЕНТІВ ТА ТЕХНОЛОГІЙ 19](#_Toc136375564)

[2.1 Засоби розробки 19](#_Toc136375565)

[2.2 Вимоги до технічного забезпечення 21](#_Toc136375566)

[2.2.1 Загальні вимоги 21](#_Toc136375567)

[2.3 Архітектура програмного забезпечення 22](#_Toc136375568)

[*2.4* Аналіз існуючих технологій 22](#_Toc136375569)

[2.4.1 Mobile App. 23](#_Toc136375570)

[2.4.2 Бібліотека компонентів 28](#_Toc136375571)

[2.4.3 Telegram Bot 28](#_Toc136375572)

[*2.5* Аналіз відомих алгоритмічних та технічних рішень 30](#_Toc136375573)

[2.5.1 iOS додаток 30](#_Toc136375574)

[*2.6* Аналіз допоміжних програмних засобів та засобів розробки 33](#_Toc136375575)

[3 Математичне забезпечення 36](#_Toc136375576)

[3.1 Змістова постановка задачі 36](#_Toc136375577)

[3.2 Математична постановка задачі 36](#_Toc136375578)

[3.2.1 Система компонентів 36](#_Toc136375579)

[3.2.2 Алгоритм роботи з пакетами завдань 36](#_Toc136375580)

[3.3 Обґрунтування методу розв’язування 38](#_Toc136375581)

[3.4 Висновок до розділу 39](#_Toc136375582)

# ****ВСТУП****

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 нам максимально використовувати наш потенціал і досягати поставлених цілей.

Незважаючи на це, багато людей справляються зі складнощами у плануванні, організації та ефективному використанні свого часу. Втрачений час, невиконані завдання та постійний стрес можуть впливати на нашу продуктивність, задоволення від життя та загальний стан самопочуття.

З цією метою, я вирішив робити проект з організації часу людини, який спрямований на надання необхідних інструментів та підходів для ефективного управління часом.

Важливо підкреслити, що мій проект не має на меті нав'язувати універсальний підхід до організації часу, але замість цього ставить за мету надати людям інструменти, які дозволять кожному індивідуалізувати свій підхід залежно від особистих потреб, пріоритетів та стилю життя.

Підставою для створення проекту стала актуальна проблема відсутності гнучких інструментів організації часу людини на ринку. Головною метою проекту є створення простого способу організувати свій дозвілля так, як бажає людина, накопичувати цінний досвід і ділитись ним з спільнотою.

Для досягнення цієї мети, проект має наступні завдання:

1. Розробити інтуїтивно зрозумілий інтерфейс, який дозволить користувачам легко планувати свій час і дозвілля.
2. Забезпечити можливість користувачам зберігати свій досвід та накопичувати влучні поради та рекомендації в особистому профілі.
3. Створити функціонал, який дозволить користувачам ділитись своїми досвідом і порадами з іншими учасниками спільноти.
4. Забезпечити механізми зворотного зв'язку та підтримки, щоб користувачі могли спілкуватись, обмінюватись ідеями та вдосконалювати проект разом.

Проект має на меті вирішити проблему неефективного використання часу і надати людям інструменти, які допоможуть їм більш насолоджуватись своїм дозвіллям, розвиватись та ділитись цим досвідом з іншими.

Одна з великих проблема мозку – обмежена оперативна пам’ять. В нашій повсякденному житті утримати, що потрібно робити просто неможливо.

Ця обмежена оперативна пам'ять нашого мозку створює значні перешкоди в ефективному утриманні всіх необхідних завдань та інформації у свідомості. Ми постійно зіткнуємося зі ситуаціями, коли забуваємо про важливі речі або не можемо відтворити точну послідовність подій. Наш мозок працює, намагаючись відсікати незначні деталі і зосереджуватися на найбільш суттєвих аспектах, але це часто призводить до втрати важливої інформації.

Намагаючись утримати все, що потрібно робити у своїй голові, ми стикаємося з перенавантаженням та втомою, що негативно впливає на нашу продуктивність та концентрацію. Саме тому знаходження ефективних стратегій для керування і обробки інформації стає насущною задачею.

Однак усі завдання мають в собі повторювальні дії, які якщо чесно виконувати, можно досягнути успіху будь-де. Тому потрібно лише дати конструктор для того, щоб його можно було збирати будь-кому і ділитися нескінченними копіями вдалих «творінь» з будь-ким

# Предметна область і аналоги

## Огляд предметної області

### Сфера

#### Огляд

Сферою проекту є організація часу людини. Тут існує велике розмаїття напрямків та систем, починаючи від простих "туду-листів" (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, орієнтовані вони на різні фокусні групи людей.

Наприклад, 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 софту для клієнтів, які зацікавлені у кращому управлінні своїм часом.

Організаційний софт в основному виконує алгоритмічну роботу з даними та створює умови для ефективного сприйняття і управління часом. Незалежно від цілей і масштабів, всі системи сводяться до найменшого "юніта" - завдання або таску.

Мій проект працює за моделлю "від продукту до клієнта" (Project to Client) і спрямований на покупців. Однак, система відрізняється від інших проектів деякими особливостями, які будуть надалі вказані. Моя задача полягає в розробці потужного інструменту для розгрузки мозку, оптимізації рутинних завдань, з накопичувальним ефектом і сильною інтеграцією з спільнотою однодумців.

#### Рішення та специфіка платформ

Проекти, спрямовані на бізнес і організацію бізнес-процесів, створюють свої продукти у вигляді нестольних застосунків чи веб-програм з не мінімалістичним інтерфейсом і великою кількістю функцій для планування, аналізу ефективності тощо. Хоча існують виняткові випадки, універсальною вимогою для настільних або веб-планувальників є модель БТБ (бізнес-до-бізнесу) з великою кількістю функцій, які важко використовувати на малих екранах.

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

#### Щодо мого проекту

Мій проект в другим типом з пункту 1.1.1.2. Я буду використовувати мобільну платформу iOS і Telegram Bota для забезпечення максимальної мобільності

### Ідеологія проекта

Проект має open-source ідею як основу і робить велику ставку на спільноту сервісу. Так як це конструктор, то потрібно зробити все, можно було ділитися своїми напрацюваннями. Спільнота буде їх оцінювати і алгоритм програми виводити найкращі рішення.

### Які проблеми вирішує проект

* Створення компонентного конструктора для користувачів.
* Створення інструментів для поширення напрацювань
* Створення системи заохочення найкращих рішень

## Аналіз існуючих рішень

Справді, на ринку існує велика кількість програмного забезпечення для організації часу. Це означає, що конкуренція серед розробників є дуже високою, і для того, щоб виділитись, потрібно мати щось особливе.

Деякі програми можуть зосередитись на інтуїтивно зрозумілому і простому інтерфейсі, що дозволяє користувачам швидко оволодіти програмою і почати організовувати свій час без складнощів. Інші можуть пропонувати інноваційні методи планування, такі як графіки Ганта або канбан-дошки, які дають змогу візуально відстежувати прогрес та керувати завданнями.

У наступних розділах ми розглянемо два найпопулярніші додатки для організації часу на сьогоднішній день, а також проведемо порівняльний аналіз їх основних характеристик, таких як підтримувані пристрої, цілі, ідеології, простоту керування процесами.

Деякі програми акцентуються на синхронізації та доступності даних. Вони можуть пропонувати можливість синхронізувати розклади та завдання між різними пристроями та платформами, такими як комп'ютери, смартфони та планшети, щоб мати постійний доступ до оновленої інформації.

Деякі програми можуть спеціалізуватись на конкретних групах користувачів або виконувати специфічні завдання. Наприклад, є програми, призначені для менеджменту проектів, нагадувань про медичні препарати, трекінгу часу роботи та багато інших.

Окрім цього, оновлення та підтримка програмного забезпечення є важливим аспектом. Розробники повинні надавати регулярні оновлення з новими функціями, виправленнями помилок та забезпечувати гарантовану підтримку користувачів, щоб забезпечити задоволення і лояльність клієнтів.

### Огляд Structured

Structured - це щоденний візуальний планувальник, який поєднує календар і список справ, щоб надати можливість розплановувати завдання, стежити за прогресом протягом дня та позначати їх як виконані, коли завдання виконані.

Сервіс, доступним лише на пристроях від Apple. Він може бути використаний на iPhone, iPad, та пристроях, що працюють під управлінням операційної системи macOS. Це означає, що користувачі, які володіють пристроями Apple, можуть насолоджуватися функціональністю та, які надає Structured для планування та відстеження своїх завдань та розкладу.

Сервіс Structured, на жаль, не має базового функціоналу для організації рутини людини, хоч якось працююча версія коштує грошей. Бізнес схема в програми геть погана.

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

#### Функції додатку

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 розплановувати завдання на календарі. Ви можете встановлювати терміни виконання, пріоритети та тривалість кожного завдання.
2. Список справ: Structured надає можливість створювати список справ, який включає окремі завдання або підзавдання. Ви можете легко організовувати свої завдання за категоріями або проектами.
3. Візуальне планування: За допомогою Structured ви можете використовувати візуальний планувальник для відстеження свого розкладу та завдань. Ви можете бачити ваші завдання на календарі і переміщати їх за допомогою перетягування, змінюючи їх дати та часи.
4. Нагадування та сповіщення: Structured дозволяє встановлювати нагадування про наближення термінів виконання завдань або про важливі події. Ви отримуватимете сповіщення, щоб не пропустити жодне важливе завдання.
5. Відстеження прогресу: Застосунок надає можливість відстежувати прогрес виконання завдань. Ви можете позначати завдання як виконані, встановлювати статуси або використовувати вбудовані мітки для класифікації завдань.
6. Синхронізація даних: Structured забезпечує синхронізацію даних між пристроями, що дозволяє вам отримувати доступ до своїх завдань та розкладу з різних пристроїв, таких як смартфони або планшети.
7. Ці функції дозволяють вам більш ефективно організувати свій час, планувати завдання та відстежувати їх виконання, забезпечуючи більшу продуктивність та організованість у вашому повсякденному житті.

#### Порівняння

Structured і мій проект мають деякі схожі функціональні можливості, але також відрізняються за деякими аспектами:

1. Пакети завдань: Обидва проекти дозволяють організовувати завдання в пакети або групи, що допомагає структурувати роботу та керувати комплексними проектами.
2. Групи користувачів: ProQ дозволяє додавати пакети до груп користувачів, що може бути корисно для спільної роботи над проектами або для делегування завдань між учасниками –коли викинути сміття чи помити посуду потрібно знати всім.
3. Відстеження прогресу: Як Structured, так і ProQ проект надають можливість відстежувати прогрес виконання завдань, але ProQ надає інструмент для аналітичної і інтерактивної роботи з данними, а система інтерактивних компонентів дає можливість нескінченно розширювати можливості користувчів взаємодіяти з інформацією
4. Компонента система дає можливість дуже гнучкі можливості для настройки завдань, а постійні оновлення поточного набору різних компонентів дають великий набір інструментів для настройки. Structured має дуже скромний і не зрозумілий для користувача інструментарій для створення рутин. Меня не подобається розуміти завдання як набір якихось настройок, в ProQ в свою чергу це завдання на яких накинуті налаштовані компоненти, це просто зрозуміло с точки зору сприйняття
5. Робота зі списками даних: Описаний проект має можливість працювати зі списками даних та алгоритмічно їх прикріпляти до завдань. Це може бути корисно для обробки та аналізу інформації, пов'язаної з конкретним завданням.
6. Запис даних до завдань: В описаному проекті користувачі можуть записувати дані до завдань. Це дозволяє включати додаткову інформацію, коментарі, примітки або відповіді, які можуть бути оброблені компонентами, пов'язаними з завданням.
7. Інтерфейс виконан дуже погано, цим застосунком не приємно користуватись, в своїх розробках я використовую переводі технології Apple для того, щоб добитись плавної і крутої роботи системи

### Огляд Routinery

Routinery - це додаток для організації рутини, який надає широкий функціонал для планування, відстеження та керування рутинними завданнями. Основні функції Routinery включають:

1. Створення рутин: Ви можете створити свої власні рутини залежно від потреб і пріоритетів. Наприклад, рутини для ранкових зарядок, щоденного тренування, підтримки чистоти вдома тощо.
2. Гнучке планування: Routinery дозволяє вам гнучко планувати свої рутини на різні дні тижня або на конкретні дати. Ви можете встановити повторюваність рутини щоденно, щотижня або вибрати власний графік.
3. Нагадування і сповіщення: Додаток Routinery надсилає нагадування та сповіщення, щоб вам не пропустити важливі рутинні завдання. Ви отримуватимете повідомлення на своєму смартфоні або інших пристроях, щоб бути в курсі поточних рутин.
4. Відстеження прогресу: Routinery дозволяє вам відстежувати прогрес у виконанні рутин. Ви можете позначати завдання як виконані або відмічати їх у разі затримки. Це допоможе вам бачити вашу продуктивність та дотримання рутин.
5. Спільний доступ: Routinery дозволяє спільний доступ до рутин між користувачами. Ви можете створювати групи рутин та додавати учасників, що спільно виконують їх. Це корисно, коли потрібно скоординувати рутини з родиною, співмешканцями або колегами.
6. Статистика і звіти: Routinery надає статистику та звіти щодо вашої рутини. Ви можете переглядати час, проведений на кожному завданні, аналізувати свою продуктивність та виявляти зв'язки між рутинами та вашим самопочуттям.
7. Кастомізація та інтеграція: Додаток Routinery надає можливість налаштовувати і персоналізувати рутини згідно з вашими вподобаннями. Ви можете додавати зображення, описи, кольори та інші елементи, що допомагають вам організувати рутину. Також, Routinery може інтегруватись з іншими додатками або сервісами, такими як календарі, напоминалки та інші, для ще більшої ефективності і зручності.

Routinery і ProQ мають декілька схожих функціональних можливостей. Проте, як я вже зазначив, у Routinery може бути потенціал, але він також має проблеми на фундаментальному рівні - незадовільний інтерфейс, відсутність нормального магазину, відсутність компонентної системи і інтерактивності, а також відсутність спільного користування завданнями.

Незважаючи на потенційні переваги проекту, він не надає справжньої цінності, яка дійсно корисна для мого життя. Важливо не просто створити програму, але також розуміти, для чого вона потрібна, які завдання вона вирішує і забезпечувати їх бездоганне виконання.

Проект може мати певний функціонал, але який сенс від нього, якщо він незрозумілий і використовувати його неприємно? Навіть якщо змушувати себе користуватись ним, він не задовольняє потреби користувача, незалежно від того, які цілі він має.

## Функціональні вимоги до проекту

Головною функція застосунку є створення пакетів завдань, доступ до магазина пакетів, у майбутньому новини і поради

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 | Відкрити меню налаштувань |
| Goals | Надати користувачу доступ до налаштувань застосунку |
| Actors | Користувач |
| Trigger | Користувач натискає мнопку налаштування |
| Pre-conditions | - |
| Flow of Events | Користувач натискає мнопку налаштування |
| Extension | - |
| Post-Condition | Користувач перейшов в налаштування |

|  |  |
| --- | --- |
| Use case name | Відкрити меню налаштувань |
| Goals | Надати користувачу доступ до налаштувань застосунку |
| Actors | Користувач |
| Trigger | Користувач натискає мнопку налаштування |
| Pre-conditions | - |
| Flow of Events | Користувач натискає мнопку налаштування |
| Extension | - |
| Post-Condition | Користувач перейшов в налаштування |

### Функціональні вимоги

### Нефункціональні вимоги

## Висновок до розділу

В цілому всі аналоги мають завжди декілька проблем, які не дають того досвіду користування, який мав би реальний вплив на життя людини, а саме:

* Поганий інтерфейс. Є документація від Apple і Google я робити гарний інтерфес, навіщо кожний раз видумувати щось своє. Хоча насправді можно піти складним шляхов і розробити свою дизайн систему, але це коштує дуже багато грошей, на жаль, такі проекти просто мають якийсь свій псевдо дизайн, який відчувається дешевого і не дає задоволення
* Відсутність чіткого розуміння як розвивати проект і його іделогії і цілі, які він вирішує. В такі додатки можуть додавати все що завгодно (у Structured, наприклад є рутинні завдання і одноденні, є можливість роботи нотатки спонтанних думок). Навіщо це все потрібно коли сам сервіс працює погано
* Всі аналоги не мають можливості додавати завдання до груп користувачів. Це просто вбиває майже всю суть
* Аналоги або мають дуже простий магазин для пакетів завдань чи зовсім не мають. Тобто ніяка або дуже поверхова робота з спільнотою системи. Це недопустимо, на мій погляд і не дає розквітнути ідеї автоматизації нагадувань в повній мірі
* Інструментарій аналогів дуже простий і обмежується повторювальними нагудвалками і додатковим описом їх. Тобто ніякої роботи з даними тут немає. Компонента система для таких додатків це «ноу-хау», де я можу роботи гнучкі завдання на любий смак

# ВИБІР ТА ОБҐРУНТУВАННЯ КОМПОНЕНТІВ ТА ТЕХНОЛОГІЙ

## Засоби розробки

При розробці даної інформаційної системи використовувались багато  
технічних засобів. Найважливіші з них будуть наведені у таблиці 4.1

|  |  |  |
| --- | --- | --- |
| Тип | Назва | Версія |
| Мова програмування | Swift, PHP, SQL, | 5.7/4.2.4/- |
| Фреймворки | SwiftUI, UIKit, TelegramSDK, Foundation |  |
| IDE | Xcode, DBvaer |  |
| RDBMS | MySQL |  |
| Засіб проектування | Figma |  |
| Система контроля версій | Git |  |
| Репозиторій | GitHub |  |

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.

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.

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.

## Вимоги до технічного забезпечення

### Загальні вимоги

#### Вимоги до серверу, на якому буде розгорнуто Telegram Bot-a

* процесор з частотою не менше 2 ГГц;
* RAM об’ємом 2 гб;
* 2 ГБ доступного місця на жорсткому диску

#### Вимоги до серверу, на якому буде розгорнуто серверну програму

* процесор з частотою не менше 2 ГГц;
* RAM об’ємом 2 гб;
* 2 ГБ доступного місця на жорсткому диску

#### Вимоги до користувача

##### Telegram

* iPhone, який може запустити Telegram. Не нижче iPhone 5

##### iOS застосунок

* iPhone, який с iOS не нижче 13 версії, тобто не нижче iPhone 6 та iPhone 6 Plus

## Аналіз існуючих технологій

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 моїх власних досвіду і преференціях, і не можуть бути розглянуті як загальноприйнята істина.

Мій особистий досвід та уподобання включають такі фактори, як:

* Знання та досвід з певними технологіями: 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: Якщо певні інструменти або мови програмування викликають у мене відчуття комфорту та продуктивності, це може вплинути на мій вибір технологій.
* Уподобання щодо стилю програмування: Якщо певна технологія відповідає моєму стилю програмування та підходу до розробки, це може бути вагомим фактором при виборі.

Однак, наряду з особистими уподобаннями, я також враховую вимоги моєї системи, включаючи:

* Функціональність: Я оцінюю, чи задовольняють обрані технології всі функціональні вимоги мого застосунку.
* Перформанс: В залежності від потреб моєї системи, я оцінюю продуктивність і швидкодію технологій, щоб забезпечити оптимальну роботу застосунку.
* Сумісність: Я переконуюся, що обрані технології сумісні з моєю цільовою платформ

Для написання застосунка для мобільних пристроїв у вас є декілька варіантів технологій: кросплатформені рішення та нативні. Кожен з цих варіантів має свої переваги і недоліки, і вибір залежатиме від ваших потреб, вмінь та обмежень проекту. Ось огляд кожного з цих варіантів:

Мій проект це декілька програм працюючих разом, а саме:

* Застосунок під iOS. Застосунок гнучка система, яка може мати будь-які функції у майбутньому. На першому етапі розробки проекту, застосунок буде мати дві основні функції: робити пакети завдань і показувати магазин пакетів
* Бот. Насамеред бот, який буде використовується виключно для нотифікацій користувача у спільних чи одиночних чатів. Основний акцент розробки бота полягатиме у вдосконаленні та оптимізації його існуючого функціоналу, а не в додаванні нових можливостей.
* Бібліотека компонентів. Повинна мати можливість бути імпортованою у бота і iOS застосунок. Має архітектуру компонентів і їх реалізації
* Бекенд. Щось найшвидше, найпростіше і найпопулярніше. У першій стадії проекту зберігає користувачів і пакети магазину

### Mobile App.

#### Платформа

1. Кросплатформені рішення:
   * **React Native**: React Native дозволяє вам використовувати JavaScript для розробки мобільних додатків для iOS та Android. Ви можете використовувати загальний код для обох платформ, що зберігає час і зусилля.
   * **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.

Кросплатформені рішення дозволяють ефективно використовувати загальний код для різних платформ, що зменшує зусилля, але можуть мати деякі обмеження в доступі до платформено-специфічних функцій та продуктивності.

1. Нативний розробка:
   * **iOS (Swift/Objective-C)**: Для розробки нативних додатків для iOS ви можете використовувати мови програмування Swift або Objective-C. Це надає повний доступ до всіх функцій і можливостей, пропонованих платформою Apple.
   * **Android (Java/Kotlin)**: Для розробки нативних додатків для Android ви можете використовувати мови програмування Java або Kotlin. Вони також забезпечують 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: 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: 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: 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: 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: 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: 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: 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: 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. Вона має потужну runtime бібліотеку, потому за допомогою інтероп у зі Swift, з нього можна зробити дуже цікаві функціональні речі. Або прострілити собі ногу. Я беру стандарт – Swift

#### Нативний фреймворк. 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
2. 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 фреймворки, 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 дозволяє досягти найкращого з обох світів, забезпечуючи 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.

### Бібліотека компонентів

Для розробки бібліотеки з логікою, яка буде взаємодіяти зі Swift, ви маєте 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 вам потрібна максимальна продуктивність або ви вже маєте існуючий код на мовах C або C++, ви можете 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 своєму Swift-коді.
3. Інші мови: Залежно від вашого випадку використання, ви також можете використовувати інші мови програмування, які підтримують міжмовний інтерфейс зі Swift, такі як Objective-C, Python, Ruby та інші. За допомогою спеціальних інтерфейсів, які надає Swift, ви зможете інтегрувати функціональність цих мов у свою бібліотеку.

Так як я роблю додаток і бота на Swift, робити бібліотеку також на цій мові мені задалось гарним варіантом.

### Telegram Bot

#### 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.

## Аналіз відомих алгоритмічних та технічних рішень

### iOS додаток

Для додатку 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 iOS розробки існує безліч архітектурних паттернів, які називаються “архітектурні паттерни”. Фактично вони базуються на тих самих MVC і Apple MVC, але розбивають їх на більші компоненти, змінюють передачу даних між компонентами на реактивний підхід або включають логіку навігації в модуль вікна. Варто зауважити, що логіка навігації є окремою схемою і окремою частиною програми, а модулі не залежать від неї і нічого про неї не знають. Таким чином, цей шар знаходиться нижче за логіку навігації.

У світі iOS розрізняють такі архітектурні паттерни:

1. MVP (Model-View-Presenter): В цьому паттерні ізнес-логіка витягується до сутності P (Presenter). Це робочий варіант, який поліпшує стандартний MVC від Apple і не перевантажує систему окремого модуля.
2. MVVM (Model-View-ViewModel): В цьому паттерні бізнес-логіка витягується до сутності VM (ViewModel), а передача даних та подій між екраном і ViewModel здійснюється за допомогою реактивного підходу. В стандартному iOS це означає використання RxSwift або Combine. Це також ефективний варіант, особливо для тих, хто вподобає реактивний підхід. Проте я не бачу в ньому великого сенсу, оскільки Apple випустила нову бібліотеку для цього, яка ще нестабільна, а RxSwift, хоч і підходить, скоро вийде з використання. Це складний вибір, який зараз не відповідає моїм потребам, ти більше, фанат реактивного підходу я також не є , тому за особистих вподобань не мій вибір.
3. VIPER: Це паттерн, в якому навігаційну логіку поміщають у R (Router), інтерактор виконує запити до сервера і передає дані презентеру, а презентер оновлює View, яка займається юзер-інтерфейсом. Це дуже перевантажений паттерн, для якого важко писати код, оскільки він має в собі логіку навігації, що робить його недостатньо перевикористовуваним і не заснованим на принципах DRY (Don't Repeat Yourself). Крім того, інтерактор не є обов'язковою частиною, його можна просто передавати через контрактний інтерфейс у бізнес-логіку презентера або ViewModel. Це не найкращий варіант, на мою думку.

Варто враховувати, що кожен паттерн має свої переваги і недоліки, і вибір залежить від конкретної ситуації та ваших особистих вподобань.

Мій варіант MVP і самописний компоновищк, для того щоб можно било зніювати логіку для окн на ходу

#### 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

Рис 1 – генеральна схема шарів застосунку

Вищим рівнем є навігація, яка використовує модулі і будує дерево програмного потоку. Сутності, відповідальні за навігаційну логіку (називаються координатори), є стандартом в iOS. Вони використовують модулі та вказують, коли вони повинні з'являтися. Координатори можна перевикористовувати в будь-якому місці.

* Бізнес-логіка залежить тільки від інтерфейсів UI, що означає, що реалізація може бути виконана ким завгодно.
* View (окна) залежать лише від UI-фреймворку.
* Networking є шаром, який використовується для отримання даних і залежить тільки від бібліотек Apple для мережевого взаємодії.
* ProQ-iOS-Core - це набір різних утиліт та бібліотек для роботи, який може включати як UI-бібліотеки, так і архітектурні бібліотеки.

#### **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. **Збереження стану та навігації**: Координатори дозволяють зберігати стан навігації та здійснювати перехід між екранами з дотриманням правил та логіки додатку.
4. **Спрощення тестування**: Координатори полегшують тестування окремих модулів, оскільки їх можна відокремити від решти додатку та незалежно тестувати їх функціональність.

#### **Впровадження залежностей у модулі**

Для внедрення залежностей існують три підходи: Service Locator, Dependency Injection (DI) і фабричний метод.

У моїй програмі, яка має невеликий обсяг, я буду використовувати іноді фабричний метод, DI і Service Locator, який подібний до фабричного методу. Однак ці підходи більш підходять для великих проектів, де є потреба у більш гнучкому управлінні залежностями.

## 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

# математичне забеспечення

## Змістова постановка задачі

* Створення розширюваної компонентної системи, на базі якої можна створювати нові компоненти
* Створення пакетів завдань. Можливість редагувати пакет, кількість завдань
* Загрузка пакетів до програми, яка буду їх оброблювати і присилати сповіщення

## Математична постановка задачі

### Система компонентів

На мій погляд найскладніший етап проектування всього проекту

#### 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….kk обробників, які в свою чергу мають l…ll 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 дуже високорівневою і не надає програмісту таких структур даних, як "масив (array) з С/С++", який забезпечує найбільшу швидкість пошуку елементів в ньому, оскільки це всього лише операція додавання i+index, плюс не потрібно аллокувати пам'ять на купі (heap), що є дорогою операцією для операційної системи. Проте масив у Swift є досить оптимізованим і дуже схожим за реалізацією на "вектор з С++".

У моєму конкретному випадку він підходить більше, оскільки він динамічно розширюється, а це є критичним для мене, оскільки я не знаю, скільки може бути користувачів або пакетів. Динамічність тут є критичною властивістю. Для більшої оптимізації мені потрібно правильно встановити властивість "capacity", щоб мінімізувати шанс аллокації нової пам'яті, коли розмір масиву наближається до кінця, а Swift Array автоматично аллоціює пам'ять вдвічі.

У деяких випадках, де послідовність даних не має значення для мене, я буду використовувати множину Array Set, просто для зберігання даних в пам'яті. Вона є швидшою, ніж Swift Array. Наприклад, для збереження координаторів або пакетів у боті.

Також корисним буде використання хеш-мапи, яка у Swift називається словник (Dictionary). Ключем буде ідентифікатор користувача у Telegram, а значенням - сам пакет. Це найкращий випадок використання..

## Висновок до розділу

# АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Опис процесів тестування

## Контрольний приклад

## Висновок до розділу

# ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Розгортання програмного забезпечення

### iOS застосунок

### Telegram Bot

### Back-end програма

## Підтримка програмного забезпечення

### iOS застосунок

### Telegram Bot

### Back-end програма

## Висновок до розділу