Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розробка мобільного додатку операційної системи iOS для здійснення контролю історії пересування користувача з використанням мови Swift та середовища Xcode

Виконав(ла): студент(ка) VI курсу, групи СПм-61
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Стандрет П.О. (прізвище та ініціали)

Керівник (підпис) Цуприк Г.Б. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) Дуда О.М. (прізвище та ініціали)

Тернопіль
2021

РЕФЕРАТ

Кваліфікаційна робота магістра на тему “Розробка мобільного додатку операційної системи iOS для здійснення контролю історії пересування користувача з використанням мови Swift та середовища Xcode” Стандрета Петра Олеговича. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121, група СПм–61, Тернопіль, 2021. С. – , рис. – , табл. – , додат. – , бібліогр. – .

Мета роботи – мобільний застосунок для запису та відображення локацій користувача. Дана робота включає розробку програмного забезпечення на основі використання баз даних.

Практичні методи полягають в використані технологій iOS SDK та мови програмування Swift включно із середовищем Xcode.

В результаті розробки розроблено мобільний застосунок для iOS який надає мінімально – необхідний функціонал для запису і відображення локацій який буде зручним для користувача.

Ключові слова: iOS, SWIFT, XCODE, SQLITE, GPS, SIGNIFICANT LOCATION, CORE LOCATION, МОБІЛЬНИЙ ЗАСТОСУОК, ЛОКАЦІЇ

ANNOTATION

Master's qualification work on the topic "Development of a mobile application of the iOS operating system for monitoring the history of user movement using the Swift language and Xcode environment" by Peter Olegovich Standret. Ivan Pulyuy Ternopil National Technical University, 121 Software Engineering, Department of Software Engineering, SPm-61 Group, Ternopil, 2021. C. - , fig. – , table. - , appendix. - , bibliogr. – .

The purpose of the project is a mobile application for recording and displaying user locations. This work includes the development of software based on the use of databases.

Practical methods include the use of iOS SDK technologies and Swift programming languages, including the Xcode environment.

As a result of development the mobile application for iOS which provides the minimum - necessary functionality for record and display of locations which will be convenient for the user is developed.

Key words: iOS, SWIFT, XCODE, SQLITE, GPS, SIGNIFICANT LOCATION, CORE LOCATION, МОБІЛЬНИЙ ЗАСТОСУОК, ЛОКАЦІЇ.

ЗМІСТ

ВСТУП

Традиційно на протязі життя людина відвідує різноманітні місця, їздить на автомобілі чи громадському транспорті, літає у відпустки за кордон або відвідує цікаві місця які часто не позначені на картах. З плином часу людська пам'ять втрачає можливість з точністю пригадати конкретне місце і точний час коли було відвідане те чи інше місце, і насправді це є проблемою бо люди часто хочуть через роки згадати і освіжити в пам’яті спогади, чи поділитись цікавими місцями із іншими. Для збереження особливих моментів люди робили фотографії на плівкові фотоапарати а сьогодні фотографують на цифрові фотоапарати чи телефони і роблять підписи до них, інші використовують спеціалізовані GPS трекери, дехто веде довідник. В даному випадку згадані варіанти вирішують проблему частково, оскільки різноманітність джерел інформації та їхній обсяг росте, і систематизувати їх стає все складніше і складніше, так само як і шукати в них необхідну інформацію.

Для того щоби вирішити дану проблему було поставлене завдання на створення мобільного застосунку який візьме на себе рутинну задачу збереження інформації і представлення її користувачеві у зручному для нього вигляді із можливістю використання різноманітних інструментів вибірок чи фільтрацій. Цінність даної розробки і дослідження яке вона передбачає є доволі вагомою оскільки головним об’єктом є користувач і його локації якими він оперує а дані зокрема хронологічні є доволі вартісні коли потрібно щось згадати через роки.

Проєкт буде корисний для групи користувачів яким важлива статистика, інформація про пересування і відвідані місця з точною хронологією та акцентом на конфіденційність локаційних даних, цільовій аудиторії яка цінує гнучкість в налаштуваннях та в різноманітті потенційних можливостей в налаштування вибірок даних і доступні можливості оперування ними.

Для імплементації даного рішення як цільову операційну систему було обрано iOS а мову програмування Swift. Застосунок буде використовувати останню версію мови програмування Swift і можливості API iOS для контролю локаційних

сервісів, зокрема це локаційні мітки (істотні місце пересування) і візити (місця де користувач залишався деякий час) а також аналіз гео-даних із фотографій. Для оптимізації роботи будуть використовуватись також сторонні бібліотеки для роботи із базою даних SQLite, розширення для мови Swift – RxSwift і бібліотека стандартних компонентів для застосунків під iOS. Застосунок буде розповсюджуватись через офіційний магазин AppStore для готових збірок та систему TestFlight для тестування та аналізу збірок що в розробці.

З точки зору програмної інженерії, кінцевий програмний продукт має записувати і надавати дані локацій користувача в хронологічному порядку, що є відфільтрованими і згрупованими по заданих параметрах і ефективно працюють на великих об’ємах даних. Власне процес дослідження і розробки наведено безпосередньо у кваліфікаційній роботі.

1 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

1.1 Аналіз вимог до програмної системи

1.1.1 Аналіз предметної області

Проблема збереження користувацьких локацій в сьогодення є доволі великою проблемою, адже як відомо людська пам’ять не є ідеальною і з плином часом ви забуваємо про відвідані місця, цікаві моменти. Для збереження особливих моментів люди робили фотографії на плівкові фотоапарати а сьогодні фотографують на цифрові фотоапарати чи телефони і роблять підписи до них, інші використовують спеціалізовані GPS трекери, дехто веде довідник.

Як один із варіантів вирішення проблем це надати можливість користувачам мати інформацію про відвідини в одному місці, нехай це будуть фотографії із GPS координатами, чи GPX подорожі записані спеціальними пристроями, чи банально прості пересування користувача в конкретний день і певній локації. Для цього мною було вирішено, спочатку для власних цілей, створити додаток який би міг збирати інформацію про пересування користувача в одному місці і надавати зручний інтерфейс для перегляду із можливість фільтрації.

Одним з лідерів на ринку є безпосередньо компанія Google із своїм продуктом Google Maps Timeline (рис. 1.1). Сама компанія вказує що Google Maps Timeline показує приблизні місця, якими ви могли бувати, і маршрути, які ви могли пройти на основі вашої історії місцезнаходжень. Ви можете будь-коли редагувати свою хронологію. Ви також можете будь-коли видалити свою історію місцезнаходжень, включаючи часові діапазони, у хронології. Хронологія є приватною, тому її можете бачити лише ви, і вона доступна на мобільних пристроях і комп’ютерах [1].

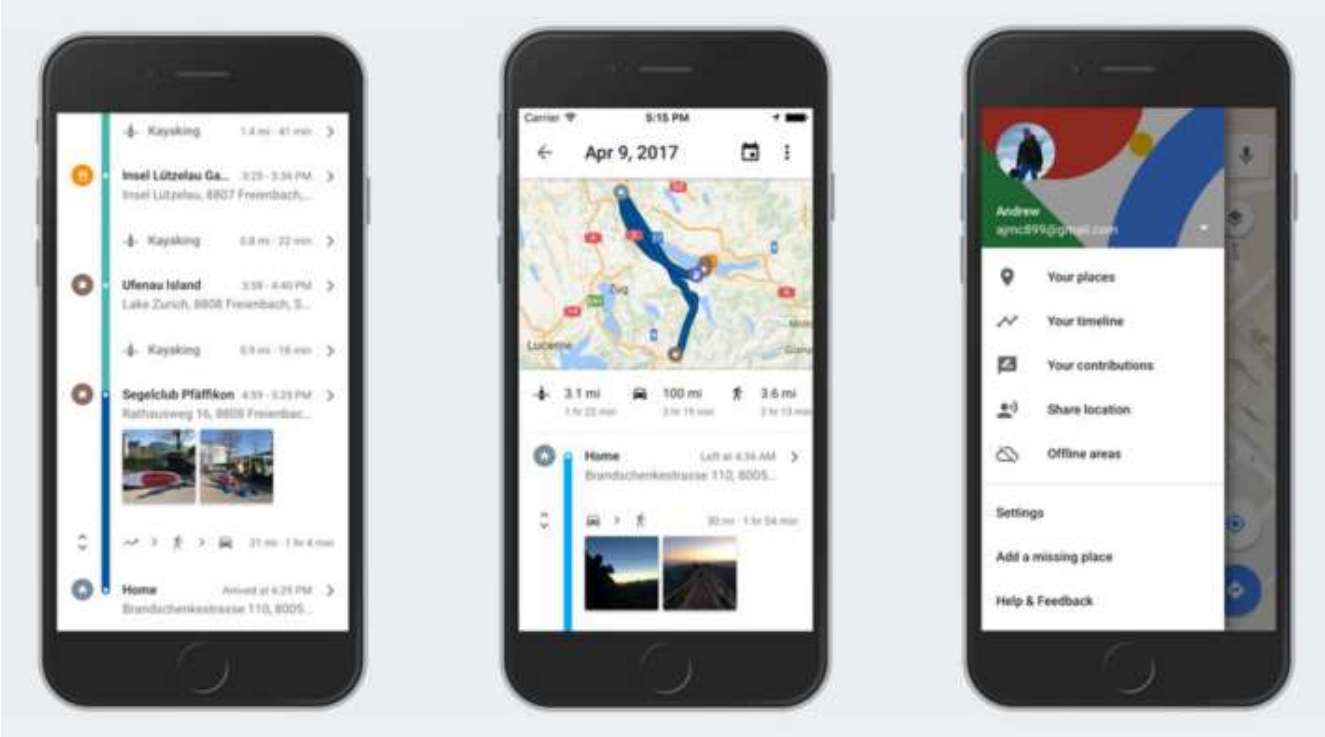

Рисунок 1.1 – Google Maps Timeline

Проте безпосередньо при своїх перевагах як от доступність на багатьох платформах чи широкий функціонал і підв'язка до систем Google тут є великий недолік, а це безпосередньо приватність даних користувача. Як вказано у описі продукту дані про місцезнаходження користувача можуть залишатись на серверах компанії навіть якщо історія відвідин була вимкненою [1]. Це є доволі великий недолік для людей які дбають про безпеку своїх особистих даних. Також дана система не підтримує на даний момент імпорту інших локацій із інших джерел.

Як аналог від компанії Apple існує інша система яка є безпосередньо вбудована в операційну систему Significant Locations (рис. 1.2). Дана система працює досить схоже до Google Maps Timeline але вона не має настільки гнучких налаштувань по періоду чи перегляду маршрутів. Сама компанія позиціонує сервіс як систему що використовує пристрої користувача, підключені до iPhone, які відстежуватимуть місця, які користувач нещодавно відвідував, а також те, як часто і коли, щоб дізнатися про важливі для нього місця. Ці дані мають наскрізне зашифровані і не можуть бути прочитані Apple [2].

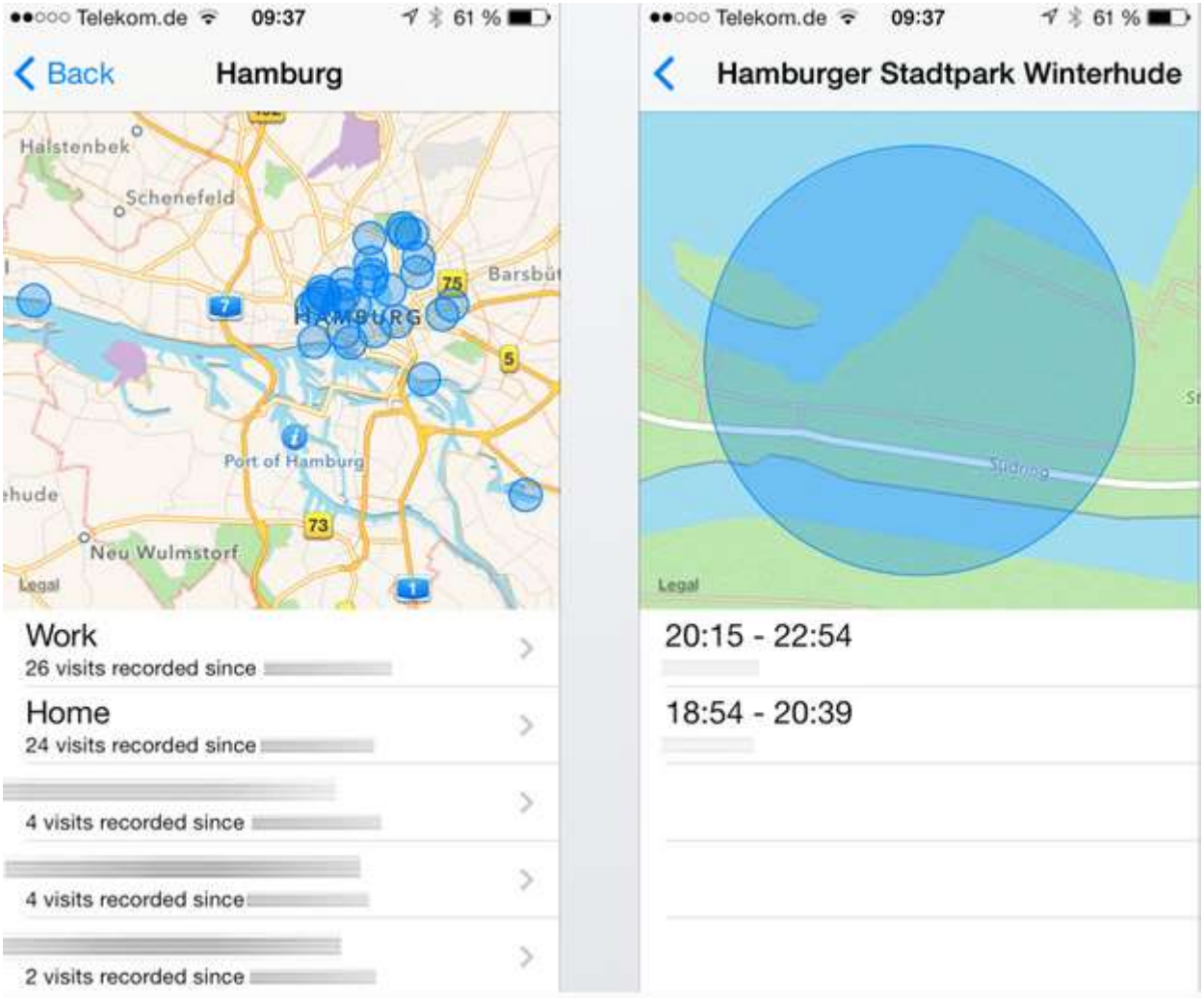

Рисунок 1.2 – Apple Significant Locations

На відміну від конкурента Apple Maps вказують що дані користувача є зашифровані і не можуть бути прочитані компанією а також те що після видалення вони моментально видаляються із серверів, що в плані безпеки є досить хорошим показником [2]. Але функціонал є доволі скромним що не дозволяє в повній мірі використовувати сервіс.

1.1.2 Постановка задачі для розробки

Завданням магістерської роботи є розробка інформаційної системи для контролю історії пересування користувача. Проаналізувавши подібні програмні забезпечення, було зроблено висновок, що існуючі подібні програмні забезпечення, на жаль мають недостатню кількість функціоналу в одному місці або не зберігаюсь конфіденційність користувача. Тому щоб моїй інформаційній системі виділитись потрібно бути кращою. Отже потрібно використовувати свіжий дизайн, прибрати інформацію яка є зайвою для користувача, надати потужні можливості для оперування інформації, зробити якнайшвидший відклик на дії користувача і що саме головне забезпечити високу конфіденційність і безпеку даних користувача. На швидкість роботи і безпеку системи також впливають і вибрані технології, тому буде вибрано найновіші та стабільні технології.

В якості основної операційної системи було вибрано систему iOS і мінімально допустиму версію було встановлено як iOS 11 для того щоби залучити побільше користувачів. Хоча дана версія була випущена 4 роки тому вона залишається доволі актуальною і стабільною [4]. Звісно вона не підтримує новий UI від Apple SwiftUI але на даному етапі це є скоріше перевагою ніж недоліком, оскільки згідно дослідження різних розробників вона є досить сирою і відповідно поки не рекомендують її використовувати її в продакшені [5]. Зважаючи на інформацію було вирішено продовжувати розробку із використанням UIKit який за роки продемонстрував стабільність і гнучкість в роботі.

Зважаючи на попередні вимоги до додатку і операційної системи було визначено наступні етапи розробки:

1. Визначення акторів та варіантів використання
2. Вибір архітектурного патерну та допоміжних бібліотек
3. Побудова схеми БД
4. Реалізація системи

1.1.3 Визначення акторів та варіантів використання

Моделювання варіантів використання – це спеціалізований тип структурного моделювання, пов’язаний з моделюванням функціональності системи. Зазвичай ми застосовуємо моделювання варіантів використання під час аналізу вимог які визначають, що повинна робити система. Моделювання варіантів використання зазвичай починається рано в проєкті і продовжується протягом усього процесу розробки системи [6, с.103].

Варіант використання - це функціональна вимога, описана в точки зору користувачів системи. Наприклад, функціональні вимоги до проєкту системи керування включає в себе: функціональні можливості безпеки (наприклад, надання користувачам можливості входити в систему та виходити з неї система), введення даних, обробка даних, формування звітів тощо.

Варіант використання визначає функціональну вимогу, яка описується як послідовність кроків, які включають дії, що виконуються системою, і взаємодії між системою та акторами. Випадки використання вирішують питання про те, як учасники взаємодіють із системою, і описують дії, які виконує система [6, с.107].

Актор — це користувач або зовнішня система, з якою взаємодіє система, що моделюється. Наприклад, наша система управління проєктами включає два типи користувачів, включаючи систему і користувача. Усі ці користувачі є акторами. Актор є зовнішнім по відношенню до системи, взаємодіє з системою, може бути користувачем-людиною або іншою системою, і має цілі та відповідальність, які необхідно задовольнити під час взаємодії з системою. Актори вирішують питання про те, хто і що взаємодіє з системою. У UML актор відображається у вигляді піктограми «фігурка» або як клас, позначений ключовим словом actor і позначений назвою класу актора [6].

Під час аналізу системи було визначено лише двох акторів – Користувач і Система. Система не передбачає більше ролей оскільки не передбачається можливостей адміністрування, різних рівнів доступу чи будь якого іншого

складнішого використання. Оскільки система проєктується для використання під мобільні пристрої то відповідно і можливе використання тільки одним користувачем одночасно.

Користувач має мати змогу переглядати і оперувати даними представленими в застосунку, при бажанні він також має мати змогу експортувати їх у спеціально визначених форматах. Система при відповідних дозволах користувача здійснює активний або пасивний моніторинг за пересуванням користувача і його контентом.

Після виявлення акторів системи побудуємо діаграму варіантів використання для кращого відображення звязків актора та визначеного функціоналу (рис. 1.3)

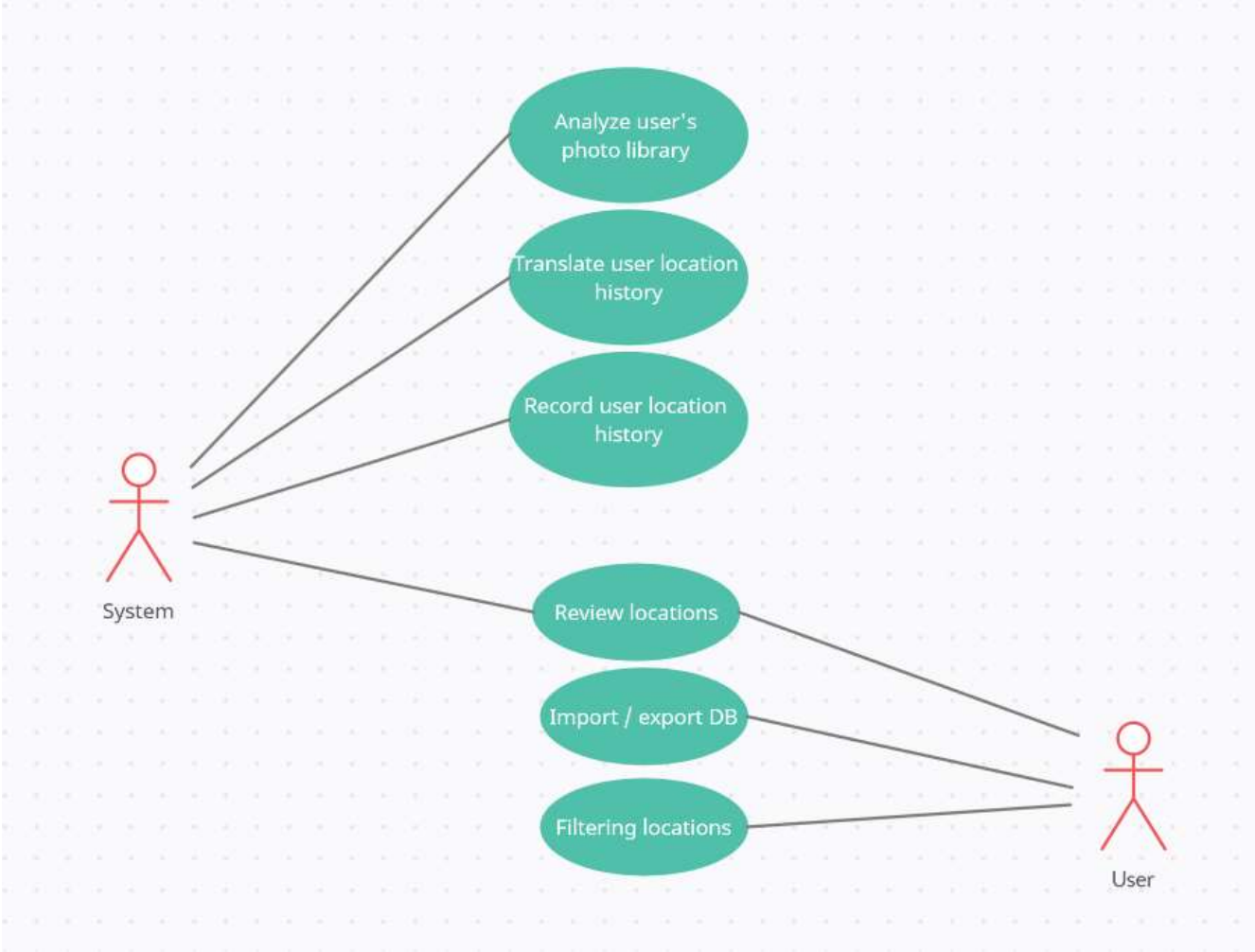

Рисунок 1.3 – Варіанти використання застосунку

Ключові варіанти використання які були визначенні під час моделювання діаграми варіантів використання наведені в таблицях нижче (1.1 – 1.6).

Таблиця 1.1 – Моніторинг пересування користувача

Actors	System
Description	Запис історії пересування користувача
Precondition	- Система підтримує моніторинг локацій за допомогою тріангуляції вишок зв’язку - Застосунку надані відповідні дозволи - Користувач увімкнув дану можливість - Додаток залишається у фоні (iOS 15)
Postcondition	- Історія пересування та візити користувача записана в базу даних - Дані про локації зберігаюсь у зашифрованому режимі

Таблиця 1.2 – Переклад місць із історії користувача

Actors	System
Description	Переклад місць із історії відвіданих локацій користувача до вибраної мови на телефоні
Postcondition	База даних локалізована без втрати попереднього перекладу

Таблиця 1.3 – Моніторинг локацій із бібліотеки фотографій користувача

Actors	System
Description	Аналіз фототеки користувача
Precondition	- Додатку наданий дозвіл на всю фотогалерею або на конкретні вибрані фотографії - Користувач увімкнув дану можливість
Postcondition	- Історія пересування базуючись на фотографіях добавляється в додаток

Таблиця 1.4 – Перегляд локацій користувача на карті

Actors	Користувач
Description	Перегляд локацій користувача на карті
Precondition	- Присутня хоч би одна локація в історії пересування користувача
Extends	- Можливість швидкого сортування (сьогодні, вчора, цей місяць, цей рік, минулий рік і весь час) - Можливість відображати локації тільки із заданою точністю - Можливість показувати специфічні локації (пересування, відвідини, точки із фотографій)

Таблиця 1.5 – Імпорт локацій користувача у заданому форматі

Actors	Користувач
Description	Імпорт бази даних користувача у заданому форматі
Precondition	- Підтримуваний формат бази даних (csv, gpx)
Postcondition	- База даних імпортована і записана у локальний інстанс бази даних не перетираючи існуючі локації

Таблиця 1.6 – Експорт локацій користувача у заданий формат

Actors	Користувач
Description	Експорт бази даних користувача у заданий формат
Precondition	- Присутня хоч би одна локація в історії пересування користувача
Postcondition	- База даних експортована у заданий формат (csv, gpx)

Таблиця 1.6 – Перегляд локацій користувача списком

Actors	Користувач
Description	Перегляд локацій користувача списком із вибраним групуванням
Precondition	- Присутня хоч би одна локація в історії пересування користувача
Extends	- Можливість відображати локації тільки із заданою точністю - Можливість показувати специфічні локації (пересування, відвідини, точки із фотографій) - Можливість видаляти каскадно певні локації (post mvp) - Можливість каскадно переглядати дані із групування по адміністративному рівні

Базуючись на результатах діаграми варіантів використання (рис. 1.3) та описі ключових варіантів використання із заданим описом (таблиці 1.1 – 1.7), передумовами, пост умовами а також можливими розширеннями існуючих варіантів можна визначити ключові пункти при проєктуванні програмної системи, зокрема:

- Доступна версія операційної системи
- Бібліотеки із відкритим кодом
- Локальна SQL подібна база даних
- Зручні інструменти для роботи із локаційними сервісами

1.2 Проєктування програмної системи

1.2.1 Вибір моделі розробки

Якщо по подивимось на керування проєктами то можна з легкістю помітити що вони змінились досить сильно за останні роки. Структури або методології варіюються від Agile до Scrum і Waterfall до Kanban. У той час як фреймворки, такі як Scrum, використовують більш жорсткий, структурований метод, інші, як Kanban, легше впроваджувати та слідувати на додаток до інших існуючих процесів. Усі вони мають свої плюси і мінуси, але спільне їх об’єднує, так це те, що всі вони забезпечують ефективне керування командою та співпрацю на робочому місці.

Agile — це популярна методологія розробки програмного забезпечення, яка допомагає команді співпрацювати для пошуку рішень шляхом безперервної еволюції. Він також включає політику для кращого планування, розробки та своєчасних поставок членами команди. Крім того, ви будете готові до раптових змін і зможете швидко реагувати [7]. Життєвий цикл розробки із використанням Agile наведений на рисунку 1.4.

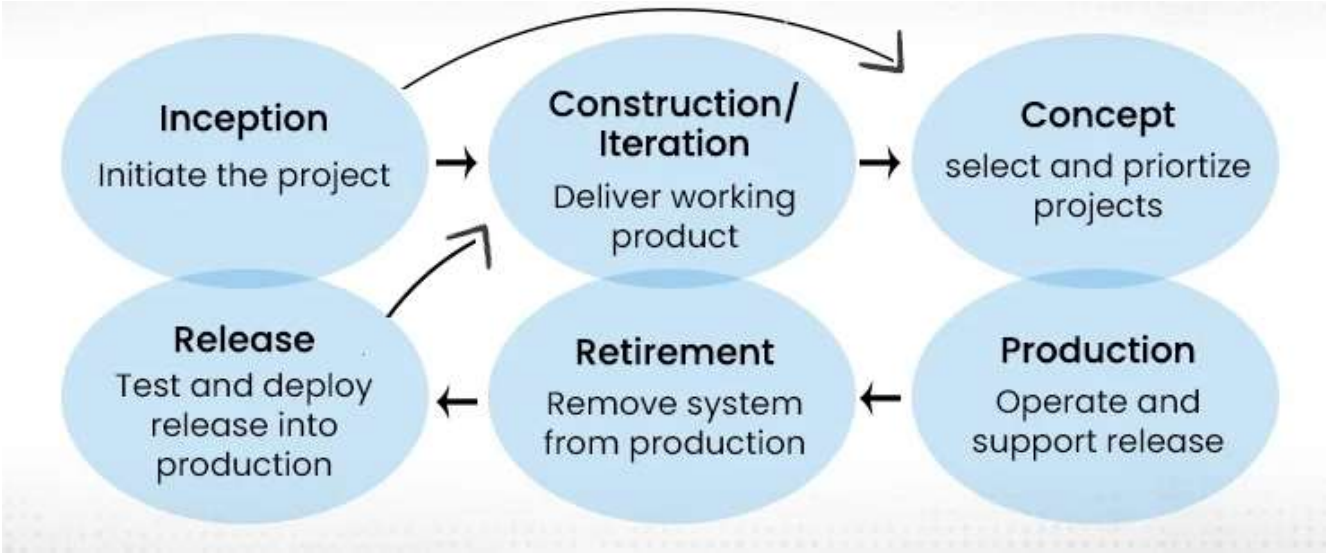

Рисунок 1.4 – Життєвий цикл розробки ПЗ із використанням Agile

Якщо порівнювати Agile серед Scrum, Kanban чи Waterfall, Agile здобув величезну популярність порівняно з іншими платформами. Основна методологія, яка лежить в основі будь-якого Agile-фреймворку, полягає в тому, що проєкти розбиваються на прості “user-stories”, а потім вони групуються по пріоритетах перед послідовним виконанням у черговій ітерації (sprint). Ось основні принципи фреймворку Agile Manifesto, які розроблені, щоб зробити розробку програмного забезпечення більш ефективною та орієнтованою на результат [7]:

- Задоволеність клієнтів і безперервна доставка програмних продуктів
- Будь які зміни до вимог приймаються на ранніх етапах розробки
- Робочі модулі випускаються часто або щотижня, або щомісяця
- Здійснюється щоденна співпраця між клієнтами та розробниками
- Проєкт будується навколо мотивованих людей, яким можна довіряти
- Сталий розвиток можна досягти постійними темпами
- Команди стають більш злагодженими та само організованими, коли вони працюють разом для досягнення єдиної мети

Стосовно переваг які надає Agile то можна відмітити те що він має чудову адаптивність, він є гнучким і легко підлаштовується під потреби замовника і ринку, дає на виході досконалий продукт який проходить перевірки в кінці кожного спринту. Проте при всіх перевагах Agile в нього є також суттєві недоліки, що для деяких клієнтів є проблемою зокрема це відсутність конкретного терміну релізу через постійні зміни і те що потрібно наділяти команду певними повноваженнями для цього. А якщо компанія до цього не мала налаштованої Agile системи це може бути доволі проблематично і дорого трансформуватись.

В якості підсумку можна сказати що Agile це сучасний і зручний формат управління, здатний вирішити проблеми класичного проєктного менеджменту. Тому він високо цінується там, де потрібно створювати інноваційні проєкти. Разом з цим, його концепція ефективна у процесній діяльності (наприклад, кафе швидкого обслуговування). Також він ідеально підходить для розвитку проєктів у сфері IT-технологій, маркетингу, реклами і PR-діяльності [8].

Kanban (рис. 1.5) здійснив революцію в автомобільній промисловості, і він має значний вплив на різні інші сектори, включаючи програмне забезпечення, IT-операції, маркетинг подій тощо. Це ще одна з популярних гнучких фреймворків, розроблених, щоб зробити життєвий цикл проєкту більш раціональним і підвищити ефективну співпрацю команди за рахунок постійного вдосконалення та простоти управління змінами. Але в той же час хочеться сказати що Kanban — це підкатегорія agile-фреймворка [7].

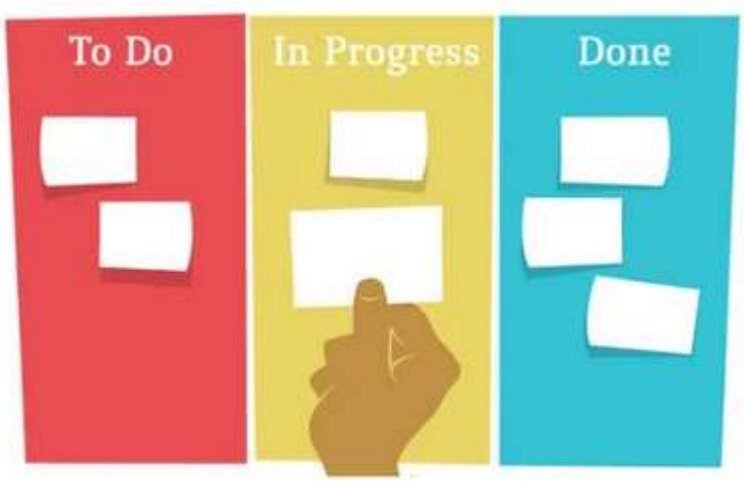

Рисунок 1.5 – Kanban методологія

Основним методом розробки із використанням методології Kanban буде дошка. Дошка Kanban – це інструмент візуалізації, який призначений для візуалізації роботи її обмеження роботи та підвищення ефективності. Дошки зазвичай використовують картки, стовпці та наліпки, щоб повідомити про стан, прогрес і проблеми, що виникають (рис. 1.5). Загалом, дошки Kanban мають 5 компонентів: візуальні сигнали, стовпці, межі виконання, точки зобов’язань і точки доставки [9].

Кожен робочий елемент на дошці Kanban називається карткою Kanban, за допомогою цієї картки команда може візуально відстежувати робоче навантаження, надаючи коротку інформацію про відповідальність, приблизний час виконання та поточний статус робочого елемента.

Як і дошки Scrum, інструменти дошки Kanban сприяють розбиттю роботи на невеликі, керовані частини, щоб дозволити візуалізувати роботу в міру її просування та переміщення по робочому процесу проєкту.

Стосовно переваг які ми отримуємо від використання Kanban це те що вся робота видна на дошці і легко може бути пріоритезована, при даному підході легко впроваджувати зміни але в той час недоліком даної системи є те що немає контролю часу, фактично ми не маємо часових рамок пов'язаних із кожною фазою.

Waterfall (рис. 1.6) є найбільш традиційним підходом до розробки програмного забезпечення, який використовується розробниками програмного забезпечення протягом багатьох років. Її також називають лінійно-послідовною моделлю життєвого циклу. Це була перша модель процесу, яка була представлена. Модель була суттєво структурована, і її було важко адаптувати до змін [7].

Рисунок 1.6 – Розробка за допомогою Waterfall

У випадку з водоспадною моделлю завдання слід виконувати послідовно, перш ніж почати наступний етап. Він просто уникає перекриття різних етапів проєкту і призначений для роботи лише в одному напрямку. Різні етапи моделі Waterfall включають ініціювання, аналіз, проєктування, конструювання, тестування, розгортання та обслуговування [10].

- Requirements Gathering and Analysis - вимоги збираються бізнес-аналітиком і аналізуються командою. На цьому етапі вимоги документуються, і можна отримати роз'яснення.

- System Design - архітектор і старші члени команди працюють над архітектурою програмного забезпечення, дизайном високого та низького рівня для проєкту.
- Implementation - над розробкою проєкту працює команда розробників. Вони беруть проєктну документацію / артефакти та гарантують, що їхнє рішення відповідає проєкту, доопрацьованому архітектором.
- Testing - команда тестування перевіряє повну програму та визначає будь-які дефекти в програмі.
- Deployment - команда створює та встановлює додаток на серверах, публікує збірки до магазинів тощо.
- Maintenance - на етапі технічного обслуговування команда забезпечує безперебійну роботу програми на серверах без простоїв.

Стосовно переваг які ми отримує при використанні Waterfall моделі це те що дана модель розробки дуже добре підходить для маленьких проєктів і де вимоги є чіткі і зрозумілі, проте величезний недолік системи це те що ми не можем легко повернутись до попередньої фази якщо появляються нові вимоги чи зміни.

Отже враховуючи проаналізовану інформацію про різні підходи до розробки для розробки даної системи було вибрано Kanban методологію, вона досить добре впишеться в рамки і масштаби проєкту і дозволить корекнтно контролювати процес розробки, закінчуючи задачі в терміни або пріоритезовуючи певні моменти при необхідності.

1.2.2 Вибір архітектурного патерну

Станом на 2021 рік існує безліч підходів до побудови застосунків під iOS, починаючи від стандартного MVC який пропонує компанія Apple, MVP який пропонує більшу гнучкість в побудові інтерфейсів і залежностей а також MVVM який пропонує інструменти із використанням реактивного програмування.

Роберт Мартін в простонародді відомий більше як Дядько Боб опублікував статтю “Screaming Architecture”, де як він описує архітектура застосунку повинна “кричати” про самий додаток, а не про те які фреймворки в ньому використовуються [11]. Безпосередньо вже в 2012 році він випускає продовження своєї статті яку називає “Clean Architecture” де вже описує застосування підходу [12]. Роберт Мартін наводить пропозицію чистої архітектури (рис. 1.7) [12]:

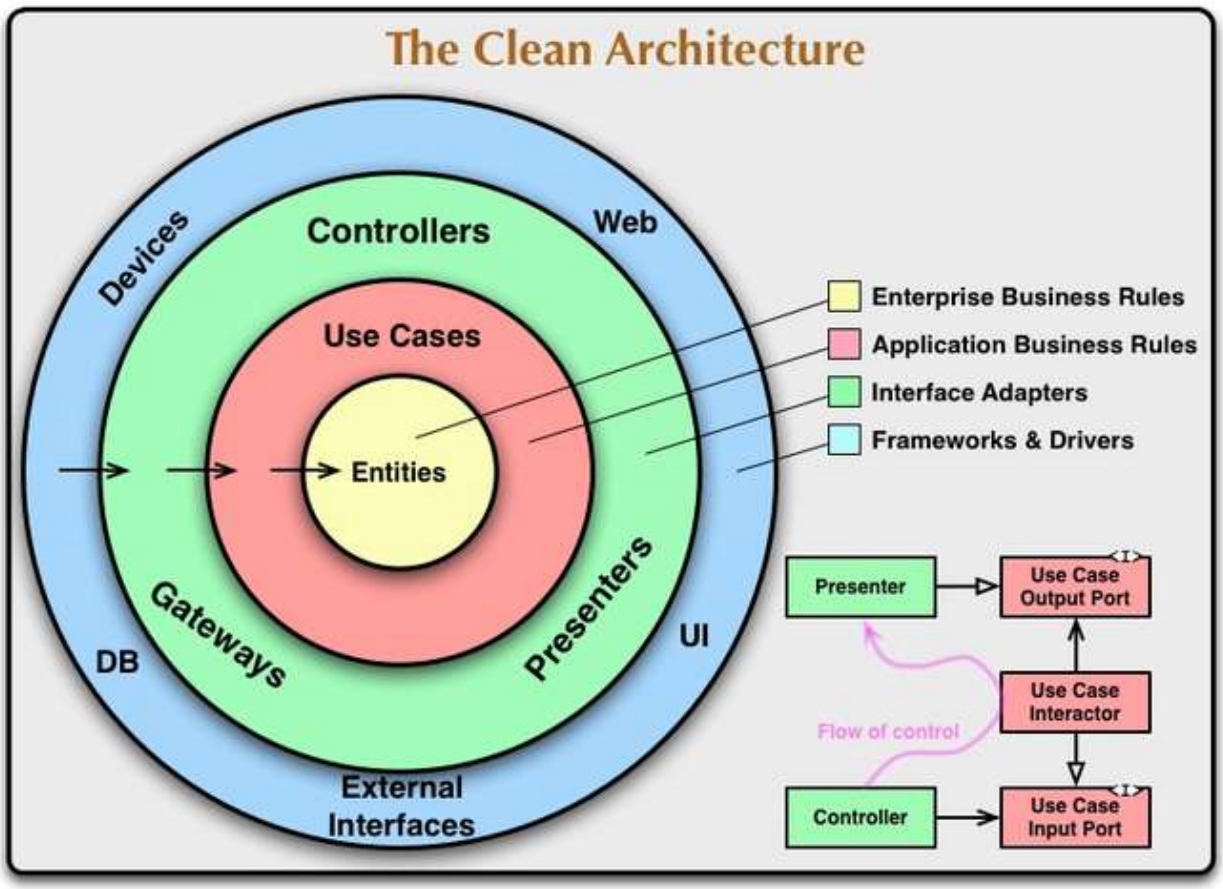

Рисунок 1.7 – Чиста архітектура Роберта Мартіна

Загалом Роберт Мартін наводить приклади і каже що Clean Architecture містить в собі декілька інших архітектурних підходів, але всі вони включати Чисту Архітектуру зводяться до наступних принципів [12]:

- Незалежність Фреймворків і бібліотек – архітектура не залежить від існуючих бібліотек в проєкті, що дозволяє останні використовувати як інструменти а не як вузькі залежності
- Тестованість – бізнес шар може легко бути протестований без користувацького інтерфейсу, бази даних чи веб сервера або будь якого стороннього інструменту
- Незалежність UI – зміна користувацького інтерфейсу відбувається доволі легко, без зміни іншої системи, для прикладу ми можем легко замінити веб інтерфейс з консольним
- Незалежність бази даних – можна легко змінити імплементацію бази даних для прикладу із SQLite на Realm
- Незалежність від зовнішніх чинників – бізнес шар не знає нічого про зовнішні фактори, бібліотеки чи інструменти

Якщо подивитись на оригінальну схему (рис. 1.7) можна легко помітити “Dependency Rule” цей принцип каже що внутрішні слої не залежать від зовнішніх, те ні в якому разі база даних не повинна знати нічого про UI систему, це є одне з чітких правил якому повинен слідувати програміст архітектор при проєктуванні майбутньої системи.

На рисунку 1.8 наведено приклад яким чином буде відбувати обробка вводу користувача, червоними стрілками наведено те як буде проходити запит, пунктирними лініями наведено границі слоїв. Коли користувач ініціює подію (UI) подальша обробка переходить безпосередньо в Presenter, останній добавляючи необхідні обробники і модифікує запит при потребі передає його в конкретний UseCase, який повинен значи що робити із запитом далі і до якого репозиторію звернутись, репозиторій в свою чергу вибирає конкретного доставщика даних і

відсилає йому запит, коли постачальник даних відповідає весь запит проходить потік назад точно слідуючи тому як запит був надісланий.

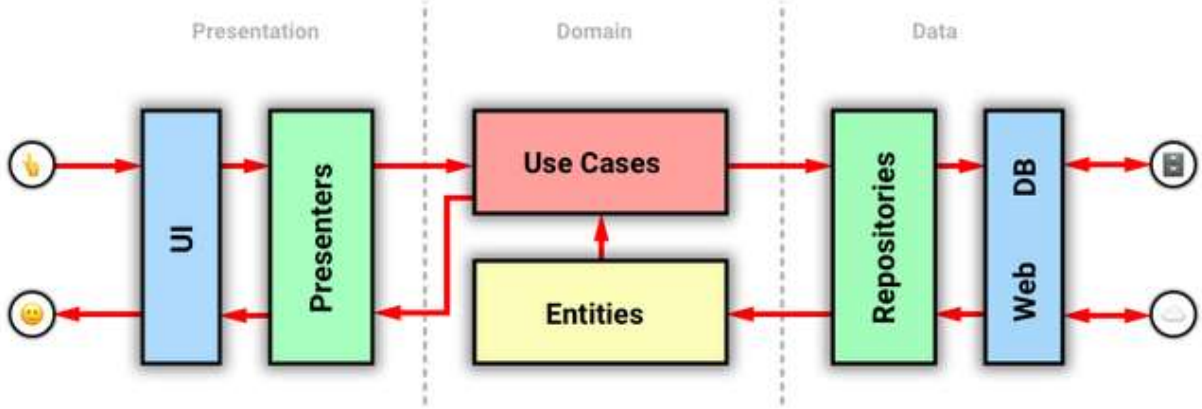

Рисунок 1.8 – Течія запиту у чистій архітектурі

На рисунку 1.9 наведений приклад із лекції Роберта Мартіна [14] можна побачити схему яку він пропонує для чистої архітектури, подвійними лініями в даному випадку зображені границі шарів, стрілками залежності між класами.

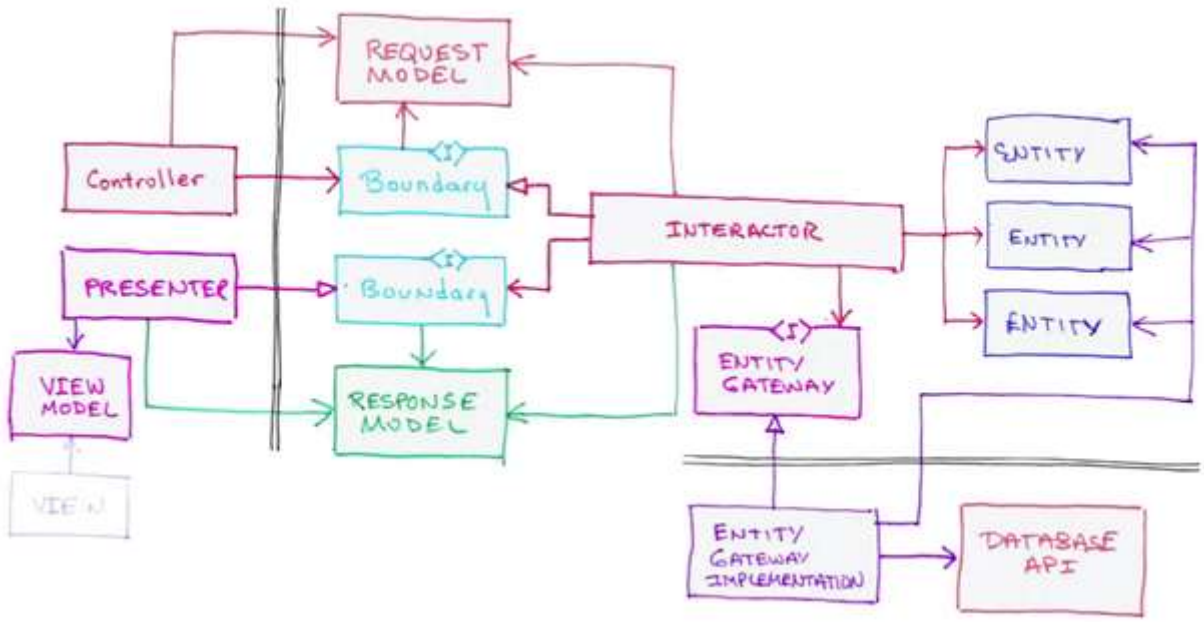

Рисунок 1.9 – Схема класів і залежностей в чистій архітектурі

Самі по собі об’єкти Entity (рис 1.9) не є об’єктами DTO, вони лише інкапсульовують логіку бізнесу, можна сказати що вони представлять те що не залежить від конкретного застосунку а буде спільним для багатьох. В той же час шар Entity містить безпосередньо сутності (Entity) які є функціями або об’єктами з методами які реалізовують логіку бізнесу загальну для багатьох додатків і також DTO об’єкти які є необхідними для переходів між шарами [13].

Interactor як його описує Роберт Мартін це я об’єкт який імплементовує Use Case через виклик бізнес об’єктів (Entity) [14]. Існує декілька способів реалізації Interactor одна з них передбачає те що ми будем мати один Interactor який міститиме набір UseCase, але в нашому випадку це не є зовсім коректно. В випадку Clean Architecture краще мати один Interactor для імплементації одного Use Case, даний підхід як не як дасть нам більше гнучкості.

Обов’язковим компонентом системи є також шар репозиторію (Repository), фактично він скриває від користувача процес отримання даних, для реалізації репозиторію можна використовувати будь який паттерн. В контексті розробки під мобільні пристрої шар репозиторію часто реалізовують як об’єкт який надає доступ до даних із можливістю вибору джерела. Як згадується в описі проблем Чистої Архітектури [13] доступ до Repository повинен бути в більшості випадків через Interactor для збереження гнучкості системи і її можливості тестування, проте якщо система проста і не передбачається обробки логіки можна легко опустити шар Interactor для спрощення системи, при чому правилом Dependency Rule це не забороняється. При цьому при потребі завжди можна добавити Interactor, і це є хорошим компромісійним рішенням між проксі інтеракторами і прямими викликами репозиторію.

Досить важливим пунктом в архітектурі також є робота з об’єктами і їхнє перетворення при передачі між шарами. Є два потенційних підходи:

- використовувати DTO на кожному шарі – це надає нам гнучкість системи оскільки зміни в одному шарі не будуть зачіпляти зміни в іншому, але це спричиняє дублікати коду що не є добре, і ще одним суттєвим недоліком є те що при зміні даних так чи інакше потрібно буде міняти необхідні “мапери”

- використанням DTO із шару Entity – має ряд суттєвих переваг ніж використання DTO на кожному шарі, зоокрема це немає дублювання коду, менше роботи але має один недолік який проявляється при зміні DTO, це те що нам потенційно можливо треба буде міняти код в інших шарах

В якості покращення і спрощення оригінальної архітектури яка була запропонована Робертом Мартіном пропонується включити реактивну частину як заміну In\Out об’єктам між Presenter та Interactor, високо рівневий варіант архітектури який використовується в застосунку MLTracker наведено на рис 1.10. Здавалось би проста заміну In\Out об’єктів на об’єкт Observable із бібліотеки RxSwift не вплине суттєво на кількість коду, але на практиці це показало досить непоганий результат.

1.2.3 Побудова UML діаграми класів

Для коректної імплементації ядра системи нам необхідно змоделювати основні класи для застосунку. При проєктуванні була звернута особлива увага на наступні частини коду

- Локаційні сервіси – так як застосунок використовуватиме активно геопозиції користувача, це є доволі важлива частина системи яку необхідно спроєктувати доволі детально
- Сервіси перекладу – якщо користувач змінює мову необхідно буде провести переклад існуючої бази даних, відповідно потрібно добре продумати яким чином це буде відбуватись
- Міграційні служби SQLite бази даних – враховуючи що весь користувацький досвід будуватиметься на читанні і запис в локальну базу даних необхідно продумати коректні взаємозв’язки між класами які відповідатимуть за міграцію між схемами БД і потенційні великі міграцію між базами

- Систему групування по адміністративних рівнях – так як це передбачається основна система для перегляду історії пересування необхідно продумати систему яка наддасть нам гнучку можливість для перегляду із фільтруванням
- Систему із плагінами для імпорту / експорту даних із БД – враховуючи що кількість і потенційні формати можуть мінятись потрібно продумати хорошу організацію яким чином буде відбуватись вибір плагіну і його можливі версії і зміни
- Глобальна фільтраційна система – це система яка наддасть велику гнучкість у фільтрації даних на глобальному рівні

Розглянемо залежності та співвідношення основних компонентів системи застосунку (рис. 1.10). Як можемо бачити із рисунку на верхньому рівні розташовані компоненти ядра такі як база даних, інструменти локаційних сервісів чи хмарне сховище (потенційно). Зі всіма цими даними працює бізнес шар, який конвертує дані ядра в моделі шару Entities, це забезпечить шар графічного інтерфейсу та презентів від неочікуваних внутрішніх змін. Безпосереднє завдання шару презентера це підготувати дані для графічного інтерфейсу користувача.

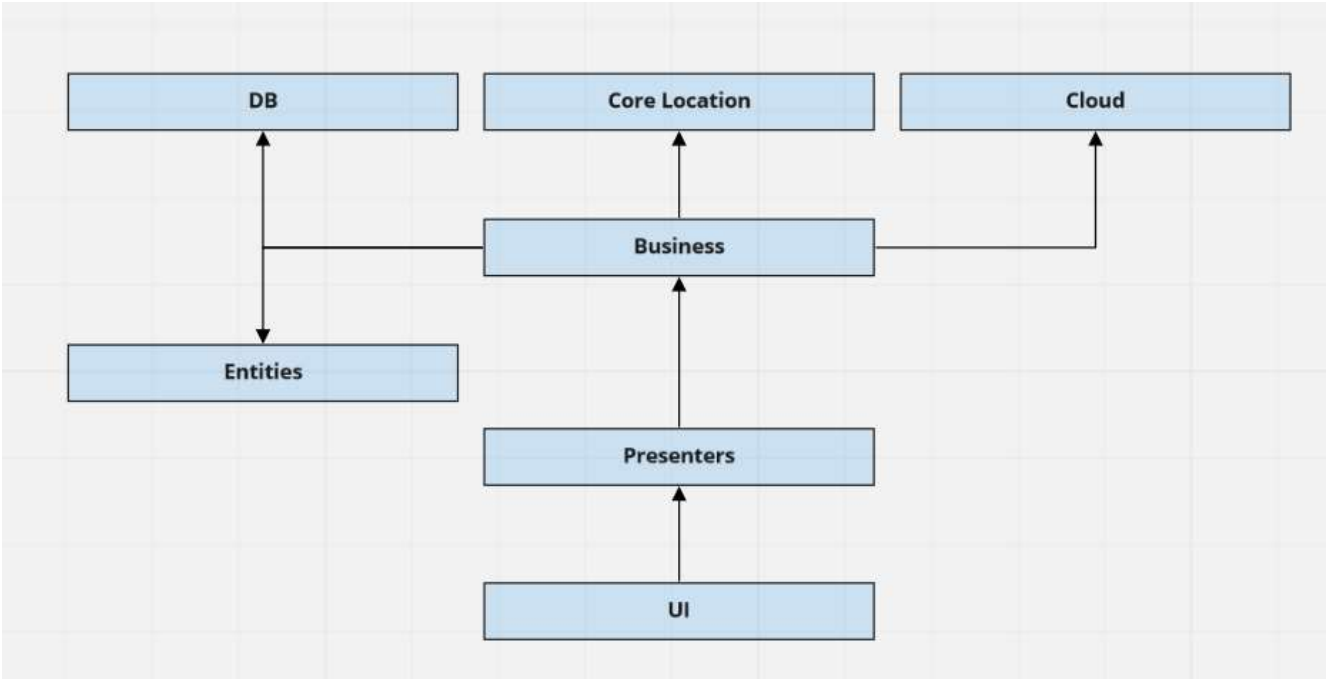

Рисунок 1.10 – Співвідношення основних систем застосунку

На рисунку 1.11 наведено співвідношення компонент міграційних служб бази даних SQLite. Основним центральним компонентом є SQLiteManager, через нього здійснюються всі виклики залежних служб, такі як міграція, підготовка бази до локалізації тощо.

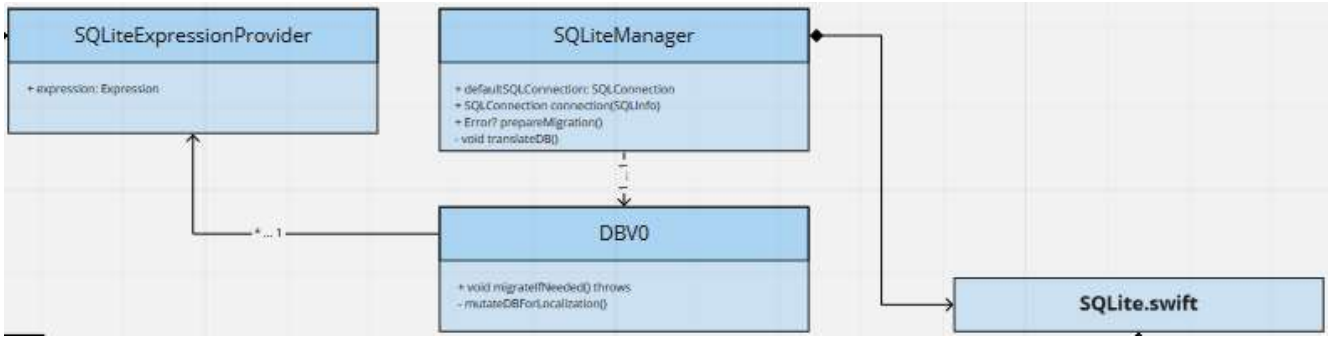

Рисунок 1.11 – Діаграма служб SQLite

На рисунку 1.12 наведено співвідношення бази даних із бізнес шаром застосунку. Кожна таблиця або логічний запит представлено як окрему модель, яка має надати ключі і типи полів за допомогою структури SQLiteExpressionProvider. Уся робота з базою даних повинна відбуватись виключно через репозиторії, в додатку передбачено на початковій стадії два основних SFRequestReposity та SFLocationGroupRepository, обидва надають певні методи для імплементації методів CRUD до даних в базі даних. SFFilterRepository виступає класом який зберігає дані сторінки із фільтрами і надає потім фільтраційні вирази іншим репозиторіям для побудови коректних SQL-запитів. Усі репозиторії мають бути реалізовані таким чином щоби не зберігати внутрішні стани в собі а виступати тільки посередником (фасадом) до доступу в базу даних. Враховуючи дану вимогу вони можуть бути представлені як обєкти-одинаки (singleton) і зберігатись в глобальному контейнері сервісів. Відповідно різні конкретні інтерактори певних модулів(сторінок) зможуть дістати точно один екземпляр певного репозиторію. Хочеться відмітити що основні методи (ті які вертають Observable<T>) мають бути асинхронними через використання системної структури планувальника потоків.

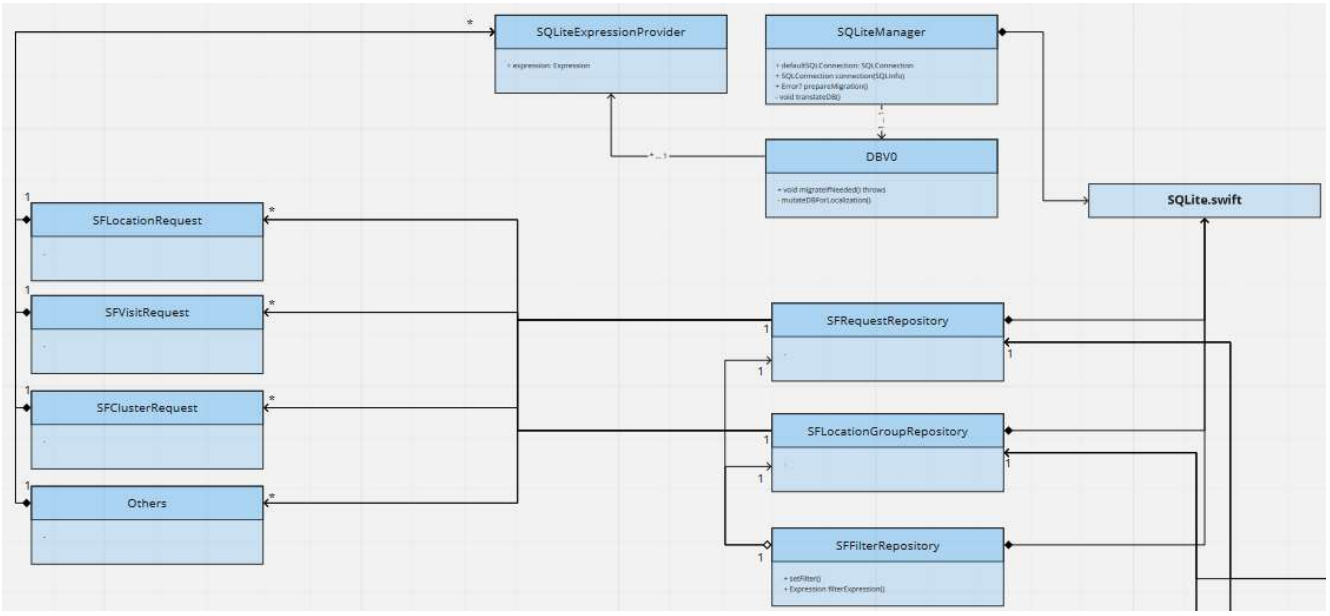

Рисунок 1.12 – Діаграма об’єктів бази даних та бізнес шару

На рисунку 1.13 наведено основні класи сервісів, які відповідальні за служби локацій, їхню взаємодію із різноманітними системами геокодингу (в початковій версії застосунку передбачається використання тільки об’єктів CoreLocation) таких як CL (Core Location) та OSM (Open Street Map).

Система сервісів поділяється на системі на сервіси застосунку. До сервісних зокрема відноситься LocationService даних сервіс призначений для роботи із API мобільної системи для доступу до геолокацій користувача, відповідальний за взяття дозволу на використанням даних GPS і результатів тріангуляції сотових вишок.

SFLocationServie – сервіс який відповідальний за проведення геокодингу бази даних використовуючи різноманітні надані йому стратегії (геокодинг за допомогою служб Core Location чи публічного API Open Street Map). Також у відповідальність даного сервісу входить обов’язок проведення перекладу бази даних якщо мова була змінена користувачем в системних налаштуваннях телефону.

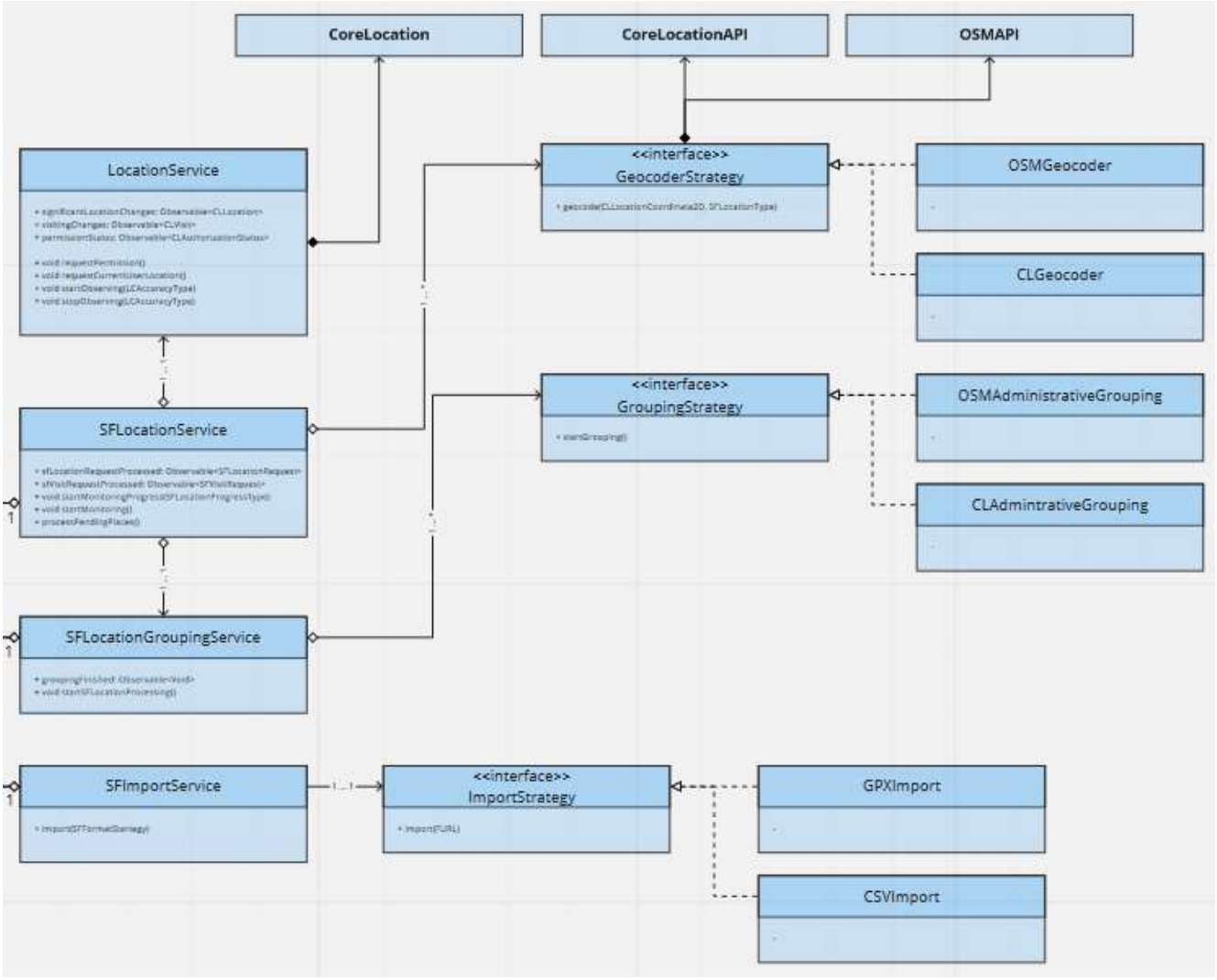

Рисунок 1.13 – Діаграма основних сервісів застосунку

SFLocationGroupingService – сервіс який відповідальний щоби згрупувати користувацькі локації згідно наданих стратегій. Зокрема на даному етапі це передбачається групування по адміністративних рівнях. Країна – область – регіон – локація – під локація (даний вид локацій характерний для англомовних країн).

Відповідним чином передбачається реалізація служб імпорту / експорту бази даних (або певних даних). Буде відповідальний сервіс який міститиме методи import / export який прийматиме структуру яка міститиме реалізацію імпорту / експорту у специфічний формат із певним способом.

Повна діаграма класів наведена в додатку А.

2 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

2.1 Вибір інструментів розробки

Для коректної і зручної розробки досить важливо вибрати коректну мову програмування, від цього заложитиме майбутнє проєкту. Неправильний вибір може призвести до провалу розробки. Сьогодні компанія Apple пропонує дві мови у створенні додатків для пристроїв: Swift та Objective-c.

Objective-C був створений у 1980-х роках Бредом Коксом і Томом Лавом на основі мови програмування C і парадигм Smalltalk. Мова Objective-C є надмножиною мови C, тому код C повністю зрозумілий компілятору Objective-C. Метою Cox and Love було вирішення проблеми повторного використання коду, щоб зменшити вимоги до системних ресурсів та покращити якість та продуктивність коду. Поріг для вступу в об'єктно-орієнтоване програмування досить високий, і для новачків це стає справжньою проблемою. У 1988 році Objective-C була ліцензована NeXT. AppKit і Foundation Kit також були додані до мови, а її бібліотеки були покращені. У 1996 році Apple купила NeXT Software, і середовище розробки NEXTSTEP стало основною технологією розробки майбутньої основної версії операційної системи Apple - OS X. Після придбання NeXT Apple взяла за основу їх SDK (компілятор, бібліотеки, IDE). для їх подальшого розвитку. IDE для розробки коду називається Xcode, а графічний інтерфейс користувача (Graphical User Interface) називається Interface Builder. До 2014 року всі нативні програми iOS створювалися за допомогою Objective-C [15].

До випуску Swift Objective-C займала 7 місце в списку найпопулярніших мов програмування (рис 2.1). Деякі власні програми все ще створюються за допомогою Objective-C, хоча Swift став більш популярним для цього. Objective-C перетворився з лідера в розробці додатків iOS в інструмент для підтримки існуючих продуктів.

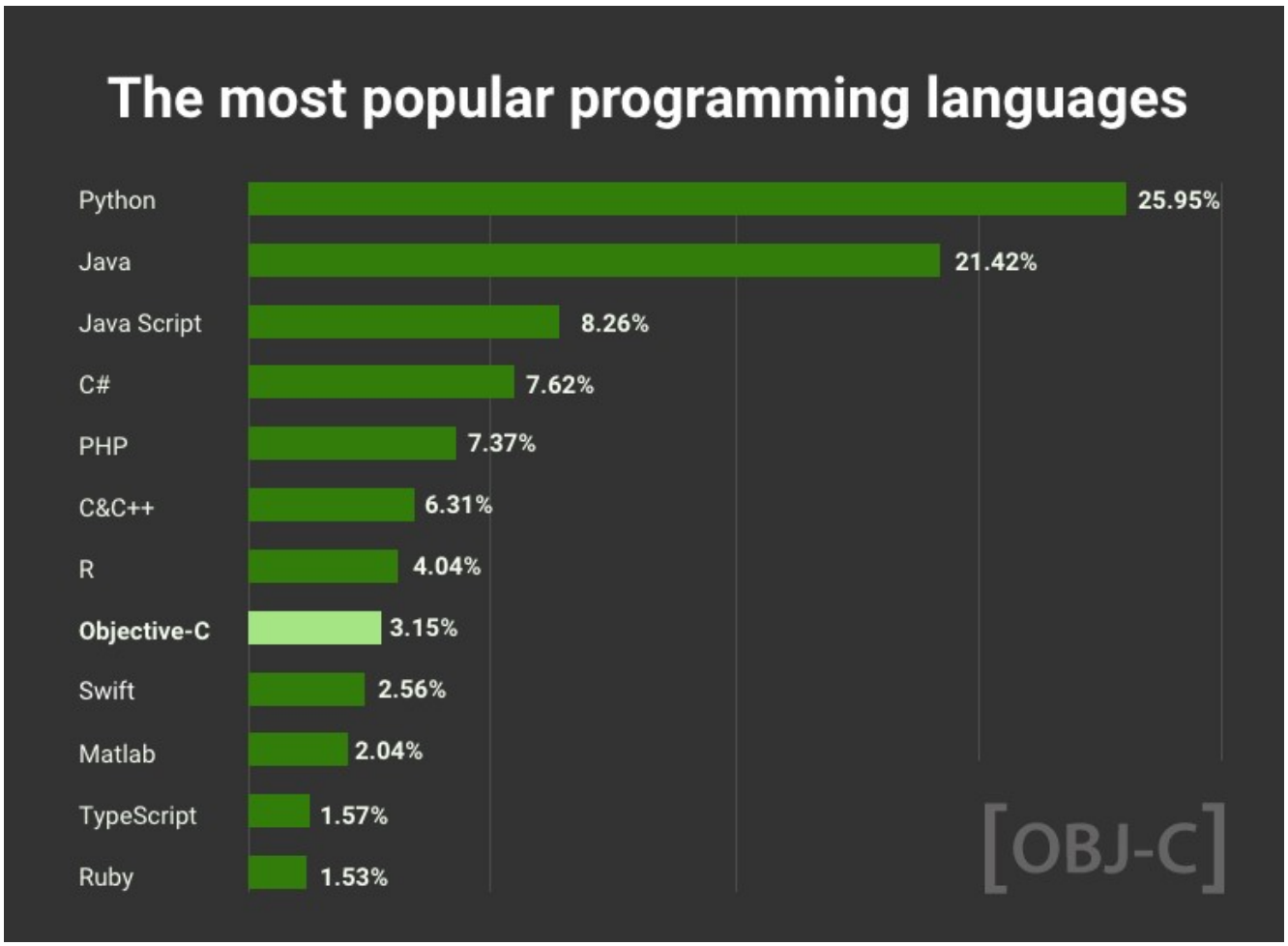

Рисунок 2.1 – Найбільш популярні мови програмування в світі

До основних переваг Objective-c можна віднести:

- Динамічність і адаптивність
- Стабільність і зрілість
- Сумісність із C та C++ бібліотеками
- Добре відлагоджена мова програмування

Недоліками Objective-c є:

- Складний синтаксис конструкцій
- Відлагодження складне і неявне
- Сумнівне майбутнє

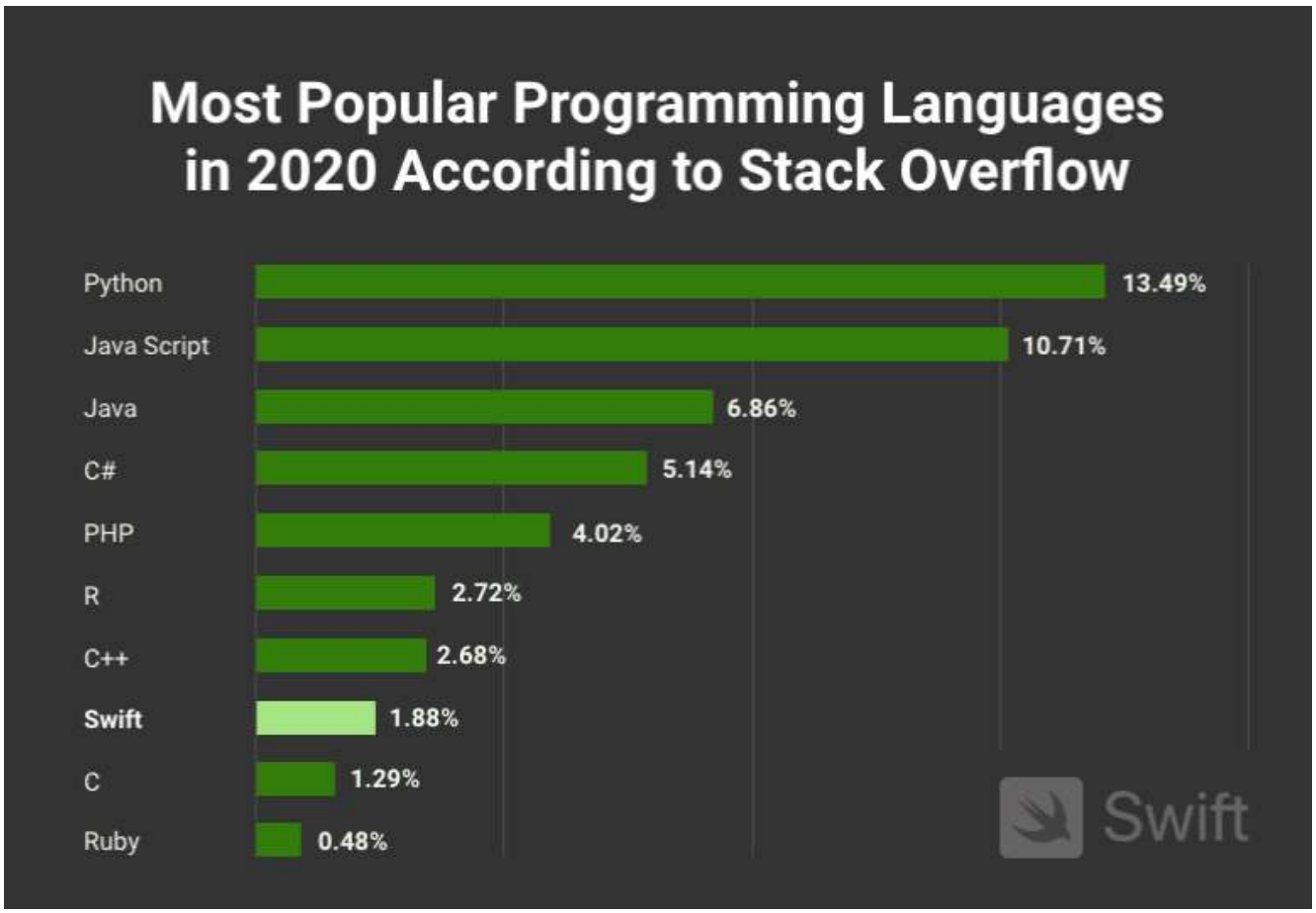

Рисунок 2.2 – Популярність мов програмування в 2020 році

Мова програмування Swift почала свій розвиток у 2010 Крісом Латтнером. Більше півтора року він тримав у секреті свій проєкт, розвиваючи його у вільний час. Тоді Латтнер вирішив поділитися своїм винаходом з вищим керівництвом Apple, яке високо оцінило цю ідею. Після цього Латтнер отримав пару розробників, які допомогли йому офіційно продовжити роботу над проєктом. Проте проєкт все ще залишався секретною розробкою. Але поки секретна розробка тривала, проєкт Латнера потрапив до списку пріоритетних напрямків Apple. Коли вона вийшла, люди, які опосередковано над нею працювали, були значно здивовані, чим насправді виявилася ця ідея. Перший випуск Swift відбувся в 2014 році. Його позиціонували як швидшу та ефективнішу мову програмування для створення додатків iOS та macOS. Вона стала найшвидше розвиваючою мовою програмування в історії. На відміну від Objective-C, Swift був створений для середнього розробника. Він простий у освоєнні та має синтаксис, який дозволяє

програмістам самостійно навчатися. У 2015 році Apple зробила Swift відкритим вихідним кодом для всіх зацікавлених. Тепер будь-який розробник програмного забезпечення, студент коледжу чи новачок може розробити та оптимізувати цю мову. У 2020 році Swift увійшов до десятки найбільш затребуваних мов програмування (рис. 2.2) [15].

Переваги мови програмування Swift

- Простота
- Безпечність типів
- Просунуті конструкції
- Висока швидкість розробки
- Можливість розробки під різні платформи
- Відкритий код

Недоліки мови програмування Swift

- Постійні зміни
- Немає підтримки ранніх версій
- Відсутність підтримки C++ import

Враховуючи наведені переваги та недоліки обох мов програмування вибір було зроблено на користь Swift. Як стверджує компанія Apple Swift майже у 3 рази має більшу швидкість ніж Objective-c мова [16]. Типізація мови Swift є статична що робить неможливим помилки із типами. Хочеться відмітити те що незважаючи що з часу випуску пройшло майже 7 років Swift продовжує активно розвиватись і підтримуватись компанією, на даний момент вже випущено офіційно 5 версію мови і активно готується вже 6 версія, тому у випадку підтримки чи виправлення недоліків Swift в порівнянні із Objective-c є фаворитом.

2.2 Вибір СУДБ та способі її локалізації

В якості основної системи СУБД було обрано SQLite, це є доволі популярною базою даних в мобільній розробці, зокрема це те що вона є підвидом SQL що відповідно надає практично всі потужності серверної версії, але мобільна версія вона працює як правило тільки з одним файлом на диску. База даних виявилась доволі популярною тому для неї написали багато ORM зокрема таких як CoreData.

База даних SQLite створює лише один файл на дисковій системі, що дозволяє переносити файл іншої бази даних. Деякі компанії використовують у своїх програмах більше однієї бази даних, тому дуже важливо, щоб база даних могла бути перенсена. SQLite також можна використовувати для нативних і крос-платформних додатків [17].

До основних переваг використання SQLite можна віднести:

- SQLite використовує SQL, тому має всі функції стандартної бази даних SQL.
- Деяким розробникам потрібні бази даних, які можуть масштабуватися та забезпечувати підтримку паралельності. SQLite, з його багатими функціями, може бути пов’язаний з будь-яким додатком у виробництві.
- Часто розробникам важко проводити тестування, коли база даних програми брудна. SQLite дуже добре підходить для тестування.
- Нульова конфігурація: SQLite не потребує складного налаштування для зберігання даних. Коли ви створюєте власні програми за допомогою Java, вони інтегруються з платформою.
- Розробники називають SQLite, базою даних без сервера, і вона справді виправдовує очікування. Вам не потрібно налаштовувати API чи встановлювати бібліотеку для доступу до даних із SQLite.
- SQLite є кросплатформним, що означає, що його можна використовувати в додатку iOS, створеному на Swift, а також у крос-платформному додатку, створеному на React Native.

Проте основним недоліком використання SQLite є відсутність керування користувачами. Будь-який користувач може читати/записувати дані без спеціального доступу. Будь-яка діяльність або процес у вашій програмі може мати прямий доступ до збережених даних. Безпека є великою проблемою в SQLite. Збережені дані можна легко ввести в будь-який час.

Загалом SQLite цілком задовольняє вимогам додатку і в якості бази даних її було і обрано. Відповідно коли база даних була обрано залишалась ще одна проблема яку необхідно було вирішити – це проблема локалізації та інтернаціоналізації БД. Це робиться із розрахунку на те що користувач може змінювати локаль телефону, чи відповідні налаштування в додатку.

Таблиці бази даних можуть містити текстові або графічні дані, які слід локалізувати. Наприклад, може бути таблиця продуктів із полем опису, що містить англійський текст. Якщо ви використовуєте базу даних з німецькою програмою, текст опису має бути перекладений німецькою мовою. В даному випадку ми використовуєм локалізацію бази даних.

Розглянемо варіанти і методи як можна локалізувати базу даних:

- Файлова локалізація – метод який створює локалізовані файли бази даних, які мають ідентичну структуру та дані, що й вихідна база даних, за винятком значень вибраних полів, які були перекладені.
- Рядкова локалізація – метод додає рядки варіантів мови. У результаті кожен рядок копіюється один раз для кожної мови.
- Локалізація по полю - заповнює специфічні для мови поля.
- Таблична локалізація - заповнює поля таблиць для певної мови.

Файлова локалізація це метод що передбачає створення локалізованих файлів бази даних для кожної мови. Перевага цього методу полягає в тому, що він зовсім не вимагає зміни структури таблиці. Недоліком є те, що він працює на локальних базах даних, які зберігають бази даних в одному файлі, таких як Access, SQL Server Compact і SQLite.

Id	Name	FieldPlayers
0	Soccer	10
1	Ice hockey	5
2	Basketball	5

Id	Name	FieldPlayers
0	Fußball	10
1	Eishockey	5
2	Basketball	5

Id	Name	FieldPlayers
0	サッカー	10
1	アイスホッケー	5
2	バスケットボール	5

Рисунок 2.3 – Файлова локалізація бази даних

Як можемо бачити на рисунку 2.3 кожна версія бази дани містить ідентичні поля як ориганільна англійська. Німецька версія бази даних має таку саму структуру, але текст у полі Ім’я перекладено німецькою мовою так само як і японський варіант.

Ще одним варіантом локалізації бази даних є рядкова локалізація (рис. 2.4). Вона є користною якщо неможливо створити локалізований файл бази даних або щоб локалізовані значення були в одній таблиці з оригіналом, в такому випадку можна використовувати локалізацію рядків. Основна ідея полягає в тому що до таблиці додається певний рядок (language) і від індикує що за рядок і якій мові відповідає [18]. Як правило у випадку використання рядкової локлазації використовується шаблон створення таблиці (лістинг 2.1)

Лістинг 2.1 – Рядкова локалізація

```
CREATE TABLE TableName
(
 Id INTEGER NOT NULL,
 Language VARCHAR(10) NOT NULL,
 ... fields ...
 PRIMARY KEY(Id, Language)
);
```

У випадку якщо ми не хочем мати комбінований первинний ключ, ми можем використовувати трохи змінену структуру. Тут поле RowId містить унікальне значення і також є первинним ключем. Крім того, у нас є ще два поля: Id і Language, які визначають рядок і мову рядка. Значення Id таке саме, що й у полі Id вихідної таблиці (лістинг 2.2).

Лістинг 2.2 – Рядкова локалізація із мовою

```
CREATE TABLE Table
(
 Id INTEGER NOT NULL,
 ItemId INTEGER NOT NULL,
 Language VARCHAR(10) NOT NULL,
 ... fields ...,
 PRIMARY KEY(Id)
);
```

Перевага локалізації рядків полягає в тому, що не потрібно змінювати структуру, якщо ми додаєм нову мову. Недоліком є те, що все множиться, а розмір бази безпосередньо пов'язаний з кількістю мов. Крім того, якщо ви оновлюєте базу даних під час виконання, ви повинні виконувати ті самі оновлення для кожного рядка мови. Це ускладнює модифікацію таблиці. Однак якщо ми тільки читаєм таблицю, то локалізація рядків є дуже хорошим рішенням.

Id	Language	Name	FieldPlayers
0	en	Soccer	10
0	de	Fußball	10
0	ja	サッカー	10
1	en	Ice hockey	5
1	de	Eishockey	5
1	ja	アイスホッケー	5
2	en	Basketball	5
2	de	Basketball	5
2	ja	バスケットボール	5

Рисунок 2.4 – Рядкова локалізація бази даних

Локалізація по полю (рис. 2.5) може бути корисна у випадку якщо ми не хочем мати кілька рядків, але хочем мати один рядок для кожного елемента, в такому випадку ми можем використовувати локалізацію по полю (лістинг 2.3) [18].

Лістинг 2.3 – Стовпчикова локалізація

```
CREATE TABLE Table
(
 Id INTEGER NOT NULL,
 Name VARCHAR(50) NOT NULL,
 Name_de VARCHAR(50) NOT NULL,
 Name_ja VARCHAR(50) NOT NULL,
 PRIMARY KEY(Id)
);
```

Id	Name	Name_de	Name_ja	FieldPlayers
0	Soccer	Fußball	サッカー	10
1	Ice hockey	Eishockey	アイスホッケー	5
2	Basketball	Basketball	バスケットボール	5

Рисунок 2.5 – Локалізація по полю

Перевага даної локалізації поля є в тому, що у нас є лише один рядок для кожного елемента. Легше змінити таблицю. Недоліком є те, що вам доведеться змінити структуру (наприклад, додати поле варіанта мови), а також змінити SQL, який ми використовуєм для доступу до даних.

Таблична локалізація (рис. 2.6) – цей метод подібний до локалізації по полю, але замість того, щоб додавати поля варіантів мови до вихідної таблиці, ми додаємо їх у таблицю варіантів мови. Якщо ми хочеи локалізувати таблицю німецькою та японською мовами, на необхідно додати локалізовану таблицю для кожної мови. Ці таблиці містять не всі поля, а лише ті поля, які потрібно локалізувати (наприклад, Ім’я). Різницю запитів для створення наведено нижче (лістинг 2.4).

Лістинг 2.4 – Таблична локалізація

```
CREATE TABLE Table_de
(
 Id INTEGER NOT NULL,
 Name VARCHAR(50) NOT NULL,
 PRIMARY KEY(Id)
);
CREATE TABLE Table_ja
(
 Id INTEGER NOT NULL,
 Name VARCHAR(50) NOT NULL,
 PRIMARY KEY(Id)
);
```

Id	Name
0	Fußball
1	Eishockey
2	Basketball

Id	Name
0	サッカー
1	アイスホッケー
2	バスケットボール

Рисунок 2.6 – Локалізація по таблиці

Перевага табличної в тому, що нам не потрібно змінювати вихідну таблицю. Недоліком є те, що ми повинні додавати нову таблицю щоразу, коли ми додаєм нову мову. Крім того, доступ до даних ускладнюється, тому що ми повинні читати дані як з оригінальної, так і з локалізованої таблиці.

Загалом кожен метод свої переваги і недоліки, результати аналізу згідно джерела були наведені на рисунку 2.7 [18].

Feature	Create localized database files	Row localization	Field localization	Table localization
Works with any database	-	yes	yes	yes
A database can be localized without changing the data structure	yes	-	-	-
Can be implemented without redundant data	-	-	yes	yes
New languages can be added without changing the data structure	yes	yes	-	-
Amount of SQL statement change you have to do in your code	minimal	little	more	more

Рисунок 2.7 – Порівняльні методи різних методів локалізаці БД

2.3 Реалізація основних класів та методів

Для створення передбачуваного функціоналу був створений проєкт Xcode із таргетом 13 iOS. Вибір 13 iOS був зумовлений тим що необхідно було отримати доступ до бібліотеки стандартних зображень. Також для того щоби контролювати версії сторонніх бібліотек було використано CocoaPods. Список використаних сторонніх бібліотек наведено нижче в лістингу 2.5.

Лістинг 2.5 – Podfile проєкту застосунку

```
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '11.0'
use_frameworks!

def shared_pods
  pod 'STT/Core', :path => '../STT' # '~> 3.0.0-rc.3'
  pod 'STT/Extensions', :path => '../STT' # '~> 3.0.0-rc.3'
  pod 'STT/Bindings', :path => '../STT' # '~> 3.0.0-rc.3'
  pod 'STT/RxExtensions', :path => '../STT' # '~> 3.0.0-rc.3'
  pod 'STT/NotificationBanner', :path => '../STT' # '~> 3.0.0-rc.3'

  pod 'XCGLogger', '~> 7.0.1'

  pod 'NGSRouter', :path => '../../NGS/NGSRouter' # "~> 1.0.0"
  pod 'Swinject'

  pod 'TinyConstraints'
  pod 'Cluster'
  pod 'CoreGPX'

  pod 'SQLite.swift'

  pod 'R.swift'
end
```

У проєкті було створено дві схеми побудови додату, одна для дебаг збірок інша для збірок для продакшену. Остання схема призначеня для тествої схеми із використанням юніт-тестів (лістинг 2.6).

Лістинг 2.6 – Схеми в залежностях проєкту

```
target 'MLTracker' do
 shared_pods
end

target 'MLTracker DEV' do
 shared_pods
end

target 'MLTracker DEVTests' do
 shared_pods
end
```

Розглянемо приклад як реалізовувався основний функціонал на прикладі LocationService.swift. Даний сервіс є доволі важливим оскільки він запитує у користувача дозволи. Запускає якщо увімкнені служби гео-моніторингу локацій за допомогою CoreLocation та повідомляє про виникнення помилок у сервісі.

Користувач може ініціювати гео-стеження за ним за допомогою GPS так і за допомогою більш енерго-ефективного варіанту “Significant Location”. Компанія Apple позиціонує дану можливість як службу що пропонує більш енергозберігаючу альтернативу для програм, які потребують даних про місцезнаходження, але не потребують частих оновлень або точності GPS. Служба покладається на альтернативи з меншим енергоспоживанням (наприклад, Wi-Fi та стільникову інформацію), щоб визначити місцезнаходження користувача. Потім він надсилає оновлення місцезнаходження до вашої програми лише тоді, коли положення користувача змінюється на 500 метрів або більше [20].

Для реалізації цієї можливості був доданий метод startObserving, який наведено нижче.

Лістинг 2.7 – Метод ініціювання спостерігання

```
func startObserving(type: LCAccuracyType) {
    ml_log.info("sf/significant/start{\(type)}")
    switch type {
    case .high(let distance): // 1..10
      locationManager.distanceFilter = distance
      locationManager.desiredAccuracy = kCLLocationAccuracyBest
    case .medium(let distance): // 10...1000
      locationManager.distanceFilter = distance
      locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters
    case .low(let distance): // 1000...3000
      locationManager.distanceFilter = distance
      locationManager.desiredAccuracy = kCLLocationAccuracyKilometer
    case .significant:
      break
    }

    if type == .significant {
      significantLocationManager.startMonitoringVisits()
      significantLocationManager.startMonitoringSignificantLocationChanges()
      isTrackingSignificantLocation = true
    } else {
      locationManager.startUpdatingLocation()
      isTrackingUserRoute = true
    }
```

Як можемо бачити у лістингу 2.7 функція приймає на вхід один параметр і в залежності від нього вмикає необхідне стеження. У випадку .high(distance) це стеження за допомогою GPS у випадку параметру significant це і є вище згадана енерго-ефективна функція.

Дана функція викликається кожен раз коли додаток перезавантажується чи то користувачем чи то системою, що необхідно для забезпечення функціоналу. Зокрема в iOS 14.8.1 нижче стеження не припинялось навіть якщо застосунок був вбитий чи телефон перезавантажувався, тільки користувач в налаштуваннях додатку міг припинити його. Проте починаючи із iOS 15 застосунок зможе вести стеження тільки якщо додаток буде знаходитись в фоновому режимі (в стеку завдань) , дана проблему була детально описана і відрапортовано, проте на момент написання даного звіту проблема ще існувала [21].

Після того як сервіс був запущений всі дані про переміщення користувача надходили в наступний обробник.

Лістинг 2.8 – Обробник локацій

```
        func locationManager(_ manager: CLLocationManager, didUpdateLocations locations:
[CLLocation]) {
            for location in locations {
                // publish each location which was received
                if manager == locationManager { // user tracking manager
                    locationChangesPublisher.onNext(location)
                } else if manager == significantLocationManager {
                    ml_log.info("sf/significant/recieved \(location)")
                    significantLocationChangesPublisher.onNext(location)
                } else if manager == preciseLocationManager {
                    ml_log.info("sf/precise/recieved \(min(location.verticalAccuracy,
location.horizontalAccuracy))")
                    preciseLocationChangesPublisher.onNext(location)
                }
            }
        }
```

Після того як приходило сповіщення про нову локацію (лістинг 2.8) вона відправлялась в Publisher який надсилав інформацію надалі в систему. Наступним важливим елементом в флові запиту є SFRequestRepository.swift який містить необхідні методи для того щоби здійснювати читання / запис чи модифікацію локацій користувача.

Після того як подія про нову локацію була надіслана в систему, спеціальний метод recordSFLocation (лістинг 2.9) здійснює запис про локацію в систему, зокрема всю інформацію про локацію (час, дата, координати, швидкість, точність). А також інформацію про геохеш.

Лістинг 2.9 – Запис даних в базу

```
func recordSFLocation(request: SFLocationRequest, parentId: Int?) -> Observable<Int> {
    return Observable.create { (observer) -> Disposable in
      do {
        try self.writeConnection.transaction {
          try self.writeConnection.run(Table.sfLocationRequest.insert(
            SFLocationRequest.timestamp.identifier <- request.timestamp,
            SFLocationRequest.latitude.identifier <- request.coordinate.latitude,
            SFLocationRequest.longitude.identifier <- request.coordinate.longitude,
            SFLocationRequest.accuracy.identifier <- request.accuracy,
            SFLocationRequest.horizontal_accuracy.identifier <- request.horizontal_accuracy,
            SFLocationRequest.vertical_accuracy.identifier <- request.vertical_accuracy,
            SFLocationRequest.speed_accuracy.identifier <- request.speed_accuracy,
            SFLocationRequest.floor_level.identifier <- request.floor_level,
            SFLocationRequest.speed.identifier <- request.speed,
            SFLocationRequest.type.identifier <- parentId == nil ?
SFLocationRequest.CType.default.rawValue : SFLocationRequest.CType.precise.rawValue,
            SFLocationRequest.accuracy.identifier <- request.accuracy,
            SFLocationRequest.geohash.identifier <- Geohash.encode(coordinate:
request.coordinate.clCoordinate, length: 2),
            SFLocationRequest.geohash3.identifier <- Geohash.encode(coordinate:
request.coordinate.clCoordinate, length: 3),
            SFLocationRequest.geohash4.identifier <- Geohash.encode(coordinate:
request.coordinate.clCoordinate, length: 4),
            SFLocationRequest.geohash5.identifier <- Geohash.encode(coordinate:
request.coordinate.clCoordinate, length: 5),
            SFLocationRequest.geohash6.identifier <- Geohash.encode(coordinate:
request.coordinate.clCoordinate, length: 6)
          ))
          let insertedId = Int(self.writeConnection.lastInsertRowid)
          // update parentId with more precision one
          if let parentId = parentId {
            try self.writeConnection.run(
              Table.sfLocationRequest
                .filter(SFLocationRequest.id.identifier == parentId)
                .update(SFLocationRequest.precise_location.identifier <- insertedId)
            )
          }
          observer.onNext(insertedId)
          observer.onCompleted()
        }
```

Geohash (рис. 2.8) — це система геокодування, винайдена Густаво Німейєром, яка дозволяє нам знати, в якій області на карті знаходиться користувач. Користувач може перебувати в будь-якому місці цієї області, не обов’язково в центрі області [22]. Дана можливість є необхідною для покращення продуктивності застосунку, зокрема при роботі із масштабом карти на основній сторінці, оскільки

в даному випадку проходить групування по geohash відповідного рівня. В застосунку його довжина варіюється від 2 до 6.

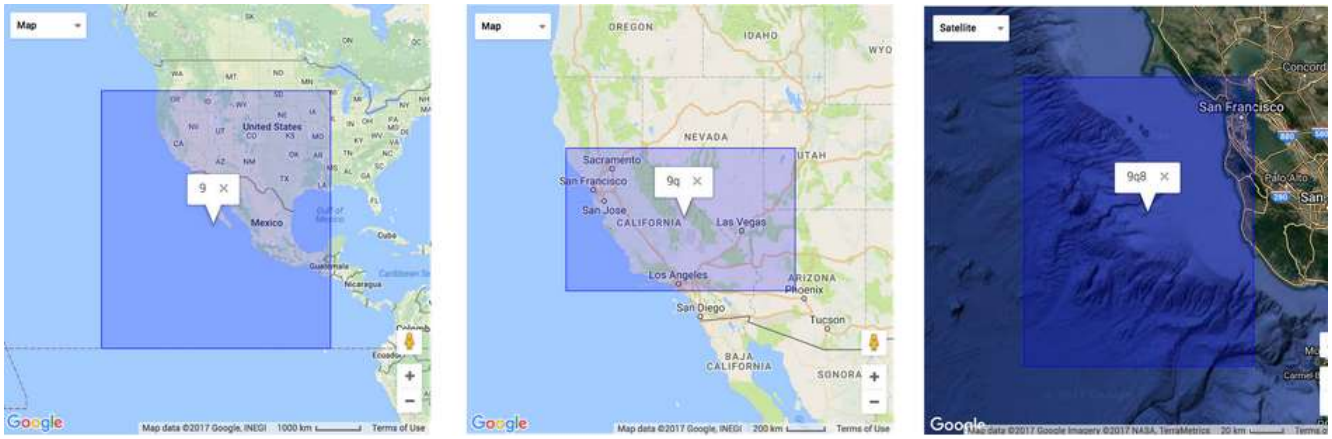

Рисунок 2.8 – Geohash

В якості прикладу як будується витяг даних із бази наведем приклад витягу даних із кластеризацією по гео-хешах (лістинг 2.10). Лістинг коду для запиту наведено нижче. В його задачу входить побудова SQL запиту, витяг даних і створення об’єктів БД.. Як можемо бачити із лістингу будь який запит працює із системою даних фільтрації, оскільки фільтри є глобальні по всьому застосунку. На даний момент вони включають інформацію про типи локацій та їхню точність.

Так як дані із різних джерел в нас зберігають в різних таблицях і мають різну семантичну структуру вони всі зводяться до одного інтерфейсу і об’єднуються за допомогою оператора UNION. Після того як запит був сформований залишається лише одна річ яку необхідно зробити, це зробити запит в базу і отримавши сирі дані створити із них об’єкти рівня БД. Спеціальні класи які містять інформацію і можуть себе створити прийнявши в конструктор сиру інформацію.

Лістинг 2.10 – Взяття локацій із БД

```
func getSFClusterRequests(
    groupId: Int?,
    clusterLevel: SFClusterLevel,
    dateIntervals: (start: Date?, end: Date?)?,
    filteredRegions: Set<String>?
  ) -> Observable<[SFClusterRequest]> {

    guard !filterRepository.currentFilter.tabs.isEmpty else {
      return Observable.just([])
    }
    return Observable.create { (observer) -> Disposable in
      // form basic select query with including SFLocation if needed
      var subQueries = self.filterRepository.currentFilter.tabs
        .map({ (rpType: SFRequestType) -> Table in
          var selectableFields: [Expressible] = rpType.selectableFields(shoudIncludeSFLocation:
false, shortProjection: true)
          selectableFields.append(clusterLevel.geohashLiuteral(for:
rpType).count.as(SFClusterRequest.Key.count))
          return rpType.sql_table
            .select(selectableFields)
            .appendClusterFilter(requestType: rpType, clusterLevel: clusterLevel, filteredRegions:
filteredRegions)
            .appendDateIntervalsFilter(requestType: rpType, dateIntervals: dateIntervals)
            .filter(self.filterRepository.buildAccuracyExpression(requestType: nil))
        })

      // create union query from subqueries
      var query = subQueries[0]
      for index in 1..<subQueries.count {
        query = query.union(subQueries[index])
      }
      // append filters
      let strQuery = query
        .asSQL()
      do {
        observer.onNext(try
SQLiteManager.default.defaultSQLConnection.prepare(strQuery).toSFClusterRequest())
        observer.onCompleted()
      } catch {
        observer.onError(SQLiteError.inner(error))
      }

      return Disposables.create()
    }
```

3 ВИКОРИСТАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Застосунок є доволі простим із точки зору користувача, зокрема на рисунку 3.1 наведено демонстрацію основного екрану який бачить користувач коли запускає додаток. На першому екрані наведено приклад останніх 30 днів пересування (користувач може вибирати із каруселі сьогодні, вчора, цей місяць, минулий місяць, рік, минулий рік та весь час). На другому екрані шпильками позначено місця в яких користувач бував в останні 24 години.

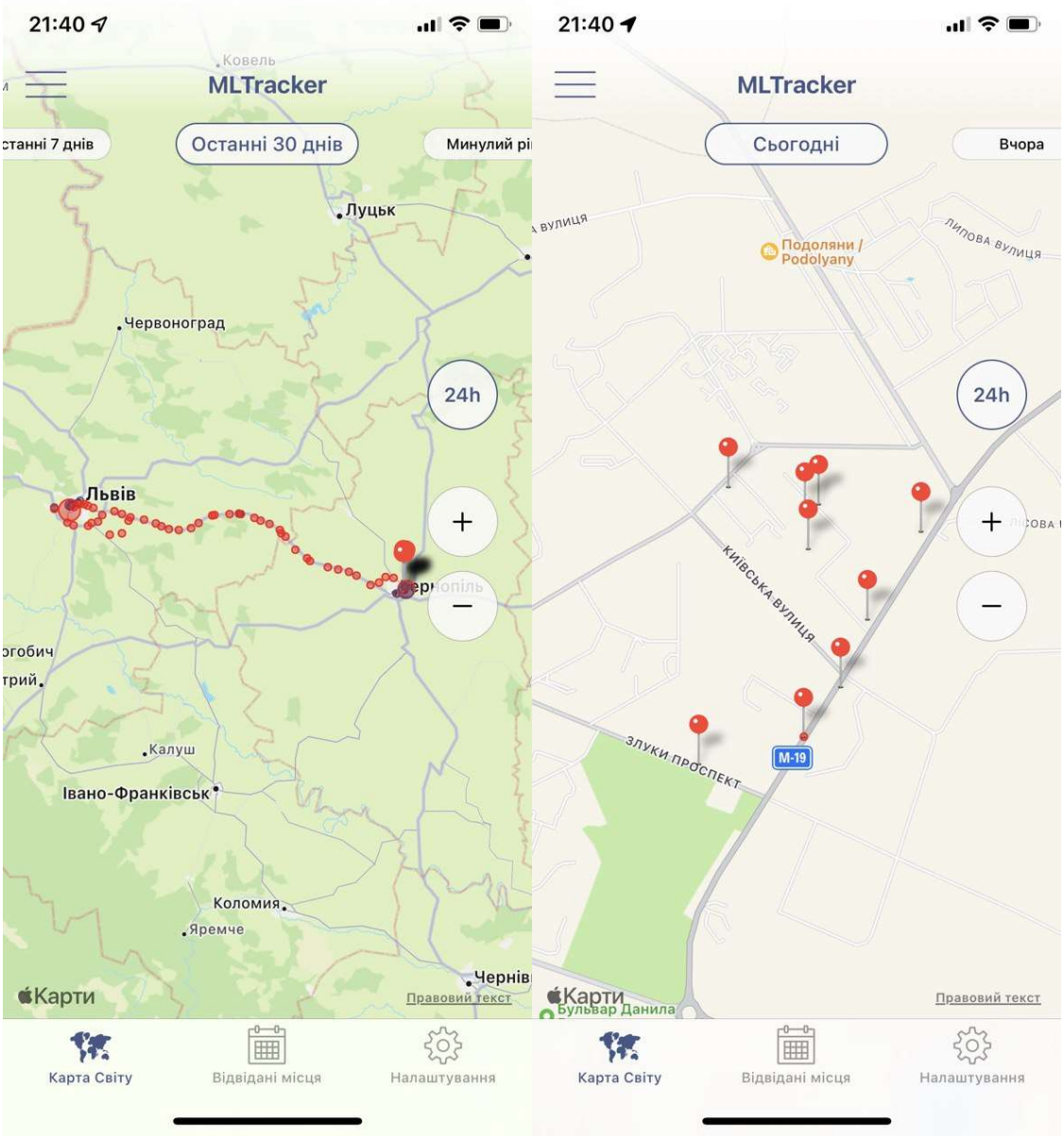

Рисунок 3.1 – Головний екран застосунку

При цьому дані на цих екранах оновлюються автоматично при отриманні нових локацій від користувача.

На рисунку 3.2 наведено приклад де користувач може бачити локації які є згруповані по адміністративному рівні, сортування йде від найбільших відвідин до найменших. Адміністративний розподіл здійснюється за країною, областю, регіоном і локацією. На карті передбачається відображення локацій у вибраному регіоні.

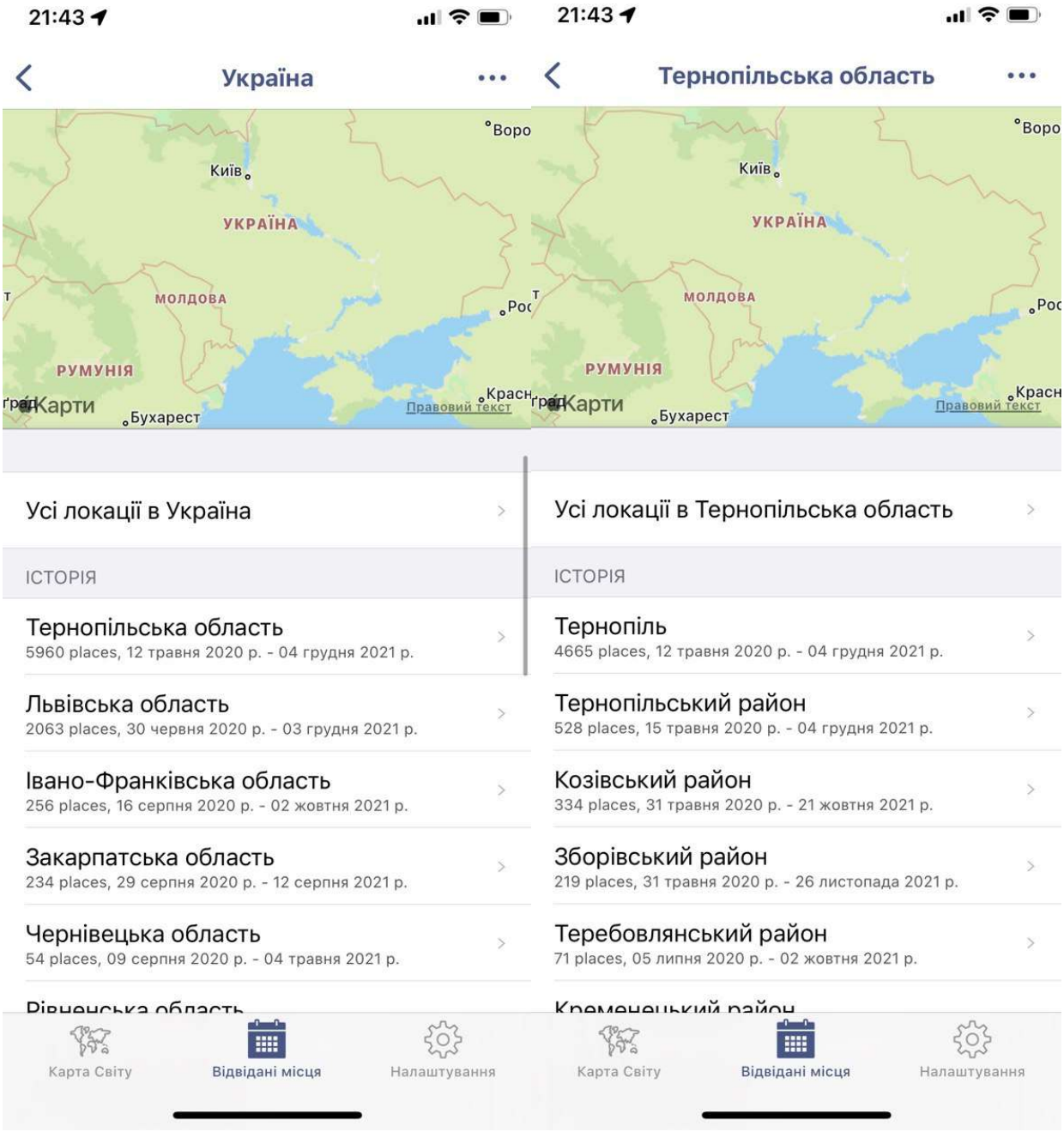

Рисунок 3.2 – Екран перегляду локацій згрупований по адміністративному рівні

На рисунку 3.3 наведено загальний вигляд екрану налаштувань (потенційні), та важлива опція яка відповідає за налаштування оптимізаційні моменти при роботі гео-трекера. Зокрема коли завантажувати необхідну інформацію про локації (коли додаток активний чи коли була отримана локація) а також чи вмикати уточнення коли приходить локація чи візит із певною похибкою.

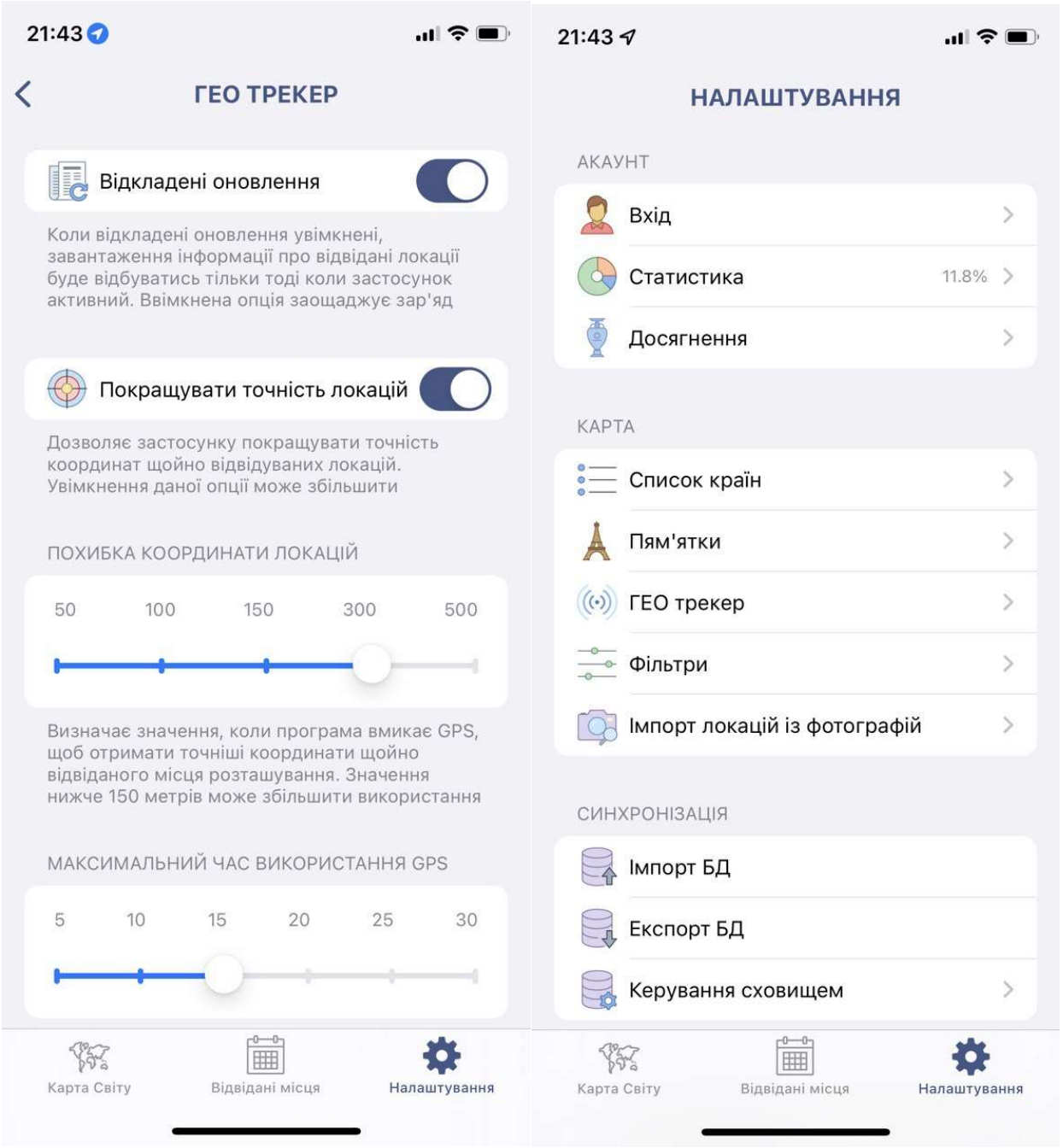

Рисунок 3.3 – Екран налаштувань

Для забезпечення можливості глобального фільтрування (рис. 3.4) зроблено можливість задавати фільтри на одному екрані але вони будуть використовуватись на всіх сторінках. Наразі фільтри підтримують тільки локації та відвідини (місце де користувач пробув певний час). А також вибрати певну точність локацій (вирішено було зробити в якості опцій але в майбутньому можливо буде перероблено на елемент слайдера як на рисунку 1.4.3. Також на рисунку 1.4.4 наведено реакцію операційної системи на недавно встановлений застосунок, коли система попереджує користувача що певні додатки використовують його місцезнаходження. Дане повідомлення буде відображатись декілька разів і опісля перестане зявлятись.

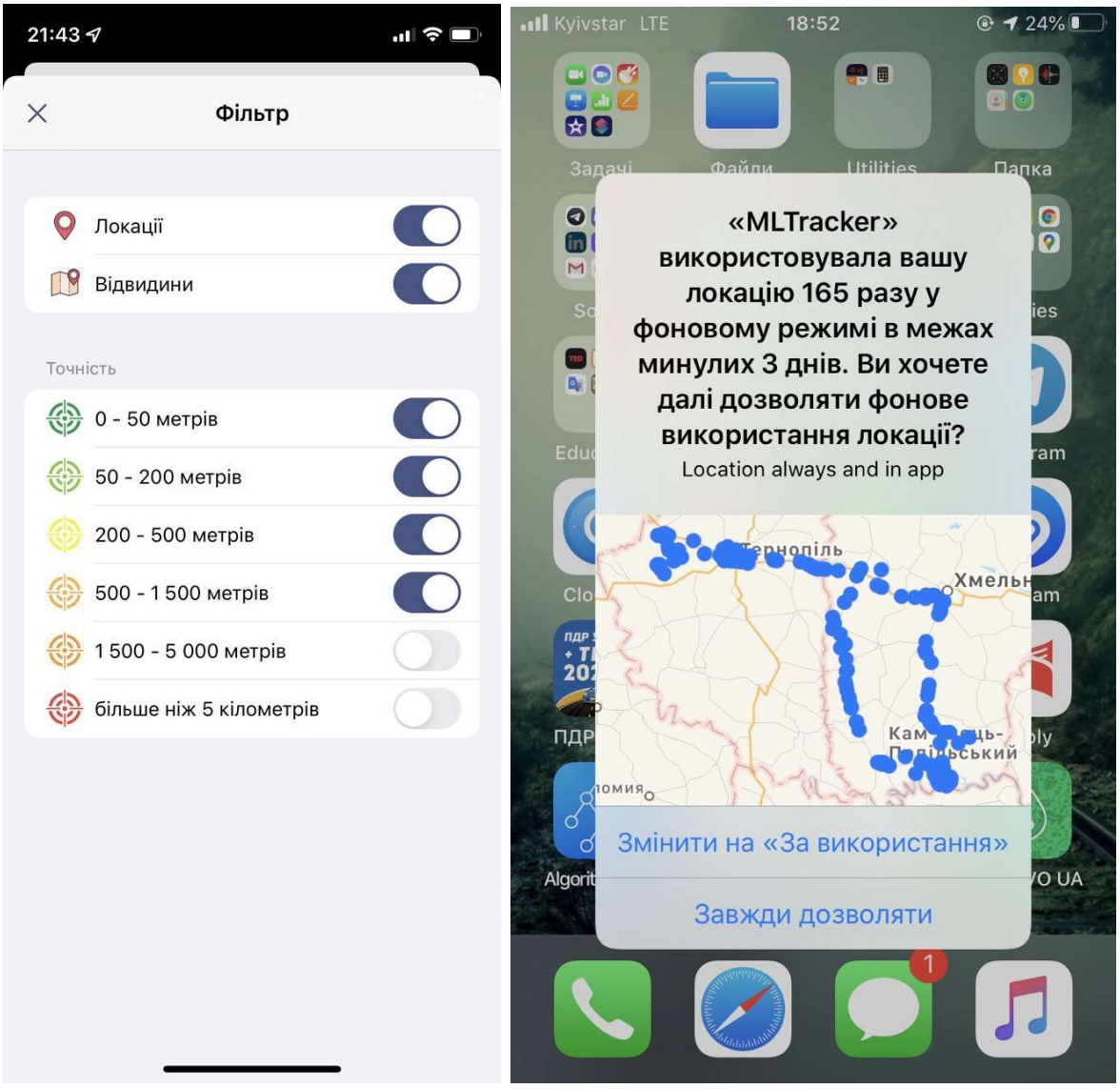

Рисунок 3.4 – Глобальні фільтри та реакція iOS системи на застосунок

4 ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для забезпечення коректності і правильності роботи застосунку проводилось як ручне тестування так і були написаний певні автоматизовані тести.

Враховуючи велику критичність міграційних служб бази даних були написані інтеграційні тести для перевірки на коректність служб визначення мови користувача (лістинг 4.2) які при зміні мови запускають процес локалізації бази даних. На лістингу 4.1 наведено код тестування міграційної служби.

Лістинг 4.1 – Юніт тести для Expression Provider

```
final class SQLiteExpressionProviderTests: MLBaseTest {
  override func setUp() {
    StubLocaleManager.stubLocale = .init(id: 22, language_identifier: "SQLSTUB",
translating_completed: false)
  }

  func testDefaultValueForLocalizableType() {
    // by default should return <fieldname>_<userlocale>
    let field = SQLiteExpressionProvider<Int>(table: "Table", field: "field", localizable: true, locale: nil)
    XCTAssertEqual(field.identifier.template, "\"field_SQLSTUB\"")
    XCTAssertEqual(field.literal.template, "\"Table\".\"field_SQLSTUB\"")
  }

  func testDefaultValueForNoLocalizableType() {
    // by default should return just <fieldname>
    let field = SQLiteExpressionProvider<Int>(table: "Table", field: "field", localizable: false, locale:
nil)
    XCTAssertEqual(field.identifier.template, "\"field\"")
    XCTAssertEqual(field.literal.template, "\"Table\".\"field\"")
  }

  func testDefaultValueForLocalizableOverriden() {
    // by default should return <fieldname>_<customlocale>
    let field = SQLiteExpressionProvider<Int>(table: "Table", field: "field", localizable: true, locale:
MLLocale(id: 0, language_identifier: "en", translating_completed: false))
    XCTAssertEqual(field.identifier.template, "\"field_en\"")
    XCTAssertEqual(field.literal.template, "\"Table\".\"field_en\"")
    // localizable is false should return plain fieldname
    let nfield = SQLiteExpressionProvider<Int>(table: "Table", field: "field", localizable: false, locale:
MLLocale(id: 0, language_identifier: "en", translating_completed: false))
    XCTAssertEqual(nfield.identifier.template, "\"field\"")
    XCTAssertEqual(nfield.literal.template, "\"Table\".\"field\"")
  }
```

Лістинг 3.2 – Юніт тести для тестування локалізації

```
  func testLocalizedFunctionForField() {
    let field = SQLiteExpressionProvider<Int>(table: "Table", field: "field", localizable: false, locale: nil)
    // should return localizable type of field
    let newField = field.localized(for: MLLocale(id: 0, language_identifier: "en",
translating_completed: false))
    XCTAssertEqual(newField.identifier.template, "\"field_en\"")
    XCTAssertEqual(newField.literal.template, "\"Table\".\"field_en\"")
  }
}
```

Враховуючи те що тестами покрити всю систему є доволі важко, проводилось також і ручне тестування системи. Це таке тестування системи під час якого QA вручну перевіряє програмне забезпечення для виявлення помилок. Для цього QA дотримуються письмового плану тестування, який описує набір унікальних сценаріїв тестування. Контроль якості необхідний для аналізу ефективності веб- або мобільного додатка з точки зору кінцевого користувача. Якщо первинне тестування не виявило проблем, збірка публікується для бета тестерів до системи TestFlight де вже аналізуються метрики додатку та баг репорти від користувачів (як правило автоматизовані повідомлення про збій).

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1 Охорона праці

Завдяки досягненням сучасних технологій більшість так би мовити, «канцелярської роботи» в офісі здійснюється з використанням комп'ютерної техніки. Якщо згадати, що в середньому робочий день офісного працівника складає 7-8 годин (як передбачено нормами Кодексу законів про працю України) при п'яти- або шестиденному робочому тижні, можна зробити висновок, наскільки багато часу доводиться проводити віч-на-віч з комп'ютером.

Перелік нормативно-правових актів, що так чи інакше регулюють дане питання, є досить широким. Так, обов'язки роботодавця щодо забезпечення працівникам комфортних та безпечних умов для здійснення роботи, а також права працівників на такі умови передбачено частиною 2 ст. 2 та ч. 1 ст. 21 КЗпП, а також ст. 13 Закону України «Про охорону праці». Даний закон визначає основні положення щодо реалізації конституційного права працівників на охорону їх життя і здоров'я у процесі трудової діяльності, на належні, безпечні і здорові умови праці, регулює за участю відповідних органів державної влади відносини між роботодавцем і працівником з питань безпеки, гігієни праці та виробничого середовища і встановлює єдиний порядок організації охорони праці в Україні.

Більшість актів у даній сфері становлять акти підзаконного рівня, а саме, численні правила, інструкції, державні санітарні правила і норми (ДСанПІН) тощо, якими врегульовуються окремі моменти щодо власне конструкції електронно-обчислювальної техніки, особливостей облаштування приміщень для роботи з нею та низки інших подібних вимог.

На сьогодні до основних підзаконних актів у даній сфері можна віднести:

- Наказ Державного комітету України з промислової безпеки, охорони праці та гірничого нагляду «Про затвердження Правил охорони праці під час експлуатації електронно-обчислювальних машин» від 26.03.2010 № 65;

- Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98, затверджені постановою Головного державного санітарного лікаря України від 10.12.1998 № 7;

- Примірну інструкцію з охорони праці під час експлуатації електронно-обчислювальних машин, затверджену наказом Міністерства доходів і зборів України від 05.09.2013 № 443.

Якщо аналізувати норми, прописані у даних документах, і спробувати викласти їх більш-менш лаконічно й водночас вичерпно мовою зрозумілою більшості читачів, то їх можна звести до такого:

Вимоги щодо розміщення і планування приміщень для роботи з комп'ютером: відповідні робочі місця заборонено облаштовувати у підвальних або цокольних приміщеннях будинків. В обладнанні приміщень забороняється використання полімерних матеріалів, що виділяють шкідливі хімічні речовини. Також слід приділити увагу забезпеченню достатнім для здійснення роботи рівнем освітлення (природного та штучного – у темну пору доби) та звукоізоляції. Для регуляції рівня освітлення природним світлом бажано застосовувати жалюзі. Окрім того, у приміщеннях, де здійснюється робота з комп'ютерами, щодня має здійснюватися вологе прибирання з метою недопущення запиленості підлоги та меблів.

Заземлені конструкції, що знаходяться в приміщеннях, де розміщені робочі місця операторів (батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном), мають бути надійно захищені діелектричними щитками або сітками з метою недопущення потрапляння людини під напругу.

Особливої уваги заслуговують заходи дотримання протипожежної безпеки. Так, у всьому офісі лінії електромережі мають бути забезпечені від виникнення короткого замикання, а також від перепадів мережевої напруги, що може спричинити збої в роботі електронно-обчислювальної техніки. Приміщення (окрім тих, де розташовуються сервери) мають бути оснащені системою автоматичної пожежної сигналізації та вогнегасниками. Під час монтажу та експлуатації ліній

електромережі необхідно повністю унеможливити виникнення електричного джерела загоряння внаслідок короткого замикання та перевантаження проводів, обмежувати застосування проводів з легкозаймистою ізоляцією і, за можливості, застосовувати негорючу ізоляцію. У приміщенні, де одночасно експлуатуються понад п'ять комп’ютерів, на помітному та доступному місці встановлюється аварійний резервний вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення.

Вимоги щодо організації та обладнання робочих місць: площа, відведена на одне робоче місце має становити не менше 6 кв. м., а об’єм – не менше 20 куб. м. Конструкція робочого місця повинна забезпечувати підтримання оптимальної робочої пози (тобто такої, яка дозволяє працівникові виконувати роботу з мінімальним напруженням тіла, і яка дозволяє уникнути перевтоми в ході і після закінчення робочого процесу). Раціональна робоча поза має важливе значення для збереження здоров’я працівника, оскільки тривале перебування його в незручній і напруженій позі може призвести до таких захворювань, як сколіоз (викривлення хребта), варикозне розширення вен, плоскостопість тощо. Установлено, що робота в зігнутому положенні збільшує затрати енергії на 20%, а при значному нахиленні — на 45% порівняно з прямим положенням корпуса.

За потреби особливої концентрації уваги під час виконання робіт суміжні робочі місця операторів необхідно відділяти одне від одного перегородками висотою 1,5 - 2 м.

Робочі місця слід розташовувати відносно джерела природного світла (вікон) таким чином, щоб світло падало збоку, переважно зліва. Також робоче місце має відповідати сучасним вимогам ергономіки:

- стіл повинен мати висоту поверхні 680 - 800 мм., ширину 600 - 1400 мм. і глибину 800 - 1000 мм. (такі параметри забезпечують можливість виконання операцій в зоні досяжності працівника);

- робочий стілець робочий стілець має бути підйомно-поворотним, з можливістю регулювання висоти, бажано зі стаціонарними або змінними

підлікотниками і напівм’якою нековзкою поверхнею сидіння, що легко чиститься і не електризується;

- екран комп’ютера має розташовуватися на оптимальній відстані від користувача, що становить 600 – 700 мм., але не менше за 600 мм. з урахуванням літерно-цифрових знаків і символів.

Вимоги безпеки під час роботи з комп’ютером:

Щодня перед початком роботи оператор повинен:

- оглянути своє робоче місце; про виявлення ознак пошкодження обладнання інформувати свого безпосереднього керівника;

- відрегулювати освітленість на робочому місці, переконатися в відсутності відблисків на екрані комп’ютера, відсутності зустрічного світла;

- перевірити правильність підключення обладнання ЕОМ до електромережі;

- очистити екран комп’ютера від пилу та інших забруднень;

- перевірити правильність організації робочого місця й за необхідності провести відповідні коригування.

Оператор під час роботи зобов'язаний:

- виконувати тільки ту роботу, яку йому було доручено;

- підтримувати порядок і чистоту на робочому місці;

- тримати відкритими всі вентиляційні отвори обладнання;

- коректно закрити всі активні завдання у разі припинення роботи з комп’ютером;

- негайно відключити комп’ютером від електричної мережі у разі виникнення аварійної ситуації.

У ході виконання робіт оператор комп’ютера повинен:

- витримувати відстань від очей до екрана комп’ютером в межах 60 - 70см;

- дотримуватися внутрішньозмінного режиму праці та відпочинку, регламентованих перерв у роботі, а саме (при 8-годинній денній робочій зміні):

- для розробників програм - тривалістю 15 хвилин через кожну годину роботи;

- для інших категорій працівників - тривалістю 15 хвилин через кожні дві години роботи;

- для операторів комп'ютерного набору - тривалістю 10 хвилин, після кожної години роботи.

Під час регламентованих перерв рекомендується виконувати комплекси вправ для очей, рук, хребта, поліпшення мозкового кровообігу тощо. Про виявлення несправності обладнання або інших факторів, які створюють загрозу для життя або здоров'я працівників, необхідно негайно інформувати свого безпосереднього керівника.

Не допускається:

- виконання ремонту та налагодження комп’ютерої техніки безпосередньо на робочому місці оператора;

- зберігання біля комп’ютера паперу, дискет, інших носіїв інформації, запасних блоків, деталей тощо, якщо вони не використовуються для поточної роботи;

- відключення захисних пристроїв, самочинні зміни в конструкції комп’ютера;

- використання комп’ютерів, на екранах яких під час роботи з'являються нехарактерні сигнали, нестабільне зображення на екрані тощо;

- доторкання до задньої панелі системного блоку при включеному живленні;

- вимикання живлення під час виконання активного завдання;

- попадання вологи на поверхню системного блоку, монітора, клавіатури, дисководів, принтерів та інших пристроїв;

- приймання напоїв та їжі на робочому місці.

Після закінчення роботи з використанням необхідно дотримуватися такої послідовності вимикання обладнання:

- закрити всі активні завдання;

- переконатися у відсутності дискет та дисків у дисководах;

- використавши опцію "Завершення роботи" у меню "Пуск", вимкнути живлення системного блоку;

- вимкнути живлення всіх комп’ютерів;
- вимкнути блок аварійного живлення (за наявності);
- відключити комп’ютер від електромережі, при цьому забороняється тягнути штепсельну вилку за дріт.

У випадку виникнення аварійної ситуації оператор зобов'язаний:

- у всіх випадках виявлення пошкодження проводів електричного живлення, несправності заземлення та інших пошкодженнях електрообладнання, виникненні запаху гарі, диму - негайно вимкнути електричне живлення і повідомити про аварійну ситуацію свого безпосереднього керівника й чергового електрика;
- при попаданні людини під електричну напругу негайно звільнити її від дії струму шляхом вимкнення електричного живлення, до прибуття лікаря надати потерпілому долікарську медичну допомогу;
- при будь-яких випадках порушень роботи технічного обладнання або програмного забезпечення негайно викликати представника технічної служби з питань експлуатації обчислювальної техніки;
- у випадку виникнення різі в очах, різкого погіршення зору, виникнення головного болю, больових відчуттів у пальцях та кистях рук, посилення серцебиття - негайно припинити роботу з використанням ЕОМ, повідомити про те, що сталося, свого безпосереднього керівника й звернутися до медичної установи;
- при загорянні обладнання негайно відключити його від електромережі;
- про загорання повідомити свого безпосереднього керівника, оперативного чергового, пожежну службу; ужити заходів щодо ліквідації вогню за допомогою вуглекислотного або порошкового вогнегасника.

5.2 Безпека в надзвичайних ситуаціях

5.2.1 Підвищення стійкості роботи підприємств приладобудівної галузі у воєнний час

Підвищення стійкості роботи об’єктів народного господарства, зокрема підприємств приладо-будівельної галузі, у воєнний час – одна із основних задач цивільної оборони України. Могутність країни базується на стійкій економіці. В сучасних умовах, коли науково-технічний прогрес у всіх сферах виробництва досяг небачених масштабів і привів до створення зброї масового ураження, в разі розгортання великомасштабної війни основні промислові центри і райони будуть головною ціллю для знищення зі сторони противника. Адже виведення економіки з ладу може призвести до того, що країна не може стояти на оборонні своїх кордонів та підтримувати життєдіяльність населення. На сьогодні, через бойові дії на сході України (Війни на Донбасі), проблема підвищення стійкості роботи підприємств приладо-будівельної галузі стоїть як ніколи гостро.

Під стійкістю роботи підприємств приладо-будівельної галузі розуміють їх здатність за умов дії надзвичайних ситуацій виробляти продукцію в запланованих обсязі та номенклатурі, а при одержанні слабких чи середніх руйнувань чи порушенні постачання сировини відновлювати своє виробництво в мінімально короткі терміни. Щоб забезпечити нормальну роботу під час війни промислових об’єктів будівництва , скоротити можливі матеріальні втрати, необхідно ще в мирний час виконати великий комплекс різних заходів, які забезпечили б їхнє функціонування.

На стійкість роботи об’єктів будівництва впливають такі фактори:

– надійність захисту робітників від дії вражаючих факторів, що виникають під час надзвичайних ситуацій;

– здатність будівельного комплексу протистояти дії вражаючих факторів;

– надійність систем постачання об’єкта сировиною для виробництва певного виду продукції;

– стійкість системи управління виробництвом та цивільною обороною в надзвичайних ситуаціях;

– готовність об’єкта до проведення рятувальних дій або робіт по відновленню виробництва;

– захищеність об’єкта від дії вторинних вражаючих факторів.

При вирішенні проблеми підвищення стійкості роботи підприємств будівельної галузі, а також інших об’єктів народного господарства, керуються єдиними принциповими положеннями:

– завчасне проведення заходів цивільного захисту, спрямованих на зниження можливих втрат та руйнувань у разі застосування з боку противника зброї масового ураження і на створення умов для швидкого відновлення виробництва після часткового руйнування;

– комплексний підхід в розробці і здійсненні заходів для всіх напрямків діяльності підприємства;

– узгодження цих заходів з територіальними і військовими органами управління.

Заходи з підвищення стійкості плануються з урахуванням місцевих умов, ступеня важливості об’єкта, його географічного положення, економічної доцільності проведення заходів. На мирний час планують, в основному, трудомісткі заходи, які потребують значних матеріальних витрат і часу, а на період загрози виникнення НС – такі заходи, які не потребують значних затрат часу чи проведення яких не є доцільним при нормальному функціонуванні. Також при проведені заходів з ЦЗ потрібно враховувати і внутрішні фактори, що впливають на стійкість: розмір виробництва, виду продукції, що випускається, чисельність працівників, рівень їх дисциплінованості і компетентності, особливості технології виробництва, системи постачання виробництва сировиною, технічною і питною водою, газо- та електроенергією.

З урахуванням розглянутих вище факторів виділяють такі основні шляхи і способи підвищення стійкості роботи підприємств будівельної галузі:

– забезпечення надійного захисту робітників і службовців: укриття робітників і службовців, які продовжують роботу на об'єкті у воєнний час;

– проведення евакуації робітників, службовців і членів їх сімей та забезпечення їх життєдіяльності; використання індивідуальних засобів захисту;

– захист основних виробничих фондів об'єкта від поразки: підвищення певною мірою опірності будівель, споруд впливу ударної хвилі, світлового випромінювання;

– укриття найбільш уразливого обладнання в захисних пристроях (шатрах, камерах, конусах і ін.);

– часткову зміну технології виробництва;

– вивезення в безпечні райони надлишків горючих речовин;

– забезпечення стійкого постачання об'єкта всім необхідним для виробництва;

– підвищення надійності роботи транспорту;

– підготовка паливно енергетичного господарства до роботи у воєнний час;

– підготовка обладнання для роботи на декількох видах палива розосередження запасів найбільш уразливого обладнання, приладів, сировини;

– встановлення виробничих контактів з дублерами постачальниками, необхідних для безперебійної роботи об'єкта;

– підвищення надійності та оперативності управління виробництвом:

– створення об'єктового і заміського пункту управління; прокладка підземних кабельних ліній зв'язку до всіх елементів об'єкта; створення оперативних змін управління для основного і заміського пунктів управління;

– підготовка до виконання робіт по відновленню об'єкта у воєнний час;

– планування відновлювальних робіт за кількома варіантами.

Кожен шлях містить кілька способів підвищення стійкості роботи підприємства, які, в свою чергу, містять кілька заходів ЦЗ або доповнюються ними. Наведені вище шляхи підвищення стійкості підприємств будівельної галузі

реалізуються за допомогою затверджених норм з ЦЗ прийнятих і обов'язкових до виконання для всіх об'єктів усіх галузей виробництва незалежно від форм власності і підпорядкування.

Норми ЦЗ призначені для:

– захисту і зниження ймовірних втрат серед населення;

– зменшення рівня руйнувань основних фондів виробництва;

– підвищення стійкості роботи об'єкта і галузей виробництва;

– забезпечення умов для ліквідації наслідків надзвичайних ситуацій;

– розробки плану проведення рятувальних робіт в осередках ураження в повному обсязі та в максимально короткі терміни.

Контроль за виконанням вимог згаданих норм покладається на структурні підрозділи з питань цивільного захисту та надзвичайних ситуацій.

5.2.2 Організація та проведення медичних заходів для забезпечення безпеки у надзвичайних ситуаціях

Навіть в умовах мирного часу можуть виникати осередки масового ураження надзвичайного характеру, котрі призводять до порушення нормальної діяльності інфраструктури України зокрема і порушення діяльності підприємств приладобудівної галузі.

Щорічно в Україні виникає близько 1000 важких НС природного та техногенного характеру, котрі призводять до загибелі тисяч людей, а матеріальні збитки сягають кількох мільярдів гривень і складають від 3,2 до 4% внутрішнього валового продукту.

За таких умов поряд з іншими завданнями щодо ліквідації наслідків НС важливе значення мають медичні завдання.

Медичний захист населення, та працівників заводів котрі мешкають в районі розташування АЕС, складає важливу частину цілого комплексу захисних заходів,

котрі реалізуються штабами та службами ЦО у випадку виникнення радіаційне небезпечної аварії на такій станції.

Основною метою таких заходів є зведення до мінімуму кількості опромінених людей та доз їх опромінення, зумовленого знаходженням на радіаційне забрудненій території.

Заходи щодо захисту населення плануються та реалізуються на підставі закону України "Про захист людини від впливу іонізуючих випромінювань" для використання в роботі з питань ЦО та НС за №54/09-1- від 24. 02. 98 р.

Планування вказаних заходів здійснюється органами управління ЦО згідно з вимогами "Типового змісту плану заходів по захисту населення у випадку загальної радіаційної аварії на АЕС".

Принципи організації медичної допомоги потерпілим внаслідок аварії на радіаційно небезпечному об'єкті. Основні заходи при організації медичної допомоги ураженим включають:

- проведення заходів протирадіаційного захисту;

- надання в максимально короткі строки першої медичної допомоги;

- організація евакуації уражених з забрудненої зони;

- проведення санітарної обробки уражених та дезактивація їх одягу та взуття;

- максимальне наближення до місця аварії формувань ЕМД, що надають першу медичну допомогу;

- організація спеціалізованої медичної допомоги; Медичне забезпечення потерпілих при аварії здійснюється як лікувально-профілактичним закладом (ЛПЗ) об'єкта атомної енергетики, так і силами ЕМД за заздалегідь розробленим планом.

Згідно з планом захисту населення розробляється "План медичного забезпечення населення у випадку загальної аварії на атомній станції", в якому більш детально висвітлюються завдання місцевих сил цивільної оборони, відповідальність посадових осіб та порядок взаємодії з іншими службами, органами управління ЦО, медично-санітарною частиною, яка забезпечує АЕС.

У випадку виникнення загальної аварії на АЕС, основними заходами місцевих сил цивільної оборони е:

- участь в контролі за рівнями радіації на місцевості;

- порядок видачі хворим та персоналу медичних закладів ЗІЗ та використання ПРУ;

- організація санітарного нагляду за радіаційною безпекою різних груп населення, а також осіб, шо приймають участь у ліквідації наслідків аварій на АЕС;

- організація медичного обстеження населення, яке зазнало дію іонізуючого випромінювання та диспансерний нагляд за ними;

- загальні санітарно-гігієнічні та протиепідеміологічні заходи. З усіх заходів, які здійснює МСЦО стосовно населення, яке

зазнало дії іонізуючого випромінювання внаслідок аварії на АЕС, найбільш важливим в початковий період після ЇЇ виникнення є йодна профілактика, яка є достатньо ефективним методом захисту щитовидної залози від дії радіаційних ізотопів, що надходять в організм людини інгаляційним шляхом. Засоби йодної профілактики в таблетках, а через 1,5--2 місяці після аварії для виведення J132, J134 -- ферроцин у вигляді порошків.

Якщо випромінювання, що прогнозується:

- не перевищує нижнього рівня, то не потрібно реалізовувати заходи;

- перевищує нижній рівень, але не досягає верхнього, то рішення про заходи захисту може бути відстрочено і повинно прийматись з урахуванням конкретної радіаційної обстановки та місцевих умов;

- перевищує або досягає верхній рівень, то проведення заходів необхідне.

Дозові критерії для заходів ранньої фази аварії відносяться до дози, яка прогнозується на короткий час (але не менше, ніж тривалість ранньої стадії). Дозові критерії для обмеження споживання забруднених продуктів та питної води відносяться до прогнозованої дози від внутрішнього опромінення радіонуклідами в залежності від кількості води та харчів, які будуть спожиті протягом 1 року.

ВИСНОВКИ

У кваліфікаційній роботі на здобуття освітнього рівня магістра було досліджено та розроблено мобільний застосунок який призначений для запису та відображення локацій користувача у хронологічному порядку із заданими фільтрами. Проєкт розроблений по стратегії MVP, саме тому містить найнеобхідніший функціонал. Дану стратегію обрано для перевірки життєздатність та конкурентоспроможності проєкту на ринку. При вдалій конкуренції на ринку система буде розвиватись.

Під час виконання були проаналізовані доступні варіанти баз даних для iOS і вибрано SQLite як основну і таку яка задовольняє поставленим вимогам, зокрема переносимість між платформами, що може бути корисно при розробці додатку під операційну систему Android. Для забезпечення можливості локалізації бази даних було обрано із чотирьох доступних варіантів – локалізацію по стовпчику. Це надало системі гнучкості в добавлянні мов а також не призводить до серйозного збільшення розміру при добавлянні нової мови. Для унфікації запитів були створені спеціальні менеджери які контролюють назви поля.

Для отримання даних користувача про геопозицію було використано два підходити, зокрема це “Significant-Location Changes” та “Visit monitoring”. Даних два способи виявились доволі енергоефективними і повністю задовольняють вимогам про запис локацій користувача, для виправлення низької точності які можуть виникати при використанні даних способів було додано опцію ввімкнення GPS для взяття точніших координат.

Система була побудована гнучкою і не залежить від конкретних систем локацій, які дозволяють отримувати інформацію про місце на основні координат, забезпечується це тим що служби геокодингу використовують абстракції над сервісами геокодингу такими як CoreLocation від Apple чи потенційний OSM. Виконується групування по адміністративних рівнях і при потребі можливий варіант гнучкого добавляння нових варіантів групування.

Інтерфейс користувача був побудований із використанням практик IOS і стандартної бібліотеки зображень таких як SFSymbols, що безумовно покращить досвід користування і інтуїтивність роботи з застосунком.

Як результат даної кваліфікаційної роботи є програмний продукт під операційну систему iOS, який є відтестованим і доволі стабільним, і може використовуватись будь якими користувачами через посилання на тестування TestFlight чи офіційний магазин Apple AppStore.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. “Google Maps Timline”. [Електронний ресурс] URL: https://support.google.com/maps/answer/6258979
2. “Apple Significant Location”. [Електронний ресурс] URL: https://www.apple.com/legal/privacy/data/en/location-services/
3. “Як розробляти MVP”. [Електронний ресурс] URL: https://clearbridgemobile.com/planning-a-minimum-viable-product-a-step-by-step-guide/
4. Matt Neuburg, iOS 13 Programming Fundamentals with Swift, O`Reilly Media, Inc., 2020
5. “SwiftUI in 2021: The Good, the Bad and the Ugly”. [Електронний ресурс] URL: https://betterprogramming.pub/swiftui-2021-the-good-the-bad-and-the-ugly-458c6ee768f9
6. Russ Miles & Kim Hamilton, Learning UML 2.0, O`Reily Media, Inc., 2003
7. “Know the difference between different methodologies range”. [Електронний ресурс] URL: https://www.janbasktraining.com/blog/agile-scrum-kanban-and-waterfall/
8. “Що таке Agile і як його застосовувати в бізнесі” [Електронний ресурс] URL: https://brainrain.com.ua/uk/chto-takoe-agile-ua/
9. “Kanban vs Scrum vs Agile” [Електронний ресурс] URL: https://svitla.com/blog/kanban-vs-scrum-vs-agile
10. “What is Waterfall model” [Електронний ресурс] URL: http://tryqa.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/
11. “Screaming architecture”, Robert C.Martin [Електронний ресурс] URL: https://blog.cleancoder.com/uncle-bob/2011/09/30/Screaming-Architecture.html
12. “Clean architecture”, Rober C.Martin [Електронний ресурс] URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

13. “Заблуждение Clean Architecture” [Електронний ресурс] URL: https://habr.com/ru/company/mobileup/blog/335382/
14. “Rober C Martin – Clean Architecture and Design” [Електронний ресурс] URL: https://www.youtube.com/watch?v=Nsjsiz2A9mg
15. “Swift or Objective-C” [Електронний ресурс] URL: https://keyua.org/blog/swift-vs-objective-c-which-is-better/
16. “Swift Language” [Електронний ресурс] URL: https://www.apple.com/swift/
17. “SQLite for iOS” [Електронний ресурс] URL: https://www.raywenderlich.com/6620276-sqlite-with-swift-tutorial-getting-started
18. “Database Localization and Internatiolization” [Електронний ресурс] URL: https://www.soluling.com/Help/Database/Index.htm
19. “Swift Style Guide” [Електронний ресурс] URL: https://google.github.io/swift/
20. “Using the Significant-Change Location Service” [Електронний ресурс] URL: https://developer.apple.com/documentation/corelocation/getting_the_user_s_location/using_the_significant-change_location_service
21. “Significant-Change Location Service not working properly on iOS 15” [Електронний ресурс] URL: https://developer.apple.com/forums/thread/694081
22. “Geohashing” [Електронний ресурс] URL: https://medium.com/@bkawk/geohashing-20b282fc9655
23. “Manual Testing” [Електронний ресурс] URL: https://www.browserstack.com/guide/manual-testing-tutorial
24. “iOS Unit Testing” [Електронний ресурс] URL: https://www.raywenderlich.com/21020457-ios-unit-testing-and-ui-testing-tutorial
25. “Integration Tests in Swift” [Електронний ресурс] URL: https://www.swiftbysundell.com/articles/integration-tests-in-swift/
26. Гогіташвілі Г. Г., Карчевські Є.-Т., Лапін В. М. Управління охороною праці та ризиком за міжнародними стандартами: Навч. посіб. – К.: Знання, 2007. – 367 с.

27. Русаловський А. В. Правові та організаційні питання охорони праці: Навч. посіб. – 4-те вид. – К.: Університет «Україна», 2009. – 295 с.
28. Третьяков О.В., Зацарний В.В., Безсонний В.Л. Охорона праці: Навчальний посібник з тестовим комплексом на CD/ за ред. К.Н. Ткачука. – К.: Знання, 2010. – 167 с. + компакт-диск.
29. Желібо Є.П., Заверуха Н.М., Зацарний В.В. Безпека життєдіяльності: Навчальний посібник для студентів вищих закладів України . – Київ: “Каравела”, 2006. – 320с.
30. Безпека життєдіяльності (забезпечення соціальної, техногенної та природної безпеки: Навч. посібник/ В.В. Бєгун, І.М. Науменко - К.: , 2004. – 328с.

ДОДАТКИ

Додаток А

Технічне завдання

Додаток Б

Публікація в науковому виданні

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

IX НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

8–9 грудня 2021 року

ТЕРНОПІЛЬ
2021

УДК 004.41

П.О. Стандрет, Г.Б Цуприк, канд. техн. наук, доц.
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ОПЕРАЦІЙНОЇ СИСТЕМИ IOS ДЛЯ ЗДІЙСНЕННЯ КОНТРОЛЮ ІСТОРІЇ ПЕРЕСУВАННЯ КОРИСТУВАЧА З ВИКОРИСТАННЯМ SWIFT

UDC 004.41

P.O. Standret, H.B. Tsupryk, PhD, Assoc.Prof.

DEVELOPMENT OF THE IOS MOBILE APPLICATION FOR CONTROLLING USER LOCATION HISTORY USING SWIFT

Традиційно на протязі життя людина відвідує різноманітні місця, їздить на автомобілі чи громадському транспорті, літає у відпустки за кордон або відвідує цікаві місця які часто не позначені на картах. З плином часу людська пам'ять втрачає можливість з точністю пригадати конкретне місце і точний час коли було відвідане те чи інше місце, і насправді це є проблемою бо люди часто хочуть через роки згадати і освіжити в пам'яті спогади, чи поділитись цікавими місцями із іншими. Для збереження особливих моментів люди робили фотографії на плівкові фотоапарати а сьогодні фотографують на цифрові фотоапарати чи телефони і роблять підписи до них, інші використовують спеціалізовані GPS трекери, дехто веде довідник. В даному випадку згадані варіанти вирішують проблему частково, оскільки різноманітність джерел інформації та їхній обсяг росте, і систематизувати їх стає все складніше і складніше, так само як і шукати в них необхідну інформацію.

Основною метою даної розробки є створення універсального застосунку для операційної системи iOS який би зміг вирішити проблему збереження і доступності даних, а також опціонально при згоді користувача записував історію пересувань, відвіданих місць, аналізував локаційні мітки з фотоальбому телефону і надавав результати користувачеві з відображенням на карті та списку який згрупований по адміністративних рівнях. Відображення в свою чергу має мати багатий вибір інструментів фільтрування чи групування. При цьому додаток має бути енергоефективним у стані пасивного моніторингу гео-локацій і надавати опції для змоги коригувати моменти які впливають на точність визначення локацій за рахунок споживання батареї. Інтерфейс застосунку має бути інтуїтивно зрозумілим із використанням рекомендацій компанії Apple для розробки мобільних додатків.

Для імплементації даного рішення як цільову операційну систему було обрано iOS а мову програмування Swift. Застосунок буде використовувати останню версію мови програмування Swift і можливості API iOS для контролю локаційних сервісів, зокрема це локаційні мітки (істотні місце пересування) і візити (місця де користувач залишався деякий час) а також аналіз гео-даних із фотографій [1]. Для оптимізації роботи будуть використовуватись також сторонні бібліотеки для роботи із базою даних SQLite, розширення для мови Swift – RxSwift і бібліотека стандартних компонентів для застосунків під iOS. Застосунок буде розповсюджуватись через офіційний магазин AppStore для готових збірок та систему TestFlight для тестування та аналізу збірок що в розробці.

Підсумовуючи, даний застосунок буде корисний для групи користувачів яким важлива статистика, інформація про пересування і відвідані місця з точною хронологією та акцентом на конфіденційність локаційних даних, цільовій аудиторії яка цінує гнучкість в налаштуваннях та в різноманітті потенційних можливостей в налаштування вибірок даних і доступні можливості оперування ними.

Література.

1. Getting the User's location [Електронний ресурс]. URL: https://developer.apple.com/documentation/corelocation/getting_the_user_s_location.

Додаток В

Діаграма класів застосунку