ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка застосунку для ознайомлення з тактико-технічними характеристиками зброї та техніки мовою Swift»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олександр СМАХТІН
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Олександр СМАХТІН

Керівник: Олена НЕГОДЕНКО
к.т.н., доцент

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Смахтіну Олександру Станіславовичу

1. Тема кваліфікаційної роботи: «Розробка застосунку для ознайомлення з тактико- технічними характеристиками зброї та техніки мовою Swift» керівник кваліфікаційної роботи к.т.н. доцент кафедри ІПЗ Олена НЕГОДЕНКО, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи перегляду технічних характеристик зброї та техніки, опис технічних етапів розробки.

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

1. Аналіз існуючих засобів для ознайомлення з військовою документацією

2. Вимоги до якості додатку.
3. Аналіз та визначення інструментів розробки.
4. Розробка застосунку

5. Перелік графічного матеріалу: *презентація*
    1. Аналіз аналогів.
    2. Вимоги до програмного забезпечення.
    3. Програмні засоби реалізації.
    4. Діаграма варіантів використання.
    5. Діаграма JSON.
    6. Навігаційна діаграма.
    7. Екранні форми.
    8. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Розробка основних вимог до якості додатку	14.03-25.03.2024	
4	Аналіз та визначення інструментів розробки	01.04-07.04.2024	
5	Розробка застосунку	8.04-05.05.2024	
6	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
7	Розробка демонстраційних матеріалів	06.05-12.05.2024	
8	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Олександр СМАХТІН
(підпис)

Керівник
кваліфікаційної роботи _____ Олена НЕГОДЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 53 стор., 1 табл., 37 рис., 6 джерел.

Мета роботи – спрощення процесу перегляду та пошуку характеристик військової техніки та зброї через розробку додатку, реалізованого на мові програмування Swift для платформи IOS.

Об'єкт дослідження – процес перегляду та ознайомлення характеристик військової техніки та зброї

Предмет дослідження – мобільний додаток для перегляду документації зброї та техніки

Короткий зміст роботи: Проведено порівняльний аналіз існуючих способів перегляду технічної документації до зброї та техніки. На основі аналізу, визначені вимоги до майбутнього додатку, серед яких відображення детальної інформації про техніку, можливість зберігання даних, а також інтуїтивний інтерфейс для користувача.

Програмне забезпечення реалізовано з використанням мови програмування Swift та фреймворку UIKit для платформи IOS. Обрано паттерн MVC для забезпечення структурованості та легкої модифікації. Використано систему контролю версій GitHub, редактор Xcode, а також вбудовану базу даних Core Data для зберігання інформації про обрані елементи зброї та техніки.

Розроблено мобільний застосунок єЗброя програмно реалізовані ключові функціональні можливості, зокрема: розділення елементів на розділи, пошук елементів, додавання елементів у розділ обрані, можливість звернутися у підтримку. Додаток завантажено в App Store. В роботі використано мову програмування Swift для підтримки операційних систем IOS, IDE Xcode для роботи з елементами коду на мові Swift, Core Data для збереження елементів у

розділ обрані.

Сферою використання застосунку є військова сфера, зокрема ознайомлення та перегляд військовослужбовцями характеристик військової зброї та техніки.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ДОДАТОК, ЄЗБРОЯ, SWIFT, UIKIT, CORE DATA, IOS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Swift - це мова програмування, розроблена компанією Apple для створення програмного забезпечення для платформ IOS, macOS, watchOS та tvOS. Swift є мовою з відкритим кодом, яка поєднує в собі високу продуктивність та безпеку типів з простотою та експресивністю.

UIKit - це фреймворк для розробки графічного інтерфейсу користувача в програмах для платформи IOS. Він надає широкий набір інструментів для створення інтерактивних та привабливих застосунків, включаючи елементи інтерфейсу, анімацію, обробку подій та багато іншого.

Xcode - це інтегроване середовище розробки (IDE) для програмування та розробки програмного забезпечення для платформ IOS, macOS, watchOS та tvOS. Воно містить широкий набір інструментів, включаючи редактор коду, компілятор, відладчик, інструменти для створення і тестування програм, а також інструменти для розгортання програм.

CoreData - це фреймворк для роботи з базами даних у програмах для платформи IOS та macOS. Він надає можливості для зберігання, витягування та маніпулювання даними в програмах, використовуючи об'єктно-орієнтований підхід до роботи з даними.

Entity - у контексті CoreData, Entity - це об'єктно-орієнтоване представлення даних в базі даних. Кожен об'єкт Entity відповідає запису в базі даних і містить

набір атрибутів, які описують цей запис.

JSON - це формат обміну даними, який використовується для передачі структурованих даних між різними системами. JSON є текстовим форматом, що базується на синтаксисі мови JavaScript, і легко зрозумілий як людьми, так і комп'ютерами.

Model - в архітектурі програмного забезпечення Model відповідає за представлення даних та бізнес-логіку програми. Він забезпечує доступ до даних і виконує операції з ними, а також забезпечує валідацію та перетворення даних.

View - в архітектурі програмного забезпечення View відповідає за відображення даних користувачу і обробку взаємодії з ним. Вона представляє інтерфейс користувача програми, включаючи всі елементи, які користувач може бачити і з якими може взаємодіяти.

Controller - в архітектурі програмного забезпечення Controller відповідає за управління взаємодією між Model та View. Він обробляє дії користувача, взаємодіє з Model для отримання та збереження даних, та оновлює View для відображення змін у даних.

VC - скорочення від ViewController

ВСТУП

В наш скрутний час, військові вимушені завжди мати з собою документацію для їх спецзасобів, зброї та техніки. Зазвичай такі дані зберігаються у довідниках, і книгах. Додаток дозволить спростити та автоматизувати цей процес за рахунок надання військовим набір інструментів для детального вивчення документації. У сучасному світі, де технології проникають у всі сфери нашого життя, мобільні додатки стають необхідними інструментами для спрощення та покращення різних аспектів нашого повсякденного і професійного життя. Однією з таких сфер є військова справа, де доступ до точної та актуальної інформації є критично важливим.

Актуальність теми:

З розвитком технологій у військовій сфері, зокрема в галузі зброї та техніки, зростає потреба в надійних та зручних інструментах для пошуку та аналізу характеристик обладнання. Забезпечення військових та фахівців з галузі важливою інформацією є ключовим елементом для успішного виконання завдань та прийняття обґрунтованих рішень.

Мобільний додаток для пошуку та аналізу інформації про характеристики зброї та техніки може відігравати важливу роль у забезпеченні зручного та швидкого доступу до необхідної інформації навіть у віддалених та польових умовах.

Цей додаток дозволить користувачам ефективно шукати та аналізувати технічні характеристики різних видів зброї та техніки, надавати доступ до актуальних даних, які допоможуть у прийнятті обґрунтованих рішень в сфері оборони та безпеки.

Такий додаток буде відповідати сучасним вимогам до зручного та ефективного доступу до інформації, забезпечуючи швидкий та точний аналіз характеристик зброї та техніки навіть у найвимогливіших умовах.

1. АНАЛІЗ ІСНУЮЧИХ ЗАСОБІВ ДЛЯ ОЗНАЙОМЛЕННЯ З ВІЙСЬКОВОЮ ДОКУМЕНТАЦІЄЮ

1.1 Wikipedia

Wikipedia - це вільна онлайн енциклопедія, яка містить широкий обсяг інформації з різних галузей знань, включаючи технічні характеристики зброї та техніки. Користувачі можуть знайти статті про конкретні види зброї, військові платформи, військові технології та багато іншого, що стосується військової сфери.

Для аналізу характеристик української зброї та техніки у Wikipedia можна використовувати наступні підходи:

1. Пошук за ключовими словами: Користувач може ввести назву конкретного зразка зброї чи техніки в пошукове поле Wikipedia. Наприклад, "БМП-1" або "Ан-225 Мрія". Результати пошуку зазвичай містять статті, де описуються технічні характеристики, історія розвитку, використання та інша важлива інформація.
2. Перегляд категорій: Wikipedia має розділи, присвячені різним категоріям статей. У випадку військової техніки, це може бути категорія "Військова техніка", "Зброя" або "Військова історія". В цих розділах користувач може знайти перелік статей, пов'язаних з певними аспектами української зброї та техніки.

Переваги Wikipedia для пошуку та аналізу характеристик української зброї та техніки:

1. Доступність: Wikipedia безкоштовно доступна для всіх користувачів Інтернету та має великий обсяг інформації.
2. Різноманітність джерел: Статті у Wikipedia часто містять посилання на джерела, що дозволяє користувачам перевірити достовірність інформації.
3. Широкий опис: Wikipedia зазвичай містить докладні статті з різних аспектів зброї та техніки, включаючи технічні характеристики, історію, використання та багато іншого.

Недоліки Wikipedia для пошуку та перегляду характеристик української зброї та техніки:

1. Недостатня достовірність: Хоча Wikipedia намагається забезпечити достовірну інформацію, іноді статті можуть містити помилки або неповну інформацію.
2. Перегляд характеристик: Wikipedia зазвичай надає загальну інформацію про зброю та техніку, а також перенасичена зайвою інформацією.

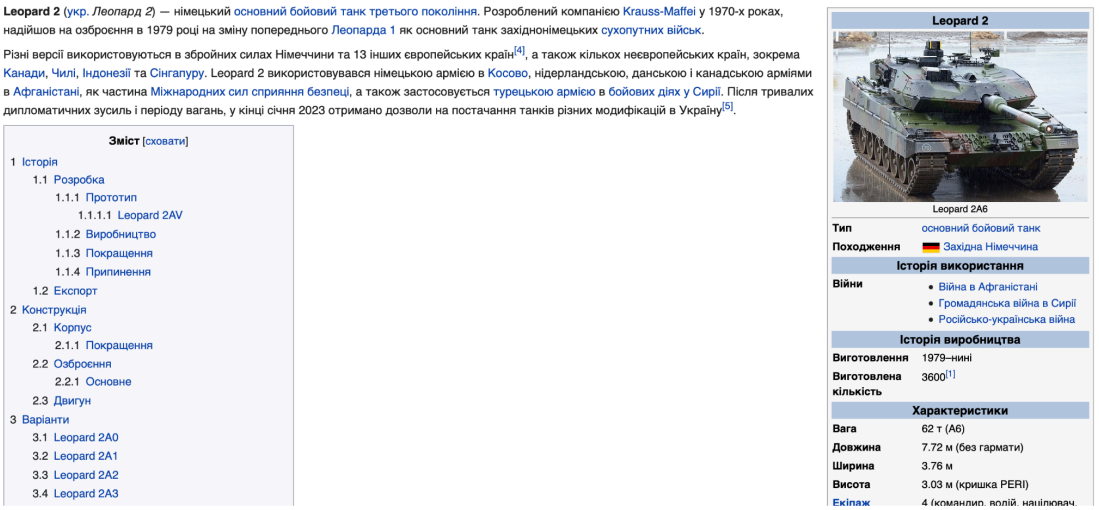


рис 1.1 Wikipedia

1.2 Ukrmillitary

Сайт "UkrMilitary" - це веб-ресурс, який містить інформацію про українську військову техніку та зброю. Користувачі можуть знайти дані про різні типи військової техніки, включаючи танки, бойові машини піхоти, авіаційну техніку та інше. Однак, варто враховувати деякі переваги та недоліки використання цього сайту для аналізу характеристик української зброї та техніки.

Переваги використання сайту "UkrMilitary" для аналізу характеристик української зброї та техніки:

1. Зібрана інформація: Сайт містить значну кількість інформації про різні види української військової техніки, що може бути корисною для аналізу.
2. Опис технічних характеристик: Для багатьох моделей зброї та техніки на сайті наведено докладні технічні характеристики, що дозволяє користувачам отримати повну картину про обладнання.

Недоліки використання сайту "UkrMilitary" для аналізу характеристик української зброї та техніки:

1. Застарілість інформації: Сайт рідко оновлюється новими даними, тому деяка інформація може бути застарілою або неактуальною.
2. Відсутній пошук елементів.

Тактико-технічні характеристики (ТТХ) 7,62-мм пістолета ТТ

Калібр, мм	7,62×25 мм
Принцип дії	віддача при короткому ході ствола, перекіс затвора
Вага пістолета (без набоїв), г	854
Вага пістолета із спорядженим магазином, г	940
Довжина, мм	195
Довжина ствола, мм	116
Висота, мм	130
Ширина, мм	30

рис 1.2 Ukrmillitary.com

1.3 Фізичні довідники

Фізичні посібники для зброї та техніки - це навчальні матеріали, які призначені для надання детальної інформації про конкретні види зброї, військової техніки та обладнання. Ці посібники зазвичай включають в себе ілюстрації, схеми, описи функцій та технічних характеристик, а також інструкції щодо експлуатації та обслуговування.

Переваги використання фізичних посібників для зброї та техніки:
1. Детальна інформація: Посібники зазвичай надають докладні технічні характеристики та інструкції щодо використання конкретних зразків зброї та техніки.
2. Візуальна підтримка: Ілюстрації та схеми допомагають краще зрозуміти будову та принцип дії обладнання.
3. Практичні поради: Деякі посібники можуть містити практичні поради щодо ефективного використання та обслуговування зброї та техніки.

Недоліки використання фізичних посібників для зброї та техніки:
1. Обмежений доступ: Фізичні посібники можуть бути обмеженими в доступі та розповсюдженні, особливо якщо вони випускаються обмеженими кількостями або підтримуються обмеженими організаціями.
2. Застаріла інформація: Посібники можуть старіти із часом, і їхні дані можуть бути неактуальними або не враховувати останніх технологічних або тактичних змін.
3. Відсутність інтерактивності: У порівнянні з веб-ресурсами або електронними джерелами інформації, фізичні посібники зазвичай не забезпечують можливості для активної взаємодії чи оновлення інформації.

1.4 Висновки

Після аналізу трьох джерел інформації про українську військову техніку та зброю - Wikipedia, сайту "UkrMilitary" та фізичних посібників, можна зробити наступний висновок:

Таблиця 1.1 - Порівняльна таблиця

Характеристики	Wikipedia	Ukrmillitary	Фізичний посібник	єЗброя
Пошук елементів	+	-	-	+
Оновлення інформації про зброю та техніку	+	-	-	+
Мобільність	Мобільна, веб-сервіс	Веб-сервіс	паперовий вигляд	мобільний застосунок
Не потребує підключення до інтернету	-	-	+	+
Додавання у обрані	-	-	-	+
Редагування обраних	-	-	-	+

Усі ці ресурси надають важливу інформацію про українську військову техніку та зброю, проте кожен має свої переваги та обмеження. Wikipedia відома своєю доступністю та широким охопленням тем, але не завжди має детальні дані та актуальну інформацію. Сайт "UkrMilitary", хоча має значну кількість інформації, часто страждає від неактуальності та застарілого дизайну, що ускладнює його використання. Фізичні посібники можуть бути корисними для глибшого розуміння технічних характеристик та інструкцій з експлуатації, але обмежуються доступністю та актуальністю інформації.

Варто зазначити, що всі ці ресурси окрім Wikipedia не мають мобільних реалізацій для зручного доступу до інформації на мобільних пристроях. Мобільна реалізація цих ресурсів могла б значно полегшити доступ до важливої інформації для користувачів, особливо для військових та фахівців з галузі оборони.

2. ВИМОГИ ДО ЯКОСТІ ДОДАТКУ

2.1 Вимоги до додатку

2.1.1 Легкість додатку

Додаток повинен не займати велику кількість простору на мобільному девайсі бажано до 10MB.

2.1.2 Автономна робота

Додаток має працювати і виконувати всі функціональні вимогу не маючи доступу до інтернету.

2.1.3 Підтримка

Додаток має працювати починаючи з версією IOS 13, для його роботи на більшій кількості приладів.

2.1.4 Операційна система

Операційна система - IOS, iPadOS, це дозволить додатку охопити більшу аудиторію.

2.2 Функціональні вимоги до додатку

2.2.1 Розділи

Додок має в собі мати окремі розділи зброї та техніки. Кожен з цих розділів

повинен поділяти на інші розділи в форматі: Зброя -> Стрілецька зброя -> Пістолети -> Glock 17.

2.2.2 Перегляд інформації

Перегляд даних та характеристик у додатку буде відбуватись за допомогою таблиці з двух колонок, де зліва - назва характеристики, справа - значення.

2.2.3 Пошук інформації

У додатку має бути реалізован екран для пошуку конкретних елементів зброї та техніки.

2.2.4 Додавання у обрані

Кожен елемент зброї та техніки повинен мати можливість бути доданим у розділ обраного.

2.2.5 Зворотній зв’язок

У додатку має бути реалізован функціонал для зв’язку з технічною підтримкою так для пропозиції та порад користувачів.

2.3 Use-case diagram

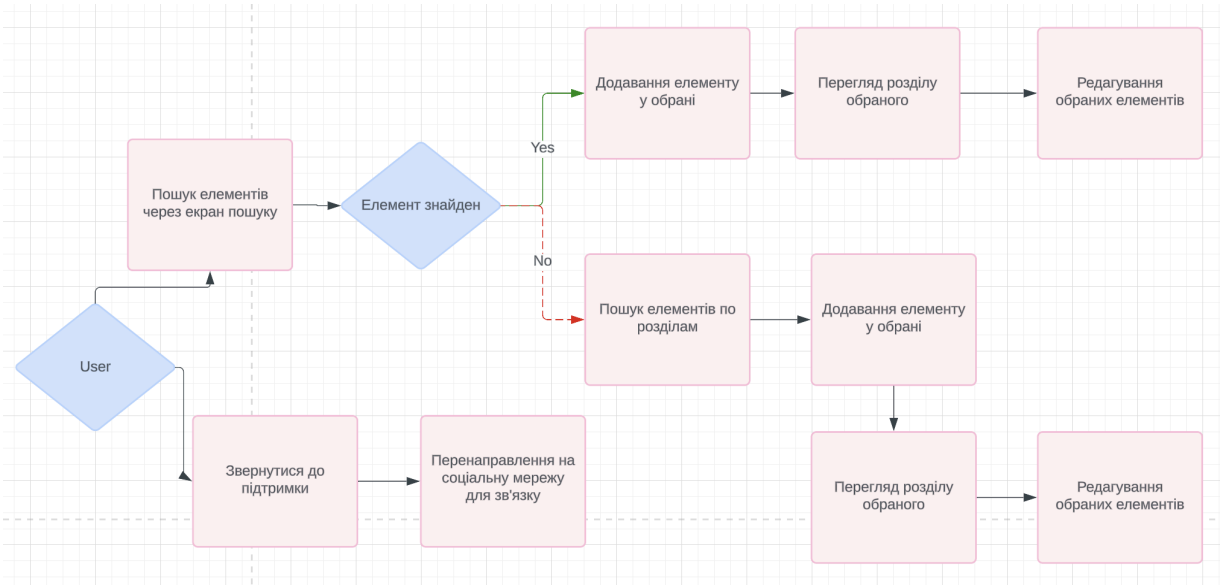


рис 2.1 Use-Case Diagram

3. АНАЛІЗ ТА ВИЗНАЧЕННЯ ІНСТРУМЕНТІВ РОЗРОБКИ

3.1 Інтрументи

3.1.1 Xcode

Xcode - це інтегроване середовище розробки (IDE) для створення програмного забезпечення для пристроїв Apple, таких як iPhone, iPad, Mac, Apple Watch та Apple TV. Воно розроблено компанією Apple і є основним інструментом для розробки програм для цих платформ.

Основні компоненти Xcode включають в себе:

1. Редактор коду: Xcode надає потужний редактор коду з підсвічуванням синтаксису, автодоповненням, перевіркою помилок та іншими корисними функціями. Редактор підтримує мови програмування, такі як Swift, Objective-C, C++, Python та інші.
2. Візуальні інтерфейси: Xcode має інтерфейсний конструктор, що дозволяє розробникам створювати інтерфейси користувача для своїх додатків шляхом перетягування та розміщення візуальних елементів, таких як кнопки, тексти, зображення тощо.
3. Інструменти тестування: Xcode має вбудовані інструменти для тестування розроблених додатків. Це включає в себе можливість запускати і відлажувати код, виконувати модульні тести, використовувати інструменти профілювання для виявлення проблем продуктивності тощо.
4. Інтеграція з іншими інструментами: Xcode інтегрується з іншими інструментами розробки від Apple, такими як Instruments (для профілювання додатків), Interface Builder (для створення інтерфейсів користувача), Simulator (для емуляції різних пристроїв), а також з різними сервісами розробки, такими

як Git для контролю версій.

Xcode є потужним інструментом для розробки програмного забезпечення для платформ Apple, і він широко використовується розробниками для створення різноманітних додатків, від ігор до корпоративних застосунків.

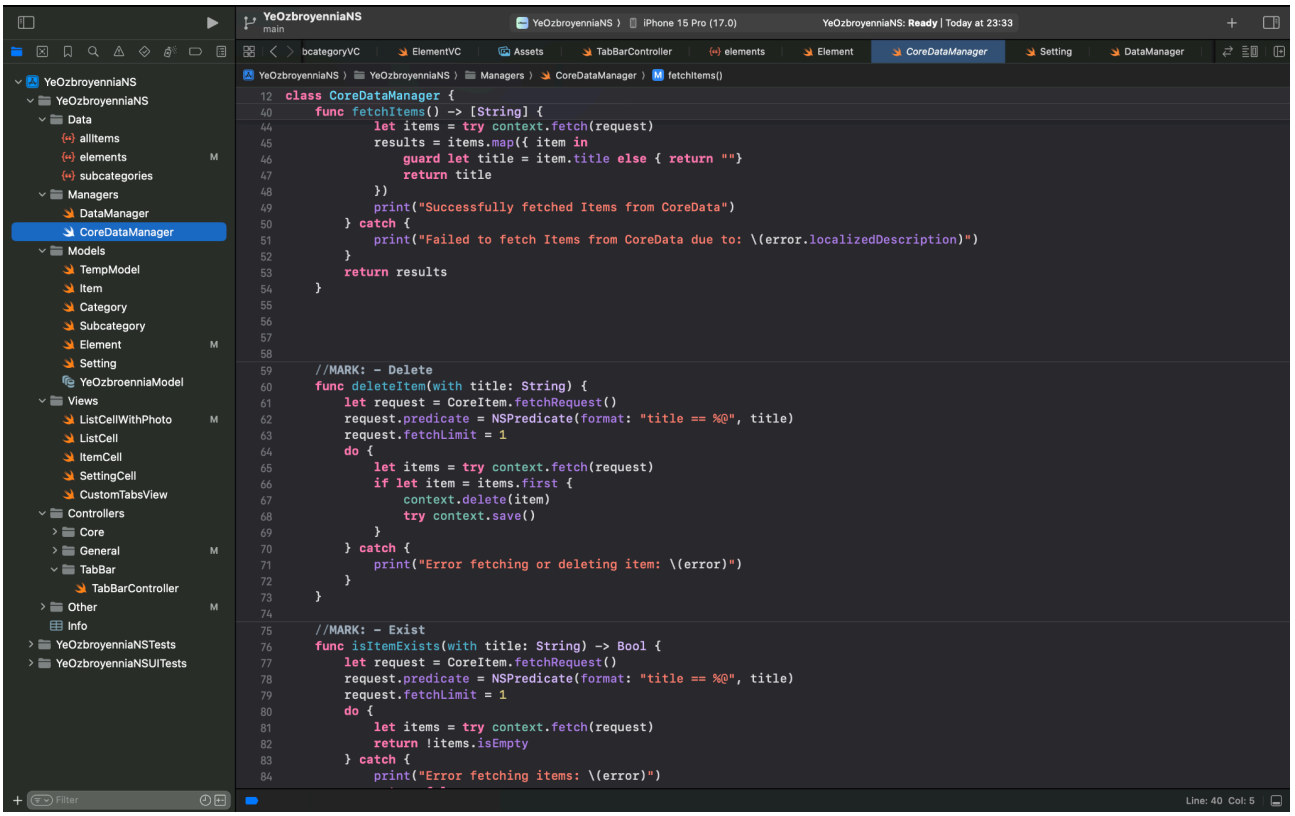


рис. 3.1 Xcode

3.1.2 Swift

Swift - це сучасна мова програмування, розроблена компанією Apple для створення програмного забезпечення для пристроїв, що працюють під управлінням операційних систем iOS, macOS, watchOS та tvOS. Її було випущено в 2014 році як альтернативу мові Objective-C, яка раніше використовувалася для розробки додатків для платформ Apple.

Swift широко використовується для розробки різноманітних додатків, від ігор та розважального програмного забезпечення до корпоративних застосунків та програм для взаємодії з різними сервісами та пристроями Apple. Вона продовжує здобувати популярність серед розробників завдяки своїм перевагам у простоті, безпеці та швидкодії.

3.1.3 UIKit

UIKit - це фреймворк для розробки інтерфейсів користувача для програм, призначених для операційних систем iOS, iPadOS, macOS, watchOS та tvOS компанії Apple. Цей фреймворк надає широкий набір інструментів та класів для створення візуально привабливих та інтерактивних інтерфейсів для додатків.

Основні компоненти та можливості UIKit:

1. UIView та UIViewController: UIKit базується на концепції UIView та UIViewController. UIView відповідає за відображення та взаємодію з візуальними елементами, такими як кнопки, тексти, зображення тощо. UIViewController використовується для управління життєвим циклом відображення та взаємодії з користувачем.

2. UIKit елементи керування: UIKit містить широкий набір вбудованих елементів керування, таких як UIButton, UILabel, UITextField, UITextView, UIImageView тощо. Ці елементи дозволяють створювати різноманітні інтерфейси користувача з використанням стандартних елементів.

3. Анімація та переходи: UIKit надає можливості для створення анімацій та переходів між екранами додатків. Розробники можуть використовувати

анімації для покращення взаємодії з користувачем та створення привабливого користувацького досвіду.

4. Auto Layout: UIKit надає систему Auto Layout для створення гнучких та адаптивних інтерфейсів, які підлаштовуються під різні розміри екранів та орієнтації пристроїв.

5. Gesture Recognizers: UIKit включає в себе різноманітні жестові визначники (gesture recognizers), такі як тапи, свайпи, пінчі, які дозволяють реагувати на різні жести користувача та створювати інтерактивні функції у додатках.

UIKit є основним інструментом для розробки інтерфейсів користувача для програм на платформах Apple. Він дозволяє розробникам швидко створювати високоякісні та інтерактивні інтерфейси, що відповідають стандартам дизайну та забезпечують зручну взаємодію з користувачем.

```
private let itemTitle: UILabel = {
    let lbl = UILabel()
    lbl.textColor = .label
    lbl.numberOfLines = 2
    lbl.textAlignment = .center
    lbl.font = UIFont.systemFont(ofSize: 20, weight: .bold)
    lbl.translatesAutoresizingMaskIntoConstraints = false
    return lbl
}()
```

Рис. 3.2 Створення текстового елементу на UIKit

3.1.4 Git та GitHub

Система контролю версій, вбудована в Xcode, та GitHub - це два різних інструменти для керування та спільної роботи над програмним кодом.

1. Система контролю версій в Xcode:

Xcode надає вбудовану підтримку системи контролю версій (СКВ), яка дозволяє розробникам зберігати та відстежувати зміни в програмному коді протягом часу. Основні можливості системи контролю версій в Xcode включають:

- *Створення репозиторію*: Розробники можуть створювати нові репозиторії прямо в Xcode або підключати існуючі репозиторії для роботи над кодом проекту.

- *Відстеження змін*: Xcode автоматично відстежує всі зміни, внесені до файлів проекту, і відображає їх відносно попередніх версій. Це дозволяє розробникам легко переглядати історію змін та відновлювати попередні версії коду.

- *Коміти та гілки*: Розробники можуть створювати коміти змін до репозиторію та створювати гілки для роботи над різними функціями або виправленнями безпеки паралельно.

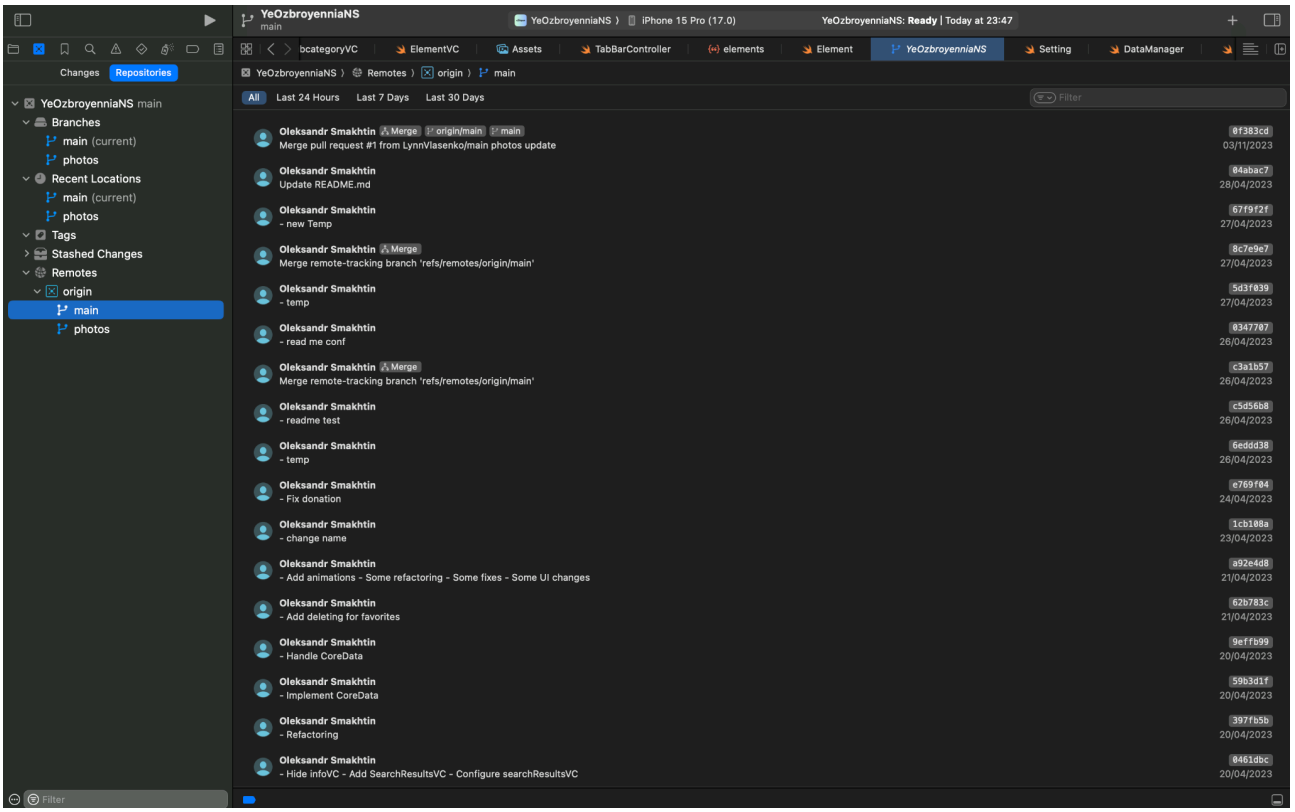


Рис. 3.3 Вбудована Git в Xcode

2. GitHub:

GitHub - це веб-сервіс для спільної роботи над програмним кодом з використанням системи контролю версій Git. Основні можливості GitHub включають:

- *Репозиторії та організації:* GitHub дозволяє користувачам створювати репозиторії для зберігання та управління кодом проектів. Він також підтримує організації, які дозволяють кільком користувачам спільно працювати над кодом.

- *Колаборація та код-рев'ю*: GitHub надає інструменти для спільної роботи над кодом, включаючи можливість запрошувати інших користувачів до спільного співпраці та відправлення запитів на злиття для обговорення змін та проведення код-рев'ю.

- *Проекти та задачі*: GitHub дозволяє створювати проекти та задачі для керування робочим процесом та відстеження виконання завдань у межах проекту.

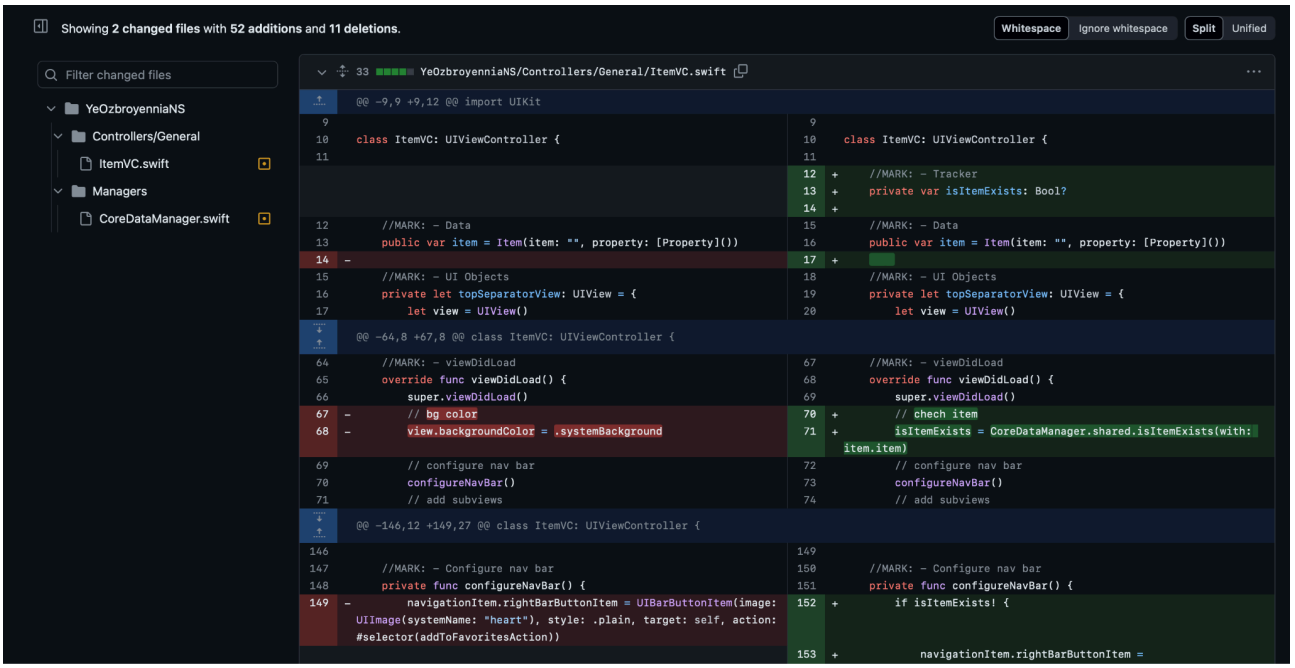


Рис. 3.4 Інтерфейс GitHub

3.1.4 Core Data

CoreData - це фреймворк, розроблений компанією Apple, який надає можливості для управління об'єктами та їхніми відносинами в додатках для платформ iOS, macOS, watchOS та tvOS. CoreData дозволяє зберігати, оновлювати, видаляти та запитувати дані в базі даних SQLite, яка використовується як основне сховище даних для додатків.

Основні компоненти та можливості CoreData:
1. Модель даних (Data Model): CoreData використовує модель даних для опису структури даних в додатку. Модель даних може містити сутності (entities), атрибути (attributes) та зв'язки (relationships) між сутностями. Ця модель використовується для створення об'єктів даних, які можна зберігати в базі даних.

2. Управління об'єктами (Object Management): CoreData надає можливості для створення, зберігання, оновлення та видалення об'єктів даних в базі даних. Вона автоматично стежить за змінами об'єктів та синхронізує їх з базою даних.

3. Запити (Queries): CoreData дозволяє виконувати складні запити до бази даних для отримання необхідної інформації. Це включає в себе можливість фільтрувати, сортувати та обмежувати результати запиту.

4. Кешування (Caching): CoreData автоматично кешує дані в оперативній пам'яті для швидкого доступу до них. Це допомагає зменшити час доступу до даних та покращити продуктивність додатка.

5. Міграція (Migration): CoreData надає можливості для міграції бази даних при зміні структури моделі даних. Це дозволяє розробникам безпечно вносити зміни до додатків та їхніх даних під час розвитку проектів.

CoreData дозволяє розробникам швидко та ефективно працювати з базою даних у своїх додатках. Вона є потужним інструментом для управління даними та забезпечення надійності та продуктивності додатків для платформ Apple.

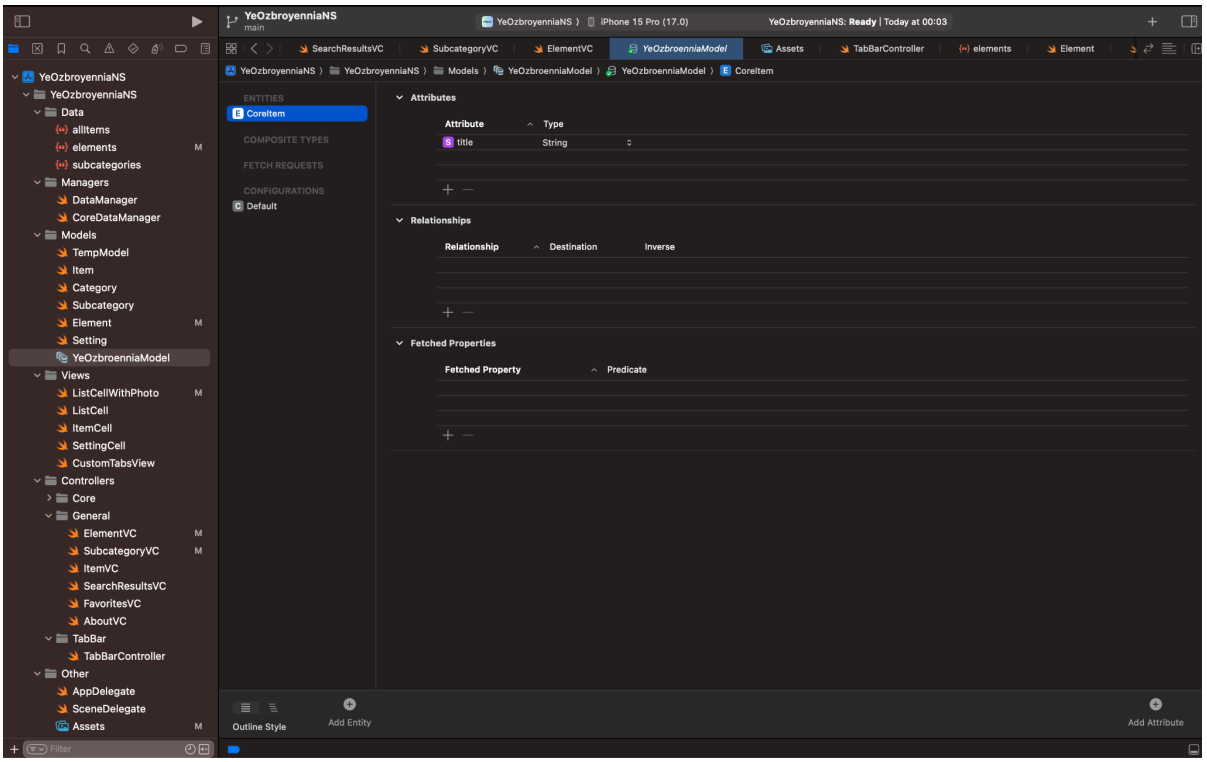


Рис. 3.5 Інтерфейс CoreData

3.2 Методи розробки

3.2.1 SOLID

SOLID - це абревіатура, яка означає п'ять базових принципів об'єктно-орієнтованого програмування (ООП), які були вперше сформульовані Робертом Мартіном. Ці принципи спрямовані на покращення дизайну програмного забезпечення та забезпечення його гнучкості, розширюваності та підтримки.

Основні принципи SOLID:

1. Принцип єдиної відповідальності (Single Responsibility Principle, SRP): Цей принцип стверджує, що клас або модуль повинен мати лише одну причину для зміни. Іншими словами, кожен клас або модуль повинен виконувати лише одну конкретну функцію або відповідати за одну частину функціональності програми. Це допомагає зберегти код чистим, зрозумілим та легко змінюваним.

2. Принцип відкритості-закритості (Open-Closed Principle, OCP): Цей принцип стверджує, що програмні сутності, такі як класи, модулі та функції, повинні бути відкритими для розширення, але закритими для змін. Це означає, що коли необхідно внести зміни в функціональність програми, це має робитися шляхом розширення існуючих сутностей, а не шляхом зміни їхнього внутрішнього коду.

3. Принцип підстановки Барбари Лісков (Liskov Substitution Principle, LSP): Цей принцип стверджує, що об'єкти базового класу повинні бути замінними об'єктами похідних класів без зміни коректності програми. Це означає, що класи-спадкоємці повинні зберігати інтерфейс базового класу та поводитися так само, як і об'єкти базового класу.

4. Принцип розділення інтерфейсів (Interface Segregation Principle, ISP): Цей принцип стверджує, що клієнти не повинні залежати від інтерфейсів, які вони не використовують. Він рекомендує розділити великі інтерфейси на більш малі та специфічні, щоб клієнти могли використовувати тільки ті методи, які їм потрібні.

5. Принцип інверсії залежностей (Dependency Inversion Principle, DIP): Цей принцип стверджує, що класи мають залежати від абстракцій, а не від конкретних реалізацій. Він також рекомендує використовувати високорівневі модулі, які залежать від абстракцій, а не від низькорівневих модулів.

3.2.2 MVC

MVC (Model-View-Controller) - це шаблон проектування програмного забезпечення, який використовується для розділення програми на три основні компоненти: модель (Model), представлення (View) та контролер (Controller). Цей підхід допомагає розділити логіку додатка, його представлення та управління користувацьким інтерфейсом, що полегшує розробку, розширення та тестування програм.

Основні компоненти MVC:

1. Модель (Model): Модель відповідає за обробку бізнес-логіки та даних додатка. Вона представляє собою структуру даних та правила, за якими ці дані можуть бути змінені та використані. Модель не залежить від представлення та контролера, що дозволяє їй бути повторно використаною та тестованою незалежно від інших компонентів.

2. Представлення (View): Представлення відповідає за відображення даних моделі користувачу та взаємодію з ним. Воно може бути у вигляді графічного інтерфейсу користувача, текстового виводу або будь-якого іншого способу представлення інформації. Представлення не має знати про існування моделі або контролера, воно лише відображає дані, отримані від моделі та реагує на події користувача.

3. Контролер (Controller): Контролер відповідає за обробку введення користувача та взаємодію з моделлю та представленням. Він приймає введення від користувача через представлення, виконує відповідні дії з моделлю та оновлює відображення згідно з результатами цих дій. Контролер дозволяє розділити логіку додатка на окремі компоненти та забезпечує відокремлення моделі та представлення від користувацького введення.

Модель-представлення-контролер є одним із найпоширеніших шаблонів проектування в розробці програмного забезпечення. Він дозволяє покращити структуру програми, зробити її більш гнучкою та легше розширюваною, що полегшує роботу розробників та підтримку програм.

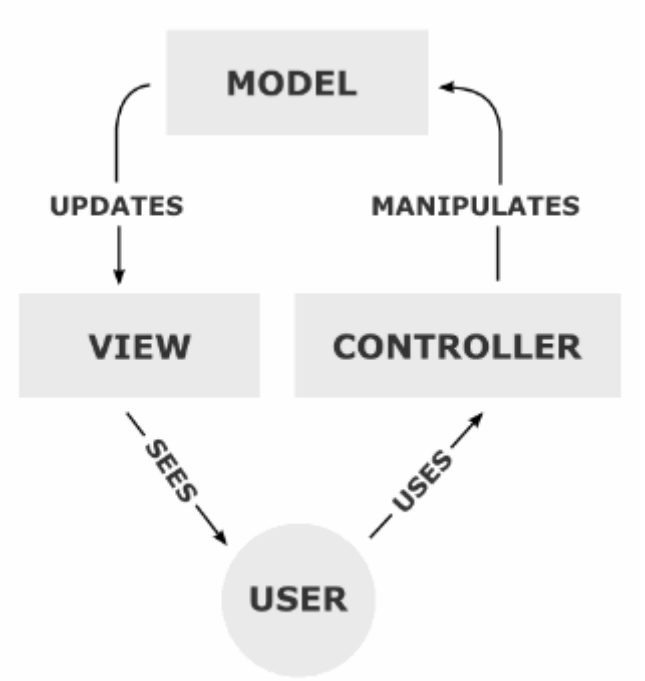


Рис 3.6 Принцип роботи MVC

3.2.3 Дані

Для економії розміру додатку та його легкості буде використовуватися локальний JSON файл, який буде містити в собі всю інформацію про елементи зброї та техніки.

JSON (JavaScript Object Notation) - це легкий формат обміну даними, який використовується для передачі структурованих даних між комп'ютерами. Він базується на синтаксисі JavaScript і зазвичай використовується для обміну даними між веб-серверами та клієнтськими додатками, а також для зберігання налаштувань та інших конфігураційних даних.

Основні характеристики JSON:

1. Простота читання та запису: JSON використовує зрозумілу для людини текстову форму, що полегшує його читання та редагування. Він використовує прості структури даних, такі як об'єкти та масиви, що робить його легким у використанні.

2. Підтримка різних типів даних: JSON підтримує різні типи даних, такі як рядки, числа, логічні значення, масиви та об'єкти. Це дозволяє створювати складні структури даних та передавати різноманітні дані між програмами.

3. Підтримка вбудованих об'єктів: JSON може включати в себе вкладені об'єкти та масиви, що дозволяє створювати складні структури даних з різним рівнем вкладеності.

4. Підтримка розширень: JSON дозволяє включати додаткові властивості або розширення у структуру даних за допомогою розширеного синтаксису, такого як "ключ: значення".

5. Легкість інтеграції з JavaScript: JSON базується на синтаксисі JavaScript, що робить його ідеальним для використання у веб-розробці, особливо з використанням AJAX та інших технологій взаємодії з веб-серверами.

JSON став широко використовуваним форматом для обміну даними через свою простоту, гнучкість та підтримку у багатьох мовах програмування. Він часто використовується у веб-розробці, веб-службах та інших сучасних додатках для передачі структурованих даних між клієнтськими та серверними компонентами.

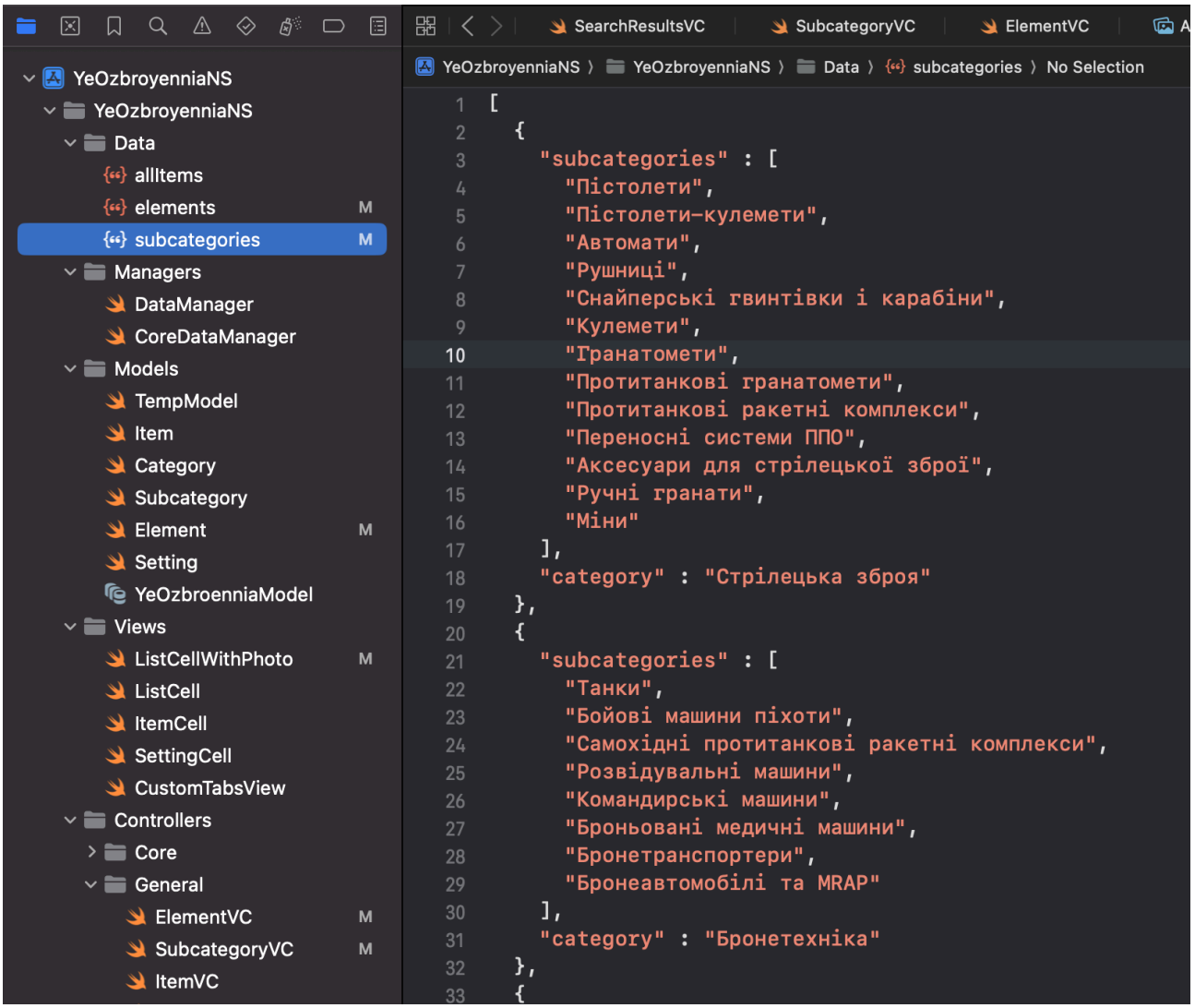


Рис 3.7 Локальний JSON файл

3.2.4 DataManagers

DataManager - це компонент програми, який відповідає за управління доступом до даних. Основна функція DataManager - це обробка даних, зчитування, запис та оновлення інформації у базі даних або інших джерелах даних. В додатку планується наявність двох менеджерів: менеджер роботи з JSON і менеджер по роботі з CoreData

Основні обов'язки DataManager можуть включати:
1. Зчитування даних: DataManager здійснює зчитування даних з визначених джерел, таких як база даних, API, файли тощо. Він може використовувати різні методи і запити для отримання необхідної інформації.

2. Збереження даних: Після отримання даних DataManager може здійснювати їх збереження у відповідних джерелах даних. Це може включати вставку нових записів, оновлення існуючих даних або видалення непотрібних даних.

3. Оновлення даних: Якщо інформація в базі даних або іншому джерелі змінюється, DataManager відповідає за оновлення цих даних у відповідних записах або ресурсах.

4. Логіка бізнес-логіки: DataManager може містити певну логіку бізнес-процесів, яка виконується під час обробки даних. Це може включати валідацію даних, обробку помилок, агрегацію даних тощо.

5. Керування даними: DataManager відповідає за управління даними та їхнім потоком у програмі. Він може вирішувати питання безпеки, доступу до даних, оптимізації запитів та інших аспектів управління даними.

DataManager є ключовим компонентом багатьох програм, особливо тих, які працюють з великим обсягом даних або потребують постійного доступу до зовнішніх джерел даних. Використання DataManager дозволяє розділити логіку доступу до даних від решти програми та забезпечити її ефективність та надійність.

```swift
import Foundation


class DataManager {
    static let shared = DataManager()

    let decoder = JSONDecoder()
    let encoder = JSONEncoder()

    //MARK: - Subcategory
    func getSubcategory(by category: String) -> Subcategory {
        var result = Subcategory(category: "", subcategories: [""])
        guard let path = Bundle.main.path(forResource: "subcategories", ofType: "json") else { return result}
        do {
            //let jsonData = try Data(contentsOf: URL(filePath: path))
            let jsonData = try Data(contentsOf: URL(fileURLWithPath: path))

            let subcategories = try decoder.decode([Subcategory].self, from: jsonData)
            let selectedSubcategory = subcategories.first { $0.category == category }
            result = selectedSubcategory!
        } catch {
            print(error.localizedDescription)
        }
        return result
    }
```

Рис 3.8 Менеджер для роботи з JSON

4. РОЗРОБКА ДОДАТКУ

4.1 Налаштування проекту

4.1.1 File management

Для більш зрозумілої навігації по проекту, було прийнято рішення поділити компоненти додатку на окремі розділи (папки).

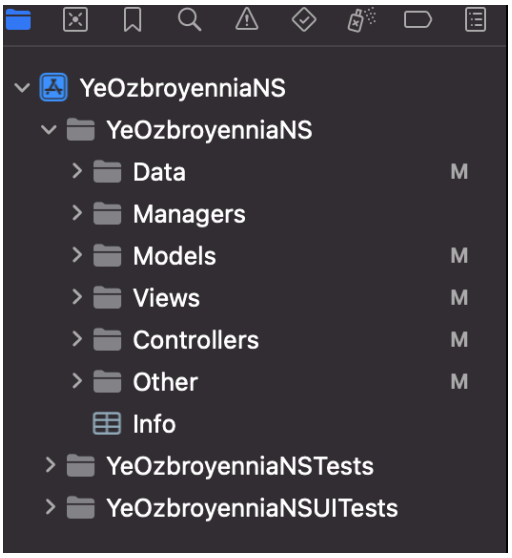


Рис. 4.1 Компоненти додатку

Більш детальніше про розділи:

- Data містить в собі всі необхідні для роботи локальні JSON файли
- Managers містить в собі менеджер для роботи з CoreData та менеджер для роботи з JSON файлами
- Models зберігає всі необхідні моделі даних
- Views містить в собі представлення View які було потрібно винести в коремий інтерфейс
- Controllers містить в собі всі ViewControllers які використовуються в додатку
- Other містить в собі необхідні файли для роботи IDE та життєвого циклу додатку а також асети із необхідними зображеннями.

4.1.2 Налаштування Git

Зазвичай, в Xcode локальний репозиторій створюється автоматично, при створенні проекту, тому нам лишається тільки створити remote репозиторій на платформі GitHub.

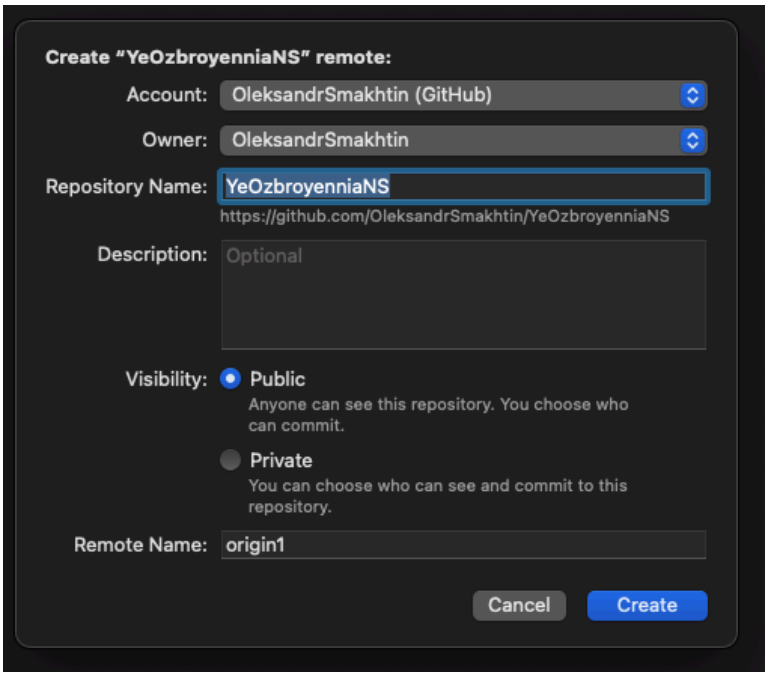


Рис. 4.1 Створення remote репозиторію

4.1.3 Налаштування CoreData

CoreData у додатку, в основному, буде використовуватися для збереження обраних елементів користувачів у розділ "Обране". Для цього знадобиться один Entity та один атрибут title (назва елементу).

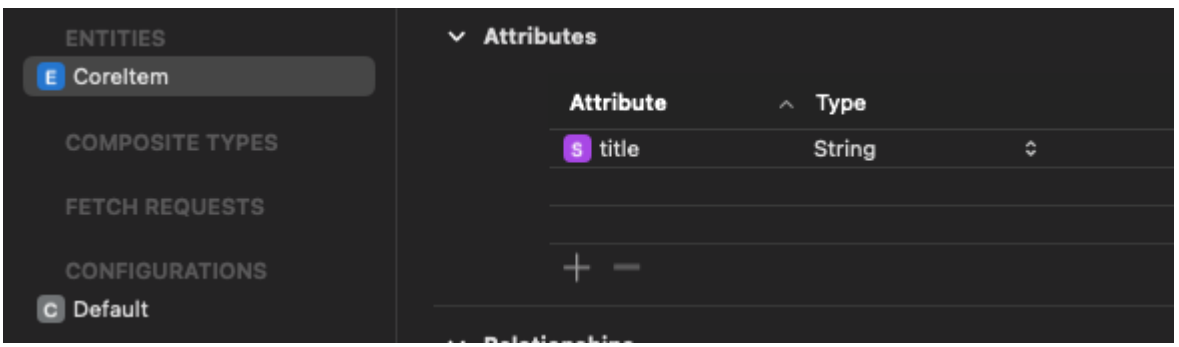


Рис. 4.2 Налаштована CoreData

4.1.4 Налаштування ассетів

В розділі ассети, зберігаються всі необхідні ресурси для візуальних змін в додатку (кольори, картинки, звуки, відео). В основному в додатку використовуються стандартні ресурси, які доступні в Xcode, але також додатку знадобляться кілька кастомних зображень та кольорів, а саме: іконки зброї, іконки техніки, кнопки назад, та фону.

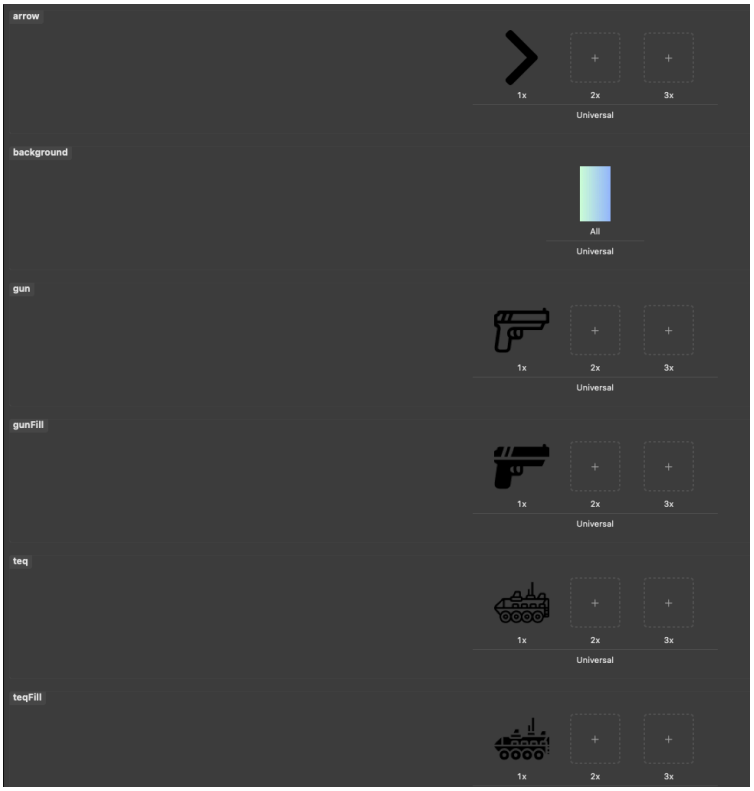


Рис. 4.4 Налаштована assets

4.2 Створення моделей

4.2.1 Setting

Модель Setting буде використовуватися в екрані налаштувань, вона складається з enum (перелік), де визначено тип налаштування. Самої структури даних яка має в собі тип, посилання на зображення, назву. Також був розроблений class SettingsData за принципом Singleton, в якому всього один метод, який буде повертати массив елементів налаштувань.

```
import Foundation

enum SettingType {
    case about
    case favorite
    case proposition
    case rate
    case share
    case thank
}

struct Setting {
    let type: SettingType
    let image: String
    let title: String
}

class SettingsData {
    static let shared = SettingsData()

    func getSettings() -> [Setting] {
        let settings = [
            Setting(type: .favorite, image: "heart", title: "Обране"),
            Setting(type: .about, image: "exclamationmark.circle", title: "Про додаток"),
            Setting(type: .proposition, image: "lightbulb", title: "Ваші пропозиції")
        ]
        return settings
    }
}
```

Рис. 4.5 - Модель Setting

4.2.2 Category

В даному випадку за модель Category був взят стандартний тип даних String. І також був визначений клас для повернення масиву категорій.

```
class Category {
    static let shared = Category()

    func getGunCategories() -> [String] {
        lazy var categories = [
            "Пістолети",
            "Пістолети-кулемети",
            "Автомати",
            "Рушниці",
            "Снайперські гвинтівки і карабіни",
            "Кулемети",
            "Гранатомети",
            "Протитанкові гранатомети",
            "Протитанкові ракетні комплекси",
            "Переносні системи ППО",
            "Аксесуари для стрілецької зброї",
            "Ручні гранати",
            "Міни"
        ]

        return categories
    }

    func getTeqCategories() -> [String] {
        lazy var categories = [
            "Бронетехніка",
            "Артилерія",
            "Системи зв'язку",
            "Радіолокаційні станції",
            "Системи протиповітряної оборони",
            "Системи берегової оборони",
            "Інженерна техніка",
            "Автомобілі",
            "Авіаційна техніка",
            "Морське озброєння",  // -
            "Оптичні прилади",  // -
            "Системи захисту"  // -
        ]

        return categories
    }
}
```

Рис. 4.6 Category

4.2.3 Subcategory

Цей код визначає структуру *Subcategory*, яка використовується для представлення підкатегорій з певної категорії. Структура містить дві властивості:

1. *category*: Рядок, який представляє назву головної категорії, до якої відносяться підкатегорії.

2. *subcategories*: Масив рядків, який містить перелік підкатегорій, що належать до вказаної категорії.

Ця структура використовується для серіалізації та десеріалізації даних з формату JSON (за допомогою протоколу Codable). Вона дозволяє зберігати дані про категорії та їх підкатегорії в структурованому форматі, який легко обробляти та передавати між компонентами програми.

```
import Foundation

struct Subcategory: Codable {
    let category: String
    let subcategories: [String]
}
```

Рис. 4.7 - Subcategory

4.2.4 Element

Модель *Element* представляє елемент з певної підкатегорії. Структура містить дві властивості:

1. *subcategory*: Рядок, який представляє назву підкатегорії, до якої відноситься елемент.

2. *items*: Масив рядків, який містить перелік елементів, що належать до вказаної підкатегорії.

```
struct Element: Codable {
    let subcategory: String
    let items: [String]
}
```

Рис. 4.8 Element

4.2.5 Item, Property

Ці моделі визначені за допомогою двох структур даних: *Item* і *Property*.

Структура *Item* містить дві властивості:

1. *item*: Рядок, що представляє назву елементу.
2. *property*: Масив об'єктів типу *Property*.

Структура *Property* використовується для представлення окремої властивості елементу. Вона містить дві властивості:

1. *name*: Рядок, що представляє назву властивості.
2. *value*: Рядок, що представляє значення властивості.

Ці структури дозволяють створювати та обробляти дані з властивостями для конкретних елементів, дозволяючи їх легку серіалізацію та десеріалізацію в форматі JSON. Вони будуть використані для відображень характеристик в таблиці для елементів.

```
struct Item: Codable {
    let item: String
    let property: [Property]
}

struct Property: Codable {
    let name: String
    let value: String
}
```

Рис. 4.9 Item, Property

4.3 Розробка менеджерів

4.3.1 DataManager

DataManager відповідає за роботу з даними зброї та техніки. Він містить різні методи для отримання інформації з JSON-файлів, які містять дані про підкатегорії, елементи та властивості.

1. Клас має статичну властивість *shared*, яка представляє собою єдиний екземпляр класу *DataManager*, з якого можна отримати доступ до методів класу.

2. Клас має два ініціалізатори для створення об'єктів JSONDecoder та JSONEncoder, які використовуються для серіалізації та десеріалізації об'єктів JSON.

3. Методи *getSubcategory, getElements, getItem* та *getRandom* використовуються для отримання даних з JSON-файлів, що містять інформацію про підкатегорії, елементи та предмети зброї та техніки.

4. Метод *getSearchedItems* використовується для пошуку предметів зброї та техніки за заданим запитом.

5. Приватний метод *getAll* використовується для отримання всіх предметів зброї та техніки.

Цей клас дозволяє легко отримувати, фільтрувати та шукати дані про зброю та техніку з JSON-файлів, що містяться в локальному JSON файлі.

```swift
class DataManager {
    static let shared = DataManager()

    let decoder = JSONDecoder()
    let encoder = JSONEncoder()

    //MARK: - Subcategory
    func getSubcategory(by category: String) -> Subcategory {
        var result = Subcategory(category: "", subcategories: [""])
        guard let path = Bundle.main.path(forResource: "subcategories", ofType: "json") else { return result}
        do {
            //let jsonData = try Data(contentsOf: URL(filePath: path))
            let jsonData = try Data(contentsOf: URL(fileURLWithPath: path))

            let subcategories = try decoder.decode([Subcategory].self, from: jsonData)
            let selectedSubcategory = subcategories.first { $0.category == category }
            result = selectedSubcategory!
        } catch {
            print(error.localizedDescription)
        }
        return result
    }

    //MARK: - Elements
    func getElements(by subcategory: String) -> Element {
        var result = Element(subcategory: "", items: [""], imagePath: [""])
        guard let path = Bundle.main.path(forResource: "elements", ofType: "json") else { return result}
        do {
            //let jsonData = try Data(contentsOf: URL(filePath: path))
            let jsonData = try Data(contentsOf: URL(fileURLWithPath: path))

            let elements = try decoder.decode([Element].self, from: jsonData)
            let selectedElement = elements.first { $0.subcategory == subcategory }
            result = selectedElement!
        } catch {
            print(error.localizedDescription)
        }
        return result
    }

    //MARK: - Item
    func getItem(by item: String) -> Item {
        var result = Item(item: "", property: [Property]())
        guard let path = Bundle.main.path(forResource: "allItems", ofType: "json") else { return result }
        do {
            //let jsonData = try Data(contentsOf: URL(filePath: path))
            let jsonData = try Data(contentsOf: URL(fileURLWithPath: path))

            let items = try decoder.decode([Item].self, from: jsonData)
            let selectedItem = items.first { $0.item == item }
```

Рис. 4.10 Фрагмент класу DataManager

4.3.2 CoreDataManager

CoreDataManager відповідає за роботу з базою даних Core Data. Основні функціональні можливості цього класу включають додавання, видалення та отримання даних з бази даних Core Data.

1. Клас має статичну властивість shared, яка представляє собою єдиний екземпляр класу *CoreDataManager*, з якого можна отримати доступ до методів класу.

2. Властивість *context* представляє контекст бази даних Core Data, який використовується для взаємодії з об'єктами у базі даних.

3. Метод *addItem* додає новий елемент до бази даних Core Data з заданим заголовком.

4. Метод *fetchItems* виконує запит до бази даних Core Data для отримання всіх елементів і повертає масив рядків, що містить заголовки цих елементів.

5. Метод *deleteItem* видаляє елемент з бази даних Core Data за його заголовком.

6. Метод isItemExists перевіряє, чи існує елемент з заданим заголовком в базі даних Core Data і повертає значення true, якщо такий елемент існує, або false, якщо ні.

Цей клас дозволяє легко взаємодіяти з базою даних Core Data для збереження, видалення та отримання даних.

```swift
class CoreDataManager {
    static let shared = CoreDataManager()

    //MARK: - Context
    let context = (UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext

    //MARK: - Add
    func addItem(with title: String) {
        let item = CoreItem(context: context)
        item.title = title
        do {
            try context.save()
            print("Successfully saved Item to CoreData")
        } catch {
            print("Failed to add Item to CoreData due to: \(error.localizedDescription)")
        }
    }

    //MARK: - Fetch
    func fetchItems() -> [String] {
        var results = [String]()
        let request = CoreItem.fetchRequest()
        do {
            let items = try context.fetch(request)
            results = items.map({ item in
                guard let title = item.title else { return ""}
                return title
            })
            print("Successfully fetched Items from CoreData")
        } catch {
            print("Failed to fetch Items from CoreData due to: \(error.localizedDescription)")
        }
        return results
    }

    //MARK: - Delete
    func deleteItem(with title: String) {
        let request = CoreItem.fetchRequest()
        request.predicate = NSPredicate(format: "title == %@", title)
        request.fetchLimit = 1
        do {
            let items = try context.fetch(request)
            if let item = items.first {
                context.delete(item)
                try context.save()
            }
        } catch {
            print("Error fetching or deleting item: \(error)")
        }
    }

    //MARK: - Exist
    func isItemExists(with title: String) -> Bool {
        let request = CoreItem.fetchRequest()
```

Рис. 4.11 Фрагмент класу CoreDataManager

4.4 Розробка допоміжних View

4.4.1 SettingsCell

SettingCell є підкласом UITableViewCell і відповідає за відображення одного рядка в таблиці на екрані. Основні функціональні можливості цього класу включають створення та налаштування рядка таблиці з відповідними UI-елементами.

1. Клас має статичну властивість *identifier*, яка представляє собою ідентифікатор рядка таблиці і використовується для переадресації клітинок.

2. В класі визначені приватні властивості *titleImageView*, *titleLbl*, *disclousereIndicator*, які представляють собою UI-елементи (UIImageView та UILabel) для відображення зображення, тексту та індикатора додаткової інформації в рядку таблиці.

3. Визначені методи *init(style:reuseIdentifier:)* та required *init?(coder:)* використовуються для ініціалізації класу та вимагають обов'язкового визначення.

4. Методи *addSubviews()* та *applyConstraints()* використовуються для додавання UI-елементів до контенту клітинки та встановлення їх обмежень відповідно.

5. Метод *configure(with:)* використовується для налаштування вмісту клітинки на основі об'єкта *Setting*, переданого в якості аргументу.

Загалом, клас *SettingCell* визначає розмітку та зовнішній вигляд рядка таблиці та дозволяє легко налаштовувати вміст цього рядка на основі даних.

```swift
class SettingCell: UITableViewCell {

    //MARK: - Identifier
    static let identifier = "SettingCell"

    //MARK: - UI Objects
    private let titleImageView: UIImageView = {
        let imageView = UIImageView()
        imageView.tintColor = .label
        imageView.translatesAutoresizingMaskIntoConstraints = false
        return imageView
    }()

    private let titleLbl: UILabel = {
        let lbl = UILabel()
        lbl.textColor = .label
        lbl.numberOfLines = 0
        lbl.translatesAutoresizingMaskIntoConstraints = false
        return lbl
    }()

    private let disclousereIndicator: UIImageView = {
        let imageView = UIImageView()
        var image = UIImage(named: "arrow", in: nil, with: UIImage.SymbolConfiguration(pointSize: 4))
        image = image?.withTintColor(.label, renderingMode: .automatic)
        imageView.image = image
        imageView.translatesAutoresizingMaskIntoConstraints = false
        return imageView
    }()

    //MARK: - Init
    override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
        // bg color
        backgroundColor = .clear
        // selection
        selectionStyle = .none
        // add subviews
        addSubviews()
        // apply constraints
        applyConstraints()
    }

    //MARK: - Required init
    required init?(coder: NSCoder) {
        fatalError()
    }

    //MARK: - Add subviews
    private func addSubviews() {
        contentView.addSubview(titleImageView)
        contentView.addSubview(titleLbl)
        contentView.addSubview(disclousereIndicator)
    }
```

Рис. 4.11 Фрагмент класу SettingsCell

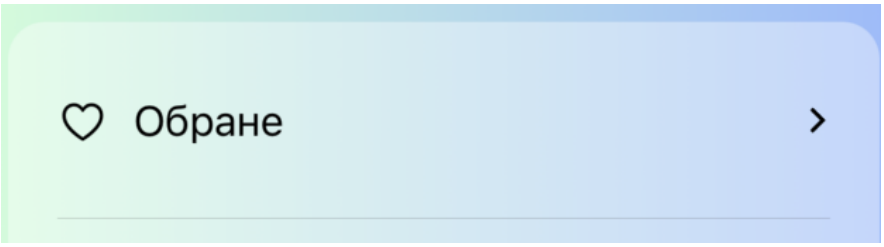


Рис. 4.12 Візуальне представлення SettingsCell

4.4.2 ListCell

Клас *ListCell* визначає вигляд та розмітку одного рядка в таблиці. Основні елементи цієї комірки включають *titleLbl* (мітка для відображення заголовку) та *disclousereIndicator* (зображення для показу індикатора). Також мається вигляд *bottomSeparatorView*, який може використовуватися для відображення роздільної лінії між комірками.

Загалом, клас `ListCell` визначає вигляд та поведінку клітинок у списку та надає можливість легко налаштовувати їх вміст на основі даних.

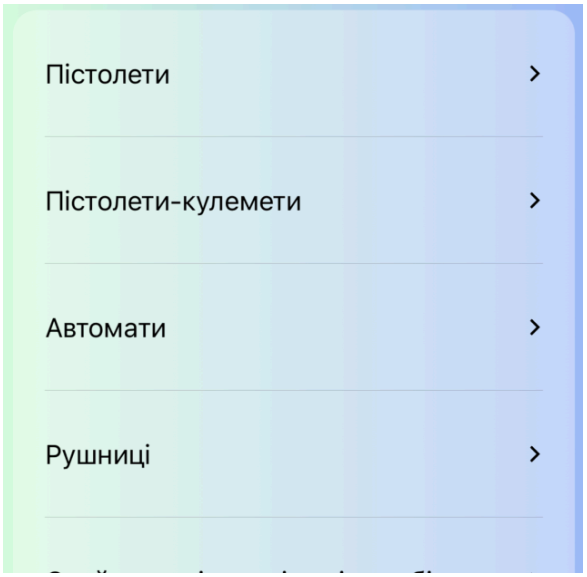


Рис. 4.13 Візуальне представлення комірок ListCell

4.4.3 ItemCell

Клас *ItemCell* визначає вигляд та розмітку одного рядка в таблиці для відображення елемента із його назвою та значенням. Основні елементи цієї комірки включають *nameLbl* (мітка для відображення назви) та *valueLbl* (мітка для відображення значення).

Загалом, клас *ItemCell* визначає вигляд та поведінку клітинок у списку та надає можливість легко налаштовувати їх вміст на основі даних.

Рис. 4.14 Візуальне представлення комірок ItemCell

4.5 Розробка ViewControllers

4.5.1 TabBarController

Цей клас, *TabBarController*, є підкласом *UITabBarController*, який використовується для управління вкладками інтерфейсу користувача. Основна мета цього класу - налаштувати та керувати таббаром, який містить чотири вкладки.

1. У класі визначено властивість *selectedTab*, яка відстежує індекс обраної вкладки. Коли ця властивість змінюється, виконується анімація переміщення індикатора під вибраною вкладкою.

2. Властивості *centerXAnchorsForIndicator* та *topAnchorsForIndicator* використовуються для зберігання обмежень для індикатора, які визначають його позицію в таббарі.

3. Визначено UI-елемент *indicatorView*, який представляє собою візуальний індикатор для позначення обраної вкладки.

4. У методі *viewDidLoad* налаштовуються контролери, що містяться на кожній вкладці, а також налаштовується зовнішній вигляд таббару та викликається метод *configureTabBar*().

5. У методі *configureTabBar*() виконується налаштування зовнішнього вигляду таббару, включаючи колір тексту, додавання індикатора та застосування обмежень для нього.

6. Перевизначений метод *tabBar(_:didSelect:)* відстежує обрану вкладку та викликає анімацію для переміщення на нову вкладку, а також змінює значення *selectedTab*.

7. Метод *animateToTab(toIndex:)* виконує анімацію переходу між вкладками.

В цілому, цей клас налаштовує та керує таббаром у додатку, забезпечуючи зручну навігацію користувача між різними частинами додатку.

```swift
class TabBarController: UITabBarController {

    //MARK: - Tracker
    private var selectedTab = 0 {
        didSet {
            for i in 0...3 {

                UIView.animate(withDuration: 0.3, delay: 0, options: .curveEaseOut) { [weak self] in
                    self?.centerXAnchorsForIndicator[i].isActive = i == self?.selectedTab ? true : false
                    self?.tabBar.layoutIfNeeded()
                } completion: { _ in }

            }
        }
    }

    //MARK: - Array of constraints for indicator
    private var centerXAnchorsForIndicator: [NSLayoutConstraint] = []
    private var topAnchorsForIndicator: [NSLayoutConstraint] = []

    //MARK: - UI Objects
    var indicatorView: UIView = {
        let view = UIView()
        view.backgroundColor = .black.withAlphaComponent(0.7)
        view.layer.cornerRadius = 2.5
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()


    //MARK: - viewDidLoad
    override func viewDidLoad() {
        super.viewDidLoad()
        // bg color
        view.backgroundColor = .systemBackground
        // configure nav bar
          //configureTabBar()
        // set controllers
        let vc1  = UINavigationController(rootViewController: GunVC())
        vc1.tabBarItem.image = UIImage(named: "gun")
        vc1.tabBarItem.selectedImage = UIImage(named: "gunFill")
        vc1.tabBarItem.imageInsets = UIEdgeInsets(top: 18, left: 18, bottom: 18, right: 18)
        vc1.tabBarItem.title = "Зброя"

        let vc2  = UINavigationController(rootViewController: TechniqueVC())
        vc2.tabBarItem.image = UIImage(named: "teq")
        vc2.tabBarItem.selectedImage = UIImage(named: "teqFill")
        vc2.tabBarItem.imageInsets = UIEdgeInsets(top: 14, left: 14, bottom: 14, right: 14)
        vc2.tabBarItem.title = "Техніка"

        let vc3 = UINavigationController(rootViewController: SearchVC())
        vc3.tabBarItem.image = UIImage(systemName: "magnifyingglass")
        vc3.tabBarItem.selectedImage = UIImage(systemName: "magnifyingglass", withConfiguration: UIImage.SymbolConfiguratio
```

Рис. 4.15 Фрагмент класу TabBarController

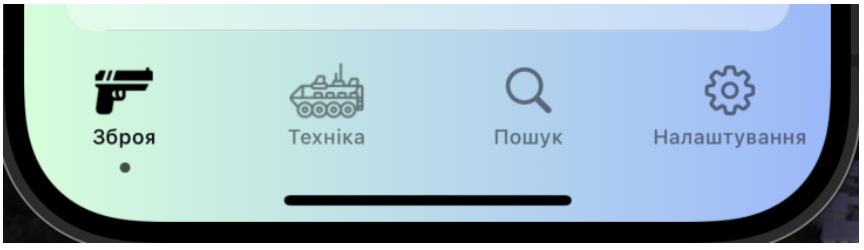


Рис. 4.16 Візуальне представлення класу TabBarController

4.5.2 AboutVC

Клас *AboutVC* відображає інформацію про додаток. Він має текстове поле *aboutTextView* для відображення опису додатка, підвид *cornerView* з круглими кутами, що оточує *aboutTextView*, та підвид bgImageView для фону. Навігаційна панель налаштовується на "Про додаток". Відсутній таббар.

4.5.3 FavoritesVC

Клас *FavoritesVC* відображає список обраних елементів. Він містить масив *favorites*, таблицю *listTable* для відображення елементів, мітку *isEmptyLbl* для відображення порожнього списку, а також підвид *bgImageView* для фону. Налаштовує навігаційну панель та приховує таббар. Методи делегування для таблиці обробляють відображення, вибір та видалення елементів.

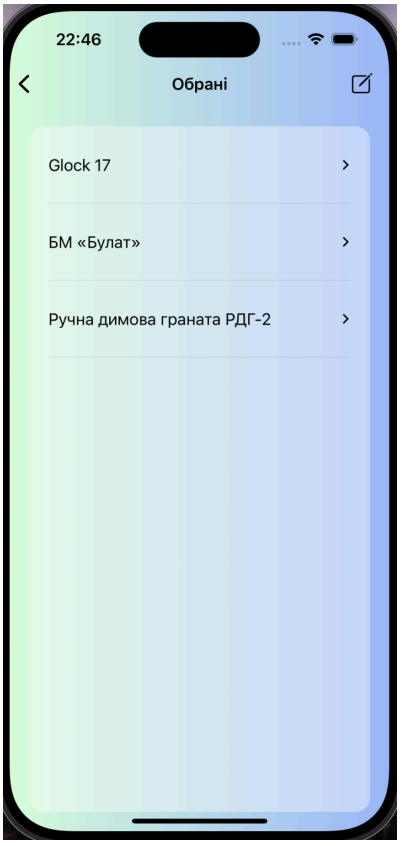


Рис. 4.17 Візуальне представлення розділу Обрані

4.5.4 ItemVC

Клас *ItemVC* відображає деталі про певний елемент. Він містить дані про елемент у властивості *item* типу *Item*, індикатор наявності елемента *isItemExists*, роздільник *topSeparatorView*, назву елемента *itemTitle*, список властивостей у *itemTable* та кнопку у навігаційній панелі для додавання або видалення з обраних. Фон сторінки встановлюється за допомогою *bgImageView*. Відображається кнопка "heart.fill" або "heart" в залежності від того, чи є елемент серед обраних.

```
class ItemVC: UIViewController {
    //MARK: - Tracker
    private var isItemExists: Bool?

    //MARK: - Data
    public var item = Item(item: "", property: [Property]())

    //MARK: - UI Objects
    private let topSeparatorView: UIView = {
        let view = UIView()
        view.backgroundColor = .clear
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    private let underlineView: UIView = {
        let view = UIView()
        view.backgroundColor = .white.withAlphaComponent(0.7)
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    private let itemTitle: UILabel = {
        let lbl = UILabel()
        lbl.textColor = .label
        lbl.numberOfLines = 2
        lbl.textAlignment = .center
        lbl.font = UIFont.systemFont(ofSize: 20, weight: .bold)
        lbl.translatesAutoresizingMaskIntoConstraints = false
        return lbl
    }()

    private let circleContentView: UIView = {
        let view = UIView()
        view.layer.cornerRadius = 15
        view.backgroundColor = UIColor(named: "tableColor")
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    private let itemTable: UITableView = {
        let table = UITableView()
        table.backgroundColor = .clear
        table.layer.cornerRadius = 5
        table.register(ItemCell.self, forCellReuseIdentifier: ItemCell.identifier)
        table.showsVerticalScrollIndicator = false
        table.translatesAutoresizingMaskIntoConstraints = false
        return table
    }()

    private let bgImageView: UIImageView = {
        let imageView = UIImageView()
        imageView.image = UIImage(named: "background")
```

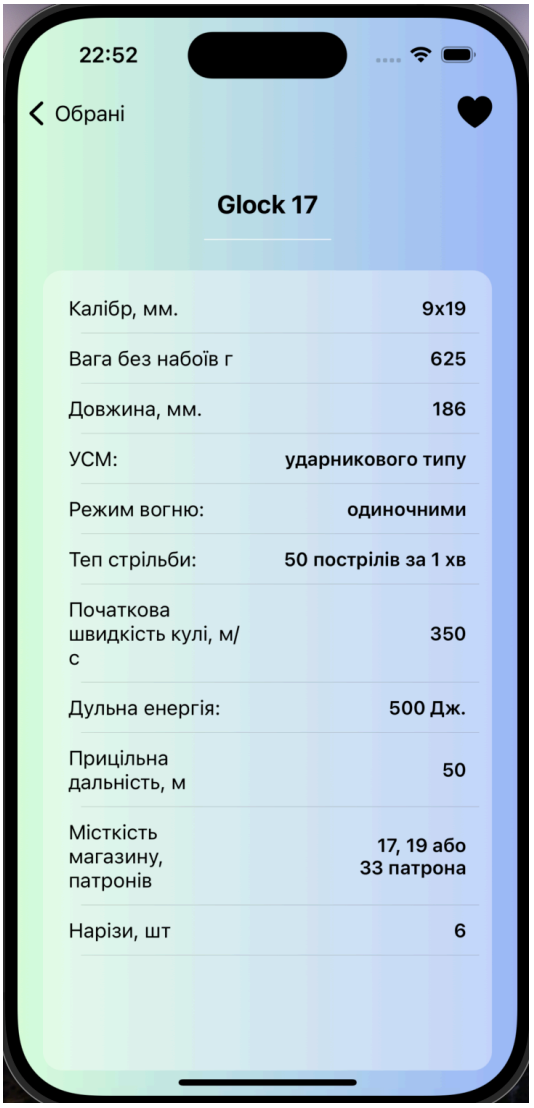


Рис. 4.17-18 Фрагмент коду ItemVC та його візуальне представлення

4.5.5 SubcategoryVC

Клас *SubcategoryVC* відображає список підкатегорій у вигляді таблиці. Він містить дані про підкатегорії у властивості *subcategory*. Фон сторінки встановлюється за допомогою *bgImageView*. Відображається роздільник *topSeparatorView* та список підкатегорій у *listTable*. Назва підкатегорії встановлюється як заголовок навігаційної панелі через метод *configureNavBar*.

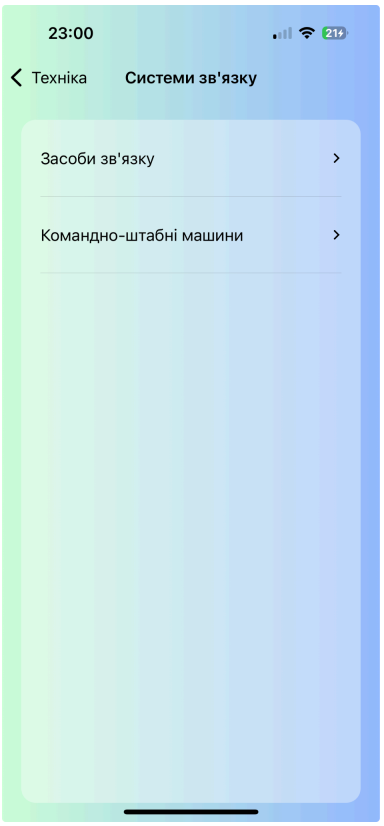


Рис. 4.19 Візуальне представлення SubcategoryVC

4.5.6 ElementVC

Клас *ElementVC* відображає список елементів у вигляді таблиці. Він містить дані про елементи у властивості *element*. Фон сторінки встановлюється за допомогою *bgImageView*. Візуально схожий на SubcategoryVC.

4.5.7 SettingsVC

Клас *SettingsVC* відображає налаштування у вигляді таблиці. Інформація про налаштування завантажується з *SettingsData.shared.getSettings()*. Фонове зображення *bgImageView* створює фон сторінки, а *topSeparatorView* відображає роздільник. Список налаштувань відображається у *settingsTable*. Заголовок навігаційної панелі змінюється за допомогою кастомного елемента *titleLbl*.

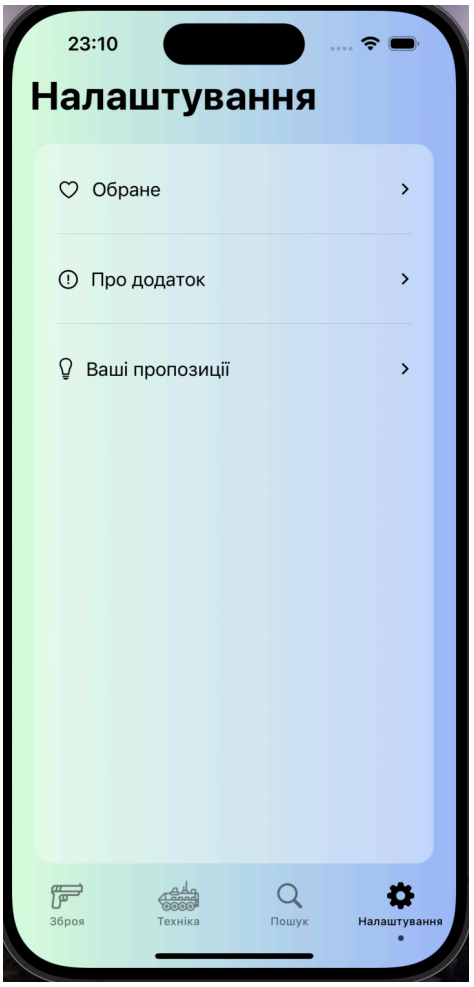


Рис. 4.20 Візуальне представлення SettingsVC

4.5.8 SearchVC

Клас *SearchVC* відображає сторінку пошуку в додатку. При завантаженні сторінки, він отримує випадкові елементи які можуть зацікавити користувача з

DataManager.shared.getRandom(). У сторінці є два основні елементи інтерфейсу:

1. *topSeparatorView*: Це UIView, який відображає верхній роздільник, щоб візуально розділити верхню частину екрану.
2. *searchTable*: Це UITableView, яке відображає результати пошуку. Кожен знайдений елемент представлений у вигляді комірки, яка відображається у списку.

При введенні користувачем тексту в поле пошуку `searchController`, виконується пошук, результати якого оновлюються в реальному часі за допомогою методу *updateSearchResults(for:)*. Результати пошуку відображаються у списку *searchTable*.

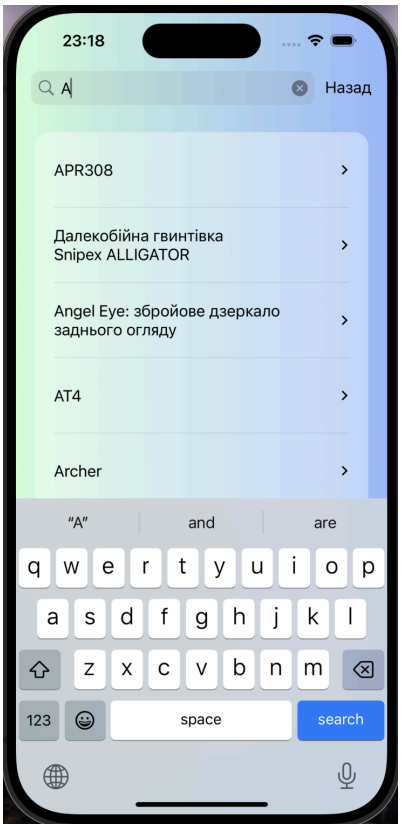


Рис. 4.21 Візуальне представлення SearchVC

4.5.9 GunVC та TechniqueVC

Клас *GunVC* та *TechniqueVC* обидва відображають список категорій (зброя та техніка відповідно) у додатку. Кожен з цих контролерів містить наступні основні елементи:

1. *Table View*: UITableView, що відображає список категорій. Кожен рядок таблиці відповідає одній категорії.
2. *Background Image View*: UIImageView, який відображає фонове зображення на екрані.

Ініціалізація кожного з цих елементів відбувається у функції *viewDidLoad()*, де також виконуються інші конфігурації, такі як налаштування навігаційної панелі та приховування або відображення панелі табуляції. Кожен контролер також має власний метод *configureNavBar()*, який встановлює відповідний заголовок для навігаційної панелі зліва.

Реалізація методів *UITableViewDelegate* та *UITableViewDataSource* відповідає за відображення даних у таблиці. Кожен рядок таблиці конфігурується з відповідною категорією.

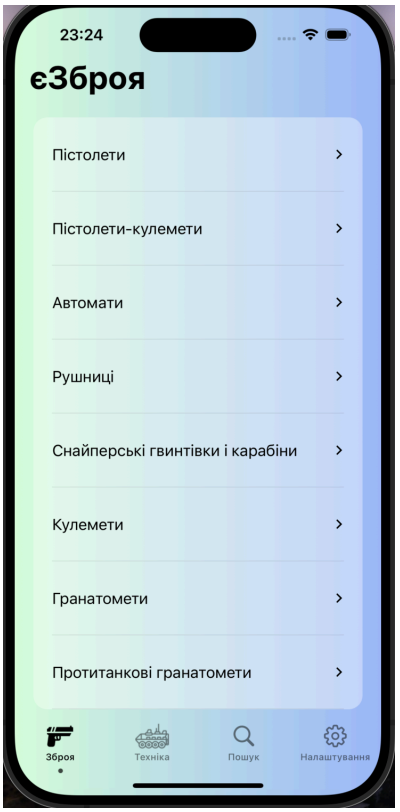


Рис. 4.22 Візуальне представлення GunsVC

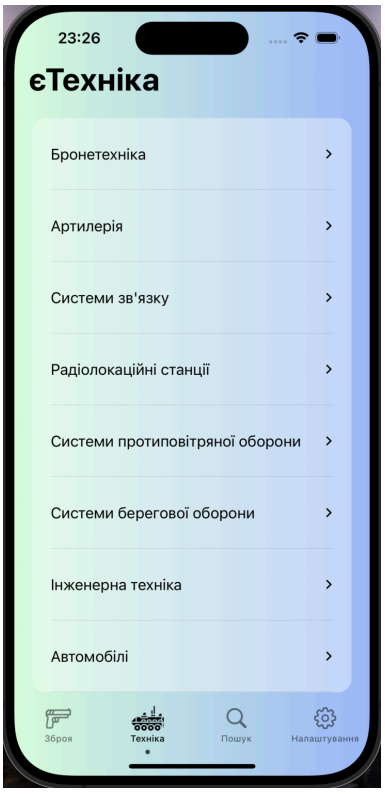


Рис. 4.23 Візуальне представлення TechniqueVC

5. ТЕСТУВАННЯ

5.1 Тестування навігації

Тестуються всі кнопки та елементи які відповідають за переходи на інші екрани, щоб переходи відповідали логіці навігації.

5.2 Тестування UI

Тестуються всі версії додатку на пристроях з різними характеристиками та перевіряється коректність відображення компонентів інтерфейсу користувача.

5.2 Тестування даних

Ключова робота з даними виконується за допомогою DataManager (4.3.1). Для впевненості в коректній роботі DataManager тестування відбувається за допомогою написання тест-кейсів.

5.2.1 testGetSubcategory

- Перевіряє, чи повертається правильна підкатегорія для заданої категорії.
- Очікує, що категорія у результаті співпадає з введеною категорією.
- Переконується, що підкатегорії не порожні.

5.2.2 testGetElements

- Перевіряє, чи повертається правильний елемент для заданої підкатегорії.
- Очікує, що підкатегорія у результаті співпадає з введеною підкатегорією.
- Переконується, що елементи не порожні.

5.2.3 testGetItem

- Перевіряє, чи повертається правильний об'єкт для заданого імені.
- Очікує, що ім'я об'єкта у результаті співпадає з введеним ім'ям.
- Переконується, що властивості об'єкта не порожні.

5.2.4 testGetRandom

- Перевіряє, чи повертається рівно 3 випадкових об'єкти.
- Переконується, що імена об'єктів не порожні.

5.2.5 testGetSearchedItems

- Перевіряє, чи повертаються правильні об'єкти для заданого запиту.
- Переконується, що результати пошуку не порожні.
- Переконується, що результати пошуку містять запит.

5.2.6 testGetAll

- Перевіряє, чи повертається список всіх об'єктів.
- Переконується, що список не порожній.

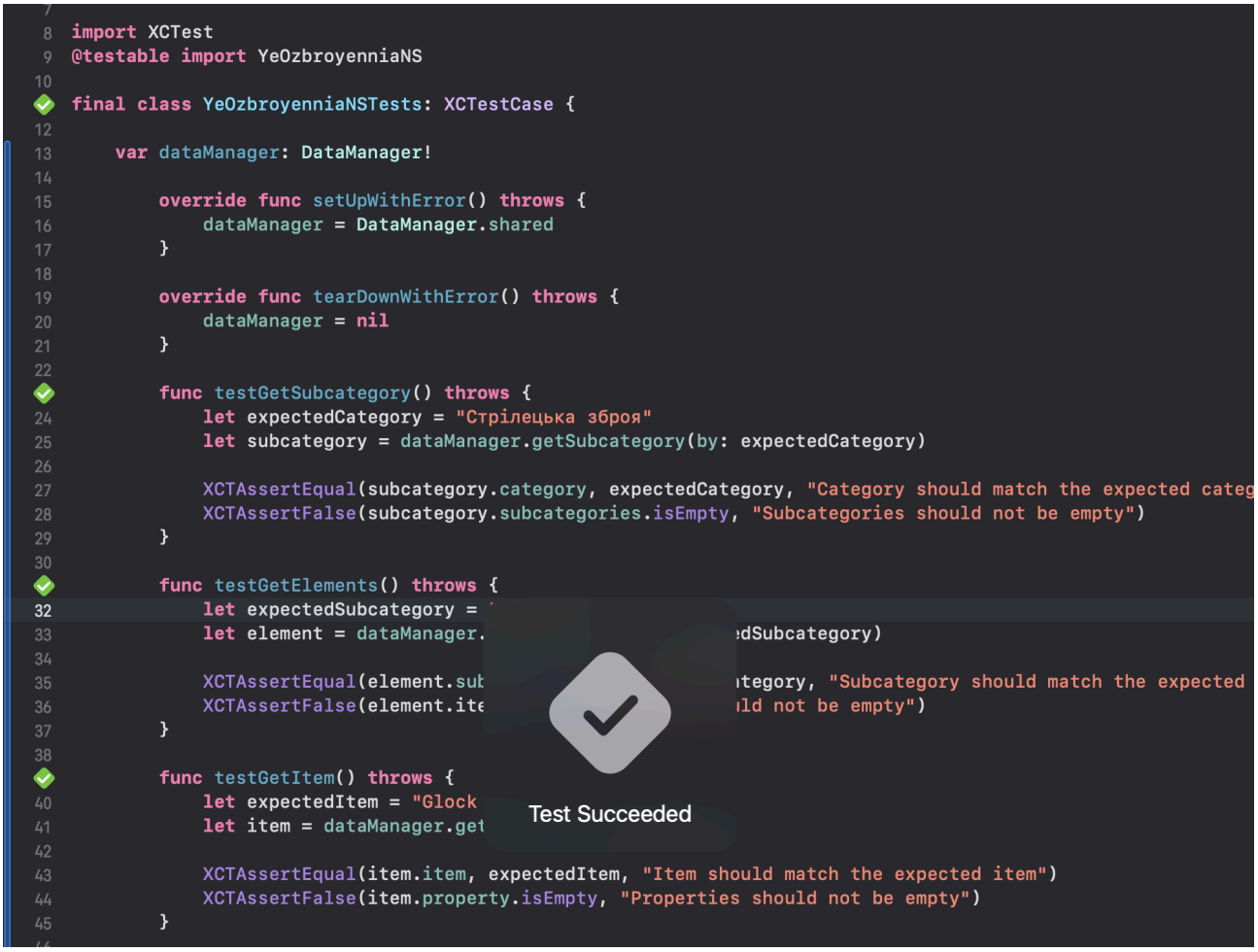


Рис. 5.1 Тест клас та результати тестування

ВИСНОВКИ

Таким чином, основною метою роботи була спрямована на проектування та розробку програмного забезпечення для перегляду характеристик до військової зброї і техніки.

1.Проведено аналіз предметної галузі, згідно теми дипломної роботи.

2.Проаналізовано три існуючих застосунки для перегляду характеристик зброї та техніки: Wikipedi, Ukrmillitary, Посібник, виділено їхні основні переваги та недоліки.

3.На основі аналізу предметної галузі та аналогів, визначено технічне завдання та розроблено функціональні та нефункціональні вимоги застосунку.

4.Проаналізовано існуючі засоби розробки застосунків та обрано наступні: фреймворки Swift UIKit, база даних Core Data, середовища розробки Xcode, та GitHub для контролю версій.

5.Спроектовано архітектуру застосунку. Архітектура застосунку була спроектована з урахуванням всіх функціональних та нефункціональних вимог.

6.Розроблено застосунок для перегляду характеристик зброї та техніки з застосуванням патерну Model-View-Controller.

7.Проведено тестування застосунку та перевірено виконання функціональних та нефункціональних вимог.

ПЕРЕЛІК ПОСИЛАНЬ

1. Swift Language Guide. [Електронний ресурс]. - Режим доступу: https://docs.swift.org/swift-book/GuidedTour/GuidedTour.html

2. Core Data Programming Guide. [Електронний ресурс]. - Режим доступу: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreData/index.html

3. Xcode User Guide. [Електронний ресурс]. - Режим доступу: https://developer.apple.com/library/archive/documentation/IDEs/Conceptual/Xcode_Overview/

4. Apple Human Interface Guidelines. [Електронний ресурс]. - Режим доступу: https://developer.apple.com/design/human-interface-guidelines/

5. App Store Connect Help. [Електронний ресурс]. - Режим доступу: https://help.apple.com/app-store-connect/

6. Model-View-Controller in iOS. [Електронний ресурс]. - Режим доступу: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка застосунку для ознайомлення з тактико-технічними характеристиками зброї та техніки мовою Swift

Виконав студент 4 курсу
групи ПД-41
Смахтін Олександр Станіславович
Керівник роботи
К.т.н. доц., доцент кафедри ІПЗ Негоденко Олена Василівна

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи**
  Спрощення процесу перегляду та пошуку характеристик військової техніки та зброї через розробку додатку, реалізованого на мові програмування Swift для платформи IOS

- **Об'єкт дослідження**
  Процес перегляду та пошуку характеристик військової техніки та зброї

- **Предмет дослідження**
  Мобільний додаток для пошуку та перегляду характеристик зброї та техніки

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести порівняльний аналіз існуючих засобів перегляду та пошуку характеристик військової техніки та зброї.

2. Сформувати функціональні та нефункціональні вимоги для реалізації мобільного додатку для перегляду та пошуку характеристик військової техніки та зброї.

3. Дослідити та вибрати засоби розробки застосунку, розробити модель архітектури застосунку та його дизайн.

4. Розробити застосунок на основі обраних інструментів та визначених вимог

5. Провести тестування застосунку

3

АНАЛІЗ АНАЛОГІВ

Характеристики	Wikipedia	Ukrmillitary	Фізичний посібник	єЗброя
Пошук елементів	+	-	-	+
Оновлення інформації про зброю та техніку	+	-	-	+
Мобільність	Мобільна, веб-сервіс	Веб-сервіс	паперовий вигляд	мобільний застосунок
Не потребує підключення до інтернету	-	-	+	+
Додавання у обрані	-	-	-	+
Редагування обраних	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Додаток має виконувати навігацію за розділами.
2. Додаток має мати наявність перегляду інформації про зброю та техніку.
3. Додаток має мати наявність пошуку інформації про зброю та техніку.
4. Додаток має додавання елементів зброї та техніки у розділ “обрані”.
5. Додаток має редагування розділу “обране”.
6. В додатку має бути наявністьзворотного зв’язку

Нефункціональні вимоги:

1. Додаток займає від 2мб до 10мб
2. Мобільний додаток не потребує підключення до інтернету
3. Мобільний додаток працює на операційній системі IOS
4. Додаток повинен підтримувати пристрої iPhone та iPad, що працюють на версіях ОС всі від 13.0
5. Мобільний додаток реалізован українською мовою

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Xcode

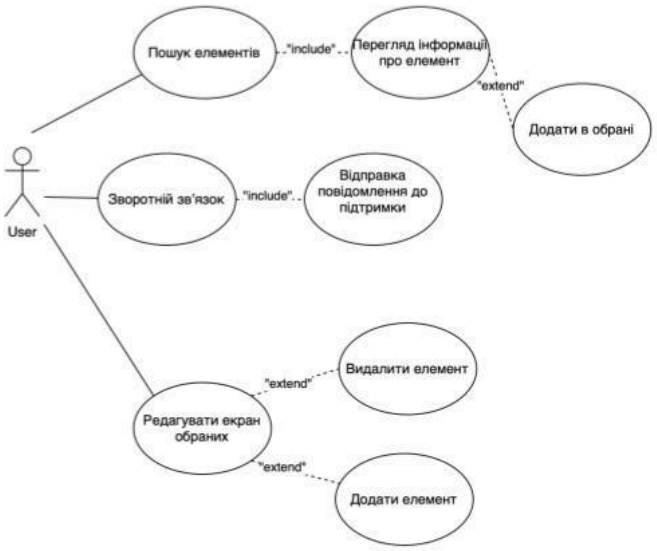
CoreData

Swift

GitHub

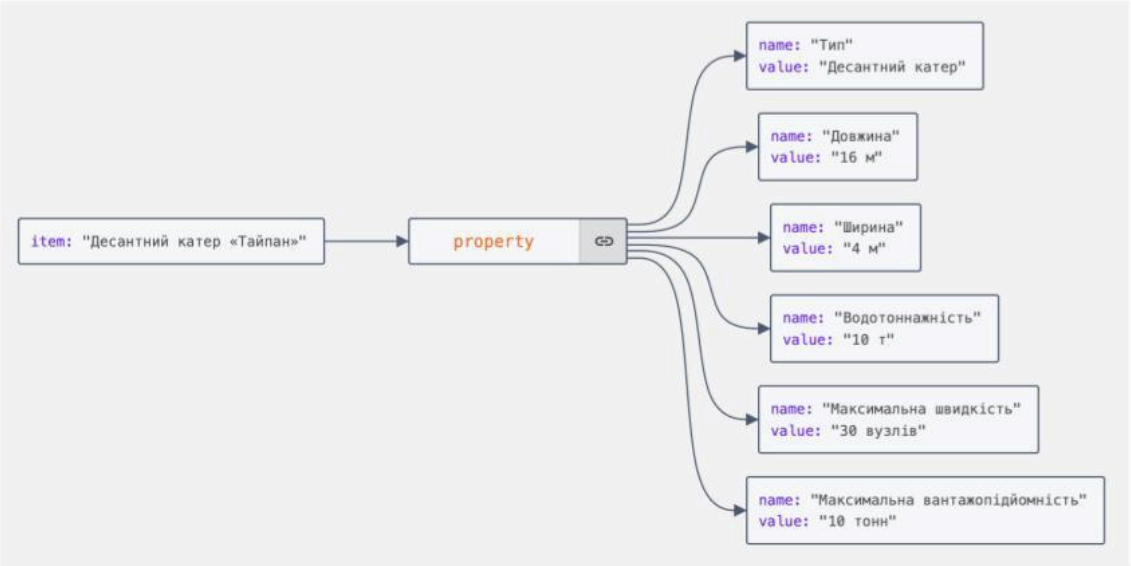
6

USE-CASE ДІАГРАМА



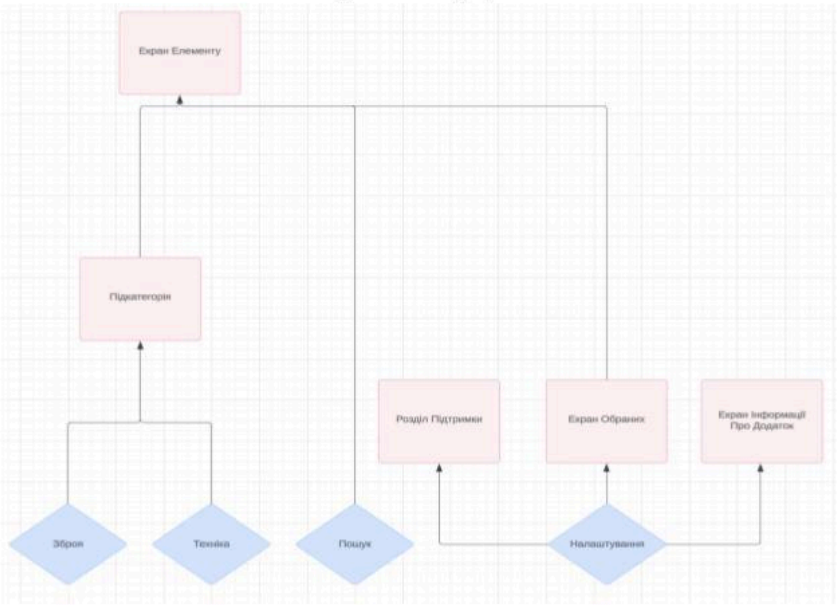
7

JSON ДІАГРАМА



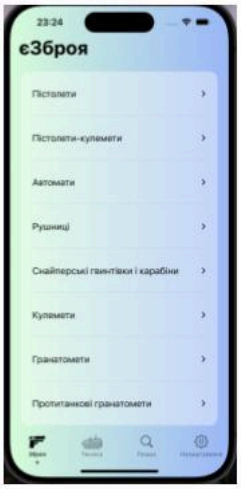
8

НАВІГАЦІЙНА ДІАГРАМА

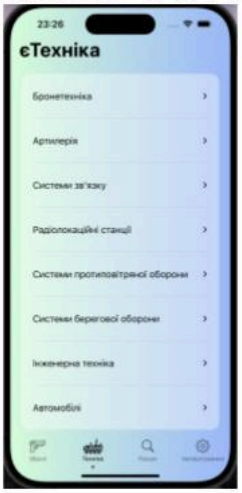


9

ЕКРАННІ ФОРМИ



Розділ зброї

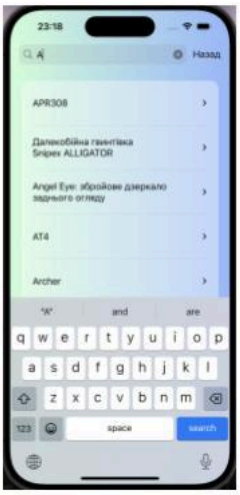


Розділ техніки

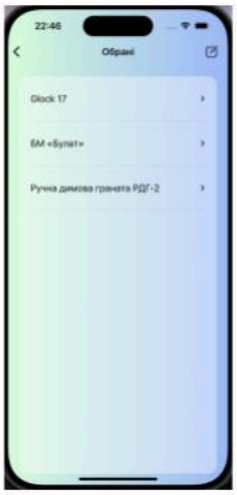
Розділ налаштувань

10

ЕКРАННІ ФОРМИ



Розділ пошуку



Розділ обраних

Розділ елементу

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Смахтін О.С., Негоденко О.В. Визначення вимог до мобільного додатку для організації ефективної роботи військових
з документацією. Всеукраїнська науково-технічна конференція “Застосування програмного забезпечення в інформаційно-комунікаційних технологіях”: 24 квітня 2024р., Київ, Державний університет телекомунікацій. Збірник тез. К.:ДУТ, 2024. С. 65

ВИСНОВКИ

1. Проведено аналіз предметної галузі, згідно теми дипломної роботи.
2. Проаналізовано три існуючих застосунки для перегляду характеристик зброї та техніки: Wikipedi, Ukrmillitary, Посібник, виділено їхні основні переваги та недоліки.
3. На основі аналізу предметної галузі та аналогів, визначено технічне завдання та розроблено функціональні та нефункціональні вимоги застосунку.
4. Проаналізовано існуючі засоби розробки застосунків та обрано наступні: фреймворки Swift UIKit, база даних Core Data, середовища розробки Xcode, та GitHub для контролю версій.
5. Спроектовано архітектуру застосунку. Архітектура застосунку була спроектована з урахуванням всіх функціональних та нефункціональних вимог.
6. Розроблено застосунок для перегляду характеристик зброї та техніки з застосуванням патерну Model-View-Controller.
7. Проведено тестування застосунку та перевірено виконання функціональних та нефункціональних вимог.

ДОДАТОК Б

```
//
//  DataManager.swift
//  YeOzbroyenniaNS
//
//  Created by Oleksandr Smakhtin on 18.04.2023.
//
import Foundation
class DataManager {
    static let shared = DataManager()

    let decoder = JSONDecoder()
    let encoder = JSONEncoder()

    //MARK: - Subcategory
    func getSubcategory(by category: String) -> Subcategory {
        var result = Subcategory(category: "", subcategories: [""])
         guard let path = Bundle.main.path(forResource: "subcategories", ofType:
"json") else { return result}
        do {
            //let jsonData = try Data(contentsOf: URL(filePath: path))
            let jsonData = try Data(contentsOf: URL(fileURLWithPath: path))

                let subcategories = try decoder.decode([Subcategory].self, from:
jsonData)
             let selectedSubcategory = subcategories.first { $0.category == category
}
            result = selectedSubcategory!
        } catch {
```

```
            print(error.localizedDescription)
        }
        return result
    }

    //MARK: - Elements
    func getElements(by subcategory: String) -> Element {
        var result = Element(subcategory: "", items: [""], imagePath: [""])
            guard let path = Bundle.main.path(forResource: "elements", ofType:
"json") else { return result}
        do {
            //let jsonData = try Data(contentsOf: URL(filePath: path))
            let jsonData = try Data(contentsOf: URL(fileURLWithPath: path))

            let elements = try decoder.decode([Element].self, from: jsonData)
            let selectedElement = elements.first { $0.subcategory == subcategory }
            result = selectedElement!
        } catch {
            print(error.localizedDescription)
        }
        return result
    }

    //MARK: - Item
    func getItem(by item: String) -> Item {
        var result = Item(item: "", property: [Property]())
            guard let path = Bundle.main.path(forResource: "allItems", ofType:
"json") else { return result }
        do {
            //let jsonData = try Data(contentsOf: URL(filePath: path))
```

```
                let jsonData = try Data(contentsOf: URL(fileURLWithPath: path))

                let items = try decoder.decode([Item].self, from: jsonData)
                let selectedItem = items.first { $0.item == item }
                if selectedItem != nil {
                    result = selectedItem!
                }
            } catch {
                print(error.localizedDescription)
            }

            return result
        }

        //MARK: - Random
        func getRandom() -> [String] {
            var results = [String]()
                guard let path = Bundle.main.path(forResource: "allItems", ofType:
"json") else { return [String]() }
            do {
                //let jsonData = try Data(contentsOf: URL(filePath: path))
                let jsonData = try Data(contentsOf: URL(fileURLWithPath: path))

                let items = try decoder.decode([Item].self, from: jsonData)
                let randomItems = items.shuffled().prefix(3)

                for item in randomItems {
                    results.append(item.item)
                }
            } catch {
```

```
            print(error.localizedDescription)
        }

        return results
    }

    private lazy var allItems = getAll()
    //MARK: - Search
    func getSearchedItems(by query: String) -> [String] {
        let searchWords = query.lowercased().split(separator: " ")

        let filteredData = allItems.filter { item in
            let itemWords = item.lowercased().split(separator: " ")
            return searchWords.allSatisfy { searchWord in
                itemWords.contains(where: { $0.hasPrefix(searchWord) })
            }
        }

        return filteredData
    }


    //MARK: - Get All
    private func getAll() -> [String] {
        var results = [String]()
            guard let path = Bundle.main.path(forResource: "allItems", ofType:
"json") else { return [String]() }
        do {
            //let jsonData = try Data(contentsOf: URL(filePath: path))
            let jsonData = try Data(contentsOf: URL(fileURLWithPath: path))
```

```
            let items = try decoder.decode([Item].self, from: jsonData)
            results = items.map { $0.item }
        } catch {
            print(error.localizedDescription)
        }

        return results
    }

}

//
//  CoreDataManager.swift
//  YeOzbroyenniaNS
//
//  Created by Oleksandr Smakhtin on 20.04.2023.
//
import Foundation
import UIKit
import CoreData
class CoreDataManager {

    static let shared = CoreDataManager()

    //MARK: - Context
                let context = (UIApplication.shared.delegate as!
AppDelegate).persistentContainer.viewContext

    //MARK: - Add
```

```
        func addItem(with title: String) {
            let item = CoreItem(context: context)
            item.title = title
            do {
                try context.save()
                print("Successfully saved Item to CoreData")
            } catch {
                                print("Failed to add Item to CoreData due to: \(error.localizedDescription)")
            }
        }

        //MARK: - Fetch
        func fetchItems() -> [String] {
            var results = [String]()
            let request = CoreItem.fetchRequest()
            do {
                let items = try context.fetch(request)
                results = items.map({ item in
                    guard let title = item.title else { return ""}
                    return title
                })
                print("Successfully fetched Items from CoreData")
            } catch {
                            print("Failed to fetch Items from CoreData due to: \(error.localizedDescription)")
            }
            return results
        }
```

```
//MARK: - Delete
func deleteItem(with title: String) {
    let request = CoreItem.fetchRequest()
    request.predicate = NSPredicate(format: "title == %@", title)
    request.fetchLimit = 1
    do {
        let items = try context.fetch(request)
        if let item = items.first {
            context.delete(item)
            try context.save()
        }
    } catch {
        print("Error fetching or deleting item: \(error)")
    }
}

//MARK: - Exist
func isItemExists(with title: String) -> Bool {
    let request = CoreItem.fetchRequest()
    request.predicate = NSPredicate(format: "title == %@", title)
    request.fetchLimit = 1
    do {
        let items = try context.fetch(request)
        return !items.isEmpty
    } catch {
        print("Error fetching items: \(error)")
        return false
    }

}
```

```
}
//
//  Item.swift
//  YeOzbroyenniaNS
//
//  Created by Oleksandr Smakhtin on 18.04.2023.
//
import Foundation
struct Item: Codable {
    let item: String
    let property: [Property]
}
struct Property: Codable {
    let name: String
    let value: String
}
//
//  Category.swift
//  YeOzbroyenniaNS
//
//  Created by Oleksandr Smakhtin on 18.04.2023.
//
import Foundation
class Category {
    static let shared = Category()

    func getGunCategories() -> [String] {
        lazy var categories = [
            "Пістолети",
            "Пістолети-кулемети",
```

```
        "Автомати",
        "Рушниці",
        "Снайперські гвинтівки і карабіни",
        "Кулемети",
        "Гранатомети",
        "Протитанкові гранатомети",
        "Протитанкові ракетні комплекси",
        "Переносні системи ППО",
        "Аксесуари для стрілецької зброї",
        "Ручні гранати",
        "Міни"
    ]

    return categories
}

func getTeqCategories() -> [String] {
    lazy var categories = [
        "Бронетехніка",
        "Артилерія",
        "Системи зв'язку",
        "Радіолокаційні станції",
        "Системи протиповітряної оборони",
        "Системи берегової оборони",
        "Інженерна техніка",
        "Автомобілі",
        "Авіаційна техніка",
        "Морське озброєння",  // -
        "Оптичні прилади",  // -
        "Системи захисту"  // -
```

```
        ]

        return categories
    }
}

//
//  Subcategory.swift
//  YeOzbroyenniaNS
//
//  Created by Oleksandr Smakhtin on 18.04.2023.
//
import Foundation
struct Subcategory: Codable {
    let category: String
    let subcategories: [String]
}

//
//  Element.swift
//  YeOzbroyenniaNS
//
//  Created by Oleksandr Smakhtin on 18.04.2023.
//
import Foundation
struct Element: Codable {
    let subcategory: String
    let items: [String]
    let imagePath: [String]?
}
```

```
//
//  Setting.swift
//  YeOzbroyenniaNS
//
//  Created by Oleksandr Smakhtin on 20.04.2023.
//
import Foundation
enum SettingType {
    case about
    case favorite
    case proposition
    case rate
    case share
    case thank
}
struct Setting {
    let type: SettingType
    let image: String
    let title: String
}
class SettingsData {
    static let shared = SettingsData()

    func getSettings() -> [Setting] {
        let settings = [
            Setting(type: .favorite, image: "heart", title: "Обране"),
                Setting(type: .about, image: "exclamationmark.circle", title: "Про додаток"),
            Setting(type: .proposition, image: "lightbulb", title: "Ваші пропозиції")
```

```
            ]
            return settings
        }
    }

    //
    //  ListCell.swift
    //  YeOzbroyenniaNS
    //
    //  Created by Oleksandr Smakhtin on 18.04.2023.
    //
    import UIKit
    class ListCellWithPhoto: UITableViewCell {
        //MARK: - Identifier
        static let identifier = "ListCellWithPhoto"

        //MARK: - UI Objects
        private let titleLbl: UILabel = {
            let lbl = UILabel()
            lbl.textColor = .label
                    //lbl.font  =  UIFont.systemFont(ofSize:  lbl.font.pointSize,  weight:
.medium)
            lbl.numberOfLines = 0
            lbl.translatesAutoresizingMaskIntoConstraints = false
            return lbl
        }()

        private let zbroyaImage: UIImageView = {
            let imageView = UIImageView()
            imageView.contentMode = .scaleAspectFit
```

```
        imageView.translatesAutoresizingMaskIntoConstraints = false
        return imageView
    }()

    private let disclousereIndicator: UIImageView = {
        let imageView = UIImageView()
                var image = UIImage(named: "arrow", in: nil, with:
UIImage.SymbolConfiguration(pointSize: 4, weight: .medium))
        image = image?.withTintColor(.label, renderingMode: .automatic)
        imageView.image = image
        imageView.translatesAutoresizingMaskIntoConstraints = false
        return imageView
    }()


    private let bottomSeparatorView: UIView = {
        let view = UIView()
        view.backgroundColor = UIColor(named: "separator")
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    //MARK: - Init
    override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
        // bg color
        backgroundColor = .clear
        // selection
        selectionStyle = .none
        // add subviews
```

```
        addSubviews()
        // apply constraints
        applyConstraints()
    }

    //MARK: - Required init
    required init?(coder: NSCoder) {
        fatalError()
    }

    //MARK: - Add subviews
    private func addSubviews() {
        //contentView.addSubview(bottomSeparatorView)
        contentView.addSubview(titleLbl)
        contentView.addSubview(disclousereIndicator)
        contentView.addSubview(zbroyaImage)
    }

    //MARK: - Apply constraints
    private func applyConstraints() {
        // titleLbl constraints
        let titleLblConstraints = [
            titleLbl.leadingAnchor.constraint(equalTo: contentView.leadingAnchor,
constant: 120),
             titleLbl.trailingAnchor.constraint(equalTo: contentView.trailingAnchor,
constant: -50),
                                          titleLbl.centerYAnchor.constraint(equalTo:
contentView.centerYAnchor)
        ]
```

```
        //image
        let zbroyaImageConstraints = [
                                        zbroyaImage.leadingAnchor.constraint(equalTo:
contentView.leadingAnchor, constant: 20),
            zbroyaImage.trailingAnchor.constraint(equalTo: titleLbl.leadingAnchor,
constant: -20),
                zbroyaImage.topAnchor.constraint(equalTo: contentView.topAnchor,
constant: 20),
                                        zbroyaImage.bottomAnchor.constraint(equalTo:
contentView.bottomAnchor, constant: -20)
        ]

        // disclousere Indicator constraints
        let disclousereIndicatorConstraints = [
                                    disclousereIndicator.trailingAnchor.constraint(equalTo:
contentView.trailingAnchor, constant: -20),
                                    disclousereIndicator.centerYAnchor.constraint(equalTo:
titleLbl.centerYAnchor),
            disclousereIndicator.heightAnchor.constraint(equalToConstant: 10),
            disclousereIndicator.widthAnchor.constraint(equalToConstant: 10)
        ]

        // activate constraints
        //NSLayoutConstraint.activate(bottomSeparatorViewConstraints)
        NSLayoutConstraint.activate(titleLblConstraints)
        NSLayoutConstraint.activate(disclousereIndicatorConstraints)
        NSLayoutConstraint.activate(zbroyaImageConstraints)
    }
    //MARK: - Configure
    public func configure(with title: String, image: String?) {
```

```
            titleLbl.text = title
            guard let image = image else {
                zbroyaImage.image = UIImage(named: "fort_14TP")
                return
            }
            zbroyaImage.image = UIImage(named: image)
        }

    }

    //
    //  ListCell.swift
    //  YeOzbroyenniaNS
    //
    //  Created by Oleksandr Smakhtin on 18.04.2023.
    //
    import UIKit
    class ListCell: UITableViewCell {
        //MARK: - Identifier
        static let identifier = "ListCell"

        //MARK: - UI Objects
        private let titleLbl: UILabel = {
            let lbl = UILabel()
            lbl.textColor = .label
                    //lbl.font = UIFont.systemFont(ofSize: lbl.font.pointSize, weight:
.medium)
            lbl.numberOfLines = 0
            lbl.translatesAutoresizingMaskIntoConstraints = false
            return lbl
```

```
    }()

    private let disclousereIndicator: UIImageView = {
        let imageView = UIImageView()
                    var image = UIImage(named: "arrow", in: nil, with: UIImage.SymbolConfiguration(pointSize: 4, weight: .medium))
        image = image?.withTintColor(.label, renderingMode: .automatic)
        imageView.image = image
        imageView.translatesAutoresizingMaskIntoConstraints = false
        return imageView
    }()


    private let bottomSeparatorView: UIView = {
        let view = UIView()
        view.backgroundColor = UIColor(named: "separator")
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    //MARK: - Init
    override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
        // bg color
        backgroundColor = .clear
        // selection
        selectionStyle = .none
        // add subviews
        addSubviews()
        // apply constraints
```

```
        applyConstraints()
    }

    //MARK: - Required init
    required init?(coder: NSCoder) {
        fatalError()
    }

    //MARK: - Add subviews
    private func addSubviews() {
        //contentView.addSubview(bottomSeparatorView)
        contentView.addSubview(titleLbl)
        contentView.addSubview(disclousereIndicator)
    }

    //MARK: - Apply constraints
    private func applyConstraints() {
        // bottomSeparatorView constraints
        let bottomSeparatorViewConstraints = [
                        bottomSeparatorView.leadingAnchor.constraint(equalTo:
contentView.leadingAnchor, constant: 20),
                        bottomSeparatorView.trailingAnchor.constraint(equalTo:
contentView.trailingAnchor),
                        bottomSeparatorView.bottomAnchor.constraint(equalTo:
contentView.bottomAnchor),
            bottomSeparatorView.heightAnchor.constraint(equalToConstant: 1)
        ]

        // titleLbl constraints
        let titleLblConstraints = [
```

```
                titleLbl.leadingAnchor.constraint(equalTo: contentView.leadingAnchor, constant: 20),
                titleLbl.trailingAnchor.constraint(equalTo: contentView.trailingAnchor, constant: -50),
                                                titleLbl.centerYAnchor.constraint(equalTo: contentView.centerYAnchor)
            ]

            // disclousere Indicator constraints
            let disclousereIndicatorConstraints = [
                                    disclousereIndicator.trailingAnchor.constraint(equalTo: contentView.trailingAnchor, constant: -20),
                                    disclousereIndicator.centerYAnchor.constraint(equalTo: titleLbl.centerYAnchor),
                disclousereIndicator.heightAnchor.constraint(equalToConstant: 10),
                disclousereIndicator.widthAnchor.constraint(equalToConstant: 10)
            ]

            // activate constraints
            //NSLayoutConstraint.activate(bottomSeparatorViewConstraints)
            NSLayoutConstraint.activate(titleLblConstraints)
            NSLayoutConstraint.activate(disclousereIndicatorConstraints)
        }
        //MARK: - Configure
        public func configure(with title: String) {
            titleLbl.text = title
        }

    }
```

```
//
//  ItemCell.swift
//  YeOzbroyenniaNS
//
//  Created by Oleksandr Smakhtin on 18.04.2023.
//
import UIKit
class ItemCell: UITableViewCell {
    //MARK: - Identifier
    static let identifier = "ItemCell"

    //MARK: - UI Objects
    private let nameLbl: UILabel = {
        let lbl = UILabel()
        lbl.numberOfLines = 0
        lbl.textAlignment = .left
        lbl.font = UIFont.systemFont(ofSize: 16)
        lbl.textColor = .label
        lbl.translatesAutoresizingMaskIntoConstraints = false
        return lbl
    }()

    private let valueLbl: UILabel = {
        let lbl = UILabel()
        lbl.numberOfLines = 0
        lbl.textAlignment = .right
        lbl.font = UIFont.systemFont(ofSize: 15, weight: .semibold)
        lbl.textColor = .label
        lbl.translatesAutoresizingMaskIntoConstraints = false
        return lbl
```

```
}()


//MARK: - Init
override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
    super.init(style: style, reuseIdentifier: reuseIdentifier)
    // bg color
    backgroundColor = .clear
    // selection
    selectionStyle = .none
    // add subviews
    addSubviews()
    // apply constraints
    applyConstraints()
}

//MARK: - Required init
required init?(coder: NSCoder) {
    fatalError()
}

//MARK: - Add subviews
private func addSubviews() {
    contentView.addSubview(nameLbl)
    contentView.addSubview(valueLbl)
}

//MARK: - Apply constraints
private func applyConstraints() {
    // nameLbl constraints
```

```
        let nameLblConstraints = [
                                        nameLbl.leadingAnchor.constraint(equalTo:
contentView.leadingAnchor, constant: 10),
                                        nameLbl.centerYAnchor.constraint(equalTo:
contentView.centerYAnchor),
                                    nameLbl.widthAnchor.constraint(equalToConstant:
contentView.frame.width/2 - 20),
                    nameLbl.topAnchor.constraint(equalTo:  contentView.topAnchor,
constant: 10),
                                        nameLbl.bottomAnchor.constraint(equalTo:
contentView.bottomAnchor, constant: -10)
        ]

        // valueLbl constraints
        let valueLblConstraints = [
                                        valueLbl.trailingAnchor.constraint(equalTo:
contentView.trailingAnchor, constant: -10),
                                        valueLbl.centerYAnchor.constraint(equalTo:
contentView.centerYAnchor),
                    valueLbl.topAnchor.constraint(equalTo:  contentView.topAnchor,
constant: 10),
                                        valueLbl.bottomAnchor.constraint(equalTo:
contentView.bottomAnchor, constant: -10),
                                    valueLbl.widthAnchor.constraint(equalToConstant:
contentView.frame.width/2 - 10)
        ]

        // activate constraints
        NSLayoutConstraint.activate(nameLblConstraints)
        NSLayoutConstraint.activate(valueLblConstraints)
```

```
    }
    //MARK: - Configure
    public func configure(with name: String, value: String) {
        nameLbl.text = name
        valueLbl.text = value
    }
}

//
//  SettingCell.swift
//  YeOzbroyenniaNS
//
//  Created by Oleksandr Smakhtin on 20.04.2023.
//
import UIKit
class SettingCell: UITableViewCell {
    //MARK: - Identifier
    static let identifier = "SettingCell"

    //MARK: - UI Objects
    private let titleImageView: UIImageView = {
        let imageView = UIImageView()
        imageView.tintColor = .label
        imageView.translatesAutoresizingMaskIntoConstraints = false
        return imageView
    }()

    private let titleLbl: UILabel = {
        let lbl = UILabel()
        lbl.textColor = .label
```

```
        lbl.numberOfLines = 0
        lbl.translatesAutoresizingMaskIntoConstraints = false
        return lbl
    }()

    private let disclousereIndicator: UIImageView = {
        let imageView = UIImageView()
                var image = UIImage(named: "arrow", in: nil, with:
UIImage.SymbolConfiguration(pointSize: 4))
        image = image?.withTintColor(.label, renderingMode: .automatic)
        imageView.image = image
        imageView.translatesAutoresizingMaskIntoConstraints = false
        return imageView
    }()

    //MARK: - Init
    override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)
        // bg color
        backgroundColor = .clear
        // selection
        selectionStyle = .none
        // add subviews
        addSubviews()
        // apply constraints
        applyConstraints()
    }

    //MARK: - Required init
    required init?(coder: NSCoder) {
```

```
            fatalError()
        }

        //MARK: - Add subviews
        private func addSubviews() {
            contentView.addSubview(titleImageView)
            contentView.addSubview(titleLbl)
            contentView.addSubview(disclousereIndicator)
        }

        //MARK: - Apply constraints
        private func applyConstraints() {

            // title image view constraints
            let titleImageViewConstraints = [
                                titleImageView.leadingAnchor.constraint(equalTo:
contentView.leadingAnchor, constant: 20),
                                titleImageView.centerYAnchor.constraint(equalTo:
contentView.centerYAnchor)
            ]

            // title lbl constraints
            let titleLblConstraints = [
                                    titleLbl.leadingAnchor.constraint(equalTo:
titleImageView.trailingAnchor, constant: 10),
                                    titleLbl.centerYAnchor.constraint(equalTo:
titleImageView.centerYAnchor)
            ]

            // disclousere Indicator constraints
```

```
            let disclousereIndicatorConstraints = [
                                    disclousereIndicator.trailingAnchor.constraint(equalTo:
contentView.trailingAnchor, constant: -20),
                                    disclousereIndicator.centerYAnchor.constraint(equalTo:
titleLbl.centerYAnchor),
                disclousereIndicator.heightAnchor.constraint(equalToConstant: 10),
                disclousereIndicator.widthAnchor.constraint(equalToConstant: 10)
            ]

            // activate constraints
            NSLayoutConstraint.activate(titleImageViewConstraints)
            NSLayoutConstraint.activate(titleLblConstraints)
            NSLayoutConstraint.activate(disclousereIndicatorConstraints)
        }
        //MARK: - Configure
        public func configure(with setting: Setting) {
            titleLbl.text = setting.title
            titleImageView.image = UIImage(systemName: setting.image)
        }
    }

    //
    //  GunVC.swift
    //  YeOzbroyenniaNS
    //
    //  Created by Oleksandr Smakhtin on 16.04.2023.
    //
    import UIKit
    class GunVC: UIViewController {
```

```
        //MARK: - Data
        let categories = Category.shared.getGunCategories()
        //MARK: - UI Objects
        private let topSeparatorView: UIView = {
            let view = UIView()
            view.backgroundColor = .clear
            view.translatesAutoresizingMaskIntoConstraints = false
            return view
        }()

        private let gunsTable: UITableView = {
            let table = UITableView()
            table.register(ListCell.self, forCellReuseIdentifier: ListCell.identifier)
            table.showsVerticalScrollIndicator = false
             table.backgroundColor = UIColor(named: "tableColor")//UIColor(named:
"tableColor")
            table.layer.cornerRadius = 15
            table.separatorInset = UIEdgeInsets(top: 0, left: 20, bottom: 0, right: 20)
            table.translatesAutoresizingMaskIntoConstraints = false
            return table
        }()

        private let bgImageView: UIImageView = {
            let imageView = UIImageView()
            imageView.image = UIImage(named: "background")
            return imageView
        }()

        //MARK: - viewDidLoad
        override func viewDidLoad() {
```

```
        super.viewDidLoad()
        // bg color
        view.backgroundColor = .systemBackground
        // configure nav bar
        configureNavBar()
        // add subviews
        addSubviews()
        // apply constraints
        applyConstraints()
        // apply delegates
        applyTableDelegates()
            tabBarController?.tabBar.backgroundColor = .clear//UIColor(named:
"tableColor")//UIColor(named: "tableColor")
    //      let json = DataPersistance.shared.encodeData()
    //      FileHandler.shared.saveJson(json: json)

    }
    //MARK: - viewDidLayoutSubviews
    override func viewDidLayoutSubviews() {
        super.viewDidLayoutSubviews()
        bgImageView.frame = view.frame
    }

    //MARK: - Add subviews
    private func addSubviews() {
        view.addSubview(bgImageView)
        view.addSubview(topSeparatorView)
        view.addSubview(gunsTable)
    }
```

```
        //MARK: - Apply constraints
        private func applyConstraints() {
            // topSeparatorView constraints
            let topSeparatorViewConstraints = [
                                topSeparatorView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
                                topSeparatorView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
                                    topSeparatorView.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor),
                topSeparatorView.heightAnchor.constraint(equalToConstant: 1)
            ]

            // gunsTable constraints
            let gunsTableConstraints = [
                    gunsTable.leadingAnchor.constraint(equalTo:  view.leadingAnchor,
constant: 20),
                    gunsTable.trailingAnchor.constraint(equalTo:  view.trailingAnchor,
constant: -20),
                                            gunsTable.topAnchor.constraint(equalTo:
topSeparatorView.bottomAnchor, constant: 20),
                                        gunsTable.bottomAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.bottomAnchor, constant: -10)
            ]

            // activate constraints
            NSLayoutConstraint.activate(topSeparatorViewConstraints)
            NSLayoutConstraint.activate(gunsTableConstraints)
        }
```

```
        //MARK: - Congifure nav bar
        private func configureNavBar() {
            let titleLbl: UILabel = {
                let lbl = UILabel()
                lbl.text = "єЗброя"
                lbl.textColor = .label
                lbl.font = UIFont.systemFont(ofSize: 35, weight: .bold)
                return lbl
            }()

                navigationItem.leftBarButtonItem = UIBarButtonItem(customView:
titleLbl)
        }
    }
    //MARK: - Lifecycle
    extension GunVC {
        //MARK: - viewWillAppear
        override func viewWillAppear(_ animated: Bool) {
            super.viewWillAppear(animated)
            tabBarController?.tabBar.isHidden = false
        }
    }
    //MARK: - UITableViewDelegate & DataSource
    extension GunVC: UITableViewDelegate, UITableViewDataSource {
        // apply table delegates
        private func applyTableDelegates() {
            gunsTable.delegate = self
            gunsTable.dataSource = self
```

```
    }

    // nubers of rows
        func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return categories.count
    }

    // cell for row
        func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
            guard let cell = tableView.dequeueReusableCell(withIdentifier: ListCell.identifier) as? ListCell else { return UITableViewCell() }
        cell.configure(with: categories[indexPath.row])
        return cell
    }

    // height for row
        func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return 80
    }

    // did select row
        func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        let model = categories[indexPath.row]
        let element = DataManager.shared.getElements(by: model)

        let vc = ElementVC()
```

```
            vc.element = element
                navigationItem.backBarButtonItem = UIBarButtonItem(title: "Зброя",
style: .plain, target: nil, action: nil)
            navigationController?.pushViewController(vc, animated: true)

        }
    }

    //
    //  TechniqueVC.swift
    //  YeOzbroyenniaNS
    //
    //  Created by Oleksandr Smakhtin on 16.04.2023.
    //
    import UIKit
    class TechniqueVC: UIViewController {

        //MARK: - Data
        private let categories = Category.shared.getTeqCategories()


        //MARK: - UI Objects
        private let topSeparatorView: UIView = {
            let view = UIView()
            view.backgroundColor = .clear
            view.translatesAutoresizingMaskIntoConstraints = false
            return view
        }()

        private let teqTable: UITableView = {
```

```
    let table = UITableView()
    table.register(ListCell.self, forCellReuseIdentifier: ListCell.identifier)
    table.showsVerticalScrollIndicator = false
    table.layer.cornerRadius = 15
    table.backgroundColor = UIColor(named: "tableColor")
    table.separatorInset = UIEdgeInsets(top: 0, left: 20, bottom: 0, right: 20)
    table.translatesAutoresizingMaskIntoConstraints = false
    return table
}()

private let bgImageView: UIImageView = {
    let imageView = UIImageView()
    imageView.image = UIImage(named: "background")
    return imageView
}()
//MARK: - viewDidLoad
override func viewDidLoad() {
    super.viewDidLoad()
    // configure nav bar
    configureNavBar()
    // add subviews
    addSubviews()
    // apply constraints
    applyConstraints()
    // apply delegates
    applyTableDelegates()
}



//MARK: - viewDidLayoutSubviews
```

```
        override func viewDidLayoutSubviews() {
            super.viewDidLayoutSubviews()
            bgImageView.frame = view.frame
        }

        //MARK: - Add subviews
        private func addSubviews() {
            view.addSubview(bgImageView)
            view.addSubview(topSeparatorView)
            view.addSubview(teqTable)
        }

        //MARK: - Apply constraints
        private func applyConstraints() {
            // topSeparatorView constraints
            let topSeparatorViewConstraints = [
                                topSeparatorView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
                                topSeparatorView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
                                        topSeparatorView.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor),
                    topSeparatorView.heightAnchor.constraint(equalToConstant: 1)
            ]

            // teqTable constraints
            let teqTableConstraints = [
                        teqTable.leadingAnchor.constraint(equalTo:  view.leadingAnchor,
constant: 20),
                        teqTable.trailingAnchor.constraint(equalTo:  view.trailingAnchor,
```

```
constant: -20),
                                                    teqTable.topAnchor.constraint(equalTo:
topSeparatorView.bottomAnchor, constant: 20),
                                                    teqTable.bottomAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.bottomAnchor, constant: -10)
            ]

            // activate constraints
            NSLayoutConstraint.activate(topSeparatorViewConstraints)
            NSLayoutConstraint.activate(teqTableConstraints)

        }

        //MARK: - Congifure nav bar
        private func configureNavBar() {
            let titleLbl: UILabel = {
                let lbl = UILabel()
                lbl.text = "єТехніка"
                lbl.textColor = .label
                lbl.font = UIFont.systemFont(ofSize: 35, weight: .bold)
                return lbl
            }()

                navigationItem.leftBarButtonItem = UIBarButtonItem(customView:
titleLbl)
        }
    }
    //MARK: - Lifecycle
    extension TechniqueVC {
        //MARK: - viewWillAppear
```

```
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        tabBarController?.tabBar.isHidden = false
    }
}
//MARK: - UITableViewDelegate & DataSource
extension TechniqueVC: UITableViewDelegate, UITableViewDataSource {
    // apply table delegates
    private func applyTableDelegates() {
        teqTable.delegate = self
        teqTable.dataSource = self
    }

    // nubers of rows
        func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return categories.count
    }

    // cell for row
        func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
                guard let cell = tableView.dequeueReusableCell(withIdentifier: ListCell.identifier) as? ListCell else { return UITableViewCell() }
        cell.configure(with: categories[indexPath.row])
        return cell
    }

    // height for row
    func tableView(_ tableView: UITableView, heightForRowAt indexPath:
```

```
IndexPath) -> CGFloat {
            return 80
        }

        // did select row
            func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
            let model = categories[indexPath.row]
            let subcategory = DataManager.shared.getSubcategory(by: model)
            let vc = SubcategoryVC()
            vc.subcategory = subcategory
              navigationItem.backBarButtonItem = UIBarButtonItem(title: "Техніка",
style: .plain, target: nil, action: nil)
            navigationController?.pushViewController(vc, animated: true)
        }
    }

    //
    //  SearchVC.swift
    //  YeOzbroyenniaNS
    //
    //  Created by Oleksandr Smakhtin on 17.04.2023.
    //
    import UIKit
    class SearchVC: UIViewController {

        //MARK: - Data
        private var searchedItems = DataManager.shared.getRandom()

        //MARK: - UI Objects
```

```
    private let topSeparatorView: UIView = {
        let view = UIView()
        view.backgroundColor = .clear
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    private let searchTable: UITableView = {
        let table = UITableView()
        table.register(ListCell.self, forCellReuseIdentifier: ListCell.identifier)
        table.showsVerticalScrollIndicator = false
        table.layer.cornerRadius = 15
        table.separatorInset = UIEdgeInsets(top: 0, left: 20, bottom: 0, right: 20)
        table.backgroundColor = UIColor(named: "tableColor")
        table.translatesAutoresizingMaskIntoConstraints = false
        return table
    }()

    private let searchController: UISearchController = {
            let controller = UISearchController(searchResultsController:
SearchResultsVC())
        controller.searchBar.placeholder = "Я шукаю..."
        controller.searchBar.searchBarStyle = .minimal
        controller.searchBar.setValue("Назад", forKey: "cancelButtonText")
        return controller
    }()

    private let bgImageView: UIImageView = {
        let imageView = UIImageView()
        imageView.image = UIImage(named: "background")
```

```
    return imageView
}()
//MARK: - viewDidLoad
override func viewDidLoad() {
    super.viewDidLoad()
    // configure nav bar
    configureNavBar()
    // add subviews
    addSubviews()
    // apply constraints
    applyConstraints()
    // apply delegates
    applyTableDelegates()
    applySearchDelegates()
}

//MARK: - viewDidLayoutSubviews
override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    bgImageView.frame = view.frame
}

//MARK: - Add subviews
private func addSubviews() {
    view.addSubview(bgImageView)
    view.addSubview(topSeparatorView)
    view.addSubview(searchTable)
}

//MARK: - Apply constraints
```

```
        private func applyConstraints() {
            // topSeparatorView constraints
            let topSeparatorViewConstraints = [
                                topSeparatorView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
                                topSeparatorView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
                                    topSeparatorView.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor),
                topSeparatorView.heightAnchor.constraint(equalToConstant: 1)
            ]

            // searchTable constraints
            let searchTableConstraints = [
                    searchTable.leadingAnchor.constraint(equalTo: view.leadingAnchor,
constant: 20),
                    searchTable.trailingAnchor.constraint(equalTo: view.trailingAnchor,
constant: -20),
                                            searchTable.topAnchor.constraint(equalTo:
topSeparatorView.bottomAnchor, constant: 20),
                                        searchTable.bottomAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.bottomAnchor, constant: -10)
            ]

            // activate constraints
            NSLayoutConstraint.activate(topSeparatorViewConstraints)
            NSLayoutConstraint.activate(searchTableConstraints)
        }

        //MARK: - Congifure nav bar
```

```
        private func configureNavBar() {
            let titleLbl: UILabel = {
                let lbl = UILabel()
                lbl.text = "Пошук"
                lbl.textColor = .label
                lbl.font = UIFont.systemFont(ofSize: 35, weight: .bold)
                return lbl
            }()
            navigationItem.searchController = searchController
            navigationController?.navigationBar.tintColor = .label
            navigationItem.hidesSearchBarWhenScrolling = false
                navigationItem.leftBarButtonItem = UIBarButtonItem(customView:
titleLbl)
        }
    }
    //MARK: - Lifecycle
    extension SearchVC {
        //MARK: - viewWillAppear
        override func viewWillAppear(_ animated: Bool) {
            super.viewWillAppear(animated)
            tabBarController?.tabBar.isHidden = false
        }
    }
    //MARK: - UITableViewDelegate & DataSource
    extension SearchVC: UITableViewDelegate, UITableViewDataSource {
        // apply table delegates
        private func applyTableDelegates() {
            searchTable.delegate = self
            searchTable.dataSource = self
        }
```

```
        // nubers of rows
            func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
            return searchedItems.count
        }

        // cell for row
            func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
                    guard let cell = tableView.dequeueReusableCell(withIdentifier: ListCell.identifier) as? ListCell else { return UITableViewCell() }
            cell.configure(with: searchedItems[indexPath.row])
            return cell
        }

        // height for row
            func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
            return 80
        }

        // did select row
            func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
            tableView.deselectRow(at: indexPath, animated: true)

            let itemTitle = searchedItems[indexPath.row]
            let item = DataManager.shared.getItem(by: itemTitle)
            let vc = ItemVC()
```

```
            vc.item = item
            vc.configureVc(with: item, imagePath: nil)
              navigationItem.backBarButtonItem = UIBarButtonItem(title: "Пошук",
style: .plain, target: nil, action: nil)
            navigationController?.pushViewController(vc, animated: true)
        }
    }
    //MARK: - UISearchResultsUpdating
    extension SearchVC: UISearchResultsUpdating {
        private func applySearchDelegates() {
            searchController.searchResultsUpdater = self
        }

        func updateSearchResults(for searchController: UISearchController) {
            let searchBar = searchController.searchBar
                    guard let query = searchBar.text, !query.trimmingCharacters(in:
.whitespaces).isEmpty, query.trimmingCharacters(in: .whitespaces).count >= 1 else {
return }
             guard let resultController = searchController.searchResultsController as?
SearchResultsVC else { return }
            resultController.delegate = self

            let searchedData = DataManager.shared.getSearchedItems(by: query)

            resultController.searchedData = searchedData

            searchedItems = searchedData
            searchTable.reloadData()
        }
    }
```

```
//MARK: - SearchResultsDelegate
extension SearchVC: SearchResultDelegate {
    func didSelectItem(with title: String) {
        DispatchQueue.main.async { [weak self] in
            let item = DataManager.shared.getItem(by: title)
            let vc = ItemVC()
            vc.configureVc(with: item, imagePath: nil)
                self?.navigationItem.backBarButtonItem = UIBarButtonItem(title:
"Пошук", style: .plain, target: nil, action: nil)
            self?.navigationController?.pushViewController(vc, animated: true)
        }
    }


}

//
//  SettingsVC.swift
//  YeOzbroyenniaNS
//
//  Created by Oleksandr Smakhtin on 16.04.2023.
//
import UIKit
class SettingsVC: UIViewController {

    //MARK: - Data
    private let settings = SettingsData.shared.getSettings()

    //MARK: - UI Objects
    private let topSeparatorView: UIView = {
```

```
        let view = UIView()
        view.backgroundColor = .clear
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    private let settingsTable: UITableView = {
        let table = UITableView()
                        table.register(SettingCell.self, forCellReuseIdentifier:
SettingCell.identifier)
        table.showsVerticalScrollIndicator = false
        table.isScrollEnabled = false
        table.layer.cornerRadius = 15
        table.separatorInset = UIEdgeInsets(top: 0, left: 20, bottom: 0, right: 20)
        table.backgroundColor = UIColor(named: "tableColor")
        table.translatesAutoresizingMaskIntoConstraints = false
        return table
    }()

    private let bgImageView: UIImageView = {
        let imageView = UIImageView()
        imageView.image = UIImage(named: "background")
        return imageView
    }()
    //MARK: - viewDidLoad
    override func viewDidLoad() {
        super.viewDidLoad()
        // configute nav bar
        configureNavBar()
        // add subviews
```

```
        addSubviews()
        // apply constraints
        applyConstraints()
        // apply delegates
        applyTableDelegates()

    }

    //MARK: - viewDidLayout
    override func viewDidLayoutSubviews() {
        super.viewDidLayoutSubviews()
        bgImageView.frame = view.frame
    }

    //MARK: - Add subviews
    private func addSubviews() {
        view.addSubview(bgImageView)
        view.addSubview(topSeparatorView)
        view.addSubview(settingsTable)
    }

    //MARK: - Apply constraints
    private func applyConstraints() {
        // topSeparatorView constraints
        let topSeparatorViewConstraints = [
                        topSeparatorView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
                        topSeparatorView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
                            topSeparatorView.topAnchor.constraint(equalTo:
```

```
view.safeAreaLayoutGuide.topAnchor),
                topSeparatorView.heightAnchor.constraint(equalToConstant: 1)
            ]

            // // settingsTable constraints
            let settingsTableConstraints = [
                settingsTable.leadingAnchor.constraint(equalTo: view.leadingAnchor,
constant: 20),
                settingsTable.trailingAnchor.constraint(equalTo: view.trailingAnchor,
constant: -20),
                                                    settingsTable.topAnchor.constraint(equalTo:
topSeparatorView.bottomAnchor, constant: 20),
                                                settingsTable.bottomAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.bottomAnchor, constant: -10)
            ]

            // activate constraints
            NSLayoutConstraint.activate(topSeparatorViewConstraints)
            NSLayoutConstraint.activate(settingsTableConstraints)
        }

        //MARK: - Congifure nav bar
        private func configureNavBar() {
            let titleLbl: UILabel = {
                let lbl = UILabel()
                lbl.text = "Налаштування"
                lbl.textColor = .label
                lbl.font = UIFont.systemFont(ofSize: 35, weight: .bold)
                return lbl
            }()
```

```
                navigationItem.leftBarButtonItem = UIBarButtonItem(customView:
titleLbl)
            navigationController?.navigationBar.tintColor = .label
        }
    }
    //MARK: - Lifecycle
    extension SettingsVC {
        //MARK: - viewWillAppear
        override func viewWillAppear(_ animated: Bool) {
            super.viewWillAppear(animated)
            tabBarController?.tabBar.isHidden = false
        }
    }
    //MARK: - UITableViewDelegate & DataSource
    extension SettingsVC: UITableViewDelegate, UITableViewDataSource {
        // apply table delegates
        private func applyTableDelegates() {
            settingsTable.delegate = self
            settingsTable.dataSource = self
        }

        // nubers of rows
             func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
            return settings.count
        }

        // height for row
            func tableView(_ tableView: UITableView, heightForRowAt indexPath:
```

```
IndexPath) -> CGFloat {
            return 80
        }

        // cell for row
            func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
                    guard let cell = tableView.dequeueReusableCell(withIdentifier:
SettingCell.identifier) as? SettingCell else { return UITableViewCell() }
            if indexPath.row == settings.count - 1 {
               cell.separatorInset = UIEdgeInsets(top: 0, left: 0, bottom: 0, right: 1000)
            }
            let setting = settings[indexPath.row]
            cell.configure(with: setting)
            return cell
        }

        // did select row
            func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
            let cellType = settings[indexPath.row].type

            switch cellType {
            case .proposition:
                guard let url = URL(string: "https://t.me/+PVWEj8dQDSk5Mjgy") else
{ return }
                    if UIApplication.shared.canOpenURL(url) {
                        UIApplication.shared.open(url, options: [:], completionHandler:
nil)
                    }
```

```
            case .about:
                let vc = AboutVC()
                  navigationItem.backBarButtonItem = UIBarButtonItem(title: "", style:
.plain, target: nil, action: nil)
                navigationController?.pushViewController(vc, animated: true)
            case .favorite:
                let vc = FavoritesVC()
                  navigationItem.backBarButtonItem = UIBarButtonItem(title: "", style:
.plain, target: nil, action: nil)
                navigationController?.pushViewController(vc, animated: true)
            case .rate:
                print("rate")
            case .share:
                let text = "Проверьте это приложение!"
                let url = URL(string: "https://example.com")!
                        let vc = UIActivityViewController(activityItems: [text, url],
applicationActivities: [])
                navigationController?.present(vc, animated: true)
            case .thank:
                print("thank")
            }
        }

    }

    //
    //  ListVC.swift
    //  YeOzbroyenniaNS
    //
    //  Created by Oleksandr Smakhtin on 18.04.2023.
```

```
//
import UIKit
class ElementVC: UIViewController {

    //MARK: - Data
    public var element = Element(subcategory: "", items: [""], imagePath: [""])

    //MARK: - UI Objects
    private let topSeparatorView: UIView = {
        let view = UIView()
        view.backgroundColor = .clear
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    private let listTable: UITableView = {
        let table = UITableView()
        table.backgroundColor = UIColor(named: "tableColor")
                    table.register(ListCellWithPhoto.self,  forCellReuseIdentifier:
ListCellWithPhoto.identifier)
        table.register(ListCell.self, forCellReuseIdentifier: ListCell.identifier)
        table.layer.cornerRadius = 15
        table.separatorInset = UIEdgeInsets(top: 0, left: 20, bottom: 0, right: 20)
        table.showsVerticalScrollIndicator = false
        table.translatesAutoresizingMaskIntoConstraints = false
        return table
    }()

    private let bgImageView: UIImageView = {
        let imageView = UIImageView()
```

```
    imageView.image = UIImage(named: "background")
    return imageView
}()

//MARK: - viewDidLoad
override func viewDidLoad() {
    super.viewDidLoad()
    // bg color
    view.backgroundColor = .systemBackground
    // configure nav bar
    configureNavBar(with: element.subcategory)
    // add subviews
    addSubviews()
    // apply constraints
    applyConstraints()
    // apply delegates
    applyTableDelegates()
}

//MARK: - viewDidLayoutSubviews
override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    bgImageView.frame = view.frame
}

//MARK: - Add subviews
private func addSubviews() {
    view.addSubview(bgImageView)
    view.addSubview(topSeparatorView)
    view.addSubview(listTable)
```

```
        }

        //MARK: - Apply constraints
        private func applyConstraints() {
            // topSeparatorView constraints
            let topSeparatorViewConstraints = [
                                topSeparatorView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
                                topSeparatorView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
                                        topSeparatorView.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor, constant: 1),
                    topSeparatorView.heightAnchor.constraint(equalToConstant: 1)
            ]

            // listTable constraints
            let listTableConstraints = [
                        listTable.leadingAnchor.constraint(equalTo:  view.leadingAnchor,
constant: 20),
                        listTable.trailingAnchor.constraint(equalTo:  view.trailingAnchor,
constant: -20),
                                                listTable.topAnchor.constraint(equalTo:
topSeparatorView.bottomAnchor, constant: 20),
                        listTable.bottomAnchor.constraint(equalTo:  view.bottomAnchor,
constant: -20)
                                                //listTable.bottomAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.bottomAnchor)
            ]

            // activate constraints
```

```
            NSLayoutConstraint.activate(topSeparatorViewConstraints)
            NSLayoutConstraint.activate(listTableConstraints)
        }


        //MARK: - Congifure nav bar
        public func configureNavBar(with title: String) {
            self.title = title
            navigationController?.navigationBar.tintColor = .label
        }
    }
    //MARK: - Lifecycle
    extension ElementVC {
        override func viewWillAppear(_ animated: Bool) {
            super.viewWillAppear(animated)
            tabBarController?.tabBar.isHidden = true
        }
    }
    //MARK: - UITableViewDelegate & DataSource
    extension ElementVC: UITableViewDelegate, UITableViewDataSource {
        // apply table delegates
        private func applyTableDelegates() {
            listTable.delegate = self
            listTable.dataSource = self
        }

        // nubers of rows
            func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
            return element.items.count
```

```
        }

        // cell for row
            func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

            if let imagePath = element.imagePath?[indexPath.row] {
                    guard let cell = tableView.dequeueReusableCell(withIdentifier: ListCellWithPhoto.identifier) as? ListCellWithPhoto else { return UITableViewCell() }
                        cell.configure(with: element.items[indexPath.row], image: element.imagePath?[indexPath.row])
                return cell
            } else {
                    guard let cell = tableView.dequeueReusableCell(withIdentifier: ListCell.identifier) as? ListCell else { return UITableViewCell() }
                        //cell.configure(with: element.items[indexPath.row], image: element.imagePath?[indexPath.row])
                cell.configure(with: element.items[indexPath.row])
                return cell
            }


        //          guard let cell = tableView.dequeueReusableCell(withIdentifier: ListCellWithPhoto.identifier) as? ListCellWithPhoto else { return UITableViewCell() }
        //              cell.configure(with: element.items[indexPath.row], image: element.imagePath?[indexPath.row])
        //      return cell
            }
```

```
        // height for row
            func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
            return 80
        }

        // did select row
            func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
            tableView.deselectRow(at: indexPath, animated: true)

            let itemTitle = element.items[indexPath.row]
            let item = DataManager.shared.getItem(by: itemTitle)

            let vc = ItemVC()
            //vc.item = item
                                        vc.configureVc(with: item, imagePath: element.imagePath?[indexPath.row])
            navigationController?.pushViewController(vc, animated: true)

        }

    }

    //
    //  SubcategoryVC.swift
    //  YeOzbroyenniaNS
    //
    //  Created by Oleksandr Smakhtin on 18.04.2023.
```

```
//
import UIKit
class SubcategoryVC: UIViewController {

    //MARK: - Data
    public var subcategory = Subcategory(category: "", subcategories: [""])

    //MARK: - UI Objects
    private let topSeparatorView: UIView = {
        let view = UIView()
        view.backgroundColor = .clear
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    private let listTable: UITableView = {
        let table = UITableView()
        table.backgroundColor = UIColor(named: "tableColor")
        table.register(ListCell.self, forCellReuseIdentifier: ListCell.identifier)
        table.showsVerticalScrollIndicator = false
        table.layer.cornerRadius = 15
        table.separatorInset = UIEdgeInsets(top: 0, left: 20, bottom: 0, right: 20)
        table.translatesAutoresizingMaskIntoConstraints = false
        return table
    }()

    private let bgImageView: UIImageView = {
        let imageView = UIImageView()
        imageView.image = UIImage(named: "background")
        return imageView
```

```
}()

//MARK: - viewDidLoad
override func viewDidLoad() {
    super.viewDidLoad()
    // configure nav bar
    configureNavBar(with: subcategory.category)
    // add subviews
    addSubviews()
    // apply constraints
    applyConstraints()
    // apply delegates
    applyTableDelegates()
}

//MARK: - viewDidLayoutSubviews
override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    bgImageView.frame = view.frame
}

//MARK: - Add subviews
private func addSubviews() {
    view.addSubview(bgImageView)
    view.addSubview(topSeparatorView)
    view.addSubview(listTable)
}

//MARK: - Apply constraints
private func applyConstraints() {
```

```
        // topSeparatorView constraints
        let topSeparatorViewConstraints = [
                                topSeparatorView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
                                topSeparatorView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
                                    topSeparatorView.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor, constant: 1),
            topSeparatorView.heightAnchor.constraint(equalToConstant: 1)
        ]

        // listTable constraints
        let listTableConstraints = [
                listTable.leadingAnchor.constraint(equalTo:  view.leadingAnchor,
constant: 20),
                 listTable.trailingAnchor.constraint(equalTo:  view.trailingAnchor,
constant: -20),
                                            listTable.topAnchor.constraint(equalTo:
topSeparatorView.bottomAnchor, constant: 20),
                 listTable.bottomAnchor.constraint(equalTo:  view.bottomAnchor,
constant: -20)
        ]

        // activate constraints
        NSLayoutConstraint.activate(topSeparatorViewConstraints)
        NSLayoutConstraint.activate(listTableConstraints)
    }


    //MARK: - Congifure nav bar
```

```
    public func configureNavBar(with title: String) {
        self.title = title
        navigationController?.navigationBar.tintColor = .label
    }

}
//MARK: - Lifecycle
extension SubcategoryVC {
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        tabBarController?.tabBar.isHidden = true
    }
}
//MARK: - UITableViewDelegate & DataSource
extension SubcategoryVC: UITableViewDelegate, UITableViewDataSource {
    // apply table delegates
    private func applyTableDelegates() {
        listTable.delegate = self
        listTable.dataSource = self
    }

    // nubers of rows
        func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return subcategory.subcategories.count
    }

    // cell for row
        func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
```

```
                guard let cell = tableView.dequeueReusableCell(withIdentifier:
ListCell.identifier) as? ListCell else { return UITableViewCell() }
            cell.configure(with: subcategory.subcategories[indexPath.row])
            return cell
        }

        // height for row
            func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) -> CGFloat {
            return 80
        }

        // did select row
            func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
            let model = subcategory.subcategories[indexPath.row]
            let element = DataManager.shared.getElements(by: model)

            let vc = ElementVC()
            vc.element = element
              navigationItem.backBarButtonItem = UIBarButtonItem(title: title, style:
.plain, target: nil, action: nil)
            navigationController?.pushViewController(vc, animated: true)
        }

    }

    //
    //  ItemVC.swift
    //  YeOzbroyenniaNS
```

```
//
//  Created by Oleksandr Smakhtin on 18.04.2023.
//
import UIKit
class ItemVC: UIViewController {

    //MARK: - Tracker
    private var isItemExists: Bool?

    //MARK: - Data
    public var item = Item(item: "", property: [Property]())
    public var imagePath: String?

    //MARK: - UI Objects
    private let topSeparatorView: UIView = {
        let view = UIView()
        view.backgroundColor = .clear
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    private let underlineView: UIView = {
        let view = UIView()
        view.backgroundColor = .white.withAlphaComponent(0.7)
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    private let itemTitle: UILabel = {
        let lbl = UILabel()
```

```
    lbl.textColor = .label
    lbl.numberOfLines = 2
    lbl.textAlignment = .center
    lbl.font = UIFont.systemFont(ofSize: 20, weight: .bold)
    lbl.translatesAutoresizingMaskIntoConstraints = false
    return lbl
}()

private let itemImageView: UIImageView = {
    let view = UIImageView()
    view.contentMode = .scaleAspectFit
    view.translatesAutoresizingMaskIntoConstraints = false
    return view
}()

private let circleContentView: UIView = {
    let view = UIView()
    view.layer.cornerRadius = 15
    view.backgroundColor = UIColor(named: "tableColor")
    view.translatesAutoresizingMaskIntoConstraints = false
    return view
}()

private let itemTable: UITableView = {
    let table = UITableView()
    table.backgroundColor = .clear
    table.layer.cornerRadius = 5
    table.register(ItemCell.self, forCellReuseIdentifier: ItemCell.identifier)
    table.showsVerticalScrollIndicator = false
    table.translatesAutoresizingMaskIntoConstraints = false
```

```
    return table
}()

private let bgImageView: UIImageView = {
    let imageView = UIImageView()
    imageView.image = UIImage(named: "background")
    return imageView
}()
//MARK: - viewDidLoad
override func viewDidLoad() {
    super.viewDidLoad()
    // chech item
    isItemExists = CoreDataManager.shared.isItemExists(with: item.item)
    // configure nav bar
    configureNavBar()
    // add subviews
    addSubviews()
    // apply constraints
    //applyConstraints()
    // apply delegates
    applyTableDelegates()
    // set title
    itemTitle.text = item.item
}

//MARK: - viewDidLayoutSubviews
override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    bgImageView.frame = view.frame
}
```

```
        //MARK: - Add subviews
        private func addSubviews() {
            view.addSubview(bgImageView)
            view.addSubview(topSeparatorView)
            view.addSubview(itemTitle)
            view.addSubview(underlineView)
            view.addSubview(circleContentView)
            view.addSubview(itemTable)
        }

        //MARK: - Apply constraints
        private func applyConstraints() {
            // topSeparatorView constraints
            let topSeparatorViewConstraints = [
                                topSeparatorView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
                                topSeparatorView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
                                        topSeparatorView.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor, constant: 1),
                topSeparatorView.heightAnchor.constraint(equalToConstant: 1)
            ]


            // itemTitle constraints
            let itemTitleConstraints = [
                itemTitle.centerXAnchor.constraint(equalTo: view.centerXAnchor),
                        itemTitle.leadingAnchor.constraint(equalTo:  view.leadingAnchor,
constant: 20),
```

```
                itemTitle.trailingAnchor.constraint(equalTo:  view.trailingAnchor,
constant: -20),
                                                itemTitle.topAnchor.constraint(equalTo:
topSeparatorView.bottomAnchor, constant: 10)
            ]

            // underlineView constraints
            let underlineViewConstraints = [
                underlineView.heightAnchor.constraint(equalToConstant: 1),
                 underlineView.topAnchor.constraint(equalTo: itemTitle.bottomAnchor,
constant: 15),
                underlineView.widthAnchor.constraint(equalToConstant: 100),
                                    underlineView.centerXAnchor.constraint(equalTo:
view.centerXAnchor)
            ]


        //        if let image = imagePath {
        //            itemImageView.image = UIImage(named: image)
        //            NSLayoutConstraint.activate([
        //                                    itemImageView.centerXAnchor.constraint(equalTo:
view.centerXAnchor),
        //            itemImageView.heightAnchor.constraint(equalToConstant: 150),
        //            itemImageView.widthAnchor.constraint(equalToConstant: 150),
        //                                        itemImageView.topAnchor.constraint(equalTo:
underlineView.bottomAnchor, constant: 20),
        //            ])
        //        }

            // circleContentView constraints
```

```
        let circleContentViewConstraints = [
                                circleContentView.leadingAnchor.constraint(equalTo:
view.leadingAnchor, constant: 20),
                                circleContentView.trailingAnchor.constraint(equalTo:
view.trailingAnchor, constant: -20),
                                circleContentView.bottomAnchor.constraint(equalTo:
view.bottomAnchor, constant: -20)
        ]

    //      if let image = imagePath {
    //          NSLayoutConstraint.activate([
    //                                  circleContentView.topAnchor.constraint(equalTo:
itemImageView.bottomAnchor, constant: 20)
    //          ])
    //      } else {
    //          NSLayoutConstraint.activate([
    //                                  circleContentView.topAnchor.constraint(equalTo:
underlineView.bottomAnchor, constant: 60)
    //          ])
    //      }


        // itemTable constraints
        let itemTableConstraints = [
                                    itemTable.leadingAnchor.constraint(equalTo:
circleContentView.leadingAnchor, constant: 16),
                                    itemTable.trailingAnchor.constraint(equalTo:
circleContentView.trailingAnchor, constant: -16),
            itemTable.topAnchor.constraint(equalTo: circleContentView.topAnchor,
constant: 16),
```

```
                                        itemTable.bottomAnchor.constraint(equalTo:
circleContentView.bottomAnchor)
            ]

            // activate constraints
            NSLayoutConstraint.activate(topSeparatorViewConstraints)
            NSLayoutConstraint.activate(itemTitleConstraints)
            NSLayoutConstraint.activate(underlineViewConstraints)
            if let image = imagePath {
               itemImageView.image = UIImage(named: image)
               NSLayoutConstraint.activate([
                                    itemImageView.centerXAnchor.constraint(equalTo:
view.centerXAnchor),
               itemImageView.heightAnchor.constraint(equalToConstant: 150),
               itemImageView.widthAnchor.constraint(equalToConstant: 150),
                                        itemImageView.topAnchor.constraint(equalTo:
underlineView.bottomAnchor, constant: 20),
               ])
            }
            NSLayoutConstraint.activate(circleContentViewConstraints)
            if let image = imagePath {
               NSLayoutConstraint.activate([
                                    circleContentView.topAnchor.constraint(equalTo:
itemImageView.bottomAnchor, constant: 20)
               ])
            } else {
               NSLayoutConstraint.activate([
                                    circleContentView.topAnchor.constraint(equalTo:
underlineView.bottomAnchor, constant: 60)
               ])
```

```
        }
        NSLayoutConstraint.activate(itemTableConstraints)
    }


    public func configureVc(with item: Item, imagePath: String?) {
        self.item = item
        self.imagePath = imagePath
        if let image = imagePath {
            view.addSubview(itemImageView)
        }
        applyConstraints()
    }

    //MARK: - Configure nav bar
    private func configureNavBar() {
        if isItemExists! {
                navigationItem.rightBarButtonItem = UIBarButtonItem(image: UIImage(systemName: "heart.fill", withConfiguration: UIImage.SymbolConfiguration(pointSize: 22)), style: .plain, target: self, action: #selector(addOrDeleteFavoritesAction))
        } else {
                navigationItem.rightBarButtonItem = UIBarButtonItem(image: UIImage(systemName: "heart", withConfiguration: UIImage.SymbolConfiguration(pointSize: 22)), style: .plain, target: self, action: #selector(addOrDeleteFavoritesAction))
        }
    }

    //MARK: - Action
```

```
        @objc private func addOrDeleteFavoritesAction() {

                let itemActualStatus = CoreDataManager.shared.isItemExists(with:
item.item)

            if itemActualStatus {
                print("Item exists. Deleting...")
                CoreDataManager.shared.deleteItem(with: item.item)
                    navigationItem.rightBarButtonItem?.image = UIImage(systemName:
"heart", withConfiguration: UIImage.SymbolConfiguration(pointSize: 22))
            } else {
                print("Item does not exist. Saving...")
                CoreDataManager.shared.addItem(with: item.item)
                    navigationItem.rightBarButtonItem?.image = UIImage(systemName:
"heart.fill", withConfiguration: UIImage.SymbolConfiguration(pointSize: 22))
            }
        }
    }
    //MARK: - Lifecycle
    extension ItemVC {
        //MARK: - viewWillAppear
        override func viewWillAppear(_ animated: Bool) {
            super.viewWillAppear(animated)
            tabBarController?.tabBar.isHidden = true

            isItemExists = CoreDataManager.shared.isItemExists(with: item.item)
            print("Item exists?: \(isItemExists)")
        }
    }
    //MARK: - UITableViewDelegate & DataSource
```

```
extension ItemVC: UITableViewDelegate, UITableViewDataSource {
    // apply table delegates
    private func applyTableDelegates() {
        itemTable.delegate = self
        itemTable.dataSource = self
    }

    // nubers of rows
        func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return item.property.count
    }

    // cell for row
        func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
                guard let cell = tableView.dequeueReusableCell(withIdentifier: ItemCell.identifier) as? ItemCell else { return UITableViewCell() }
        let properties = item.property
                    cell.configure(with: properties[indexPath.row].name, value: properties[indexPath.row].value)
        return cell
    }
}

//
//  SearchResultsVC.swift
//  YeOzbroyenniaNS
//
//  Created by Oleksandr Smakhtin on 20.04.2023.
```

```
//
import UIKit
protocol SearchResultDelegate: AnyObject {
    func didSelectItem(with title: String)
}
class SearchResultsVC: UIViewController {

    //MARK: - Delegate
    weak var delegate: SearchResultDelegate?

    //MARK: - Data
    public var searchedData = [String]() {
        didSet {
            DispatchQueue.main.async {
                self.searchTable.reloadData()
            }
        }
    }
    //MARK: - UI Objects
    private let topSeparatorView: UIView = {
        let view = UIView()
        view.backgroundColor = .clear
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    private let searchTable: UITableView = {
        let table = UITableView()
        table.register(ListCell.self, forCellReuseIdentifier: ListCell.identifier)
        table.showsVerticalScrollIndicator = false
```

```
    table.layer.cornerRadius = 15
    table.separatorInset = UIEdgeInsets(top: 0, left: 20, bottom: 0, right: 20)
    table.backgroundColor = UIColor(named: "tableColor")
    table.translatesAutoresizingMaskIntoConstraints = false
    return table
}()

private let bgImageView: UIImageView = {
    let imageView = UIImageView()
    imageView.image = UIImage(named: "background")
    return imageView
}()

//MARK: - viewDidLoad
override func viewDidLoad() {
    super.viewDidLoad()
    // add subviews
    addSubviews()
    // apply constraints
    applyConstraints()
    // apply delegates
    applyTableDelegates()

}

//MARK: - viewDidLayoutSubviews
override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    bgImageView.frame = view.frame
}
```

```
        //MARK: - Add subviews
        private func addSubviews() {
            view.addSubview(bgImageView)
            view.addSubview(topSeparatorView)
            view.addSubview(searchTable)
        }

        //MARK: - Apply constraints
        private func applyConstraints() {
            // topSeparatorView constraints
            let topSeparatorViewConstraints = [
                                topSeparatorView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
                                topSeparatorView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
                                    topSeparatorView.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor),
                topSeparatorView.heightAnchor.constraint(equalToConstant: 1)
            ]

            // searchTable constraints
            let searchTableConstraints = [
                    searchTable.leadingAnchor.constraint(equalTo: view.leadingAnchor,
constant: 20),
                    searchTable.trailingAnchor.constraint(equalTo: view.trailingAnchor,
constant: -20),
                                            searchTable.topAnchor.constraint(equalTo:
topSeparatorView.bottomAnchor, constant: 20),
                                        searchTable.bottomAnchor.constraint(equalTo:
```

```
view.safeAreaLayoutGuide.bottomAnchor, constant: -10)
            ]

            // activate constraints
            NSLayoutConstraint.activate(topSeparatorViewConstraints)
            NSLayoutConstraint.activate(searchTableConstraints)
        }
    }
    //MARK: - UITableViewDelegate & DataSource
    extension SearchResultsVC: UITableViewDelegate, UITableViewDataSource {
        // apply table delegates
        private func applyTableDelegates() {
            searchTable.delegate = self
            searchTable.dataSource = self
        }

        // nubers of rows
            func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
            return searchedData.count
        }

        // cell for row
            func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
                    guard let cell = tableView.dequeueReusableCell(withIdentifier:
ListCell.identifier) as? ListCell else { return UITableViewCell() }
            cell.configure(with: searchedData[indexPath.row])
            return cell
        }
```

```
        // height for row
            func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
            return 80
        }

        // did select row
            func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
            delegate?.didSelectItem(with: searchedData[indexPath.row])
    //        tableView.deselectRow(at: indexPath, animated: true)
    //
    //        let itemTitle = searchedData[indexPath.row]
    //        let item = DataManager.shared.getItem(by: itemTitle)
    //        let vc = ItemVC()
    //        vc.item = item
    //         navigationItem.backBarButtonItem = UIBarButtonItem(title: "Пошук", style: .plain, target: nil, action: nil)
    //        navigationController?.pushViewController(vc, animated: true)
        }
    }

    //
    //  FavoritesVC.swift
    //  YeOzbroyenniaNS
    //
    //  Created by Oleksandr Smakhtin on 20.04.2023.
    //
    import UIKit
```

```
class FavoritesVC: UIViewController {

    //MARK: - Data
    private var favorites = [String]()
    //MARK: - UI Objects
    private let topSeparatorView: UIView = {
        let view = UIView()
        view.backgroundColor = .clear
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    private let listTable: UITableView = {
        let table = UITableView()
        table.backgroundColor = UIColor(named: "tableColor")
        table.register(ListCell.self, forCellReuseIdentifier: ListCell.identifier)
        table.showsVerticalScrollIndicator = false
        table.layer.cornerRadius = 15
        table.separatorInset = UIEdgeInsets(top: 0, left: 20, bottom: 0, right: 20)
        table.translatesAutoresizingMaskIntoConstraints = false
        return table
    }()

    private let isEmptyLbl: UILabel = {
        let lbl = UILabel()
        lbl.text = "Схоже, що у вас поки що немає обраних"
        lbl.textColor = .black
        lbl.numberOfLines = 0
        lbl.textAlignment = .center
        lbl.font = UIFont.systemFont(ofSize: 20)
```

```
    lbl.translatesAutoresizingMaskIntoConstraints = false
    return lbl
}()

private let bgImageView: UIImageView = {
    let imageView = UIImageView()
    imageView.image = UIImage(named: "background")
    return imageView
}()

//MARK: - viewDidLoad
override func viewDidLoad() {
    super.viewDidLoad()
    // configure nav bar
    configureNavBar()
    // add subviews
    addSubviews()
    // apply constraints
    applyConstraints()
    // apply delegates
    applyTableDelegates()
}

//MARK: - viewDidLayoutSubviews
override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    bgImageView.frame = view.frame
}
//MARK: - Add subviews
private func addSubviews() {
```

```
        view.addSubview(bgImageView)
        view.addSubview(isEmptyLbl)
        view.addSubview(topSeparatorView)
        view.addSubview(listTable)
    }

    //MARK: - Apply constraints
    private func applyConstraints() {
        // topSeparatorView constraints
        let topSeparatorViewConstraints = [
                        topSeparatorView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
                        topSeparatorView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
                            topSeparatorView.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor, constant: 1),
            topSeparatorView.heightAnchor.constraint(equalToConstant: 1)
        ]

        // isEmptyLbl constraints
        let isEmptyLblConstraints = [
            isEmptyLbl.centerXAnchor.constraint(equalTo: view.centerXAnchor),
            isEmptyLbl.centerYAnchor.constraint(equalTo: view.centerYAnchor),
            isEmptyLbl.widthAnchor.constraint(equalToConstant: 250)
        ]

        // listTable constraints
        let listTableConstraints = [
                listTable.leadingAnchor.constraint(equalTo:  view.leadingAnchor,
constant: 20),
```

```
                listTable.trailingAnchor.constraint(equalTo: view.trailingAnchor,
constant: -20),
                                                listTable.topAnchor.constraint(equalTo:
topSeparatorView.bottomAnchor, constant: 20),
                listTable.bottomAnchor.constraint(equalTo: view.bottomAnchor,
constant: -20)
            ]

            // activate constraints
            NSLayoutConstraint.activate(topSeparatorViewConstraints)
            NSLayoutConstraint.activate(isEmptyLblConstraints)
            NSLayoutConstraint.activate(listTableConstraints)
        }

        //MARK: - Configure nav bar
        private func configureNavBar() {
            title = "Обрані"
                navigationItem.rightBarButtonItem = UIBarButtonItem(image:
UIImage(systemName: "square.and.pencil"), style: .plain, target: self, action:
#selector(editAction))
        }

        @objc private func editAction() {
            guard !favorites.isEmpty else { return }

            if listTable.isEditing {
                listTable.isEditing = false

                listTable.reloadData()
                    navigationItem.rightBarButtonItem = UIBarButtonItem(image:
```

```
UIImage(systemName: "square.and.pencil"), style: .plain, target: self, action: #selector(editAction))
            } else {
                listTable.isEditing = true

                listTable.reloadData()
                 navigationItem.rightBarButtonItem = UIBarButtonItem(title: "Готово", style: .plain, target: self, action: #selector(editAction))
            }
        }
    }
    //MARK: - Lifecycle
    extension FavoritesVC {
        override func viewWillAppear(_ animated: Bool) {
            super.viewWillAppear(animated)
            tabBarController?.tabBar.isHidden = true

            favorites = CoreDataManager.shared.fetchItems()
            print(favorites)
            DispatchQueue.main.async {
                self.listTable.reloadData()
            }

            if favorites.isEmpty {
                isEmptyLbl.isHidden = false
            } else {
                isEmptyLbl.isHidden = true
            }
        }
    }
```

```
//MARK: - UITableViewDelegate & DataSource
extension FavoritesVC: UITableViewDelegate, UITableViewDataSource {
    // apply table delegates
    private func applyTableDelegates() {
        listTable.delegate = self
        listTable.dataSource = self
    }

    // nubers of rows
        func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return favorites.count
    }

    // cell for row
        func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
            guard let cell = tableView.dequeueReusableCell(withIdentifier: ListCell.identifier) as? ListCell else { return UITableViewCell() }
        cell.configure(with: favorites[indexPath.row])
        return cell
    }

    // height for row
        func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return 80
    }

    // did select row
```

```
        func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
            tableView.deselectRow(at: indexPath, animated: true)
            let itemTitle = favorites[indexPath.row]
            let item = DataManager.shared.getItem(by: itemTitle)
            let vc = ItemVC()
            //vc.item = item
            vc.configureVc(with: item, imagePath: nil)
            navigationController?.pushViewController(vc, animated: true)
        }
        // trailing swipe action
                        func tableView(_ tableView: UITableView, trailingSwipeActionsConfigurationForRowAt indexPath: IndexPath) -> UISwipeActionsConfiguration? {

            let deleteAction = UIContextualAction(style: .destructive, title: nil) { _, _, completionHandler in
                // delete item from data
                let item = self.favorites[indexPath.row]
                CoreDataManager.shared.deleteItem(with: item)
                self.favorites = CoreDataManager.shared.fetchItems()

                // delete row
                tableView.deleteRows(at: [indexPath], with: .fade)

                // show emtyLbl if needed
                if self.favorites.isEmpty {
                    self.isEmptyLbl.isHidden = false
                    self.listTable.isEditing = false
                      self.navigationItem.rightBarButtonItem = UIBarButtonItem(image:
```

```
UIImage(systemName: "square.and.pencil"), style: .plain, target: self, action: #selector(self.editAction))
                }

                completionHandler(true)
            }
            deleteAction.image = UIImage(systemName: "trash")
            deleteAction.backgroundColor = .systemRed

            let configuration = UISwipeActionsConfiguration(actions: [deleteAction])
            return configuration

        }
    }

    //
    //  AboutVC.swift
    //  YeOzbroyenniaNS
    //
    //  Created by Oleksandr Smakhtin on 24.04.2023.
    //
    import UIKit
    class AboutVC: UIViewController {

        //MARK: - Data


        //MARK: - UI Objects
        private let aboutTextView: UITextView = {
            let textView = UITextView()
```

```
        textView.backgroundColor = .clear
        textView.font = UIFont.systemFont(ofSize: 18)
         textView.text = "  єЗброя - додаток, що створений з метою полегшити процес вибору та використання озброєння та техніки військовими. \n\n  Списки зброї та техніки за категоріями та їх характеристики дозволяють знайти потрібний елемент, а функція пошуку за назвою зробить процес ще швидшим та ефективнішим. \n\n  Додаток працює без інтернету, що дозволяє військовим мати доступ до необхідної інформації в будь-який момент. \n\n\n Слава Україні🇺🇦"
        textView.translatesAutoresizingMaskIntoConstraints = false
        return textView
    }()

    private let cornerView: UIView = {
        let view = UIView()
        view.backgroundColor = UIColor(named: "tableColor")
        view.layer.cornerRadius = 15
        view.translatesAutoresizingMaskIntoConstraints = false
        return view
    }()

    private let bgImageView: UIImageView = {
        let imageView = UIImageView()
        imageView.image = UIImage(named: "background")
        return imageView
    }()
    //MARK: - viewDidLoad
    override func viewDidLoad() {
        super.viewDidLoad()
        // configure nav bar
```

```
        configureNavBar()
        // add subviews
        addSubviews()
        // apply constraints
        applyConstraints()
    }

    //MARK: - viewDidLayoutSubviews
    override func viewDidLayoutSubviews() {
        super.viewDidLayoutSubviews()
        bgImageView.frame = view.frame
    }

    //MARK: - Add subviews
    private func addSubviews() {
        view.addSubview(bgImageView)
        view.addSubview(cornerView)
        view.addSubview(aboutTextView)
    }

    //MARK: - Apply constraints
    private func applyConstraints() {
        // cornerView constraints
        let cornerViewConstraints = [
            cornerView.leadingAnchor.constraint(equalTo: view.leadingAnchor,
constant: 20),
            cornerView.trailingAnchor.constraint(equalTo: view.trailingAnchor,
constant: -20),
                                        cornerView.topAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.topAnchor, constant: 20),
```

```
                cornerView.bottomAnchor.constraint(equalTo: view.bottomAnchor,
constant: -20)
            ]

            // aboutTextView constraints
            let aboutTextViewConstraints = [
                                aboutTextView.leadingAnchor.constraint(equalTo:
cornerView.leadingAnchor, constant: 10),
                                aboutTextView.trailingAnchor.constraint(equalTo:
cornerView.trailingAnchor, constant: -10),
                aboutTextView.topAnchor.constraint(equalTo: cornerView.topAnchor,
constant: 10),
                                aboutTextView.bottomAnchor.constraint(equalTo:
cornerView.bottomAnchor, constant: -10)
            ]

            // activate constrants
            NSLayoutConstraint.activate(cornerViewConstraints)
            NSLayoutConstraint.activate(aboutTextViewConstraints)
        }

        //MARK: - Congifure nav bar
        private func configureNavBar() {
            title = "Про додаток"
        }
    }
    //MARK: - Lifecycle
    extension AboutVC {
        override func viewWillAppear(_ animated: Bool) {
            super.viewWillAppear(animated)
```

```
            tabBarController?.tabBar.isHidden = true
        }
    }

    //
    //  TabBarController.swift
    //  YeOzbroyenniaNS
    //
    //  Created by Oleksandr Smakhtin on 16.04.2023.
    //
    import UIKit
    class TabBarController: UITabBarController {

        //MARK: - Tracker
        private var selectedTab = 0 {
            didSet {
                for i in 0...3 {

                    UIView.animate(withDuration: 0.3, delay: 0, options: .curveEaseOut)
{ [weak self] in
                                    self?.centerXAnchorsForIndicator[i].isActive = i ==
self?.selectedTab ? true : false
                        self?.tabBar.layoutIfNeeded()
                    } completion: { _ in }

                }
            }
        }

        //MARK: - Array of constraints for indicator
```

```
        private var centerXAnchorsForIndicator: [NSLayoutConstraint] = []
        private var topAnchorsForIndicator: [NSLayoutConstraint] = []

        //MARK: - UI Objects
        var indicatorView: UIView = {
            let view = UIView()
            view.backgroundColor = .black.withAlphaComponent(0.7)
            view.layer.cornerRadius = 2.5
            view.translatesAutoresizingMaskIntoConstraints = false
            return view
        }()


        //MARK: - viewDidLoad
        override func viewDidLoad() {
            super.viewDidLoad()
            // bg color
            view.backgroundColor = .systemBackground
            // configure nav bar
              //configureTabBar()
            // set controllers
            let vc1  = UINavigationController(rootViewController: GunVC())
            vc1.tabBarItem.image = UIImage(named: "gun")
            vc1.tabBarItem.selectedImage = UIImage(named: "gunFill")
              vc1.tabBarItem.imageInsets = UIEdgeInsets(top: 18, left: 18, bottom: 18,
right: 18)
            vc1.tabBarItem.title = "Зброя"

            let vc2  = UINavigationController(rootViewController: TechniqueVC())
            vc2.tabBarItem.image = UIImage(named: "teq")
```

```
        vc2.tabBarItem.selectedImage = UIImage(named: "teqFill")
         vc2.tabBarItem.imageInsets = UIEdgeInsets(top: 14, left: 14, bottom: 14,
right: 14)
        vc2.tabBarItem.title = "Техніка"

        let vc3 = UINavigationController(rootViewController: SearchVC())
        vc3.tabBarItem.image = UIImage(systemName: "magnifyingglass")
                    vc3.tabBarItem.selectedImage = UIImage(systemName:
"magnifyingglass", withConfiguration: UIImage.SymbolConfiguration(weight:
.bold))
        vc3.tabBarItem.title = "Пошук"

        let vc4 = UINavigationController(rootViewController: SettingsVC())
        vc4.tabBarItem.image = UIImage(systemName: "gearshape")
        vc4.tabBarItem.selectedImage = UIImage(systemName: "gearshape.fill")
        vc4.tabBarItem.title = "Налаштування"

        setViewControllers([vc1, vc2, vc3, vc4], animated: true)
        configureTabBar()
    }

    //MARK: - Configure nav bar
    private func configureTabBar() {
        tabBar.tintColor = .black
        tabBar.unselectedItemTintColor = .clear.withAlphaComponent(0.5)
        tabBar.addSubview(indicatorView)
        applyConstraints()
    }

    //MARK: - Apply constraints
```

```
        private func applyConstraints() {
            // get indicator constraints
            for (index, item) in tabBar.subviews.enumerated() {
                if index == tabBar.subviews.count - 1 {
                    continue
                }
                 let centerXAnchor = indicatorView.centerXAnchor.constraint(equalTo:
item.centerXAnchor)
                centerXAnchorsForIndicator.append(centerXAnchor)
                        let topAnchor = indicatorView.topAnchor.constraint(equalTo:
item.bottomAnchor, constant: 5)
                topAnchorsForIndicator.append(topAnchor)
            }

            // indicatorView constraints
            let indicatorViewConstraints = [
                centerXAnchorsForIndicator[0],
                topAnchorsForIndicator[0],
                indicatorView.heightAnchor.constraint(equalToConstant: 5),
                indicatorView.widthAnchor.constraint(equalToConstant: 5)
            ]

            // activate constraints
            NSLayoutConstraint.activate(indicatorViewConstraints)
        }

        //MARK: - Did select item
        override func tabBar(_ tabBar: UITabBar, didSelect item: UITabBarItem) {
            // animation
            if let index = tabBar.items?.firstIndex(of: item) {
```

```
            animateToTab(toIndex: index)
        }

        // indicator changes
        switch item.title {
        case "Зброя":
            selectedTab = 0
        case "Техніка":
            selectedTab = 1
        case "Пошук":
            selectedTab = 2
        case "Налаштування":
            selectedTab = 3
        default:
            selectedTab = 0
        }
    }

    //MARK: - Transition animation
    func animateToTab(toIndex: Int) {
        guard let fromView = selectedViewController?.view, let toView = viewControllers?[toIndex].view, fromView != toView else {return}
        UIView.transition(from: fromView, to: toView, duration: 0.2, options: [.transitionCrossDissolve], completion: nil)
    }
}
//
//  AppDelegate.swift
//  YeOzbroyenniaNS
//
```

```
//  Created by Oleksandr Smakhtin on 16.04.2023.
//
import UIKit
import CoreData
@main
class AppDelegate: UIResponder, UIApplicationDelegate {
                        func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        return true
    }
    // MARK: UISceneSession Lifecycle
     func application(_ application: UIApplication, configurationForConnecting connectingSceneSession: UISceneSession, options: UIScene.ConnectionOptions) -> UISceneConfiguration {
        // Called when a new scene session is being created.
        // Use this method to select a configuration to create the new scene with.
                return UISceneConfiguration(name: "Default Configuration", sessionRole: connectingSceneSession.role)
    }
        func application(_ application: UIApplication, didDiscardSceneSessions sceneSessions: Set<UISceneSession>) {
    }


    // MARK: - Core Data stack
    lazy var persistentContainer: NSPersistentContainer = {
        /*
         The persistent container for the application. This implementation
```

```
             creates and returns a container, having loaded the store for the
             application to it. This property is optional since there are legitimate
             error conditions that could cause the creation of the store to fail.
            */
            let container = NSPersistentContainer(name: "YeOzbroenniaModel")
                container.loadPersistentStores(completionHandler: { (storeDescription,
error) in
                if let error = error as NSError? {
                        // Replace this implementation with code to handle the error
appropriately.
                        // fatalError() causes the application to generate a crash log and
terminate. You should not use this function in a shipping application, although it may
be useful during development.

                    /*
                     Typical reasons for an error here include:
                     * The parent directory does not exist, cannot be created, or disallows
writing.
                        * The persistent store is not accessible, due to permissions or data
protection when the device is locked.
                     * The device is out of space.
                     * The store could not be migrated to the current model version.
                     Check the error message to determine what the actual problem was.
                     */
                    fatalError("Unresolved error \(error), \(error.userInfo)")
                }
            })
            return container
        }()
        // MARK: - Core Data Saving support
```

```
    func saveContext () {
        let context = persistentContainer.viewContext
        if context.hasChanges {
            do {
                try context.save()
            } catch {
                    // Replace this implementation with code to handle the error appropriately.
                    // fatalError() causes the application to generate a crash log and terminate. You should not use this function in a shipping application, although it may be useful during development.
                let nserror = error as NSError
                fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
            }
        }
    }
}
//
//  SceneDelegate.swift
//  YeOzbroyenniaNS
//
//  Created by Oleksandr Smakhtin on 16.04.2023.
//
import UIKit
class SceneDelegate: UIResponder, UIWindowSceneDelegate {
    var window: UIWindow?
        func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions: UIScene.ConnectionOptions) {

        guard let windowScene = (scene as? UIWindowScene) else { return }
```

```
        window = UIWindow(windowScene: windowScene)
        window?.rootViewController = TabBarController()
        window?.makeKeyAndVisible()



    }
    func sceneDidDisconnect(_ scene: UIScene) {
        // Called as the scene is being released by the system.
         // This occurs shortly after the scene enters the background, or when its session is discarded.
         // Release any resources associated with this scene that can be re-created the next time the scene connects.
          // The scene may re-connect later, as its session was not necessarily discarded (see `application:didDiscardSceneSessions` instead).
    }
    func sceneDidBecomeActive(_ scene: UIScene) {
         // Called when the scene has moved from an inactive state to an active state.
        // Use this method to restart any tasks that were paused (or not yet started) when the scene was inactive.
    }
    func sceneWillResignActive(_ scene: UIScene) {
         // Called when the scene will move from an active state to an inactive state.
        // This may occur due to temporary interruptions (ex. an incoming phone call).
    }
    func sceneWillEnterForeground(_ scene: UIScene) {
        // Called as the scene transitions from the background to the foreground.
        // Use this method to undo the changes made on entering the background.
```

```
    }
    func sceneDidEnterBackground(_ scene: UIScene) {
        // Called as the scene transitions from the foreground to the background.
        // Use this method to save data, release shared resources, and store enough
scene-specific state information
        // to restore the scene back to its current state.
    }
}
//
//  YeOzbroyenniaNSTests.swift
//  YeOzbroyenniaNSTests
//
//  Created by Oleksandr Smakhtin on 16.04.2023.
//
import XCTest
@testable import YeOzbroyenniaNS
final class YeOzbroyenniaNSTests: XCTestCase {
    var dataManager: DataManager!

    override func setUpWithError() throws {
        dataManager = DataManager.shared
    }
    override func tearDownWithError() throws {
        dataManager = nil
    }
    func testGetSubcategory() throws {
        let expectedCategory = "Стрілецька зброя"
        let subcategory = dataManager.getSubcategory(by: expectedCategory)

            XCTAssertEqual(subcategory.category, expectedCategory, "Category
```

```
should match the expected category")
                XCTAssertFalse(subcategory.subcategories.isEmpty, "Subcategories
should not be empty")
        }
        func testGetElements() throws {
            let expectedSubcategory = "Автомати"
            let element = dataManager.getElements(by: expectedSubcategory)

                    XCTAssertEqual(element.subcategory,  expectedSubcategory,
"Subcategory should match the expected subcategory")
            XCTAssertFalse(element.items.isEmpty, "Items should not be empty")
        }
        func testGetItem() throws {
            let expectedItem = "Glock 17"
            let item = dataManager.getItem(by: expectedItem)

                XCTAssertEqual(item.item, expectedItem, "Item should match the
expected item")
                 XCTAssertFalse(item.property.isEmpty, "Properties should not be
empty")
        }
        func testGetRandom() throws {
            let randomItems = dataManager.getRandom()

                 XCTAssertEqual(randomItems.count, 3, "Should return exactly 3
random items")
            for item in randomItems {
                XCTAssertFalse(item.isEmpty, "Item names should not be empty")
            }
        }
```

```
        func testGetSearchedItems() throws {
            let query = "Glock 17"
            let searchResults = dataManager.getSearchedItems(by: query)

              XCTAssertFalse(searchResults.isEmpty, "Search results should not be
empty")
            for result in searchResults {
                  XCTAssertTrue(result.lowercased().contains(query.lowercased()),
"Search results should contain the query")
            }
        }
    }
```