«СТВОРЕННЯ СЕРВІСУ ДЛЯ СОЦМЕРЕЖІ REDDIT НА ПЛАТФОРМІ IOS»

Національний університет “Києво-Могилянска академія”

Автор: Прокоф’єв Андрій

Ідея виникла під час відключень світла

- 12 годин на день робити нічого, з друзями не поспілкуватись, сумно. Хотілось дивитись пости в інтернеті, але готового рішення не було.

А які клієнти вже були?

- **Офіційний клієнт Reddit для iOS**
  Пропонує базовий функціонал (стрічка, лайки, коментарі, фільтри), проте не підтримує офлайн-режим: дані підвантажуються заново при кожному запуску.
- **Apollo**
  Має розширений інтерфейс і жести, кешує зображення, але не забезпечує автоматичної фонового оновлення стрічки — після тривалого простою вимагає ручного оновлення.
  І додатково його закрили.
- **Narwhal**
  Відрізняється мінімалізмом і високою швидкістю запуску, проте використовує фіксовані розміри комірок, що призводить до обрізання контенту, і не має механізму фонового збереження постів.
- **Виявлені загальні обмеження**
  - Відсутність повноцінного офлайн-режиму
  - Ручне оновлення стрічки, відсутність Background Fetch

Що точно має бути?

Формулювання завдань

- **Забезпечення повноцінного офлайн-режиму**
  Автоматичне фонове завантаження та збереження постів у Core Data для доступу без інтернету.
- **Динамічний адаптивний інтерфейс**
  Розрахунок висоти комірок UITableView залежно від пропорцій зображень, відео та тексту.
- **Фонове оновлення стрічки**
  Використання BGAppRefreshTaskRequest / Background Fetch для періодичного оновлення контенту без участі користувача.
- **Гнучке керування налаштуваннями та кастомайз**
  Можливість додавати/видаляти сабредіти та перемикати фільтри (hot, new, top) «на льоту».

З цього виходить технологічний стек:

- **Слайд 6. Технологічний стек**
- **Мова та платформа**
  - Swift 5
  - iOS 14+
- **UI**
  - UIKit (UITableView, Auto Layout, UIContextualAction)
  - Human Interface Guidelines
- **Мережа та дані**
  - URLSession по HTTPS
  - JSONDecoder для серіалізації/десеріалізації
  - Reddit API

•Локальне збереження
- Core Data (NSPersistentContainer, приватні контексти)
- Background Fetch / BGAppRefreshTaskRequest

•Кешування та оптимізація
- Kingfisher для кешування зображень
- DispatchGroup для паралельних завантажень
- URLCache

•Безпека
- App Transport Security / SSL-сертифікати
- NSFileProtectionCompleteUntilFirstUserAuthentication
- App Sandbox

•Налаштування та UX
- UserDefaults (SettingsManager)
- Dynamic Type, локалізація (англ./укр.)
- Світла/темна тема (overrideUserInterfaceStyle)

Перша дія – Reddit API

- **MemeApiHandler**
  - Окремий модуль для форматованих API-запитів
  - Використання URLSession по HTTPS із ATS-перевіркою SSL
- **Формування запиту**
  - https://www.reddit.com/r/[subreddit]/[filter].json?limit=[limit]
  - Параметри сабредіту, фільтра та ліміту беруться з налаштувань користувача
- **Обробка відповіді**
  - JSONDecoder → Swift-структури із полями id, title, mediaURL, тип, час створення
  - Приватний NSManagedObjectContext (.privateQueueConcurrencyType) для запису в Core Data
  - Політика злиття mergeByPropertyObjectTrump
- **Фонове оновлення**
  - Background Fetch / BGAppRefreshTaskRequest для періодичного завантаження нових постів
  - Завершальний completionHandler інформує iOS про результат (.newData / .noData)

Як і де це зберігати?

- **Модель даних**
  - Сутність PostEntity з атрибутами: id, title, urlString, mediaType, mediaData, width, height, isDownloaded, isFavorite
  - Відсутність зв’язків між сутностями — фільтрація за сабредітом та типом у запитах
- **CoreDataManager**
  - NSPersistentContainer(name: "MemeShuffler") з політикою mergeByPropertyObjectTrump
  - viewContext для UI-операцій, performBackgroundTask для фонових записів
- **Background Fetch**
  - application.setMinimumBackgroundFetchInterval(…) + application(_:performFetchWithCompletionHandler:) в AppDelegate
  - CoreDataManager.preloadPosts(count:completion:) з DispatchGroup для паралельних завантажень та ctx.save() у приватному контексті
- **Оптимізація сховища**
  - Налаштування localSaveLimit для керування об’ємом локального кешу
  - Підготовка до використання NSBatchDeleteRequest для видалення застарілих постів

UI/UX: Рішення та обґрунтування

- **Динамічний автолейаут**
  - UITableView.automaticDimension + констрейнт співвідношення сторін для медіа

  Чому? Забезпечує відображення зображень, відео та тексту без обрізань і спотворень, адаптується під будь-який контент.

- **Жести для швидких дій**
  - Swipe-дії: «Share» і «Favorite» через UIContextualAction

  Чому? Інтуїтивно зрозуміло, не потребує додаткових кнопок, економить місце на екрані.

- **Надшар UI для перемикання екранів**
  - Двохекранна схема (фід + налаштування) із загальною шапкою

  Чому? Мінімізує кількість переходів, користувач одразу бачить і стрічку, і доступ до налаштувань.

- **Модульні налаштування**
  - SettingsViewController із вертикальним стэком UIView-модулів та анімаціями

  Чому? Легко додавати/приховувати секції, плавна анімація покращує враження.

- **Підтримка світлої/темної тем і локалізації**
  - overrideUserInterfaceStyle, AppleLanguages

  Чому? Враховує особисті уподобання та забезпечує зручність у будь-яких умовах освітлення та мови.

- **Dynamic Type**
  - UIFont.preferredFont(forTextStyle:)

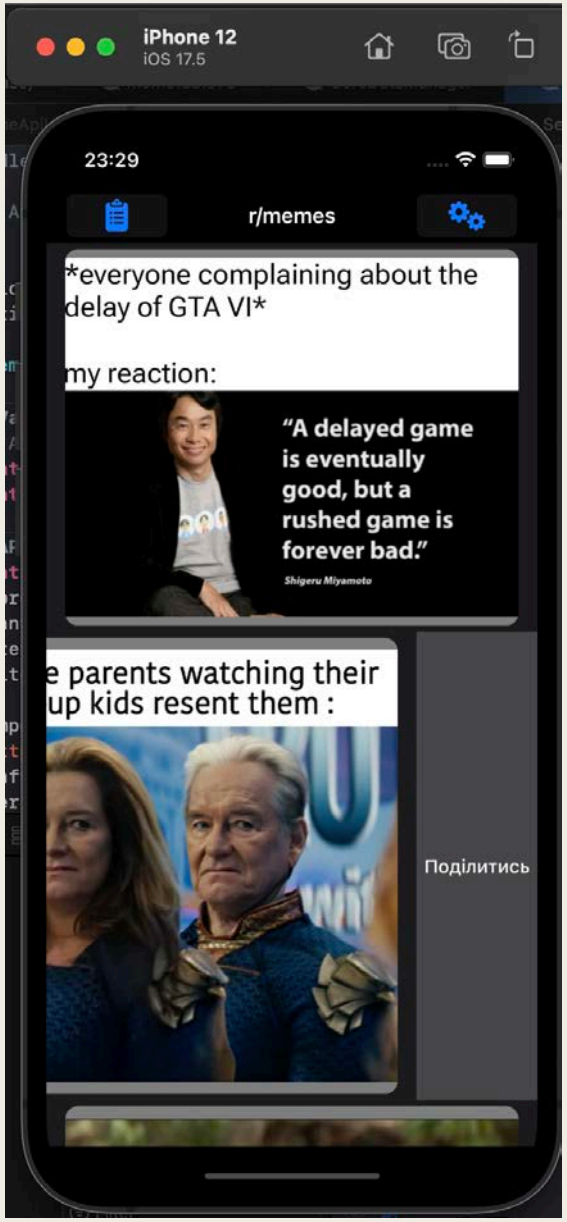
  Чому? Автоматичне масштабування тексту під налаштування користувача, підвищує читабельність на малих екранах.

Умовний розподіл проекту

- **MemeApiHandler**
  - Мережевий шар: формування HTTPS-запитів, повторні спроби, декодування JSON
- **CoreDataManager**
  - CRUD-операції з PostEntity, приватні контексти, фонова синхронізація через preloadPosts
- **MemeViewController**
  - Жива стрічка мемів: UITableView з динамічною висотою комірок, відображення тексту, зображень, відео, GIF
- **Background Fetch / BGTaskScheduler**
  - application(_:performFetchWithCompletionHandler:), BGAppRefreshTaskRequest, DispatchGroup
- **UIContextualAction**
  - Опції «Share» (UIActivityViewController) і «Favorite» (CoreDataManager.favorite/unfavorite) по свайпу
- **SettingsViewController**
  - Налаштування сабредітів, фільтрів (hot/new/top), теми, мови, ліміту кешу — модульний UI з анімаціями

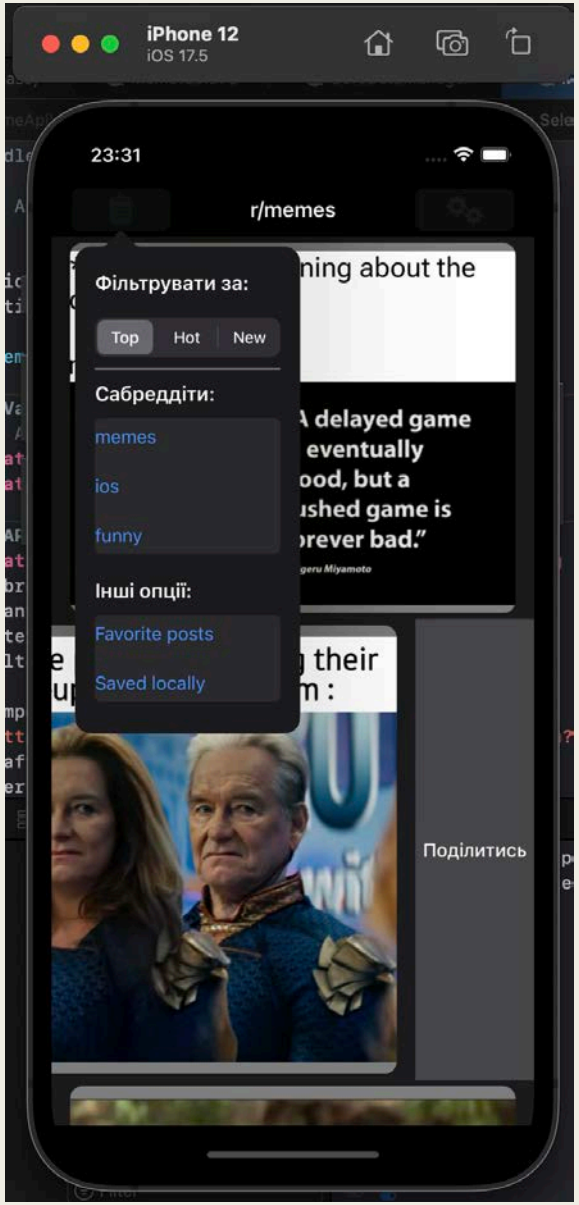
1. MemeViewController

- Відповідає за відображення живої стрічки постів у UITableView з динамічною висотою комірок.
- Обробляє події свайпів для опцій «Share» та «Favorite» через UIContextualAction.
- Ініціює виклики MemeApiHandler для завантаження нових мемів та передає їх у CoreDataManager для збереження.
- Слідкує за результатами NSFetchedResultsController і оновлює таблицю без повного перезавантаження екрану.



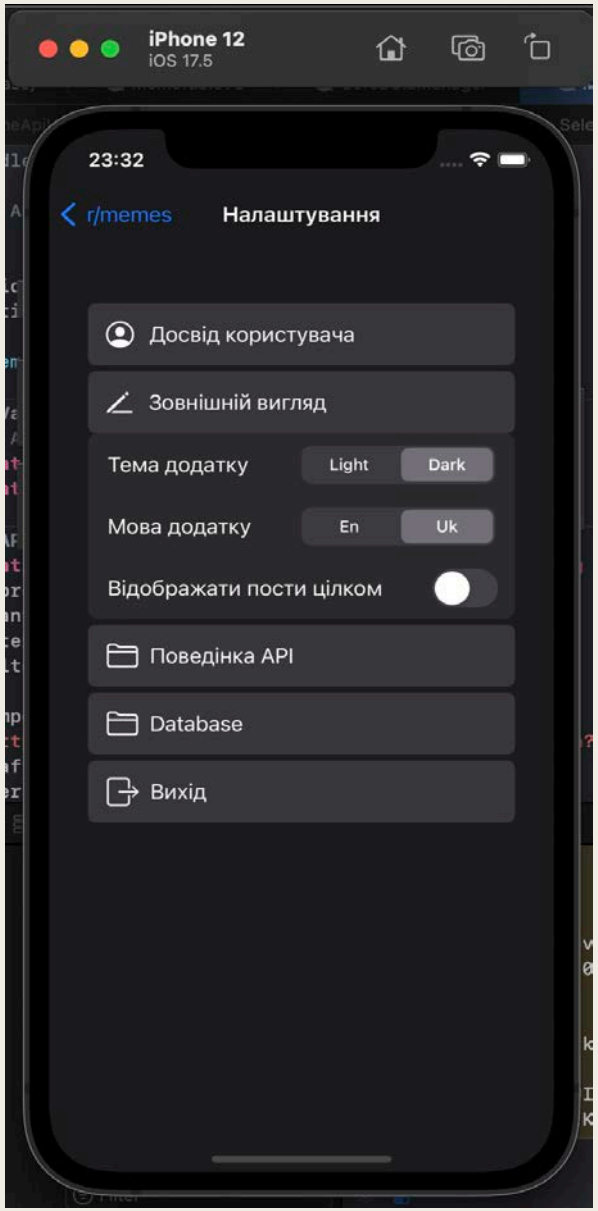
1. SelectorViewController

Призначення
• Забезпечує зручний вибір сабредітів та фільтрів без необхідності переходу на окремий екран.
Інтерфейс
• Розташований у вигляді горизонтального меню або сегментованого контролу над стрічкою постів.
Функції
• Динамічне підвантаження списку користувацьких сабредітів із Core Data.
• Швидке перемикання між фільтрами “hot”, “new”, “top” з миттєвим оновленням стрічки.
• Плавні анімації та візуальний feedback при торканні до елементів.
• Викликає методи MemeApiHandler з новими параметрами сабредіту/фільтра.
• Не блокує основний UI-потік завдяки використанню асинхронних запитів.
• Інтегрується з ContainerViewController для відображення/приховування надшару без перезавантаження всього екрану.



1. SettingsViewController

- **Інтерфейс:**
  Вертикальний UIStackView із секціями для кожного типу налаштувань:
  - Сабредіти (список + кнопки додати/видалити)
  - Фільтри (Hot, New, Top)
  - Тема (світла/темна) та мова (англ./укр.)
  - Ліміт кешу та кнопка очищення
- **Взаємодія:**
  - UISwitch і слайдер для швидкого керування параметрами
  - UIView.animate() для плавного розгортання і згортання секцій
- **Логіка:**
  - Збереження налаштувань через SettingsManager (UserDefaults)
  - Валідація введення сабредітів (без порожніх рядків, пробілів)
  - Надсилання повідомлень через NotificationCenter для миттєвого оновлення Live-Feed
- **Переваги:**
  - Вся персоналізація на одному екрані—без зайвих переходів
  - Модульність: легко додавати нові секції
  - Швидке застосування змін: UX-дружній підхід для користувача



Головна фішка

- **Реєстрація завдання**
  - У AppDelegate.didFinishLaunchingWithOptions викликає scheduleAppRefresh()
- **Обробка оновлення**
  - Метод handleAppRefresh(task:) повторно планує завдання та викликає MemeApiManager.loadMemesCompilation { memes in ... }
  - Після завантаження даних — task.setTaskCompleted(success:)
- **Фонова синхронізація у CoreDataManager**
  - Виклик persistentContainer.performBackgroundTask { ctx in ... }
  - Паралельний DispatchGroup для завантаження зображень через Kingfisher
  - Збереження контексту із mergeByPropertyObjectTrump

Що не вдавалось і чого навчився:

- **Certificate Pinning**
  – При спробі жорсткого pinning сертифіката з’являлися помилки довіри через зміни в проміжному CA.
  – Вивчив налаштування URLSessionDelegate для обробки urlSession(_:didReceive:completionHandler:) та реалізував перевірку хешів сертифікатів із використанням SHA-256.
  – Розібрався, як підтримувати декілька вендорських сертифікатів і уникати проблем при оновленні серверних ключів.

- **Декодування JSON**
  – Стандартний JSONDecoder не справлявся з обхідними випадками: змішані типи (String/Int), вкладені масиви, відсутні поля.
  – Реалізував кастомні init(from:) для Codable, використовуючи decodeIfPresent і власні контейнерні методи.
  – Освоїв створення JSONDecodingStrategy для дат і KeyDecodingStrategy.convertFromSnakeCase, щоб узгодити імена властивостей із API Reddit.

- **Динамічна висота комірок**
  – Початково UITableView.automaticDimension конфліктував із констрейнтами співвідношення сторін, через що ячейки “стрибали” при скролі.
  – Навчився правильно встановлювати estimatedRowHeight перед викликом reloadData(), а також виконувати tableView.layoutIfNeeded() у viewDidLayoutSubviews().
  – Зрозумів важливість мінімізації числа викликів layoutIfNeeded() задля збереження плавності UI.

- **Core Data Конфлікти та фонові збереження**
  – Merge-конфлікти виникали при одночасному записі з головного і приватного контекстів.
  – Впровадив NSMergePolicy.mergeByPropertyObjectTrump і переконався в коректності роботи через додаткові тести з фонових тасків.
  – Освоїв оптимізацію performBackgroundTask для пакетного збереження й уникнення блокувань UI.

Background Fetch та BGTaskScheduler
– BGAppRefreshTaskRequest іноді взагалі не спрацьовував: через невірно зареєстрований ідентифікатор чи відсутність виклику scheduleAppRefresh() в applicationDidEnterBackground.
– Навчився дебажити фонові таски, використовуючи Xcode Scheme → Options → Background Fetch, і встановив адекватний earliestBeginDate.
– Зрозумів, як передавати expirationHandler та своєчасно викликати task.setTaskCompleted(success:), щоб iOS не позначав таск як провалений.

UI/UX та адаптивність
– Dynamic Type не завжди застосовувався до кастомних UILabel: потрібні були label.adjustsFontForContentSizeCategory = true.
– Дізнався, як працюють Trait Variations у Interface Builder для різних класів пристроїв і орієнтації екрану.
– Допрацював механізм перемикання світлої/темної теми через overrideUserInterfaceStyle, враховуючи, що дочірні UIViewController наслідують стиль від контейнера.

Оптимізація кешування
– Kingfisher кеш іноді не очищався в пам’яті при didReceiveMemoryWarning.
– Вивчив різницю між clearMemoryCache() та clearDiskCache() і запустив ImageCache.default.cleanExpiredDiskCache() для видалення застарілих файлів.
– Реалізував програмне скидання кешу при тестуванні, щоб уникнути “забруднення” старими зображеннями.

Менеджмент налаштувань
– UserDefaults не зберігав масив сабредітів при оновленні значень без синхронізації.
– Додав UserDefaults.standard.synchronize() після важливих змін та перейшов на використання AppGroup для тестування спільного доступу.

Дякую за увагу!