Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Курсова робота

освітній ступінь – бакалавр

на тему:

«Створення сервісу для соцмережі Reddit на платформі IOS»

Виконав: студент 3-го року навчання,

Спеціальності

121 «Інженерія Програмного Забезпечення»

Студента Прокоф’єва Андрія

Керівник Борозений С.О.

Старший викладач

Київ – 2025

Тема: Створення сервісу для соцмережі Reddit на платформі IOS

Календарний план виконання роботи:

№	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1	Отримання індивідуального завдання та затвердження теми «MemeShuffler»	09.10.2024	
2	Аналіз вимог і постановка задач проекту	16.10.2024	
3	Огляд аналогічних сервісів та наукової літератури з алгоритмів мемів	30.10.2024	
4	Проектування архітектури системи та моделі даних	15.11.2024	
5	Реалізація бекенд-модулів (API, логіка перемішування мемів)	30.11.2024	
6	Розробка клієнтського інтерфейсу (UI/UX)	15.01.2025	
7	Інтеграція фронтенду з бекендом	28.02.2025	
8	Оптимізація продуктивності, безпеки	16.03.2025	
9	Остаточне доопрацювання, оформлення фінальної версії додатку	23.04.2025	
10	Підготовка текстової частини курсової роботи та оформлення схем і рисунків	04.05.2025	
11	Захист курсової роботи		

ЗМІСТ

Анотація 4

Вступ 5

РОЗДІЛ 1: Аналіз предметної області, огляд існуючих клієнтів для Reddit, формулювання завдань. 7

1.1 Постановка мети та завдань дослідження 7

1.2 Огляд існуючих клієнтських додатків для Reddit 8

1.3 Опис об'єкту та предметів дослідження 9

РОЗДІЛ 2: Проектування інтерфейсу та архітектури додатку 11

2.1 Функціональні та нефункціональні вимоги 11

2.2 Архітектурна модель клієнт-додатка на UIKit 12

2.3 Проектування моделі даних у CoreData та інших параметрів 15

2.4 Розробка UI: макети, autolayout, адаптивність 17

2.5 Організація навігації, вибір сабреддітів та фільтрів 19

2.6 Налаштування локалізації, світлого/темного оформлення 21

РОЗДІЛ 3: Реалізація функціональних модулів додатку 22

3.1 Інтеграція з Reddit API 22

3.2 Відображення контенту UITableView із динамічною висотою ячеєк. 23

3.3 BackgroundFetch та збереження даних у CoreData 24

3.4 Реалізація опції "Share" для поширення постів. 25

3.5 Реалізація опції "Favorite" для збереження постів. 26

РОЗДІЛ 4: Безпека та оптимізація додатку 28

4.1 Оптимізація продуктивності та кешування 28

4.2 Забезпечення безпеки HTTP-заголовків і захист даних 30

Висновки 32

Анотація

У цій курсовій роботі представлено розробку iOS-додатка **MemeShuffler** — сумісного клієнт-інтерфейсу з платформою Reddit, що дозволяє переглядати та керувати медіа-контентом сабредітів безпосередньо на iPhone. Додаток являє собою інструмент зі вдалим UI/UX дизайном спрямованим на зручність використання та кастомізацію під користувача. У додатку реалізовані функції як live-перегляду медіа-контенту прямо з сайту, використовуючи API-запити на сервер, так і збереження цього контенту на пристрій за допомогою локальної БД. Користувач може налаштовувати цей медіа-контент під себе, перемикаючись між сабреддітами(джерелами) та фільтрами, також ділитись контентом з іншими. Додатково, були реалізовані такі параметри налаштувань як:

- Світла/Темна теми інтерфейсу на вибір
- Повна локалізація двома мовами на вибір (анлглійська, українська)
- Перелік сабреддітів(джерел), з яких братиметься медіа-контент
- Повний контроль над локально збереженими даними з можливістю очищення та закачки в один клік
- Презентування зацензурованого/18+ контенту

Вступ

У ході вторгнення російської Федерації енергетична інфраструктура України зазнала численних ударів, спрямованих на диспетчерські центри та підстанції. Як повідомляє EAWORLDVIEW: «Unfortunately Russia continues to carry out missile strikes on Ukraine's civilian and critical infrastructure. Almost half of our energy system is disabled.» [1] Станом на 19 листопада 2022 року майже 50 % енергосистеми було виведено з ладу, залишивши без світла близько 10 мільйонів людей. Протягом 2022-2023 років інтенсивні обстріли й окупація ключових потужностей призвели до втрати приблизно 21 ГВт генерації й днів, й де світло було відсутнє по 12 годин по версії The Guardian [2], у 2023-2024 роках відновилися багатогодинні планові відключення, коли споживачі щодня сиділи без світла, аналогічно минулому року. І протягом того ж 2024 року доля виведених потужностей досягла 60-80 відсоків, що навіть змусило імпортувати енергію з інших стран. І це враховуючи підтримку країн-союзників, що активно поставляли чисельні партії генераторів для бізнесів та критично важливої інфраструктури.

Ця ситуація стала тяжким іспитом для багатьох сфер господарства, де зупинялись виробничі процеси, медичні установи та навчальні заклади. Для людей це значило відсутність можливості працювати за комп'ютером, мати зв'язок з іншими людьми та неможливість нормального проведення часу/відпочинку. Як результат - стрес та поглиблення відчуття ізоляції.

У таких умовах виникла нагальна потреба в інструменті, що дозволяв би зберігати розважальний контент для офлайн перегляду у ті рідкі моменти коли світло є. Мій проект присвячений саме цьому, тобто додатку-клієнта до платформи Reddit, що є соціальною мережею та, значною мірою, є джерелом якісного розважального контенту. Для реалізації була обрана платформа iOS та використано відповідні інструменти розробки, а саме UIKit та чисельні бібліотеки/фреймворки для тієї ж платформи.

Сама робота складається з чотирьох розділів, що містять підрозділи для кращої структуризації матеріалу.

- **Розділ 1**:
  Аналіз предметної області, огляд існуючих клієнтів для Reddit, формулювання завдань.

Цей розділ описує сучасний стан мобільних клієнтів Reddit, визначає ключові вимоги до функціоналу та UX-дизайну, а також формулює основні задачі реалізації iOS-додатка MemeShuffler .

- **Розділ 2**:
  Проектування архітектури застосунку, моделі даних у Core Data [9] і макетів інтерфейсу.

Цей розділ містить обґрунтування вибору UIKit для розробки клієнтської частини, опис моделі даних у Core Data із врахуванням механізму Background Fetch та створення макетів інтерфейсу з адаптивним автолейаутом для підтримки різних форматів контенту.

- **Розділ 3**:
  Реалізація функціональних модулів додатку.

Цей розділ детально розглядає інтеграцію з Reddit API для отримання медіа-контенту, реалізацію динамічної висоти ячеєк контенту, механізм фонового збереження в Core Data, функцію «Share» для обміну контентом через стандартні сервіси iOS, екрани налаштувань та спеціальний селектор для налаштувань перегляду в режимі реального часу.

- **Розділ 4**:
  Тестування, оптимізація та безпека.

Цей розділ присвячений методам перевірки коректності роботи ключових модулів, оптимізації продуктивності через кешування і мінімізацію мережевих запитів, а також впровадженню базових заходів безпеки

РОЗДІЛ 1: Аналіз предметної області, огляд існуючих клієнтів для Reddit, формулювання завдань.

1.1Постановка мети та завдань дослідження

Метою даної роботи є, власне, розробка клієнт-інтерфейсу MemeShuffler, який забезпечить користувачеві зручний перегляд, фільтрацію та офлайн-доступ до медіа-контенту з платформи Reddit, а також гнучке налаштування сабредітів, фільтрів, локалізації та тем оформлення.

Для досягнення поставленої мети передбачається виконати такі завдання:

1. Проаналізувати наявні мобільні клієнти Reddit (офіційний додаток, Apollo, Narwhal тощо) та виокремити їхні сильні й слабкі сторони.
2. Визначити функціональні та нефункціональні вимоги до **додатку,** зокрема щодо продуктивності, стійкості роботи в умовах нестабільного зв’язку й обсягу локального зберігання.
3. Спроектувати архітектуру клієнт-додатка з урахуванням моделі даних у **Core Data** і механізму **Background Fetch** для фонової синхронізації.
4. Розробити інтерфейс користувача на UIKit із адаптивним автолейаутом для підтримки різних співвідношень сторін медіа й динамічним розрахунком висоти **UITableView**-ячеєк.
5. Реалізувати інтеграцію з Reddit REST API для отримання та обробки тексту, зображень, відео й GIF.
6. Забезпечити офлайн-доступ до контенту через фонове завантаження нових постів і їхнє збереження в **Core Data**.
7. Створити систему налаштувань: керування списком сабредітів, перемикання фільтрів **top/hot/new**, локалізація (англійська/українська) та теми оформлення (світла/темна).
8. Інтегрувати функцію «Share» для швидкого поширення посилань на пости через стандартні сервіси iOS.
9. Оптимізувати продуктивність клієнта та впровадити базові заходи безпеки HTTP-запитів і захисту користувацьких даних.

1.2 Огляд існуючих клієнтських додатків для Reddit

Для цієї платформи існує багато клієнтів, але розповсюдженими є лише декілька з них, а саме: **Офіційний, Apollo, Naewhal.** Кожен з них в чому відрізняється від іншого, але в усіх них суть дуже схожа.

Офіційний клієнт Reddit для iOS пропонує користувачам базовий набір функцій: перегляд стрічки з різних сабредітів, лайки, коментарі та обмежені налаштування фільтрів («hot», «new», «top»). Проте він абсолютно не підтримує офлайн-режим: за відсутності інтернету стрічка не відображається, а всі дані підвантажуються «з нуля» при кожному запуску. Крім того, фіксована висота ячеєк зображень та відео призводить до обрізання контенту, а єдиний темний або світлий режим налаштовується лише глобально.

Apollo[3], вважався одним із найпросунутіших сторонніх клієнтів, що був зосереджений насамперед на гнучкості інтерфейсу та розширених жестах управління. Цей клієнт був закритий за вимогою Reddit у 2023 році. Він пропонував розширені можливості сортування, кешування кешу зображень та підтримку деяких офлайн-функцій, але не мав інтегрованого фонового завантаження стрічки: після тривалого простою додаток вимагав ручного оновлення, що не гарантувало своєчасного збереження нових постів. Центральним сховищем був власний кеш, який очищувався системою iOS, а механізми локалізації та тем оформлення часто обмежувались англійською мовою та стандартними темами, без варіантів кастомізації.

Narwhal[4]відрізняється від попередніх своєю мінімалістичністю та швидкістю: додаток запускається миттєво і дозволяє легко скролити контент, але він жодним чином не передбачає фонової синхронізації або збереження постів у локальну базу. Ячеїки відображаються у фіксованому форматі, тому різні співвідношення сторін призводять до спотворення або

обрізання зображень і GIF. Керування підписками та фільтрами відбувається лише через довгий тап і меню налаштувань, що не найзручніше у термінових ситуаціях, наприклад, під час проблем із мережею.

Тож робимо висновок, що на відміну від усіх перелічених клієнтів, **MemeShuffler** поєднує переваги адаптивного інтерфейсу та фонового завантаження: автоматично підлаштовує висоту комірок під будь-які формати медіа, періодично оновлює стрічку у фоновому режимі за допомогою Background Fetch та надійно зберігає всі пости в Core Data для офлайн-доступу. Додаток реалізує гнучку систему фільтрів і динамічне управління списком сабредітів, підтримує двомовну локалізацію та зміну світлої/темної теми, що забезпечує безперебійну та комфортну роботу навіть за нестабільного інтернет-з'єднання або вимкненого електропостачання.

1.3 Опис об'єкту та предметів дослідження

Об'єктом достідження виступає сам мобільний додаток «MemeShuffler» для операційної системи iOS, розроблений на фреймворку UIKit. Додаток забезпечує автоматичний потік медіа-контенту з обраних сабредітів платформи Reddit, динамічне відображення зображень, відео, GIF та тексту у вигляді адаптивних комірок **UITableView**, а також реалізує механізм фонової синхронізації та збереження даних у локальній базі Core Data за допомогою **Background Fetch** для забезпечення офлайн-доступу до контенту.

Враховуючи зазначені функціональні можливості та архітектурні рішення, обрані для додатка, у рамках дослідження необхідно детально вивчити ті технологічні й дизайнерські компоненти, які безпосередньо забезпечують

його роботу в умовах нестабільного з’язку та потребу офлайн-доступу. Саме тому основними **предметами дослідження** обрано:

1. Інтеграцію з **Reddit REST API** для надійного завантаження тексту, зображень, відео та GIF, що є джерелом основного контенту.
2. Механізм **Background Fetch** та збереження даних у **Core Data**, який гарантує своєчасну синхронізацію й доступ до постів без інтернет-з’єднання.
3. Стратегії кешування в **Core Data** з урахуванням обмежень пам’яті та продуктивності пристрою, що дозволяють зменшити затримки та навантаження на мережу.
4. Динамічний автолейаут і розрахунок висоти комірок **UITableView** під різні формати медіа.
5. Налаштування локалізації (англійська/українська), тем оформлення (світла/темна) та фільтрів контенту (**top**, **hot**, **new**) - для адаптації інтерфейсу під індивідуальні потреби користувачів.

РОЗДІЛ 2: Проектування інтерфейсу та архітектури додатку

2.1 Функціональні та нефункціональні вимоги

Після аналізу стороннії клієнтів у пункті 1.2, зрозуміло на чому саме робити акцент з точки зору функціональних/нефункціональних вимог:

Функціональні вимоги, як було попередньо зазначено декілька разів, складаються з:

- Перегляд стрічки постів із вибраних сабредітів Reddit у **UITableView**, що підтримує відображення тексту, зображень, відео й GIF із автоматичним підлаштуванням висоти ячеєк.
- Екран для live налаштування з фільтрами **top**, **hot** і **new**, а селектором сабреддітів як джерела даних та перемикач режимів “Favorite posts”/”Saved locally” відповідно для 2х режимів офлайн перегляду
- Функція “**Share**” з використанням **swipeGesture** для швидкого обміну посиланнями на пости через стандартні сервіси iOS.
- Функція “**Favorite**” за використанням **longTapGesture** для зберігання конкретних постів з черги. Такі пости теж зберігатимуться у БД для офлайн-перегляду
- Фонова синхронізація нових матеріалів за допомогою **Background Fetch** та їх збереження в локальній базі **Core Data** для офлайн-доступу.
- Інтерфейс налаштувань із можливістю перемикання локалізації (eng/ukr), тем оформлення (light/dark), фільтрації контенту, що буде презентований на головному екрані стрічки, модулем контролю локально збережених даних

Щодо нефункціональних вимог, вийшов такий перелік:

- **Стійкість роботи** в умовах нестабільного чи відсутнього інтернет-з’єднання: додаток має коректно відображати раніше завантажений контент і не аварійно завершуватися.
- **Продуктивність**: час завантаження й оновлення стрічки не повинен перевищувати 2–3 секунди на сучасних моделях iPhone; динамічний розрахунок висоти ячеєк має відбуватися без помітних затримок.
- **Обсяг локального зберігання**: максимальний розмір бази Core Data чітко корегується користувачем. Для цього, він може обрати ліміт завантаження у відповідному модулі у налаштуваннях, де може встановити значення від 0 (не завантажувати взагалі) до 500 постів. Також, користувач матиме змогу очистити всі локально збережені дані (улюблені та завантажені)
- **Адаптивність інтерфейсу**: підтримка всіх стандартних розмірів екрану iPhone (від SE до Max), з урахуванням різних орієнтацій, що вимагатиме Autolayout у storyboard-і
- **Безпека**: застосування HTTPS для всіх API-запитів.

2.2 Архітектурна модель клієнт-додатка на UIKit

Для того щоб визначити найвдаліший підхід побудови архітектури, треба розглянути основні моделі архітектури на iOS. Передивившись офіційну документацію Apple[] та переглянув інформацію в інтернеті я прийшов до висновку, що основних підходів до архітектури всього 3: **MVC[5], MVP[6]. MVVM[7].** Їх ми зараз розберемо детально.

1. **MVC** архітектура будується за рахунок поєднання **Model-View-Controller**, де:
   Model відповідає за дані та бізнес-логіку: десеріалізація JSON, управління Core Data, правила фільтрації та кешування.

View складається з UI-елементів UIKit (UIView, UITableView, TableViewCell), що лише відображають дані та передають події користувача контролеру.
Controller (в нашому випадку UIViewController) зв'язує Model і View, обробляє взаємодію користувача, виконує мережеві запити через, наприклад, модуль ApiService, та оновлює View.
Серед переваг цього підходу можна зазначити простоту інтеграції з UIKit, навіть Apple рекомендує цей підхід[] у офіційній документації та власних гайдах.
Щодо недоліків, єдиним є властивість Controller-а розрастатись паралельно розробці проекту, що ускладнює орієнтацію в коді.

2. Наступним підходом є **MVP**, що є акронімом для
Model-View-Presenter
Model як і в MVC, містить дані й бізнес-логіку.
View - мінімальний UI-шар, який делегує більшість подій Presenter'у й лише відображає отримані від нього дані.
Presenter отримує події від View, взаємодіє з Model, форматує дані й передає назад у View.
Серед переваг цього підходу: чітке розділення відповідальності, адже Presenter легко юніт-тестувати, бо він не залежить від UIKit, та менші Controller/View-класи, оскільки логіка винесена в Presenter.
Мінуси: додатковий шар Presenter призводить до більшого обсягу "підйомного" коду та при складному UI Presenter-класи можуть стати дуже великими й важкими для підтримки. (Тобто вже на цьому етапі цей підхід гірше за перший)

3. Третім підходом є MVVM, що значить Model-View-ViewModel
Частково схожий на перший підхід, але з нюансами.

Model - як у MVC/MVP.

View - UIKit-компоненти, зв’язані з ViewModel через різні спостерігачі (KVO, Closure, RxSwift/Combine). Це є надлишковим в нашому випадку через обмежену асинхронність, яку досить легко обробляти звичайнимим методами.

ViewModel інкапсулює всю UI-логіку, перетворює Model у моделі, готові для відображення, і надає їх View, сам не залежить від UIKit. Перевагами є: добре поєднується з реактивними підходами (RxSwift, Combine), що дозволяє створювати декларативні View. Також, ViewModel не містить посилань на View, полегшуючи тестування та повторне використання.

Але:

1. Введення реактивних фреймворків та зв’язку View–ViewModel підвищує складність проєкту.
2. Для UIKit-проєктів “на чистому” Swift без RxSwift MVVM може виявитися надмірним.

Проаналізувавши ці підходи, я прийшов до висновку, що найвдалішим для мого додатку буде метод MVC, з такою аргументацією:

- **Сумісність з UIKit.** Офіційна документація Apple орієнтована на MVC, тож цей патерн найпростіше пристосувати до UIViewController, UITableView та інших компонентів.
- **Помірний обсяг бізнес-логіки.** Основні операції (мережа, кешування, автолейаут) добре розподіляються між Model і Controller без потреби вводити додаткові абстракції.

- **Простота підтримки.** Менша кількість шарів дозволяє швидше вносити зміни під час розробки курсової роботи та легше пояснити структуру коду керівнику.
- **Швидкість.** MVC не вимагає робити додаткові запити чи прописувати надмірну логіку для функціонування додатку, тому користування буде в задоволення.

2.3 Проектування моделі даних у CoreData та інших параметрів

Для того щоб зрозуміти, як зберігати окремий пост у БД, я переглянув офіційну документацію Reddit API[], офіційну документацію CoreData й додатково тестував запити використовуючи сервіс Postman [10]. Мною були протестовані звичайні запити, запити з параметрами, запити на різні сабреддіти у різному вигляді. Після цього, провівши аналіз того, який JSON повертається для мене, було прийнято рішення зберігати таку структуру з такими параметрами:

{ id: String, title: String, postHint: String,
urlString: String, mediatype: String,
mediaData: RawBinary, width: Double, height: Double,
isDownloaded: Boolean, isFavorite:Boolean }

З такою аргументацією:
Id є унікальним ідентифікатором поста в системі реддіт та ключем у БД
title є заголовком поста
posthint є підказкою формату контенту (“image”, “gif”, “video”), для live
urlString – URL основного медіа посту, для завантаження у інтерфейс.
mediaType – кастомна підказка типу медія, для БД

mediaData – закешовані сирі дані медія для офлайн відображення
width, height – розміри основного медіа, вираховуються динамічно.
isDownloaded/isFavorite – кастомні флаги для помітки чи був пост збережений чи поміщений у «улюблені»
У моделі немає зв’язків (“relationships”) між сутностями, оскільки вся фільтрація за сабредітом і типом здійснюється на рівні запитів із параметрами. Всі операції з запису й читання виконуються через окремий приватний NSManagedObjectContext з mergePolicy = .mergeByObjectProperty, що дозволяє безпечно застосовувати зміни, завантажені у фоновому режимі через Background Fetch, без блокування основного потоку UI.

Додатково, зберігаються налаштування самого додатку за допомогою UserDefaults за допомогою класу **SettingsManager.** Він інкапсулює ключі й обробку значень таких властивостей:

- showCensoredPosts, showSpoilerPosts, allowVideoAutoplay, showFullPostInfo – налаштування UX, впливають на перегляд контенту
- interfaceTheme, interfaceLanguage – налаштування UI, впливають на зовнішній вигляд додатку
- defaultSubreddit, defaultLoadingQuantity, localSaveLimit, savedSubreddits – API параметри, потрібні для організації запитів на сервер Reddit та для кешування даних.

Таким чином, поєднання Core Data для збереження об’єктів постів і SettingsManager для параметрів користувача забезпечує максимальну надійність, швидкий доступ до даних і гнучкість налаштувань при мінімальних затратах пам’яті та ресурсів пристрою.

2.4 Розробка UI: макети, autolayout, адаптивність

Ще на етапі проектування було зрозуміло, що матимуть місце 3 контроллери:

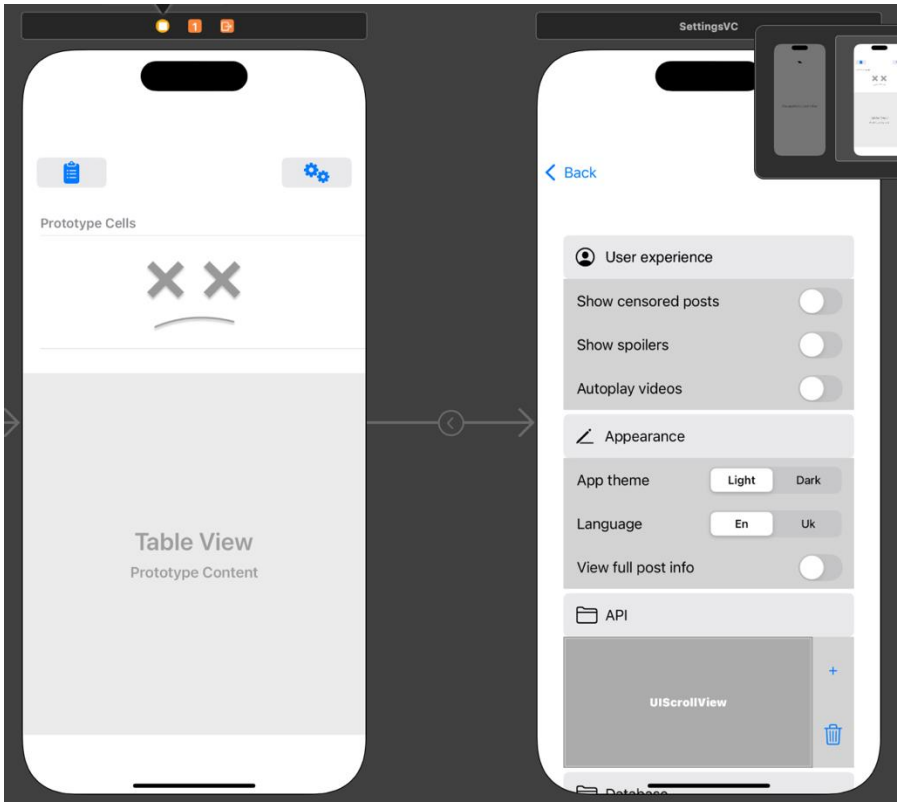
Контроллер з live-feed, який відповідає за відтворення постів.

Контроллер-селектор з фільтрами та вибіром сабреддітів/збережених постів

Контроллер з налаштуваннями, де відбуватиметься вся кастомізація користувачем.

Але, після перших прототипів інтерфейсу я дійшов до розуміння, що окремий екран для вибору фільтрів швидко починає дратувати адже вимагає забагато зайвих маніпуляцій та переходів. Тому, було прийнято рішення спростити задачу та зробити 2 екрани та 1 View що є «надслойкою»

над, в моєму випадку, фідом.



Тому, фінальна версія містить лише 2 екрани (фід зліва, налаштування справа)

Фундоментально ми маємо MemeViewController, який містить елемент tableView, де кожна ячейка – окремий пост. Ячейки самі є елементом MemeTableViewCell, вже в якому відбувається відображення медіа.
Для кожної клітинки використано динамічні констрейнти:
Зображення або відео приєднано до верхньої, лівої і правої границь контейнера з констрейнтом співвідношення сторін, який оновлюється на основі полів width, height з CoreData

Далі, ми маємо контроллер налаштувань SettingsViewController, що складається з умовних модулей на основі UIView, що скріплені елементом VerticalStack, що динамічно й з мінімальними витратами ресурсів підлаштовує позиції контенту відносто себе ж незалежно від екрану.
Також, на цьому екрані наявні кастомні анімації за допомогою UIView.animate, що дозволяє охайно й швидко показувати лише один модуль за раз.

Окрім цього, загальну адаптивність забезпечено через:

- **Trait Variations** у Interface Builder для підтримки різних розмірів екрану (Compact/Regular Width & Height). Це потрібно для визначення чи працюватиме якийсь constraint у певній орієнтації, що є максимально ефективним.
- **Dynamic Type** і UIForn.preferredFont(forTextStyle:) для автоматичного масштабування тексту згідно з налаштуваннями користувача, що є корисним на малих екранах
- Підтримку світлої та темної теми через overrideUserInterfaceStyle у кожному контроллері, що дозволяє миттєво перемикатися між режимами.

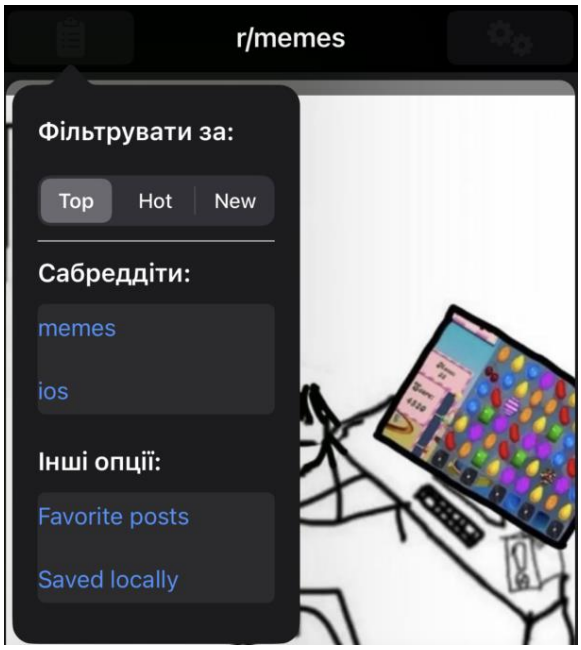
Таке поєднання схематичного проектування, Auto Layout та адаптивних можливостей UIKit гарантує збереження цілісності UI на всіх моделях iPhone та при будь-яких налаштуваннях користувача.

2.5 Організація навігації, вибір сабреддітів та фільтрів

Навігація в **MemeShuffler** побудована на базі UINavigationController, кореневим контролером якого є MemeViewController. Завдяки цьому, ми маємо навігаційну панель, де містяться 2 кнопки:

- Права кнопка (SettingsButton) відкриває SettingsViewController, який дає доступ до вибору мови, теми оформлення, ліміту кешу та інших параметрів. Після повернення до MemeViewController всі налаштування зберігаються автоматично.
- Ліва кнопка модально відкриває SelectorViewController, що є міні-вью з фільтрами та налаштуванням feed-у

Виглядає це наступним чином:



Зверху панелі розташован UISegmentedControl з фільтрами, які ми можемо перемикати. У центрі цієї панелі розміщено ScrollView з усіма збереженими сабреддітами, контент з яких ми можемо переглядати. Перехід між сабредітом і фільтром здійснюється миттєво, оскільки вибрані значення defaultSubreddit, filterType зберігаються в UserDefault без перезавантаження контроллера.

Трохи нижче розташовані 2 кнопки:

“FavoritePosts”/”Saved locally” відповідно для перегляду улюблених/збережених постів, обидва режима використовують БД й працюють офлайн.

Така схема навігації гарантує інтуїтивне керування джерелами контенту та їх фільтрами з одним натисканням у верхній панелі у межах MemeViewController, і інтуїтивний перехін між екранами фіду та налаштувань у межах всього додатку.

2.6 Налаштування локалізації, світлого/темного оформлення

При проєктуванні інтерфейсу, відповідно до Human Interface Guidelines, слід «design your interface to respect the system’s light and dark appearance modes, adapting dynamically to user preferences to ensure visual consistency and accessibility.» [17]

Для зручності користувачів MemeShuffler підтримує по дві мови інтерфейсу та оформлення.

Для цього, в екрані налаштувань, модулі Appearance, користувач може окремо обирати варіант який йому до вподоби.
Для обирання теми оформлення було додано UISegmentedControl з двома опціями (“Light”, “Dark”), вибір якого автоматично перезапише значення обраної теми за допомогою overrideUserInterfaceStyle і збереже обране значення у UserDefault. Таким чином при завантаженні кожного контроллера у методі viewWillAppear(_:) перевіряється поточне значення SettingsManager.interfaceTheme та встановлюється відповідний стиль.
Для обирання локалізації було створено файл з локалізацією Localizable, що містить ключ й набір значень для різних обраних мов.
Приклад: ключ: error – end: “Error” – ukr: “Помилка”
Для того щоб побачити зміни треба перезайти у додаток, адже динамічна зміна локалізації не задумана Apple
Таким чином, можна ефективно перемикати мову абсолютно всіх надписів у додатку.

РОЗДІЛ 3: Реалізація функціональних модулів додатку

3.1 Інтеграція з Reddit API

У MemeShuffler усі запити до Reddit API [8] виконуються за допомогою стандартного URLSession по захищеному протоколу HTTPS із перевіркою SSL-сертифікатів операційною системою iOS. Для цього був спеціально написаний окремий модуль MemeApiHandler, що являє собою повноцінний – міні проект, який здатен робити форматовані запити й повертати значення у вигляді масиву структур. Для формування запиту на отримання стрічки постів застосовується шаблонна адреса https://www.reddit.com/r/[subreddit]/[filter].json?limit=[limit], де subreddit, та filter (“top”, “hot” або “new”) підставляються на основі вибору користувача в інтерфейсі, а limit визначається через налаштування завантажуваної кількості постів. У разі відсутності мережі або нестабільного з’єднання реалізовано механізм автоматичного повторення запиту з лінійною затримкою та відображенням дружнього повідомлення користувачу про тимчасову недоступність контенту.

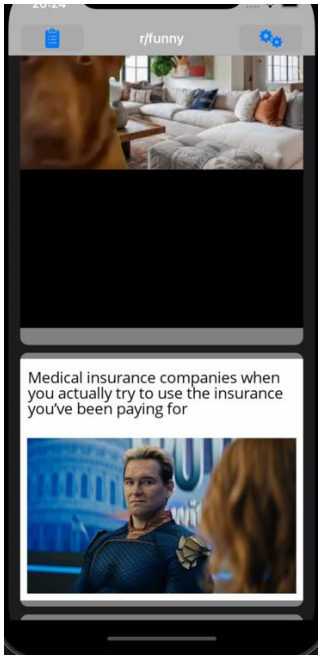
```
private static func requestMemesPortion(completion: @escaping ([Meme]?, Error?) -> Void) {
    let subreddit = parameters.getSubredditName()
    let quantity = parameters.getQuantity()
    let after = parameters.getAfter()
    let filter = parameters.getFilter().lowercased()

    var components = URLComponents(string:
        "https://www.reddit.com/r/\(subreddit)/\(filter).json?limit=\(quantity)")!
```

Отриманий від сервера JSON-документ спочатку декодується за допомогою JSONDecoder в проміжні Swift-структури, що відображають основні поля Reddit-API (ідентифікатор, заголовок, URL медіа, тип, час створення). Після цього для збереження даних у локальній базі використовується приватний контекст Core Data з .privateQueueConcurrencyType: кожна сутність PostEntity створюється або оновлюється в ізольованій черзі, що дозволяє не блокувати головний потік

UI. Далі зміни комітуються в NSPersistentContainer з політикою злиття mergeByPropertyObjectTrump, а отримані об’єкти у вигляді NSManagedObject передаються в головний контекст через perform(_:) для автоматичного оновлення UITableView без перезавантаження контролера.

Крім того, інтеграція з фоновим механізмом Background Fetch забезпечує періодичну синхронізацію нових постів у момент, коли додаток знаходиться у фоні. Після виконання фонової задачі система передає керування приватному контексту, який зберігає нові сутності в Core Data, а потім викликає завершальний хендлер, щоб операційна система знала про успішний або частково вдалий результат. Завдяки цьому користувач завжди має останню доступну копію стрічки, навіть якщо інтернет-з’єднання зникає на тривалий час.

3.2 Відображення контенту UITableView із динамічною висотою ячеєк.



Для створення ефекту “обіймання” медіа-контенту будь-якого типу, де кожна ячейка має унікальну висоту, використовується певний механізм: Поки на не відомий розмір медіа контенту, ми спочатку задаємо дефолтний розмір ячейки у 400 пікселів.

Далі, поли ми отримуємо Media й його розміри, ми вираховуємо співвідношення сторін цього медіа-контенту, щоб потім перенести це співвідношення на ячейку для коректного відображення й тим самимробимо більш привабливий вигляд постів й заодно запобігаємо обрізанню.

```
) {
    let ratio     = resolution.height / resolution.width
    let newHeight = tableView.frame.width * ratio
    memes[indexPath.row].width  = Double(resolution.width)
    memes[indexPath.row].height = Double(resolution.height)
    cellHeights[indexPath]      = newHeight

    DispatchQueue.main.async {
        self.tableView.beginUpdates()
        self.tableView.endUpdates()
    }
}
```

3.3 BackgroundFetch та збереження даних у CoreData

Фонову синхронізацію організовано за допомогою BGTaskScheduler: під час запуску в AppDelegate реєструється завдання оновлення з унікальним ідентифікатором, далі кожного разу при переході в бекграунд викликається метод, який створює запит на виконання BGAppRefreshTaskRequest із затримкою щонайменше в 15 хвилин. Коли система пробуджує додаток у фоні, виконується функція обробки завдання, яка запускає завантаження свіжих мемів за допомогою MemeApiManager.loadMemesCompilation і очікує на результат, а потім повідомляє iOS про успіх або невдачу через setTaskCompleted, гарантуючи безперебійну роботу фону навіть за тривалих перерв у роботі мережі.

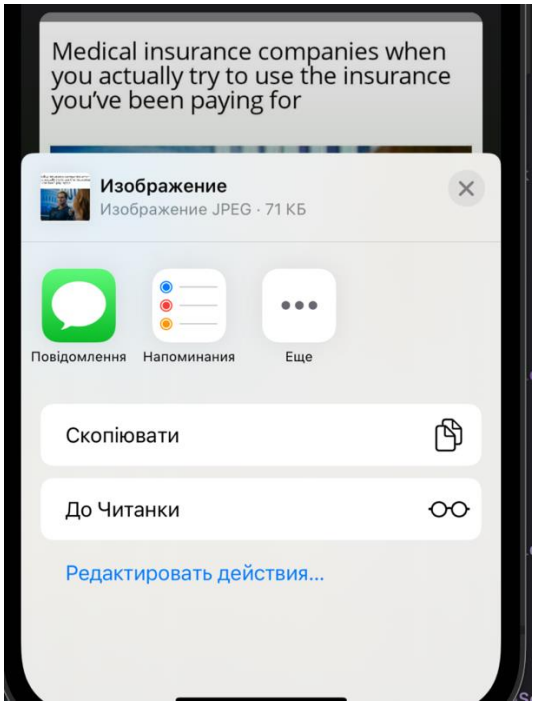
Після отримання масиву моделей Meme обробка переходить до CoreDataManager.preloadPosts, де метод performBackgroundTask створює приватний контекст Core Data, який виконує завантаження і збереження даних незалежно від основного потоку. Усередині цього контексту створюється група синхронізації, у якій для кожного мему через KingfisherManager завантажується зображення і розраховуються його розміри, після чого створюється сутність Post із полями і прапорцем

isDownloaded. Лише після завершення всіх завантажень і очікування group.wait() контекст виконує збереження, а успіх або помилка передаються в зворотний виклик, що дозволяє AppDelegate коректно завершити фонове завдання.

Крім цього, у AppDelegate реалізовано обробник didReceiveMemoryWarning, який очищує кеш зображень Kingfisher [11] та URLCache, щоб запобігти перевитраті пам'яті, а також кожен запуск scheduleAppRefresh гарантує, що оновлення стрічки відбуватиметься системно та безперервно. Такий комплексний підхід поєднує можливості BackgroundTasks і Core Data, забезпечуючи користувачеві актуальну локальну копію мемів навіть за умов тривалих відключень інтернету чи живлення й без блокування інтерфейсу

3.4 Реалізація опції "Share" для поширення постів.

У MemeViewController опцію «Share» реалізовано через метод UITableViewDelegate tableView(:trailingSwipeActionsConfigurationForRowAt:): при свайпі ліворуч по комірці створюється UIContextualAction зі стилем .normal і заголовком «Share», а в його обробнику формується URL поточного поста із властивості post.urlString та передається в UIActivityViewController(activityItems: [shareURL], applicationActivities: nil). Після налаштування зовнішнього вигляду дії (наприклад, *backgroundColor* = .systemBlue і systemImage «square.and.arrow.up») контролер відображається модально через present(:animated:), а виклик completionHandler(true) інформує UITableView про успішне виконання дії — це дозволяє користувачеві за лічені секунди відправити посилання на улюблений мем будь-яким доступним у системі iOS способом.



3.5 Реалізація опції "Favorite" для збереження постів.

У MemeShuffler позначення постів як улюблених організовано таким чином, щоб користувач міг одним жестом на екрані швидко зберегти або видалити мем із розділу «Улюблені» без зайвих інструментів чи діалогових вікон. Після свайпу по комірці відповідний елемент отримує інтуїтивний напис «Favorite» або «Unfavorite» залежно від поточного стану — це дає користувачеві чітку візуальну підказку. При виборі цієї дії додаток відразу реагує на кшталт миттєвого підтвердження: колір кнопки змінюється, інтерфейс плавно оновлюється, і в таблиці з'являється або зникає відповідна позначка, що наочно демонструє успіх операції.

Внутрішня логіка обробки фаворитів реалізована через єдиний клас-менеджер CoreDataManager, який інкапсулює всі операції з базою Core Data. Коли користувач позначає пост як улюблений, менеджер перевіряє наявність сутності з таким ідентифікатором у локальному сховищі. Якщо запис уже існує, він просто оновлює прапорець isFavorite, інакше створює

новий об’єкт із необхідними полями (ідентифікатор, заголовок, посилання на медіа та розміри), задає йому статус «в улюблених» і зберігає контекст. Навпаки, при знятті позначки менеджер знаходить відповідний запис і змінює лише поле isFavorite, залишаючи всі інші дані незмінними. Такий підхід дозволяє зберігати історію всіх завантажених постів і водночас формувати окрему стрічку улюблених, не дублюючи об’єкти в базі.

Інтерфейс автоматично підтягує новий стан через зв’язок з NSFetchedResultsController або еквівалентом, тому користувач бачить оновлену таблицю без затримок і повного перезавантаження екрана. Для оптимізації роботи передбачено індексування поля isFavorite, що дозволяє миттєво отримувати список лише улюблених постів навіть при великій кількості записів у базі. Завдяки поділу операцій на читання та запис у різних контекстах Core Data, а також використанню фонових черг, додаток уникає блокування інтерфейсу й забезпечує високий рівень продуктивності.

Нарешті, стан улюблених постів зберігається між сеансами завдяки постійному збереженню контексту Core Data, а менеджер неодноразово обробляє потенційні помилки збереження та вилучення, щоб уникнути втрати даних. Якщо виникає збій під час оновлення, користувача інформують через ненав’язливе сповіщення, і система автоматично робить повторну спробу, аби гарантувати цілісність улюбленої колекції.

РОЗДІЛ 4: Безпека та оптимізація додатку

4.1 Оптимізація продуктивності та кешування

Для забезпечення швидкої та надійної роботи клієнт-додатка особливу увагу приділено оптимізації продуктивності та ефективному використанню кешування. Розглянуті рішення включають інтеграцію бібліотек для кешування зображень, налаштування фонових завдань та управління локальними даними.

1. Використання Kingfisher для кешування зображень
   Бібліотека Kingfisher автоматично зберігає завантажені зображення в своєму кеші, тому при повторному показі тих самих URL додаток не виконує зайвих HTTP-запитів і миттєво відображає меми з локального сховища. Це значно пришвидшує скролінг стрічки та знижує навантаження на мережу.

```
KingfisherManager.shared.retrieveImage(with: url) { result in
    defer { group.leave() }
    let (w, h): (Double, Double)
    if case .success(let value) = result {
        w = Double(value.image.size.width)
        h = Double(value.image.size.height)
    } else {
        w = 0; h = 0
    }
```

2. Очищення кешів при низькій пам’яті
   У AppDelegate реалізовано обробник memory warning, який очищує кеш пам’яті Kingfisher та загальний HTTP-кеш URLCache. Це допомагає запобігти аварійним завершенням програми на старіших пристроях або при обмежених ресурсах.

```swift
// MARK: - Memory Warning Handler
@objc private func didReceiveMemoryWarning() {
    ImageCache.default.clearMemoryCache()
    URLCache.shared.removeAllCachedResponses()
}
```

3. Фонова синхронізація через BGTaskScheduler

   Замість постійного пулу чи таймера для оновлення стрічки додаток реєструє BGAppRefreshTaskRequest із інтервалом не менше 15 хвилин. iOS сама пробуджує додаток у фоні, коли є ресурси, що значно знижує витрати батареї й забезпечує регулярне оновлення контенту.

```swift
// MARK: - BackgroundTasks Scheduling
private func scheduleAppRefresh() {
    let request = BGAppRefreshTaskRequest(identifier: refreshTaskIdentifier)
    request.earliestBeginDate = Date(timeIntervalSinceNow: 15 * 60)
    do {
        try BGTaskScheduler.shared.submit(request)
    } catch {
        print("Could not schedule app refresh: \(error)")
    }
}
```

4. Паралельне завантаження з DispatchGroup

   Для одночасного завантаження великої кількості медіа використовується DispatchGroup. Група чекає завершення всіх запитів перед збереженням у базу, що скорочує загальний час підготовки офлайн-копії стрічки.

```swift
MemeApiManager.loadMemesCompilation { newMemes in
    guard let memes = newMemes, !memes.isEmpty else { completion(false); return }
    self.persistentContainer.performBackgroundTask { ctx in
        let group = DispatchGroup()
        for meme in memes {
            guard let urlStr = meme.urlString, let url = URL(string: urlStr) else {
                continue }
            group.enter()
```

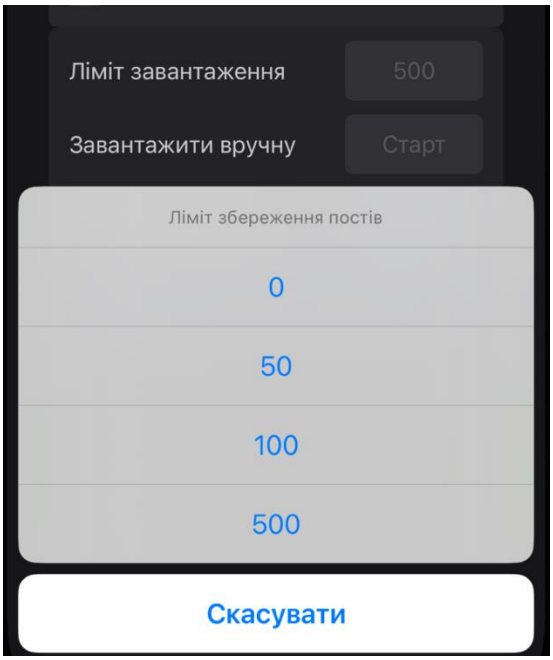
5. Політика злиття Core Data

   NSPersistentContainer налаштовано з mergePolicy = mergeByPropertyObjectTrump, що автоматично вирішує конфлікти при паралельному записі з фонового та головного контекстів без втрати змін.

6. Підготовка до обмеження кешу

   Налаштування localSaveLimit у SettingsManager задає максимальний обсяг локального кешу. Хоча видалення старих записів ще не автоматизовано, це створює основу для використання NSBatchDeleteRequest у майбутніх релізах і дозволяє контролювати розмір сховища

```
public static var localSaveLimit: Int {
    get { UserDefaults.standard.integer(forKey: Keys.localSaveLimit) }
    set { UserDefaults.standard.set(newValue, forKey: Keys.localSaveLimit) }
}
```



4.2 Забезпечення безпеки HTTP-заголовків і захист даних

Усі мережеві запити в MemeShuffler здійснюються виключно по захищеному протоколу HTTPS відповідно до вимог App Transport Security (ATS) iOS [13]. Для цього URLSession використовує стандартну перевірку SSL-сертифікатів операційною системою, що гарантує захист від Sniffing та MITM-атак на рівні транспортного шляху. Додатково в заголовках запитів автоматично відправляється поле Accept-Language, значення якого

синхронізоване з налаштуваннями SettingsManager.interfaceLanguage, це дозволяє серверу віддавати локалізовані ресурси й унеможливлює підміни контенту.

На стороні збереження даних застосовано кілька рівнів захисту. Локальна база Core Data розміщується у каталозі додатка з політикою файлового шифрування NSFileProtectionCompleteUntilFirstUserAuthentication [15], що забезпечує автоматичне шифрування файлу сховища до першого успішного розблокування пристрою. Параметри інтерфейсу (UserDefaults) і кеші Kingfisher зберігаються в App Sandbox[16], а при отриманні нотифікації didReceiveMemoryWarning[12] усі кешовані відповіді та зображення очищуються, щоб запобігти потенційному переповненню та витоку невикористаних даних.

Важливий аспект безпеки - обробка введених користувачем даних (список сабредітів). При додаванні нової адреси ім’я сабредіту автоматично обрізається від пробілів і проходить найпростішу валідацію на порожні рядки, що мінімізує ризик спроб ін’єкцій чи навмисного формування некоректних URL. Таким чином, поєднання захищеного каналу передачі, надійного шифрування локального сховища та базової валідації вхідних даних гарантує цілісність і конфіденційність інформації в додатку навіть за нестабільних умов мережі чи нападах.

Висновки

У результаті виконання курсової роботи було створено мобільний iOS-додаток **MemeShuffler**, який поєднує інтуїтивно зрозумілий інтерфейс і надійну бекенд-логіку для перегляду контенту платформи Reddit за будь-яких умов мережевого зв’язку.

Аналіз існуючих клієнтів (офіційного, Apollo, Narwhal) виявив обмеження щодо офлайн-режиму, динамічного масштабування комірок і фонового оновлення стрічки. У відповідь на це в **MemeShuffler** реалізовано повноцінну підтримку фонової синхронізації через BGAppRefreshTaskRequest і Background Fetch, а контент зберігається у локальній базі Core Data із приватними контекстами та політикою mergeByPropertyObjectTrump.

Архітектура додатка вибудована за патерном MVC із чітким розділенням Model (мережеві запити та обробка JSON, управління Core Data), View (UIKit-елементи з Auto Layout для адаптивного відображення зображень, відео й тексту) та Controller (UIViewController, які управляють логікою UI, викликають ApiService і оновлюють таблицю). Такий підхід забезпечує простоту підтримки та інтеграцію з документацією Apple.

Для оптимізації продуктивності застосовано кешування зображень через Kingfisher, HTTP-кешування URLCache, паралельну обробку запитів через DispatchGroup та фонове виконання запису в базу через performBackgroundTask. Обробка помилок мережі з повторними спробами та дружні повідомлення гарантують стійку роботу навіть при нестабільному інтернеті.

Питання безпеки вирішено через ATS із суворою перевіркою SSL-сертифікатів, шифрування файлів Core Data політикою

NSFileProtectionCompleteUntilFirstUserAuthentication, захист App Sandbox та базову валідацію введення підписок. Це забезпечує цілісність даних і захищає від MITM-атак.

Користувацькі налаштування (сабредіти, фільтри, мова, тема, ліміт кешу) зберігаються в UserDefaults через клас SettingsManager, що дозволяє гнучко адаптувати роботу додатка під індивідуальні потреби.

У підсумку **MemeShuffler** демонструє ефективне поєднання сучасних підходів iOS-розробки, гарантує безперебійний офлайн-доступ до розважального контенту та закладає фундамент для подальшого розвитку функціоналу та розширення масштабів застосунку.

Джерела

EAWORLDVIEW. Ukraine war energy grid attacks – almost 50 % of Ukrainian energy system down, 10 million people without power (19 Nov 2022). URL: https://eaworldview.com/2022/11/ukraine-war-energy-russia-attacks/

The Guardian. City life during Kyiv's power blackouts – in pictures (Aug 5 2024). URL: https://www.theguardian.com/world/article/2024/aug/05/city-life-during-kyivs-power-blackouts-in-pictures

Balackburn. Apollo – a powerful iOS Reddit client with advanced gestures and caching. GitHub. URL: https://github.com/Balackburn/Apollo

PawlowskiAlex. Narwhal – minimalistic and fast Reddit client for iOS. GitHub. URL: https://github.com/pawlowskialex/Narwhal

Apple Developer. Model–View–Controller in Cocoa/Cocoa Touch. URL: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

FortechRomania. ios-mvp-clean-architecture – example of MVP and Clean Architecture for iOS in Swift. GitHub. URL: https://github.com/FortechRomania/ios-mvp-clean-architecture

Kodeco Team. iOS MVVM Tutorial: Refactoring from MVC. Kodeco. URL: https://www.kodeco.com/6733535-ios-mvvm-tutorial-refactoring-from-mvc

Reddit, Inc. Reddit API Documentation. URL: https://www.reddit.com/dev/api/

Apple Developer. Core Data Framework. URL: https://developer.apple.com/documentation/coredata

Postman, Inc. Postman Documentation: Testing APIs. URL: https://www.postman.com/docs/getting-started/introduction/

onevcat. Kingfisher – A lightweight, pure-Swift library for downloading and caching images. GitHub. URL: https://github.com/onevcat/Kingfisher

Apple Developer. UIApplication.didReceiveMemoryWarningNotification – Handling memory warnings. URL: https://developer.apple.com/documentation/uikit/uiapplication/didreceivememorywarningnotification

Apple Developer. App Transport Security – Network Security Policy. URL: https://developer.apple.com/documentation/bundleresources/information_property_list/nsapptransportsecurity

Apple Developer. Preventing Man-in-the-Middle Attacks. URL: https://developer.apple.com/documentation/security/preventing_mitm_attacks

Apple Developer. NSFileProtectionCompleteUntilFirstUserAuthentication – File Protection. URL: https://developer.apple.com/documentation/foundation/nsfileprotectioncompleteuntilfirstuserauthentication

Apple Developer. App Sandbox Design Guide. URL: https://developer.apple.com/library/archive/documentation/Security/Conceptual/AppSandboxDesignGuide/Introduction/Introduction.html

Apple Developer. Human Interface Guidelines (iOS). URL: https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/

Stack Overflow. Merge conflict: “Core Data merge conflict”. URL: https://stackoverflow.com/questions/51235259/merge-conflict-core-data

Stack Overflow. performBackgroundTask crashing on save.
URL: https://stackoverflow.com/questions/43502245/coredatas-performbackgroundtask-method-crashes

Stack Overflow. NSFetchedResultsController not updating UITableView.
URL: https://stackoverflow.com/questions/32621535/nsfetchedresultscontroller-not-updating-when-saving-to-core-data

Stack Overflow (RU). UIActivityViewController share sheet not appearing.
URL: https://ru.stackoverflow.com/questions/1098808/Ошибка-при-вызове-share-sheet

Stack Overflow. didReceiveMemoryWarning not being called.
URL: https://stackoverflow.com/questions/2430728/how-to-implement-didreceivememorywarning

Stack Overflow. SwiftPM warning about dependency conflicts when working on a local package.
URL: https://stackoverflow.com/questions/71635299/swiftpm-warning-about-dependency-conflicts-when-working-on-a-local-package

Stack Overflow. SwiftUI “The data couldn’t be read because it isn’t in the correct format”.
URL: https://stackoverflow.com/questions/69590624/swiftui-the-data-couldn-t-be-read-because-it-isn-t-in-the-correct-format

Stack Overflow. Decoding multiple types for the same key swift JSONDecode decoding arrays fails if single element decoding fails.
URL: https://stackoverflow.com/questions/70844544/decoding-multiple-types-for-the-same-key

Stack Overflow. Swift JSONDecode decoding arrays fails if single element decoding fails.
URL: https://stackoverflow.com/questions/46344963/swift-jsondecode-decoding-arrays-fails-if-single-element-decoding-fails

Stack Overflow. “Unable to simultaneously satisfy constraints” with UITableViewCell.
URL: https://stackoverflow.com/questions/69738850/unable-to-simultaneously-satisfy-constraints-with-uitableviewcell

Stack Overflow. dequeue reusable cell always crashes – UITableViewCell reuse identifier issue.
URL: https://stackoverflow.com/questions/40557899/dequeue-reusable-cell-always-crashes

Stack Overflow. UIView overrideUserInterfaceStyle not working.
URL: https://stackoverflow.com/questions/73252436/uiview-overrideuserinterfacestyle-is-not-working

Stack Overflow. Dynamic Type не змінює розмір шрифту в UILabel
URL: https://stackoverflow.com/questions/66848765/after-changing-the-preferred-content-size-the-font-size-in-some-of-my-reusable

Stack Overflow. UIImageView contentMode rules break Auto Layout
URL: https://stackoverflow.com/questions/25869308/why-changing-uiimageview-content-mode-affect-its-auto-layout-in-xcode6

Stack Overflow. BackgroundFetch із BGAppRefreshTaskRequest повертає

noData

URL: https://stackoverflow.com/questions/55688796/when-would-i-ever-need-to-use-nodata-when-calling-the-complete-function-on-ba