МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ
ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач випускової кафедри

_____Аліна САВЧЕНКО

«_____»_____2021 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ»

Тема: **«Реалізація функціоналу мобільного додатку MEDiary на платформі iOS»**

Виконавець: Каспаревич Альона Олександрівна

Керівник: к.т.н., доц. Кірхар Наталія Володимирівна

Нормоконтролер: к.т.н., доц. Боровик Володимир Миколайович

КИЇВ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *кібербезпеки, комп'ютерної та програмної інженерії*

Кафедра *Прикладної інформатики*

Спеціальність *122 «Комп'ютерні науки та інформаційні технології»*

Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ:

Завідувач кафедри ПІ

_____ Аліна САВЧЕНКО
(підпис)

«_____»_____2021 р.

ЗАВДАННЯ

на виконання дипломної роботи

Каспаревич Альони Олександрівни

(ПІБ випускника)

1. Тема роботи «Реалізація функціоналу мобільного додатку MEDiary на платформі iOS » затверджена наказом ректора від «12» жовтня 2021 р. №

2. Термін виконання роботи: з 12.10.2021р. по 14.12.2021р.

3. Вихідні дані до роботи: загальний огляд платформ, мов програмування та видів мобільних додатків, ознайомлення з програмним забезпеченням, створення контенту та реалізація мобільного додатку.

4. Зміст пояснювальної записки:

РОЗДІЛ 1. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

РОЗДІЛ 2. АНАЛІЗ ТА ПІДГОТОВКА НАПОВНЕННЯ ДОДАТКУ

РОЗДІЛ 3. РОЗРОБКА ЗАСТОСУНКУ

5. Перелік обов’язкового ілюстративного матеріалу: інформативні пояснювальні рисунки, таблиці, презентація в *MS PowerPoint*.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Ознайомлення з постановкою задачі та вивчення літератури Написання 1 розділу, представлення керівнику	12.10.21 – 12.10.21	
2.	Ознайомлення з програмним забезпеченням та початок роботи. Написання 2 розділу, представлення керівнику	12.10.21 – 15.10.21	
3.	Розробка додатку. Написання 3 розділу, представлення керівнику	15.10.21 – 01.12.21	
5.	Загальне редагування та друк пояснювальної записки.	02.12.21 – 14.12.21	
6.	Проходження нормоконтролю, перепліт пояснювальної записки.	14.12.21 – 15.12.21	
7.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	15.12.21 – 16.12.21	

7. Дата видачі завдання 10.10.2021

Керівник дипломної роботи _____ Кірхар Н.В.
(підпис керівника)

Завдання прийняв до виконання _____ Каспаревич А.О.
(підпис випускника)

РЕФЕРАТ

Дипломна робота складається з: вступу, чотирьох розділів, висновків та списку використаної літератури. Обсяг роботи складає 95 сторінок. Список використаної літератури містить 36 джерел.

Метою роботи є проектування та розробка додатку мовою програмування Swift.

В дипломній роботі було розглянуто сферу мобільних діяльності додатків в категорії “Освіта”, основні мови програмування, платформи, інтеграції бази данних.

Підсумком роботи є розроблений мобільний додаток в сфері “Освіта”, підключена база даних.

Ключові слова: РОЗРОБКА, IOS, SWIFT, ОСВІТА, БАЗА ДАНИХ, МОБІЛЬНИЙ ДОДАТОК.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

IOS - це власницька мобільна операційна система від Apple. Розроблена спочатку для **iPhone.**

Xcode — інтегроване середовище розробки (IDE) виробництва Apple..

Grid або сітка - являє собою набір горизонтальних і вертикальних ліній, що перетинається, - один набір визначає стовпці, а інший рядки. Елементи можуть бути поміщені в сітку, відповідно до рядків і стовпців.

SDK (Software Development Kit) - набір інструментів розробки програмного забезпечення в одному інсталяційному пакеті. Вони полегшують створення програм, маючи компілятор, налагоджувач і, можливо, програмну структуру. Зазвичай вони специфічні для комбінації апаратної платформи та операційної системи. Для створення програм із розширеними функціями, такими як реклама,push-сповіщення, тощо.

IQ (Intelligence Quotient) - коефіцієнт інтелекту це загальна оцінка, отримана з набору стандартизованих тестів або субтестів, призначених для оцінки інтелекту людини. Абревіатура «IQ» була придумана психологом Вільямом Стерном для німецького терміну Intelligenzquotient, його терміну для методу оцінки тестів на інтелект в Університеті Бреслау, який він відстоював у книзі 1912 року.

SQL (Structured Query Language) — це стандартизована мова програмування, яка використовується для керування реляційними базами даних і виконання різних операцій над даними в них.

ВСТУП

Смартфон – це стільниковий телефон із можливостями комп'ютера та іншими функціями, які спочатку не були пов'язані з телефонами, такими як операційна система, перегляд веб-сторінок та можливість запуску програмних додатків.

Смартфони можуть використовуватися приватними особами як у споживчому, так і в діловому контексті, і зараз вони майже невід'ємні елементи повсякденного сучасного життя.

Багато споживачів використовують свої смартфони для спілкування з друзями, родиною та брендами в соціальних мережах. Соціальних медіа, такі як Facebook, Instagram, Twitter і LinkedIn, створюють свої мобільні додатки, які користувач може завантажити з магазину програм свого телефону. Ці програми дозволяють користувачам смартфонів публікувати особисті оновлення та фотографії.

Іншим поширеним використанням смартфонів є відстеження здоров'я та самопочуття. Стають більш поширеним трекінги харчування, активності, онлайн тренування, марафони в смартфоні. Окрім здорового тіла люди також мають можливість розвивати своє емоційне самопочуття завдяки додаткам з медитації, та тренуватися у розвитку когнітивних функцій.

Проект, який розробляється в рамках дипломної роботи, створюється для тренування мозику. Програмне рішення буде представлено у вигляді мобільного додатка для звичайних користувачів.

Програмний продукт надає можливості дізнатись свій рівень IQ, та мати можливість прокачувати свої знання граючи ігри. Користувач повинен лише мати смартфон та доступ в мережу Інтернет.

Метою дипломної роботи є розробка застосунку в галузі Освіта.

Для досягнення мети необхідно виконати наступні задачі:

- ознайомитися з загальним представлення про категорію мобільного додатку;
- обрати середовище для розробки;

- зробити наповнення для додатку;
- розробити застосунок;
- протестувати;
- виправити помилки.

Даний проект орієнтований на аудиторію, що може використовувати її як з науковими, так і з пізнавальними цілями, а саме: можна розвивати свою пам’ять, уважність, логіку, математику, дізнатись рівень IQ в ігровій формі, прокачувати свій мозок та розвиватись.

РОЗДІЛ 1
ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

1.1. Обґрунтування вибору ОС мобільного додатка

Розглянуто дві найбільш популярні мобільні ОС на сьогоднішній день та проаналізовано, яку буде доцільно обрати.

1.1.1. Огляд ОС iOS

iOS – мобільна операційна система, яку створила та розробила компанія Apple. Вона призначена використовуватись тільки на апаратному забезпеченні компанії Apple. Це відмінна характеристика від мобільних ОС Android та Windows Phone. iOS – це друга за популярністю ОС у світі після Android (29% від кількості всіх проданих смартфонів за 2021 рік).

Для розповсюдження додатків компанія Apple використовує інтернет магазин додатків App Store. За офіційною статистикою станом на кінець 2021 року в App Store розміщено понад 8,1 мільярдів додатків. В порівнянні з початком роботи App Store у 2008 році, там було розміщено 500 програмних додатків.

Компанія Apple створила IDE Xcode для розробки під ОС iOS. IDE є зручною через редактор вихідного коду, де можна легко змінювати програмний код, швидко компілювати, запускати виконувані файли та тестові набори даних. Також Xcode надав можливість спільно працювати над програмним кодом у команді. Apple зробила цей процес безпечним та простим, використовуючи хмарні технології та унікальний ключ SSH.

Ця можливість була надана за допомогою наступних продуктів від систем контролю версій:

- GitHub;
- Bitbucket;
- GitLab.com.

Також Xcode містить потужні засоби відлагодження, які значно економлять час розробника при пошуку програмної помилки. Якісний вбудований симулятор дозволяє легко тестувати розроблені мобільні додатки, а зокрема інтерфейси. У розробника немає потреби володіти фізично всіма пристроями компанії Apple для успішного тестування. Всі необхідні симулятори моделей пристроїв та версій iOS містяться в IDE.

Перевагою мобільної ОС iOS є централізованість платформи. Всі пристрої одного користувача з'єднані одним акаунтом AppleID. Вся інформація буде синхронізуватись в одному місці, паролі зберігаються для всіх пристроїв і користувач відчуває себе в єдиній екосистемі, що безперечно є дуже зручним рішенням. Також, це полегшує роботу розробникам системи. Смартфони на базі iOS будуть підтримуватись компанією Apple, тож користувачі iOS-смартфонів можуть не турбуватись про помилки та неточності в роботі з системою, бо їх виправлять в наступній версії оновлень з великою ймовірністю.

Перевагою ОС iOS є те, що її користувачі є більш платоспроможними, тому що Apple завжди пропагувала якість та безпеку своїх програмних продуктів. Неможливо завантажити мобільний додаток на iPhone, без використання App Store. Всі сервіси iOS дуже добре захищені, бо компанія приділяє цьому питанню багато уваги. Це означає кращу перспективу мобільного додатка для монетизації.

Мови програмування для мобільної ОС iOS: Objective-C та Swift. IDE: Xcode.

1.1.2. Огляд ОС Android

Android – мобільна операційна система з відкритим кодом, яку створила та розробила компанія Google. Основою для її створення було ядро Linux. Android – це найбільш популярна ОС у світі, яка використовується на смартфонах (починаючи з 2011 року). З цієї мобільної ОС частіше всього виходять в мережу Інтернет.

Для розповсюдження додатків для користувачів компанія Google використовує власний офіційний інтернет-магазин додатків Google Play. Також Google Play надає користувачам похідні сервіси для пошуку та

завантаження книг, музики та фільмів. За офіційною статистикою станом на кінець 2017 року в Google Play розміщено приблизно 3,5 мільйонів додатків. А за статистикою 2021 року програмні додатки з Google Play було завантажено 182 разів.

Для розробки додатків для ОС Android можна використовувати різні середовища розробки. Найбільш популярними рішеннями є Eclipse від компанії IBM та Android Studio від компанії Google. Перевагами Eclipse над іншими середовищами розробки є безкоштовне та безпечне використання. Завдяки таким сильним перевагам, компанія IBM заробила прихильність

більшості Java розробників до сьогоднішнього дня. Eclipse містить велику кількість плагінів для виконання майже будь-якої задачі. Значним недоліком середовища розробки Eclipse є низька продуктивність через використання старої віртуальної машини JVM, яка була розроблена на початку 2000-х років разом з середовищем розробки.

Android Studio в порівнянні з Eclipse Android Studio має ряд переваг:

- вбудовані SDK, які автоматично завантажують необхідні пакети та інструменти для розробки;

- підтримка декількох мов програмування (Java, C, C++); – підтримка компанії Google. Тобто через Android Studio компанія Google реалізовує своє бачення розвитку Android-додатків.

Але значним мінусом цієї IDE є повільна робота вбудованого емулятора Android.

Для його коректної роботи розробник системи повинен мати якісне та потужне апаратне забезпечення, яке має високу вартість.

В порівнянні цих двох середовищ розробки з Xcode, то останній має всі переваги, що й IDE для Android-додатків, і не має їхніх недоліків.

Перевагою мобільної ОС Android є те, що користувачу доступно велика кількість пристроїв від різних виробників, з варіацією параметрів та ціновим діапазоном. Але для розробників це значний недолік. Через встановлення мобільної ОС Android на пристроях різних компаній, ОС втрачає свою якість, бо неможливо створити настільки універсальну ОС.

Для розв’язання цієї проблеми виробники використовують власні оболонки для коректної роботи пристрою з ПЗ від Google [20]. Найбільш відомі – Xiaomi, Meizu.

Через наявність великої кількості виробників Android-пристроїв, виникає проблема поєднання ПЗ та АЗ, так званого «заліза». Всі пристрої мають різну оптимізацію та витрачають ресурси по-різному. Навіть при більш продуктивному АЗ, пристрій на базі Android може працювати повільніше через недостатню оптимізацію ОС та поєднанню її з АЗ. Цієї проблеми немає в iOS-пристроїв через централізованість платформи. Тому доцільним є використання пристроїв на базі так званого «чистого» Android, розробкою яких займаються компанія Google. Наприклад, лінія смартфонів та планшетів Nexus та Pixel.

Через ці особливості Android страждає інтерфейс ОС. Пристрої с ОС

Android мають широку кількість розмірів екранів, тому розробники використовують XML файли для створення графічних інтерфейсів. Це означає, що вміст мобільного додатку відображається єдиним способом на екранах.

1.2. Обґрунтування вибору мови програмування

1.2.1. Мова програмування Swift

Swift – об’єктно-орієнтовна мова програмування, яку створила компанія Apple для розробки програмних додатків для ОС iOS та macOS. Основою для створення Swift стала мова програмування Objective-C. Swift замислювався як сучасна версія старої мови програмування Objective-C з простим синтаксисом, щоб зробити роботу розробника легше. Платформа: iOS, macOS та інші платформи від Apple.

Основна IDE: Xcode.

Swift - це фантастичний спосіб писати програми для телефонів, для десктопних комп'ютерів, серверів, та й чогось ще, що запускає та працює за допомогою коду. Swift - безпечна, швидка та інтерактивна мова програмування. Swift увібрав у себе найкращі ідеї сучасних мов із мудрістю інженерної культури Apple. Компілятор оптимізовано для продуктивності, а мова оптимізована для розробки без компромісів з одного чи іншого боку.

Swift доброзичливий по відношенню до новачків у програмуванні. Це перша мова програмування промислової якості, яка також зрозуміла і захоплююча, як скриптова мова. Написання коду в пісочниці дозволяє експериментувати з кодом Swift та бачити результат миттєво, без необхідності компілювати та запускати програму.

Переваги мови програмування Swift:

- інтуїтивна робота програм – програми починають працювати до того, як буде написаний весь програмний код;
- використання швидкого компілятора – в порівнянні з іншими інтерпретаційними мовами PHP і JS. Зберігання програм у вигляді набору виконуваних файлів;
- гарантована безпека написання програмного коду від розробників Apple;
- існування допоміжних інформативних ресурсів – товариство програмістів Swift та багато форумів;
- якісна навчальна література – офіційна документація від Apple, яка постійно оновлюється, безкоштовний посібник в iBooks, та популярна книга В. Усова «Swift. Основи розробки додатків»;
- зручна IDE Xcode – функція автоматичного доповнення (те, чого не вистачає популярному IDE Notepad.exe).

Недоліки мови програмування Swift:

- менш динамічний, ніж Objective-C;
- важче робота з C-кодом;
- мала кількість проектів написаних на Swift.

1.2.2. Мова програмування Objective-C

Objective-C – об’єктно-орієнтовна мова програмування, яку створила компанія Apple на початку 1980-х років для розробки програмних додатків для ОС iOS та macOS. Вона розроблена у вигляді набору розширень стандартної мови програмування C. Більшість програмних додатків у App

Store написані саме мовою програмування Objective-C. До випуску компанією Apple у 2014 році нової мови програмування Swift Objective-C була єдиною мовою програмування для створення iOS додатків.

Платформа: iOS, macOS та інші платформи від Apple.

Основна IDE: Xcode.

При створенні цієї мови програмування була вирішена проблема повторного використання програмного коду. Таке рішення підвищити продуктивність та якість написаного програмного коду. Objective-C слабо типізована мова програмування. Objective-C є розширенням мови програмування C, тому Objective-C може використовувати всі існуючі інструменти, бібліотеки для мови C. Objective-C підтримує перевантаження оператора та забороняє множинне успадкування на відміну від C++.

Компанія Apple випустила Objective-C 2.0 у 2006 році, яка включала в себе функції актуальні для нового покоління розробки програмних додатків. Вони зробили синтаксис мови більш зрозумілим для зниження порогу входу для вивчення мови, покращили продуктивність, але оновлений garbage collector не був новим рішенням для поставленої проблеми сміття в програмному коді. *Переваги мови програмування Objective-C:*

- більшість існуючого ПЗ для продукції компанії Apple написане на Objective-C (понад 1,2 мільйона програмних додатків);
- велика кількість технічної літератури та офіційна документація; – динамічна типізація мови програмування;
- сумісництво з C-подібними мовами, що дозволяє використовувати компілятори та інструменти мови програмування C для розробки програмних додатків на Objective-C.

Недоліки мови програмування Objective-C:

- низька читабельність програмного коду;
- низька продуктивність мови через динамічну типізацію та повільна швидкість роботи розробників;
- різне розширення для файлів модулів та файлів заголовків, що є незручним для створення нових файлів;
- нерегулярні оновлення мови програмування;
- менша популярність в перспективі через Swift.

1.2.3. Мова програмування Java

Java – об’єктно-орієнтовна мова програмування, яку створила компанія Sun Microsystems (але пізніше її викупила Oracle) у 1995 році. За офіційною статистикою 2020 року, Java стала однією з найпопулярніших мов програмування згідно проаналізованому програмному коду на GitHub. Її використовують 9 мільйонів розробників.

Концепція мови програмування Java – мати якомога менше залежностей реалізації. Тобто програмний код, написаний та скомпільований на Java, може працювати без перекомпіляції на всіх платформах, в яких підтримується Java. Машинний код, в який компілюється програмний код на Java буде працювати на будь-якій JVM, не зважаючи на базову архітектуру комп’ютера. Java була створена на основі C та C++, про що стверджує схожий синтаксис, але Java є більш високорівневою мовою програмування .

Платформа: Android, Android Wear.

Основні IDE: Android Studio, Eclipse.

Java використовується як основа для Android SDK, а сама мобільна ОС Android використовує ядро Linux, який написаний на C. Машинний код

Android SDK не сумісна з машинним кодом Java, тому Android SDK мусить працювати на власній віртуальній машині. Вона спеціально оптимізована під смартфони та планшети, де використовується мобільна ОС Android.

Переваги мови програмування Java:

- використання ООП та популяризація його серед широких мас розробників; – простий синтаксис через вплив мов програмування С та С++;
- наявність функцій, що запобігають критичним помилкам, наприклад Security Manager;
- незалежність від платформи використання завдяки JVM; – наявність багатопоточності;
- наявність АММ, що автоматично очищує пам’ять, якщо розробник не зробив цю дію вручну;
- підтримка великої кількості БД;
- наявність великої кількості навчальних матеріалів та офіційної технічної документації, що постійно оновлюється та доповнюється;
- оновлення старих версій мови від компанії Oracle.

Недоліки мови програмування Java:

- низька продуктивність через використання JVM та некоректну роботу АММ;
- відсутність зручних способів для створення графічного інтерфейсу користувача;
- платне використання Java SE 8 в комерційних цілях.

1.2.4. Мова програмування Kotlin

Kotlin – мова програмування зі статичною типізацією, яку створила компанія Jet Brains у 2011 році. Kotlin напряму взаємодіє з JVM. Концепція мови програмування Kotlin – бути більш лаконічною та безпечнішою, ніж Java та не такою складною як Scala. Компіляція програмного коду мовою програмування Kotlin є швидшою в порівнянні із Scala. 7 травня 2019 року відомий американський IT-ресурс Tech Crunch визнав мову програмування Kotlin кращою мовою Google для розробки програмних додатків для мобільної ОС Android. Платформа: Android.

Основна IDE: Android Studio.

Використання мови програмування Kotlin для розробки Android додатків виконується на основі Java 7 зі своїми надбудовами, такими як

обробка використання нульового вказівника та використання функцій розширення. В сумісництві з мовою програмування Java та IDE Android Studio мова програмування Kotlin надає поліпшену читабельність програмного коду, розширені можливості Android SDK і, як наслідок, прискорену розробку ПЗ.

Переваги мови програмування Kotlin:

- лаконічність мови. Kotlin потребує менше коду, ніж Java для однакового класу;
- зручні властивості класів, запобігання помилки ділення на нуль, функції розширення, зручне форматування рядків, ;
- підтримка мови програмування від Google, про що свідчать декілька стабільних версій;

- повна сумісність з Java. Класи, написані на Kotlin, можна використовувати в Java і навпаки в межах одного проекту;
- інтерфейси можуть реалізовувати методи.

Недоліки мови програмування Kotlin:

- невелика популярність мови програмування через використання Java;
- слабке просування для потенційних користувачів від компанії розробника Jet Brains.

Таким чином, проаналізувавши інформацію про мови програмування для мобільних ОС iOS та Android було виділено наступні переваги мови програмування Swift над Objective-C:

- якісна та регулярна підтримка від компанії Apple. З початку випуску мови (з 2014 року) Apple було випущено 4 версії цієї мови. Це свідчить про те, що Swift активно розвивається для пропагується компанією Apple для використання. Сьогодні мова програмування Objective-C вже застаріла і майже не підтримується;
- швидка та зручна робота з програмним кодом Swift. Його набагато простіше читати розробнику системи, а також модифікувати та створювати. В Objective-C необхідно створити окремий файл для моделі та для хедера та логічно їх поєднати;
- офіційна технічна література від Apple та активне оточення розробників. Враховуючи переваги та недоліки кожної мови програмування в цій роботі буде доцільно використовувати мову програмування Swift.

1.3. Обґрунтування вибору системи керування базами даних

1.3.1. Firebase

Firebase – платформа для розробки програмних додатків, яку розробила компанія Firebase. У 2014 році її викупила компанія Google. На даний момент Firebase має 18 програмних продуктів, які використовують 1,5 мільйони програмних додатків.

Найвідоміші програмні додатки – аналітика Firebase, Firebase Cloud Messaging, Firebase Auth, Firebase Realtime, хостинг Firebase. Також, для мобільних ОС Android та iOS було створено інфраструктуру для тестування програмних додатків. Всі необхідні матеріали про результати тестування доступні в зручному вигляді. Є можливість автоматичного тестування при умові, що розробник ПЗ не передбачив тестовий код для розробленої системи.

Firebase Realtime – це сервіс, який надає БД, в основі якої лежить хмарна технологія. Дані зберігаються у форматі JSON і синхронізуються в реальному часі. Використовуючи SDK Firebase для розробки під мобільну ОС, то розробники отримують автоматичне оновлення з новими даними. Дані залишаються доступними при переході в автономний режим користування.

Основні функції Firebase:

- синхронізація даних в режимі реального часу;
- швидкий доступ до документів і колекцій, що зберігаються в БД; – SDK для iOS, Android і Web з офлайн-доступом до даних; – автоматична реплікація даних з сильною узгодженістю; – серверні SDK для Node, Python, Go і Java;
- простота у використанні, що є пріоритетом для всіх продуктів Firebase; – синхронізація даних між пристроями в режимі реального часу, що

дозволяє створювати додатки з автоматичною синхронізацією даних;

- використання колекцій і документів для структурування та обробки даних.

При цьому підході продуктивність запитів не залежить від обсягу даних, і результат роботи з базою з 100 і 100 мільйонів документів буде однаково ефективний; – забезпечення без серверної розробки. Клієнтський SDK Cloud Firestore сам піклується про забезпечення працездатності аутентифікації користувачів. Бекенд надає набір утиліт, які контролюють доступ до даних. Повна інтеграція з платформою Firebase.

1.3.2. MySQL

MySQL – реляційна система управління реляційними базами даних, є СУБД з відкритим вихідним кодом. Була створена компанією «TcX» у 1995 році, нині нею володіє компанія Oracle. Метою створення MySQL було підвищити швидкість обробки великих об’ємів даних. Вона підтримується багатьма відомими мовами програмування. MySQL – це найбільш поширена серверна СУБД. MySQL часто використовується для веб-додатків, її використовують такі відомі сервіси як Facebook, Twitter, Flickr, YouTube.

Існує два видання для цієї СУБД: MySQL Server, який має відкритий вихідний код та фірмовий Enterprise Server. Останній відрізняється певною кількістю власних розширень, які встановлюються як серверні плагіни, але обидва видання будуються на однаковій кодовій базі.

Основні функції MySQL:

- крос-платформне використання та підтримка;
- відповідність стандартам SQL;
- використання тригерів;
- використання курсорів для полегшення обробки запитів вилучення,

додавання і видалення записів бази даних;

- наявність статистики про роботу сервера та продуктивність запитів; – кількість рядків у таблицях до 50 мільйонів записів;
- дотримання властивостей ACID;
- кешування запитів;
- вкладені запити;
- вбудована підтримка реплікації;
- синхронна, асинхронна та напівсинхронна реплікація;
- підтримка Unicode;
- використання протоколу SSL для забезпечення додаткового захисту СКБД;
- повнотекстовий пошук та індексація;
- вбудована бібліотека баз даних;
- рідні системи зберігання: InnoDB, MyISAM, Merge, Memory, Federated, Archive, CSV, Blackhole, NDB Cluster.

Перевагами MySQL є простота в роботі, здатність до масштабування, швидкість роботи, надійність, якісне наслідування функціоналу від SQL, безкоштовне користування для некомерційних проектів.

Недоліками MySQL є менша надійність транзакцій через те, що при створенні СУБД їх не було розроблено, більш повільний процес розробки в порівнянні з сучасними БД, обмеження функціоналу, який не вистачає для розробки сучасних програмних додатків, невідповідність стандартам SQL в певних принципових питаннях. MySQL не може виконувати багато процесів одночасно, що ставить її ефективність під сумнів.

Таким чином, проаналізувавши інформацію про сервіс Firebase Realtime та СУБД MySQL та порівнявши їх можливості, було виділено наступні переваги Firebase над MySQL:

- швидкодія роботи з даними. Розробник та користувач може дуже швидко доступитись до даних;
- простота в роботі з системою. Не потрібно створювати велику кількість файлів, треба лише зареєструвати програмний додаток на офіційному сайті Firebase;
- активна підтримка від Google та безкоштовний зворотній зв’язок від спеціалістів по розробці.

Враховуючи переваги та недоліки кожного способу зберігання даних в цій роботі буде доцільно використовувати сервіс для зберігання даних Firebase Realtime.

1.4. Шаблон проектування MVC

Model-View-Controller (MVC) – архітектурний шаблон проектування, відноситься до глобального шаблону проектування в проекті, який беруть за основу при проектуванні та розробці ПЗ. Суть цього шаблону проектування в поділі розроблюваної системи на три частини, які зв’язані між собою (див. рис. 2). Це модель даних, вигляд системи та система керування програмного додатка. Дуже часто цей шаблон проектування використовується для GUI у веб-додатках. Багато мов програмування, наприклад, JavaScript, Python, Ruby, PHP, Java, C# містять шаблони MVC для продуктивного створення програмних додатків.

Мета шаблону – створити гнучкий програмний додаток, в якому не буде виникати проблем при масштабуванні, зміні певного функціонала та дизайну.

Також, другорядною метою є використання програмного коду декілька разів для більшої продуктивності програмного додатка. А також відокремити логічне представлення програмного додатка від того, що бачить

користувач. При використанні шаблону MVC великі програмні додатки стають більш зрозумілими для самих розробників та для користувачів як наслідок (рис.1.1.).

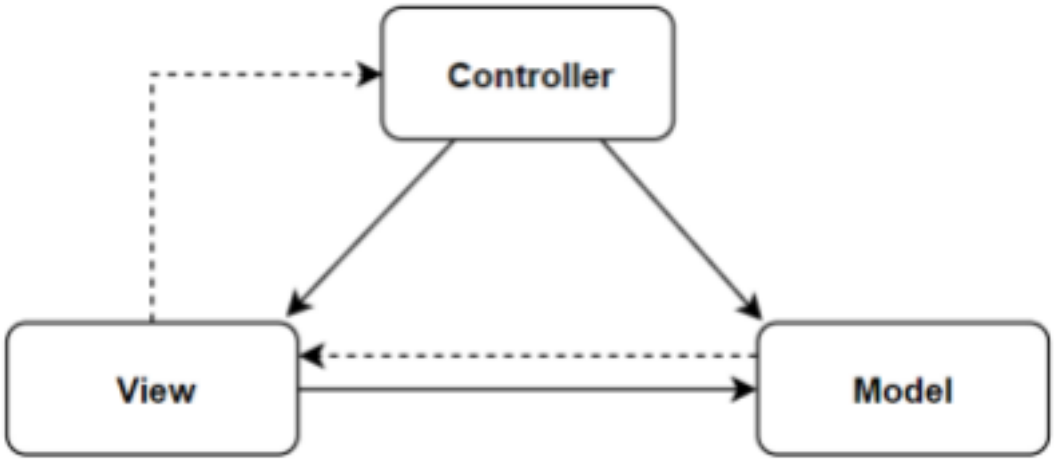


Рис.1.1. Діаграма взаємодії між елементами шаблону MVC

Модель – одна с частин шаблону проектування MVC, яка відповідає за логіку програмного додатка. Модель повинна бути повністю незалежною від інших частин програмного додатка. Дані у моделі можуть змінюватись, але при цьому спосіб їх представлення повинен бути незалежним та незмінним. Ознаки моделі:

- інкапсуляція ядра даних;
- містить логіку програмного додатка;
- як правило – це робота з БД.

Представлення – одна с частин шаблону проектування MVC, яка відповідає за представлення даних програмного додатка, які були отримані від моделі. Воно не може впливати на модель, а доступатись лише для отримання даних та реалізовувати їх відображення.

Приклади представлення – сторінка HTML, Windows Form. Контроллер – одна с частин шаблону проектування MVC, яка відповідає за керування представленням програмного додатка. Контроллер

може вплинути на модель для надання представленню інших даних. Також, він відповідає за відстежування дій користувача – зміни положення курсора миші, натискання кнопки, ввід даних в текстове поле. Потім в залежності від типу дії, він передає запит до моделі або до представлення для виконання дії користувача.

Окрім MVC існують похідні шаблони проектування – MVP, MVVM. Model-View-Presenter (MVP) – використовується, коли логічне поєднання даних неможливе. Прикладом використання цього шаблону є Windows Form.

Model-View-View Model (MVVM) – використовується, коли логічне поєднання даних можливе без використання інтерфейсів представлення. Прикладом MVP є технологія WPF.

У мові програмування Swift концепція шаблону проектування MVC досягається за рахунок стандартних інструментів розробки. В результаті використання MVC програмний додаток стає надійним та здатним до масштабування.

РОЗДІЛ 2
АНАЛІЗ ТА ПІДГОТОВКА НАПОВНЕННЯ ДОДАТКУ

Маючи документацію функціоналу додатку, починається опрацювання наповнення, по основним напрямкам нашого застосунку. Весь застосунок буде ділитися на ігри, головоломки, IQ тест, та тренування в які групуються ігри.

2.1. Пошук та аналіз IQ тестів

Для того щоб повноцінно оцінити іq користувачів, проаналізувавши в доступних ресурсах реальні тести, вибір пав на тест “Прогресивні матриці Рівена”.

Методика призначена для вивчення логіки мислення.

Людині, яка проходить тест, показують малюнки з фігурами, пов’язаними певним співвідношенням. Не вистачає однієї фігури, а внизу вона дана серед 6-8 інших фігур.

Завдання випробуваного - встановити закономірність, що з'єднує фігури на малюнку, і на анкеті вказати номер потрібної фігури із запропонованих варіантів (рис. 2.1.).

Тест складається з 60 таблиць (5 серій). Кожна серія таблиць містить завдання зростаючої проблеми. При цьому характерним є ускладнення типу завдань із серії в серію.

У серії А - використовується принцип встановлення зв'язку в структурі матриць. Тут стоїть завдання доповнити недостатню частину основного зображення одним із фрагментів, поданих у кожній таблиці.

Виконання завдання вимагає ретельного обстеження. Аналіз структури основного зображення та виявлення тих самих ознак в одному з кількох фрагментів. Потім відбувається злиття фрагмента, порівняння його з середовищем основної частини таблиці.

У серії Б - використовується побудована за принципом аналогії між парами фігур. Випробуваний повинен знайти принцип, за яким будується фігура в кожному конкретному випадку, і, виходячи з цього, підібрати відсутній фрагмент. Важливо визначити вісь симетрії, за якою фігури розташовані в основному зразку.

У серії С - побудована за принципом прогресивної зміни фігур матриць. Ці фігури в рамках однієї матриці стають дедалі складнішими, є лише їх безперервний розвиток. Збагачення фігур новими елементами підпорядковується чіткому принципу, знайшовши який, можна вибрати відсутню фігуру.

У серії D - побудована за принципом перестановки фігур у матриці. Суб'єкт повинен знайти цю перебудову, яка відбувається в горизонтальному і вертикальному положеннях.

У серії Е використовується принцип розкладання фігур основного зображення на елементи. Відсутні фігури можна знайти, зрозумівши принцип аналізу та синтезу фігур.

Методичні вказівки до тесту.

Інструкція: Тест суворо регламентований за часом, а саме: 20 хвилин. Щоб утримати час, необхідно подбати про те, щоб спільна команда: «Почати перевірку» — ніхто не відкривав таблицю і не підглядав. Через 20 хвилин дається команда, наприклад: «Закрити столи для всіх». Мета цього тесту така: "Всі наші дослідження проводяться виключно в наукових цілях, тому у відповідях потрібна чесність, продуманість, щирість і точність. Цей тест призначений для прояснення логіки вашого мислення". Потім візьміть таблицю і відкрийте для всіх 1-шу сторінку: "На малюнку немає жодної

фігури. Праворуч 6-8 пронумерованих фігур, одна з яких потрібна. Потрібно визначити візерунок, який з'єднує фігури в малюнок і вкажіть номер потрібної фігури у виданому вам аркуші» (можна показати на прикладі одного зразка). При виконанні тестових завдань необхідно контролювати, щоб респонденти не списували один одного. Через 20 хвилин подайте команду: «Закрийте всі столи!»

Зберіть для них бланки та таблиці. Переконайтеся, що номер теми позначено олівцем у правому кутку реєстраційної форми.

Так як наш додаток позиціонує себе як релаксійно-освітнє, додано більше часу користувачам для проходження данного тесту, та за дизайном треба адаптувати його під застосунок, а також дещо змінити варіації ілюстрацій, не змінюючи зміста запитань, для того щоб преподнести тест як більш унікальний, заохочуючи користувачів проходити його.

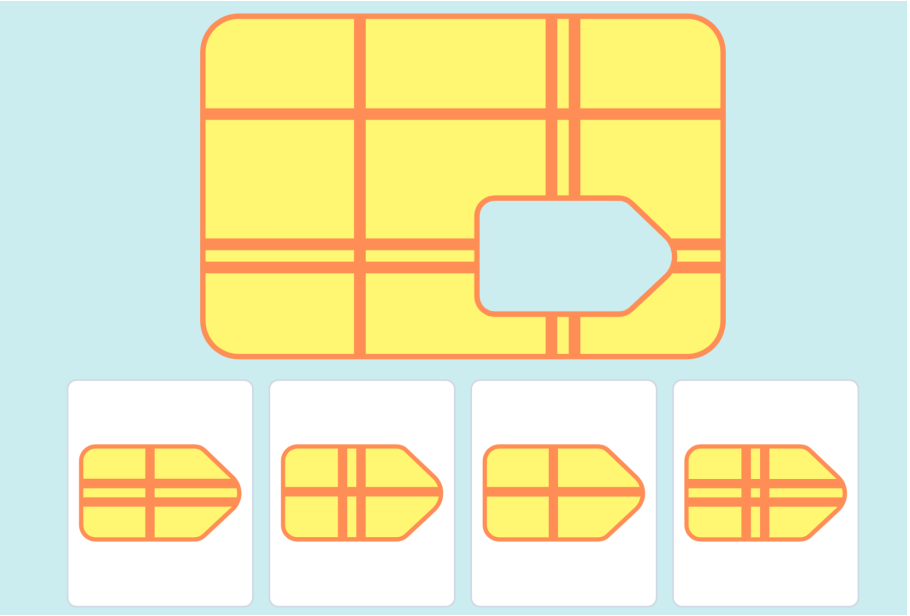


Рис.2.1. Приклад завдання тесту IQ

2.2. Підготовка репорту для тесту.

Використано шкалу для оцінки IQ користувача (табл. 2.1.) беремо за максимум значення 145, за мінімум значення 0. Використано ключ-значень для кожної секції с оригінального тесту.

	1	2	3	4	5	6	7	8	9	10	11	12
1-я серия A	4	5	1	2	6	3	6	2	1	3	4	2
2-я серия B	5	6	1	2	1	3	5	6	4	3	4	8
3-я серия C	5	3	2	7	8	4	5	1	7	1	6	2
4-я серия D	3	4	3	8	7	6	5	4	1	2	5	6
5-я серия E	7	6	8	2	1	5	1	3	6	2	4	5

Табл. 2.1. Відповідність балів

Репорт буде більш поширений, завдяки блокам де користувач може звірити дані свого результату з середніми результатами людей свого віку, подивитись за який відсоток людей він краще, подивитись яким професіям відповідають рівні iq та на яку професію користувач може звернути увагу за своїм результатом.

Шкала для відповідностей iq результатів за віком:

до 17 - 101

18-24 - 107

25-34 - 102

35-44 - 100

45-60 - 99

>60 - 94

Професії за рівнем IQ більш важко розділити, але проаналізувавши дані з різних ресурсів можна виділити такі основні напрямки та їх співвідношення до середнього рівня результату.

- фізика та астрономія - 133;
- математичні науки - 130;
- філософія - 129;
- економіка - 128;
- інженерія - 127;
- банківська справа та фінанси - 126;
- комп'ютерна наука - 125;
- хімія -124;
- біологічні науки - 121;

- гуманітарні науки та мистецтво - 120;
- політологія - 120;
- мови та література - 119;
- історія - 119;
- архітектура - 118;
- соціальні науки - 115;
- сільське господарство - 115;
- соціологія - 114;
- бізнес - 114;
- психологія - 113;
- комунікації - 111;
- медичні науки - 111;
- бізнес Менеджмент - 111;
- освіта - 110;
- бухгалтерський облік - 110;
- інша освіта - 109;
- початкова освіта - 109;
- дошкільне навчання -104.

Для того щоб повноцінно зрозуміти якість репорту та тесту, в додатку обов’язково необхідно залишати можливість користувачу залишити відгук. В нашому застосунку, якщо користувач виходить з тесту який вперше проходить не закінчивши його, то відкриється опитувальник - “Що саме не задовольнив користувача в проходженні данного тесту”. Всі відгуки збираються та аналізуються на предмет вивчення помилок.

2.3. Головоломки.

Для того щоб перевірити винахідливість, знання людини та розвинути нестандартне мислення в нашому додатку створено головоломки. В головоломці розгадувач складе частини логічним способом, щоб прийти до правильного або цікавого рішення головоломки. Треба подбати про

різноманітність. В додатку представлено 6 видів пазл і бонусні рівні з відкритими замислуватими загадками.

2.3.1. Перший вид головоломок

Набір картинок в яких є закономірність та 4 відповіді (рис.2.2 - 2.3.). Користувач повинен зрозуміти закономірності на картинках та знайти правильну відповідь.

Для того щоб вирішити такого виду завдання потрібно мислити креативно, шукати на перший погляд неочевидні рішення, підключати логіку та частіш за все відповідь є легкою та простою.

Рис.2.2. Приклад головоломок першого типу

Правильна відповідь: посередині, фіолетова частина рухається за годинниковою стрілкою.

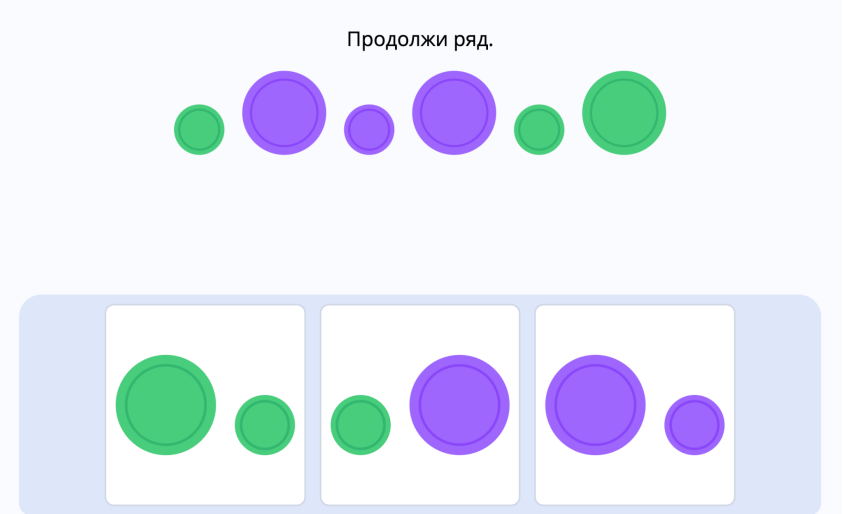


Рис.2.3. Приклад першого типу пазлів

Правильна відповідь: посередині(маленька зелена куля, велика фіолетова). Тут немає значення колір, як можна побачити на малюнку - чередуються лише розміри куль, спочатку маленька, потім велика.

Такого виду головоломок буде 30шт, для того щоб було складніше додаєм 4-й варіант відповіді.

2.3.2. Другий вид головоломок

Пошук закономірностей в цифровому ряді, дан ряд цифр (рис.2.4-2.5.), треба підібрати паттерн та знайти яка цифра продовжує ряд. Для того щоб вирішити такого виду завдання треба трошки математики та логічного мислення, в деяких закономірностях використовуються дії додавання зі збільшенням або зменшенням доданку, в деяких використовується множення, цифри можуть між собою додаватись або множитись, зводитись у квадрат.

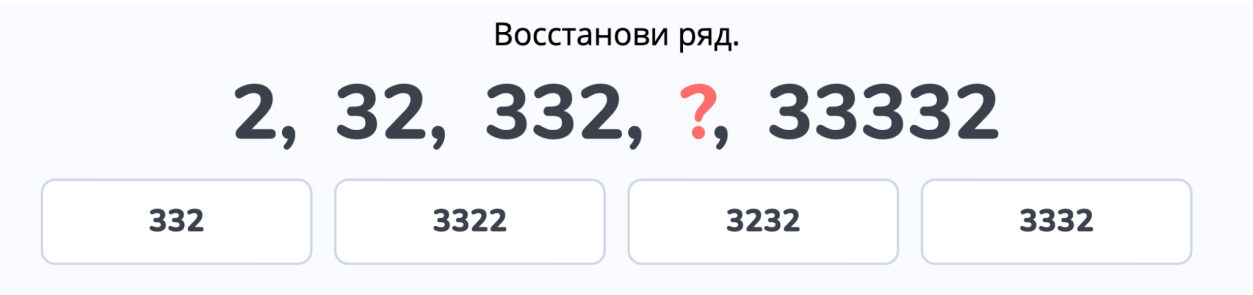


Рис.2.4. Приклад головоломок другого типу

Правильна відповідь: 3332. До першої 2 додається зліва 3 - 32, далі зліва додається ще 3 - 332, і так далі кожен раз до цифри додається зліва цифра. Все просто.

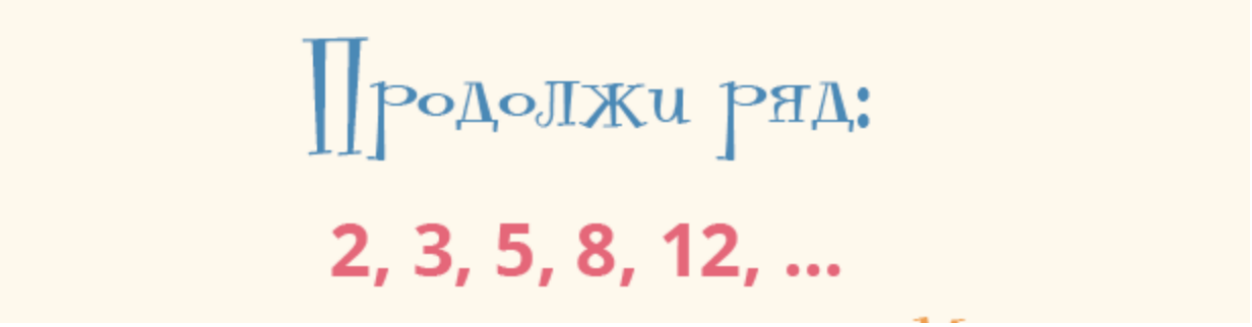


Рис.2.5. Приклад головоломок другого типу

Правильна відповідь: 17. До цифри 2 додається 1, потім до цифри 3 додається 2, і так кожного разу другий доданок збільшується на 1, тобто до 12 додається 5.

Такого виду головоломок буде також 30 різноманітних варіацій.

2.2.3. Третій вид пазлів

Вид зверху. Буде дано картинку в тривимірному форматі і треба зрозуміти як вона виглядає зверху(рис.2.6.-2.7.). Для того щоб вирішити такого виду завдання застосовується уважність та когнітивне мислення.

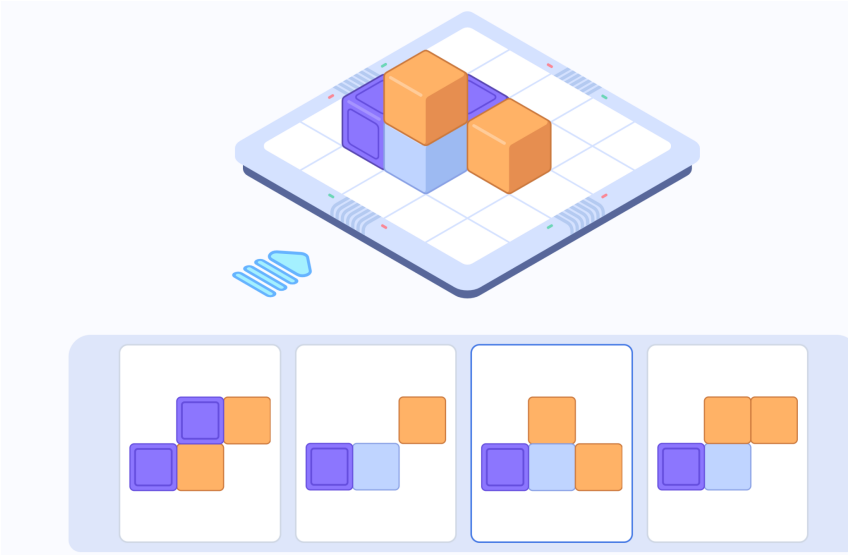


Рис.2.6. Приклад пазлів третього типу

Правильна відповідь: 1-ша. Для вирішення по перше визначено як виглядає фігура, потім підбібрано правильні кольори.

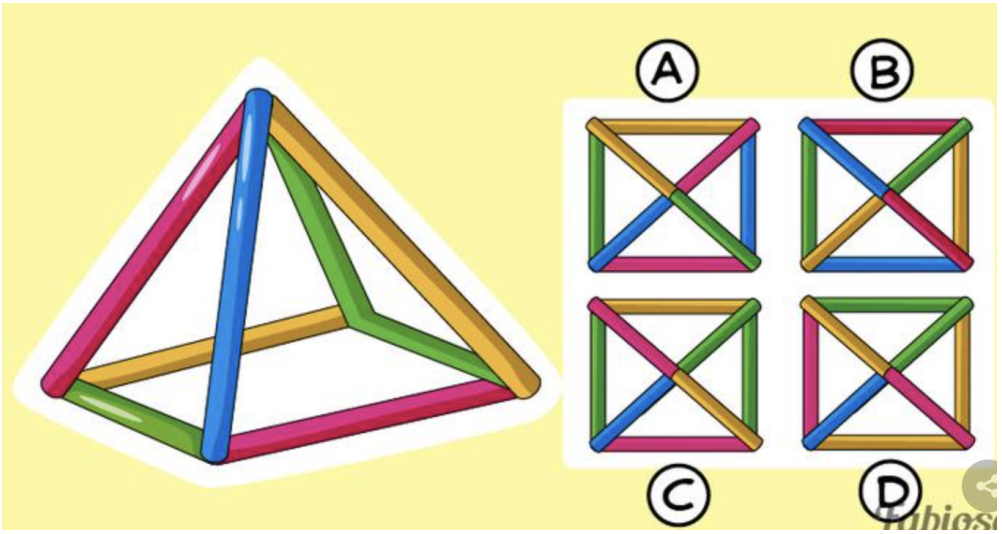


Рис.2.7. Приклад пазлів третього типу

Правильна відповідь: С, звіряючи який колір поруч з яким легко можна зрозуміти яка правильна відповідь.

2.2.4. Четвертий вид пазлів

Ррівняння. Буде дано рівняння з кількома предметами та потрібно вирішуючи знайти значення кожного предмета та вирішити останній приклад(рис.2.8.-2.9.). Для вирішення даного типу завдань необхідно мати знання в математиці, та вирішувати рівняння, або методом підбору знаходити невідомі значення.

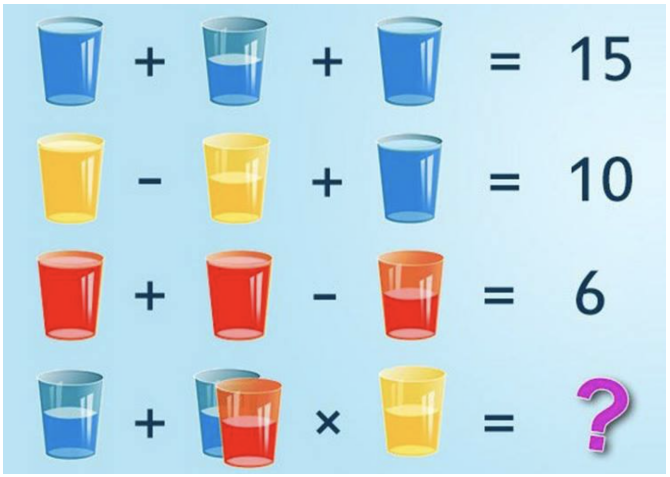


Рис.2.8. Приклад пазлів четвертого типу

Правильна відповідь: 23. Повний синій стакан = 6, пів стакана = 3 (6+3+6=15), жовтий повний стакан = 8, пів стакана = 4 (8-4+6 =10), червоний повний стакан = 4, пів стакана =2 (4+4-2 = 6). Вирішуємо останній приклад підставляючи значення - 3+ (3+2)*4 = 3+ 20 = 23.

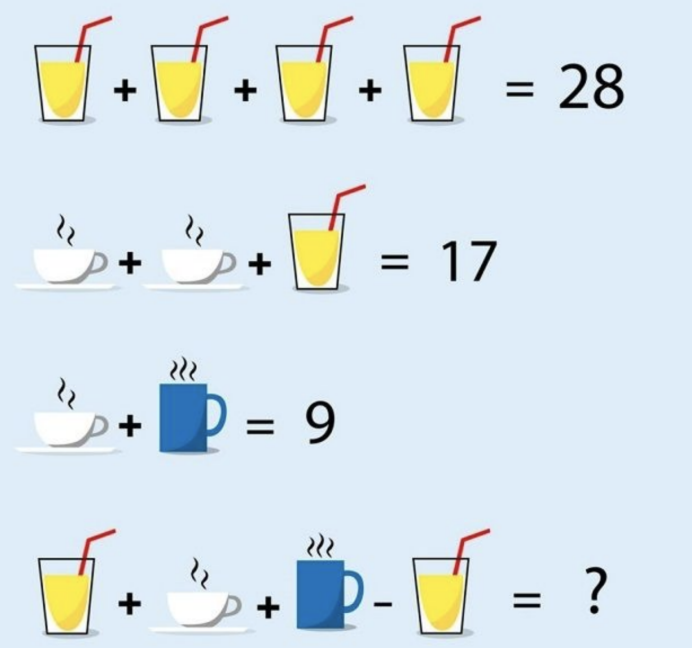


Рис.2.9. Приклад пазлів четвертого типу

Правильна відповідь: 9.

Перший спосіб вирішення: 28/4 = 7, жовтий коктейль = 7, 17-7/2 = 5, біла чашка = 5, 9-5=4, синя чашка =4, підставляємо значення в останній приклад. 7+5+4-7 = 9.

Другий спосіб: подивившись на останнє рівняння - можна його скоротити то вигляду біла чашка+синя чашка, і в третьому прикладі ми маємо відповідь 9.

Такого виду пазлів також буде 30 штук. Та додамо 4 варіанти відповідей.

2.3.5. П’ятий тип головоломок

Симетрія. Тут починає працювати уявлення, лист склали один, два або 4 рази, вирізали фігуру, треба зрозуміти як буде виглядати фігура після того як лист розгорнути(рис.2.10-2.11.).

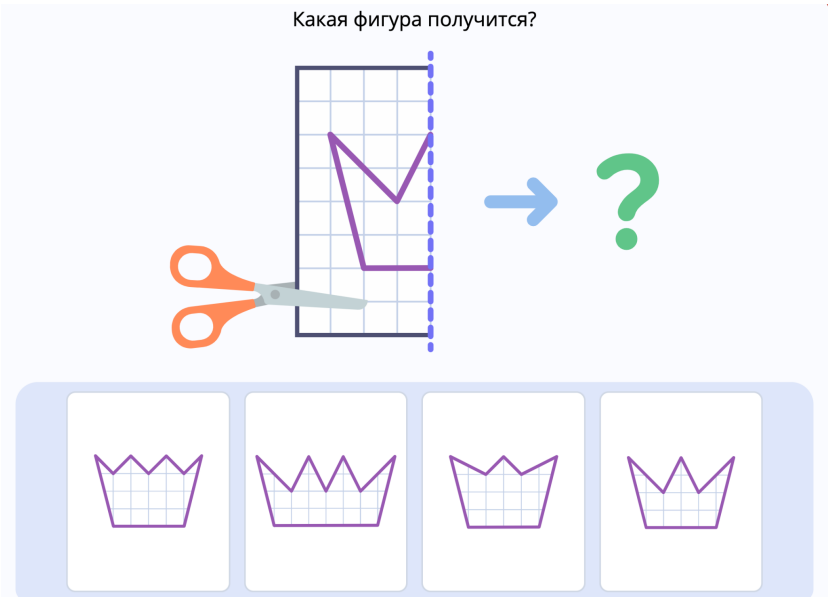


Рис.2.10. Приклад пазлів п’ятого типу

Правильна відповідь: 4-а. Треба уявно підставити такий же малюнок зправа від лінії, просчитати клітинки на точках та правильна відповідь стане зрозумілою.

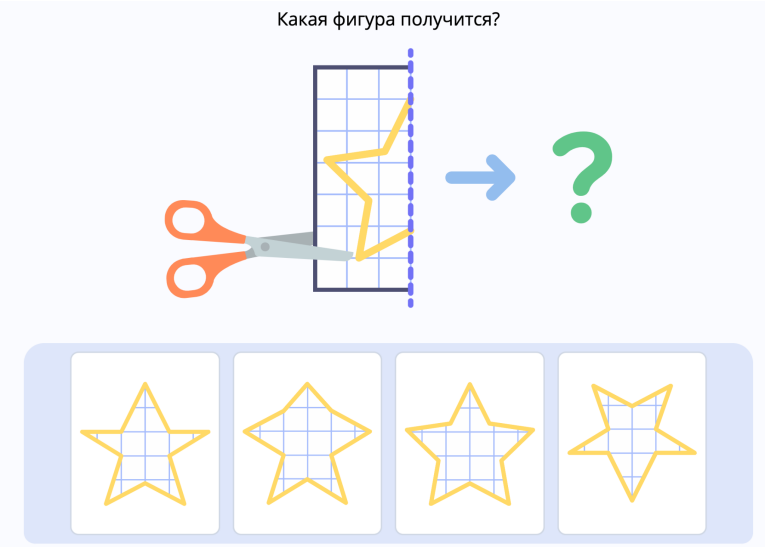


Рис.2.11. Приклад пазлів п’ятого типу

Правильна відповідь: 3-а. Уявно підставляємо та отримаємо відповідь.

Для створення таких пазлів використовуємо лист бумаги та створюємо 30 видів головоломок різної складності.

2.3.6. Шостий тип пазлів

Математичні завдання. Найскладніша частина головоломок. Використовуються математичні знання, уважність, нестандартне мислення.

Буде дана певна закономірність в картинках де використовується математичні дії(рис.2.12-2.13.).

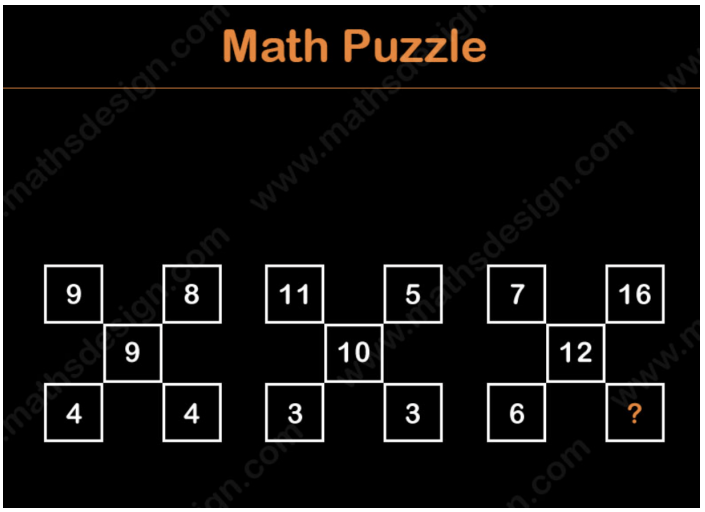


Рис.2.12. Приклад пазлів шостого типу

Правильна відповідь: 5. 9+8 - (4+4) =9. 11+5 - (3+3) = 10. 7+16 - (6+?) =12

7+16-6-12 = 5. Спочатку потрібно зрозуміти за яким паттерном зв'язані числа на перших двух фігурах, потім підставляти паттерн для рішення в останню.

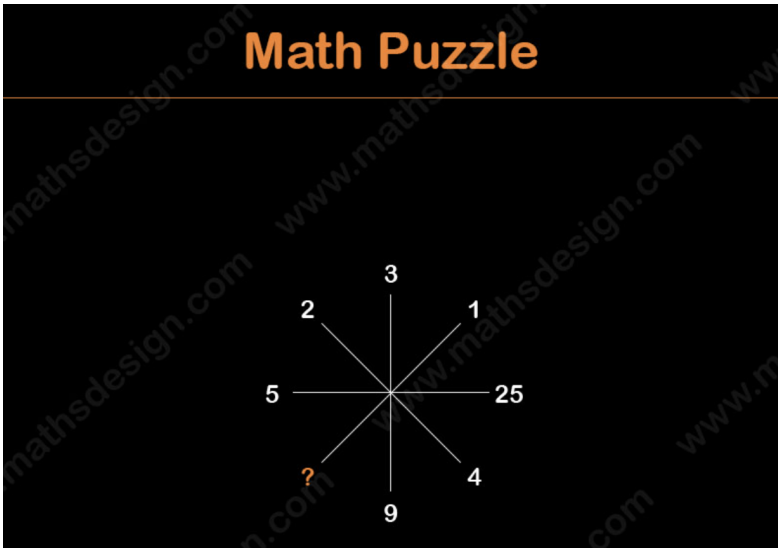


Рис.2.13. Приклад пазлів шостого типу

Правильна відповідь: 1. 2*2=4, 3*3=9, 5*5=25, 1*1 =1.

Такого типу завдань також буде 30 штук з 4-мя варіантами відповідей.

2.3.7. Бонусні пазли

Бонусні пазли будуть складатись з різного типу питань з ілюстраціями, питання будут загадками(рис.2.14-2.15).

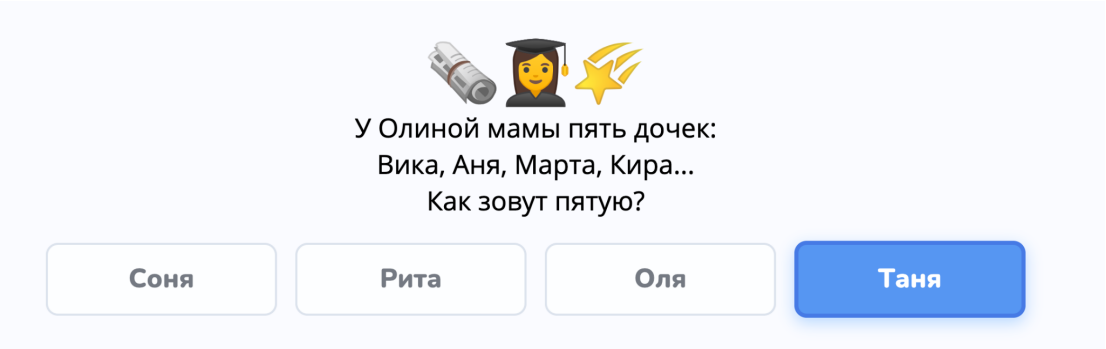


Рис.2.14. Приклад бонусних рівнів на пазлах

Правильна відповідь: Оля, це дуже просто.

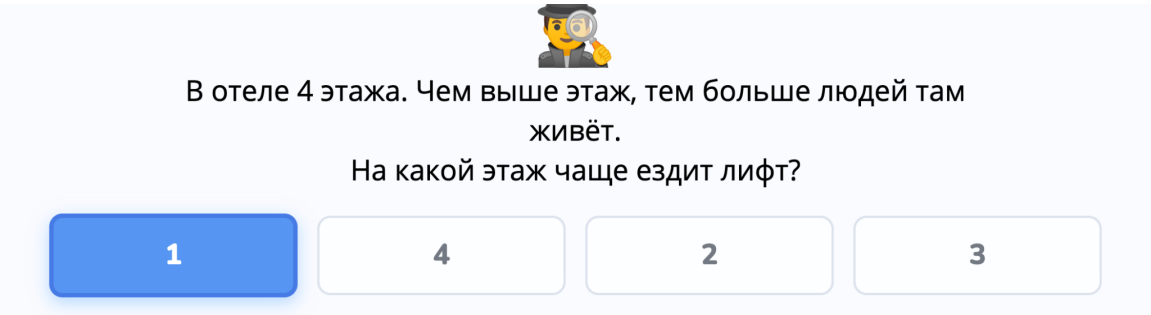


Рис.2.15. Приклад бонусних рівнів на пазлах

Правильна відповідь: 1, люди з усіх поверхів чаще за всіх приїжджають на 1-й поверх.

Таких питань буде 30, тому зо з 180 пазлів кожен 7й бонусний.

Частина наповнення пазлів готова, та передається на дизайн, щоб зробити унікальний вид підходящий до основного вигляду додатку.

2.4. Наповнення ігор.

Ігри будуть поділені на кілька категорій, для розвитку уважності, пам’яті, фокусування, математики. В кожній категорії буде по декілька ігор.

2.4.1. Категорія “Пам’ять”

Знайди пару. Гра складається з 5 раундів, та 10 рівнів складності.

Перша наша гра буде основуватись на пошуку однакових предметів, перед нами буде грід на якому декілька пар предметів(рис.2.16.).

Відкривається грід, показуються всі пари предметів , закриваються, необхідно запам'ятати та знайти всі пари. По тапу на плітку гриду предмет показується:

- якщо пара знайдена - ця пара пропадає, коли знайдено всі пари на гриді, то в центрі з'являється зелений кругляш, вібро та звук правильної відповіді, гравець переходить на наступний раунд.
- якщо пара не знайдена - відкриті 2 предмети перевертаються, користувач продовжує шукати пари.

У користувача є обмежений час щоб закінчити 5 раундів, якщо час закінчився, а користувач не закінчив всі раунди в центрі екрана з'являється іконка будильника і перехід на екран з результатами.

Якщо час не закінчився і користувач пройшов всі 5 раундів, то в центрі екрану з'являється зелений кружок із прапорцями та перехід на екран з результатами. Підвищується рівень складності.

Гра складається з десяти рівнів складності, у кожному п’ять раундів, раунди складаються з дев’яти доступних ігрових рівнів

У кожному ігровому рівні є розмір гриду, очки за кожну знайдену пару, час показу предметів, кількість пар.

Паузи:

Якщо користувач ставить на паузу, коли показуються предмети або не знайдена жодна пара або закінчено раунд - після повернення в гру предмети показуються заново, але вже інші.

Якщо користувач ставить на паузу коли вже знайдено одну і більше пар - стан гри після паузи такий самий як і перед.

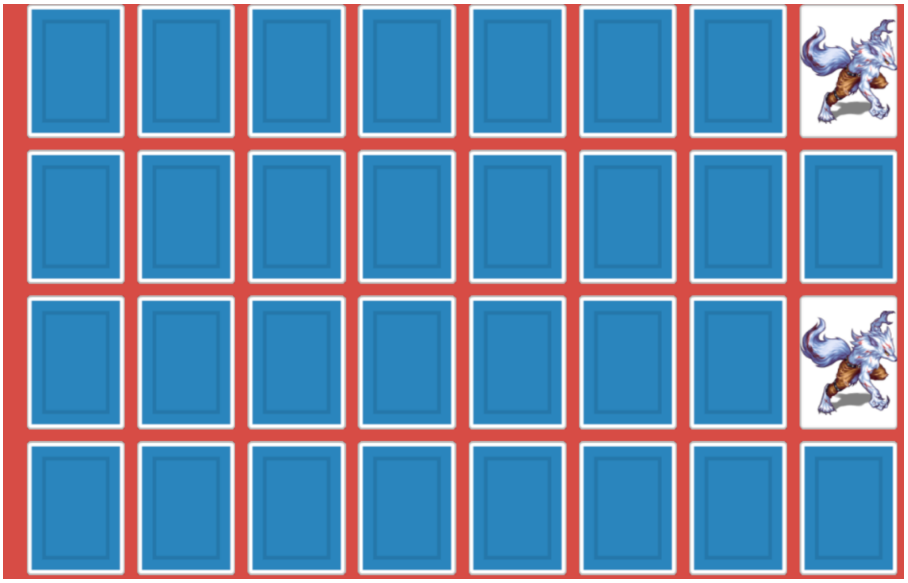


Рис. 2.16. Приклад гри до категорії “Пам’ять”

2.4.2. Категорія уважність.

Гра пошук анаграм.

Мета гри: знайти анаграму вихідного слова із 4-х варіантів за певний час.

Анаграма – слово, отримане внаслідок перемішування буквеного складу вихідного слова.

Після показу слова та 4 блоків з варіантами відповіді - тапнути на блок з відповідним варіантом(рис.2.17.):

- якщо відповідь вірна - фон блоку змінює колір на зелений, у центрі екрана з'являється зелений кружок+ вібро та звук правильної відповіді, гравець переходить на наступний раунд;
- якщо відповідь не вірна - фон блоку змінює колір на червоний, зеленим фоном виділяється правильна відповідь, у центрі екрану з'являється червоний кружок + вібро та звук неправильної відповіді, гравець повертається на раунд назад (але не менше ніж 1).

У користувача є обмежений час щоб завершити всі десять раундів, якщо час закінчився в центрі екрана з'являється іконка будильника та перехід на екран з результатами.

Якщо користувач закінчив всі 10 раундів протягом зазначеного часу, в центрі екрана з'являється зелений кружок із прапорцями та перехід на Score screen. Підвищується рівень складності.

Гра складається з десяти рівнів складності, по десять раундів у кожному рівні. Раунди складаються з дев’ятнадцяти ігрових рівнів.

В ігровому рівні є бали за проходження, кількість букв у слові, кількість неправильних букв якими будуть замінені букви в анаграмах.

У рамках одного рівня складності мають бути унікальні слова.

За правильну відповідь у конкретному раунді нараховуються бали, за неправильну – віднімаються бали рівні 30% від переможних цього раунду.

Створюється список слів, для бази звідки брат ці слова. Спочатку буде переставлятися місцями слова в яких 3- 4 букви, та в не правильних відповідях щоб заплутати мінятись по одній букві. Також зазначимо які букви можуть мінятись на які для того щоб підібрати схожі букви. За зростанням рівня складності, збільшується кількість букв в слові, та кількість виправлених букв в варіантах відповідей.

Рис .2.17 Приклад гри для категорії “Уважність”

Правильна відповідь: 1.

2.4.3. Категорія фокусування.

Гра на впорядкування цифр.

Мета гри: Якнайшвидше відсортувати числа в порядку зростання від 1 до N.

З'являється грід з плитками та числами на них (колір та розмір шрифту чисел може бути різним залежно від складності рівня), необхідно в порядку зростання тапати на числа від 1 до N(рис.2.18.), по натисканню на плитку:

- якщо відповідь вірна - фон плитки змінює колір на синій, колір цифри на білий, якщо вибрано всі цифри на гриді - у центрі екрана з'являється зелений кружок, вібро+звук правильної відповіді. Гравець переходить до наступного раунду;
- якщо відповідь не вірна - плитка змінює колір на червоний(підсвічується на кілька мілісекунд), користувач далі продовжує вибирати цифри в рамках цього раунду.

У користувача є обмежений час щоб завершити всі п’ять раундів, якщо час закінчився в центрі екрана з'являється іконка будильника і перехід на екран результатів.

Якщо користувач закінчив всі п’ять раундів протягом зазначеного часу, у центрі екрана з'являється зелений кружок з прапорцями та перехід на екран результатів. Підвищується рівень складності.

З ростом складності збільшується кількість цифр, та змінюється розмір шрифтів, тобто можуть бути маленькі і великі, та додається більше різноманіття кольорів.

Приклад гри:

Рис.2.18 Приклад гри до категорії “Фокусування”

2.4.4. Категорія математика.

Гра на вирішення математичних прикладів.

Мета гри: Визначати правильно вирішено приклад чи ні.

Відбувається показ вже вирішеного прикладу з відповіддю, необхідно вибрати правильно або неправильно вирішено приклад (рис.2.19.).

Якщо відповідь користувача вірна - у центрі екрана з'являється зелений кружок, вібро та звук правильної відповіді. Гравець переходить до наступного раунду.

Якщо відповідь користувача не вірна - з'являється екран з правильною відповіддю. Гравець повертається на раунд назад (але не менше ніж 1й).

У користувача є обмежений час щоб завершити всі сім раундів, якщо час закінчився в центрі екрана з'являється іконка будильника і перехід на екран з результатом.

Якщо користувач закінчив усі сім раундів протягом зазначеного часу, у центрі екрана з'являється зелений кружок з прапорцями та перехід на екран з результатом. Підвищується рівень складності.

Гра складається з десяти рівнів складності, по сім раундів у кожному. Раунди складаються з 25-ти ігрових.

У кожному ігровому рівні є бали за проходження рівня, діапазон чисел для кожної позиції та дії у прикладі. За правильну відповідь у конкретному раунді нараховуються бали, за неправильну – віднімаються бали рівні 30% від переможних цього раунду.

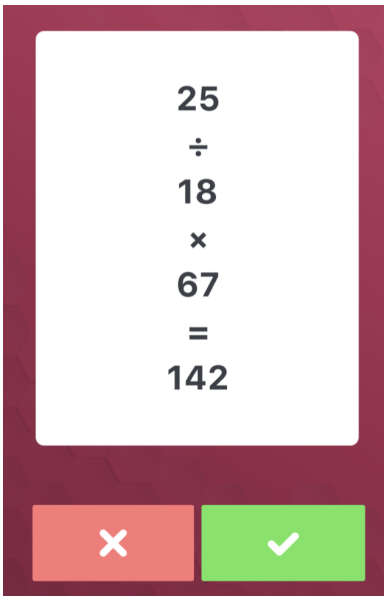


Рис. 2.19. Приклад гри до категорії “Математика”

Наповнення ігор готово, далі дані яки були зібрано та приклади вигляду ігор передаються на дизайн для того щоб вони мали уніфікований дизайн та відповідали тематиці нашого застосунку, були адаптовані для сприймання користувачеві.

2.5. Поділ на платний та безкоштовний контент для користувачів.

Для того щоб додаток приносив прибуток - необхідно обрати області для поширення платних функцій, та обрати оптимальні засоби монетизації.

В нашому застосунку будемо комбінувати фримиум та підписки.

Фримиум є природною моделлю монетизації. Це більш поширений метод монетизації для додатків. Розробник відкриває безкоштовний доступ до основних функцій, а розширені доступні лише за плату. Власне, це принцип Shareware, успадкований від комп’ютерних програм – умовно-безкоштовний.

Приклад програми з такою моделлю монетизації: ToDo list, в якому список справ можна зберігати безкоштовно, а за синхронізацію з хмарою доведеться заплатити $0,99.

Переваги цієї моделі:

- Звично для користувачів;
- Простота виконання;
- Сама програма залишається безкоштовною, що дозволяє легко залучати користувачів і створювати лояльну аудиторію.

Мінуси моделі фріміум:

- Як і в разі покупок через додаток, платить дуже невеликий відсоток користувачів;
- Необхідно зробити безкоштовний досвід максимально преміальним, щоб користувач хотів купити відсутню частину - якщо безкоштовні функції програми недостатньо хороші, користувачі втратять мотивацію платити;
- Якщо це не модель підписки, то буде один платіж від одного користувача, і його LTV буде складатися лише з нього.

Підписка - самий «модний» спосіб монетизації, який зараз активно пропагують Apple і Google. Загалом це схоже на фріміум, тільки гроші з користувачів списуються на регулярній основі — у додатку є якась безкоштовна частина або контент, а додатковий контент доступний лише за щомісячну плату.

Приклад додатка з такою моделлю монетизації: додаток із серіалами, який дозволяє переглядати за щомісячну плату.

Переваги цієї моделі:

- Регулярні виплати;
- збільшення LTV та лояльності користувачів;

- Власники платформ настійно заохочують використовувати відповідну підписку (наприклад, Apple на другий рік з ціни підписки вже не бере стандартні 30%, а лише 15%).

Мінуси:

- Дуже важко зібрати лояльну платну аудиторію;
- Потрібен дійсно преміальний контент, за який люди хотіли б платити – і не просто платити, а регулярно;
- Порівняльна складність технічної реалізації та відстеження передплатної діяльності;
- Підписка не підходить для більшості програм вмісту, які не мають регулярних оновлень. Наприклад, ви, ймовірно, не зможете отримати рецензію в магазині на гру, для якої потрібна підписка.

Безкоштовний контент:

1. IQ тести - проходження тесту буде безкоштовним.

А ось репорт та його результат буде доступний лише для преміум користувачів, рівень вони свій можуть дізнатись безкоштовно, в деталізацію за репортом вже підписавшись(рис.2.20.).

Але для користувачів які не хочуть брати підписку - дамо можливість просто купити репорт. Після покупки - користувач отримує лише репорт, інший платний контент залишається недоступним.

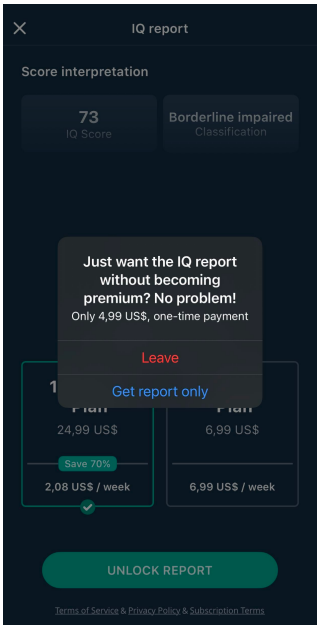


Рис. 2.20. Покупка репорту

2. Головоломки - безкоштовний контент, але додано обмежену кількість життів.

Користувач може грати в пазли безкоштовно, але в нього буде обмежена кількість можливостей помилитись, коли користувач помиляється віднімається 1 життя, якщо життя закінчились то не може проходити, і потрібен час щоб вони відновились. Щоб не чекати коли відновлюються серця пропонуємо купити підписку щоб відкрити повний контент включаючи безлімітні життя(рис.2.21.).

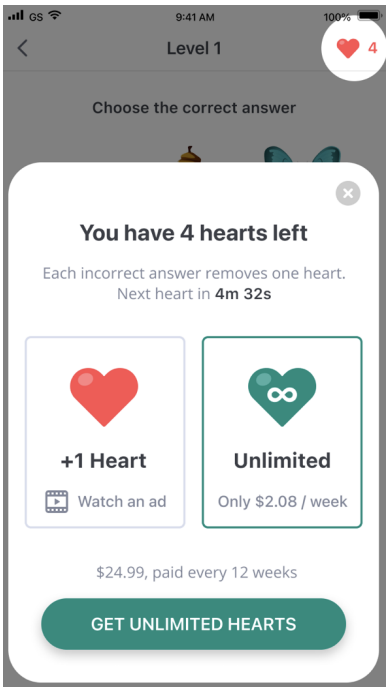


Рис. 2.21. Підписка на пазлах

Інший контент буде платний, але зробимо сторінку з тренуваннями де кожного дня буде даватись обмежена кількість (3шт) безкоштовно(рис.2.22.), а якщо користувач хоче грати в ігри далі то ми дамо широкий вибір підписок та можливостей для оплати нашої праці.

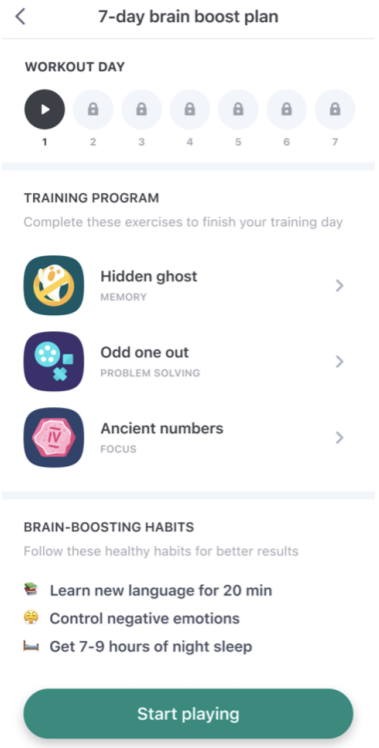


Рис. 2.22 Сторінка з тренуваннями для безкоштовних користувачів

2.6. Аналіз підписок.

2.6.1 Безкоштовна пробна версія та вступ

Перед покупкою завжди хочеться «помацати» товар, оцінити плюси і мінуси. Не позбавляємо користувачів цієї можливості – нехай вони оцінять наш додаток протягом пробного періоду. Можливість 3 дні користуватись безкоштовно, а далі буде активована підписка на 7 днів(рис.2.23-2.24.).

2.6.2 Довічна підписка

Для користувачів які хочуть відкрити весь контент одним платежем, для них є можливість довічно користуватись преміальним контентом. Пропонуємо лайфтайм після вступу та всередині додатку після того як

користувач натискатиме на “преміальний” закритий контент буде екран з підписками(див.рис.2.23-2.24.).

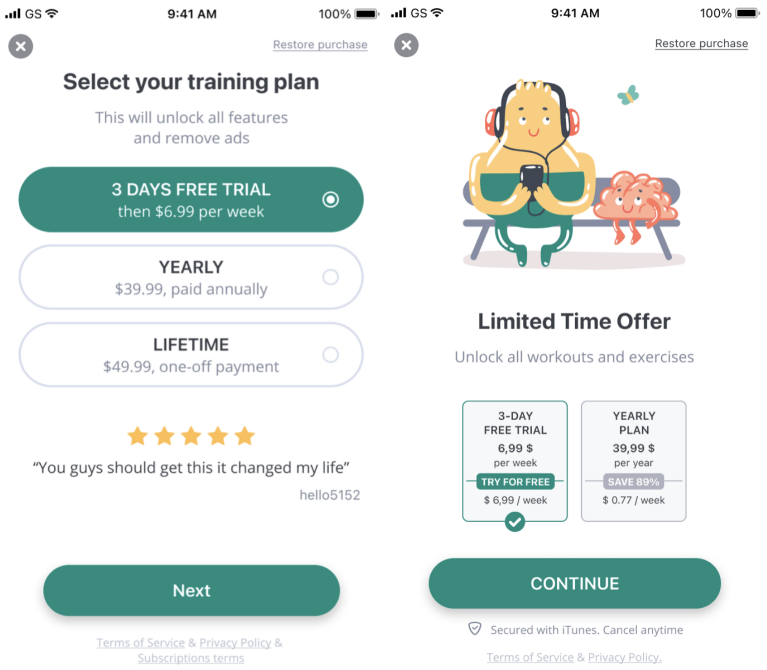


Рис. 2.23. Підписка на онбордингу Рис. 2.24. Підписку всередині

2.6.3 Річна підписка.

Можливість відкриття контенту для користувачеві на 1 рік. Цей офер буде бачити користувач при кожному заході, він дешевше ніж довічна(рис.2.25).

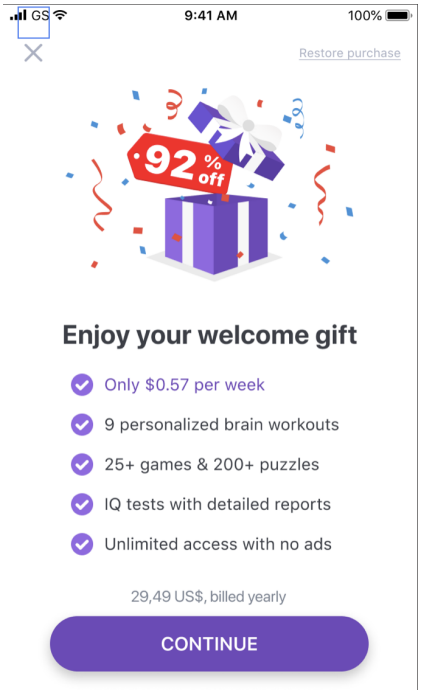


Рис. 2.25. Річна підписка у додатку

2.6.4 Знижка.

Довгострокові знижки на підписку переконують користувачів зробити покупку краще, ніж будь-яке рекламне відео. Скажімо, додаток коштує 50 доларів на місяць. Запропонована знижка 10% на річну підписку каже користувачам, що можна заощадити 60 доларів(рис.2.26.).

Якщо користувач все ще хоче піти після закінчення періоду знижки, можна продовжити дію знижки. Краще отримати на 5-10% менше прибутку, ніж нічого не отримати.

Для користувачів які активно грають дамо можливість дешевше річну підписку.

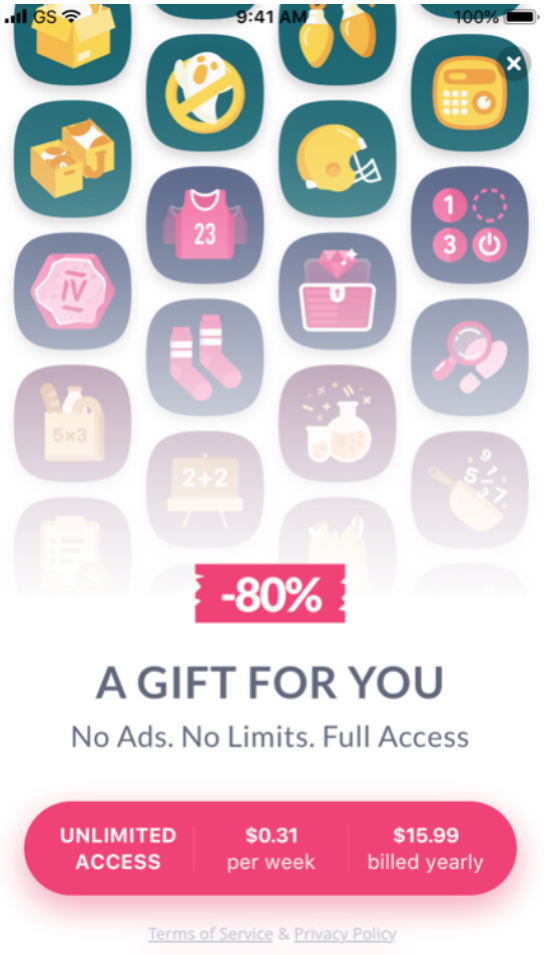


Рис.2.26. Екран зі знижкою активних користувачів всередині додатку

Користувач може обрати підписку яка йому більш підходить за фінансами.

Контент який буде закрит для не підписаних користувачів: ігри в вільному користуванні - лише обмежена кількість на день та з можливістю 1 раз зіграти.

Репорт після проходження тесту та безмежна кількість життів для пазлів.

РОЗДІЛ 3

РОЗРОБКА ЗАСТОСУНКУ

3.1 Xcode ознайомлення та створення проекту

Xcode — це інтегроване середовище розробки (IDE) Apple, яке надає розробникам інструменти для створення додатків для iPhone, iPad, Mac, Apple Watch і Apple TV. Остання версія - Xcode 8 - доступна для безкоштовного скачування. Xcode працює лише на комп’ютерах під керуванням OS X (iMac, Macbook і Mac Mini). Річна ліцензія розробника на публікацію програми в iTunes або Mac OS X App Store коштує 99 доларів.

Середовище розробки Xcode забезпечує ефективність як малих, так і великих команд розробників. Xcode IDE використовує схему поділу даних програми Model-View-Controller (MVC) для сегментації кожного рівня програми. Це полегшує внесення змін до коду. Наприклад, шар інтерфейсу користувача розділений такими інструментами, як новий Interface Builder, який можна використовувати для розміщення візуальних елементів керування на екрані. Auto Layout дозволяє динамічно керувати поданням об'єктів для екранів різного розміру; за допомогою Storyboard зручно розміщені екрани додатків; Режим попереднього перегляду швидко покаже, як виглядають екрани програми. Жоден з цих інструментів не впливає на код, який створюється.

Раніше коди писали на Objective-C. У червні 2014 року Apple представила Swift, нову мову для створення мобільних додатків. Це найшвидше вивчення мови в порівнянні з іншими мовами програмування. Людям з Apple знадобилося достатньо часу, щоб розробити Swift. В результаті всіх зусиль з'явилася мова, яку розробникам вивчити набагато легше, ніж той же Objective-C. Крім того, Swift і Objective-C можуть бути присутніми в одному проекті.

Xcode 8 — це радикально швидка версія, вона містить майже все, що потрібно для розробки додатків для всіх пристроїв Apple. Зокрема, нові редакційні розширення. Параметр «Проблеми під час виконання» попереджає вас про дефекти, які Xcode автоматично виявляє. Thread Sanitizer відстежує зміни даних та інші помилки. Інтерфейс перевірено View Debugger, сучасним інструментом з високою візуальною точністю. Налагоджувач пам'яті попереджає вас про «витік пам'яті» та приховані помилки.

Окремі розробники можуть працювати з Xcode. Програмний код перевіряється в репозиторії Git, після чого ним можна поділитися з іншими. Підтримується концепція безперервної інтеграції та інструментів тестування. Поточна версія Xcode також включає інструмент Test Assistants – забезпечує коректність коду та тестів; Тестовий інструмент Test Navigator; Підтримка ботів у Xcode Server, які запускаються після перевірки коду в елементі, є інструменти для перевірки продуктивності, асинхронності та тестів UI.

Щоб розмістити програму, створену в IDE Xcode в iTunes App Store, знадобиться ліцензія розробника, яка надаватиме доступ до iTunes Connect, інструменту для розміщення програм. Тут не потрібен iTunes Connect для корпоративних програм, але знадобиться сертифікат, щоб зареєструвати кожну програму, перш ніж публікувати її в особистому магазині.

Для розробки під iOS нам знадобиться спеціальне середовище програмування, яке називається XCode. XCode дозволяє використовувати

мови Swift та Objective-C для створення програм під iOS та Mac OSX. Вона є безкоштовною, її можна встановити з App Store(рис.3.1.).

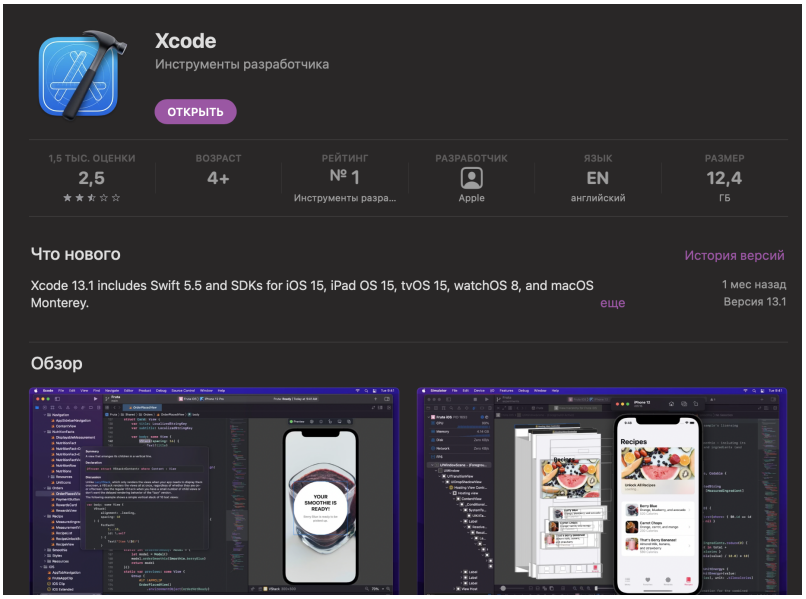


Рис. 3.1. Сторінка стора з програмою

Після встановлення запустимо XCode. За замовчуванням нам відкривається стартовий екран із вибором опцій для створення нового проекту, а також список раніше створених проектів(рис.3.2.).

Рис.3.2. Початок роботи

XCode Playground

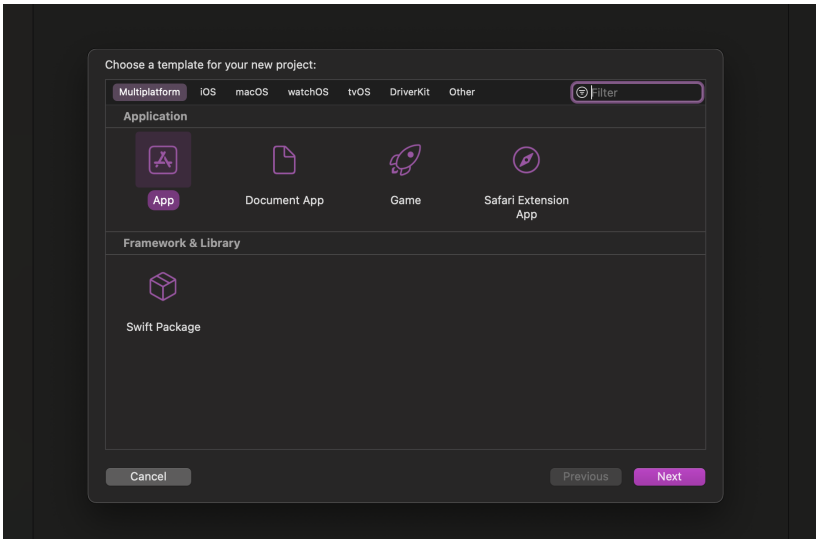


Рис. 3.3. Створення проекту

У цьому вікні виберіть пункт Get started with a playground.

Після цього потрібно буде вказати тип проекту. Виберемо перший тип проекту (рис.3.3.).

Далі треба буде вказати, під яким ім'ям зберігатиметься проект. Вкажемо як ім'я MedDiary і натиснемо на кнопку Next, team указіваем персональную команду(рис.3.4.).

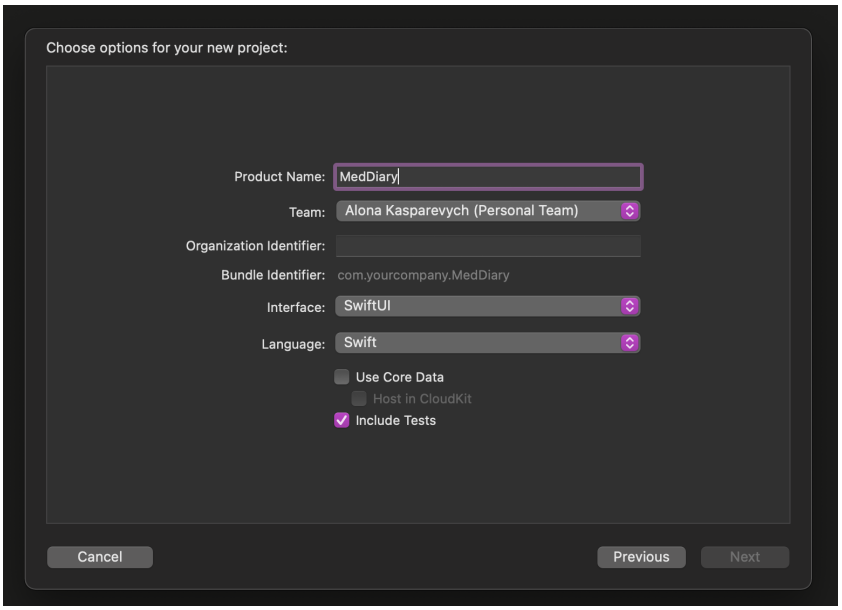


Рис. 3.4. Створення типу проекту

Після того локально створено директорію(рис.3.5.).

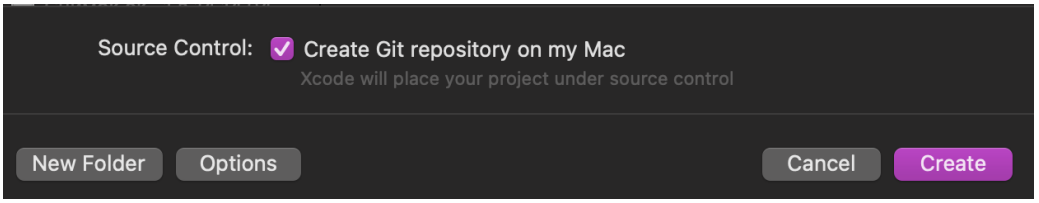


Рис.3.5. Створення директорії

І після всіх цих налаштувань нам відкриється саме середовище Playground. По суті вона є текстовим редактором з вікном консольного виводу, в якому ми можемо потренуватися з виразами та операціями мови Swift.

Далі можно створювати файли Swift, Json, view models.

3.2. Додавання бібліотек, структурування проекту

3.2.1. CocaPods - імпорт SDK.

CocoaPods керує залежностями бібліотеки для проектів Xcode.

Залежності для проекту вказуються в одному текстовому файлі, який називається Podfile. CocoaPods вирішує залежності між бібліотеками, отримує вихідний код, а потім з'єднує його разом у робочому просторі Xcode для створення проекту.

Зрештою, мета полягає в тому, щоб покращити можливість виявлення та залучення до сторонніх бібліотек із відкритим кодом шляхом створення більш централізованої екосистеми.

Установка.

CocoaPods створено за допомогою Ruby, і його можна буде встановити за умовчанням Ruby, доступним у macOS.

Використання стандартної установки Ruby вимагатиме використання sudo під час встановлення gems. (Однак це проблема лише на час встановлення дорогоцінного каменю.)

```
$ sudo gem install cocoapods
```

В новому X-code проекті:

Відкрито вікно терміналу та $ cd у каталозі проекту.

Створено файл Podfile. Це можна зробити, запустивши $ pod init.

Відкрито свій Podfile. Перший рядок має вказати платформу та підтримувану версію.

```
platform :ios, '9.0'
```

Створено цільовий розділ, написавши цільовий "$TARGET_NAME" do і закінчивши через кілька рядків.

Додайте CocoaPod, вказавши pod '$PODNAME' в одному рядку всередині цільового блоку.

```
target 'MyApp' do
pod 'ObjectiveSugar'
end
```

Збережено свій Podfile.

Запущено $ pod install

Відкрито створену MedDiary.xcworkspace. Це файл, який використовується щодня для створення програми.

3.2.2. Firebase

Firebase — це кросплатформна платформа мобільних баз даних у реальному часі, яка дозволяє програмісту зосередитися на тому, що вони найкраще пишуть у своїх програмах, не турбуючись про проблеми DevOps, як-от інфраструктура сервера та моделювання баз даних. Firebase, що підтримується Google, бере на себе складність роботи з базами даних у реальному часі, аутентифікації користувачів та роботи з робочими процесами офлайн-синхронізації.

Хоча існує багато рішень для BaaS, таких як Realm (перегляньте мій посібник з Realm.io тут на Envato Tuts +). Firebase не потребує налаштування конфігурації сервера інфраструктури, оскільки платформа піклується про хостинг і, у свою чергу, надає SDK.

На додаток до баз даних NoSQL у реальному часі, Firebase отримує аналітику, звіти про аварійне завершення роботи, автентифікацію користувачів, обмін повідомленнями в хмарі, push-сповіщення тощо. Пов'язані витрати також залежать від проекту: у міру зростання йде перехід від моделі freemium до моделі для кожного використання.

Налаштовано Firebase на iOS за допомогою CocoaPods.

Налаштування проекту.

Почнемо з клонування проекту за допомогою GitHub:

$git@github.com:tutsplus /

get-started-with-firebase-authentication-for-ios.git

...

$ git fetch --all --tags

...

$ git checkout теги / START

Відкрито файл під назвою Podfile, розташований у кореневому каталозі проекту. Цей файл який встановлює бібліотеки, які ми хочемо використовувати в нашому проекті. Додано такі рядки до визначення Firebase:

```
1  pod 'FirebaseUI'
2
3  Pod 'Firebase'
```

Збережено, а потім введено у свій термінал наступне:

```
pod install
```

Використовувати будемо MedDiary.xcworkspace замість MedDiary.xccodeproj, що дозволить нам працювати з бібліотеками залежностей, які ми налаштовуємо на CocoaPods, тому заходимо і відкрито робочу область, а потім переходимо до свого браузера.

Тепер на панелі Firebase створено новий проект(рис.3.6.).

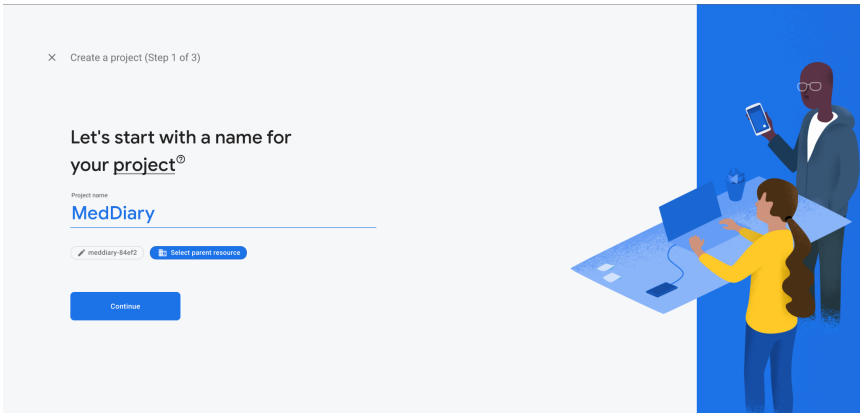


Рис. 3.6. Створення проекту в Firebasr

Потім натиснуто «Додати Firebase до вашої програми iOS», яка проведе вас через покроковий процес реєстрації програми у Firebase(рис.3.7.).

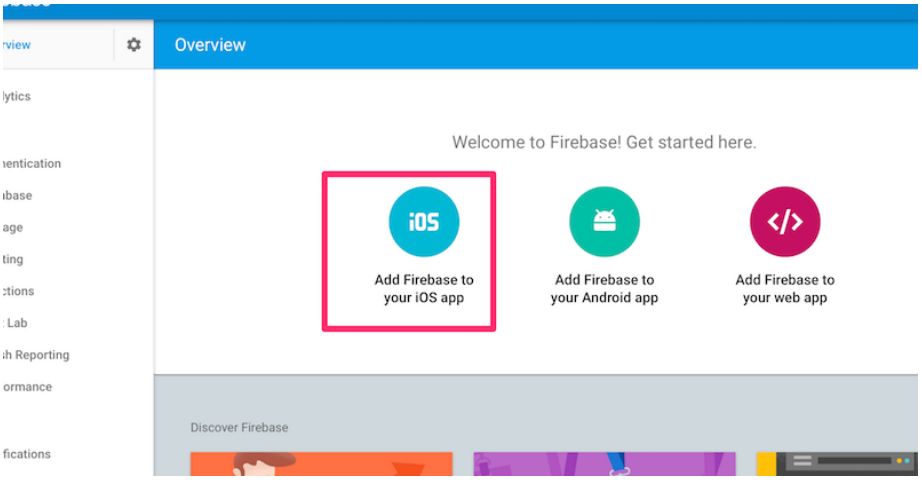


Рис. 3.7. Інтегруємо до IOS додатку

В процесі створення - налаштовується кабінет, а основні бібліотеки загружені з CocoaPods.

В Xcode, з програмою go to File> Swift Packages> Add Package Hang.

Коли з'явився запит, додано репозиторій Firebase Apple Platform SDK:

https://github.com/firebase/firebase-ios-sdk

Обрано SDK version яку будемо використовувати.

Коли Google Analytics налагоджен в нашому проекті Firebase, використовується як до Firebase Analytics. Для Analytics без IDFA колекція здатності, add FirebaseAnalyticsWithoutAdId instead.

Xcode автоматично запускає резолюцію і завантажує наші залежності в фоновому режимі.

Останній крок - додати код ініціалізації у програму. Імпортовано модуль Firebase в UIApplicationDelegate :

```
import Firebase
FirebaseApp.configure()
```

Налаштувавши основні бібліотеки можемо приступати до розробки основних елементів продукту та розробки функціональностей.

3.3.Основні екрани та навігація.

3.3.1. Tab bar

Почнемо зі створення UITabBarController и UINavigationController

AppDelegate.swift фрагмент:

```
var window: UIWindow?
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.
    window = UIWindow(frame: UIScreen.main.bounds)
    let journalVC = JournalTableViewController()
    let navController = UINavigationController(rootViewController: journalVC)
    window?.rootViewController = navController
    window?.makeKeyAndVisible()
    return true
}
```

JournalTableViewController.swift фрагмент:

```
var tabBarCnt = UITabBarController()
override func viewDidLoad() {
    super.viewDidLoad()
  tabBarCnt = UITabBarController()
```

```
    tabBarCnt.tabBar.barStyle = .black
    let journalVC = JournalTableViewController()
    journalVC.tabBarItem = UITabBarItem(tabBarSystemItem: .favorites, tag: 0)
    tabBarCnt.viewControllers = [journalVC]
    self.view.addSubview(tabBarCnt.view)
}\
```

В нашому додатку буде всього 5 вкладок. Вкладка тренувань, ігор, пазлів, тестів, профайлу(рис.3.8.).

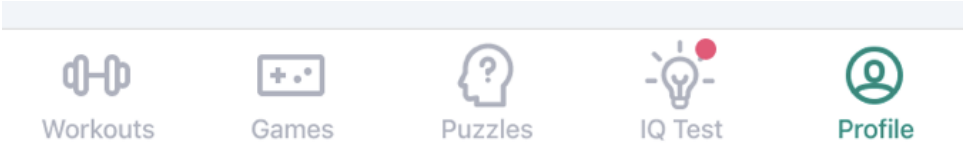


Рис.3.8. Tab bar застосунку

В звичайному стані іконки мають сірий колір, якщо вкладка має активний стан - змінює колір на зелений(рис.3.9.).

```
final class StatisticsTabBarController:
    TabBarController<StatisticsTabBarItemView, StatisticsTabBarView>,
    EventPoducer,
    ProfileBarButtonItemProtocol
{

    private var standardAppearanceShadowColor: UIColor = .clear
    private var standardAppearanceShadowImage: UIImage = .empty
    private var scrollEdgeAppearanceShadowColor: UIColor = .clear
    private var scrollEdgeAppearanceShadowImage: UIImage = .empty
    private var navigationShadowWasModified: Bool = false
```

Рис.3.9. Зміна стану іконок

3.3.2 Navigation bar

UINavigationBar – це перегляд. Як правило, його положенням управляє UINavigationController, але, як і інші view, його можна використовувати самостійно.

UINavigationItem - це клас, який описує стан (схожий на viewModel) для конфігурації UINavigationBar. Просто клас з властивостями, які будуть передані в UINavigationBar.

UINavigationBar містить масив [UINavigationItem]. За допомогою методів pushItem, popItem та setItems можна анімувати перехід одного стану UINavigationBar до іншого.

Кожен UIViewController містить UINavigationItem. UINavigationController сам управляє додаванням цієї властивості в стек item-ів UINavigationBar-а (рис.3.10.).

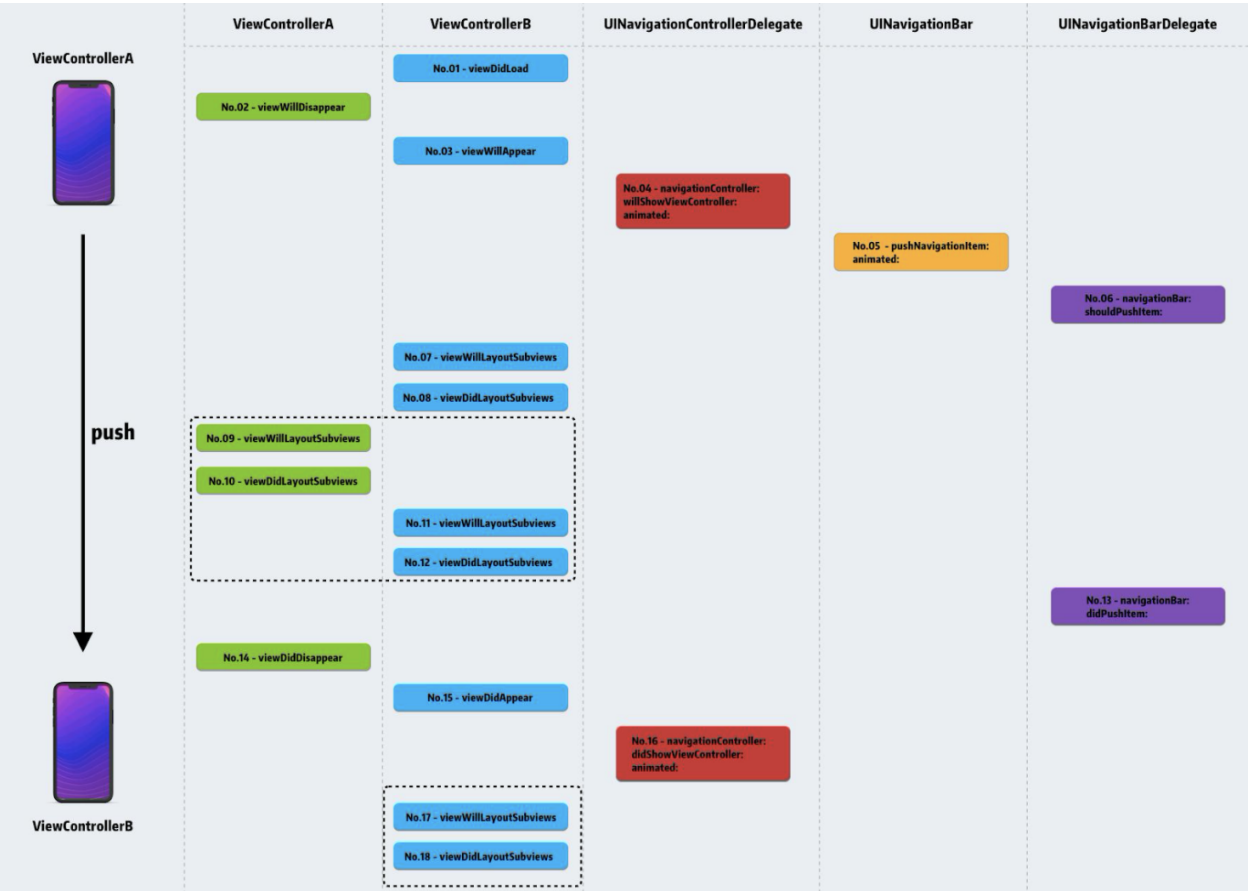


Рис.3.10. Схема повного життєвого циклу з UINavigationBar-ом.

Додавання контролера навігації

Виділено контролер, у верхньому меню переходимо до Editor-> Embed In і обрано опцію Navigation Controller. В результаті побачимо наступні зміни у сториборді(рис.3.11.)

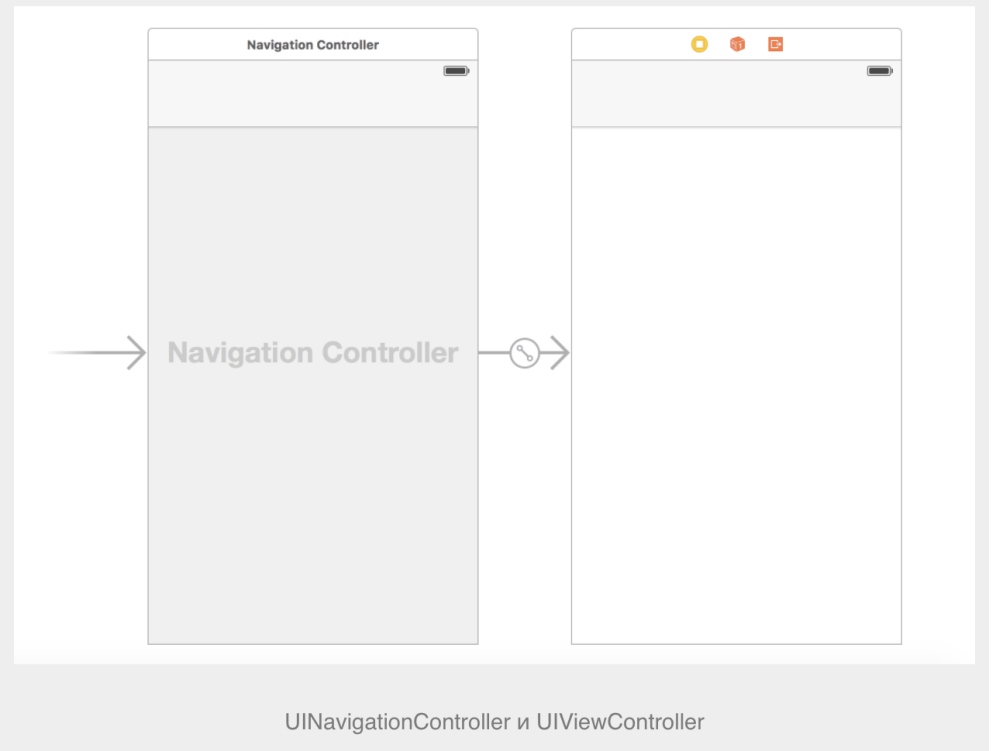
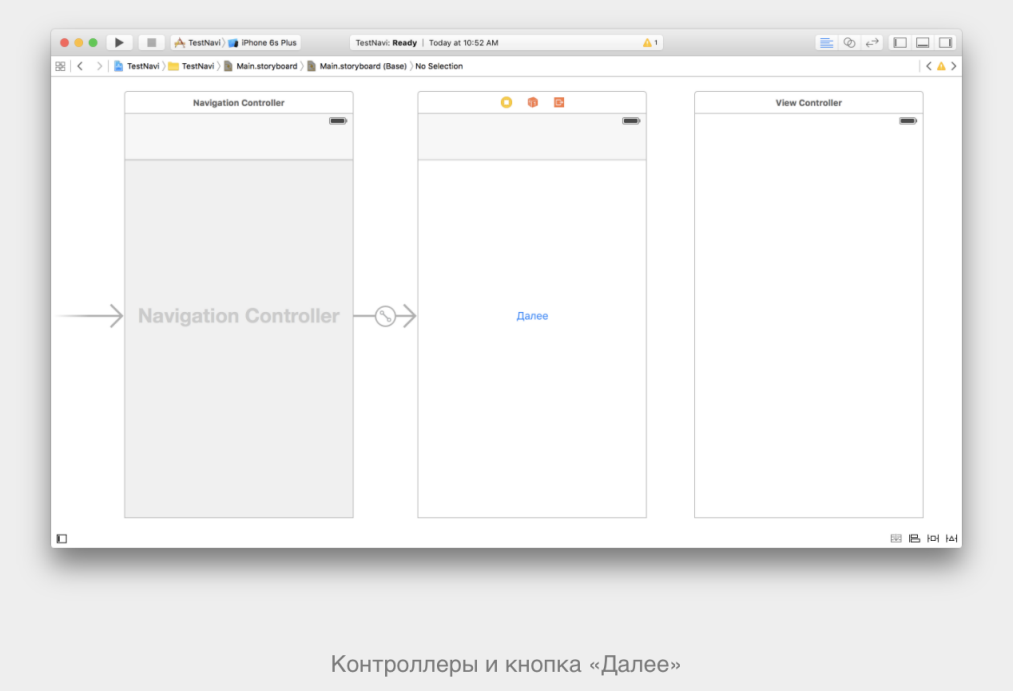


Рис.3.11. Контролер навігації

Такий тип зв’язку між контролерами в iOS називається segue. Окрім того, що segue візуально відображає зв’язки між сценами, він також служить для передачі даних між контролерами та налаштування переходів.

Навігація з UINavigationController

Тепер, коли підготували контролери, настав час перейти до самої навігації. Додайте інший UIViewController до розкадрування, перетягнувши його з бібліотеки Object. Ми також розмістимо кнопку (UIButton) на оригінальному контролері перегляду та дамо йому назву, наприклад, «Далі». В результаті у вас повинно вийти щось подібне(рис.3.12.)



Контроллеры и кнопка «Далее»

Рис. 3.12. UI Controller

Для того, щоб завершити перехід на контроллер представлення #2, нам не прийдеться написати ні одну строку коду. Достаточно додати «зв'язок» між кнопками та контроллером і в появі списку опцій вибрати Push.

Такі підключення можна зробити не тільки з кнопки. Часто використовуються клітинки таблиці або колекції і навіть прямі зв'язки між одним контролером та іншим.

Як можна помітити, контролер навігації автоматично додає панель навігації до кожного «вкладеного» контролера, а також автоматично додає кнопку «Назад», що дозволяє повернутися до попередньої сцени.

Кожній групі можна дати власну назву, яка допоможе нам не загубитися в численних сценах. Виділіть панель у першому контролері та в розділі Інспектор атрибутів розділу Утиліти вкажіть заголовок «Пуск». Встановіть другий контролер на «Середній». Після зміни заголовків Xcode автоматично відобразить їх у розкадровці. Також необіхдно вказати назву для кнопки Назад. Введіть «Назад» у полі «Назад» і натисніть Enter.

3.3. Створення ігор.

Додаємо фон нашої гри та спочатку екран буде виглядати так(рис.3.13.)

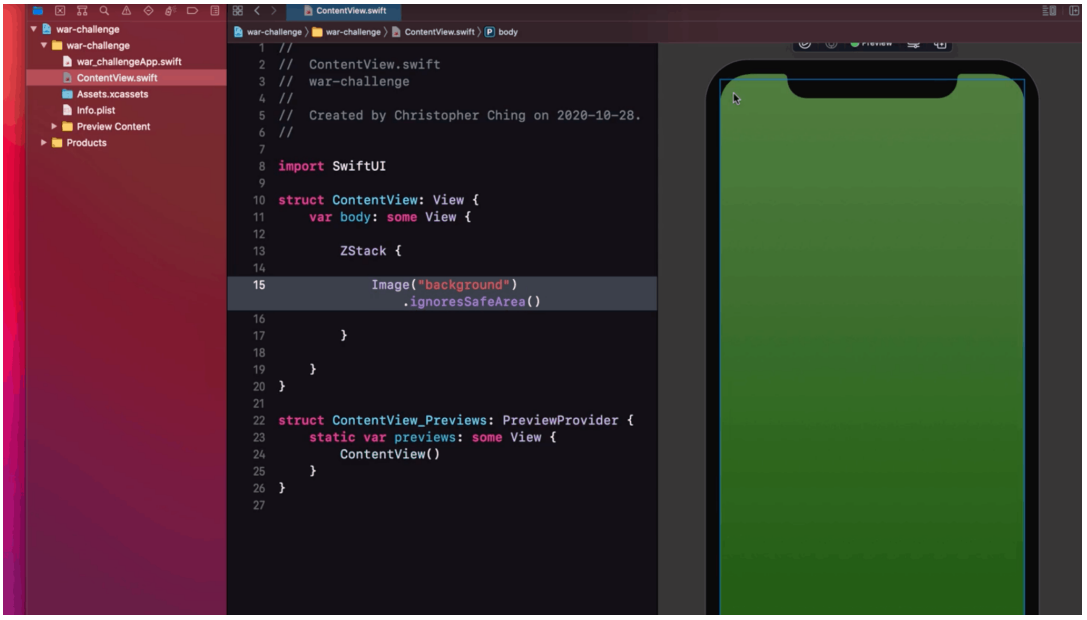


Рис. 3.13. Початковий стан еркану

3.3.1 Створення гріду з елементами

Основні поняття Grid Layout

Grid Layout — це двовимірна сітка. Сітка може використовуватися для розміщення основних областей сторінки або невеликих елементів інтерфейсу користувача.

Grid(сітка) — це набір горизонтальних і вертикальних ліній, що перетинаються, одна з яких визначає стовпці, а інша — рядки. Елементи можна розміщувати в сітці, відповідно рядках і стовпцях. Сітка виконує такі функції:

Фіксовані та гнучкі розміри смуг

Можемо створити сітку з фіксованими розмірами смуг - наприклад, за допомогою пікселів.Також можна створити сітку з гнучкими розмірами, використовуючи відсоток або нову одиницю виміру, призначену для цієї мети.

Місцезнаходження об'єкта

Можемо розміщувати елементи у визначеному місці сітки, використовуючи номери рядків, імена або прив’язуючи їх до області сітки. Сітка також містить алгоритм керування розміщенням елементів, які не мають явної позиції в сітці.

Створення додаткових смуг для зберігання контенту

Можемо визначити явну сітку з макетом сітки, але специфікація також застосовується до вмісту, доданого за межами оголошеної сітки, додаючи додаткові рядки та стовпці, якщо необхідно. Включаються такі функції, як додавання "стільки стовпців, скільки поміститься в контейнері".

Управління вирівнюванням

Сітка містить функції вирівнювання, щоб ми могли контролювати, як елементи вирівнюються після розміщення в області сітки, а також як вирівнюється вся сітка.

Управління вмістом, що перекривається

У клітинку сітки може бути розміщено більше одного елемента, або області можуть частково перекриватися. Потім цей пакет можна відстежувати за допомогою z-індексу.

В мові програмуваня SwiftUI є представлення LazyVGrid і LazyHGrid, які використовуємо для створення макетів із сітками елементів.

LazyVGrid і LazyHGrid — це два нових типи переглядів, які надаються для створення супер настроюваного макета на основі сітки елементів. Єдина відмінність між ними – вісь наповнення. LazyVGrid заповнює доступний простір у вертикальному напрямку. LazyHGrid розміщує свої дочірні елементи в горизонтальному напрямку. Це єдина різниця між ними.

Створено сітку з трьох столбців, кожен столбець якого має фіксований розмір у 40 пунктів.

Визначено стовпці в макеті сітки масивом який є параметром column. Щоб описати стовпець, використано тип GridItem. Вирівняно сітку за допомогою параметра alignment, перерахувавши HorizontalAlignment для LazyVGrid і VerticalAlignment для LazyHGrid. Працює так само, як і вирівнювання стека.

Параметр інтервалу визначає відстань між кожним рядком всередині LazyVGrid або проміжок між кожним стовпцем всередині LazyHGrid.

Визначено параметри фіксації за допомогою pinnedViews верхніх і нижніх колонтитулів розділів (колонтитулів). За замовчуванням він порожній, що означає, що верхні та нижні колонтитули поводяться як вміст і зникають під час прокручування. Вимикаємо прив’язки верхнього та нижнього колонтитулів, у цьому випадку вони перекриватимуть вміст і стають видимими назавжди.

struct GridItem

Використано екземпляри GridItem для налаштування макета елементів у представленнях LazyHGrid і LazyVGrid. Кожен елемент сітки визначає властивості макета, такі як відстань і вирівнювання, які використовується для перегляду сітки для розміру та розташування всіх елементів у заданому стовпці або рядку.

Найцікавіший варіант – адаптивний. Опція реагування дозволяє нам розміщувати кілька елементів у просторі одного гнучкого стовпця.

Наша гра розрахована на 19 рівнів складності, які діляться на різні розміри гридів від грида 2x3, до 8x8.

В залежності від того як підвищується рівень складності будуть різні гріди. Так виглядає після розробки стандартний грід(рис.3.14.).

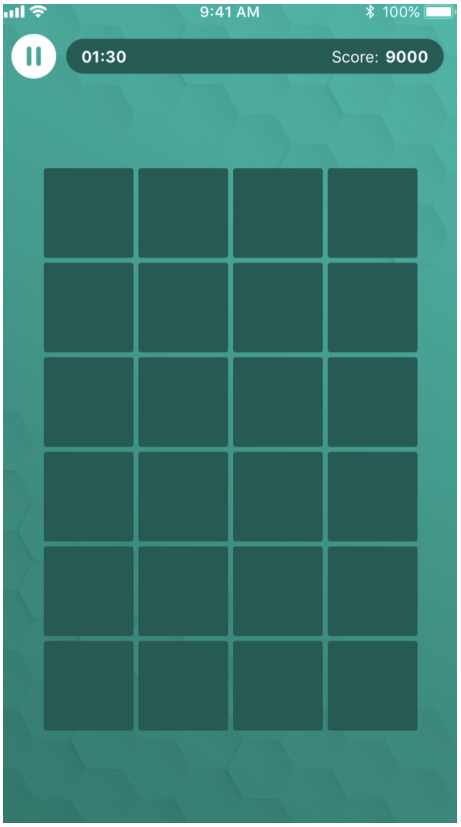


Рис. 3.14. Сітка на екрані з ігрою

В нас є 2 умовних гріда:

- перший який бачимо на рисунку - пустий грид елементів,
- другий - прихований грід з розташуванням елементів які необхідно знайти

По тапу на клітинку пустого гріда показується елемент який там прихован, далі тап на другу клітинку - відкривається другий елемент. Порівняно елементи - якщо вони однакові то зникають, якщо ні - елементи змінюють свій стан на попередній. І так поки всі клітинки не зникнуть - клітинки зникли, наступний стан змінює рівень, змінна раунду в 1й грі збільшується на 1цю, з'являються нові 2 умовних гріда. І так доки змінна з раундами не досягне 5ти. Тоді перехід екрану на екран результату.

3.3.2. Створення анімації перегортання елементів.

Розуміння Flipperooo

Додано модифікатор анімації, rotation3DEffect. Це не означає, що анімація буде насправді відбуваються. Є й інші компоненти успішної анімації, яких повинні дотримуватися, щоб переконатися, що вона діє так, як очікуєте.

Використано змінну @State, щоб переконатися, що подання повторно відтворюється.

Використано модифікатор ефекту для перегляду, який буде виконуватися, коли відбудеться анімація.

Використовуйте потрійний оператор, щоб змінити ефект на основі змінної стану. Викликано анімацію або за допомогою закриття withAnimation.

Маємо модифікатор ефекту, rotation3DEffect. Цей ефект дозволяє нам створити обертання в тривимірному просторі, наданому екраном. В рамках ефекту ми використовуємо змінну стану, щоб прийняти ефективне рішення на основі стану. Для тривимірного обертання ми повинні вказати кут повороту, який буде завершено, коли ефект буде в дії.

Оскільки ми хочемо перевертати картку вперед і назад (з одного боку на інший), використано потрійний оператор для перегортання на 180 градусів, коли вона перевертається, а потім на 0 (тому це, по суті, буде -180 градусів від 180- обертання в градусах) градусів, коли він не перевернутий. Нарешті, ми повинні вказати вісь, на якій ми хочемо обертати картку.

Координатний простір обертання(рис.3.15.)

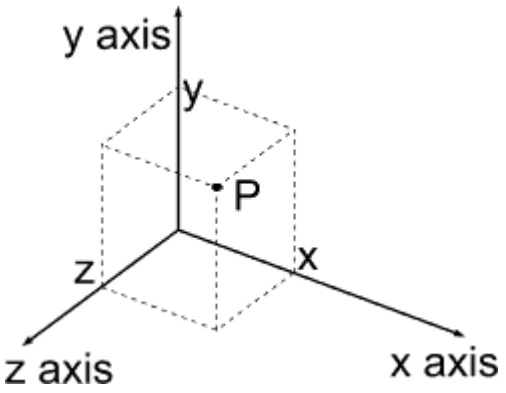


Рис.3.15. 3D координатна площина

Вісь y вказує на верхню частину iPhone, вісь x буде від кнопки гучності до кнопки блокування, а вісь z буде йти безпосередньо у телефон.

Отже, якщо обертаємо по осі x, карта буде виглядати так, ніби вона перевертається на Rolodex (або зверху вниз). Якщо обертання по осі z, то карта буде виглядати так, ніби її крутять, як парасольку. Обертання навколо осі Y виглядатиме так, ніби гортається сторінка книги (рух зліва направо).

Збираємося повернути нашу картку по осі Y для цього виду, щоб створити такий вигляд.

Для цього встановили значення X і Z на 0. Робимо це, оскільки хочемо, щоб обертання відбувалося виключно по осі Y. Отже, встановивши іншу вісь на 0, ми матимемо лише обертання над Y, що дасть нам ефект перевертання карти.

Виклик анімації

Тепер, коли встановили наш погляд і ефект. Нам потрібно вказати представленню виконувати анімацію, коли стан змінюється. Ми можемо зробити це двома способами: ми можемо явно вказати подання про повторне відтворення з анімацією, коли ми оновлюємо змінну стану за допомогою закриття withAnimation, або ми можемо використовувати модифікатор анімації.

Через те що нам потрібно це робити по тапу - використано модифікатор на дію. Далі додано таймаут, аналізується співвідношення елементів і

викликається закриття withAnimation якщо потрібно ще раз повернути елементи.

Для того щоб з’являлись елементи на гріду використано Index Delay анімацію(рис.3.16.).

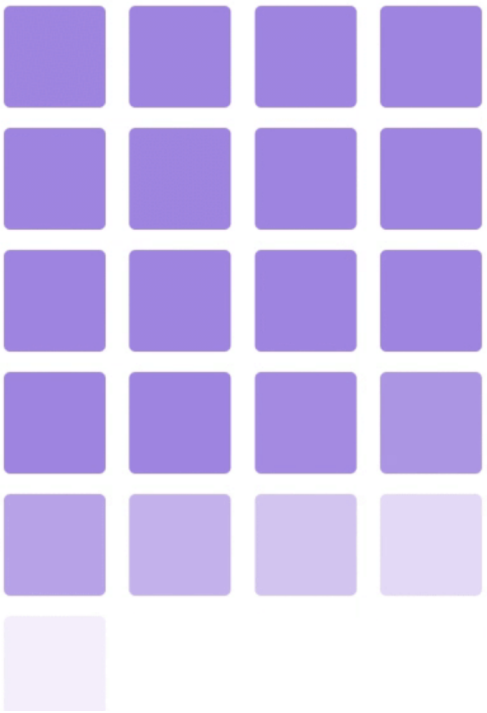


Рис. 3.16. Грід в процесі появи

3.3.3. Таймер

Якщо об’єднаємо Foundation, SwiftUI та Combine, ми можемо додати таймер до нашого додатка, щоб трохи напружувати користувача. Проста реалізація цього не вимагає багато роботи, але вона також має помилку, яка вимагає додаткової роботи для виправлення.

Для нашого першого проходу таймера створено дві нові властивості: сам таймер, який запускатиметься раз на секунду, і властивість timeRemaining, від якої ми будемо віднімати 1 щоразу, коли таймер запускається. Це дозволить нам показати, скільки секунд залишилося в поточному запуску програми, що повинно дати користувачеві м’який стимул до прискорення.

Додано дві нових властивостей до ContentView:

```
@State private var timeRemaining = 100
let timer = Timer.publish(every: 1, on: .main, in: .common).autoconnect()
```

Це дає користувачеві 100 секунд для початку, а потім створює та запускає таймер, який запускається раз на секунду в основному потоці.

Щоразу, коли спрацьовує цей таймер, ми хочемо відняти 1 від timeRemaining, щоб він почав зворотній відлік.

Додано цей модифікатор onReceive() до крайнього ZStack у ContentView:

```
.onReceive(timer) { time in
    if self.timeRemaining > 0 {
        self.timeRemaining -= 1
    }
}
```

Цей код запускає наш таймер зі 100 і змушує його відлік до 0, але нам потрібно фактично його відобразити. Додано його на панель статусу.

Виводимо значення 100 в окремий параметр, де для кожного рівня складності буде різне значення, та в залежності від того який рівень користувач обирає буде запускатись різний таймер.

Коли таймер закінчується закривається екран і відкривається екран із результатами гри.

Грід і таймер використовується в кожній грі, тому можемо завжди посилатися на те що маємо на даному етапі.

До кожного елемента прив’яжемо відповідні картинки, щоб грід повноцінно виглядав ось так(рис.3.17.)

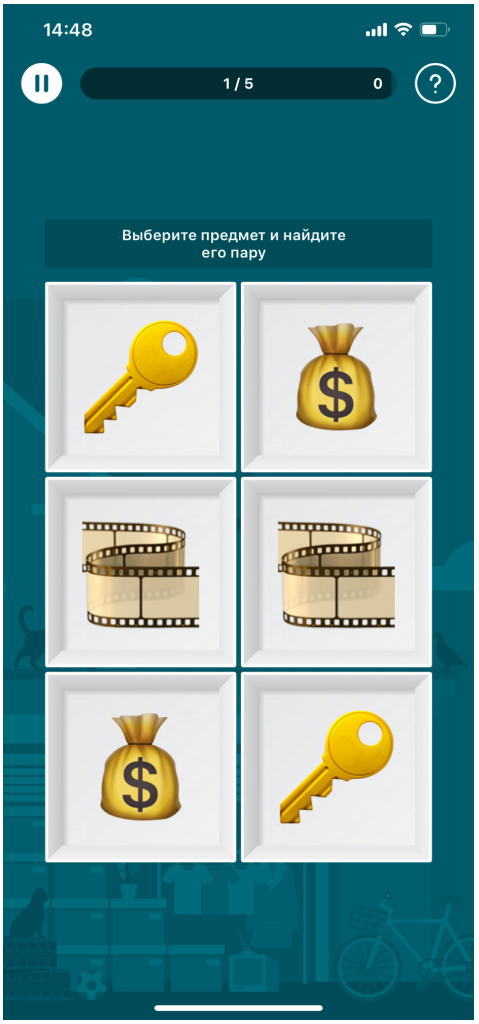


Рис.3.17. Всі елементи на гриді

Score.

Створено змінну яка буде зберігати дані, кожна правильна відповідь - має своє значення для додавання - використано готову таблицю даних яку підготували і витягнуто з неї інформацію. Набраний скор також виводиться в статус і потім відображається на екрані результатів.

3.4. Екран результатів

На екрані скору виводиться елементи(рис.3.18.):

- картинка стану - або кубок або зелений кружок
- заповнюється шкала скору
- відображення рівня складності
- та відображаються дані за грою які збираються протягом гри
- кнопки продовжити або вийти

Стани:

- завершив за вказаний час - кубок
- закінчився час - зелений кружок
- набран максимальний скор - анімація і жовтий фон
- не набран максимальний скор - немає анімації та білий текст
- ранг - з таблиці співвідношень
- підвищив рівень складності - анімація
- не підвищив стан не змінюється

Створено клас, що містить усі функції та стани(додавання результатів, аналіз станів. Це викликається з головного перегляду ContentView. У цій частині ми встановили мітку стану на заголовок типу шрифту (з використанням шрифту), ми також змінили колір тексту (використовуючи foregroundColor) на білий. Також додано невеликий відступ внизу, щоб додати місце для мітки партитури.

Для партитури ми встановили мітку типу шрифту (за допомогою шрифту) largeTitle, ми також змінили колір тексту (за допомогою foregroundColor) на білий або жовтий. Для виведення скору : використовуючи ту ж логіку. Однак можете помітити, що ми не можемо просто встановити значення playerScore для тексту. Це тому, що існує невідповідність типів даних. Текст зазвичай запитує рядок, але наша змінна playerScore має тип Int. Найпростіший спосіб обійти це - просто ввести значення, виконавши String(value).

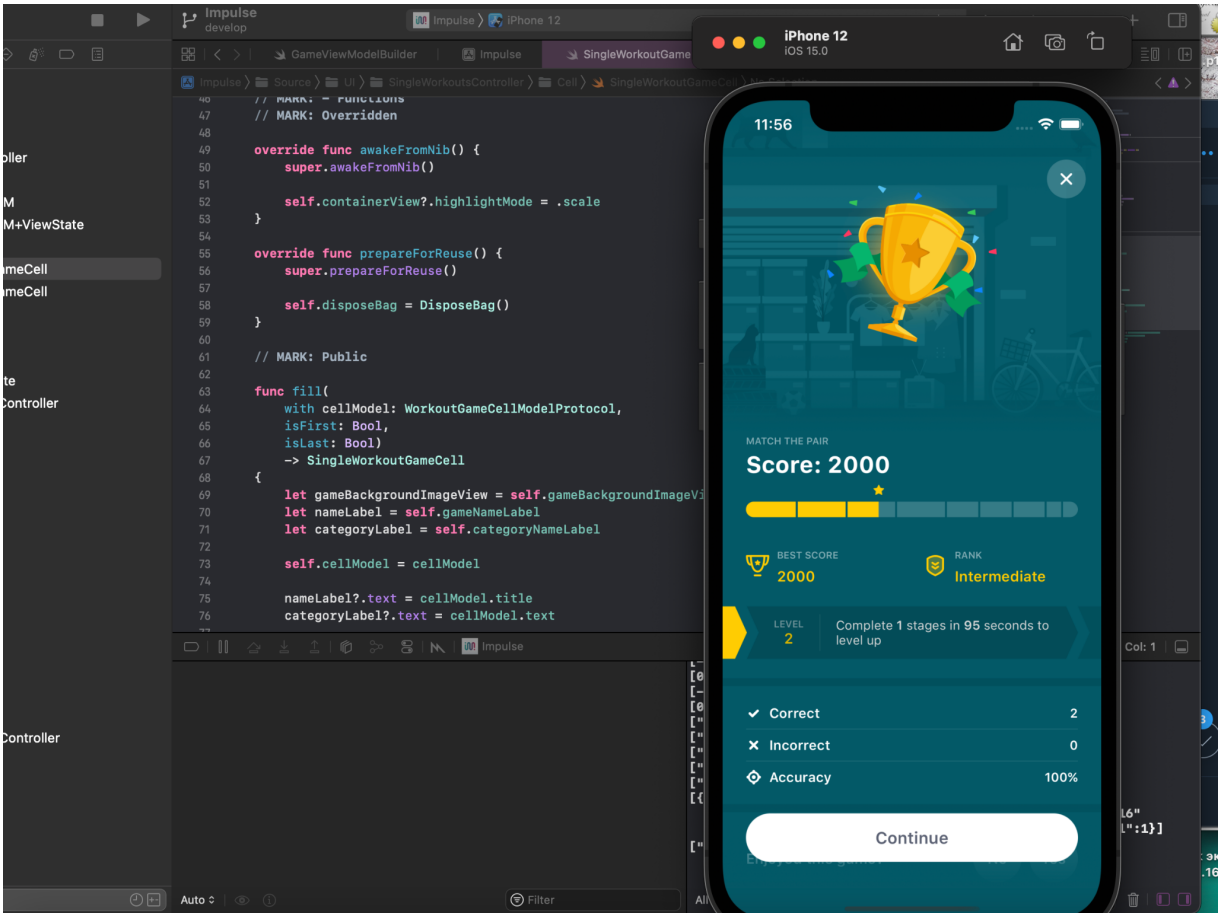


Рис. 3.18. Фінальний вигляд екрану результатів

3.5. Інтеграція

3.5.1. Вкладка ігор

Оформляємо вкладку ігор таку як за прототипом.

У GfmeTab.swift додано перелік категорії та властивість категорії до структури Landmark.

Файл gameTabData.json вже містить значення категорії для кожного орієнтира з одним із трьох рядкових значень. Збігаючи імена у файлі даних, можемо покладатися на відповідність структури Codable для завантаження даних.

У ModelData.swift додано обчислений словник категорій з назвами категорій як ключами та масивом пов’язаних орієнтирів для кожного ключа.

У CategoryHome.swift створіть об’єкт середовища modelData.

Знадобиться доступ до категорій прямо зараз, а також до інших даних про орієнтири пізніше. Відображення категорій у Орієнтирах за допомогою списку.

Назва випадку Landmark.Category ідентифікує кожен елемент у списку, який має бути унікальним серед інших категорій, оскільки це перерахування. Створіть стовпець категорій.

Орієнтири відображають кожну категорію в стовпці, що прокручується по вертикалі. Додано новий тип перегляду для представлення стовпця, а потім відображано усі орієнтири для цієї категорії в новому поданні. Повторно використано частини перегляду орієнтирів, щоб створити знайомий попередній перегляд орієнтира. Визначено новий користувацький вигляд CategoryRow для зберігання вмісту стовпця. Додано властивості для назви категорії та списку елементів у цій категорії. Відображено назву категорії.

Помістимо елементи категорії в VStack і згрупуйте їх з назвою категорії в HStack. Дамо вмісту трохи простору, вказавши високий кадр (ширина:висота:), додавши відступи та обгорнувши HStack у режимі прокручування.

Оновлення попереднього перегляду перегляду з більшою вибіркою даних полегшує перевірку правильності прокручування. Створено нове користувацьке представлення під назвою CategoryItem, яке відображає один орієнтир.

У CategoryRow.swift замінено текст, який містить назву орієнтира, новим поданням CategoryItem.

Завершимо перегляд категорій. Додано рядки та запропоноване зображення на домашню сторінку ігор.

Оновлено тіло CategoryHome, щоб передати інформацію про ігри екземплярам типу рядка.

Далі додано пропонований орієнтир у верхній частині перегляду. Для цього знадобиться додаткова інформація з даних орієнтирів.

У GameTab.swift додайте нову властивість isFeatured. Як і для інших властивостей орієнтирів, які додали, це логічне значення вже існує в даних — просто оголошено нову властивість. У ModelData.swift додано новий масив обчислюваних функцій, який містить лише орієнтири, для яких isFeatured встановлено значення true.

У CategoryHome.swift додано зображення першого пропонованого орієнтира вгору списку. Далі перетворено це представлення на інтерактивну карусель. Наразі він відображає одну з пропонованих орієнтирів із масштабованим та обрізаним зображенням попереднього перегляду.

Встановлено вставки країв на нуль для обох видів попереднього перегляду орієнтирів, щоб вміст міг поширюватися до країв дисплея.

Для виведення на основну вкладку нашої гри треба створити керування сторінками.

Оскільки всі орієнтири, класифіковані за різними категоріями та іграми, відображаються в поданні, користувачеві потрібен спосіб дістатися до кожного розділу програми. Використовуйте API навігації та презентації, щоб зробити категорію домашньою, детальним переглядом і списком вибраного доступними з перегляду вкладок.

У CategoryRow.swift обернемо наявний CategoryItem посиланням NavigationLink.

Сам елемент категорії є міткою для кнопки, а його призначення — детальним переглядом орієнтира для орієнтира, представленого карткою. Закріплено попередній перегляд, щоб ми могли побачити вплив наступного кроку на CategoryRow.

Змінемо вигляд навігації елементів категорії, застосувавши модифікатори renderingMode(_:) і foregroundColor(_:).

Текст, який передано як мітку для навігаційного посилання, відображається з використанням акцентного кольору середовища, а

зображення можуть відображатися як зображення шаблону. Можемо змінити будь-яку поведінку, щоб якнайкраще відповідати нашому дизайну.

Далі змінено перегляд основного вмісту програми, щоб показати вкладку, яка дозволяє користувачеві вибирати між щойно створеним переглядом категорії та наявним списком орієнтирів. Відкріплено попередній перегляд, і до ContentView додано перелік вкладок для відображення.

Додано змінну стану для вибору вкладки та надано їй значення за замовчуванням. Створено подання вкладок, яке охоплює список LandmarkList, а також нову CategoryHome.

Модифікатор tag(_:) у кожному з представлень відповідає одному з можливих значень, які може прийняти властивість вибору, щоб TabView міг координувати, яке подання відображати, коли користувач робить вибір в інтерфейсі користувача. Дано кожній вкладці мітку.

Додано спеціальний елемент керування сторінкою. Створено власний UIPageControl до свого представлення, загорнутого в подання SwiftUI UIViewRepresentable.

Створенр новий файл перегляду SwiftUI під назвою PageControl.swift. Оновіть тип PageControl, щоб він відповідав протоколу UIViewRepresentable.

Типи UIViewRepresentable і UIViewControllerRepresentable мають однаковий життєвий цикл з методами, які відповідають їхнім основним типам UIKit.

Замініть текстове поле елементом керування сторінкою, перейшовши з VStack на ZStack для макета.

Оскільки передано кількість сторінок і прив’язку до поточної сторінки, елемент керування сторінкою вже показує правильні значення.

Далі зробимо елемент керування сторінкою інтерактивним, щоб користувачі могли торкатися однієї або іншої сторони для переходу між сторінками.

Створено вкладений тип координатора в PageControl і додайте метод makeCoordinator(), щоб створити та повернути новий координатор.

Оскільки підкласи UIControl, такі як UIPageControl, використовують шаблон цільової дії замість делегування, цей Координатор реалізує метод @objc для оновлення поточної прив'язки сторінки.

Додано координатор як ціль для події valueChanged, вказавши метод updateCurrentPage(sender:) як дію, яку потрібно виконати.

Нарешті, у CategoryHome замінено зображення заповнювача на новий перегляд сторінки.

Тепер спробуємо всі різні взаємодії — PageView показує, як подання та контролери UIKit і SwiftUI можуть працювати разом.

Фіналізовано та запущено продукт, повний вигляд вкладок для різних користувачів(рис.3.19.-3.20.).

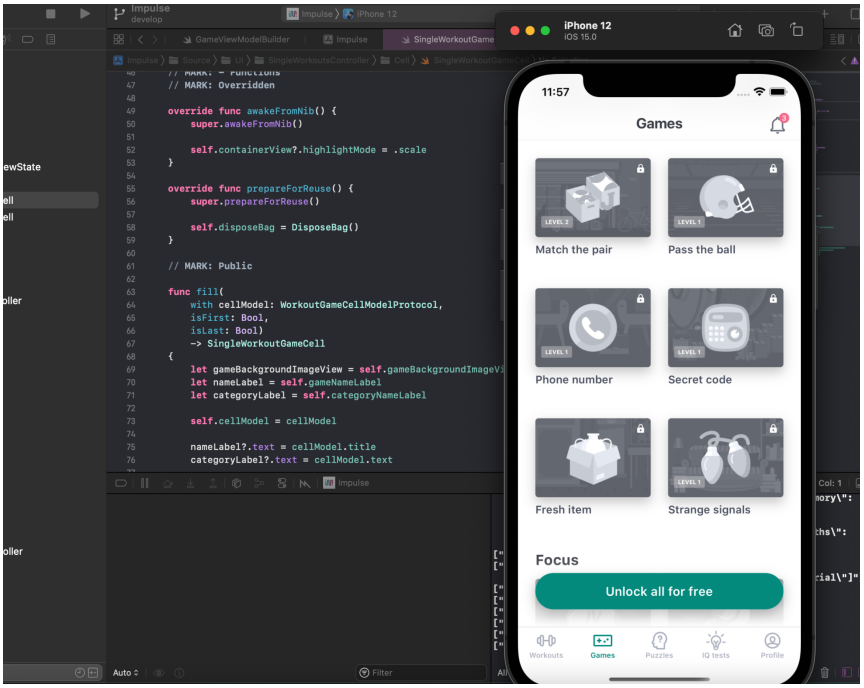


Рис. 3.19. Стан для не преміум користувачів

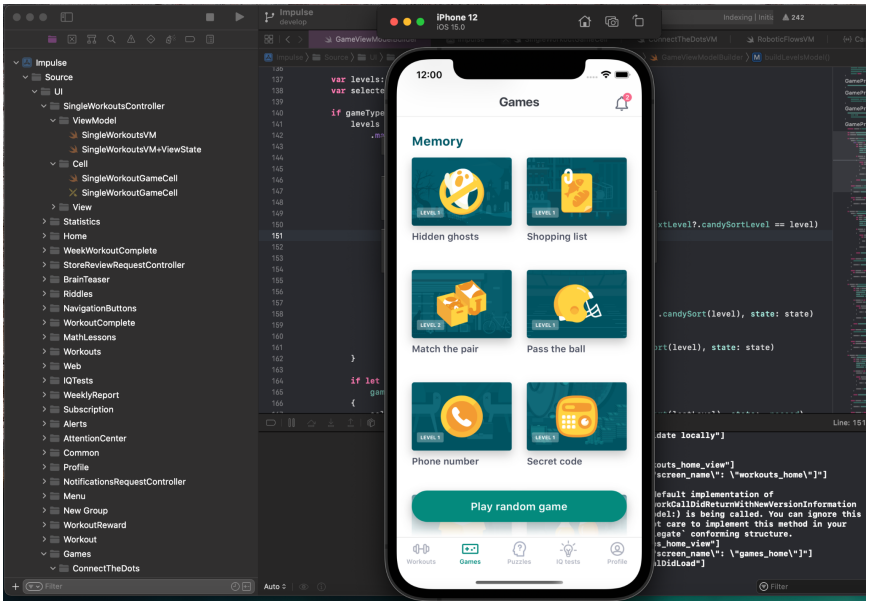


Рис. 3.20. Стан для преміум користувачів

За аналогією створено решта вкладок. IqTestViewModel.Swift, IqTestReportViewModel.Swift, RiddleCoordinator.Swift.

3.6. Створення підписок

Як відомо, Apple анонсувала свою нову декларативну структуру SwiftUI. В додатку реалізувано екрани покупок за допомогою SwiftUI та обробляється статус підписки, яка автоматично поновлюється.

Нам потрібен помічник StoreKit для використання в нашому проекті. На нашому головному екрані ми покажемо список усіх наших підписок. Створено одиночний клас ProductsStore, який ініціалізує наші продукти. Потім створено в ContentView спільний екземпляр передається як параметр.

В @Published var anyString = "123" додано маленьку хитрість, щоб примусово перезавантажити ContentView з PurchaseView, просто змінивши будь-яке значення Published

Клас ProductsStore. Це невеликий клас, який має одну функцію: завантажує об’єкти SKProduct і передає їх у представлення SwiftUI за допомогою спостереження за об’єктами. ProductsStore відповідає ObservableObject.

ObservableObject це спеціальний протокол для відстеження змін його властивостей (позначених ключовим словом @Published) і відображення цих змін у представленнях SwiftUI. Єдина вимога протоколу ObservableObject — мати властивість з ключовим словом @Published, яке зміниться через деякий час. Розглянуто його як центр сповіщень для переглядів SwiftUI. У нашому прикладі, коли змінюється масив продуктів, сповіщення буде надіслано всім представленням, які очікують змін. Щоб прослухати зміни, слід додати властивість до свого представлення за допомогою спеціального ключового слова @ObservedObject.

@ObservedObject вар. productsStore : ProductsStore

Цей рядок коду слід додано до нашого ContentView, щоб прослухати зміни в ProductsStore. ContentView буде автоматично перезавантажено, коли ми отримаємо продукти від StoreKit.

У нашому додатку продукти зберігаються в ProductsStore. Але ми вже маємо продукти в IAPManager. Хоча дублювання одного масиву не є хорошим підходом, обрано цей підхід для спостереження за об'єктами. Крім того, не завжди можливо змінити існуючі бібліотеки StoreKit, тому що ми використовуємо Cocoapods.

Варто зазначити, що окрім техніки спостереження за об'єктами, є також простіше ключове слово @State для спостереження за основними типами (Int, String) і більш глобальний @EnvironmentObject, що дозволяє оновлювати всі уявлення одночасно без необхідності зберігати властивість.

Відображено статуси підписок для кожного продукту. Використовуючи ForEach, створено текстові подання та призначано рядок із розширення. Оскільки ми спостерігаємо за властивістю productsStore, наше представлення буде перезавантажуватися щоразу, коли змінюється масив продуктів.

subscriptionStatus — це проста функція розширення, яка повертає стан підписки. Тепер перейдемо до екрана покупки. Як відомо, на екрані покупки ми повинні включити довгий текст про підписку. Тому тут краще використовувати ScrollView.

Код для PurchaseView. Усередині ScrollView створено два TextView. Далі є ряд функцій. Розділено кожен вид на одну функцію. Це зроблено з 3 причин:

Щоб показати, що можна розділити код на функції всередині блоку ViewBuilder.

Щоб зробити код більш читабельним і зрозумілим.

На момент написання Xcode 11 Beta 2 зависав і не міг скомпілювати цей складний блок ViewBuilder. Розбиття коду допомогло компілятору.

Створено представлення HStack і за допомогою циклу ForEach додано в нього спеціальні представлення PurchaseButton.

Це проста кнопка, яка зберігає блок завершення і викликає його в блоці дії. І якийсь стиль застосовується. Назва кнопки повертається з функції розширення SKProduct.

Після виклику успішного блоку платежу викликається метод handleUpdateStore ProductsStore. Це дозволяє нашому кореневому ContentView примусово перезавантажувати. Коли PurchaseView закриває, root view оновлює їхні статуси підписок.

Реалізація модального звільнення. SwiftUI — це декларативна структура, і не можна просто відхилити представлення, коли захочете. Але існує спосіб відхилення, викликавши метод dismiss() обгортки властивості presentationMode середовища.

Синтаксис досить складний, але це означає, що ми пов’язано одне зі значень середовища з нашою властивістю. Властивість presentationMode є частиною значень середовища, спеціального набору глобальних функцій і властивостей. Немає можливості виконувати жодних дій під час виконання: все прив’язано на початку. Ось так виглядає PurchaseView наш основний екран з підписками в додатку(рис.3.21.).

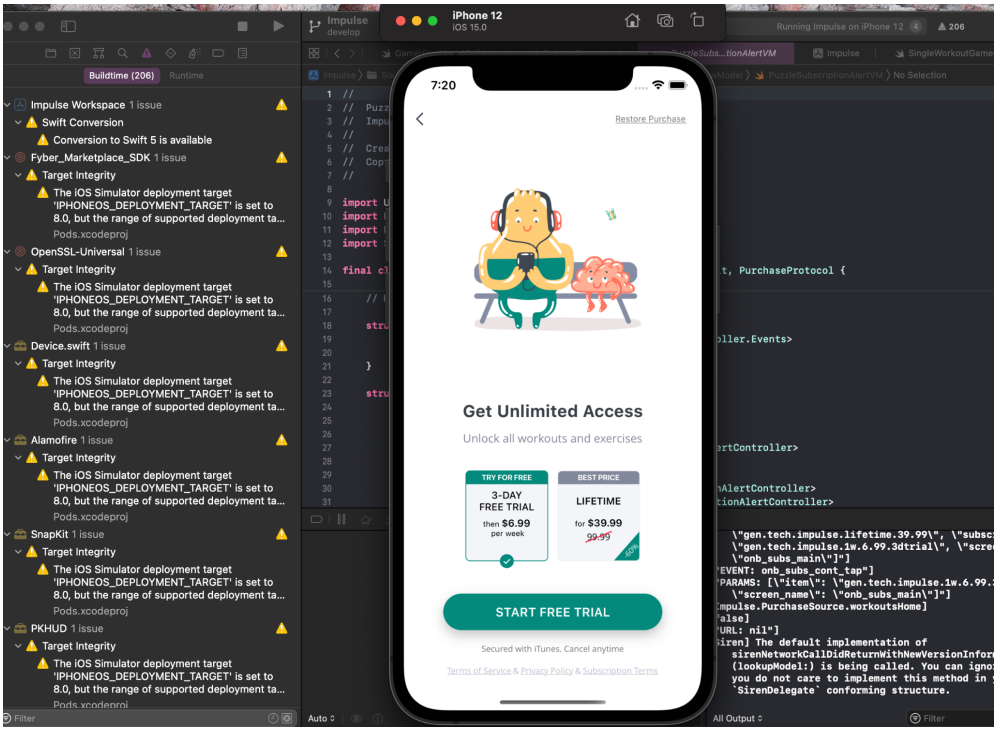


Рис.3.21. Основна підписка в додатку

Таким чином ми створено різні види підписок та екрани, навігацію де будуть з’являтись та за якими умовами.

Приклад реалізації основного екрану в додатку річної підписки(3.22.).

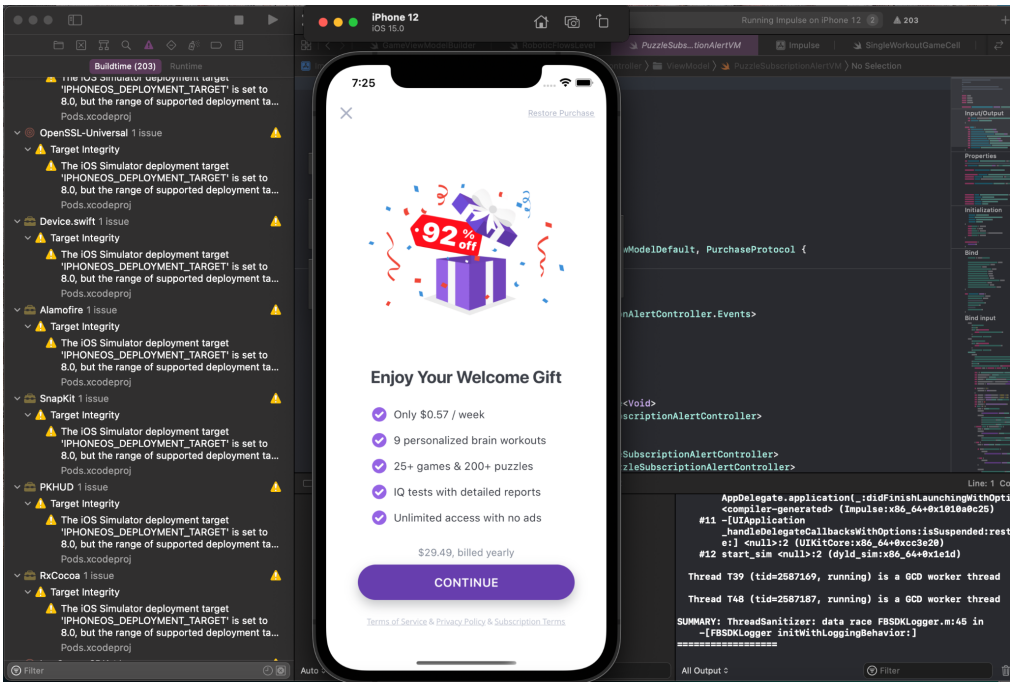


Рис.3.22. Річна підписка в додатку

3.7. Тестування мобільного додатку

Тестування це невід’ємна частина розробки застосунку, спочатку для того щоб пересвідчитись в тому, що кожна функція працює корректно було створено Unit Tests, це тести які прості, але покривають перевірками код застосунку для того щоб уникнути помилок. Одиницею може бути майже все— рядок коду, метод або клас. Але, як правило, менше, тим краще. Менші тести дають набагато більш детальне уявлення про ефективність коду. Існує також практичний аспект: коли тестуються дуже маленькі одиниці, тести можна виконувати швидко; як тисяча тестів за секунду.

В допомогу Unit тестам було руками протестовано додаток, виявлено та виправлено багато помилок. Розберемо декілька з них.

3.8. Створення чек листу до iq тестів

Для того щоб повноцінно протестувати додаток по перше маючи вимоги до того як буде працювати наш функціонал створено чек-ліст перевірок(табл.3.1.- 3.3.), те на що треба приділити увагу та переконатись в правильності обробки даних та відпрацьовуванні функцій як ми того очікуємо.

Є декілька видів тестування які використано:

- UI - пересвідчитись що наш застосунок відповідає дизайну
- Функціональне - пересвідчитись що функціонал застосунку відповідає до вимог

Таблиця тестування:

Maket ui	**check**
Анимация показа экрана	показ екрану - виїзд справа по тапу на картку тесту. Закриття - аналогічне по свайпу і тапу на кнопку назад
Анимация появления задания	виїзд справа та заїзд вліво -як слайди
Анимация окончания теста	зникають елементи та перехід на екран результатів

Табл.3.1 Перевірка анімацій та інтерфейсу

Открытие/закр. экрана паузы по тапу на X	анімація показу екрану паузи і виходу з нього - просто показ екрану як зараз в іграх зроблено;
Сам екран паузи з 2ма кнопками: Continue & Leave test /	
Continue button	Игра продолжается, время в которое пользователь был на экране паузы не засчитывается в прохождении теста
Leave button	у випадку натискання на Leave test показуємо додатковий дефолтний поп-ап з попередженням про втрату прогресу -
Выход с экрана результатов	на X & Done - закриваємо екран;
Выход с экрана результатов	на Show details - показуємо дефолтний поп-ап з однією кнопкою ОК.

Табл. 3.2. Перевірка взаємодій

Пройти всі три тести на відповідність відповідей
Перевірити відповідності результатів
Повторне проходження тесту
Вийшов час

Табл. 3.3. Перевірка функціоналу

3.9. Виявлення помилок

Помилка №1:

При запуску тесту креш додатку.

Помилка №2:

Порушено порядок анімацій на початку тесту

- Актуальна черговість появи елементів:

  Напис → завдання → відповіді
- Очікувана черговість появи елементів:

  Завдання → напис → відповіді

Помилка №2:

При запуску тесту креш додатку.

3.10. Виправлення помилок.

Помилка №1 вирішення:

Через те що були неправильно загружені картинки, відрізнялась назва - додаток не міг знайти елемента до якого ми звертаємось, а ми вказали що він там обов’язково є. Тому додаток не впорався з пошуком та перестав працювати. Оновлено назви та пересвідчено що все працює.

Помилка №2 вирішення:

Винесли окремо анімацію показу напису, завдання і відповіді щоб окремо їх запускатив правильному порядку та тривалість анімації

```
enum DescriptionLabel {
```

```
        static let duration: TimeInterval = 0.2
    } Додано функцію для появи кожного елементу щоб потім додати в потік анімацій
  private func animateDescriptionLabel(_ completion: @escaping () -> Void) {
        guard let scene = scene else {
            completion()
            return }
        let c = C.DescriptionLabel.self
        let animations: () -> Void = {
            scene.descriptionLabel.alpha = 1 }
        let completion: (Bool) -> Void = { _ in
            completion() }
        UIView.animate(withDuration: c.duration,
                       delay: 0,
                       options: [.curveEaseOut],
                       animations: animations,
                       completion: completion)
```

Ось так виглядає виклик додання анімацій:

```
        addAnimation(animateHeaderAndFooter)
            .addAnimation(animateQuestionView)
            .addAnimation(animateDescriptionLabel)
            .addAnimation(animateAnswersView)
            .start { [scene] in
                scene?.isUserInteractionEnabled = true
```

ВИСНОВОК

В ході виконання дипломної роботи було розроблено додаток для тренувань мозку на мові SwiftUI.

Проаналізувавши різні мобільні операційні системи - була обрана система IOS, перевагою обранной системи було централізованість платформи. Користувач відчуває себе в єдиній екосистемі, що безперечно є дуже зручним рішенням. Смартфони на базі iOS підтримуються компанією Apple, тож користувачі iOS-смартфонів можуть не турбуватись про помилки та неточності в роботі з системою, бо їх виправлять в наступній версії оновлень з великою ймовірністю. Простота в розумінні, єдина система, проста та приваблива візуально.

Проаналізувавши мови програмування було обрано SwiftUI.

SwiftUI - це фантастичний спосіб писати програми для телефонів, для десктопних комп'ютерів, серверів, та й чогось ще, що запускає та працює за допомогою коду. SwiftUI - безпечна, швидка та інтерактивна мова програмування. SwifUI увібрав у себе найкращі ідеї сучасних мов із мудрістю інженерної культури Apple. Компілятор оптимізований для продуктивності, а мова оптимізована для розробки без компромісів з одного або іншого боку.

SwiftUI дружелюбний по відношенню до новачків у програмуванні. Це перша мова програмування промислової якості, яка також зрозуміла і цікава, як скриптова мова. Написання коду в пісочниці дозволяє експериментувати з кодом Swift та бачити результат миттєво, без необхідності компілювати та запускати програму.

Для інтегрування баз даних було обрано Firebase - це кросплатформна платформа мобільних баз даних у реальному часі, яка дозволяє програмісту зосередитися на тому, що вони найкраще пишуть у своїх програмах, не турбуючись про проблеми DevOps, як-от інфраструктура сервера та моделювання баз даних. Firebase, що підтримується Google, бере на себе

складність роботи з базами даних у реальному часі, аутентифікації користувачів та роботи з робочими процесами офлайн-синхронізації. Проста в налаштуванні та дуже зрозуміла і зручна в використанні. За допомогою firebase крім обробки даних можна в майбутньому використовувати remote config для налаштування a/b тестів в додатку - це дуже оптимальна річ для бізнесу.

Аналізом сфер діяльності застосунків обрано галузь “Освіта”. В ігровій формі було подано тренування для розвитку пам’яті, уважності, математики, когнітивних функцій. Користувач може дивитись на свій прогрес проходячи IQ тестування. Користувач може розширити зони розвитку купивши підписку в додатку, або заходити кожного дня в щоденні тренування де відкриваються 3 безкоштовні ігри.

Цікаві пазли можут занурити в пошук відповідей мислячи нестандартно, змусити працювати логіку та показати що іноді прості рішення можуть бути правильними.

При розробці найскладнішим було - інтегрувати зроблену механіку та розробити навігацію між всіма екранами які є в застосунку, та обробляти всі стани з яких користувач може опинитись на конкретному екрані. Створення підписок також було складним, тому що зараз немає apple аккаунту який мав би надавати інформацію щодо покупок, тому додано було локальну перевірку.

Тестування виявило більше 50ти багів, в розділі 4 надано лише декілька прикладів в одній області, завдяки тестуванню фінальний додаток працює як ми того і очікували. В дизайн документі вже створено нові ігри і інші екрани для розвитку нашого застосунку, тому в подальшому функціонал додатку буде збільшуватись і зростати.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. PYPL (Popularity of Programming Language Index) [Електронний ресурс] – 2019. – Режим доступу до ресурсу: http://pypl.github.io/PYPL.html
2. A brief post about what Firebase is all about, and it's new NoSQL Database Cloud Firestore [Електронний ресурс] – 2017. – Режим доступу до ресурсу: https://hackernoon.com/introduction-to-firebase-218a23186cd7
3. MapKit Framework [Електронний ресурс] – 2019. – Режим доступу до ресурсу: https://developer.apple.com/documentation/mapkit
4. MapKit Framework [Електронний ресурс] – 2019. – Режим доступу до ресурсу: https://developer.apple.com/documentation/corelocation 5. Firebase [Електронний ресурс] – 2019. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Firebase
5. iOS против Android. Что всё-таки лучше? [Електронний ресурс] – 2017. – Режим доступу до ресурсу: https://www.iphones.ru/iNotes/759337
6. A Swift tour [Електронний ресурс] – 2019. – Режим доступу до ресурсу: https://docs.swift.org/swift-book/GuidedTour/GuidedTour.html
7. Firebase Realtime database [Електронний ресурс] – 2019. – Режим доступу до ресурсу: https://firebase.google.com/docs/database
8. Apple Xcode IDE от Apple [Електронний ресурс] – 2019. – Режим доступу до ресурсу: http://wnfx.ru/apple-xcode-ide-ot-apple
9. Основы использования Cocoapods в разработке приложений под iOS [Електронний ресурс] – 2015. – Режим доступу до ресурсу: https://habr.com/ru/post/261711
10. Sketch Tutorial for iOS Developers [Електронний ресурс] – 2016. – Режим доступу до ресурсу: https://www.raywenderlich.com/1379-sketch-tutorialfor-ios-developers
11. Основы Auto Layout — Концепция, строение, применение [Електронний ресурс] – 2016. – Режим доступу до ресурсу: https://habr.com/ru/post/312782

12. Математические основы Auto Layout [Електронний ресурс] – 2019. – Режим доступу до ресурсу: https://habr.com/ru/company/oleg-bunin/blog/437584

13. NoSQL базы данных: понимаем суть [Електронний ресурс] – 2012. – Режим доступу до ресурсу: https://habr.com/ru/post/152477

14. About an Xcode IDE for Apple application development [Електронний ресурс] – 2018. – Режим доступу до ресурсу: https://medium.com/worst-iosdeveloper/about-an-xcode-ide-for-apple-application-development-68b161088e53

15. Google Maps SDK for iOS [Електронний ресурс]. - https://developers.google.com/maps/documentation/ios/?hl=ru

16. Map Kit Framework Reference [Електронний ресурс]. - https://developer.apple.com/library/ios/documentation/MapKit/Reference/MapKit_Framework_Reference/_index.html

17. Cocoa Application Competencies for iOS [Електронний ресурс]. - https://developer.apple.com/library/ios/documentation/general/conceptual/Devpedia-CocoaApp/Storyboard.html

18. Instruments User Guide [Електронний ресурс]. - https://developer.apple.com/library/mac/documentation/developertools/conceptual/instrumentsuserguide/Introduction/Introduction.html

19. App Store Resource Center [Електронний ресурс]. - https://developer.apple.com/appstore/index.html

20. MVVM: проектирование приложений [Електронний ресурс]. – Режим доступу: https://skillbox.ru/media/code/

21. Pixaki [Електронний ресурс] //Режим доступу: https://rizer.co/pixaki/

22. Procreate [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Procreate_(software)

23. Interface-Builder [Електронний ресурс] //Режим доступу: https://developer.apple.com/xcode/interface-builder/

24. Interface-Builder [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Interface_Builder

25. Xcode [Електронний ресурс] //Режим доступу: https://ru.wikipedia.org/wiki/Xcode

26. Xcode [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Xcode

27. Human Interface Guidelines [Електронний ресурс] //Режим доступу: https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/navigation/

28. iOS architecture patterns [Електронний ресурс] //Режим доступу: https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52#.tliwdfd60

29. Swift programming language [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Swift_(programming_language)

30. IOS [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/IOS

31. Nintendo Switch system software [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Nintendo_Switch_system_software

32. Microsoft Windows [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Microsoft_Windows

33. Android operating system [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Android_(operating_system)

34. Casual game [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Casual_game

35. Adventure game [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Adventure_game

36. Role-playing game [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Role-playing_game

ДОДАТОК А. FeatureCard.swift

```
import SwiftUI
struct FeatureCard: View {
  var landmark: Landmark
  var body: some View {
     landmark.featureImage?
        .resizable ()
        .aspectRatio (3/2, contentMode: .fit)
        .overlay {
           TextOverlay (landmark: landmark)
        }
  }
}
struct TextOverlay: View {
  var landmark: Landmark
  var gradient: LinearGradient {
     .linearGradient (
        Gradient (colors: [.black.opacity (0.6), .black.opacity (0)]),
        startPoint: .bottom,
        endPoint: .center)
  }
  var body: some View {
     ZStack (alignment: .bottomLeading) {
        gradient
        VStack (alignment: .leading) {
           Text (landmark.name)
              .font (.title)
              .bold ()
           Text (landmark.park)
```

```
            }
            .padding ()
        }
        .foregroundColor (.white)
    }
}
struct FeatureCard_Previews: PreviewProvider {
    static var previews: some View {
        FeatureCard (landmark: ModelData (). Features [0])
    }
}
    func updateUIViewController (_ pageViewController: UIPageViewController,
context: Context) {
        pageViewController.setViewControllers (
            [UIHostingController (rootView: pages [0])], direction: .forward, animated:
true)
    }
    class Coordinator: NSObject {
        var parent: PageViewController
        init (_ pageViewController: PageViewController) {
            parent = pageViewController
        }
    }
}
```

ДОДАТОК Б. PageViewController.swift

```
import SwiftUI
import UIKit
struct PageViewController <Page: View>: UIViewControllerRepresentable {
  var pages: [Page]
  @Binding var currentPage: Int
  func makeCoordinator () -> Coordinator {
    Coordinator (self)
  }
  func makeUIViewController (context: Context) -> UIPageViewController {
    let pageViewController = UIPageViewController (
      transitionStyle: .scroll,
      navigationOrientation: .horizontal)
    pageViewController.dataSource = context.coordinator
    pageViewController.delegate = context.coordinator
      return pageViewController
  }
  func updateUIViewController (_ pageViewController: UIPageViewController,
context: Context) {
    pageViewController.setViewControllers (
      (context.coordinator.controllers [currentPage []], direction: .forward,
animated: true)
  }
  class Coordinator: NSObject, UIPageViewControllerDataSource,
UIPageViewControllerDelegate {
    var parent: PageViewController
    var controllers = [UIViewController] ()

    init (_ pageViewController: PageViewController) {
```

```
        parent = pageViewController
        controllers = parent.pages.map {UIHostingController (rootView: $ 0)}
    }
    func pageViewController (
        _ pageViewController: UIPageViewController,
        viewControllerBefore viewController: UIViewController) ->
UIViewController?
    {
        guard let index = controllers.firstIndex (of: viewController) else {
            return nil
        }
        if index == 0 {
            return controllers.last
        }
        return controllers [index - 1]
    }
    func pageViewController (
        _ pageViewController: UIPageViewController,
        viewControllerAfter viewController: UIViewController) ->
UIViewController?
    {
        guard let index = controllers.firstIndex (of: viewController) else {
            return nil
        }
        if index + 1 == controllers.count {
            return controllers.first
        }
        return controllers [index + 1]
    }
```

```
        func pageViewController (
            _ pageViewController: UIPageViewController,
            didFinishAnimating finished: Bool,
            previousViewControllers: [UIViewController],
            transitionCompleted completed: Bool) {
            if completed,
              let visibleViewController = pageViewController.viewControllers? .first,
              let index = controllers.firstIndex (of: visibleViewController) {
               parent.currentPage = index
            }
        }
    }
}
```

ДОДАТОК В. PageControl.swift

```
import SwiftUI
import UIKit
struct PageControl: UIViewRepresentable {
  var numberOfPages: Int
  @Binding var currentPage: Int
  func makeCoordinator () -> Coordinator {
      Coordinator (self)
  }
  func makeUIView (context: Context) -> UIPageControl {
      let control = UIPageControl ()
      control.numberOfPages = numberOfPages
      control.addTarget (
          context.coordinator,
          action: #selector (Coordinator.updateCurrentPage (sender :)),
          for: .valueChanged)
```

```
        return control
    }
    func updateUIView (_ uiView: UIPageControl, context: Context) {
        uiView.currentPage = currentPage
    }
    class Coordinator: NSObject {
        var control: PageControl
        init (_ control: PageControl) {
            self.control = control
        }
        @objc
         func updateCurrentPage (sender: UIPageControl) {
             control.currentPage = sender.currentPage
         }
     }
}
```