**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»**

***Спеціальність: 121 «Комп'ютерна інженерія»***

***Освітня програма: «Розробка програмного забезпечення»***

***Група: 4РП-07***

# Дипломний проєкт

**здобувача освіти денної форми навчання**

**РП.07.22.000.ДП**

***ТУРКОЛА ОЛЕКСІЯ ДМИТРОВИЧА***

**м. Одеса**
**2024 р**.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Комп'ютерна інженерія»**

Освітньо-професійна програма: ***«Розробка програмного забезпечення»***

Група: **4РП-07**

# ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту на тему:

## Розробка та дизайн мобільного програмного забезпечення для творчих виступів

Проєктний матеріал складається з пояснювальної записки на **96** сторінках та графічного (презентаційного) матеріалу на **12** аркушах (слайдах).

Дипломник _____ ( Туркол О. Д. )

Керівник _____ ( Закроев Ю.М. )

**Консультанти:**

з економічного розділу _____ ( Іванченков В. С. )

з розділу охорони праці та техніки безпеки _____ ( Чорновол Н. І. )

з нормоконтролю _____ ( Петрашова В. І. )

старший консультант _____ ( Кривченко Ю. В. )

**До захисту допущений**

Голова циклової комісії _____ ( Кривченко Ю. В. )

Завідувач відділення _____ ( Скорнякова О. В. )

Захист «26» 06 2024 р. Протокол ЕК № 3

Оцінка ЕК 4 (добре) / 75

Секретар ЕК _____

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»**

Відділення *ком'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *121 – «Комп'ютерна інженерія»*

Освітня програма *«Розробка програмного забезпечення»*

**ЗАТВЕРДЖУЮ**

Заст. дир. з НВР *Беркань І. В.*

« 15 » 11 *2024* року

**З А В Д А Н Н Я**

**на дипломний проєкт (роботу)**

*Турколу Олексію Дмитровичу*

1. Тема проєкту (роботи) *Розробка та дизайн мобільного програмного забезпечення для творчих виступів*

Затверджена наказом по коледжу від « 02 » 11 *2024 р.*, наказ № 244-А2-02

2 Термін здачі закінченого проєкту (роботи) 10.06.24

3. Вихідні дані до проєкту (роботи)

*Розробити мобільний застосунок, використовуючи Flutter та мовою програмування Dart*

*Розробити алгоритм роботи засобами мови програмування Dart*

*Має бути меню налаштувань для варіативності виступів і можливість зберігати шаблони роботи для майбутніх виступів*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

*Вступ. Аналіз предметної галузі; Огляд існуючих рішень, Технології та засоби розробки Технічне завдання на розробку; Проєктування дизайну застосунку, Реалізація застосунку Створення алгоритму, Створення головного меню; Створення панелі налаштувань, Створення усіх екранів ,Економічний Розділ, Аналіз та безпека умов праці працівника на робочому місці*

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

*Презентація Power Point – 12 слайдів*

*Вступ.Засоби розробки мобільного застосунку.Етапи створення мобільного застосунку. Структура мобільного застосунку. Структура головної сторінки. Структура сторінки налаштувань .Структура "робочих" екранів. Логіка роботи.*

6. Консультанти по проєкту, із зазначенням розділів проєкту, що їх стосується

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| Основний розділ | *Закроєв Ю.М* | | |
| Економічний розділ | *Іванченков В. С.* | | |
| Розділ охорони праці | *Чорновол Н. І.* | | |
| Нормоконтроль | *Петрашова В. І.* | | |
| Старший консультант | *Кривченко Ю. В.* | | |

7. Дата видачі завдання 15.01.2025

Керівник *Закроев Ю.М.* (підпис)

Завдання прийняв до виконання *Туркол О.Д.* (підпис)

## КАЛЕНДАРНИЙ ПЛАН

| № з/р | Назва етапів дипломного проєкту (роботи) | Термін виконання етапів дипломного проєкту (роботи) | Відмітка про виконання |
|---|---|---|---|
| 1 | *Вступ. Постановка мети та задач проектування* | *29.04.24* | виконано |
| 2 | *Аналіз предметної області* | *01.05.24* | виконано |
| 3 | *Вибір технологій та засобів розробки* | *03.05.24* | виконано |
| 4 | *Проєктування дизайну інтерфейсу* | *05.05.24* | виконано |
| 5 | *Розробка алгоритму роботи додатка* | *07.05.24* | виконано |
| 6 | *Розробка графічного дизайну* | *09.05.24* | виконано |
| 7 | *Реалізація мобільного-застосунку* | *15.05.24* | виконано |
| 8 | *Створення алгоритму* | *17.05.24* | виконано |
| 9 | *Економічний розрахунок* | *19.05.24* | виконано |
| 10 | *Опис охорони праці та техніки безпеки* | *21.05.24* | виконано |
| 11 | *Аналіз результатів проєктування.* | *23.05.24* | виконано |
| 12 | *Оформлення пояснювальної записки* | *30.05.24* | виконано |
| 13 | *Оформлення графічної (презентаційної) частини* | *05.06.24* | виконано |
| 14 | *Підготовка доповіді для захисту* | *07.06.24* | виконано |
| 15 | *Малий захист дипломного проєкту* | *10.06.24* | виконано |
| | | | |
| | | | |

Дипломник (підпис)

Керівник (підпис)

| Формат | Зона | Поз. | Позначення | Назва | Кіл. | Примітка |
|---|---|---|---|---|---|---|
| | | | | Документація | | |
| | | | РП 07. 22 000. 00 ДП | Дипломний проект | | |
| А4 | | | РП 07. 22 000. 00 ДП ПЗ | Пояснювальна записка | 1 | |

| Зм. | Арк. | № докум. | Підпис | Дата | РП 07. 22 000. 00 ДП | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Туркол О.Д | | 14.06 | Розробка та дизайн мобільного програмного забезпечення для творчих виступів | Літ. | Аркуш 4 / Аркушів 84 |
| Перевірив | | Закроєв Ю.М. | | 14.06 | | Н Д П | |
| Н. контр. | | Петрашова В.І. | | 14.06 | | ВСП "ОТФК ОНТУ" гр.4РП-07 | |
| Затверд. | | Кривченко Ю.В. | | 14.06 | | | |

# ЗМІСТ

# ВСТУП

Flutter— один з найпопулярніших і найвідоміших інструментів для розробки мобільних додатків. Завдяки своїй багатофункціональності, зручності використання та широким можливостям, вони стають першим вибором для багатьох розробників. Додатки, створені за допомогою Flutter, отримують популярність завдяки їхній здатності працювати на різних платформах з однією кодовою базою. Використання Android Studio як інтегрованого середовища розробки (IDE) забезпечує потужний інструментарій та спрощує процес створення, тестування та налагодження додатків.

Мета цього проєкту — створення зручного додатку для фокусів на базі Flutter. Такий застосунок допоможе розвивати мислення та розвинути навички магії. Розробка додатку для фокусів може бути джерелом задоволення для розробників, адже вона надає творчий вихід та почуття досягнення.

Використання Flutter дає можливість освоїти сучасні мобільні технології, такі як Dart і вбудовані віджети Flutter. Це дозволяє створювати додатки з чудовою продуктивністю та привабливим інтерфейсом. Вивчення методів оптимізації коду та підвищення продуктивності додатку на різних пристроях забезпечує ефективність та відмінний користувацький досвід.

Результатом розробки є зручний застосунок для фокусів на базі Flutter, який збіднює ефект нав'язування та вільного вибору глядача. Застосунок дозволяє глядачеві назвати будь-яке число, під яким миттєво з'являється заздалегідь визначене значення, що відповідає передбаченню фокусника. Глядач, вибравши число, бачить результат, який фокусник заздалегідь знає. Особливістю додатку є те, що він імітує робочий екран телефону, і людина навіть не здогадується, що запущено якийсь застосунок. Це робить трюк ще більш вражаючим та загадковим.

# 1 Основний розділ

## 1.1Аналіз предметної галузі

### 1.1.1 Огляд існуючих рішень

Додатки для фокусів є одними з найпопулярніших мобільних додатків, відомих своєю здатністю створювати захоплюючі ілюзії. Вони здебільшого зосереджені на тому, щоб допомогти фокусникам вражати глядачів.

Так, головною відмінністю додатків для фокусів від звичайних є їх здатність дивувати. Глядачі насолоджуються незвичайними фокусами, які створюють атмосферу таємничості і викликають захват. Такі додатки стимулюють творчість і розвиток репертуару виступаючих осіб, допомагаючи їм розширювати свої магічні навички та надавати глядачам неперевершені враження.

Розглянемо наступні аналоги до фокусних застосунків:

Застосунок “iForce”

На рисунку 1.1 Зображено Застосунок “iForce”


Рисунок 1.1 Застосунок для фокусів “iForce”

Застосунок iForce імітує рисувальну програму, де глядач може малювати будь-які малюнки. Після того як глядач створює малюнок, він може видалити його за допомогою спеціальної кнопки. Потім телефон передається фокуснику.

Непомітно для глядача, фокусник натискає на кнопку гучності, після чого малюнок, створений глядачем, знову відновлюється. Фокусник дивуватися на малюнок та називає його імітуючи читання думок чи бачення майбутнього

Розглянемо переваги додатку “iForce”

1. Робота офлайн
2. Можливість дізнатися, що загадав глядач
3. Простота використання

Розглянемо недоліки додатку “iForce”

1. Застаріла графіка та підозріле вигляд додатка
2. Відсутність варіативності та непомітності

Аналог 2 “Trick Pix”

Trick Pix це застосунок для камери, який поєднує в собі магію та жарти, дозволяючи вам розіграти друзів або здивувати повних незнайомців, коли вони фотографують вас. Уявіть, що ви просите когось зробити вам фотографію, але коли вони роблять знімок, ви не з'являєтесь на фото... або хтось інший! Це обмежується лише вашою уявою. Цей розумний застосунок дозволяє вам легко зробити неможливе всього лише одним натисканням кнопки!

На рисунку 1.2 Зображено застосунок “ Trick Pix ”


Рисунок 1.2 Застосунок “Trix Pix”

Розглянемо переваги додатку “Trix Pix”

1. Простота використання: Легко навчитися та використовувати, не потрібно спеціальних навичок або знань.

2. Різноманітність варіантів: Можливість вибору з різних магічних ефектів та жартів, що розширює можливості для творчості.

Розглянемо недоліки додатку “Trix Pix”

1. Застаріла графіка та підозріле вигляд додатка (2018 рік)
2. Ціна
3. На даний момент враховуючи впровадження нейромереж та повсюдне використання фотошопу даний ефект вже не буде виробляти вау ефект.

Аналог 3 “Magic Tricks By Mikael Montier”

Magic Tricks By Mikael Montier  це мобільне застосунок, яке знайомить користувачів з вражаючим рядом цифрових магічних ефектів, які вважаються одними з найповніших колекцій, доступних для смартфонів. Пропонуючи в цілому 24 унікальних ефекти з картами. Він має сумісність із понад 50 типами карток, які часто використовують, розширюючи свою привабливість та корисність. Застосунок простий у використанні, надаючи 12 цифрових магічних трюків безкоштовно, а також можливість розблокування додаткових функцій.

На рисунку 1.3 Зображено застосунок “ Magic Tricks By Mikael Montier ”

Рисунок 1.3 “ Magic Tricks By Mikael Montier ”

Розглянемо переваги додатку “ Magic Tricks By Mikael Montier”

1. Велика кількість різноманітних готових фокусів
2. Безкоштовна базова версія
3. Простота та інтуїтивний дизайн

Розглянемо недоліки додатку “ Magic Tricks By Mikael Montier ”

1. Відсутність незрозумілості (глядач відразу розуміє що це застосунок)

2. Доступ до всіх фокусів лише у платній версії
3. Відсутність варіативності (фокуси лише з картами)

Аналог 4 “The Architect of predictions”

“The Architect of predictions” - Програма є більш розвиненою Photo Prediction. Суть полягає в тому, що глядач загадує будь-яку кількість або карту, а фокусник дістає телефон, відкриває галерею, і там знаходиться фото з тим, що загадала людина.

На рисунку 1.4 Зображено застосунок “ The Architect of predictions”

Рисунок 1.4 “ The Architect of predictions”

Розглянемо переваги додатку “ The Architect of predictions”

1. Широкий вибір передбачень: Застосунок пропонує величезний вибір передбачень, включаючи числа, картини, події та багато іншого.
2. Персоналізовані трюки: Ви можете налаштувати передбачення з урахуванням ваших власних уподобань та ситуації.
3. Простий у використанні інтерфейс: Інтуїтивно зрозумілий і легкий у використанні інтерфейс робить процес створення передбачень простим і приємним.

Розглянемо недоліки додатку “ The Architect of predictions”

Тільки платна версія з ціною понад 100 доларів

Розглянемо порівняльну таблицю аналогів.

Таблиця 1.1 Порівняльна таблиця аналогів

| | Аналог 1 | | Аналог 2 | Аналог 3 | Аналог 4 |
|---|---|---|---|---|---|
| Простота використання | Так | | Так | Так | Так |
| Варіативність | Так | | Так | Ні | Так |
| Велика кількість фокусів | Ні | | Ні | Так | Так |
| Безкоштовна версія | Так | | Ні | Частково | Ні |
| Наявність української мови | Ні | | Ні | Ні | Ні |
| Непомітність | Ні | | Так | Ні | Так |

Основний висновок між додатками “iForce”, “Trix Pix”, “ Magic Tricks By Mikael Montier ” , “ The Architect of predictions” полягає в тому, що кожен з них пропонує унікальний підхід до жанру мобільної магії, надаючи фокусникам різноманітні можливості та інноваційні механіки та збільшує їх репертуар

-“iForce” може допомогти доповнити класичні фокуси з читання думок і включити у виступи малюнки.

-"Trix Pix", зосереджено на піднятті настрою глядачів, а також перевіряє їх на уважність і розширює репертуар артиста роботою з камерою та фотографіями.

-" Magic Tricks By Mikael Montier " допомагають фокусникам-початківцям вивчити основи фокусів з картами, так само розширюють звичайні карткові фокуси додаючи до них ефект передбачень і проявів карт

-“ The Architect of predictions” Дозволяє розширити свій виступ будь-якою кількістю фокусів, які обмежені лише уявою. Є найпотужнішим і непомітним інструментом у руках артиста

Отже, кожен з цих додатків має свої унікальні особливості та привабливості, які роблять їх цікавими для широкого кола артистів. Однак

можна сказати, що не всі аналоги є доступними, тільки один з них має повний безкоштовний функціонал, а так само жодний з них не має підтримки української мови. Тому розробка додатку, що розглядається в цій роботі є доцільним.

### 1.1.2 Технології та засоби розробки

Розробка моб додатків включає використання різноманітних технологій та інструментів, які допомагають створювати логіку, графіку та загалом інтерфейс моб додатка

На сьогоднішній день існує багато рішень для створення мобільного програмного забезпечення (Mobile App Development), які можна використовувати для розробки додатків на різних платформах (iOS, Android, та ін.).

Нижче наведено огляд деяких популярних інструментів та підходів:

Flutter

Це кросплатформений фреймворк для розробки мобільних, веб- та десктопних додатків, створений компанією Google. Основною мовою програмування для Flutter є Dart. Основна ідея полягає в тому, щоб розробляти один код та використовувати його для створення додатків для різних платформ.

Flutter пропонує гарячу перезавантаження, що дозволяє розробникам швидко вносити зміни в код і миттєво бачити їх вплив на застосунок без перезапуску. Це значно полегшує ітераційний процес розробки.

Фреймворк вражає своїм багатим набором готових віджетів, які розробники можуть використовувати для швидкої побудови привабливого та функціонального інтерфейсу. Flutter дозволяє створювати красиві та ефективні додатки завдяки власному двигуну малої ваги Skia, який гарантує високу продуктивність та гладкість анімацій.

Однією з ключових переваг є можливість кросплатформеної розробки, що дозволяє використовувати один і той самий код для створення додатків для iOS та Android. Це значно зекономлює час та зусилля розробників.

Щодо інструментів для розробки, Flutter постачається з власними інструментами, такими як Flutter DevTools, які полегшують аналіз та відлагодження додатків.

Проте, треба враховувати, що додатки, розроблені на Flutter, можуть мати більший розмір порівняно з тими, що розроблені нативно. Також, не всі функції нативних платформ можуть бути доступні безпосередньо через Flutter, що може вплинути на інтеграцію з певними пристроями або функціями. Деякі оптимізації та адаптації також можуть виявитися складними для досягнення порівняно з іншими підходами.

У підсумку, Flutter - це потужний фреймворк, який надає розробникам гнучкість та продуктивність при створенні кросплатформених додатків. Він вирізняється своєю ефективністю та зручністю розробки, роблячи його привабливим вибором для тих, хто шукає ефективний інструмент для створення мобільних додатків на різних платформах.

React Native:

React Native - це кросплатформений фреймворк для мобільної розробки, створений компанією Facebook. Він використовує мову програмування JavaScript та побудований на основі популярної бібліотеки React для веб-розробки. Основна ідея React Native полягає в тому, щоб дозволити розробникам використовувати той самий код для створення додатків для iOS та Android.

Однією з головних переваг React Native є широка спільнота розробників та велика кількість готових компонентів, які значно полегшують розробку. Він дозволяє використовувати нативні компоненти для побудови інтерфейсу, що забезпечує більш нативний вигляд та відчуття додатків.

Гнучкість та можливість використання стандартних веб-технологій, таких як JavaScript та React, роблять React Native привабливим для розробників з досвідом у веб-розробці. Крім того, фреймворк дозволяє гарячу перезавантаження, що полегшує швидку перевірку та впровадження змін в код.

React Native також має можливість використовувати нативні модулі для взаємодії з різними функціями пристроїв, що дає змогу використовувати унікальні можливості кожної платформи

Проте, є обмеження в можливостях React Native для доступу до низькорівневого функціоналу пристроїв, іноді вимагає використання модулів для вирішення специфічних завдань.

У підсумку, React Native - це потужний фреймворк для кросплатформеної мобільної розробки з активною спільнотою та великим розмаїттям готових рішень. Він вигідний для розробників, які шукають ефективний спосіб створення мобільних додатків з використанням знайомих веб-технологій.

Android Studio (Java/Kotlin):

Android Studio - це офіційна інтегрована середовища розробки (IDE) для розробки мобільних додатків для платформи Android. Це інструмент, розроблений компанією Google, і він став стандартним вибором для багатьох Android-розробників.

Основними мовами програмування для розробки Android-додатків є Java та Kotlin. Java була довгий час стандартною мовою для розробки Android-додатків, але з часом Kotlin отримав широкий розпізнавання та підтримку від Google, ставши офіційною мовою для розробки Android-додатків.

Android Studio надає розробникам доступ до повного набору інструментів для створення високоякісних мобільних додатків. Він має потужний редактор коду, інтегровану систему збірки та дебаггінгу, а також багатий комплект інструментів для профілювання та відлагодження додатків.

Важливо відзначити, що Android Studio надає розробникам повний доступ до всіх можливостей Android API та нативних функцій платформи. Це дозволяє створювати додатки, які повністю використовують потужності пристроїв і операційної системи Android.

Однією з переваг використання Android Studio є активна спільнота розробників, а також велика кількість документації та ресурсів, доступних для навчання та вдосконалення навичок розробки.

Недоліками можуть бути складність інтеграції з іншими платформами, такими як iOS, що може вимагати використання інших інструментів для кросплатформеної розробки.

Узагальнюючи, Android Studio з Java або Kotlin залишається основним інструментом для розробки Android-додатків, забезпечуючи розробникам високий рівень функціональності, доступ до всіх можливостей Android та велику спільноту для підтримки.

Xcode (Swift/Objective-C):

Xcode - це офіційна інтегрована середовища розробки (IDE) для платформи iOS, macOS, watchOS та tvOS. Розроблений компанією Apple, Xcode є стандартним інструментом для розробки додатків для всіх пристроїв, що працюють на операційних системах Apple.

Основні мови програмування для розробки iOS-додатків у Xcode - це Swift та Objective-C. Swift є новішою мовою, розробленою самою Apple, і швидко стає стандартним вибором для розробників завдяки його читабельності та безпекові. Однак Objective-C, яка довше використовується, є також підтримуваною мовою в Xcode.

Xcode надає повний доступ до всіх функцій та API, які доступні на платформах Apple. Розробники можуть використовувати весь спектр нативних компонентів та можливостей для створення додатків, які ідеально інтегруються з операційною системою та пристроями Apple.

Однією з переваг використання Xcode є інтегрована система розробки, тестування та відлагодження, що полегшує весь процес створення додатків. Xcode також має власний інтерфейс для створення і редагування інтерфейсу користувача, анімацій та інших елементів додатків.

Іншою важливою рисою Xcode є можливість використання Interface Builder для візуального створення і редагування інтерфейсу додатків.

Недоліками може бути те, що розробка додатків в Xcode обмежена платформами Apple, тому для кросплатформених рішень доводиться шукати інші варіанти.

У підсумку, Xcode разом з Swift або Objective-C є основним інструментом для розробки додатків для платформ Apple. Він забезпечує розробникам великі можливості та інтегрований підхід для створення високоякісних додатків, спеціально адаптованих до пристроїв Apple.

Adobe PhoneGap:

Adobe PhoneGap, також відомий як Apache Cordova, є кросплатформеним фреймворком для розробки мобільних додатків з використанням веб-технологій, таких як HTML, CSS та JavaScript. Цей фреймворк дозволяє створювати додатки, які можуть працювати як на Android, так і на iOS, а також на інших мобільних платформах.

Однією з ключових переваг Adobe PhoneGap є простота використання. Розробники можуть використовувати відомі веб-технології і знайомі мови програмування для створення мобільних додатків, що робить його привабливим для тих, хто вже має досвід у веб-розробці.

Ще однією важливою особливістю є можливість використовувати готові бібліотеки та фреймворки для розширення функціоналу додатків. PhoneGap підтримує ряд плагінів, які надають доступ до різноманітних функцій пристроїв, таких як камера, геолокація, контакти та інше.

Ще однією перевагою є можливість використовувати хмарні сервіси Adobe для полегшення розгортання та управління додатками.

Проте, Adobe PhoneGap може мати деякі обмеження в продуктивності порівняно з іншими кросплатформеними фреймворками. Взаємодія з низькорівневими функціями може вимагати використання плагінів, що може бути складним у деяких випадках.

Загалом, Adobe PhoneGap є добрим вибором для розробників, які шукають простий та швидкий спосіб створення кросплатформених мобільних додатків з використанням веб-технологій. Однак для додатків, які вимагають великої продуктивності або взаємодії з низькорівневими функціями пристроїв, може бути вигідніше розглядати інші альтернативи.

Unity**:**

Unity - це інтегрована середовища розробки (IDE) та двигун для створення ігор та інтерактивних 3D та 2D додатків. Цей фреймворк є дуже популярним і використовується не тільки для розробки ігор, але і для створення різноманітних віртуальних та доповнених реальностей, симуляторів, тренажерів та інших інтерактивних віртуальних середовищ.

Однією з ключових переваг Unity є його кросплатформеність. Розробники можуть створювати додатки, які працюють на різних платформах, таких як iOS, Android, Windows, macOS, Linux та інші. Це робить Unity відмінним вибором для тих, хто шукає універсальний фреймворк для розробки на різних пристроях.

Unity використовує мову програмування C# для розробки, що дозволяє розробникам використовувати потужний та ефективний мовний інструментарій. Багатий екосистем Unity включає в себе широкий набір ресурсів, бібліотек, готових моделей та інструментів для полегшення розробки.

Unity також володіє потужним інструментарієм для створення реалістичних 3D-графіки, анімацій та ефектів. Завдяки вбудованим можливостям фізичного моделювання, штучного інтелекту та мережевого взаємодії, Unity робить можливим створення високоякісних ігор та додатків.

Недоліками Unity може бути великий розмір додатків, які створюються на цьому двигуні, а також вивчення деякої кривої навчання для новачків.

Загалом, Unity - це потужний та універсальний фреймворк для розробки ігор та інтерактивних додатків, який знайшов широке застосування у галузі віртуальної реальності, навчання, симуляцій та розваг.

Розглянемо порівняльну таблицю рішень для створення мобільного програмного забезпечення.

Таблиця 1.2 Порівняльна таблиця рішень для створення мобільного програмного забезпечення

| Характеристики | Flutter | React Native | Xcode | Android Studio | Adobe PhoneGap: | Unity |
|---|---|---|---|---|---|---|
| Мови програмування | Dart | JavaScript | Swift/Objective-C | Java/Kotlin | JavaScript/CSS/HTML | C# |
| Платформи розробки | Windows, macOS Android, | Windows, macOS, Android, | iOS, macOS, watchOS та tvOS | Android | Android/IOS | Windows, macOS, Linux |
| Інтеграція з платформами: | Обмежена інтеграція, але можлива за допомогою плагінів. | Native: Інтеграція з нативними компонентами платформ Android та iOS. | Нативна підтримка для розробки додатків для платформ iOS та macOS. | Інтегрується прямо з Android SDK та сервісами Google. | Може використовувати бібліотеки Cordova для доступу до функціональності пристрою. | Широкі можливості інтеграції з різними платформами та сервісами. |
| Спільні ресурси та підтримка: | Активна підтримка Google та ростуча спільнота. | Велика спільнота та активна розробка Facebook. | Офіційний інструмент розробки для платформ Apple, з великою кількістю ресурсів та документації | Офіційна підтримка Google та велика спільнота розробників. | Має підтримку Adobe та користується популярністю серед веб-розробників. | Велике співтовариство та широка документація. |

**Висновок**:

Вибір програмні засоби та інструменти, шаблони проектування, мова програмування, стратегії та парадигми проектування для розробки мобільних додатків залежить від ряду факторів, таких як ваший досвід, складність проекту, вимоги до продуктивності та можливості кросплатформеності. Кожен інструмент має свої переваги та обмеження, і важливо ретельно вибрати той, який найкраще відповідає вашим потребам. Для своєї роботи я оббрав Flutter з IDE Android Studio

### 1.2 Проектування програми

#### 1.2.1 Технічне завдання на розробку

Зазначені вимоги до розробки програмного забезпечення визначають основні напрямки та параметри проекту, які слід враховувати для створення успішного додатку. Вони створюють каркас, на основі якого розробники можуть визначити конкретні етапи роботи та вибрати оптимальні технічні рішення.

Розробити мобільний застосунок для творчих виступів (а саме для фокусів) на фреймворку флаттер використовуючи мову програмування Dart

Розробити та реалізувати механіку імітації вільного вибору користувача та нав'язування предметів . А також можливість зберігати шаблони для майбутніх виступів

Розробити інтерфейс та органи управління

Розробити дизайн так що б глядач не міг здогадатися, що зараз працює застосунок і його вибір ні на що не впливає.

#### 1.2.2 Проєктування дизайну застосунку

Розуміння потреб та вимог користувачів є ключовим етапом у розробці дизайну. Проектування дизайну починається з глибокого вивчення потреб та

вимог користувачів, оскільки це визначає напрямок подальшої роботи. Дослідження користувачів включає аналіз цільової аудиторії, опитування, інтерв'ю та аналіз конкурентів. Воно допомагає зрозуміти, хто саме користуватиметься продуктом, їх потреби, звички та очікування. Аналіз конкурентів дозволяє виявити успішні рішення на ринку та визначити недоліки, які можна врахувати у власному продукті. Формулювання вимог до дизайну включає розробку функціональних вимог, таких як зрозуміле управління та інтерфейс, що відповідає потребам користувачів.

Для розробки дизайну програми для творчих виступів я проаналізував мінуси конкурентів і відгуки користувачів. І дійшов висновку, що в "мобільній магії" є один головний недолік. Те, що людина розуміє, що це робить програма, а не "магія". З цього можна дійти висновку, що непомітність є гланим критерієм для таких програм.

Для досягнення ефекту непомітності я вирішив, що моє програмне забезпечення імітуватиме головний екран мобільного телефону. Для референсів я взяв скріншоти екранів різних моделей телефонів і дійшов висновку, що екран iPhone є стандартним і викликає найменше підозр через закритість системи iOS

На рисунку 1.5 зазначего приклад дизайну головного экрану iOS


Рисунок 1.5 Приклад дизайну головного экрану iOS

## 1.3 Реалізація застосунку

### 1.3.1 Створення алгоритму

Блок-схеми алгоритмів є важливим інструментом для розуміння та візуалізації роботи програм чи алгоритмів. Вони дозволяють ілюструвати кроки, які потрібно виконати для досягнення певної мети, та вказують на зв'язки між цими кроками. Зараз розберемо блок схему розробки мого застосування

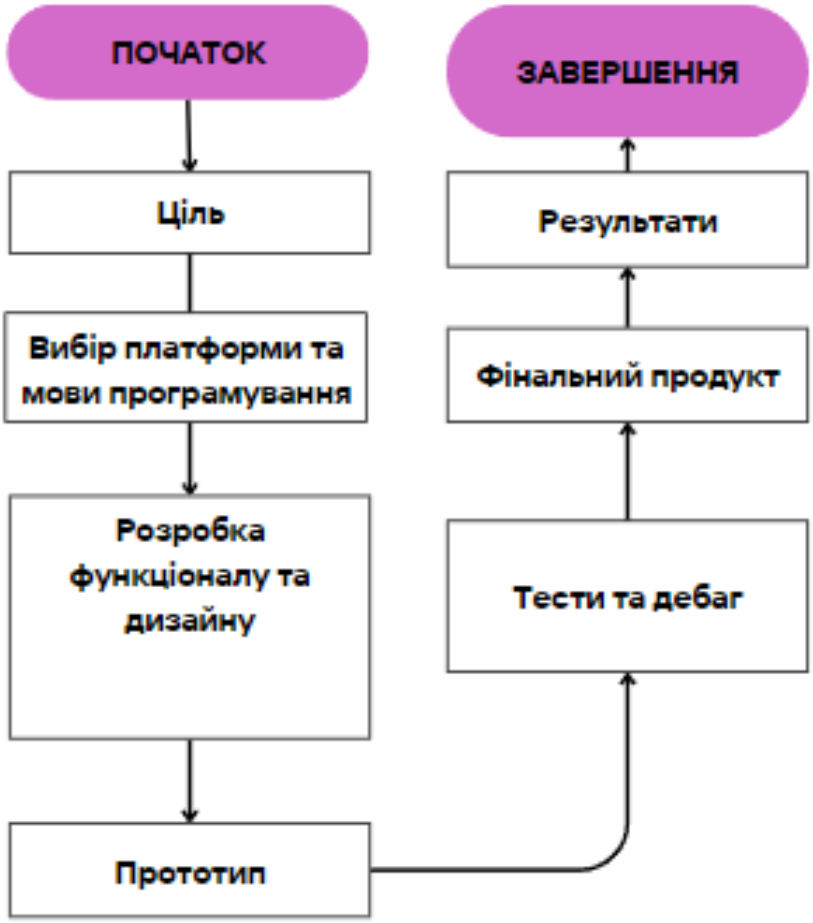



Рисунок 1.6 блок-схема розробки застосунку

Також розглянемо блок-схему самого додатка

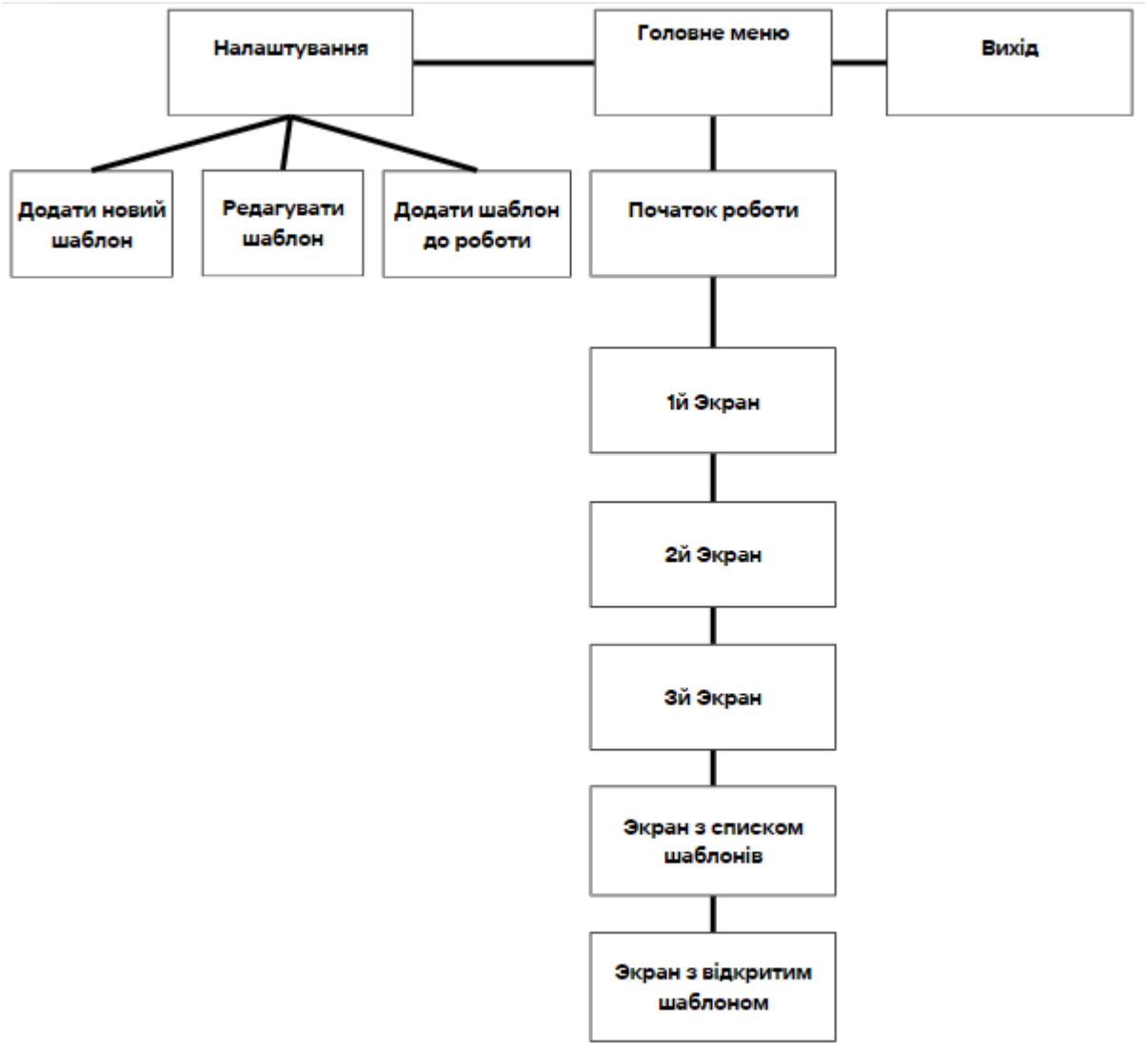



Рисунок 1.7 Блок-схема програмного забезпечення для творчих виступів

### 1.3.2 Створення головного меню

Початок створення проекту на флаттер починається з прописування тема а також як застосунок взаємодіє з нижньою та верхньою панеллю телефону

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'app_settings.dart'; //страница налаштувань
import 'notes_screen.dart'; // панель нотаток

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    SystemChrome.setEnabledSystemUIMode(SystemUiMode.manual,          overlays:
[SystemUiOverlay.top]);
    SystemChrome.setSystemUIOverlayStyle(SystemUiOverlayStyle(
      statusBarColor: Colors.transparent, // Укажите color
      statusBarIconBrightness: Brightness.light, // Вкажіть бажану яскравість
піктограм (чорні або білі).
    ));
    return MaterialApp(
      home: WelcomeScreen(),
    );
  }
} }
```

Рисунок 1.8 Фрагмент коду MyApp

Створення головного екрану: У класі MyApp я встановлюю налаштування для системної панелі, таких як колір фону та яскравість піктограм. Потім створюється екземпляр класу MaterialApp, який відображає головний екран додатка, яким є WelcomeScreen.

Головний екран додатка: Клас WelcomeScreen відповідає за відображення головного екрану. Він містить декілька кнопок, які ведуть на інші екрани додатка, такі як MyPageView та SettingsScreen. Також є кнопка для виходу з додатка. Екрани MyPageView та SettingsScreen: Ці екрани відповідають за показ різного вмісту або налаштувань. Вони можуть бути реалізовані відповідно до потреб мого дипломного дослідження, наприклад, MyPageView може показувати вміст презентації, а SettingsScreen - налаштування додатка.

Функціональність: В коді також реалізована базова функціональність, така як навігація між екранами, відображення налаштувань і обробка подій кнопок.

Далі я прописав сам вид головного меню

```
class WelcomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ElevatedButton(
              onPressed: () {
                Navigator.of(context).pushReplacement(
                  MaterialPageRoute(
                    builder: (context) => MyPageView(),
                  ),
                );
              },
              child: Text("Почати показ"),
            ),
            SizedBox(height: 16),
            ElevatedButton(
              onPressed: () {
                Navigator.of(context).push(
                  MaterialPageRoute(
                    builder: (context) => SettingsScreen(),
                  ),
                );
              },
              child: Text("Налаштування"),
            ),
            SizedBox(height: 16),
            ElevatedButton(
              onPressed: () {
                SystemNavigator.pop();
              },
              child: Text("Вихід"),
            ),
          ],
        ),
      ),
    );
  }
}
```

Рисунок 1.9 Фрагмент коду WelcomeScreen

Цей фрагмент коду відображає головний екран додатка. Він містить кнопки для переходу на інші екрани або виходу з додатка. Використовуючи віджети Scaffold, Center і Column, ми створюємо вертикальний макет з кнопками. Кнопки створені за допомогою віджету ElevatedButton, який має піднятий ефект при натисканні. Щоб поліпшити розташування кнопок, ми

використовуємо віджет SizedBox, який створює вертикальний проміжок між ними

Нижче приведена стартоваий екран головного меню

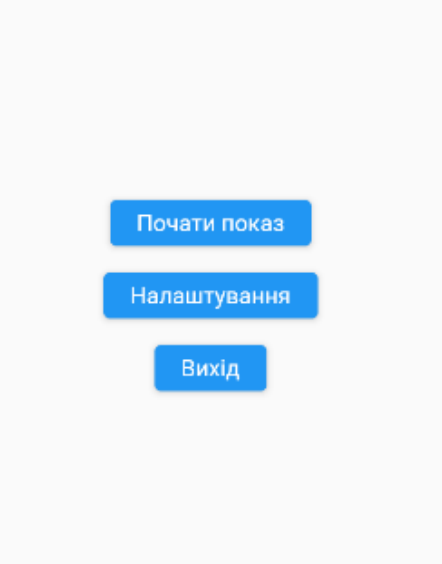



Рис 1.10 Стартовий екран головного меню

Прописуючи класс WelcomeScreen У методі build я використовую Scaffold, щоб організувати структуру екрана. У центрі екрана я розміщую Column, щоб відобразити кілька кнопок одну за одною.

У кожній кнопці ElevatedButton є свій текст і функція onPressed, яка викликається при натисканні на неї.

- Перша кнопка дозволяє користувачам почати показуючи їм MyPageView, і я використовую Navigator.of(context).pushReplacement, щоб замінити поточний екран на новий.
- Друга кнопка відкриває екран налаштувань, дозволяючи користувачам налаштувати параметри додатку.
- Третя кнопка просто закриває застосунок за допомогою SystemNavigator.pop().

### 1.3.3 Створення панелі налаштувань

Далі я створив екран налаштувань і переніс його до окремого файлу app_setings.dart. У ньому я прописав клас налаштувань та кнопки всередині нього

```
Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Налаштування"),
      ),
      body: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            children: [
              for (int i = 0; i < appSettings.templates.length; i++)
                Card(
                  color:  appSettings.templates[i].selected  ?  Colors.green  :
null,
                  child: ListTile(
                    title: Text(appSettings.templates[i].item),
                    subtitle: Text(appSettings.templates[i].title),
                    trailing: Row(
                      mainAxisSize: MainAxisSize.min,
                      children: [
                        IconButton(
                          icon: Icon(Icons.edit),
                          onPressed: () {
                            editTemplate(i);
                          },
                        ),
                        IconButton(
                          icon: Icon(Icons.delete),
                          onPressed: () {
                            deleteTemplate(i);
                          },
                        ),
                    onTap: () {
                      setState(() {
                        appSettings.templates[i].selected                    =
!appSettings.templates[i].selected;
                      });
                      ),
              ElevatedButton(
                onPressed: () {
                  addTemplate();
                },
                child: Text("Додати шаблон"),
              ),
              SizedBox(height: 16),
              ElevatedButton(
                onPressed: () {
                  appSettings.saveSettings(context,"main");
                  setState(() {
                    appSettings.selectedTemplates.forEach((template) {
                      print('Selected       Template:       ${template.item},
${template.title}, ${tem-plate.list}');
                    });

                  });
                },
                child: Text("Зберегти"),
```

Рис 1.11 Фрагмент коду app_setings.dart.

Нижче приклад екрану налаштувань без доданих шаблонів

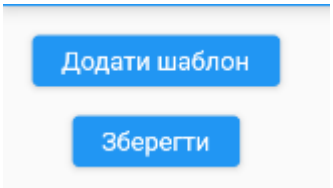



Рисунок 1.12 Экран налаштувань без доданих шаблонів

У методі build() я створюю основну структуру екрану за допомогою віджету Scaffold. Він містить заголовок віджету за допомогою AppBar, де вказано "Налаштування". Основний контент екрану розміщений у віджеті SingleChildScrollView, що дозволяє прокручувати його у разі необхідності.

У середині екрану розміщені різні елементи у вигляді карток (Card), кожна з яких представляє собою окремий параметр налаштувань. Крім того, користувач може редагувати або видаляти ці налаштування за допомогою відповідних кнопок.

На екрані також присутні кнопки "Додати шаблон" та "Зберегти", які викликають відповідні функції для додавання нових налаштувань та збереження змін.

### 1.3.4 Экран “Додати шаблон”

При натисканні на кнопку "Додати шаблон" у нас відкривається новий екран з полями для введення та збереження даних

```
class _TemplateAddScreenState extends State<TemplateAddScreen> {
  final TextEditingController itemController = TextEditingController();
  final TextEditingController titleController = TextEditingController();
  final TextEditingController listController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Додати шаблон"),
      ),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            TextField(
              controller: titleController,
              decoration: InputDecoration(labelText: "Назва нотатки"),
```

```
            ),//Назва
            TextField(
              controller: itemController,
              decoration: InputDecoration(labelText: "Предмет"),
            ),//Предмет
            TextField(
              controller: listController,
              decoration: InputDecoration(
                labelText:  "Список  (введіть  значення,  розділяючи  символом
перенесення рядка)",
              ),
              maxLines: null,
            ),//Список
            SizedBox(height: 16),
            ElevatedButton(
              onPressed: () {
                saveTemplate();
              },
              child: Text("Зберегти"),
            ),//
          ],
        ),
      ),
    );
  }
```

Рисунок 1.13 Фрагмент коду TemplateAddScreenState

Клас _TemplateAddScreenState розширюється від State<TemplateAddScreen>, що означає, що цей стан використовується для управління станом віджета TemplateAddScreen.

У цьому стані створюються три об'єкти TextEditingController, які використовуються для керування вмістом текстових полів. Один контролер для введення назви (titleController), один для предмету (itemController), і один для списку (listController).

У методі build створюється віджет Scaffold, який містить AppBar з назвою "Додати шаблон" і body, який містить SingleChildScrollView, щоб забезпечити прокрутку контенту, коли екран маленький. У SingleChildScrollView є Column, який містить три TextField для введення назви, предмету і списку. Кожен TextField використовує відповідний контролер, який був створений раніше.

Далі йде пустий відступ (SizedBox(height: 16)), і після цього розміщений ElevatedButton з текстом "Зберегти". При натисканні на цю кнопку виконується метод saveTemplate().

Нижче приклад, як виглядає цей екран

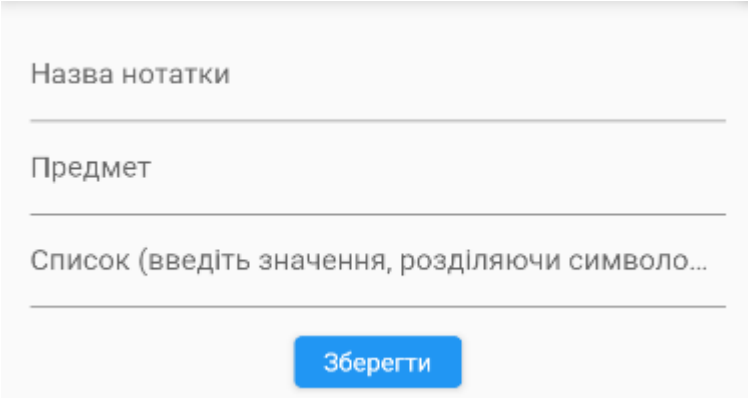



Рисунок 1.14 “Экран додавання шаблонів”

```
void saveTemplate() {
    final newTemplate = SettingsTemplate(
      itemController.text,
      titleController.text,
      listController.text.split('\n'),
    );
    widget.appSettings.templates.add(newTemplate);
    widget.appSettings.saveSettings(context,"add");
    Navigator.of(context).pop(true);
  }
}
```

Рисунок 1.15 Метод saveTemplate

Цей метод `saveTemplate()` виконує декілька дій при натисканні на кнопку "Зберегти":

1. Створюється новий об'єкт `SettingsTemplate`, який ініціалізується з текстом, який був введений у текстові поля (`itemController.text`, `titleController.text`) і розділеним списком значень, які були введені у текстове поле списку (`listController.text.split('\n')`).

2. Новий шаблон додається до списку шаблонів, який знаходиться у `appSettings`.

3. Зберігаються налаштування за допомогою методу `saveSettings()` з об'єкту `appSettings`, передаючи параметр `'"add"'`, щоб вказати, що відбувається додавання нового шаблону.

4. Викликається метод `pop(true)` для закриття поточного екрану та повернення на попередній екран. Відповідність `true` може бути використана вище для індикації, що операція збереження була успішно виконана.

Після збереження шаблону меню налаштувань тепер буде виглядати так

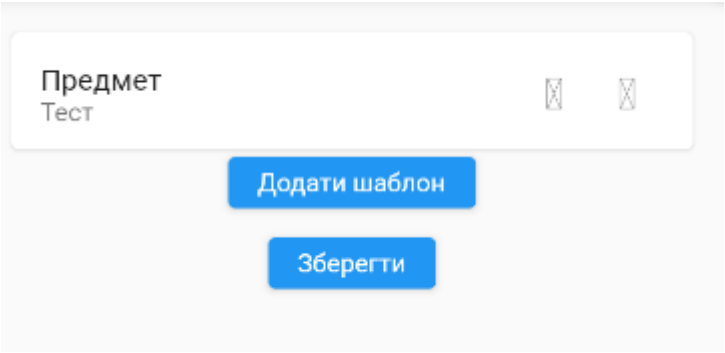



Рисунок 1.16 “Экран налаштувань з доданим шаблоном”

Для запам'ятовування шаблону та запуску його в "роботу" можна просто натиснути на нього. Він змінить колір на зелений та нажати на зберегти.

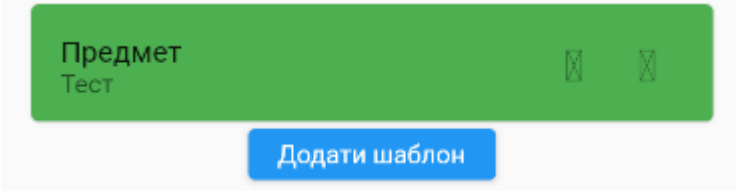



Рисунок 1.17 “Вибраний шаблон ”

Після натискання триває кілька перевірок. 1я. Якщо не виділять жодного шаблону, то вийде таке попередження

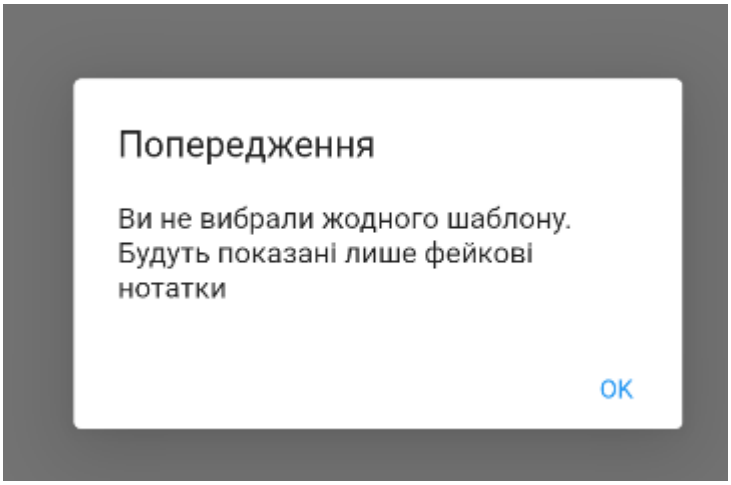



Рисунок 1.18 Попередження при збереженні

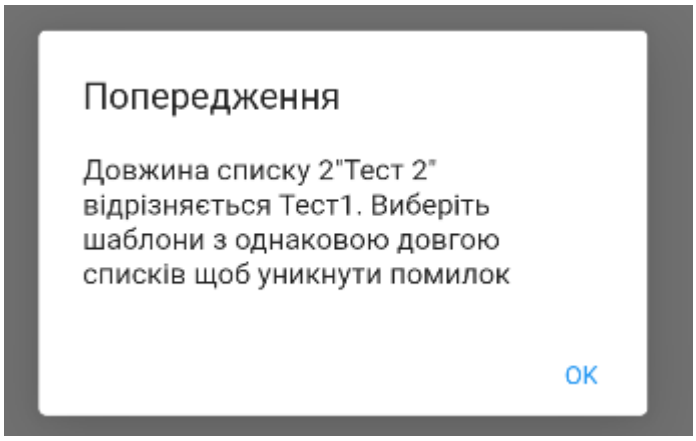



Рисунок 1.19 Попередження при різних довжинах списків в шаблонах

```
if (FromWhat == "main"){
      if (selectedTemplates.isNotEmpty) {
        final int firstListLength = selectedTemplates[0].list.length;
        final String firstListTitle = selectedTemplates[0].title;

        for (final template in selectedTemplates) {
          if (template.list.length != firstListLength) {
            // Виводимо діалогове вікно із попередженням, використовуючи
переданий context
            showDialog(
              context: context,
              builder: (context) {
                return AlertDialog(
                  title: Text('Попередження'),
                  content:                Text('Довжина                списку
${template.item}"${template.title}" відрізняється $firstListTitle. Виберіть
шаблони з однаковою довгою списків щоб уникнути помилок'),
                  actions: [
                    TextButton(
                      onPressed: () {
                        Navigator.of(context).pop();
                      },
                      child: Text('OK'),
                    ),
                  ],
                );
              },
            );
            break;
          }
        }
      }
      else{
        showDialog(
          context: context,
          builder: (context) {
            return AlertDialog(
              title: Text('Попередження'),
              content: Text('Ви не вибрали жодного шаблону. Будуть показані
лише фейкові нотатки'),
              actions: [
                TextButton(
                  onPressed: () {
                    Navigator.of(context).pop();
                  },
                  child: Text('OK'),
                ),
              ],
            );
          },
```

Рисунок 1.20 Фрагмент коду перевірки шаблону

Цей фрагмент коду перевіряє, чи є обрані шаблони, і якщо так, перевіряє, чи всі вони мають однакову довжину списків. Якщо довжини списків відрізняються, показується діалогове вікно з попередженням. Якщо обрані шаблони відсутні, також показується попередження

Якщо `FromWhat` дорівнює `"main"`, це виконується. У кожному випадку показується `AlertDialog` з відповідним текстом попередження. Кнопка "OK" дозволяє закрити діалогове вікно.

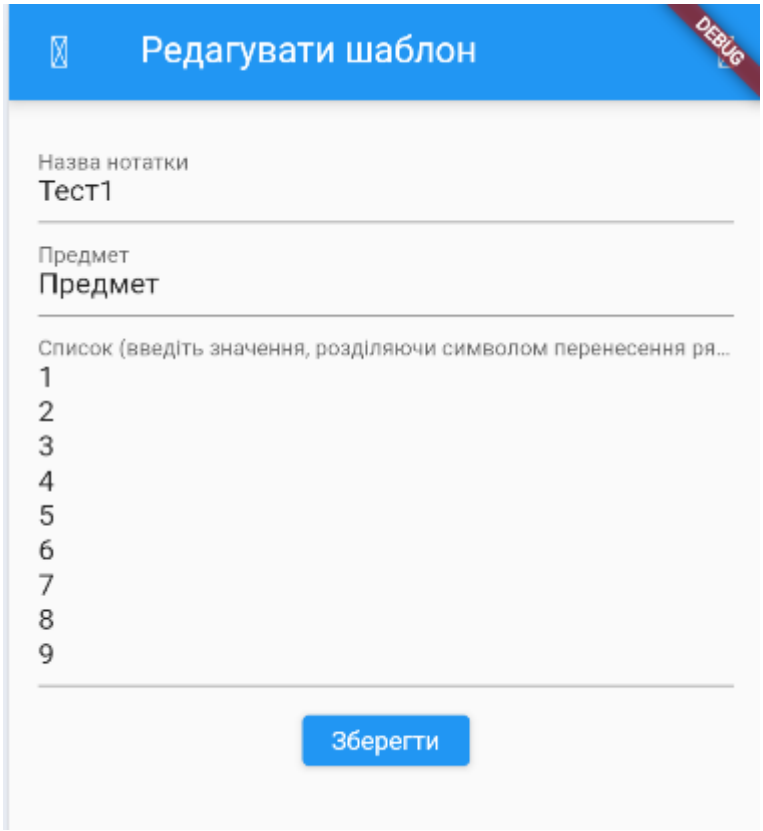



Рисунок 1.21 Экран редагування

```
class _TemplateEditScreenState extends State<TemplateEditScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Редагувати шаблон"),
        actions: [
          IconButton(
            icon: Icon(Icons.delete),
            onPressed: () {
              deleteTemplate();
            },
          ),
        ],
      ),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(16.0),
```

```
        child: Column(
          children: [
            TextField(
              controller: widget.titleController,
              decoration: InputDecoration(labelText: "Назва нотатки"),
            ),
            TextField(
              controller: widget.itemController,
              decoration: InputDecoration(labelText: "Предмет"),
            ),
            TextField(
              controller: widget.listController,
              decoration: InputDecoration(
                labelText: "Список (введіть значення, розділяючи символом перенесення рядка)",
              ),
              maxLines: null,
            ),
            SizedBox(height: 16),
            ElevatedButton(
              onPressed: () {
                saveTemplate();
              },
              child: Text("Зберегти"),
            ),
          ],
        ),
      ),
    );
  }

  void saveTemplate() {
    final updatedTemplate = SettingsTemplate(
      widget.itemController.text,
      widget.titleController.text,
      widget.listController.text.split('\n'),
      selected: widget.appSettings.templates[widget.templateIndex].selected,
    );
    widget.appSettings.templates[widget.templateIndex] = updatedTemplate;
    widget.appSettings.saveSettings(context,"edit");
    Navigator.of(context).pop(true);
  }
```

Рисунок 1.22 Фрагмент коду TemplateEditScreenState

Використовуються ті самі методи, що при додаванні але перевіряється "звідки пішов виклик". Так як він пішов з панелі редагування, то нового не створюється, а лише редагується старий шаблон.

При натисканні на кнопку видалити запитає підтвердження видалення шаблону.

```
void deleteTemplate() {
    showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          title: Text("Видалити шаблон"),
          content: Text("Ви впевнені, що хочете видалити цей шаблон?"),
          actions: [
            TextButton(
              onPressed: () {
                Navigator.of(context).pop();
              },
              child: Text("Відміна"),
            ),
            TextButton(
              onPressed: () {
                widget.appSettings.templates.removeAt(widget.templateIndex);
                widget.appSettings.saveSettings(context,"delete");
                Navigator.of(context).pop();
                Navigator.of(context).pop(true);
              },
              child: Text("Видалити"),
            ),
          ],
        );
      },
    );
  }
}
```

Рисунок 1.23 Фрагмент методу deleteTemplate

Метод showDialog відображає модальне діалогове вікно з заголовком "Видалити шаблон" та текстом "Ви впевнені, що хочете видалити цей шаблон?".

Кнопка "Відміна" дозволяє користувачу закрити діалогове вікно без виконання жодних дій. Викликається метод Navigator.of(context).pop() для закриття діалогу.

Копка "Видалити" виконує видалення вибраного шаблону:

- Видаляється шаблон з колекції appSettings.templates за індексом widget.templateIndex.
- Викликається метод saveSettings() для збереження змін в налаштуваннях.

- Діалогове вікно закривається двома викликами Navigator.of(context).pop(), де перший закриває діалог, а другий закриває екран редагування шаблону, повертаючи true для індикації успішного видалення.

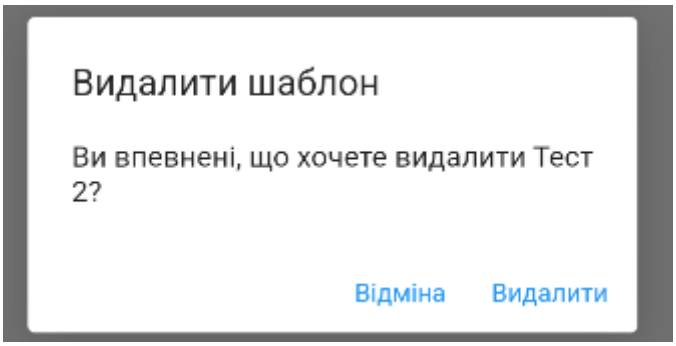



Рисунок 1.24 Видалення шаблону

### 1.3.5 Створення 1 екрану

Оскільки ми маємо імітувати головний екран телефону. Я скопіював дизайн iOS

Рисунок 1.25. 1й Экран

```
class Page1 extends StatelessWidget {
  final Function nextPage;
  final Function previousPage;
  final Function updateShowBottomPanel;

  Page1({required this.nextPage, required this.previousPage, required this.updateShowBottomPanel});



  List<List<String>> imagePaths1 = [
    ['assets/images1/weather.png', 'Погода'],
    ['assets/images1/find-my.png', 'Локатор'],
    ['assets/images1/my-shortcuts.png', 'Команди'],
    ['assets/images1/home.png', 'Дім'],
    ['assets/images1/contacts.png', 'Контакти'],
    ['assets/images1/files.png', 'Файли'],

  ];

  @override
  Widget build(BuildContext context) {
    double screenWidth = MediaQuery.of(context).size.width;
    double screenHeight = MediaQuery.of(context).size.height;
    double iconSize = (screenWidth / 13) * 2;
    double spacing = (screenWidth / 13) / 2;
    double labelSize = iconSize/4.5;
    double HeightSpace = screenHeight*0.045;
    double Razmer = iconSize+HeightSpace;



    return Scaffold(
      backgroundColor: Colors.transparent,
      body: Padding(
        padding: EdgeInsets.only(right: spacing, left: spacing,top: HeightSpace),
        child: GridView.builder(
          shrinkWrap: true,
          physics: NeverScrollableScrollPhysics(),
          gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
            crossAxisCount: 4,
          ),
          itemCount: imagePaths1.length,
          itemBuilder: (context, index) {

            return Center(
              child: Container(
                width: (screenHeight-iconSize)/4,
                height: Razmer,
                child: Column(
                  children: [
                    GestureDetector(
                      onVerticalDragStart: (details) {
                        Navigator.of(context).push(
                          MaterialPageRoute(
                            builder: (context) => WelcomeScreen(),
```

```
                        ),
                      );
                      // При свайпе вверх  WelcomeScreen
                    },
                    onHorizontalDragStart: (details) {
                      first = index;
                      // Додайте код для переходу на наступний екран
                      nextPage();
                    },
                    child: ClipRRect(
                      borderRadius: BorderRadius.circular(14),
                      child: Image.asset(
                        imagePaths1[index][0],
                        width: iconSize,
                        height: iconSize,
                        fit: BoxFit.cover,
                      ),
                    ),
                  ),
                  Padding(
                    padding:    EdgeInsets.only(left:    spacing,    right:
spacing),
                    child: FittedBox(
                      fit: BoxFit.none, // Це зменшить розмір тексту, щоб
він вписувався в доступний простір
                      child: Text(
                        imagePaths1[index][1],
                        style: TextStyle(
                            fontSize: 7,
                            color: Colors.white),
                      ),
                    ),
                  ),

                ],
              ),
            ),
          );
        },
      ),
    ),
  );
```

Рисунок 1.26. Фрагмент коду Page1

Цей код представляє віджет Page1, який відображає сітку з іконками та підписами. Дані про шляхи до зображень та підписи зберігаються в imagePaths1. У методі build відбувається побудова сітки за допомогою GridView.builder. Кожен елемент сітки містить обрізане зображення, а також підпис, що відображається під ним. Використовуються обробники подій onHorizontalDragStart та onVerticalDragStart для обробки горизонтального та вертикального перетягування. При горизонтальному перетягуванні викликають функцію nextPage(). Шляхи до зображень та підписи динамічно зчитуються зі

списку imagePaths1. Відстань між елементами та розміри обчислюються на основі розмірів екрану.

При свайпі вниз відбувається повернення до меню налаштувань.

Також якщо свайп почався з іконки, передається індекс іконки за наступною схемою.

```
switch (firstIndex) {
    case 13:
    case 14:
    case 15:
      OneIndex = firstIndex - 12;
      break;
    case 17:
    case 18:
    case 19:
      OneIndex = firstIndex - 13;
      break;
    case 21:
    case 22:
    case 23:
      OneIndex = firstIndex - 14;
      break;
    default:
      OneIndex  = -1;
  }
```

Рисунок 1.27. Фрагмент коду схеми обробки  індекса

Цей фрагмент коду використовує оператор `switch` для обробки значення `firstIndex`. Згідно з його значенням, змінній `OneIndex` присвоюється певне значення.

Ось пояснення кожного `case` в `switch`:

- У випадку, якщо `firstIndex` дорівнює 13, 14 або 15, від `firstIndex` віднімається 12, і результат присвоюється `OneIndex`.
- У випадку, якщо `firstIndex` дорівнює 17, 18 або 19, від `firstIndex` віднімається 13, і результат присвоюється `OneIndex`.
- У випадку, якщо `firstIndex` дорівнює 21, 22 або 23, від `firstIndex` віднімається 14, і результат присвоюється `OneIndex`.
- У всіх інших випадках значення `OneIndex` встановлюється як -1.

Цей код дозволяє ефективно визначати значення `OneIndex` в залежності від значення `firstIndex`. Для кожного екрану зображення розміщені в окремій

директорії assets. Потім кожна картинка виводиться методом gridView і займає своє становище. Так само, як на головному екрані телефону

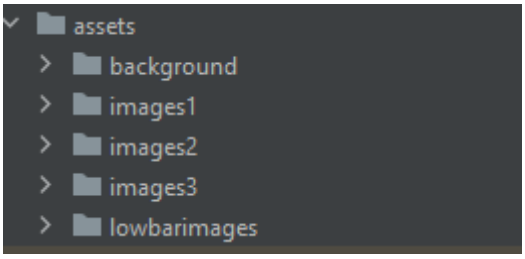



Рисунок 1.28 папка assets

```
Widget build(BuildContext context) {
    return Scaffold(
      body: GestureDetector(
          onHorizontalDragStart: (details) {
            _startX = details.localPosition.dx;
            _startY = details.localPosition.dy;
          },
          onHorizontalDragUpdate: (details) {
            double dx = details.localPosition.dx - _startX;

            if (dx > 0) {
              previousPage();
            } else if (dx < 0) {
              nextPage();
            }
          },
          child: Stack(
            children: <Widget>[
              Container(
                decoration: BoxDecoration(
                  image: DecorationImage(
                    image: AssetImage('assets/background/background.jpg'),
                    fit: BoxFit.cover,
                  ),
                ),
              ),
              if (showBottomPanel) // Перевірка, показувати чи ні нижню
панель
                Positioned(
                  left: 9,
                  right: 9,
                  bottom: 9,
                  child: Container(
                    height: 70.0,
                    decoration: BoxDecoration(
                      color: Colors.grey.withOpacity(0.4),
                      borderRadius: BorderRadius.circular(25.0),
                    ),
                    child: Row(
                      mainAxisAlignment: MainAxisAlignment.spaceAround,
                      children: List.generate(4, (index) {
                        return GestureDetector(
                          onTap: () {
```

```
                            print("Нажал кнопку с нижней панели $index");
                            // Обробник натискання елемент нижньої панелі
                          },
                          child: Column(
                            mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
                            children: [
                              Padding(
                                padding: EdgeInsets.all(8.0),
                                child: Image.asset(
                                  imagePaths[index],
                                  width: 50.0,
                                  height: 50.0,
                                ),
                              ),
                            ],
                          ),
                        );
                      }),
                    ),
                  ),
                ),
              PageView(
                controller: _pageController,
                children: <Widget>[
                  Navigator(
                    onGenerateRoute: (settings) {
                      return MaterialPageRoute(
                        builder: (context) => Page1(
                          nextPage: nextPage,
                          previousPage: previousPage,
                          updateShowBottomPanel: _updateShowBottomPanel,
                        ),
                      );
                    },
                  ),
                  Navigator(
                    onGenerateRoute: (settings) {
                      return MaterialPageRoute(
                        builder: (context) => Page2(
                          nextPage: nextPage,
                          previousPage: previousPage,
                          updateShowBottomPanel: _updateShowBottomPanel,
                        ),
                      );
                    },
                  ),
                  Navigator(
                    onGenerateRoute: (settings) {
                      return MaterialPageRoute(
                        builder: (context) => Page3(
                          nextPage: nextPage,
                          previousPage: previousPage,
                          updateShowBottomPanel: _updateShowBottomPanel,
```

Рисунок 1.29 Фрагмент коду розміщення нижньої панелі

Цей віджет використовує Scaffold для розміщення вмісту. Він містить GestureDetector, який слухає події перетягування для перемикання між сторінками. Цей GestureDetector містить Stack, який об'єднує контейнер з фоновим зображенням, нижню панель та PageView. У контейнері Stack зображення фону відображається за допомогою DecorationImage, щоб заповнити весь екран. Нижня панель відображається у Positioned, щоб вона розміщалася внизу сторінки. Вона містить рядок з чотирма елементами, які обробляються GestureDetector. Кожен елемент представляє собою зображення, яке відображається за допомогою Image.asset. PageView відображає три різні сторінки (Page1, Page2, Page3), кожна з яких має власний навігатор Navigator.

Рисунок 1.30. Нижня панель

Кожен Navigator може переходити на наступний або попередній екрани за допомогою функцій nextPage та previousPage.

```
void nextPage() {
    _pageController.nextPage(duration:   Duration(milliseconds:   600),   curve:
Curves.ease);
  }

  void previousPage() {
    _pageController.previousPage(duration:  Duration(milliseconds:  600),  curve:
Curves.ease);
  }
```

Рисунок 1.31. Функції nextPage та previousPage.

### 1.3.6 Створення 2 екрану

```
    class Page2 extends StatelessWidget {
      final Function nextPage;
      final Function previousPage;
      final Function updateShowBottomPanel;

      Page2({required  this.nextPage,  required  this.previousPage,  required
    this.updateShowBottomPanel});

      List<List<String>> imagePaths1 = [
        ['assets/images2/instagram.png', 'Instagram'],
      ];

      @override
```

```
    Widget build(BuildContext context) {
      double screenWidth = MediaQuery.of(context).size.width;
      double screenHeight = MediaQuery.of(context).size.height;
      double iconSize = (screenWidth / 13)*2;
      double spacing = (screenWidth / 13) / 2;
      double labelSize = iconSize/4.5;
      double HeightSpace = screenHeight*0.045;
      double Razmer = iconSize+HeightSpace;

      return Scaffold(
        backgroundColor: Colors.transparent,
        body: Padding(
          padding:  EdgeInsets.only(right:  spacing,  left:  spacing,top:
  HeightSpace),
          child: GridView.builder(
            shrinkWrap: true,
            physics: NeverScrollableScrollPhysics(),
           gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
              crossAxisCount: 4,
            ),
            itemCount: imagePaths1.length,
            itemBuilder: (context, index) {
                return Center(
                child: Container(
                  width: (screenHeight-iconSize)/4,
                  height: Razmer,
                  child: Column(
                    children: [
                      GestureDetector(
                        onHorizontalDragStart: (details) {
                          second = index;
                          // наступний екран
                          nextPage();
                        },
                        child: ClipRRect(
                        borderRadius: BorderRadius.circular(14.0),
                          child: Image.asset(
                          imagePaths1[index][0],
                            width: iconSize,
                          height: iconSize,
                          fit: BoxFit.cover,
                        ),
                      Padding(
                        padding:  EdgeInsets.only(left:  spacing,  right:
  spacing),
                        child: FittedBox(
                          fit:  BoxFit.none,  //  Це  зменшить  розмір  тексту,
  щоб він вписувався в доступний простір
                          child: Text(
                          imagePaths1[index][1],
                          style: TextStyle(
                              fontSize: 7,
                              color: Colors.white),
```

Рисунок 1.32. Фрагмент коду Page2

Принципово другий екран від першого відрізняється лише у 2-х речах. Відсутня можливість повернення на екран налаштувань. А також інші шляхи картинок і передається другий індекс! За такою ж схемою, як на першому

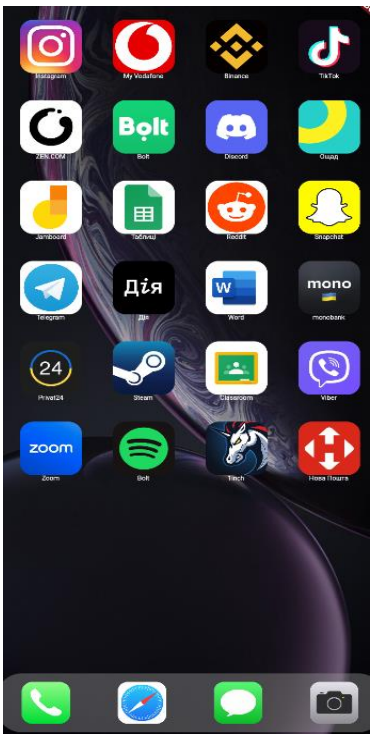



Рисунок 1.33 2 Экран

### 1.3.7 Створення 3 экрану

3й екран у нас будувався так само, як і перші 2. Але тепер у нас є додатковий функціонал. При натисканні на іконку "Нотатки" перші 2 індекси, які у нас передавалися з 1го та 2го екрану, поєднуються і видає фінальний індекс.

```
String OneString = OneIndex.toString();
  String TwoString = TwoIndex.toString();

  if (OneIndex > 0 && TwoIndex >= 0 ){
    String resultString = OneString + TwoString;
    int? finalIndex = int.tryParse(resultString);
    if (finalIndex != null) {
      return(finalIndex);
    } else {
      finalIndex = 0;
    }
  } else if (OneIndex < 0 && TwoIndex> 0 ){
    return TwoIndex;
  }
  return finalIndex;
```

Рисунок 1.34 Фрагмент коду схеми обробки фінального індекса

Цей фрагмент коду виконує операції конвертації та обробки значень `OneIndex` та `TwoIndex`, а потім повертає результат. 1. Спочатку значення `OneIndex` та `TwoIndex` конвертуються в рядковий формат за допомогою методу `toString()` і зберігаються в змінних `OneString` та `TwoString` відповідно. 2. Умовний оператор `if` перевіряє, чи `OneIndex` більше за 0 та `TwoIndex` більше або дорівнює 0. Якщо це так, вони об'єднуються в один рядок `resultString`, після чого цей рядок конвертується в ціле число `finalIndex`. Якщо конвертація пройшла успішно, повертається `finalIndex`, в іншому випадку `finalIndex` ініціалізується як 0. 3. Якщо `OneIndex` менше за 0 і `TwoIndex` більше за 0, то повертається значення `TwoIndex`. 4. Якщо жодна з цих умов не виконалась, повертається значення `finalIndex`. Цей код забезпечує правильне оброблення і повернення значення в залежності від значень `OneIndex` та `TwoIndex`.

```
class Page3 extends StatelessWidget {
  final Function nextPage;
  final Function previousPage;
  final Function updateShowBottomPanel;

  Page3(
      {required this.nextPage, required this.previousPage, required
this.updateShowBottomPanel});

  List<List<String>> imagePaths1 = [
    ['assets/images3/notes.png', 'Нотатки'],
    // Додати інші іконки тут
  ];


  @override
  Widget build(BuildContext context) {
    double screenWidth = MediaQuery
        .of(context)
        .size
        .width;
    double screenHeight = MediaQuery
        .of(context)
        .size
        .height;
    double iconSize = (screenWidth / 13) * 2;
    double spacing = (screenWidth / 13) / 2;
    double labelSize = iconSize / 4.5;
    double HeightSpace = screenHeight * 0.045;
    double Razmer = iconSize + HeightSpace;

    return Scaffold(
```

```
      backgroundColor: Colors.transparent,
      body: Padding(
        padding: EdgeInsets.only(
            right: spacing, left: spacing, top: HeightSpace),
        child: GridView.builder(
          shrinkWrap: true,
          physics: NeverScrollableScrollPhysics(),
          gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
            crossAxisCount: 4,
          ),
          itemCount: imagePaths1.length,
          itemBuilder: (context, index) {
            return Center(
              child: Container(
                width: (screenHeight - iconSize) / 4,
                height: Razmer,
                child: Column(
                  children: [
                    GestureDetector(
                      onTap: () {
                        int? noteNumber = calculateNoteNumber(first, second);
                        if (noteNumber != null && noteNumber >= 0) {
                          List<List<String>>           listOfLists           =
createListOfLists(noteNumber);

                          Navigator.of(context).push(
                            MaterialPageRoute(
                              builder:    (context)    =>    NotesScreen(myList:
listOfLists),
                            ),
                          );
                        }
                      },
                      child: ClipRRect(
                        borderRadius: BorderRadius.circular(14.0),
                        child: Image.asset(
                          imagePaths1[index][0],
                          width: iconSize,
                          height: iconSize,
                          fit: BoxFit.cover, //
                        ),
                      ),
                    ),
                    Padding(
                      padding: EdgeInsets.only(left: spacing, right: spacing),
                      child: FittedBox(
                        fit: BoxFit.none,
                        // зменшення тексту
                        child: Text(
                          imagePaths1[index][1],
                          style: TextStyle(
                              fontSize: 7,
                              color: Colors.white),
                        ),
                      ),
```

Рисунок 1.35 Фрагмент коду Page3

Як ми можемо побачити далі, викликається нове вікно (экран нотаток) і в нього передається фінальний індекс.

### 1.3.8 Створення  экрану нотаток

В окремому файлі. notes_screen. dart я прописав наступний код

```
import 'package:flutter/material.dart';
import 'app_settings.dart'; //сторінка налаштувань



class NotesScreen extends StatelessWidget {
  final List<List<String>> myList;
  final AppSettings appSettings = AppSettings();

  NotesScreen({required this.myList});

  bool isDigit(String char) {
    // Перевірка, чи є символом числом
    return int.tryParse(char) != null;
  }

  @override
  Widget build(BuildContext context) {
    List<Widget> listOfTiles = [];

    for (int index = 0; index < appSettings.selectedTemplates.length;
index++) {
      final template = appSettings.selectedTemplates[index];
      List<String> ZametkiList = myList[index];
      ListTile newTile = ListTile(
        title: Text(template.title),
        subtitle: Text("10.06.24 ${ZametkiList[0]}"),
        onTap: () {
          openNoteDetailScreen(context, template.title, ZametkiList);
        },
      );

      listOfTiles.add(newTile);
    }

    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.orangeAccent,
        title: Text("Нотатки"),
      ),
      body: Column(
        children: <Widget>[
          ListTile(
            title: Text("Купити води"),
            subtitle: Text("11.06.24 Купити води"),
            onTap: () {
            },
          ),
          ListTile(
            title: Text("7-00 Зустріч"),
```

```
            subtitle: Text("10.06.24 Максим"),
            onTap: () {
            },
          ),
          ...listOfTiles,
          ListTile(
            title: Text("Закупка"),
            subtitle: Text("10.06.24 Молоко"),
            onTap: () {
            },
          ),
// Додаємо вже згенеровані ListTile
        ],
      ),
    );
```

Рисунок 1.36 Фрагмент коду NotesScreen

Цей віджет `NotesScreen` відображає сторінку з нотатками. У цьому віджеті використовується список `myList`, який містить дані про нотатки для кожного шаблону. У методі `build` створюється список `listOfTiles`, який містить `ListTile` для кожного вибраного шаблону. Кожен `ListTile` має назву та підзаголовок, що відображаються з додатковою інформацією про нотатку. Нотатки, які відображаються за замовчуванням (наприклад, "Купити води" та "7-00 Зустріч"), також включені безпосередньо в `body` віджету. Цей віджет також містить `AppBar` з назвою "Нотатки".

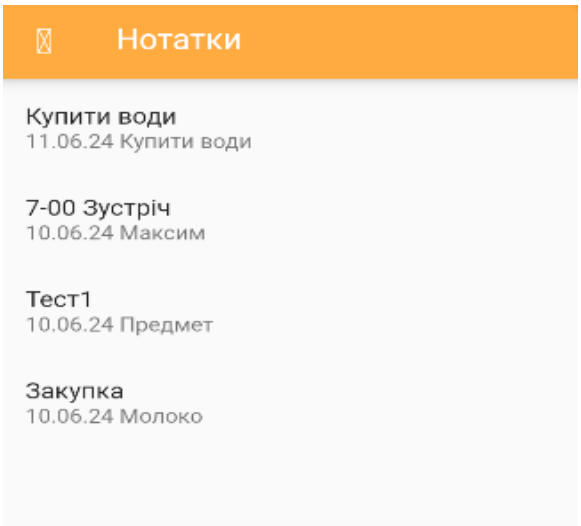



Рисунок 1.37 Экран нотаток

При натисканні на наші "шаблонні нотатки" викликається наступне вікно

```
class NoteDetailScreen extends StatelessWidget {
  final String noteTitle;
  final List<String> myList;

  NoteDetailScreen({required this.noteTitle, required this.myList});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.orangeAccent,
        title: Text(""),
      ),
      body: Padding(
        padding: EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: <Widget>[
            Text(
              noteTitle,
              style: TextStyle(
                fontSize: 24,
                fontWeight: FontWeight.bold,
              ),
            ),
            Expanded(
              child: ListView.separated(
                itemCount: myList.length,
                separatorBuilder:   (context,   index)   =>   SizedBox(),   //
Забираємо роздільник
                itemBuilder: (context, index) {
                  return ListTile(
                    title: Text('${index + 1}. ${myList[index]}'), // Додаємо
номер перед елементом
                    // Інші властивості елемента списку, якщо потрібно
                  );
                },
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

Рисунок 1.38 Фрагмент коду NoteDetailScreen

Відкриваються деталі "нотатки". І на місці фінального індексу у нас виходить предмет. А якщо індекс більший за довжину списку він ставиться на останнє місце. Ось кілька прикладів.


Рисунок 1.39 Положення при FinalIndex 2


Рисунок 1.40 Положення при FinalIndex 8

Віджет NoteDetailScreen заповнюється вмістом залежно від переданих з попереднього екрану даних, використовуючи шаблони для відображення заголовка та списку елементів. Коли цей екран створюється, він отримує два

параметри: noteTitle і myList. Ці параметри передаються через конструктор і визначають вміст, який буде відображено.

Заголовок замітки (noteTitle) відображається у вигляді текстового віджету, який використовує шаблон для стилізації. Цей шаблон включає збільшений розмір шрифту та жирне накреслення, щоб зробити заголовок більш помітним і читабельним.

Список елементів (myList) заповнюється за допомогою ListView.separated, де кожен елемент списку відображається у вигляді ListTile. Для кожного елемента використовується шаблон, який включає текстове поле з номером елемента і його вмістом. Нумерація додається перед текстом кожного елемента, що забезпечує структурованість і легкість сприйняття списку.

Таким чином, вміст NoteDetailScreen залежить від даних, переданих з попереднього екрану, і заповнюється за допомогою шаблонів, що забезпечує узгоджений і привабливий інтерфейс.

Якщо ж ми не вибрали жодного шаблону, то будуть показані тільки стандартні вбудовані нотатки з якими не можна взаємодіяти

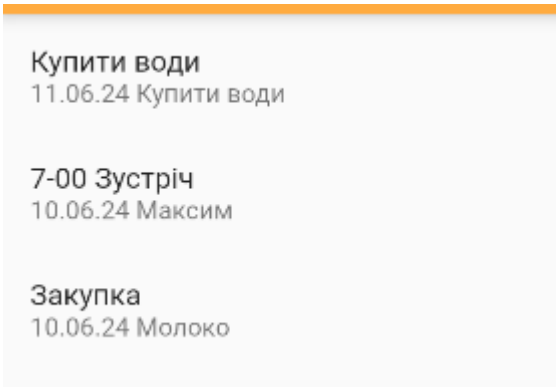



Рисунок 1.41 Коли не обрано жодного шаблона

# 2 ЕКОНОМІЧНИЙ РОЗДІЛ:

## 2.1 Резюме

У даному дипломному проєкті розроблено мобільний застосунок для фокусів, який надає користувачам можливість навчатися та виконувати різноманітні фокуси. Цей застосунок розроблений з використанням Flutter, що дозволяє забезпечити його сумісність з платформами Android та iOS.

Ефективність мобільного додатку визначається його якістю та ефективністю процесу розробки. Якість додатку визначається такими складовими: з точки зору користувача, використання ресурсів, та відповідність вимогам до програмного забезпечення.

Оцінка якості мобільного додатку з точки зору користувача визначається необхідним обсягом оперативної пам'яті, витратами машинного часу, та пропускною спроможністю каналів передачі даних. Оцінка якості включає визначення трудомісткості та вартості його створення.

## 2.2 Розрахунок ціни мобільного додатку нормативним методом

### 2.2.1 Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки мобільного додатку залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку. Методом структурної аналогії по відповідних каталогах аналогів програмного забезпечення визначається обсяг програмних засобів, у тисячах умовних машинних команд програми аналога.

Таблиця 1.3 Аналоги програмного забезпечення

| Найменування ПП | Обсяг функції ПП – $V_o$, усл. машинних командах |
|---|---|
| 1. ПП автоматизації засобів по каталогу | 680 – 7000 |
| 2. ПП автоматизованих розрахунків | 1300 – 8600 |
| 3. ПП введення інформації | 1060 – 5750 |

Вибравши аналог ПП, що містить Vo в умовних машинних командах, трудомісткість визначаємо на основі Таблиці 1.4

Таблиця 1.4. Норма часу на розробку аналога програмного забезпечення

| Обсяг ПП, тис.умов.машинних команд | Норма часу, люд/год |
|---|---|
| 1.00 | 229 |
| 2.00 | 244 |
| 3.00 | 262 |
| 4.00 | 283 |
| 5.00 | 306 |
| 6.00 | 330 |
| 7.00 | 357 |
| 8.00 | 385 |
| 9.00 | 414 |
| 10.00 | 445 |

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8): Тар=229х0,8=183,2 (люд/годин).

$$T_{ТЗ} = T^a p \times L_1 \times K_H \quad (1.3)$$

$$T_{ТП} = T^a p \times L_2 \times K_H \quad (1.4)$$

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \quad (1.5)$$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. Табл. 1.3)

Кн –поправочний коефіцієнт, що враховує ступінь новизни (див. Табл. 1.3)

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. Табл 1.5)

Таблиця 1.5. Значення питомих коефіцієнтів трудомісткості стадії

| Код стадії | Ступінь новизни | | |
|---|---|---|---|
| | А | Б | В |
| ТЗ ($L_1$) | 0,15 | 0,12 | 0,12 |
| ТП ($L_2$) | 0,16 | 0,15 | 0,11 |
| РП ($L_3$) | 0,55 | 0,58 | 0,61 |

Для нашого варіанта виділено сірим кольором

Таблиця 1.6. Поправочний коефіцієнт, що враховує ступінь новизни

| Код ступеня новизни | Ступінь новизни | Значення $К_н$ |
|---|---|---|
| А | Принципово нові ПО | 1,75 – 1,2 |
| Б | ПО – розвиток визначеног параметричного ряду | 1,0 – 0,8 |
| В | ПО маючий аналог | 0,7 |

Для нашого варіанта виділено сірим кольором.

Таблиця 1.7. Коефіцієнт ступеня використання типових програм

| Ступінь охоплення реалізованих функцій розроблювального ПО типовими програмами, % | Значення $К_т$ |
|---|---|
| 60 і вище | 0,6 |

| | |
|---|---|
| 40-60 | 0,7 |
| 20-40 | 0,8 |
| До 20 | 0,9 |

Для розробки нашого мобільного додатку використовується 40-60% існуючих функцій, тому Кт = 0,7.

### 2.2.2 Розрахунок трудомісткості по кожному етапу розробки

Таблиця 1.8. Трудомісткість по кожному етапу

| Найменування етапів | Розрахунок, годин. | | |
|---|---|---|---|
| 1.ТЗ | $T_{Pтз}$=15,39 | $T_{кк}=0,7*N_{тз}=$ 0,7*2=1,4 | $T_{нк}=0,15*N_{тз}=$ 0,15*2=0,30 |
| 2.Розробка ТП | $T_{Pтп}$=14,12 | $T_{кк}=0,7*N_{тп}=$ 0,7*40=28 | $T_{нк}=0,15*N_{тп}=$ 0,15*40=6.0 |
| 3.Розробка РП | $T_{Pрп}$=54,76 | $T_{кк}=0,7*N_{рп}=$ 0,7*9=6,3 | $T_{нк}=0,15*N_{рп}=$ 0,15*9=1,35 |
| 4.Розробка ПЗ | $T_{пз}=1,5**N_{пз}=$ 1,5*25 =37,5 | $T_{кк}=0,7*N_{тз}=$ 0,7*25=17,5 | $T_{нк}=0,15*N_{пз}=$ 0,15*25 =3,75 |
| Усього, в т.ч.: | 186,37 | | |
| на розробку | $T_{р}$=121,77 | | |
| -контроль | | $T_{кк}$=53,2 | |
| - нормоконтроль | | | $T_{нк}$=11,4 |

### 2.2.3 Розрахунок основної заробітної плати виконавців

Відповідно до статті 8 «Закону про Державний бюджет України на 2023» встановлено мінімальну заробітну плату у місячному розмірі з 1 квітня 2024 року – 8000 гривень; мінімальну погодинну тарифну ставку – 46 грн.

Таблиця 1.9. Розрахунок основної заробітної плати виконавців

| Найменування робіт | Трудомісткість робіт, години | Погодинна тарифна ставка, грн. | Розрахунок, грн. |
|---|---|---|---|
| 1.Розробка ПП | 121,77 | 46,00 | 5601,42 |
| 2.Контроль керівника | 53,20 | 80,00 | 4256,00 |
| 3.Нормоконтроль | 11,4 | 80,00 | 912,00 |
| Усього | - | - | Зо= 10769,42 |

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в табл. 1.10

### 2.2.4 Розрахунок матеріальних витрат на розробку ПП

Таблиця 1.10. Розрахунок матеріальних витрат

| Найменування матеріальних витрат | Тип, модель | Кількість | Ціна одиниці, грн. | Вартість, грн. |
|---|---|---|---|---|
| Папір | Лист А4 | 96 | 4.00 | 384,0 |
| Транспортно – заготівельні Витрати (10%) | | | | $Втр\_з = 0{,}1 \times Вм1 =$ 0,1х384=38.4 |
| Усього | | | | $В_м = В_{мі} + В_{тр\_з} = 422.4$ |
| Папір | Лист А4 | 96 | 4.00 | 422.4 |

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в табл. 1.11.

### 2.2.5 Розрахунок статей витрат планової собівартостъ

Таблиця 1.11. Розрахунок статей витрат

| Стаття витрат | Значення, грн | Формула розрахунку |
|---|---|---|
| Матеріали | 422.4 | |
| Основна заробітна плата | 10066,89 | |
| Додаткова заробітна плата | 1006,67 | 0,1 * 10066,89 |
| Відрахування до ЄФСВ | 2436,19 | 0,22 * (10066,89 + 1006,67) |
| Накладні витрати | 12080,27 | |
| Разом витрати (С) | 25988,52 | |
| Прибуток (П) | 779,63 | |
| Виробнича собівартість (Сп) | 26772,15 | 25892,52 + 779,63 |
| ПДВ (20%) | 5334,43 | 0,2 * 26672,15 |
| Повна собівартість (Сп) | 32107,7 | |

### 2.3.6 Висновки

В результаті розробки мобільного додатку для фокусів було визначено його собівартість, яка складає 32107,7 гривень. Цей застосунок призначений для використання як на платформі Android, так і на iOS, що дозволяє охопити широку аудиторію користувачів. Завдяки розрахунку витрат та оцінці ринку, ми впевнені, що цей застосунок буде комерційно успішним і сприятиме розвитку навичок улюблених користувачів фокусів.

# 3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Охорона праці означає систему заходів, спрямованих на збереження життя, здоров'я та працездатності людини під час виконання роботи. Покращення методів забезпечення безпеки праці є важливим шляхом підвищення ефективності виробництва, оскільки травматизм є значною причиною непродуктивних втрат робочої години. Безпека праці також впливає на продуктивність, оскільки високу продуктивність може бути досягнуто лише в умовах безпечного працюючого середовища. У рамках розділу про охорону праці в дипломному проекті вирішується питання забезпечення безпеки програміста під час розробки за ПК.

## 3.1 Аналіз та безпека умов праці працівника на робочому місці

Охорона праці означає систему заходів, спрямованих на збереження життя, здоров'я та працездатності людини під час виконання роботи. Покращення методів забезпечення безпеки праці є важливим шляхом підвищення ефективності виробництва, оскільки травматизм є значною причиною непродуктивних втрат робочої години. Безпека праці також впливає на продуктивність, оскільки високу продуктивність може бути досягнуто лише в умовах безпечного працюючого середовища. В розділі охорони праці дипломного проекту розглядається питання розробки мобільного додатку для виступів. Тому об’єктом дослідження беремо безпеку праці на робочому місці програміста.

При кольоровому оформленні виробничих і допоміжних приміщень необхідно враховувати орієнтацію їхніх вікон стосовно частин світу і використовувати гармонійне сполучення кольорів. Для стін і робочих поверхонь використовують мало насичені (основні) кольори, для невеликих помешкань або ділянок, що рідко потрапляють у поле зору працюючих, а також для створення контрастності – кольори середньої насиченості (допоміжні), для маленьких по площі поверхонь – насичені (акценти) – як функціональне фарбування. Стелі у всіх приміщеннях повинні бути білими. Поверхні

устаткування в приміщеннях повинні бути матовими або напівматовий, для виключення випадку відблисків світла в очі працюючого, а стіни бути пофарбованими фарбами пастельних тонів

Забезпечення безпечних та здорових умов праці значною мірою залежить від правильної оцінки небезпечних та шкідливих факторів, які можуть впливати на працюючу людину. Різні фактори, такі як виробниче середовище, фізичні та розумові навантаження, стрес і т.д., можуть спричиняти складні зміни у фізичному стані людини. Оператори та програмісти можуть стикатися з різними фізично небезпечними та шкідливими факторами, такими як підвищений рівень шуму, підвищена температура, недостатнє освітлення, електричний струм, статична електрика та інші. На робочому місці програміста необхідно створити умови для безпечної та продуктивної праці, що включає в себе не лише усунення фізичних та ергономічних стресів, але й забезпечення відповідної психологічної атмосфери та можливості регулярних перерв для відпочинку та відновлення. Вимоги Державних санітарних правил і норм 3.3.2.007-98 визначають об'ємно-планувальні рішення будівель та приміщень для роботи з ВДТ. У приміщеннях з робочими місцями важливо забезпечити правильне освітлення, яке сприяє комфортній та продуктивній праці. Зазвичай використовується система загального рівномірного освітлення, що розподіляє світло по всьому приміщенню рівномірно. Якщо діяльність передбачає переважну роботу з документами або інші види робіт, які потребують більшого концентрованого освітлення, можна використовувати систему комбінованого освітлення, яка поєднує загальне освітлення з додатковими світильниками місцевого освітлення, щоб забезпечити більшу якість освітлення на конкретному робочому місці.

На робочому місці програміста повинні бути створені умови для безпечної та високопродуктивної праці.

Освітлення приміщення має природне та штучне походження. Природне освітлення подається через віконні прорізі, бокове. Для штучного освітлення у приміщенні використовуються люмінесцентні лампи, які в порівнянні з

лампами розжарювання мають ряд істотних переваг. Так за спектральним складом світла вони близькі до природного світла, мають підвищену світлову віддачу, триваліший термін служби. Норма освітленості на робочих місцях складає 300-500лк.

Приміщення, в яких планується установка та подальша робота з комп'ютером, повинні відповідати проектній документації будинку. Роботодавець повинен враховувати санітарні нормативи освітлення, вимоги до параметрів мікроклімату (температура, відносна вологість), ступеня і сили вібрації, звукового шуму і вогнестійкості приміщення, а також характеристики електромагнітного, ультрафіолетового та інфрачервоного полів

Приміщення для роботи з ПК мають мати природне та штучне освітлення відповідно до вимог ДБН В.2.5-28-2006. Особливу увагу слід звернути на вікна, які, у разі роботи з відео терміналами, краще розташовувати з орієнтацією на північ або північний захід. Щоб регулювати рівень освітленості та захищати від прямого сонячного світла, на вікнах доцільно встановлювати штори або жалюзі. Для штучного освітлення рекомендується використовувати люмінесцентні лампи типу ЛБ, які мають ряд переваг у порівнянні з лампами розжарювання, таких як близький до природного світла спектральний склад, підвищена світлова віддача та триваліший термін служби до 10 тисяч годин. Щодо параметрів повітря у робочій зоні, температура повітря має коливатися від 16°С до 25°С, залежно від сезону та складності роботи. Допустимий діапазон становить від 12°С до 30°С. Оптимальна відносна вологість повітря повинна бути в межах 40-60%, з допустимим підвищенням до 75%. Для підтримки нормального складу повітря та видалення шкідливих речовин використовується вентиляція. Це може бути як механічна вентиляція (кондиціонери, вентилятори), так і природна вентиляція через вікна. У виробничих приміщеннях на робочих місцях необхідно забезпечувати оптимальні значення параметрів мікроклімату згідно з ГОСТ 12.1.005-88 та СН 4088-86: температури, відносної вологості та рухливості повітря.

Вібрація - це рух твердих тіл або частин обладнання, який сприймається людиною як струс і часто супроводжується шумом. Вона може негативно впливати на центральну нервову систему, травну систему, вестибулярний апарат, викликати запаморочення, оніміння кінцівок та спричиняти захворювання суглобів. Тривалий вплив вібрації може призвести до професійного захворювання, відомого як вібраційна хвороба. Шум є формою фізичного (хвильового) забруднення навколишнього середовища. Він може дратувати, заважаючи працювати, відпочивати та зосереджуватися. Шкідливий вплив шуму на людське життя добре відомий. У разі розумової праці, яка вимагає зосередженості, припустимий рівень шуму становить 50 дБ. Для зменшення шуму і вібрації в приміщенні обладнання, апарати і прилади встановлюють на спеціальні прокладки, що амортизують.

Пожежна безпека охоплює комплекс заходів, спрямованих на захист людей і майна від вогню. В приміщеннях з електричними мережами діють нормативні документи, такі як ГОСТ 12.1.033-81 та ГОСТ 12.1.004-85, які визначають вимоги до пожежної безпеки. Робота оператора ЕОМ повинна здійснюватися в приміщенні категорії Д пожежної безпеки, де зберігаються негорючі матеріали в холодному стані. Усі приміщення повинні мати необхідне обладнання для пожежогасіння, таке як пожежні крани, пожежні щити з відповідним інструментарієм, а також вуглекислотні або порошкові вогнегасники. У випадку пожежі слід негайно відключити електропостачання, зателефонувати за номером 101 для виклику пожежної команди, евакуювати людей за планом евакуації і розпочати ліквідацію пожежі.

# Висновки

У результаті цього проекту було розроблено зручний застосунок для фокусів на базі Flutter. Цей застосунок допомагає розвивати мислення та вдосконалює навички магії. Використання Flutter дає можливість створювати додатки з чудовою продуктивністю та привабливим інтерфейсом, а Android Studio як інтегроване середовище розробки (IDE) спрощує процес створення, тестування та налагодження додатків. Результатом розробки є зручний застосунок для фокусів, який ефективно виконує передбачення фокусника та збійснює ефект нав'язування. Застосунок дозволяє глядачеві вибрати будь-яке число, під яким миттєво з'являється заздалегідь визначене значення. Важливо відзначити, що застосунок працює офлайн, що робить його доступним у будь-яку годину та в будь-якому місці. Особливістю цього додатка є те, що він імітує робочий екран телефону, що робить трюк ще більш вражаючим та загадковим для глядачів.

Мною було задіяно широкий спектр інструментів, що забезпечують ефективну та якісну розробку проєкту В пояснювальній записці розглянуті всі питання передбачені технічним завданням на дипломне проєктування, проведено аналіз предметної області; детально описано технології та засоби, які використовувалися при створенні проєкту; проведено розрахунок економічної ефективності створення та впровадження створеного проєкту; розглянуті питання охорони праці та наведений перелік використаних джерел.

# ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ


1. Що таке охорона праці?. [Веб-сайт]. URL: https://oppb.com.ua/news/chtotakoe-ohrana-truda. (дата звернення: 27.05.2024)
2. Охорона праці в офісі. [Веб-сайт]. URL: https://yur-gazeta.com/publications/practice/trudove-pravo/ohorona-praci-v-ofisi-.html#:~:text=Штучне%20освітлення%20в%20приміщеннях%20з,додатково%20встановлюються%20світильники%20місцевого%20освітлення) (дата звернення: 27.05.2024)
3. Що таке охорона праці?. [Веб-сайт]. URL: https://oppb.com.ua/news/chtotakoe-ohrana-truda. (дата звернення: 27.05.2024)
4. О. Шевченко. Крос-платформне програмування: Навчальний посібник. Львів: Вид-во ЛТЕУ, 2019.
5. Flutter Docs. [Веб-сайт]. URL: https://docs.flutter.dev/. (дата звернення: 25.05.2024)

# ДОДАТОК А. Програмний код основної логіки веб-застосунку

## Файл main.dart

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'app_settings.dart'; //страница налаштувань
import 'notes_screen.dart'; // панель нотаток




void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    SystemChrome.setEnabledSystemUIMode(SystemUiMode.manual, overlays: [SystemUiOverlay.top]);
    SystemChrome.setSystemUIOverlayStyle(SystemUiOverlayStyle(
      statusBarColor: Colors.transparent, // Укажите color
      statusBarIconBrightness: Brightness.light, // Вкажіть бажану яскравість піктограм (чорні або білі).
    ));
    return MaterialApp(
      home: WelcomeScreen(),
    );
  }
}

class WelcomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ElevatedButton(
              onPressed: () {
                Navigator.of(context).pushReplacement(
                  MaterialPageRoute(
                    builder: (context) => MyPageView(),
                  ),
                );
              },
```

```
            child: Text("Почати показ"),
          ),
          SizedBox(height: 16),
          ElevatedButton(
            onPressed: () {
              Navigator.of(context).push(
                MaterialPageRoute(
                  builder: (context) => SettingsScreen(),
                ),
              );
            },
            child: Text("Налаштування"),
          ),
          SizedBox(height: 16),
          ElevatedButton(
            onPressed: () {
              SystemNavigator.pop();
            },
            child: Text("Вихід"),
          ),
        ],
      ),
    ),
  );
 }
}

class MyPageView extends StatefulWidget {

 @override
 _MyPageViewState createState() => _MyPageViewState();
}

class _MyPageViewState extends State<MyPageView> {
 PageController _pageController = PageController(initialPage: 0);
 double _startX = 0;
 double _startY = 0;
 bool showBottomPanel = true; // Змінна для показу/приховування нижньої панелі


 void _updateShowBottomPanel(bool show) {
  setState(() {
   showBottomPanel = show;
  });
 } //можливість приховати нижню панель
 List<String> imagePaths = [
  'assets/lowbarimages/phone.png',
  'assets/lowbarimages/safari.png',
  'assets/lowbarimages/ios-message.png',
  'assets/lowbarimages/camera.png',
 ];
```

```
@override
void initState() {
  super.initState();
}

void nextPage() {
  _pageController.nextPage(duration: Duration(milliseconds: 600), curve: Curves.ease);
}

void previousPage() {
  _pageController.previousPage(duration: Duration(milliseconds: 600), curve: Curves.ease);
}

Widget build(BuildContext context) {
  return Scaffold(
    body: GestureDetector(
        onHorizontalDragStart: (details) {
          _startX = details.localPosition.dx;
          _startY = details.localPosition.dy;
        },
        onHorizontalDragUpdate: (details) {
          double dx = details.localPosition.dx - _startX;

          if (dx > 0) {
            previousPage();
          } else if (dx < 0) {
            nextPage();
          }
        },
        child: Stack(
          children: <Widget>[
            Container(
              decoration: BoxDecoration(
                image: DecorationImage(
                  image: AssetImage('assets/background/background.jpg'),
                  fit: BoxFit.cover,
                ),
              ),
            ),
            if (showBottomPanel) // Перевірка, показувати чи ні нижню панель
              Positioned(
                left: 9,
                right: 9,
                bottom: 9,
                child: Container(
                  height: 70.0,
                  decoration: BoxDecoration(
                    color: Colors.grey.withOpacity(0.4),
                    borderRadius: BorderRadius.circular(25.0),
                  ),
                  child: Row(
```

```
          mainAxisAlignment: MainAxisAlignment.spaceAround,
          children: List.generate(4, (index) {
            return GestureDetector(
              onTap: () {
                print("Нажал кнопку с нижней панели $index");
                // Обробник натискання елемент нижньої панелі
              },
              child: Column(
                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                children: [
                  Padding(
                    padding: EdgeInsets.all(8.0),
                    child: Image.asset(
                      imagePaths[index],
                      width: 50.0,
                      height: 50.0,
                    ),
                  ),
                ],
              ),
            );
          }),
        ),
      ),
    ),
  PageView(
    controller: _pageController,
    children: <Widget>[
      Navigator(
        onGenerateRoute: (settings) {
          return MaterialPageRoute(
            builder: (context) => Page1(
              nextPage: nextPage,
              previousPage: previousPage,
              updateShowBottomPanel: _updateShowBottomPanel,
            ),
          );
        },
      ),
      Navigator(
        onGenerateRoute: (settings) {
          return MaterialPageRoute(
            builder: (context) => Page2(
              nextPage: nextPage,
              previousPage: previousPage,
              updateShowBottomPanel: _updateShowBottomPanel,
            ),
          );
        },
      ),
      Navigator(
        onGenerateRoute: (settings) {
```

```
                return MaterialPageRoute(
                  builder: (context) => Page3(
                    nextPage: nextPage,
                    previousPage: previousPage,
                    updateShowBottomPanel: _updateShowBottomPanel,
                  ),
                );
              },
            ),
          ],
        ),
      ],
    )
  ),
 );
}



}

int first = 0;
int second = 0;

int calculateNoteNumber(int firstIndex, int secondIndex) {
  int? OneIndex;
  int? TwoIndex;
  int? finalIndex = 0;

  // Обробка першого індексу
  switch (firstIndex) {
    case 13:
    case 14:
    case 15:
      OneIndex = firstIndex - 12;
      break;
    case 17:
    case 18:
    case 19:
      OneIndex = firstIndex - 13;
      break;
    case 21:
    case 22:
    case 23:
      OneIndex = firstIndex - 14;
      break;
    default:
      OneIndex  = -1;
  }

  // Обробка 2 індексу
  switch (secondIndex) {
```

```
    case 9:
      TwoIndex = 0;
      break;
    case 13:
    case 14:
    case 15:
      TwoIndex = secondIndex - 12;
      break;
    case 17:
    case 18:
    case 19:
      TwoIndex = secondIndex - 13;
      break;
    case 21:
    case 22:
    case 23:
      TwoIndex = secondIndex - 14;
      break;
    default:
      TwoIndex = -1;
  }

  String OneString = OneIndex.toString();
  String TwoString = TwoIndex.toString();

  if (OneIndex > 0 && TwoIndex >= 0 ){
    String resultString = OneString + TwoString;
    int? finalIndex = int.tryParse(resultString);
    if (finalIndex != null) {
      return(finalIndex);
    } else {
      finalIndex = 0;
    }

  } else if (OneIndex < 0 && TwoIndex> 0 ){
    return TwoIndex;

  }


  return finalIndex;
} //обробляємо індекси іконок

List<List<String>> createListOfLists(int noteNumber) {
  List<List<String>> listOfLists = []; // Створюємо пустий список для зберігання інших списків

  AppSettings appSettings = AppSettings();
  appSettings.selectedTemplates.forEach((template) {
    List<String> newList = [];
    newList.addAll(template.list);
    if (newList.length >= noteNumber) {
      newList[noteNumber - 1] = template.item; //Заміна у списку предмета за індексом
```

```
      listOfLists.add(newList);
    } else {
      newList[newList.length - 1] = template.item; //Додавання до кінця списку у разі перевищення індексу
      listOfLists.add(newList);
    }
  });
  print(listOfLists);

  return listOfLists;
}



class Page1 extends StatelessWidget {
  final Function nextPage;
  final Function previousPage;
  final Function updateShowBottomPanel;

  Page1({required this.nextPage, required this.previousPage, required this.updateShowBottomPanel});



  List<List<String>> imagePaths1 = [
    ['assets/images1/weather.png', 'Погода'],
    ['assets/images1/find-my.png', 'Локатор'],
    ['assets/images1/my-shortcuts.png', 'Команди'],
    ['assets/images1/home.png', 'Дім'],
    ['assets/images1/contacts.png', 'Контакти'],
    ['assets/images1/files.png', 'Файли'],
    ['assets/images1/translate.png', 'Переклад'],
    ['assets/images1/books.png', 'Книги'],
    ['assets/images1/facetime.png', 'FaceTime'],
    ['assets/images1/itunes.png', 'iTunes Store'],
    ['assets/images1/photo.png', 'Фото'],
    ['assets/images1/mail.png', 'Пошта'],
    ['assets/images1/measure.png', 'Рулетка'],
    ['assets/images1/reminders.png', 'Нагадування'],
    ['assets/images1/clock.png', 'Годинник'],
    ['assets/images1/apple-tv.png', 'TV'],
    ['assets/images1/podcasts.png', 'Подкасти'],
    ['assets/images1/app-store.png', 'App Store'],
    ['assets/images1/apple-map.png', 'Карти'],
    ['assets/images1/apple-music.png', 'Музика'],
    ['assets/images1/wallet.png', 'Wallet'],
    ['assets/images1/settings.png', 'Налаштування'],
    ['assets/images1/Gooogle_aut.png', 'Authenticator'],
    ['assets/images1/health.png', 'Health'],


  ];
```

```
@override
Widget build(BuildContext context) {
  double screenWidth = MediaQuery.of(context).size.width;
  double screenHeight = MediaQuery.of(context).size.height;
  double iconSize = (screenWidth / 13) * 2;
  double spacing = (screenWidth / 13) / 2;
  double labelSize = iconSize/4.5;
  double HeightSpace = screenHeight*0.045;
  double Razmer = iconSize+HeightSpace;



  return Scaffold(
    backgroundColor: Colors.transparent,
    body: Padding(
      padding: EdgeInsets.only(right: spacing, left: spacing,top: HeightSpace),
      child: GridView.builder(
        shrinkWrap: true,
        physics: NeverScrollableScrollPhysics(),
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 4,
        ),
        itemCount: imagePaths1.length,
        itemBuilder: (context, index) {

          return Center(
            child: Container(
              width: (screenHeight-iconSize)/4,
              height: Razmer,
              child: Column(
                children: [
                  GestureDetector(
                    onVerticalDragStart: (details) {
                      Navigator.of(context).push(
                        MaterialPageRoute(
                          builder: (context) => WelcomeScreen(),
                        ),
                      );
                      // При свайпе вверх  WelcomeScreen
                    },
                    onHorizontalDragStart: (details) {
                      first = index;
                      // Додайте код для переходу на наступний екран
                      nextPage();
                    },
                    child: ClipRRect(
                      borderRadius: BorderRadius.circular(14),
                      child: Image.asset(
                        imagePaths1[index][0],
                        width: iconSize,
                        height: iconSize,
```

```
                    fit: BoxFit.cover,
                  ),
                ),
              ),
              Padding(
                padding: EdgeInsets.only(left: spacing, right: spacing),
                child: FittedBox(
                  fit: BoxFit.none, // Це зменшить розмір тексту, щоб він вписувався в доступний простір
                  child: Text(
                    imagePaths1[index][1],
                    style: TextStyle(
                        fontSize: 7,
                        color: Colors.white),
                  ),
                ),
              ),

            ],
          ),
        ),
      );
    },
  ),
 ),
 );
 }
}
class Page2 extends StatelessWidget {
  final Function nextPage;
  final Function previousPage;
  final Function updateShowBottomPanel;

  Page2({required this.nextPage, required this.previousPage, required this.updateShowBottomPanel});

  List<List<String>> imagePaths1 = [
    ['assets/images2/instagram.png', 'Instagram'],
    ['assets/images2/vodafone.png', 'My Vodafone'],
    ['assets/images2/binance.png', 'Binance'],
    ['assets/images2/tiktok-app-icon.png', 'TikTok'],
    ['assets/images2/zen.png', 'ZEN.COM'],
    ['assets/images2/Bolt.png', 'Bolt'],
    ['assets/images2/discord-square-color-icon.png', 'Discord'],
    ['assets/images2/oshad.png', 'Ощад'],
    ['assets/images2/GJamboard.png', 'Jamboard'],
    ['assets/images2/googlesheets.png', 'Таблиці'],
    ['assets/images2/reddit.png', 'Reddit'],
    ['assets/images2/snapchat.png', 'Snapchat'],
    ['assets/images2/telega.png', 'Telegram'],
    ['assets/images2/diya.png', 'Дія'],
    ['assets/images2/word.png', 'Word'],
```

```
  ['assets/images2/mono.png', 'monobank'],
  ['assets/images2/privat24.png', 'Privat24'],
  ['assets/images2/steam.png', 'Steam'],
  ['assets/images2/GClassRoom.png', 'Classroom'],
  ['assets/images2/viber.png', 'Viber'],
  ['assets/images2/zoom (1).png', 'Zoom'],
  ['assets/images2/spotyfi.webp', 'Bolt'],
  ['assets/images2/1inch.png', '1inch'],
  ['assets/images2/novaposhta.png', 'Нова Пошта'],



];

@override
Widget build(BuildContext context) {
  double screenWidth = MediaQuery.of(context).size.width;
  double screenHeight = MediaQuery.of(context).size.height;
  double iconSize = (screenWidth / 13)*2;
  double spacing = (screenWidth / 13) / 2;
  double labelSize = iconSize/4.5;
  double HeightSpace = screenHeight*0.045;
  double Razmer = iconSize+HeightSpace;

  return Scaffold(
    backgroundColor: Colors.transparent,
    body: Padding(
      padding: EdgeInsets.only(right: spacing, left: spacing,top: HeightSpace),
      child: GridView.builder(
        shrinkWrap: true,
        physics: NeverScrollableScrollPhysics(),
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 4,
        ),
        itemCount: imagePaths1.length,
        itemBuilder: (context, index) {

          return Center(
            child: Container(
              width: (screenHeight-iconSize)/4,
              height: Razmer,
              child: Column(
                children: [
                  GestureDetector(
                    onHorizontalDragStart: (details) {
                      second = index;
                      // наступний екран
                      nextPage();
                    },
                    child: ClipRRect(
                      borderRadius: BorderRadius.circular(14.0),
                      child: Image.asset(
```

```
                    imagePaths1[index][0],
                    width: iconSize,
                    height: iconSize,
                    fit: BoxFit.cover,
                  ),
                ),
              ),
              Padding(
                padding: EdgeInsets.only(left: spacing, right: spacing),
                child: FittedBox(
                  fit: BoxFit.none, // Це зменшить розмір тексту, щоб він вписувався в доступний простір
                  child: Text(
                    imagePaths1[index][1],
                    style: TextStyle(
                        fontSize: 7,
                        color: Colors.white),
                  ),
                ),
              ),

            ],
          ),
        ),
      );
    },
  ),
 ),
 );
 }
}
class Page3 extends StatelessWidget {
 final Function nextPage;
 final Function previousPage;
 final Function updateShowBottomPanel;

 Page3(
    {required this.nextPage, required this.previousPage, required this.updateShowBottomPanel});

 List<List<String>> imagePaths1 = [
  ['assets/images3/notes.png', 'Нотатки'],
  // Додати інші іконки тут
 ];


 @override
 Widget build(BuildContext context) {
  double screenWidth = MediaQuery
      .of(context)
      .size
      .width;
  double screenHeight = MediaQuery
```

```
    .of(context)
    .size
    .height;
double iconSize = (screenWidth / 13) * 2;
double spacing = (screenWidth / 13) / 2;
double labelSize = iconSize / 4.5;
double HeightSpace = screenHeight * 0.045;
double Razmer = iconSize + HeightSpace;

return Scaffold(
  backgroundColor: Colors.transparent,
  body: Padding(
    padding: EdgeInsets.only(
        right: spacing, left: spacing, top: HeightSpace),
    child: GridView.builder(
      shrinkWrap: true,
      physics: NeverScrollableScrollPhysics(),
      gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 4,
      ),
      itemCount: imagePaths1.length,
      itemBuilder: (context, index) {
        return Center(
          child: Container(
            width: (screenHeight - iconSize) / 4,
            height: Razmer,
            child: Column(
              children: [
                GestureDetector(
                  onTap: () {
                    int? noteNumber = calculateNoteNumber(first, second);
                    if (noteNumber != null && noteNumber >= 0) {
                      List<List<String>> listOfLists = createListOfLists(noteNumber);

                      Navigator.of(context).push(
                        MaterialPageRoute(
                          builder: (context) => NotesScreen(myList: listOfLists),
                        ),
                      );
                    }
                  },
                  child: ClipRRect(
                    borderRadius: BorderRadius.circular(14.0),
                    child: Image.asset(
                      imagePaths1[index][0],
                      width: iconSize,
                      height: iconSize,
                      fit: BoxFit.cover, //
                    ),
                  ),
                ),
                Padding(
```

```
                padding: EdgeInsets.only(left: spacing, right: spacing),
                child: FittedBox(
                  fit: BoxFit.none,
                  // зменшення тексту
                  child: Text(
                    imagePaths1[index][1],
                    style: TextStyle(
                        fontSize: 7,
                        color: Colors.white),
                  ),
                ),
              ),

            ],
          ),
        ),
      );
    },
  ),
 ),
 );
 }
}
```

## Файл notes_screen.dart

```
import 'package:flutter/material.dart';
import 'app_settings.dart'; //сторінка налаштувань



class NotesScreen extends StatelessWidget {
  final List<List<String>> myList;
  final AppSettings appSettings = AppSettings();

  NotesScreen({required this.myList});

  bool isDigit(String char) {
    // Перевірка, чи є символом числом
    return int.tryParse(char) != null;
  }

  @override
  Widget build(BuildContext context) {
    List<Widget> listOfTiles = [];

    for (int index = 0; index < appSettings.selectedTemplates.length; index++) {
      final template = appSettings.selectedTemplates[index];
      List<String> ZametkiList = myList[index];
      ListTile newTile = ListTile(
        title: Text(template.title),
        subtitle: Text("10.06.24 ${ZametkiList[0]}"),
        onTap: () {
          openNoteDetailScreen(context, template.title, ZametkiList);
        },
      );

      listOfTiles.add(newTile);
    }

    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.orangeAccent,
        title: Text("Нотатки"),
      ),
      body: Column(
        children: <Widget>[
          ListTile(
            title: Text("Купити води"),
            subtitle: Text("11.06.24 Купити води"),
            onTap: () {
            },
          ),
          ListTile(
            title: Text("7-00 Зустріч"),
            subtitle: Text("10.06.24 Максим"),
```

```
            onTap: () {
            },
          ),
          ...listOfTiles,
          ListTile(
            title: Text("Закупка"),
            subtitle: Text("10.06.24 Молоко"),
            onTap: () {
            },
          ),
// Додаємо вже згенеровані ListTile
        ],
      ),
    );
  }

  void openNoteDetailScreen(BuildContext context, String noteTitle, List<String> myList) {
    Navigator.of(context).push(
      MaterialPageRoute(
        builder: (context) => NoteDetailScreen(noteTitle: noteTitle, myList: myList),
      ),
    );
  }

}

class NoteDetailScreen extends StatelessWidget {
  final String noteTitle;
  final List<String> myList;

  NoteDetailScreen({required this.noteTitle, required this.myList});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.orangeAccent,
        title: Text(""),
      ),
      body: Padding(
        padding: EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: <Widget>[
            Text(
              noteTitle,
              style: TextStyle(
                fontSize: 24,
                fontWeight: FontWeight.bold,
              ),
            ),
            Expanded(
```

```
            child: ListView.separated(
              itemCount: myList.length,
              separatorBuilder: (context, index) => SizedBox(), // Забираємо роздільник
              itemBuilder: (context, index) {
                return ListTile(
                  title: Text('${index + 1}. ${myList[index]}'), // Додаємо номер перед елементом
                  // Інші властивості елемента списку, якщо потрібно
                );
              },
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

## Файл app_setings.dart

```
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart'; //збереження та завантаження налаштувань

class SettingsTemplate {
  String item;
  String title;
  List<String> list;
  bool selected;

  SettingsTemplate(this.item, this.title, this.list, {this.selected = false});
}

class AppSettings {
  List<SettingsTemplate> templates = [];
  List<SettingsTemplate> selectedTemplates = [];

  static final AppSettings _singleton = AppSettings._internal();

  factory AppSettings() {
    return _singleton;
  }

  AppSettings._internal() {
    loadSettings();
  }

  Future<void> loadSettings() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    final List<String>? templatesData = prefs.getStringList('templates');

    if (templatesData != null) {
      templates = templatesData
          .map((data) {
        final List<String> parts = data.split(',');
        final bool selected = false;
        return SettingsTemplate(parts[0], parts[1], parts.sublist(2), selected: selected);
      })
          .toList();
    }
  }

  Future<void> saveSettings(BuildContext context,String FromWhat) async {
    SharedPreferences prefs = await SharedPreferences.getInstance();

    final List<String> templatesData = templates
        .map((template) =>
    '${template.item},${template.title},${template.list.join(',')}')
        .toList();
```

```
    await prefs.setStringList('templates', templatesData);

    selectedTemplates = templates.where((template) => template.selected).toList();

    // Перевірка на різну довжину списків вибраних шаблонів
    if (FromWhat == "main"){
      if (selectedTemplates.isNotEmpty) {
        final int firstListLength = selectedTemplates[0].list.length;
        final String firstListTitle = selectedTemplates[0].title;

        for (final template in selectedTemplates) {
          if (template.list.length != firstListLength) {
            // Виводимо діалогове вікно із попередженням, використовуючи переданий context
            showDialog(
              context: context,
              builder: (context) {
                return AlertDialog(
                  title: Text('Попередження'),
                  content: Text('Довжина списку ${template.item}"${template.title}" відрізняється $firstListTitle. Виберіть шаблони з однаковою довгою списків щоб уникнути помилок'),
                  actions: [
                    TextButton(
                      onPressed: () {
                        Navigator.of(context).pop();
                      },
                      child: Text('OK'),
                    ),
                  ],
                );
              },
            );
            break;
          }
        }
      }
      else{
        showDialog(
          context: context,
          builder: (context) {
            return AlertDialog(
              title: Text('Попередження'),
              content: Text('Ви не вибрали жодного шаблону. Будуть показані лише фейкові нотатки'),
              actions: [
                TextButton(
                  onPressed: () {
                    Navigator.of(context).pop();
                  },
                  child: Text('OK'),
                ),
              ],
            );
```

```
        },
      );
    };
  }



 }}

class SettingsScreen extends StatefulWidget {
  @override
  _SettingsScreenState createState() => _SettingsScreenState();
}

class _SettingsScreenState extends State<SettingsScreen> {
  final AppSettings appSettings = AppSettings();

  @override
  void initState() {
    super.initState();
    appSettings.loadSettings().then((_) {
      setState(() {}); // Оновіть стан поточного віджету після завантаження установок
    });

    // Завантаження налаштувань під час ініціалізації
  }


  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Налаштування"),
      ),
      body: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            children: [
              for (int i = 0; i < appSettings.templates.length; i++)
                Card(
                  color: appSettings.templates[i].selected ? Colors.green : null,
                  child: ListTile(
                    title: Text(appSettings.templates[i].item),
                    subtitle: Text(appSettings.templates[i].title),
                    trailing: Row(
                      mainAxisSize: MainAxisSize.min,
                      children: [
                        IconButton(
                          icon: Icon(Icons.edit),
                          onPressed: () {
                            editTemplate(i);
                          },
```

```
                    ),
                    IconButton(
                      icon: Icon(Icons.delete),
                      onPressed: () {
                        deleteTemplate(i);
                      },
                    ),
                  ],
                ),
                onTap: () {
                  setState(() {
                    appSettings.templates[i].selected = !appSettings.templates[i].selected;
                  });
                },
              ),
            ),
          ElevatedButton(
            onPressed: () {
              addTemplate();
            },
            child: Text("Додати шаблон"),
          ),
          SizedBox(height: 16),
          ElevatedButton(
            onPressed: () {
              appSettings.saveSettings(context,"main");
              setState(() {
                appSettings.selectedTemplates.forEach((template) {
                  print('Selected Template: ${template.item}, ${template.title}, ${template.list}');
                });

              });
            },
            child: Text("Зберегти"),
          ),
        ],
      ),
    ),
  );
}

void addTemplate() async {
  final updated = await Navigator.of(context).push(
    MaterialPageRoute(
      builder: (context) => TemplateAddScreen(appSettings),
    ),
  );

  if (updated == true) {
    setState(() {
    });
```

```
    }
  }

  void editTemplate(int index) async {
    final itemController = TextEditingController(text: appSettings.templates[index].item);
    final titleController = TextEditingController(text: appSettings.templates[index].title);
    final listController = TextEditingController(text: appSettings.templates[index].list.join('\n'));

    final updated = await Navigator.of(context).push(
      MaterialPageRoute(
        builder: (context) => TemplateEditScreen(appSettings, index, itemController, titleController,
listController),
      ),
    );

    if (updated == true) {
      setState(() {
        appSettings.selectedTemplates.forEach((template) {
          print('Selected Template: ${template.item}, ${template.title}, ${template.list}');
        });
      });
    }
  }

  void deleteTemplate(int index) {
    showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          title: Text("Видалити шаблон"),
          content: Text("Ви впевнені, що хочете видалити ${appSettings.templates[index].title}?"),
          actions: [
            TextButton(
              onPressed: () {
                Navigator.of(context).pop();
              },
              child: Text("Відміна"),
            ),
            TextButton(
              onPressed: () {
                appSettings.templates.removeAt(index);
                appSettings.saveSettings(context,"delete");
                Navigator.of(context).pop();
                setState(() {});
              },
              child: Text("Видалити"),
            ),
          ],
        );
      },
    );
  }
```

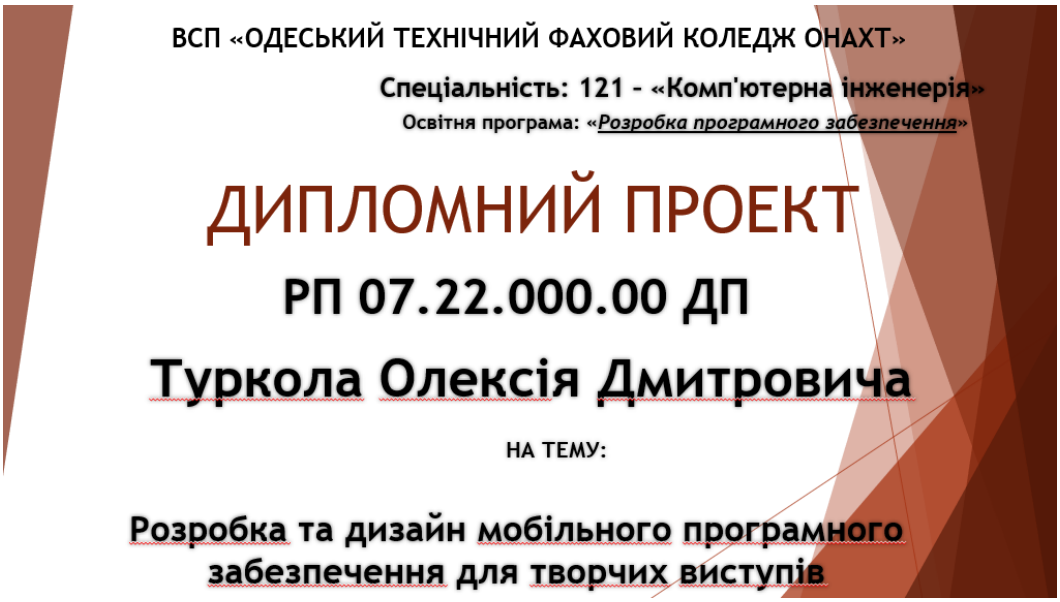
```
}

class TemplateAddScreen extends StatefulWidget {
  final AppSettings appSettings;

  TemplateAddScreen(this.appSettings);

  @override
  _TemplateAddScreenState createState() => _TemplateAddScreenState();
}

class _TemplateAddScreenState extends State<TemplateAddScreen> {
  final TextEditingController itemController = TextEditingController();
  final TextEditingController titleController = TextEditingController();
  final TextEditingController listController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Додати шаблон"),
      ),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            TextField(
              controller: titleController,
              decoration: InputDecoration(labelText: "Назва нотатки"),
            ),//Название
            TextField(
              controller: itemController,
              decoration: InputDecoration(labelText: "Предмет"),
            ),//Предмет
            TextField(
              controller: listController,
              decoration: InputDecoration(
                labelText: "Список (введіть значення, розділяючи символом перенесення рядка)",
              ),
              maxLines: null,
            ),//Список
            SizedBox(height: 16),
            ElevatedButton(
              onPressed: () {
                saveTemplate();
              },
              child: Text("Зберегти"),
            ),//Кнопка сохранить
          ],
        ),
      ),
    );
```

```
  }

  void saveTemplate() {
    final newTemplate = SettingsTemplate(
      itemController.text,
      titleController.text,
      listController.text.split('\n'),
    );
    widget.appSettings.templates.add(newTemplate);
    widget.appSettings.saveSettings(context,"add");
    Navigator.of(context).pop(true);
  }
}

class TemplateEditScreen extends StatefulWidget {
  final AppSettings appSettings;
  final int templateIndex;
  final TextEditingController itemController;
  final TextEditingController titleController;
  final TextEditingController listController;

  TemplateEditScreen(this.appSettings, this.templateIndex, this.itemController, this.titleController,
this.listController);

  @override
  _TemplateEditScreenState createState() => _TemplateEditScreenState();
}

class _TemplateEditScreenState extends State<TemplateEditScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Редагувати шаблон"),
        actions: [
          IconButton(
            icon: Icon(Icons.delete),
            onPressed: () {
              deleteTemplate();
            },
          ),
        ],
      ),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            TextField(
              controller: widget.titleController,
              decoration: InputDecoration(labelText: "Назва нотатки"),
            ),
            TextField(
```

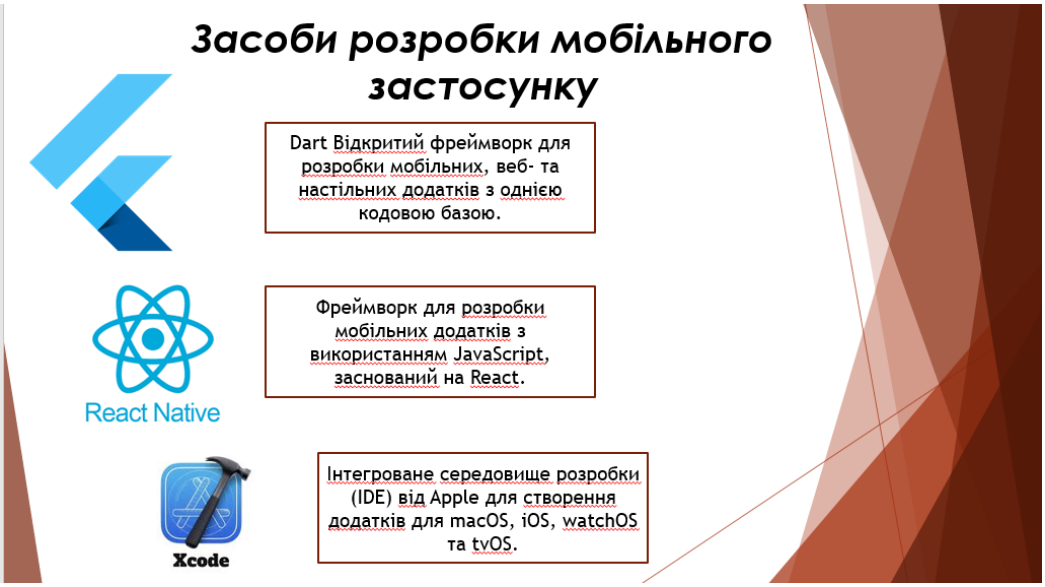
```
            controller: widget.itemController,
            decoration: InputDecoration(labelText: "Предмет"),
          ),
          TextField(
            controller: widget.listController,
            decoration: InputDecoration(
              labelText: "Список (введіть значення, розділяючи символом перенесення рядка)",
            ),
            maxLines: null,
          ),
          SizedBox(height: 16),
          ElevatedButton(
            onPressed: () {
              saveTemplate();
            },
            child: Text("Зберегти"),
          ),
        ],
      ),
    ),
  );
}

void saveTemplate() {
  final updatedTemplate = SettingsTemplate(
    widget.itemController.text,
    widget.titleController.text,
    widget.listController.text.split('\n'),
    selected: widget.appSettings.templates[widget.templateIndex].selected,
  );
  widget.appSettings.templates[widget.templateIndex] = updatedTemplate;
  widget.appSettings.saveSettings(context,"edit");
  Navigator.of(context).pop(true);
}

void deleteTemplate() {
  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: Text("Видалити шаблон"),
        content: Text("Ви впевнені, що хочете видалити цей шаблон?"),
        actions: [
          TextButton(
            onPressed: () {
              Navigator.of(context).pop();
            },
            child: Text("Відміна"),
          ),
          TextButton(
            onPressed: () {
              widget.appSettings.templates.removeAt(widget.templateIndex);
```

```
                widget.appSettings.saveSettings(context,"delete");
                Navigator.of(context).pop();
                Navigator.of(context).pop(true);
              },
              child: Text("Видалити"),
            ),
          ],
        );
      },
    );
  }
}

void main() {
  runApp(MaterialApp(
    home: SettingsScreen(),
  ));
}
```

# ДОДАТОК Б. Слайди мультимедійної презентації



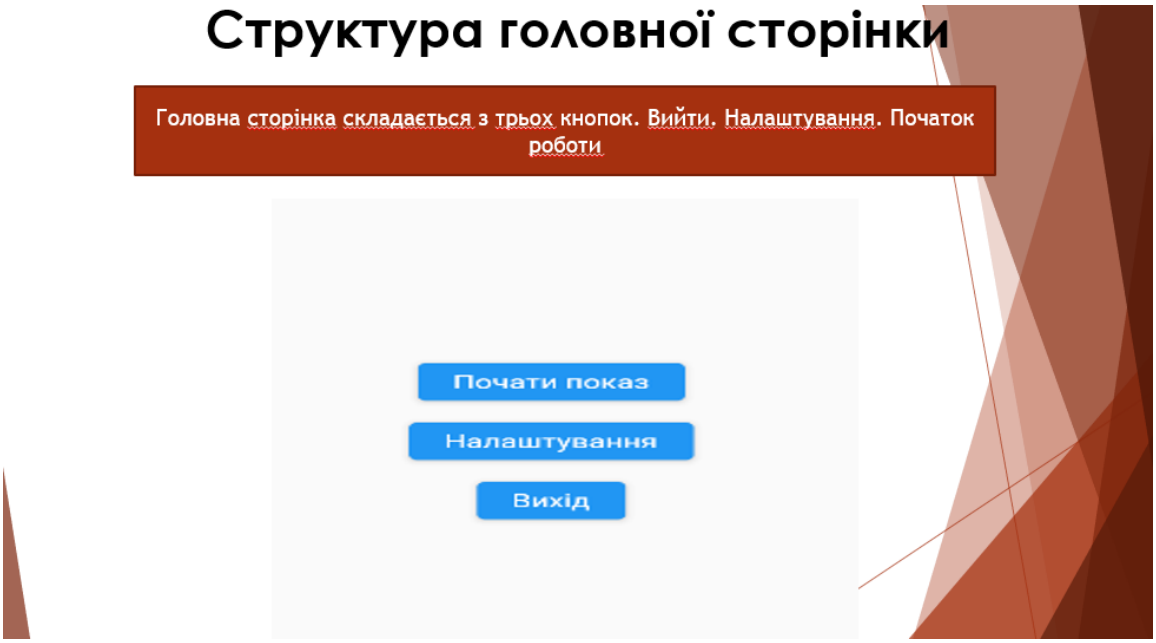

Слайд 1. Вступ



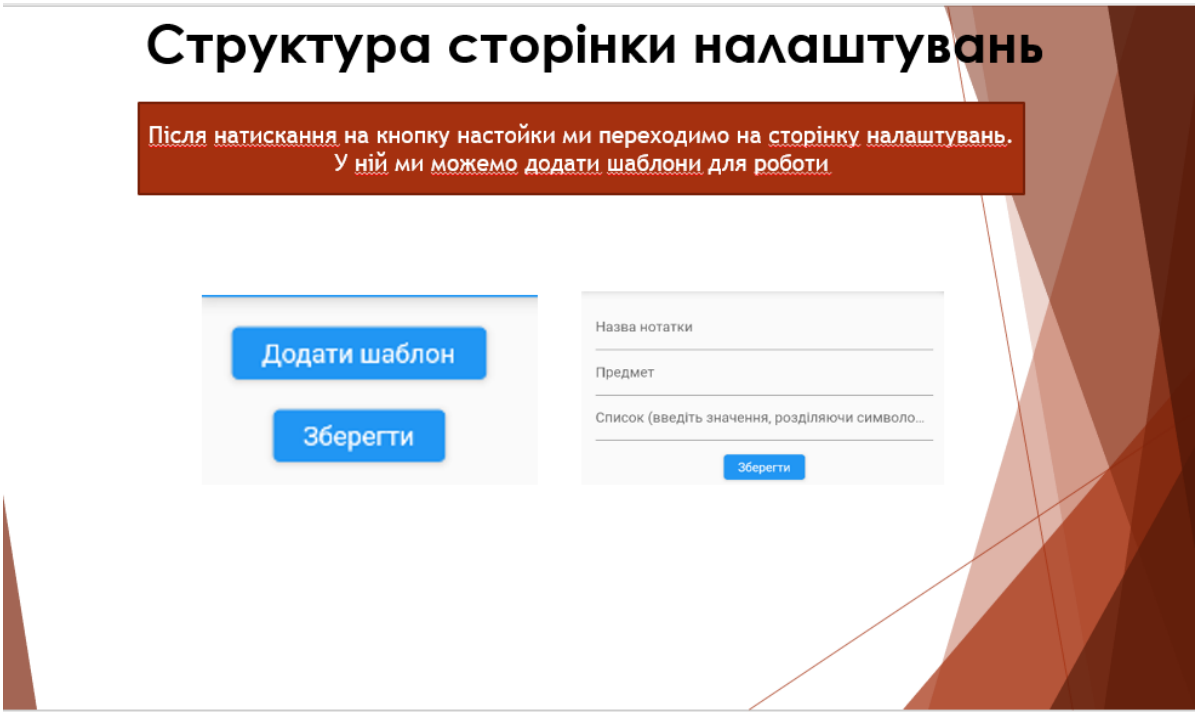

Слайд 2. Засоби розробки мобільного застосунку


Слайд 3. Етапи створення моб.застосунку

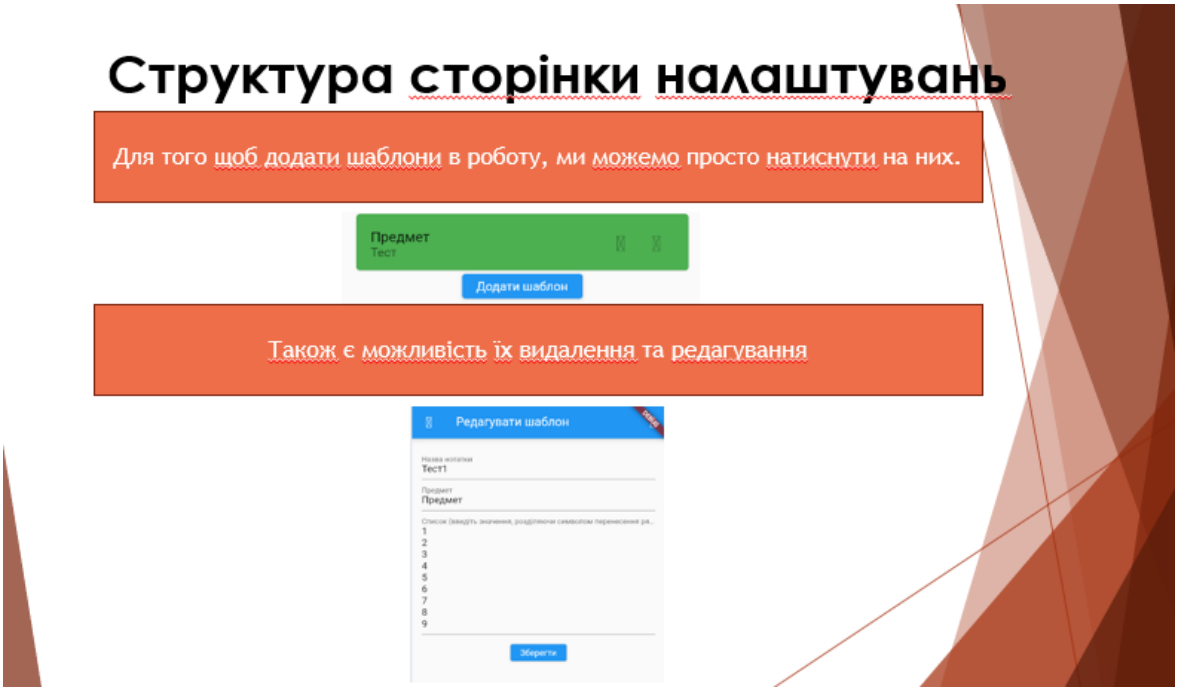

Слайд 4. Структура моб. застосунку




Слайд 5. Структура головної сторінки

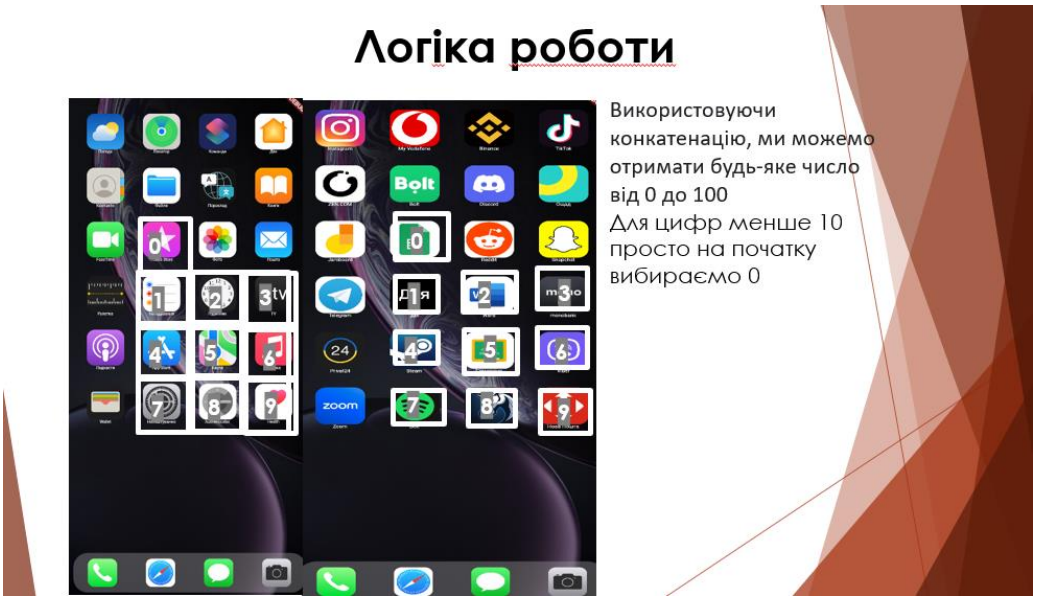



Слайд 6. Структура сторінки налаштувань

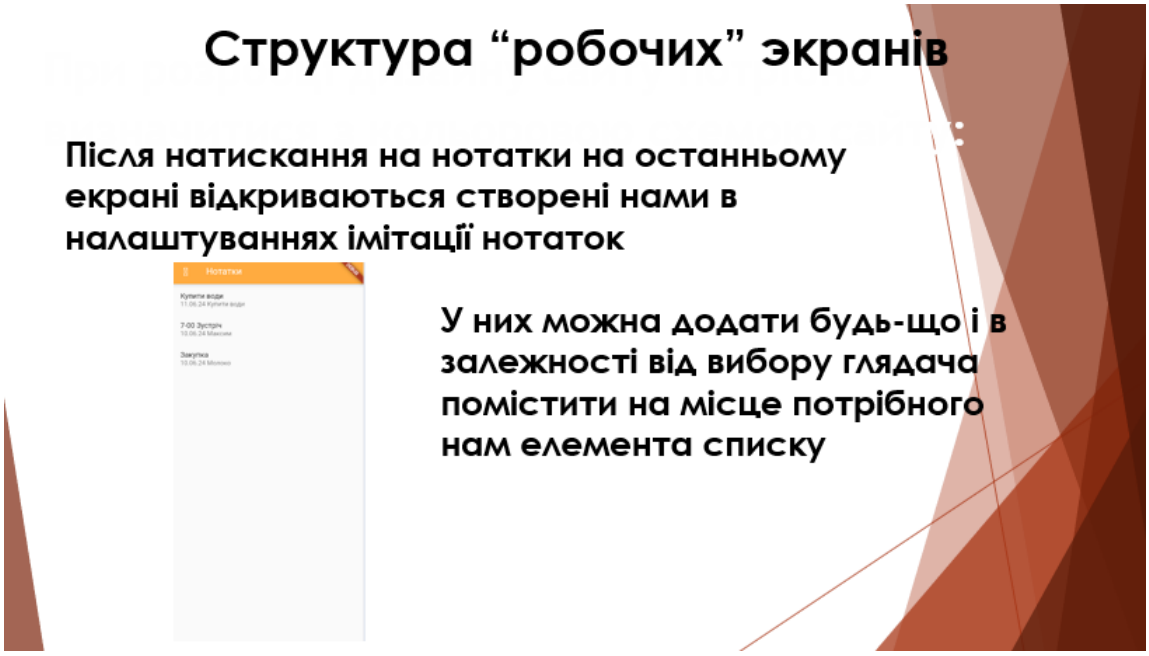



Слайд 7. Структура сторінки налаштувань

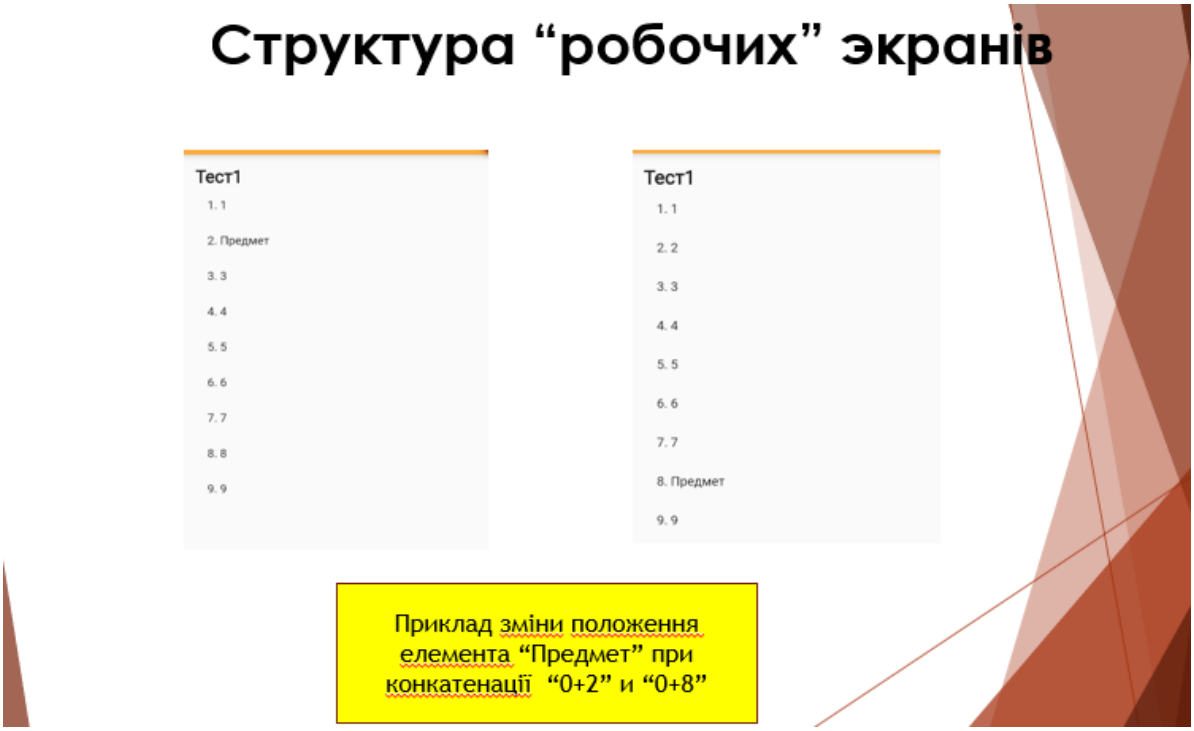

Слайд 8. Структура “робочих” экранів




Слайд 9. Логіка роботи




Слайд 10. Структура “робочих” экранів




Слайд 11. Структура “робочих” екранів


Слайд 12. Дякую за увагу

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

# ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Туркола Олексія Дмитровича

(прізвище, ім'я та по батькові)

Спеціальність: 121 "Інженерія програмного забезпечення"

Освітня програма: «Розробка програмного забезпечення»

Тема дипломного проекту: Розробка та дизайн мобільного програмного забезпечення для творчих виступів

## ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки)

Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка містить 64 сторінки. У пояснювальній записці виконано опис етапів розробки сайту а також технології розробки. Графічна частина складається з 12 слайдів мультимедійної презентації, які також містять пункти, передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини добра, розробку виконано в повному обсязі.

б) самостійність роботи над проектом:

Протягом всього строку дипломного проектування та переддипломної практики здобувач освіти Туркол О.Д. поступово та послідовно виконув всі етапи розробки. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника.

в) теоретична підготовка випускника (випускниці):

Здобувач освіти Туркол О.Д. під час роботи над дипломним проектом вивчала достатню кількість літературних джерел та матеріалів за даною тематикою. Вважаю, що теоретична підготовка дипломника добра і віна готовий до захисту дипломного проекту

г) вміння розв'язувати виробничі та конструкторські питання

Під час дипломного проектування здобувач освіти Туркол О.Д. мав змогу самостійно приймати окремі рішення з реалізації дизайн мобільного програмного забезпечення та показав вміння організовано працювати над поставленим завданням, розробляти логіку та дизайн застосунку за допомогою сучасних комп'ютерних програмних засобів та мов програмування.

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Відмінно

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові керівника дипломного проекту

Закроєв Юрій Михайлович

Місце роботи і посада керівника дипломного проекту

ТОВ «БІГ ВОШ», директор

Підпис

«10» 06 2024 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

# РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

**Туркола Олексія Дмитровича**

(прізвище, ім'я та по батькові)

Спеціальність **121 Інженерія програмного забеспечення**

Освітня програма **« Розробка програмного забезпечення»**

Керівник дипломного проекту (роботи) Закроєв Ю.М

(прізвище, ім'я та по батькові)

Тема дипломного проекту (роботи) **Розробка та дизайн мобільного програмного забезпечення для творчих виступів**

Обсяг розрахунково-пояснювальної записки 96 сторінок

Обсяг графічної (презентаційної) частини 12 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлений на рецензію робота відповідає затверджений темі та виконаний відповідно технічному завданню. Дипломний проект є актуальним з погляду останніх рекомендацій для розробки та дизайну мобільного програмного забезпечення для творчих виступів

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Пояснювальна записка складається з технологічної частини, розробки структури програми та засобів програмування програми для розробки та дизайну мобільного програмного забезпечення для творчих виступів, економічної частини, розділу охорони праці та додатку. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано. Розділ охорони праці містить загальну інформацію та вимоги до техніки безпеки оператора КТ. Економічна частина проекту містить розрахунок затрат на виконання програми

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи)

Графічна частина складається з 12 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять структуру проекту, скріншоти роботи програми, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання графічної частини проекту та пояснювальної записки висока, розробку виконано у повному обсязі

г) перелік позитивних якостей дипломного проекту (роботи)

У роботі обґрунтовано розробку мобільного програмного забезпечення для творчих виступів

д) основні недоліки дипломного проекту (роботи)

1. Етапи розробки недостатньо структуровано, немає блок-схем алгоритмів та діаграм, пов'язаних з реалізацією мобільного програмного забезпечення.
2. Наявні помилки оформлення пояснювальної записки.

Оцінка розрахункової частини Добре

Оцінка графічної частини Добре

Загальна оцінка Добре

Прізвище, ім'я, по батькові рецензента к.т.н. Селіванова Алла Віталіївна

Місце роботи і посада рецензента Одеський національний технологічний університет, декан факультету комп'ютерної інженерії, програмування та кіберзахисту

Підпис:

«19» 06 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григоріївна Краснокутська

ID перевірки:
1016370717

Дата перевірки:
18.06.2024 08:37:06 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
18.06.2024 08:41:56 EEST

ID користувача:
100011688

Назва документа: 4РП-07 Туркол

Кількість сторінок: 62 Кількість слів: 10032 Кількість символів: 81873 Розмір файлу: 1.72 MB ID файлу: 1016177823

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

## 6.94%
## Схожість

Найбільша схожість: 2.29% з Інтернет-джерелом (https://card-file.ontu.edu.ua/server/api/core/bitstreams/34a6756b-592.

6.94% Джерела з Інтернету | 580 ..... Сторінка 64

Не знайдено джерел з Бібліотеки

## 0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

## 0%
## Вилучень

Немає вилучених джерел

## Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | 50

Підозріле форматування | 14 сторінок

**ДОЗВІЛ**
**НА РОЗМІЩЕННЯ**
**ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ**
**В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»**

Ми, що нижче підписалися,

***Туркол Олексій Дмитрович***
здобувач освіти гр. 4РП-07, та

***Закроєв Юрій Михайлович***
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи фахового молодшого бакалавра на тему:

***«Розробка та дизайн мобільного програмного забезпечення для творчих виступів» (автор роботи – Туркол О.Д., керівник роботи – Закроєв Ю.М.)***

виконаного у ТОВ «БІГ ВОШ» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи, і даємо згоду на обробку персональних даних.

Виконавець _____ / Туркол О.Д./

Керівник _____ / Закроєв Ю.М./

«10» 06 2024 р.