Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра мультимедійних систем

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ПІД ОПЕРАЦІЙНУ СИСТЕМУ IOS

Текстова частина до курсової роботи
за спеціальністю 122 «Комп'ютерні науки»

Керівник курсової роботи
ст.в.___Борозенний С.О.___
(прізвище та ініціали)
_____
(підпис)
"____" _____ 2022 р.
Виконав студент Миронович О.Ю.
(прізвище та ініціали)
_____
(підпис)
"____" _____ 2022 р.

Київ-2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Завідувач кафедри мультимедійних систем,

канд. фіз-мат. наук, доц. – Жежерун О.П.

_____ *(підпис)*

“____” _____ 2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студенту Мироновичу Олександру Юрійовичу
Факультету інформатики 4 р.н. бакалаврської програми
ТЕМА: Розробка мобільного застосунку під операційну систему iOS

Зміст ТЧ до курсової роботи:
Календарний план
Анотація
Вступ
1 Аналіз предметної області
2 Розробка концепції та дизайну
3 Розробка мобільного застосунку
Висновки
Список використаної літератури та посилань
Додатки (за необхідністю)

Дата видачі “____” _____ 2022 р. Керівник _____

(підпис)

Завдання отримав _____
(підпис)

Тема: Розробка мобільного додатку для операційної системи iOS

Календарний план виконання роботи:

/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	жовтень	
2.	Дослідження предметної області.	жовтень - грудень	
3.	Проведення опитування.	грудень	
4.	Розробка дизайну.	січень	
5.	Розробка застосунку.	січень - квітень	
5.	Написання текстової частини.	квітень - травень	
6.	Виправлення текстової частини.	травень	
7.	Надсилання на перевірку.	Згідно з розкладом ЕК	

Студент Миронович О. Ю.

Керівник Борозенний С. О.

“_____”_____

Зміст

Вступ

Кожного року 300 - 400 тисяч людей складають Зовнішнє незалежне оцінювання, щоб підтвердити свій рівень знань та вступити до престижного вищого навчального закладу, але аби отримати хороший результат треба прикласти багато зусиль, часу та сил, шукати курси або репетиторів, які коштують грошей, або ж якщо навчатись самостійно, то витрачати не аби яку кількість часу на пошуки потрібних матеріалів, завдань та тестів.

Саме тому було вирішено розробити сервіс для допомоги самостійної підготовки до ЗНО.

Znoshka – зручний у використанні та розумінні застосунок, який дозволяє будь-якій людині самостійно підготуватись до екзамену та скласти ЗНО на 200.

Метою даної курсової роботи – допомогти людям, в особливості випускникам шкіл, абітурієнтам, ефективно отримати всі потрібні знання аби скласти ЗНО на 200.

Завданням даної курсової роботи є розробка такого сервісу під різні платформи, яке задовольнятиме основні потреби користувачів у підготовці до екзаменів з різних предметів і буде простим та зручним у використанні та розумінні.

Об’єктом дослідження є створення мобільного застосунку під iOS для підготовки до ЗНО.

Предметом дослідження є питання наявності на ринку актуальних сервісів та створення власної альтернативи.

Практичне значення одержаних результатів дослідження полягає в тому, що аналіз наявних сервісів дозволяє врахувати їхні переваги та недоліки і на основі цих даних розробити власний сервіс, що буде поєднувати в собі сильні сторони наявних платформ.

Для командної роботи по реалізації дослідження предметної області було використано:

- Для структурування всієї інформації на початковому етапі: Miro [1] - онлайн-дошка, яка дозволяє віддалено працювати в команді.
- Для побудови прототипу сайту на початковому етапі: Figma [2] - онлайн-сервіс розробки інтерфейсів та прототипування.

Для створення мобільного застосунку під iOS Znoshka було використано:

- мова програмування Swift з використанням бібліотеки UIKit, SwiftUI та Combine.

- Для встановлення та структурування пакетів: SPM [3] - Swift Package Manager.
- Для контроля та написання коду: середовище Xcode [4].

Робота складається зі вступу, трьох основних розділів, кожен з яких містить підрозділи, висновків, списку використаних джерел та додатків.

У першому розділі проаналізовано тему як актуальне питання для дослідження, проаналізовано предметну область, розглянуто аналоги розробки, які наразі перемагають на ринку, проведено опитування задля кращого розуміння користувача та визначення основних критеріїв та функціоналу сервісу.

Другий розділ висвітлює основні відомості технічного завдання, розробку дизайну та user flow сервісу, а також визначення всіх функціональних вимог.

У третьому розділі описується сама реалізація проекту. Визначені всі необхідні вимоги до даних, які представлені у застосунку, схематично зображені інтерфейси, сценарії користувачі, наведені деталі процесу розробки, обґрунтовано вибір використаних технологій розробки та продемонстровано результати розробки.

У третьому розділі описується сама реалізація проекту. Визначені всі необхідні вимоги до даних, які представлені у застосунку, схематично зображені інтерфейси, сценарії користувачі, наведені деталі процесу розробки, обґрунтовано вибір використаних технологій розробки та продемонстровано результати розробки.

Розділ 1. Дослідження та аналіз предметної області

1.1. Аналіз сучасного стану питання та предметної області та обґрунтування теми

За даними аналітичного порталу Слово та Діло [5] за 2020 рік на Зовнішнє Незалежне Оцінювання зареєструвалися трохи більше ніж 379 тисяч осіб, з них найпопулярнішими предметами були обрані українська мова та література, математика та історія України, на які зареєструвалися відповідно 369, 186 та 264 тисяч осіб.

Отже, без сумніву тема ЗНО була актуальною та стає все актуальнішою з кожним роком, але хоч питання ЗНО і є достатньо популярним, досі присутні проблеми та недоліки пов'язані із підготовкою.

Спираючись на статистичні дані основної сесії ЗНО 2021 року Українського центру оцінювання якості освіти [6], середнім балом після складання іспиту по предмету українська мова та література складає 143,5 балів, що є достатньо низьким показником та вказує на присутність проблем або у складності самого тесту, або ж у підготовці учнів.

Згадуючи свої роки наполегливої підготовки до іспиту та маючи друзів, які здавали екзамен у попередніх роках, можна зробити висновок, що достатньо багато є проблем пов'язаних із складанням ЗНО та якісною й ефективною підготовкою і достатньо мало ресурсів, які б справді допомагали вирішувати більшість із них.

1.2. Опитування

Задля кращого розуміння самих проблем та потреб учнів-випускників було створено та розіслано у доступні чати абітурієнтів різних ВНЗ анонімне анкетування у якому взяло участь 318 людей.

Посилання на саму анкету - https://docs.google.com/forms/d/1sy3ZFISI8iG0nLq8dKx6Aen7WwnukyawLxo3lMP1sM4

1.2.1. Ключові інсайди

Проаналізувавши результати, було виділено такі ключові пункти:

1. 74,8% опитуваних готуються від 1 місяця до 1 року

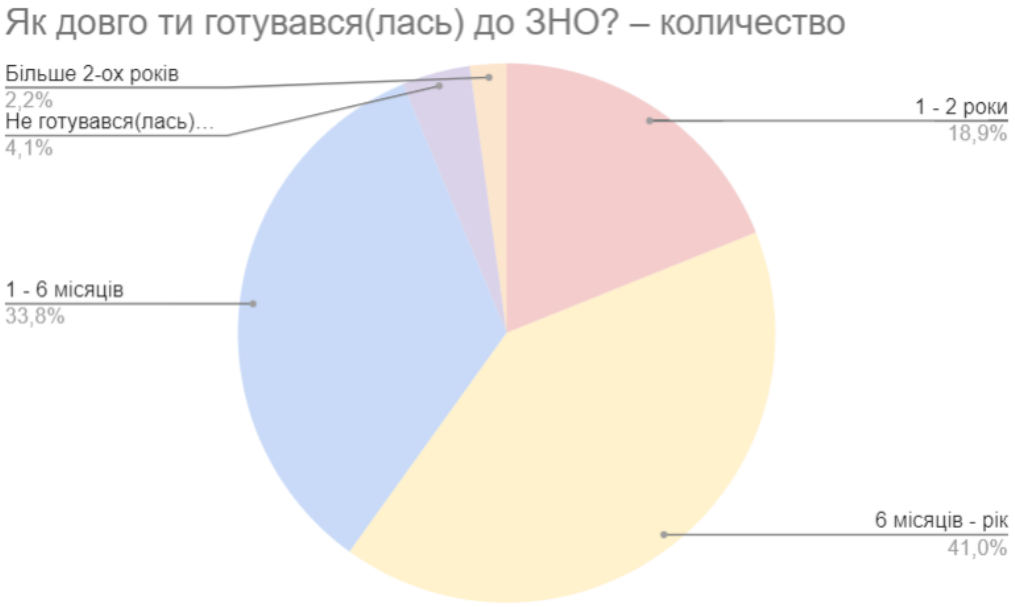

Рисунок 1 – Результати опитування «Як довго готувались до ЗНО?»

2. Не зважаючи на проблему того, що весь матеріал розкиданий по різних ресурсах та книжках, люди більше загострювали увагу на тому, що не вистачає якихось абстрактних речей: часу, мотивації, бажання, самодисципліни.
   Якщо розміщувати за спадання:
   - Часу (17, 6%)
   - Мотивації (12, 2%)

- Бажання (7, 1%)
- Практики (5,7%)
- Самодисципліни (5,4%)
- Пояснення (4,1%)
- Терпіння (3,7%)
- Ментора (3,0%)
- Концентрації(2,7%)

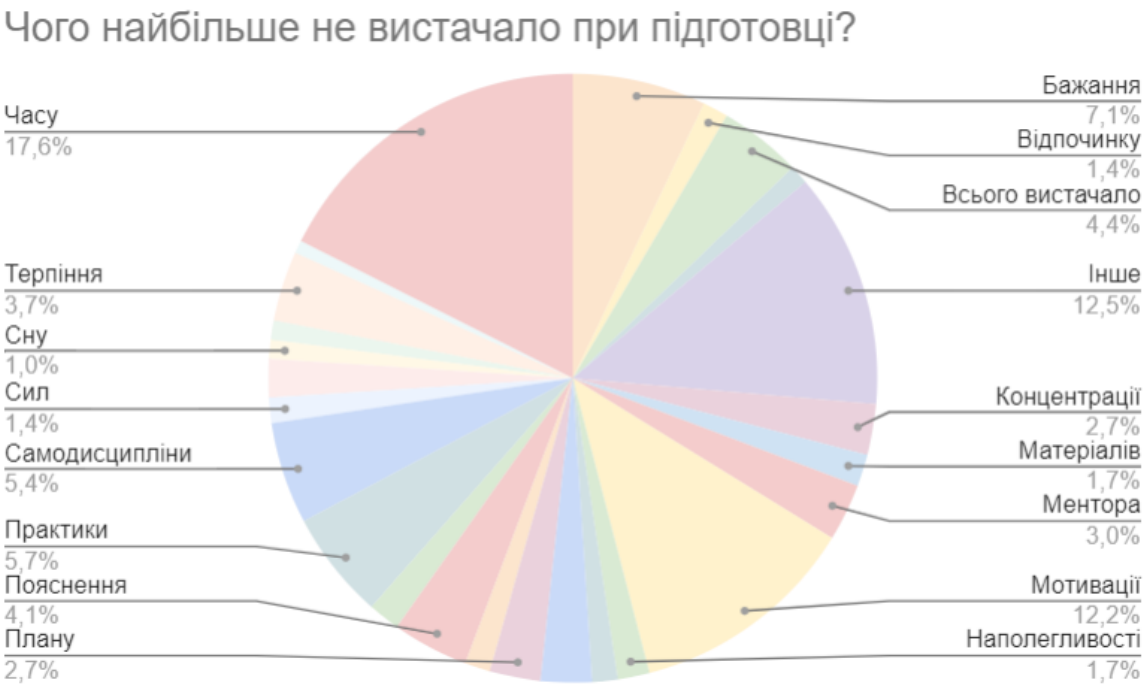

Рисунок 2 – Результати опитування «Чого найбільше не вистачило при підготовці»?

3. Найбільше при підготовці люди використовували такі ресурси як (за спаданням):
    - Самостійна підготовка (78, 9%)
    - Репетитор (65, 9%)

- Курси (33,1%)

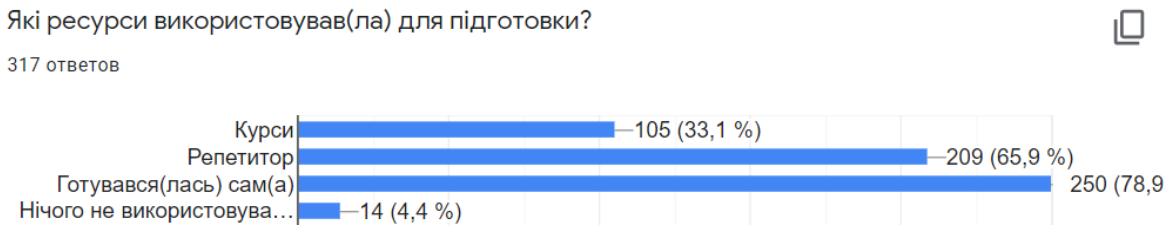

Рисунок 3 – Результати опитування «Які ресурси використовувались для підготовки?»

4. Найефективнішим люди вважають такі методи підготовки як (за спаданням):
    - Репетитор (65,3%)
    - Самостійна підготовка (50,8%)
    - Перегляд відео матеріалу в Youtube (41%)
    - Курси (34,4%)

Рисунок 4 – Результат опитування «Який метод підготовки вважаєш найефективнішим?»

5. На рахунок того, які основні вимоги були б до ідеального сервісу підготовки до іспиту, люди виділили наступні частини:
    - Більше тестів (завдань які за структурою саме схожі на завдання ЗНО)
    - Різна, стисла, інформативна доступна подача матеріалу, таблиці, графіки, картинки, меми, стислі вижимки творів, формули, відео та аудіо подача матеріалу
    - Можливість знайти однодумців, готуватися разом, обговорення, чати
    - Відслідковування прогресу та календар

- Детальне пояснення завдань які правильно і неправильно розв'язав

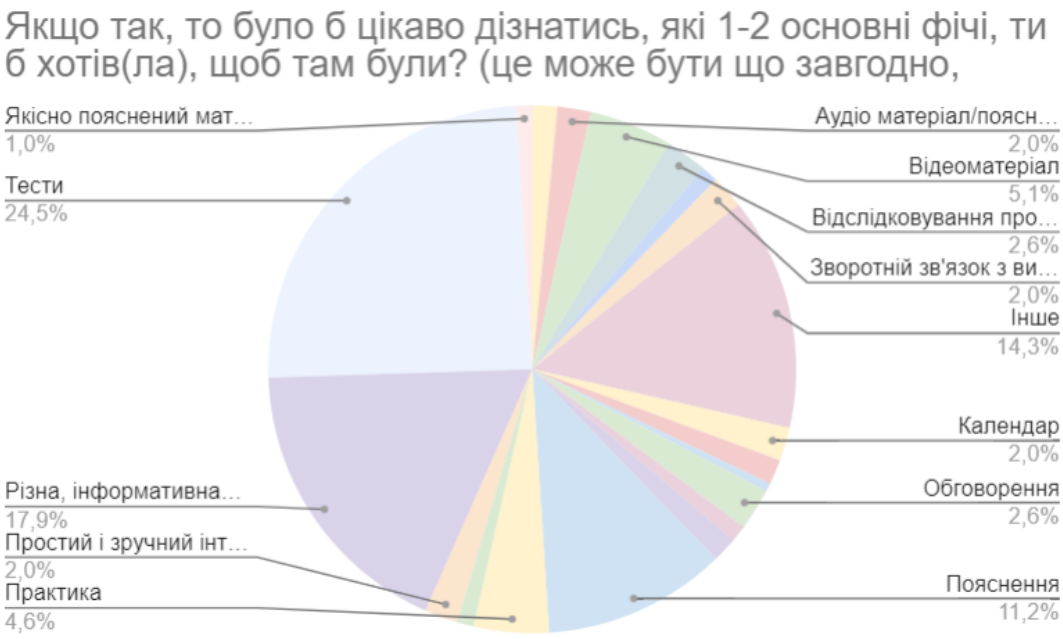

Рисунок 5 – Результати опитування «Який би функціонал хотіли бачити»?

Головне різноманітність подачі стислого матеріалу, багато тестів з поясненнями, відслідковування прогресу та можливість знайти однодумців

1.2.2. Висновок

Отже, проаналізувавши відповіді учнів, можна зробити певні висновки по критеріям, які справді важливі людям під час підготовки.

По-перше, справді існує проблема того, як подається матеріал і як його знаходити: багато хто вважає, що матеріал розкиданий по різним ресурсам, що сповільнює концентрацію та мотивацію самих учнів. Багато хто також зазначив, що не вистачає різноманітності подачі матеріалу, тобто подавати теми підручників не тільки як конспект, а було б добре отримувати один і той самий матеріал у вигляді конспекту, таблиці, картинок, схем і так далі, що допоможе засвоїти матеріал краще.

По-друге, самопідготовка вважається однією з основних і дієвим методів підготовки до ЗНО, саме тому, достатньо багато людей вибирають його, а не підготовку на курсах і репетиторів. Також, самопідготовка не коштує стільки грошей як перелічені вище способи, що також грає суттєву роль.

До того ж дуже важливо, щоб було детальне роз'яснення тестів за завдань та можливість поспілкуватись з іншими на рахунок помилок або нерозумінь певних вправ.

І останній суттєвий пункт, відслідковування прогресу та статистики є одним із ключових пунктів, яке впливало на мотивацію учнів за весь період підготовки.

1.3. Огляд існуючих сервісів-аналогів

Задля кращого розуміння ситуації на ринку було взято декілька відомих сервісів, які допомагають учням підготуватись до Зовнішнього Незалежного Оцінювання.

Маючи вище наведені критерії для застосунків після проведення опитування, можемо виділити основні пункти по яким можна перевірити сервіси-аналоги:

- різноманітність подання інформації(відео, текст, аудіо, картинки, таблиці, схеми, меми і тд.)
- різноманітність та велика кількість завдань і тестів
- детальне роз'яснення задач
- можливість знайти однодумців або обговорити певну задачу
- статистика та прогрес користувача

1.3.1. iLearn

iLearn [7] – це безкоштовна гейміфікована платформа з навчальними онлайн-курсами, тестами та вебінарами для всіх, хто бажає навчатися та успішно скласти Зовнішнє Незалежне Оцінювання. Функціонал платформи з ігровими елементами, що робить процес навчання інтерактивним та цікавим. Під час реєстрації на платформі користувач створює свого віртуального героя. Його можна покращувати впродовж користування платформою за віртуальну валюту – вчибакси, які нараховуються за проходження тестів та перегляд навчальних вебінарів.

Критерії	*Наявність*	*Коментар*
Різноманітність подання інформації	+ -	Є вебінари, які наведені у відео форматі, тести зно та квести у ігровому форматі
Різноманітність завдань та тестів	-	Є тести до вебінарів, їх достатньо багато, але більше вибору нема
Детальне роз'яснення задач	-	Немає варіанту переглянути правильні відповіді, якщо відповів не правильно
Можливість знайти однодумців або обговорити певну задачу	+ -	У профілі користувача є окрема секція під назвою “Друзі”, де можна кинути запит комусь та встановити контакт з іншими юзерами

Статистика та прогрес користувача	+	У профілі користувача є окрема секція під назвою “Прогрес”, де відображається вебінарам, тестам, тренажерам та курсам

Таблиця. 1. iLearn

1.3.2. На всі двісті

На всі двісті [8] – це веб-застосунок з навчальними відео на тему української мови та літератури та тестами. Місія - допомагати старшокласникам самостійно готуватися до ЗНО лише з української мови та літератури, створюючи новітні, якісні освітні матеріали, що мотивують вчитися.

Критерії	*Наявність*	*Коментар*
Різноманітність подання інформації	-	Наявні тільки відеоуроки
Різноманітність завдань та тестів	+	Достатньо велика кількість тестів до кожної теми
Детальне роз'яснення задач	+	У кожного блоку тесту можна переглянути правильні відповіді та зробити роботу над помилками
Можливість знайти однодумців	+ -	У Телеграмі для спілкування з Анною (куратором) й іншими старшокласниками

або обговорити певну задачу		
Статистика та прогрес користувача	+	До кожного тесту та теми можна подивитись свій прогрес

Таблиця. 2. На всі двісті

1.3.3. Освіта.ua

Освіта.ua [9] – сайт для проходження тестів ЗНО онлайн та підготовки майбутніх абітурієнтів для проходження зовнішнього незалежного оцінювання. На сайті розміщені тести, що складали абітурієнти під час зовнішнього незалежного оцінювання 2009-2021 років, а також варіанти тестів, що пропонувались вступникам до вищих навчальних закладів України під час пробного зовнішнього незалежного оцінювання 2009-2021 років.

Критерії	*Наявність*	*Коментар*
Різноманітність подання інформації	-	Наявні тільки тести
Різноманітність завдань та тестів	+	Наявні всі тести з 2009 по 2021 рік, хоч і немає власних вправ для підготовки
Детальне роз'яснення задач	+	До кожного пункту тесту можна переглянути коментар (попередньо авторизувавшись) та переглянути роз'яснення

Можливість знайти однодумців або обговорити певну задачу	-	-
Статистика та прогрес користувача	-	-

Таблиця. 3. Освіта.ua

1.3.4. ПростеЗНО

ПростеЗНО [10] – додаток для підготовки до Зовнішнього Незалежного Оцінювання. ПростеЗНО включає в себе 7+ ігрових режимів проходження тестів, усі тести ЗНО з 2009 по 2021 роки з усіх предметів, зручну та зрозумілу теорія, що постійно поповнюється новими темами, відповідно до програми ЗНО, та файлами-помічниками, розумний підбір питань, що допоможе краще засвоїти матеріал та відточити навички на помилках та інше.

Критерії	*Наявність*	*Коментар*
Різноманітність подання інформації	+ -	До кожного предмету наявні теоретичні теми з файлами pdf з повною інформацією
Різноманітність завдань та тестів	+	Наявні всі тести минулорічних ЗНО, а також власні вправи для підготовки

Детальне роз'яснення задач	-	Наразі роз'яснень немає, хоч і показує правильну відповідь, якщо відповів неправильно
Можливість знайти однодумців або обговорити певну задачу	+	У вкладці “Спільнота” є посилання на телеграм канал, до якого можна долучитись
Статистика та прогрес користувача	+ -	У профілі користувача є небагато статистики по пройденим тестам

Таблиця. 4. ПростеЗНО

1.3.5. Складу ЗНО

Складу ЗНО [11] – ще один додаток для підготовки до Зовнішнього Незалежного Оцінювання. У ньому знаходиться величезна кількість теоретичних і практичних матеріалів для підготовки до ЗНО та ДПА.

Критерії	*Наявність*	*Коментар*
Різноманітність подання інформації	+ -	До кожного предмету наявні теоретичні теми з файлами pdf з повною інформацією
Різноманітність завдань та тестів	+ -	Наявні тільки тести минулорічних зно

Детальне роз'яснення задач	-	Наразі роз'яснень немає, хоч і показує правильну відповідь, якщо відповів неправильно
Можливість знайти однодумців або обговорити певну задачу	-	-
Статистика та прогрес користувача	-	-

Таблиця. 5. Складу ЗНО

1.3.6. Порівняльна таблиця

Критерії	*iLearn*	*На всі двісті*	*Освіта. ua*	*ПростеЗНО*	*Складу ЗНО*
Різноманітність подання інформації	+ -	-	-	+ -	+ -
Різноманітність завдань та тестів	-	+	+	+	+ -
Детальне роз'яснення задач	-	+	+	-	-
Можливість знайти однодумців або	+ -	+ -	-	+	-

огроворити певну задачу					
Статистика та прогрес користувача	+	+	-	+ -	-

Таблиця. 6. Порівняльна таблиця всіх аналогів

1.3.7. Висновок

Проаналізувавши існуючі на ринку застосунки можна побачити, що не існує ідеального сервісу для підготовки до ЗНО, який задовольнив би всі потрібні критерії користувачів. Існують достатньо хороші сайти та додатки з різноманітною подачею інформації, але бракує, наприклад, статистики або детального розбору вправ, або ж навпаки.

1.4. Постановка завдання

1.4.1. Розробка стейкхолдера

Після проведення опитування та отримавши ширшу картину існуючих сервісів на сучасному ринку, можемо розробити стейкхолдера продукту:

Рисунок 6 - стейкхолдер

Маємо школярку Марію, яка навчається в 11 класі, готується до ЗНО від 6 місяців до 1 року, дуже багато вчиться, адже є школа і також окремо підготовка до іспиту з репетиторами, курсами та самопідготовка. Марія не розуміє, що буде далі, вигорає через надмірне навантаження, не вистачає часу, мотивації та бажання щось робити далі.

1.4.2. Основні проблеми та варіанти їх вирішення

Після проведення опитування, аналізу проблем з якими стикаються учні під час підготовки до іспиту та аналізом уже існуючих сервісів, їх переваг та недоліків, можна виділити такі проблема та варіанти їх вирішення:

1. Марії не вистачає вправ та завдань для хорошої підготовки.

   Варіанти рішення:

   - Додати посилання на додаткові матеріали до кожної теми предмету.

- Додати відеоролики до кожної теми.
- До кожної теми предмету додавати тести..
- Створити секцію обговорень у кожній темі до теми де учні можуть додавати щось від себе.
- Додати тести типу ЗНО.
- Додати тести реального ЗНО.
- Контент до кожної теми повинен бути у вигляді тексту, таблиць, графіків, малюнків, схем і тд.

2. Марії потрібно бачити свій прогрес, щоб не втрачати мотивацію та бажання рухатись далі.

   Варіанти рішення:

   - Додати особистий календар.
   - Відображати статистику юзера у нього в профілі.
   - До кожного предмета показувати скільки відсотків предмету користувач вже пройшов.

3. Марії не вистачає якісного та детального пояснення завдань.

   Варіанти рішення:

   - Прописувати детальне пояснення до кожного теста або задачі, яке було б доступне юзеру у кожній вправі
   - Наводити релевантні приклади з життя.
   - Сам матеріал до тем подавати у різному форматі: текст, таблиці, схеми і тд.
   - Надавати можливість іншим постити свої рішення задач.

4. Марія хоче знайти однодумців, щоб спілкуватись у разі виникнення проблем.

   Варіанти рішення:

   - Створити телеграм чат для обговорення.

- До кожної вправи мати блок обговорення або місця, де інші б змогли поділитись своїми розв'язками.

5. Марія хотіла б мати можливість переглядати теоретичну частину у різному форматі, аби краще запам'ятати матеріал.

   Варіанти рішення:

   - Надавати у теоретичній частині інформацію у вигляді конспекту, таблиць, картинок, схем, мемів, флешкарт, аудіо формату, відео формату.

1.4.3. Основні вимоги

Можна визначити, що завданням даної курсової роботи є розробка мобільного застосунку для підготовки до ЗНО, що матиме наступні критерії:

- Користувач повинен мати можливість зареєструватись у системі або зайти, якщо вже має обліковий запис.
- Користувач матиме можливість переглядати предмети та тести.
- Для кожного предмету будуть доступні теоретична та практична частина.
- У теоретичній частині предмету буде можливість переглянути конспект, таблиці, схеми, картинки та посилання на додаткові матеріали, які б допомогли користувачу ще більше заглибитись у тему.

- У практичній частині предмету буде перелік вправ, схожих на ті, що фігурують у самому іспиті, які користувач може пройти та перевірити.
- У кожній вправі буде можливість зайти в розділ з різними розв'язками, подивитись їх, поставити вподобайку, антивподобайку, або залишити комент, що дозволить користувачам спілкуватись та дискутувати на рахунок розбіжностей.
- Користувач матиме можливість пройти або ж переглянути вже пройдений ним тест.
- Користувач матиме можливість зайти у профіль та переглянути певну свою статистику.

Розділ 2. Розробка концепції та дизайну

2.1. Основна мета

Основною метою цієї курсової роботи, є проектування та розробка мобільного застосунку, що допоможе абітурієнтам успішно підготуватись до здачі зовнішнього незалежного оцінювання.

Основний акцент повинен робитись на практичних заняттях, щоб користувач міг з легкістю набити руку та звикнути до самих типів завдань та структури тестів.

Іншим важливим аспектом є також засвоєння самої теорії, а саме, підготовка до тестів та завдань. Для цього в застосунку має бути можливість засвоювати інформацію у різних поданнях, таких як: конспекти, таблиці, відео, картинки і тд.

Також, однією з головних цілей застосунку, є надання користувачеві детальної аналітики щодо його результатів, адже це є важливим інструментом, що допоможе знати свої слабкі та сильні сторони і відповідно знати, як краще готуватись.

2.2. Визначення технічного завдання

Завдяки раніше зробленому опитуванню, було розроблено перелік основних функцій, що повинні бути присутні у мобільному застосунку:

1) Реєстрація користувача
2) Авторизація користувача
3) Перегляд предметів, до яких можна готуватись
4) Можливість пройти тести минулих років
5) Можливість пройти додаткові тести відфільтровані по темам
6) Доступ до навчальних матеріалів
7) Можливість переглядати аналітичних даних по кожному предмету
8) Можливість обговорювати певні завдання з іншими користувачами

9) Можливість оцінювати рішення інших користувачів

2.3. Користувацький потік

Для кращого розуміння користувацького потоку було розроблено спеціальні діаграми, що вказують звідки та куди користувач може переходити.

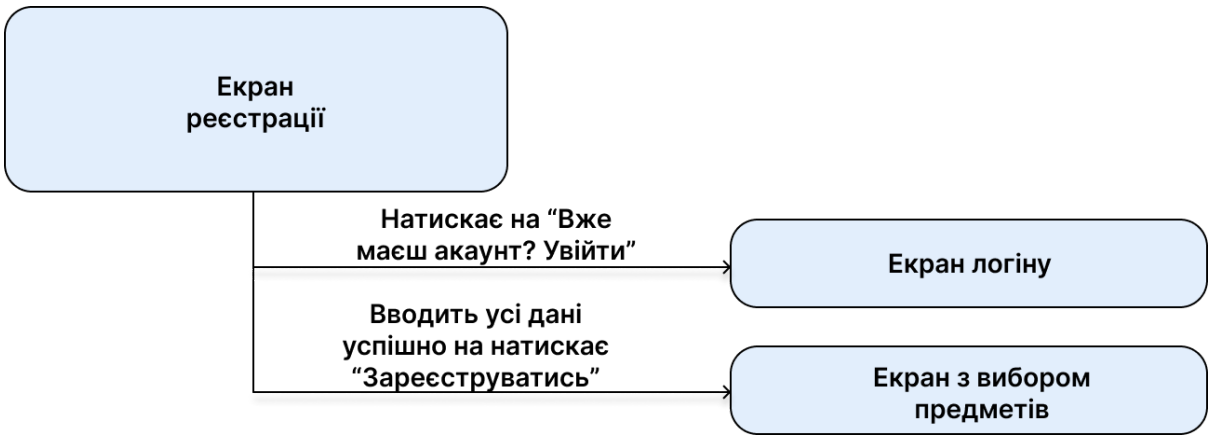

Рисунок 7 – потік користувача на екрані реєстрації

Рисунок 8 – потік користувача на екрані логіну

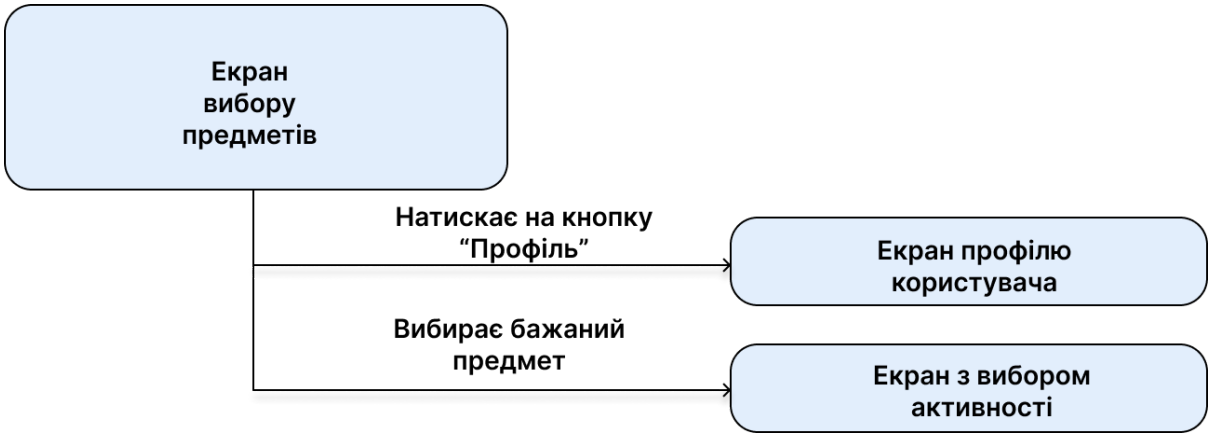

Рисунок 9 – потік користувача на екрані вибору предмету

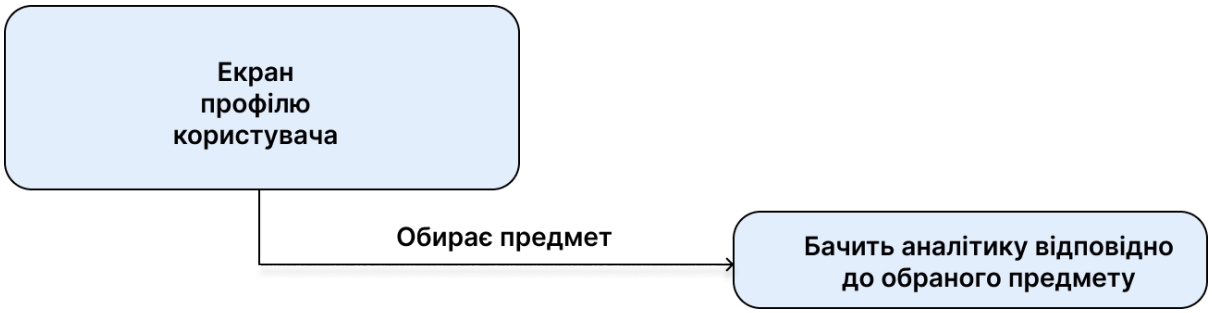

Рисунок 10 – потік користувача на екрані профілю користувача

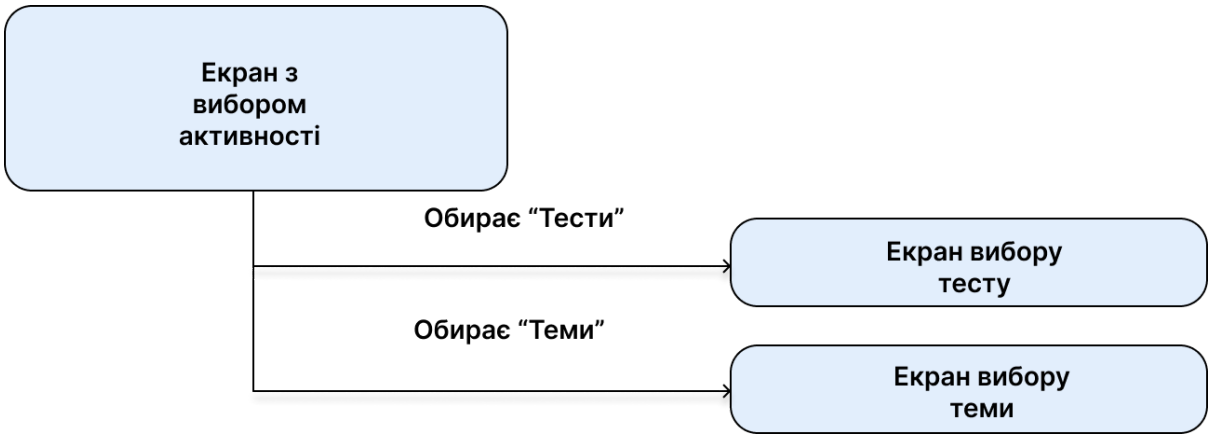

Рисунок 11 – потік користувача на екрані вибору бажаної активності

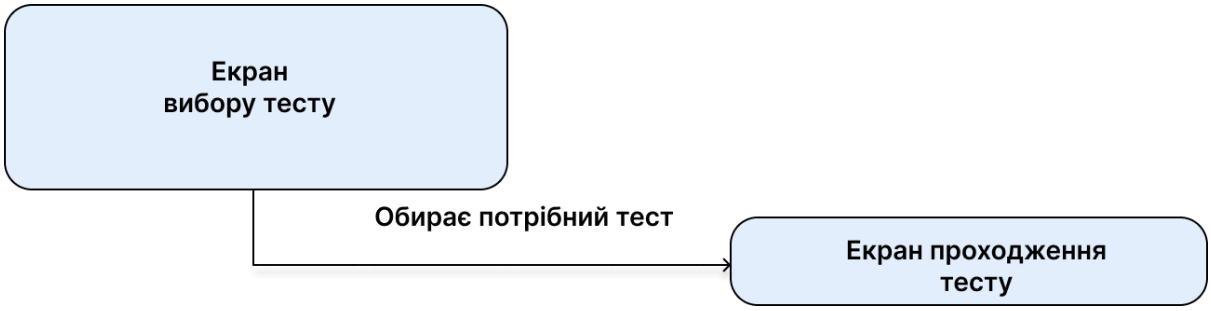

Рисунок 12 – потік користувача на екрані вибору тесту

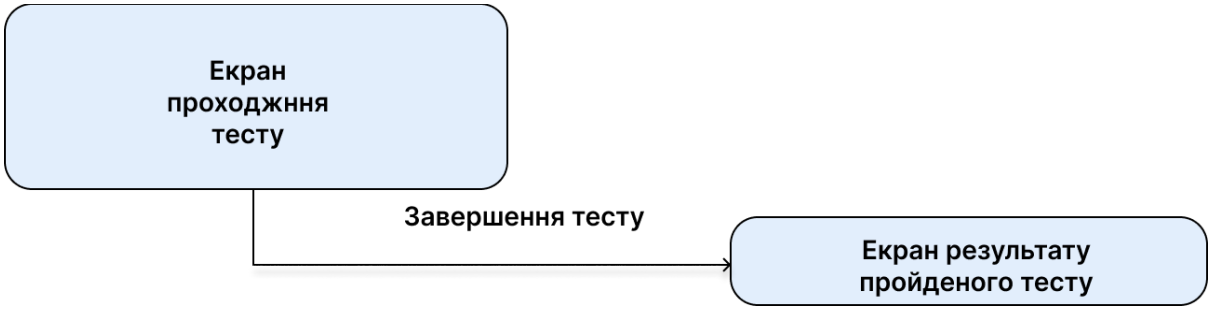

Рисунок 13 – потік користувача на проходження тесту

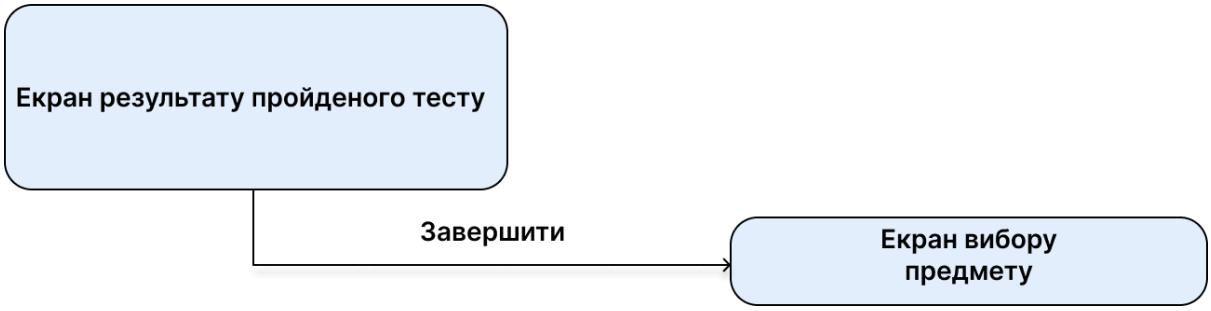

Рисунок 14 – потік користувача на екрані пройденого тесту

Також важливо виділити те, що на всіх вище приведених екранах є завжди можливість повернутись до попереднього стану.

2.4. Розробка користувацького інтерфейсу відповідно до вимог

2.4.1. Розробка дизайн системи

Для розробки користувацького інтерфейсу було спеціально розроблено дизайн систему що включає в себе наступні елементи:

- шрифт
- поле для введення
- дизайн клітинки предмету
- дизайн клітинки тесту
- дизайн клітинки пройденого тесту
- дизайн клітинки теми
- кнопки
- палітру кольорів

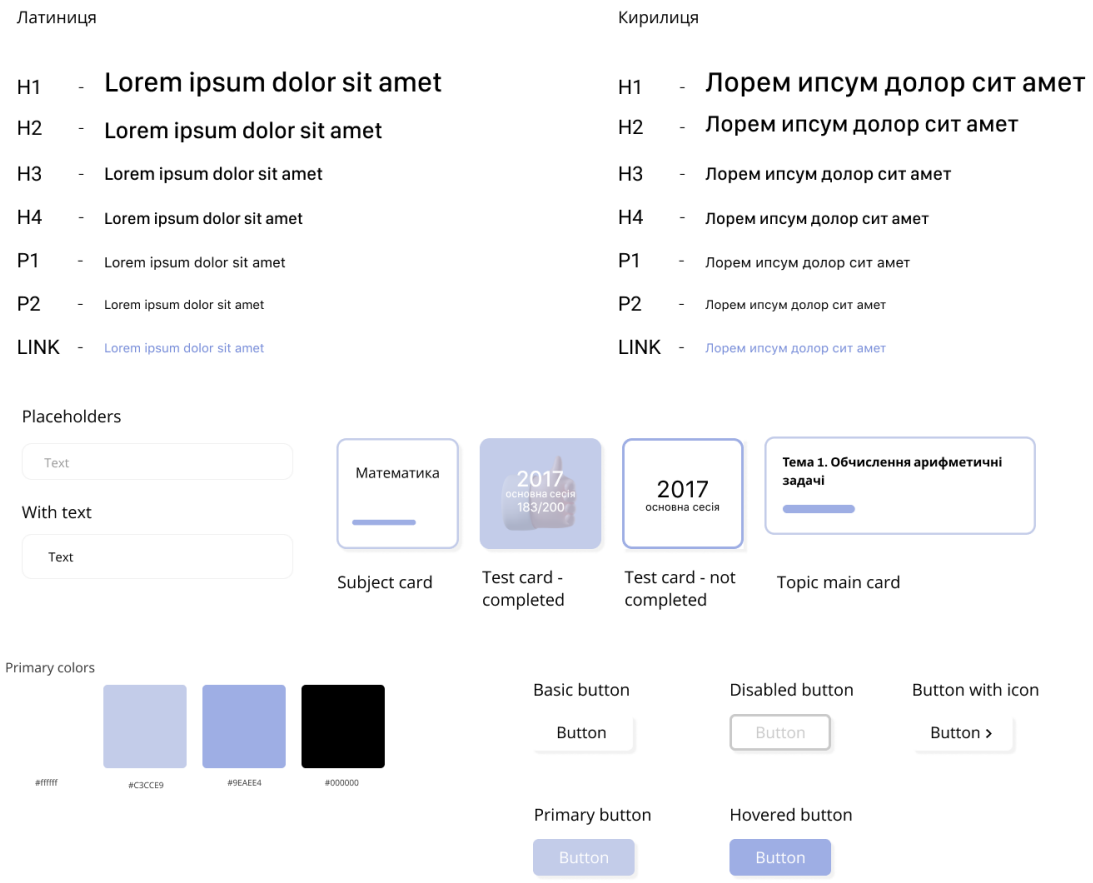

Рисунок 15 – дизайн система

2.4.2. Розробка дизайну інтерфейсу

За допомогою дизайн системи, що була створена раніше, було розроблено дизайн для всіх потрібних екранів застосунку.

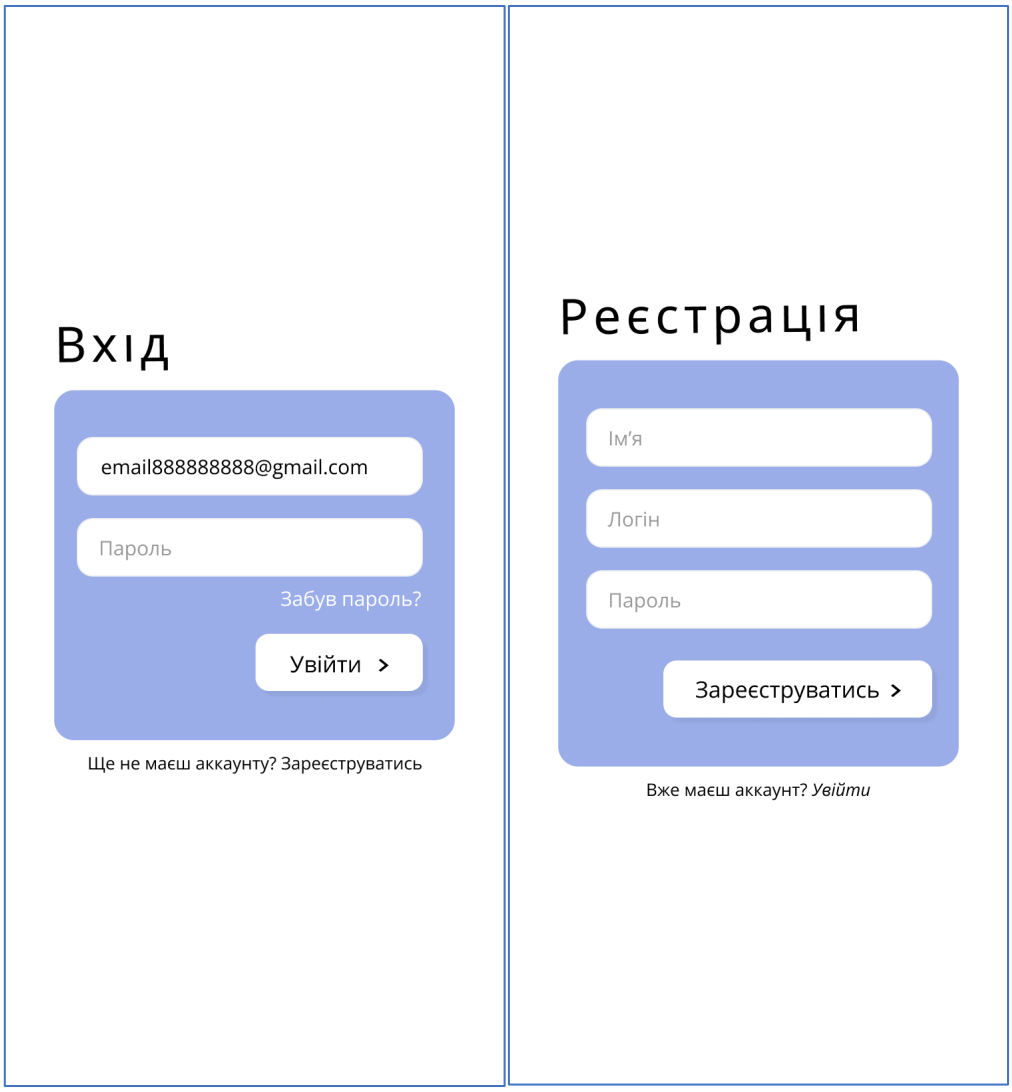

Рисунок 16 – дизайн для сторінок авторизації та реєстрації

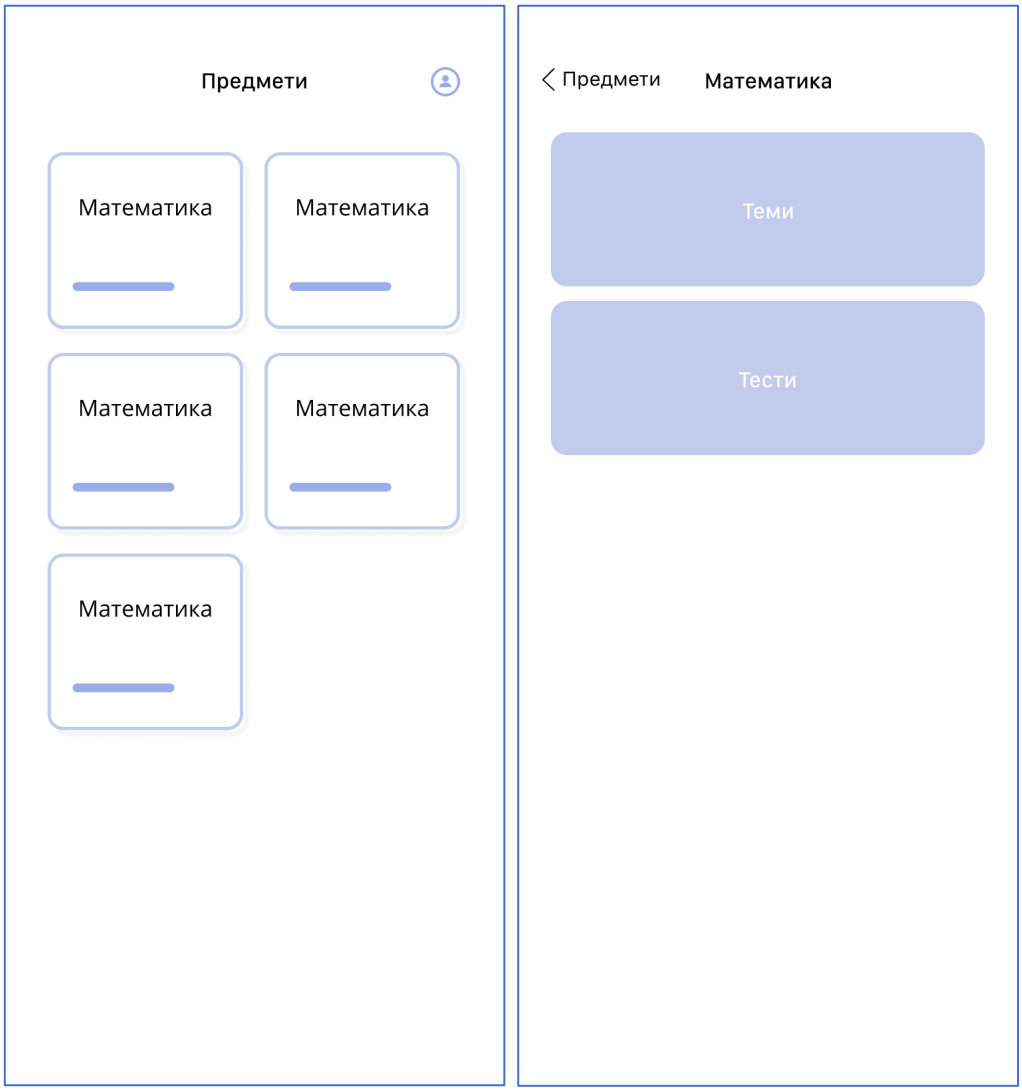

Рисунок 17 – екран вибору предмету та активності

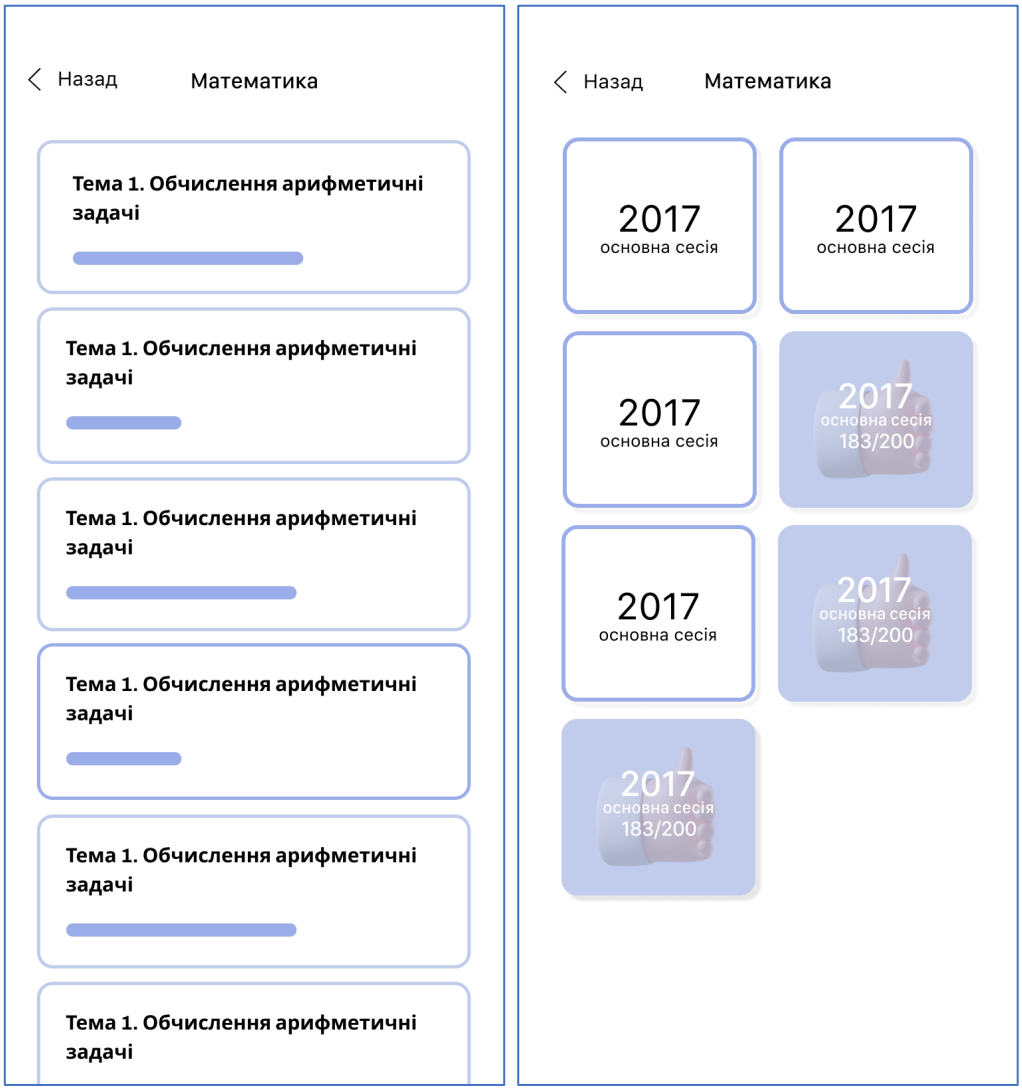

Рисунок 18 – екрани вибору тесту та теми

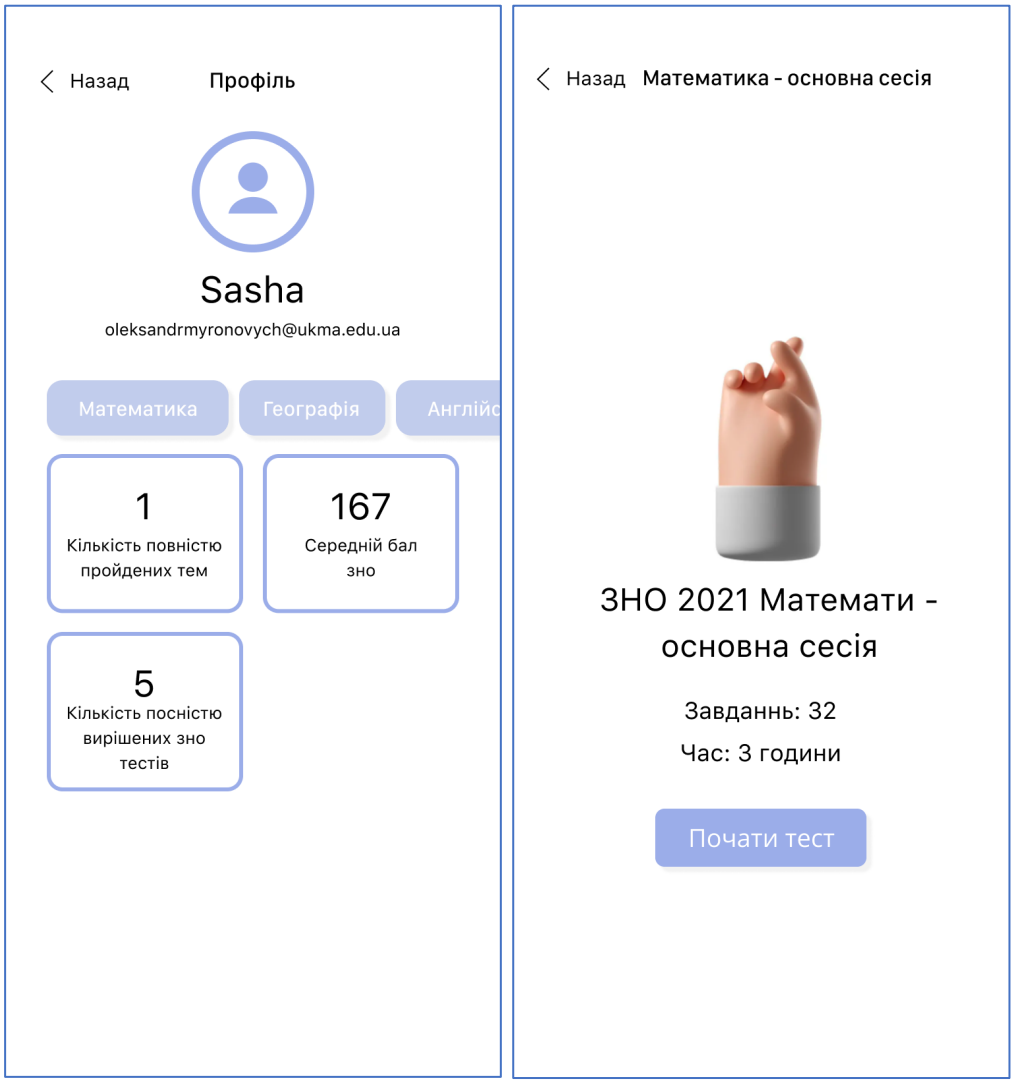

Рисунок 19 – екран профілю користувача та екран для початку проходження тесту

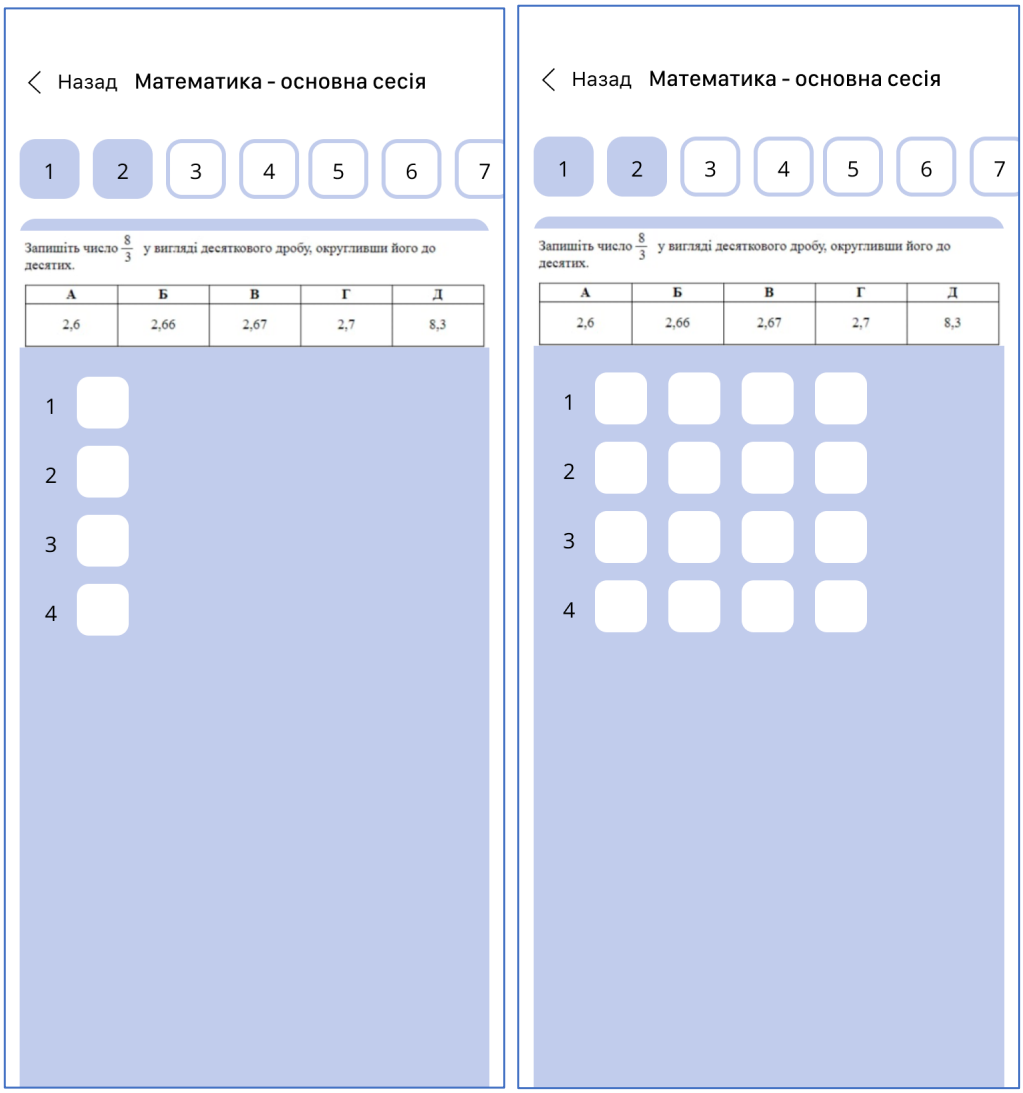

Рисунок 20 – екрани приклади тестових завдань

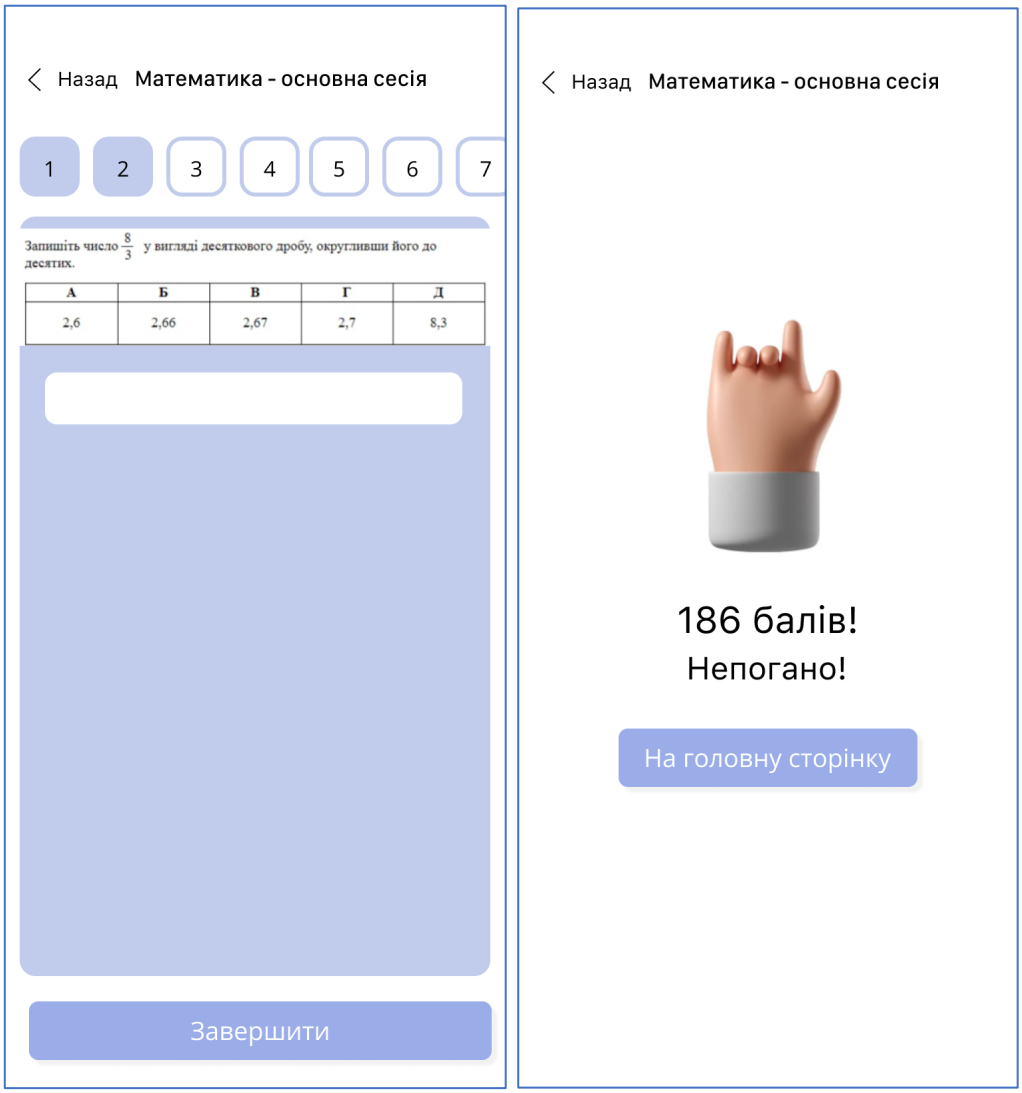

Рисунок 21 – екрани, що демонструють завершення проходження тесту

Розділ 3. Розробка мобільного застосунку

3.1. Архітектура застосунку

Сам застосунок Znoshka планується поступово розробляти, підтримувати поточний функціонал і розвивати, саме тому дуже важливо обрати правильну архітектуру.

Хотілось обрати архітектуру таку щоб можна було досить швидко та зручно розробляти новий функціонал і розвивати вже існуючий без особливих зусиль. Для цього було обрано архітектуру MVVM+Coordinator.

Сама по собі архітектура MVVM (Model-View-ViewModel) передбачає основні 3 компоненти.

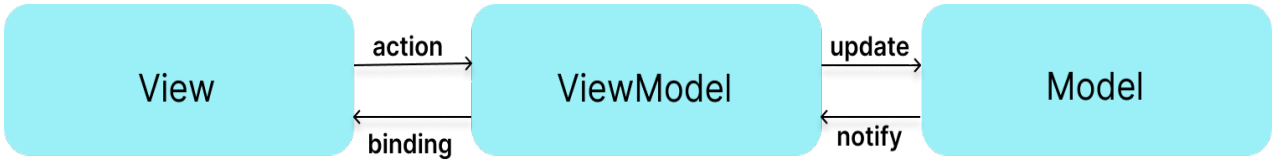

Рисунок 22 – архітектура MVVM.

Сама в'ю напряму володіє в'ю моделлю та після взаємодії користувача передає сигнал в'ю моделі. Після чого в'ю модель реагує та відпрацьовує потрібну бізнес логіку після чого оновлює модель, якою володіє і наша в'ю реагує на зміну в'ю моделі за допомогою раніше прокинутих байндінгів.

Я вирішив обрати саме цю архітектуру, бо порівняно з архітектурою MVC, яку рекомендують Apple, вона дозволяє нам застосувати набагато краще розподілення відповідальності, також уникнути сильного зв'язку між основними компонентами, а отже писати набагато більш кращий код, який можна тестувати, і що також нам дозволить набагато легше розширювати новий функціонал.

Проте в цій архітектурі упущений дуже важливий аспект, такий як – навігація. Немає певного стандарту як краще переходити з одного екрану на інший, відповідно в цій зоні створюється певне непорозуміння та хаос, адже не зовсім правильно відносити навігацію до шару в'ю так само як і не правильно відносити його до в'ю моделі. В даному випадку, використання шаблону «Координатор» дозволить нам усунути логіку навігації з наших в'ю, що також дозволить нам зробити їх більш перевикористованими та покращить розподілення відповідальності у проекті.

Відповідно архітектура буде мати наступний вигляд:

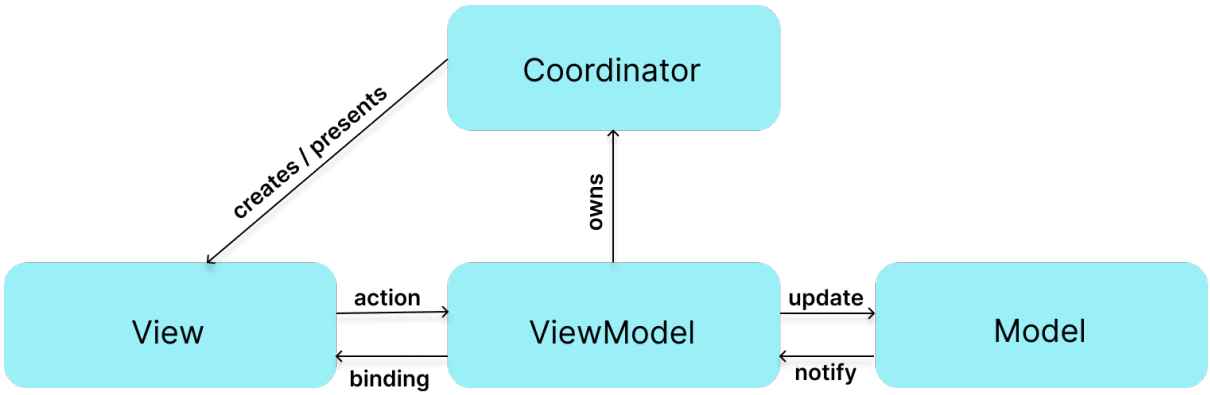

Рисунок 23 – архітектура MVVM-C

Після застосування даної архітектури, відповідальність за навігацію, створення нових екранів та їх наповнення залежностями бере на себе об'єкт що буде служити координатором. Кожен координатор буде реалізовувати наступний протокол (інтерфейс).

```
protocol Coordinator: AnyObject {
    var navigationController: UINavigationController { get set }
    var parentCoordinator: Coordinator? { get set }
    var childCoordinators: [Coordinator] { get set }

    func start()

    func addDependency(_ coordinator: Coordinator)
    func removeDependency(_ coordinator: Coordinator)
}
```

Рисунок 24 – протокол координатора

У протоколі зазначено наступні атрибути:

1) navigationController – зберігаємо посилання на обєкт типу UINavigationController, щоб за допомогою нього показувати / приховувати екрани.
2) parentCoordinator – об'єкт, що також реалізує протокол Coordinator, він потрібен для того, щоб повідомити коли поточний координаток завершив свою роботу, і батьківський координаток продовжив свою.
3) childCoordinators – дочірні координатори, масив об'єктів, що реалізують Coordinator. Ми можемо мати декілька координаторів, де кожен може відповідати за навігацію певної частини, що також дозволяє нам краще розбити відповідальність.

Також зазначено 3 функції:

1) start() – функція, що запускає координатор, а саме створює та презентує потрібний екран.
2) addDependancy(_ coordinator: Coordinator) – функція, що додає переданий координатор до childCoordinators.
3) removeDependancy(_ coordinator: Coordinator) – функція, що прибирає зазначений координатор з childCoordinators.

Оскільки, addDependancy та removeDependancy будуть однакові незалежно від координатора, ці дві функції було реалізовано за допомогою розширення до протоколу Coordinator.

```
extension Coordinator {
    func addDependency(_ coordinator: Coordinator) {
        if childCoordinators.contains(where: { $0 === coordinator }) {
            return
        }
        childCoordinators.append(coordinator)
    }

    func removeDependency(_ coordinator: Coordinator) {
        childCoordinators = childCoordinators.filter { !($0 === coordinator) }
    }
}
```

Рисунок 25 – розширення протоколу Coordinator

3.2. Огляд інструментів для створення користувацького інтерфейсу

На сьогоднішній день користувацький інтерфейс є однією з ключових складових успішного технологічного продукту. Коли мобільна розробка тільки починалась з'являтись то юзер інтерфейси були досить простими та використовували лише основні компоненти без будь-якої їх кастомізації. Розробка мобільних застосунків з часом сильно розвивалась і разом з нею розвивались підходи до створення їх дизайнів. Ще з самого початку як було представлено iPhone та iOS разом з ними Apple продемонструвала свій перший фреймворк для створення користувацького інтерфейсу під назвою UIKit. Також відносно нещодавно, а саме у 2018 році, було представлено бібліотеку під назвою SwiftUI. Пропоную розглянути ці інструменти.

3.2.1. Бібліотека UIKit

Американська компанія представила UIKit у 2008 році. Це фреймворк, що дозволяє будувати користувацький інтерфейс, який може

обробляти події користувача та вхідні дані. До цього він розроблявся 2 роки приватно і лише після цього був представлений публіці.

UIKit було створено за допомогою мови Objective-C, не дивлячись на це, його можна спокійно використовувати за допомогою мови Swift. Ця бібліотека надає багато основних об’єктів, включаючи ті через які відбувається спілкування з системою, запускається головний цикл подій і відображення контенту на екрані.

UIKit надає більшість об’єктів для відображення, а саме реалізує основний клас UIView, від якого успадковуються усі інші класи, що представляють елементи користувацького інтерфейсу.

Важливо згадати про те, що UIKit дозволяє використовувати його декількома способами:

1) Storyboard – це спосіб створення користувацького інтерфейсу без написання коду, а саме перетягуванням та розташуванням ui елементів у спеціальний файл з розширенням .storyboard.

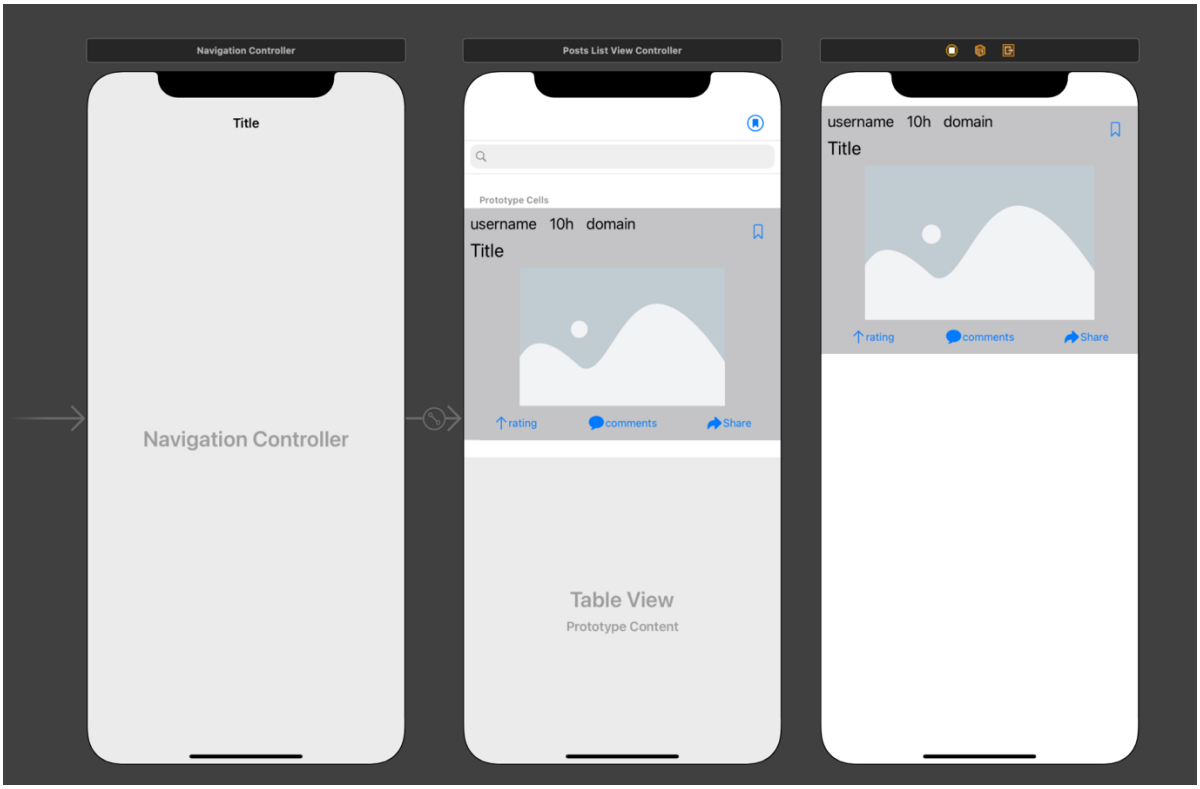

Рисунок 26 – створення інтерфейсу через storyboard

2) Програмно – об’єкти інтерфейсу створюються та задаються через код.

```
let verticalStack: UIStackView = {
    let stackView = UIStackView()
    stackView.translatesAutoresizingMaskIntoConstraints = false
    stackView.axis = .vertical
    return stackView
}()

private func setupStack() {
    addSubview(verticalStack)

    NSLayoutConstraint.activate([
        verticalStack.leadingAnchor.constraint(equalTo: leadingAnchor),
        verticalStack.topAnchor.constraint(equalTo: topAnchor),
        verticalStack.leadingAnchor.constraint(equalTo: leadingAnchor),
        verticalStack.bottomAnchor.constraint(equalTo: bottomAnchor)
    ])
}
```

Рисунок 27 – створення інтерфейсу програмно за допомогою autolayout.

3.2.2. SwiftUI

SwiftUI відносно новий фреймворк у екосистемі Apple, що був представлений у 2019 році. На ньому можна лише розробляти під iOS 13 та вище, але навіть під iOS 13 він має багато обмежень тому більшість його використовують лише для iOS 14 та вище.

SwiftUI використовує декларативний синтаксис, що допомагає набагато швидше та простіше будувати інтерфейс, код також виглядає простіше для читання, що також допомагає зберегти час на етапі підтримки.

```
import SwiftUI

struct Content : View {

    @State var model = Themes.listModel

    var body: some View {
        List(model.items, action: model.selectItem) { item in
            Image(item.image)
            VStack(alignment: .leading) {
                Text(item.title)
                Text(item.subtitle)
                    .color(.gray)
            }
        }
    }
}
```

Рисунок 28 – створення інтерфейсу за допомогою SwiftUI

3.2.3. Вибір інструменту для створення користувацького інтерфейсу

Після аналізу доступних бібліотек можемо їх порівняти між собою. Разом з UIKit у нас є можливість створювати графічний інтерфейс без особливих навичок програмування, використовуючи storyboard, але у такого підходу є негативні сторони. Використовуючи дрег н дроп при створенні інтерфейсу мають певний рівень обмежень: не можливо

реалізувати складний та динамічний дизайн, важко вирішувати мердж конфлікти, буде два джерела правди (в коді та в storyboard).

Що ж стосується створення інтерфейсу програмно, використовуючи UIKit, то порівняно з SwiftUI це набагато складніше, оскільки потрібно задавати все самому, бо використовується імперативний стиль.

Якщо ж говорити про SwiftUI, то це декларативний стиль, а отже розробка набагато швидша. Але з мінусів є те, що це ще досі нова технологія і має багато недоліків, одна з найбільших є проблема з навігацією.

Підсумувавши переваги та недоліки можемо сказати, що найкращим рішенням буде поєднання UIKit з SwiftUI. UIKit використовувати для комплексних дизайнів та навігації, а SwiftUI для чогось простого.

3.3. Організація кодової бази

Для розділення та можливості перевикористання було вирішено винести інтерфейсні компоненти у окремий фреймворк, що отримав назву ZNOshkaUI. Він включає в себе елементи з дизайн системи, що в майбутньому дозволить легко перевикористати всі елементи.

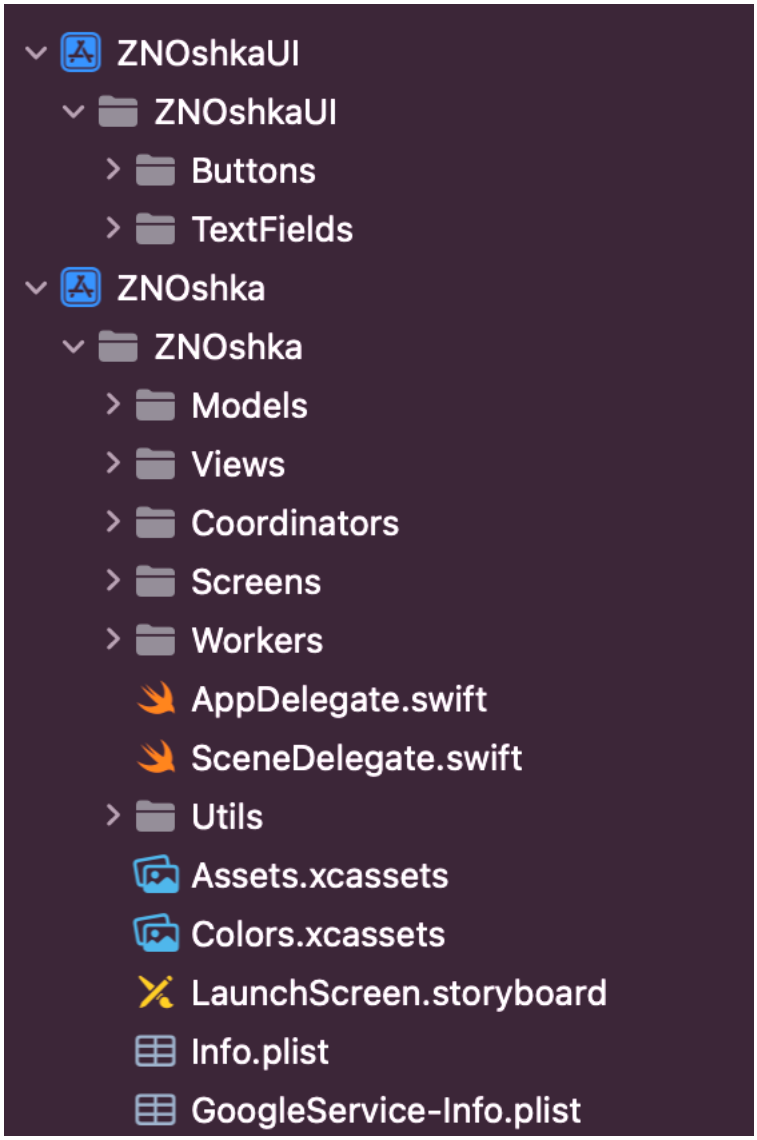

Рисунок 29 – структура проекту

Основний проект був розділений на декілька папок:

- Models – містить файли, що описують потрібні доменні структури моделей
- Views – містить файли, що є частинами користувацького інтерфейсу, але не підлягають перевикористанню.
- Coordinators – папка, що містить в собі усі координатори застосунку

Рисунок 30 – зміст папка Coordinators

- Screens – папка, що містить інші папки, відфільтровані відповідно до кожного екрану

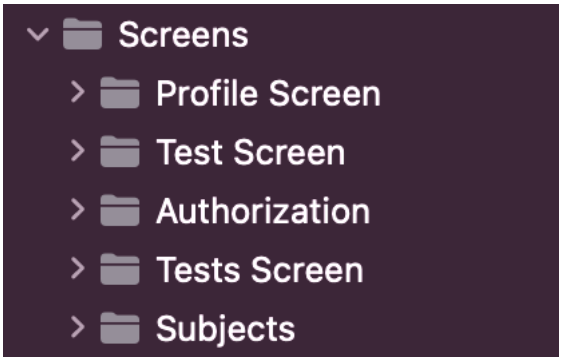

Рисунок 31 – вміст папки Screens

- Workers – папка, що містить файли, де реалізуються додаткові сервіси

3.4. Реалізація застосунку

Мобільний застосунок був успішно реалізований відповідно до раніше зазначеної архітектури MVVM-C. Пропоную розглянути реалізацію на прикладі екрану реєстрації.

RegisterView володіє власною в'ю моделлю RegisterViewViewModel, після того як ця в'ю модель встановлена, одразу ж створюються байндинги між текстовими полями та атрибутами в'ю моделі.

```
class RegisterView: UIView {

    var viewModel: RegisterViewViewModel? {
        didSet {
            setUpBindings()
        }
    }

    private var subscriptions = Set<AnyCancellable>()

    let nameTextField = DefaultTextField()
    let emailTextField = DefaultTextField()
    let passwordTextField = DefaultTextField()
    let signUpButton = DefaultButton()
```

Рисунок 32 – основні атрибути RegisterView.

Байндинги створюються з використанням Combine та CombineCocoa, що допомагає нам напряму отримати доступ до паблішера UIKit елементу.

```
private func setUpBindings() {
    guard let viewModel = viewModel else {
        return
    }

    nameTextField.textPublisher
        .compactMap { $0 }
        .assign(to: \.name, on: viewModel)
        .store(in: &subscriptions)

    emailTextField.textPublisher
        .compactMap { $0 }
        .assign(to: \.email, on: viewModel)
        .store(in: &subscriptions)

    passwordTextField.textPublisher
        .compactMap { $0 }
        .assign(to: \.password, on: viewModel)
        .store(in: &subscriptions)
}
```

Рисунок 33 – метод налаштування байндингів

Сама ж RegisterViewViewModel містить в собі потрібні їй моделі, в даному випадку це звичайні атрибути name, email та password, що мають тип String, та також володіє об’єктом типу RegisterCoordinatorProtocol та окремим об’єктом який імплементує інтерфейс AuthWorkerProtocol, що грає роль сервісу авторизації. До методу register() передається комплішн

хендлер, що повідомляє координатор, що реєстрація пройшла успішно і координатор переходить на подальший екран.

```
class RegisterViewViewModel {
    var name = ""
    var email = ""
    var password = ""

    weak var coordinator: RegisterCoordinatorProtocol?
    var authWorker: AuthWorkerProtocol?

    init(coordinator: RegisterCoordinatorProtocol?, authWorker: AuthWorkerProtocol) {
        self.coordinator = coordinator
        self.authWorker = authWorker
    }

    func register() {
        guard let authWorker = authWorker else {
            return
        }

        authWorker.register(email: email, password: password, username: name) { [weak self] in
            guard let self = self else { return }
            self.coordinator?.didRegister()
        }
    }
```

Рисунок 34 – RegisterViewViewModel

```
protocol AuthWorkerProtocol {
    var currentUser: User? { get }

    func register(email: String, password: String, username: String, completion: @escaping () -> Void )
    func signIn(email: String, password: String, completion: @escaping () -> Void)
    func signOut()
}
```

Рисунок 35 - AuthWorkerProtocol

Висновок

У цій роботі була проведена аналітична частина, що визначила подальший напрямок у якому рухатись та що розробляти. Основним функціоналом, який потрібно реалізувати стали: проходження тесті, можливість вивчення нового матеріалу та аналітична складова, що буде допомагати абітурієнтам визначати над чим їм варто більше працювати.

Також було розроблено дизайн систему елементів та власне сам дизайн застосунку за допомогою веб-застосунку Figma, спираючись на Apple Human Interface Guideline.

Для створення самого мобільного додатку Znoshka під iOS було використано мову Swift, яка чудово підійшла під потреби. Що ж стосується використаних бібліотек, то їх було вжито вкрай мала кількість:

- вбудований UIKit
- вбудований SwiftUI
- вбудований Combine
- CombineCocoa, що знаходиться у відкритому доступі
- власний ZnoshkaUI.

Також важливо зазначити, що в написані цієї роботи було використано архітектуру MVVM-C(Model-View-ViewModel-Coordinator), що дала чудову змогу створити кодову базу уникаючи утворення Massive View Controller-a.

В майбутньому додаток буде розвиватись, та одними з основних функцій які ще не були додані але будуть є: перегляд відео матеріалів, додання секції з коментарями та обговореннями, нагадування про навчання або ж відпочинок.

Список використаних джерел

1. Посилання веб сторінку Miro [Електронний ресурс] https://miro.com/index/
2. Посилання на веб сторінку Figma [Електронний ресурс] https://www.figma.com/design/
3. Посилання на огляд SPM [Електронний ресурс] https://www.swift.org/package-manager/
4. Посилання на огляд Xcode [Електронний ресурс] https://developer.apple.com/xcode/
5. Стаття зі статистикою по результатам ЗНО за останні 6 років [Електронний ресурс] https://www.slovoidilo.ua/2021/06/25/infografika/suspilstvo/rezultaty-zno-shist-rokiv-skilky-osib-zdaly-provalyly-testuvannya-osnovnyx-predmetiv
6. Стаття зі статистикою по результатам ЗНО від УЦОЯО [Електронний ресурс] https://zno.testportal.com.ua/opendata
7. Посилання на головну сторінку застосунку iLearn [Електронний ресурс] https://ilearn.org.ua/
8. Посилання на головну сторінку застосунку На всі двісті [Електронний ресурс] https://navsi200.com/
9. Посилання на головну сторінку застосунку Освіта.ua [Електронний ресурс] https://zno.osvita.ua/
10. Посилання на головну сторінку застосунку ПростеЗНО [Електронний ресурс] https://play.google.com/store/apps/details?id=ua.com.prostezno&hl=uk&gl=US
11. Посилання на головну сторінку застосунку Складу ЗНО [Електронний ресурс] https://apps.apple.com/us/app/%D1%81%D0%BA%D0%BB%D0%B0%D0%B4%D1%83-%D0%B7%D0%BD%D0%BE/id1277061354

Додатки

RegisterViewController.swift – файл екрану реєстрації

```
import UIKit
import Combine

class RegisterViewController: UIViewController {
    let viewModel: RegisterViewModel

    private let registerLabel = UILabel()
    private let registerView = RegisterView()
    @objc private let alreadyRegisteredButton = UIButton()
```

```
    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = .white
        setupRegisterView()
        setupRegisterLabel()
        setupAlreadyRegisteredButton()

        addTargets()
    }

    init(viewModel: RegisterViewModel) {
        self.viewModel = viewModel
        super.init(nibName: nil, bundle: nil)
    }

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

    private func setupRegisterView() {
        view.addSubview(registerView)
        registerView.translatesAutoresizingMaskIntoConstraints = false
        let registerViewViewModel = RegisterViewViewModel(coordinator:
viewModel.coordinator, authWorker: viewModel.authWorker)
        registerView.viewModel = registerViewViewModel

        NSLayoutConstraint.activate([
            registerView.centerYAnchor.constraint(equalTo: view.centerYAnchor),
            registerView.leadingAnchor.constraint(equalTo: view.leadingAnchor,
constant: 40),
            registerView.trailingAnchor.constraint(equalTo: view.trailingAnchor,
constant: -40),
            registerView.heightAnchor.constraint(equalToConstant: 280)
        ])
    }

    private func setupRegisterLabel() {
        view.addSubview(registerLabel)
        registerLabel.translatesAutoresizingMaskIntoConstraints = false

        registerLabel.font = UIFont.systemFont(ofSize:
Appearance.FontSize.authTitle)
        registerLabel.textColor = .black
        registerLabel.text = "Реєстрація"

        let padding: CGFloat = 8

        NSLayoutConstraint.activate([
            registerLabel.bottomAnchor.constraint(equalTo: registerView.topAnchor,
constant: -padding),
            registerLabel.leadingAnchor.constraint(equalTo:
registerView.leadingAnchor)
        ])
    }

    private func setupAlreadyRegisteredButton() {
        view.addSubview(alreadyRegisteredButton)
        alreadyRegisteredButton.translatesAutoresizingMaskIntoConstraints = false

        alreadyRegisteredButton.setTitle("Вже маєш аккаунту? Увійти", for: .normal)
        alreadyRegisteredButton.setTitleColor(.black, for: .normal)
        alreadyRegisteredButton.titleLabel?.font = UIFont.systemFont(ofSize: 14)

        let padding: CGFloat = 8

        NSLayoutConstraint.activate([
            alreadyRegisteredButton.topAnchor.constraint(equalTo:
registerView.bottomAnchor, constant: padding),
```

```
            alreadyRegisteredButton.centerXAnchor.constraint(equalTo:
view.centerXAnchor)
        ])
    }

    @objc
    private func alreadyRegisteredPressed() {
        viewModel.goToLogin()
    }

    private func addTargets() {
        alreadyRegisteredButton.addTarget(nil, action:
#selector(alreadyRegisteredPressed), for: .touchUpInside)
    }

    deinit {
        print("RegisterVC deinited")
    }
```

RegisterViewModel.swift – файл в'ю моделі екрану реєстрації

```
class RegisterViewModel {
    weak var coordinator: RegisterCoordinatorProtocol?
    let authWorker: AuthWorkerProtocol

    init(authWorker: AuthWorkerProtocol) {
        self.authWorker = authWorker
    }

    func goToLogin() {
        coordinator?.showLoginPage()
    }

    deinit {
        print("RegisterVM deinited")
    }
}
```

RegisterCoordinatorProtocol.swift – файл, в якому описується інтерфейс координатора реєстрації

```
protocol RegisterCoordinatorProtocol: Coordinator {
    func showLoginPage()
    func didRegister()
}
```

AuthorizationCoordinator.swift – файл, в якому реалізується координатор авторизації

```
import UIKit

protocol AuthorizationCoordinatorDelegate: AnyObject {
    func didLogIn(authCoordinator: AuthorizationCoordinator)
}

final class AuthorizationCoordinator: Coordinator {
```

```
    var navigationController: UINavigationController
    weak var parentCoordinator: Coordinator?
    var childCoordinators: [Coordinator] = []

    weak var delegate: AuthorizationCoordinatorDelegate?

    private var authWorker: AuthWorkerProtocol

    init(navigationController: UINavigationController, authWorker:
AuthWorkerProtocol) {
        self.navigationController = navigationController
        self.authWorker = authWorker
    }

    func start() {
        showRegistrationScreen()
    }

    func showRegistrationScreen() {
        let viewModel = RegisterViewModel(authWorker: authWorker)
        viewModel.coordinator = self
        let vc = RegisterViewController(viewModel: viewModel)

        navigationController.pushViewController(vc, animated: true)
    }

    deinit {
        print("AuthCoord deinited")
    }
}

extension AuthorizationCoordinator: RegisterCoordinatorProtocol {
    func didRegister() {
        delegate?.didLogIn(authCoordinator: self)
    }

    func showLoginPage() {
        let viewModel = LoginViewModel(authWorker: authWorker, coordinator: self)
        let vc = LoginViewController(viewModel: viewModel)

        navigationController.pushViewController(vc, animated: true)
    }
}

extension AuthorizationCoordinator: LoginCoordinatorProtocol {
    func didLogin() {
        delegate?.didLogIn(authCoordinator: self)
    }

    func showBackRegistrationPage() {
        navigationController.popViewController(animated: true)
    }
}
```

TestsViewController.swift – файл, в якому реалізовано екран, що дає змогу обрати бажаний тест.

```
import UIKit

class TestsViewController: UIViewController {

    var viewModel: TestsViewModel?

    private let itemsPerRow = 2
```

```
    private let sectionInset = UIEdgeInsets(top: 9, left: 18, bottom: 9, right: 18)

    var layout: UICollectionViewFlowLayout = {
        var layout: UICollectionViewFlowLayout = UICollectionViewFlowLayout()
        return layout
    }()

    var collectionView: UICollectionView?

    override func viewDidLoad() {
        super.viewDidLoad()
        navigationItem.title = "Тести"
        setup()
    }

    private func setup() {
        view.backgroundColor = .white
        setupCollectionView()
    }

    private func setupCollectionView() {
        collectionView = UICollectionView(frame: .zero, collectionViewLayout:
layout)
        guard let collectionView = collectionView else {
            return
        }
        collectionView.backgroundColor = Colors.mainPurple
        collectionView.dataSource = self
        collectionView.delegate = self
        collectionView.register(TestCell.self, forCellWithReuseIdentifier:
"TestCell")

        view.addSubview(collectionView)
        collectionView.translatesAutoresizingMaskIntoConstraints = false

        NSLayoutConstraint.activate([
            collectionView.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor),
            collectionView.leadingAnchor.constraint(equalTo: view.leadingAnchor),
            collectionView.trailingAnchor.constraint(equalTo: view.trailingAnchor),
            collectionView.bottomAnchor.constraint(equalTo: view.bottomAnchor)
        ])
    }

    deinit {
        print("TestsVC deinited")
    }

}

extension TestsViewController: UICollectionViewDelegate, UICollectionViewDataSource
{
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection
section: Int) -> Int {
        return 50
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt
indexPath: IndexPath) -> UICollectionViewCell {
        return collectionView.dequeueReusableCell(withReuseIdentifier: "TestCell",
for: indexPath)
    }

    func collectionView(_ collectionView: UICollectionView, didSelectItemAt
indexPath: IndexPath) {
        viewModel?.selected(index: indexPath.item)
    }
```

```
}

extension TestsViewController: UICollectionViewDelegateFlowLayout {
    func collectionView(_ collectionView: UICollectionView, layout
collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -
> CGSize {
        let paddingWidth = sectionInset.left * CGFloat((itemsPerRow + 1))
        let availableWidth = collectionView.frame.width - paddingWidth
        let itemWidth = availableWidth / CGFloat(itemsPerRow)
        return CGSize(width: itemWidth, height: 100)
    }

    func collectionView(_ collectionView: UICollectionView, layout
collectionViewLayout: UICollectionViewLayout, insetForSectionAt section: Int) ->
UIEdgeInsets {
        return sectionInset
    }

    func collectionView(_ collectionView: UICollectionView, layout
collectionViewLayout: UICollectionViewLayout, minimumInteritemSpacingForSectionAt
section: Int) -> CGFloat {
        return sectionInset.bottom
    }
}

extension TestsViewController {
    override func viewWillDisappear(_ animated: Bool) {
        if isMovingFromParent { viewModel?.disappears() }
    }
}
```

TestsCoordinatorProtocol.swift – протокол координатоку тестів

```
protocol TestsCoordinatorProtocol: Coordinator {
    var delegate: TestsCoordinatorDelegateProtocol? { get set }

    func disappears()
    func select(test: Test)
}
```

TestViewController.swift – файл, в якому реалізовано екран проходження тесту

```
import UIKit
import Combine

class TestViewController: UIViewController {

    var viewModel: TestViewModel?

    var taskChooserCollectionView: TaskChooserCollectionView = {
        let layout: UICollectionViewFlowLayout = UICollectionViewFlowLayout()
        layout.scrollDirection = .horizontal
        layout.sectionInset = UIEdgeInsets(top: 5, left: 5, bottom: 5, right: 5)
        layout.itemSize = CGSize(width: 40, height: 40)
        return TaskChooserCollectionView(frame: .zero, collectionViewLayout:
layout)
    }()
    var taskChooserDataSource: TaskChooserCollectionDataSource?

    var questionCollectionView: QuestionCollectionView = {
```

```
        let layout: UICollectionViewFlowLayout = UICollectionViewFlowLayout()
        layout.scrollDirection = .horizontal
        layout.sectionInset = UIEdgeInsets(top: 10, left: 10, bottom: 10, right:
10)
        return QuestionCollectionView(frame: .zero, collectionViewLayout: layout)
    }()

    var subscriptions = Set<AnyCancellable>()

    override func viewDidLoad() {
        super.viewDidLoad()

        setup()
        createBindings()
    }

    private func setup() {
        view.backgroundColor = .white
        setupTaskChooserCollectionView()
        setupQuestionCollectionView()
    }

    private func setupTaskChooserCollectionView() {
        taskChooserDataSource = TaskChooserCollectionDataSource(viewModel:
TaskChooserViewModel(questionCounter: viewModel?.test.questions.count ?? 0))
        taskChooserCollectionView.taskChooserDataSource = taskChooserDataSource

        view.addSubview(taskChooserCollectionView)
        taskChooserCollectionView.translatesAutoresizingMaskIntoConstraints = false

        NSLayoutConstraint.activate([
            taskChooserCollectionView.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor),
            taskChooserCollectionView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
            taskChooserCollectionView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
            taskChooserCollectionView.heightAnchor.constraint(equalToConstant: 60)
        ])
    }

    private func setupQuestionCollectionView() {
        if let viewModel = viewModel {
            questionCollectionView.viewModel =
QuestionCollectionViewModel(questions: viewModel.test.questions)
        }
        view.addSubview(questionCollectionView)
        questionCollectionView.translatesAutoresizingMaskIntoConstraints = false

        NSLayoutConstraint.activate([
            questionCollectionView.topAnchor.constraint(equalTo:
taskChooserCollectionView.bottomAnchor),
            questionCollectionView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
            questionCollectionView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
            questionCollectionView.bottomAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.bottomAnchor)
        ])
    }

    private func createBindings() {
        if let viewModel = viewModel {


taskChooserCollectionView.taskChooserDataSource?.viewModel.$selectedTaskCellIndex
                .removeDuplicates().sink { value in
                    viewModel.currentQuestionIndex = value
```

```
                }.store(in: &subscriptions)

            questionCollectionView.$currentCellIndex.removeDuplicates().sink {
value in
                viewModel.currentQuestionIndex = value
            }.store(in: &subscriptions)

            viewModel.$currentQuestionIndex.removeDuplicates().sink { [unowned
self] value in
                self.questionCollectionView.currentCellIndex = value
            }.store(in: &subscriptions)

            viewModel.$currentQuestionIndex.removeDuplicates().sink { [unowned
self] value in

self.taskChooserCollectionView.taskChooserDataSource?.viewModel.selectedTaskCellInd
ex = value
            }.store(in: &subscriptions)
        }
    }

}
```

DefaultTextField.swift – файл, в якому реалізовано звичайне поле для вводу

```
import UIKit

public class DefaultTextField: UITextField {

    private let padding = UIEdgeInsets(top: 0, left: 16, bottom: 0, right: 16)

    public override init(frame: CGRect) {
        super.init(frame: frame)
        setupAppearance()
        setUpBorders()
    }

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

    private func setupAppearance() {
        heightAnchor.constraint(equalToConstant: 44).isActive = true
        backgroundColor = .white
        textColor = .black
        attributedPlaceholder = NSAttributedString(
            string: "Placeholder",
            attributes: [NSAttributedString.Key.foregroundColor: UIColor.gray]
        )
    }

    private func setUpBorders() {
        layer.cornerRadius = 12
        layer.borderWidth = 1
        layer.borderColor = UIColor(displayP3Red: 209/255, green: 209/255, blue:
209/255, alpha: 0.3).cgColor
    }

    override open func textRect(forBounds bounds: CGRect) -> CGRect {
        return bounds.inset(by: padding)
    }

    override open func placeholderRect(forBounds bounds: CGRect) -> CGRect {
        return bounds.inset(by: padding)
    }
```

```
    override open func editingRect(forBounds bounds: CGRect) -> CGRect {
        return bounds.inset(by: padding)
    }
}
```