НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр Коваль

«___»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра

спеціальності 122 «Комп'ютерні науки»

освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

на тему: «Мобільний додаток для роботи з GitHub**»**

Виконав:
студент IV курсу, групи ТР-71
Чухліб Кирило Валентинович _____

Керівник:
Професор, доктор технічних наук, доцент
Шушура Олексій Миколайович _____

Рецензент:
Зав. Кафедри, д.т.н, проф.
Сторчак К.П. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

спеціальності 122 «Комп’ютерні науки»

освітня програма «Комп’ютерний моніторинг та геометричне моделювання процесів і систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр Коваль

(підпис)

”___”_____2021р.

ЗАВДАННЯ

на дипломну роботу студенту

Чухліб Кирилу Валентиновичу

(прізвище, ім’я, по батькові)

1. Тема роботи: «Мобільний додаток для роботи з GitHub»

керівник роботи: Шушура Олексій Миколайович, д.т.н., доцент

(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”24” травня 2021р. № 1267-с

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: мова програмування Swift, середовище розробки XCode.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити). Постановка задачі, аналіз існуючих програмних рішень для роботи із сервісом

GitHub, вибір засобів розробки програмного продукту, розробка структури програмного забезпечення, створення програмного продукту, робота користувача з програмною системою.

5. Перелік ілюстративного матеріалу:

Схема роботи GIT, огляд існуючих програм, актуальність, постановка задачі, вимоги до системи, засоби розробки, паттерн MVVM, блок схема залежностей екрану, UML діаграма використання, екрани логіну у додатку, екран налаштувань та профілю, екран пошуку та спілкування, екран коду та статистики, висновки, апробація результатів.

6. Дата видачі завдання ”10” жовтня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	09.10.2020	
2.	Вивчення та аналіз задачі	01.04.2021	
3.	Розробка архітектури та загальної структури системи	14.04.2021	
4.	Розробка структур окремих підсистем	18.04.2021	
5.	Програмна реалізація системи	20.04.2021	
6.	Оформлення пояснювальної записки	04.05.2021	
7.	Захист програмного продукту	11.05.2021	
8.	Передзахист	25.05.2021	
9.	Захист	17.06.2021	

Студент _____ К.В. Чухліб

(підпис) (прізвище та ініціали,)

Керівник роботи _____ О.М. Шушура

(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою дипломної роботи є покращення процесу взаємодії використання сервісу GitHub шляхом створення спеціалізованого мобільного додатку. Для реалізації програмного продукту була обрана мова програмування Swift з використанням моделі MVVM - Model-View-ViewModel, що у майбутньому дозволить простіше розроблювати, тестувати та модифікувати мобільний додаток. Проаналізовано декілька сервісів контролю версій, вибрано платформу для розробки та реалізовано програмний продукт для повноцінного його використання.

У роботі розглянуто декілька архітектурних рішень для реалізації проекту: MVC та MVVM. Вибір проводився аналізуючи усі недоліки та переваги що має кожен з цих патернів.

У процесі розробки було реалізовано програмне забезпечення для роботи з сервісом GitHub, а саме: пошук потрібної інформації, спілкування з людьми, нагадування, декілька варіантів робочого інтерфейсу, підсвітка коду кожної мови програмування тощо. Виконано дослідження щодо аналогічних програмних продуктів, їх плюсів та недоліків. Виконано аналіз серед найпопулярніших сервісів для контрою версій програмних продуктів.

Даний продукт було розроблено за допомогою IDE XCode та мови Swift 5.1 з використанням сторонніх бібліотек.

Дипломну роботу виконано на 63 аркушах, вона містить 40 ілюстрацій, 12 посилань на літературу та 4 додатки.

Ключові слова: iOs, Swift, MVVM, REST API V3, GUI, XCode.

ABSTRACT

The purpose of the thesis is to improve the process of interaction with the use of the GitHub service by creating a specialized mobile application. To implement the software product, the Swift programming language was chosen, using the MVVM model - Model-View-ViewModel which will make it easier to develop, test and modify the mobile application. Several version control services were analyzed, a platform for development was selected and a software product was implemented for its full use.

The paper considers several architectural solutions for the project: MVC and MVVM. The choice was made by analyzing all the disadvantages and advantages of each of these patterns.

In the process of development, software for working with the GitHub service was implemented, namely: searching for the necessary information, communicating with people, guessing, several options of the working interface, highlighting the code of each programming language, etc. Research has been conducted on similar software products, their advantages and disadvantages. An analysis of the most popular services for controlling software versions has been performed.

This product was developed using the XCode IDE and Swift 5.1 using third-party libraries.

The thesis is made on 63 sheets, it contains 40 illustrations, 12 references and 4 addition.

Keywords: iOS, Swift, MVVM, REST API V3, GUI, XCode.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GUI – графічний інтерфейс користувача (перекл. Graphical User Interface)

API - прикладний програмний інтерфейс (перекл. Application Programming Interface)

ARC - Automatic Reference Counting

DSL - це мова, створена для вираження конкретного домену або вирішення певної проблеми.

JSON – спеціалізований формат запису даних.

ВСТУП

Git - одна з найкращих та найбільш бажаних доступних систем контролю версій програмного забезпечення. Сучасному розробнику потрібно швидко отримувати доступ до коду бібліотек, програм та open-source проектів для обговорення, пошуку потрібної для себе інформації тощо, яка представлена на Git. Сьогодні присутній різноманітний ряд інтерфейсів Git для різних операційних систем таких як: Android, iOS, Mac, Linux тощо. Вибір між мобільними додатками або веб-сайтами як варіантами доступу до інформації залежить від вартості їх розробки, зручності використання всіх інтерфейсів та необхідних функцій для аудиторії, для якої вони розроблюються.

З огляду на це, дуже часто користувачі більше віддають перевагу мобільним додаткам, ніж їх мобільним веб-сайтам. Це і є вагома причина для створення мобільних додатків для звернення до потенційних (і існуючих) клієнтів з тією чи іншою послугою. Так само у мобільних додатках набагато більше варіантів взаємодії з самим користувачем через push повідомлення, геолокація, вподобання які дозволяє бачити система смартфона, активність користувача тощо.

Саме тому метою дослідження є покращення процесу використання сервісу гіт хаб шляхом створення спеціалізованого мобільного додатку мовою програмування Swift з використанням моделі MVVM, що у майбутньому дозволить простіше розроблювати, тестувати та модифікувати мобільний додаток.

Для реалізації додатку обрана мова програмування Swift 5.1 з використанням моделі MVVM - Model-View-ViewModel, що у майбутньому дозволить простіше розроблювати, тестувати та модифікувати мобільний додаток. В такому масштабному проекті, звичайно, не обійтися без менеджера бібліотек для Xcode.

Новизна дипломної роботи аргументується відсутністю сучасних аналогів, програмних продуктів які були написані з використанням останніх оновлень API GitHub та мови програмування Swift 5.1.

Практична значимість – запропонований додаток дозволить користувачам переглядати великий збірник різних бібліотек та програм з повним кодом прямо у смар-

тфоні. Можна передивлятись свої роботи як готові продукти для майбутнього працевлаштування, оскільки популярні аккаунти GitHub дуже добре оцінюються роботодавцями, спілкуватись з іншими розробниками тощо.

В тому числі кожен користувач може допомагати з існуючими проектами, оскільки це open-source система, тому всі проекти які там розміщені дуже добре написані та не мають багів чи недоліків. Використання таких програм чи бібліотек тільки поліпшить роботу будь-якого іншого програмного додатку де вони будуть використовуватись. А отже – це все буде супроводжуватися підвищенням якості розроблення програмних продуктів.

1 ЗАДАЧА РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ ДЛЯ GITHUB

Метою дипломної роботи є покращення процесу використання сервісу гіт хаб шляхом створення спеціалізованого мобільного додатку.

Для досягнення поставленної мети необхідно вирішити наступні задачі:

– виконати огляд існуючих аналогів;
– провести аналіз документації API GitHub;
– вибрати засоби розробки та методи реалізації;
– спроектувати схеми екранів додатку;
– розробити програмне забезпечення;
– протестувати створений додаток;
– описати взаємодію користувача із системою.

Загальні вимоги до системи:

– зручний для зрозумілий інтерфейс;
– операційна система iOs 14 або новіша;
– підтримка всіх телефонів Apple з iOs 14 і вище.

Функціональні вимоги до системи:

– реалізувати вікна перегляду репозиторіїв та користувачів;
– створити фільтри користувачів, мов програмування;
– дати доступ до інформації яка пов’язана з репозиторієм: деталі, події;
– реалізувати пошук та перегляд історії будь-якого репозиторію;
– відобразити інформацію по особистому аккаунту GitHub;
– реалізувати перегляд комітів;
– створити різні дизайни додатку;
– розробити перегляд програмного коду з підсвіткою;
– реалізувати декілька варіантів зображення кольорової теми;
– створити варіанти взаємодії с іншими людьми.

2 АНАЛІЗ ХАРАКТЕРИСТИК СИСТЕМ КОНТРОЛЮ ВЕРСІЙ

Система контролю версіями - це інструмент для керування різними варіантами однієї одиниці інформації: коду програми, веб-сторінки, веб-сайту тощо. Усі версії автоматично позначаються унікальними символами, в цих версіях і записуються зміни проекту. Додатково до цього зберігається час та інформація про користувача який вніс зміни. Деякі інструменти, щоб контролювати версії, вже входять до складу найпопулярніших середовищ розробки.

2.1 Схема роботи системи контролю версій

Існує дуже багато різних систем, таких як: Git, Mercurial, CVS, SVN тощо. Так само вони діляться на централізовані та розподілені. Всі вони різняться як по структурі так і за підходом до використання.

На рисунку 2.1 можна побачити схему роботи централізованої системи контролю версій при паралельній роботі декількох програмістів над одним проектом.

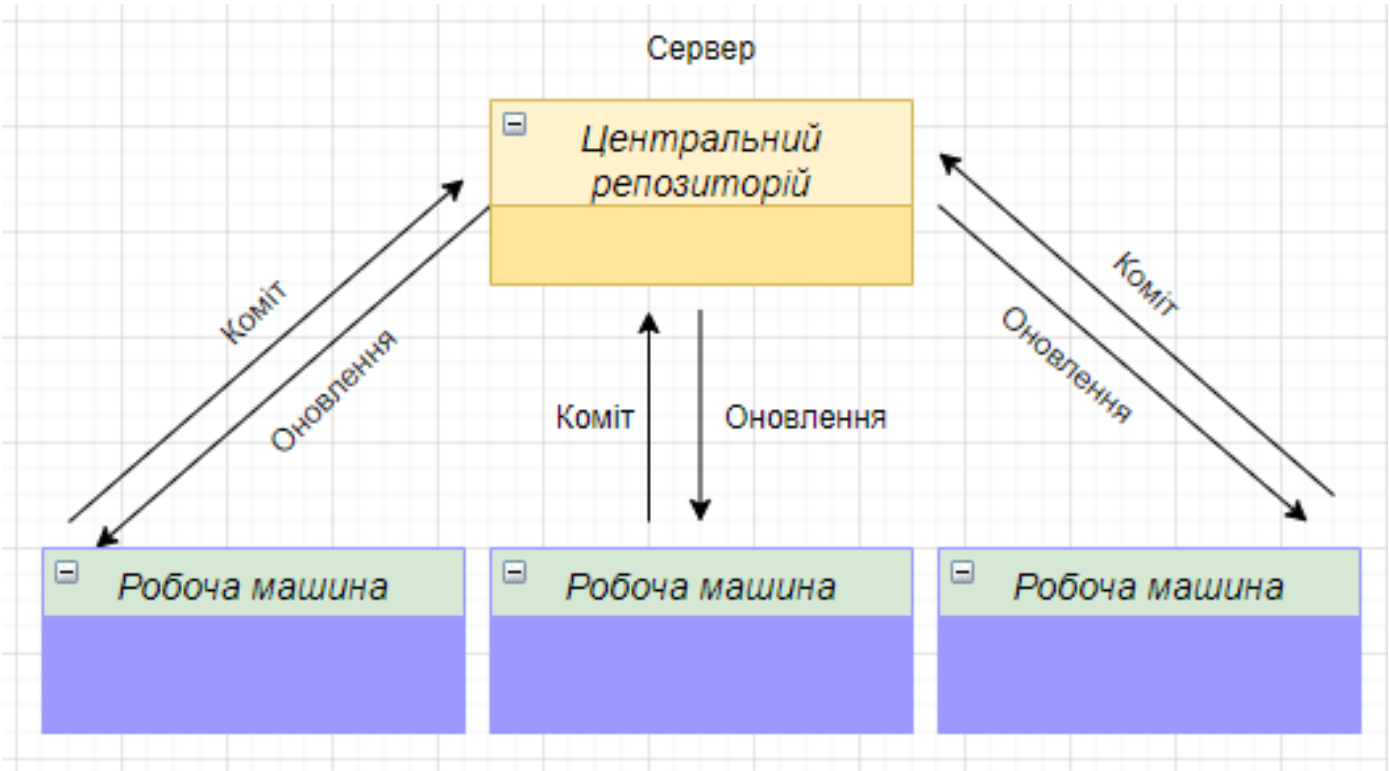

Рисунок 2.1 – Схема роботи централізованої системи контролю версій

На рисунку 2.2 схема роботи розподіленої системи контролю версій при паралельній роботі декількох працівників.

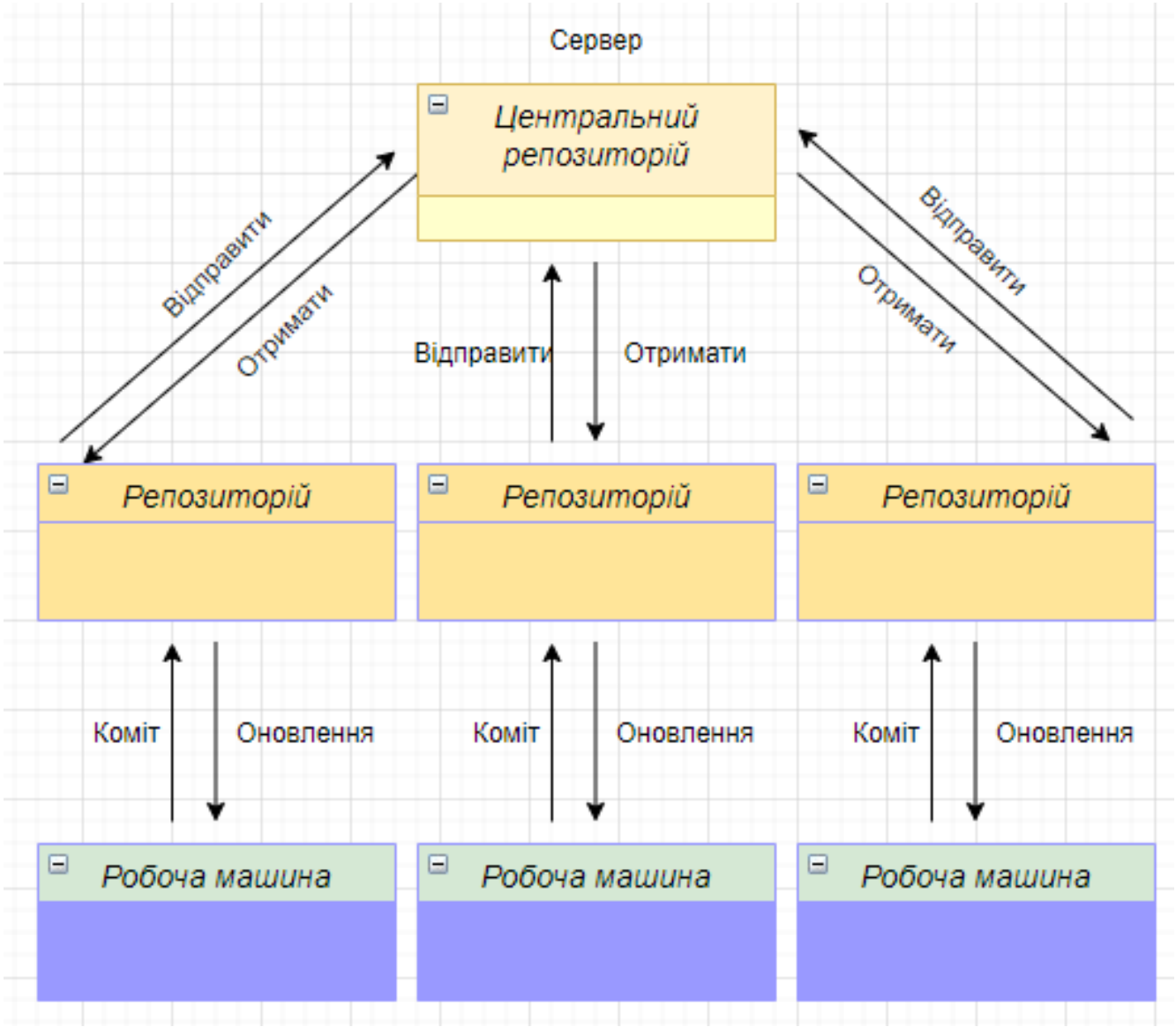

Рисунок 2.2 – Схема роботи розподіленої системи контролю версій

Далі проаналізуємо кожну з них та порівняємо за та проти щодо їх використання.

2.2 Робота системи контролю Git

Git - це безкоштовна та розподілена система контролю версій з відкритим кодом. Мета Git – слідкувати за проектами та файлами, тому що вони можуть змінюються з часом, оскільки різні користувачі можуть вносити правки та зміни до коду. Git, в свою чергу, зберігає усю інформацію про зміни проектних файлів у своєму сховищі. У ньому є коміти до цього проекту або самі посилання на них, які мають назву head. Вся інформація зберігається у підпапці під назвою .git, в тій же папці, що і сам проект, майже у всіх системах ця папка є прихованою.

Отже, в основному Git відстежує зміни, які декілька людей робить в одному проекті, а після цього об'єднує код, де люди працювали над різними частинами, в один проект.

Тобто, якщо хтось допустить помилку при написанні коду, то інші мають можливість дуже легко її відслідкувати за допомогою комітів. Після того як всі частини програми працюють коректо, зміни вносяться до головної гілки, яка називається master.

Потім ви завантажуєте цей код на хмарне сховище. Для того аби поділитися своїм кодом з іншими, щоб він був доступний в Інтернеті, ви можете завантажити його на GitHub.

Особливості Git:

- система контролю версій з відкритим кодом;
- забезпечує потужну підтримку для нелінійного розвитку;
- модель розподіленого сховища;
- сумісний із існуючими протоколами, такими як http, ftp, ssh;
- здатний ефективно обробляти малі та великі проекти;
- криптографічна автентифікація історії;
- періодична явна упаковка об'єктів;
- сміття накопичується до поєднання у єдину вітку.

Переваги використання Git:

– над швидка та ефективна робота;
– крос-платформа;
– зміни коду можна дуже легко та чітко відстежувати;
– легко обслуговується;
– пропонує дивовижну утиліту командного рядка, відому як git bash;
– також пропонує графічний інтерфейс git, де ви можете дуже швидко повторно відскакувати, змінити стан, підписатись, зафіксувати та швидко натиснути код лише за кілька кліків.

Недоліки використання Git:

– складний і великий журнал історії стає важким для розуміння та аналізу;
– не підтримує розширення ключових слів та збереження міток часу.

На рисунку 2.3 можна побачити життєвий цикл Git. В ньому присутні 5 станів: ініціалізація, оновлення, зміни та повернення. Ці стани можуть виконуватись на: робочому місці користувача, у папці з файлами де зберігається проект, на локальному репозиторії юзера та на віддаленому репозиторії де зберігається фінальний варіант проекту.

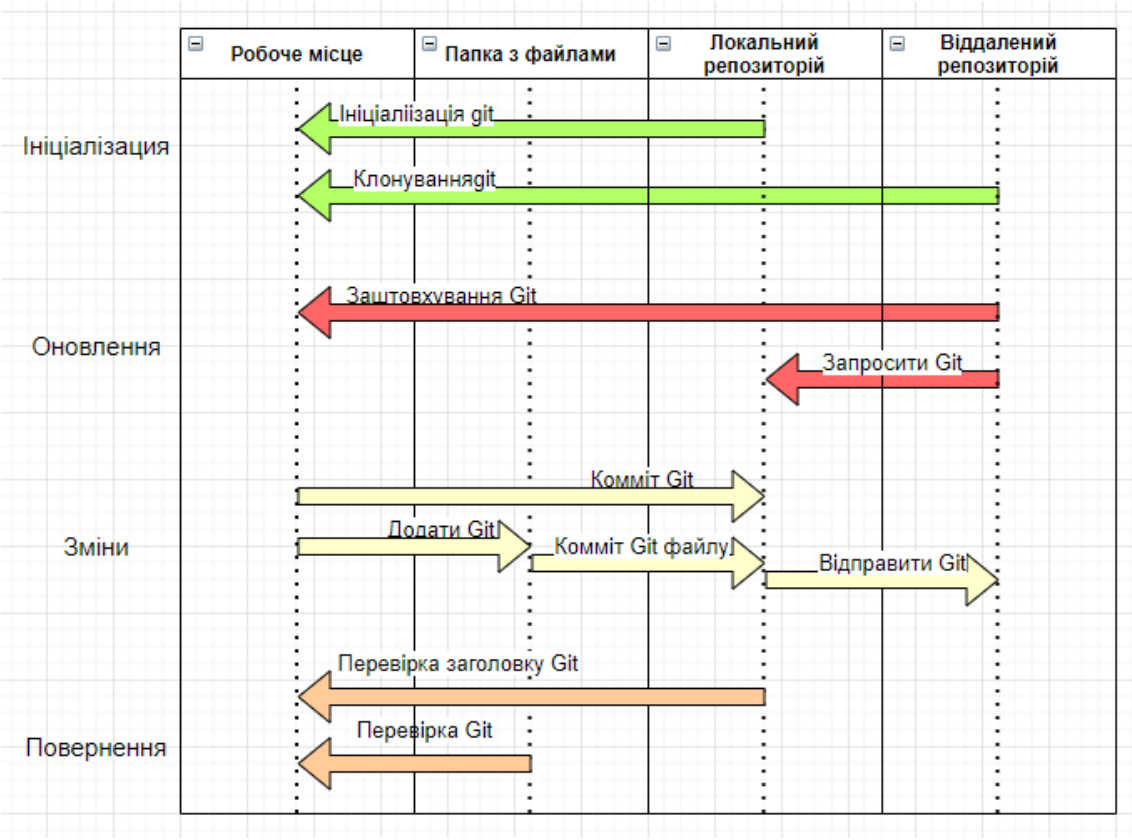

Рисунок 2.3 – Життєвий цикл Git

На рисунку 2.4 зображено у якому стані можуть бути файли. У кожного файлу може бути чотири стани:

- файл не відслідковується – це відбувається в момент додавання нового файлу або видалення старого;
- файл ще не був модифікований;
- файл вже модифікували - під час зміни файлу або коміту;
- файл був інсценізований до відправки.

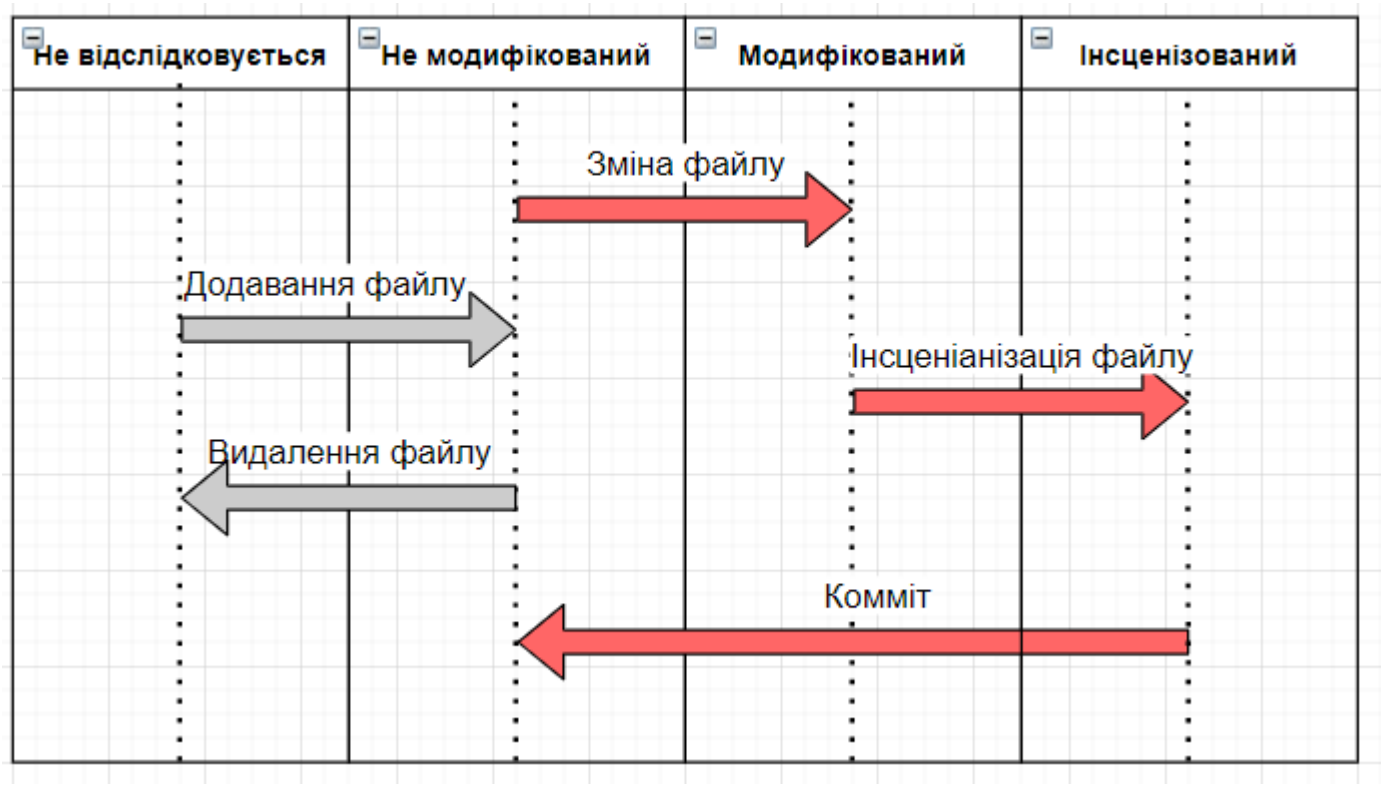

Рисунок 2.4 – Стан файлів

2.3 Робота системи контролю mercurial

Mercurial - це безкоштовна, швидка та легка для розуміння система. Розробник пиши що вона може обробляти проекти будь-якого розміру та пропонує простий інтерфейс. Але насправді це і є головна проблема системи Mercurial, що вона тяжко масштабується з великим проектами та погано або зовсім не підтримує інші додатки.

Особливості Mercurial:

- висока продуктивність та масштабованість;
- розширені можливості розгалуження та злиття;
- повністю розподілений спільний розвиток;
- децентралізована;
- надійно обробляє як звичайний текст, так і двійкові файли;
- володіє інтегрованим веб-інтерфейсом.

Переваги використання Mercurial:

- швидкий і потужний;
- легко вивчити як його використовувати;
- легкий і портативний.

Недоліки використання Mercurial:

- досить проблематичний при використанні з додатковими розширеннями;
- частковий контроль заборонений.

2.4 Робота системи контролю cvs

CVS відстежує модифікації лише залежно від файлу, тоді як SVN відстежує весь коміт як нову редакцію. Існує також проблема одночасного коміту. Не виключено, що двоє людей, які здійснюють спільні дії в CVS, можуть конфліктувати між собою, втрачаючи деякі дані та приводячи проект в невідповідний стан. І нарешті, навколо CVS вже не розробляється нових інструментів, оскільки ця система є досить старою.

Особливості CVS:

- клієнт-серверна модель сховища;
- кілька розробників можуть паралельно працювати над одним проектом;
- клієнт cvs буде постійно оновлювати робочу копію файлу і вимагає втручання вручну лише тоді, коли виникає конфлікт редагування;
- веде історичний знімок проекту;
- анонімний доступ для читання;
- може підтримувати різні гілки проекту;

- виключає символічні посилання, щоб уникнути загрози безпеці;
- використовує техніку дельта-стиснення для ефективного зберігання.

Переваги використання CVS:

- відмінна підтримка між платформами;
- надійний і повнофункціональний клієнт командного рядка дозволяє використовувати потужні сценарії;
- корисна підтримка широкої спільноти cvs дозволяє добре переглядати вебсховище вихідного коду;
- це дуже старий, добре відомий та зрозумілий інструмент.

Недоліки використання CVS:

- немає перевірки цілісності сховища вихідного коду;
- погана підтримка розподіленого контролю джерел;
- не підтримує підписані версії та відстеження об’єднань.

2.5 Робота системи контролю svn

Також відомий як Subversion, SVN представляє одну із найпопулярніших централізованих систем контролю версій. За допомогою централізованої системи всі файли та історичні дані зберігаються на центральному сервері. І розробники фіксують свої зміни безпосередньо в цьому центральному сховищі серверів. Робота складається з трьох частин:

- Trunk - це центр вашого поточного стабільного коду та продукту. Він включає лише перевірений, непошкоджений код;
- Branches - тут ви розміщуєте новий код та функції. Використовуючи копію магістрального коду, члени команди проводять дослідження та розробки у проєкті. Це дозволяє кожному члену команди працювати над вдосконаленими функціями, не порушуючи прогрес один одного;
- Tags - це дублікат гілки в даний момент часу. Теги не використовуються під час розробки, а тільки після закінчення коду гілки. Позначення коду тегами полегшує його перегляд і, при необхідності, повернення коду.

Особливості SVN:

- клієнт-серверна модель сховища;
- каталоги мають версії;
- операції копіювання, видалення, переміщення та перейменування також мають версії;
- версійні символічні посилання;
- версійні метадані у вільній формі;
- ефективне просторове двійкове диференційне зберігання;
- розгалуження не залежить від розміру файлу;
- інші функції - відстеження злиття, блокування файлів, автономна робота сервера.

Переваги використання SVN:

- має перевагу хороших інструментів графічного інтерфейсу, таких як tortoisesvn;
- підтримує порожні каталоги;
- простота налаштування та адміністрування;
- добре інтегрується з windows, провідними інструментами ide та agile.

Недоліки використання SVN:

- досі містить помилки, пов’язані з перейменуванням файлів та каталогів;
- недостатня кількість команд управління сховищем;
- повільна порівняльна швидкість;
- не підтримує підписані редакції.

2.6 Висновки до розділу

Аналізуючи все вище описане, обрана система Git за рахунок її популярності, швидкій роботі з великими проектами, крос-платформеності, великої кількості підтримуючих додатків тощо.

3 ОГЛЯД ІСНУЮЧИХ СЕРВІСІВ ДЛЯ РОБОТИ ІЗ GIT

Сьогодні доступний дуже різноманітний ряд клієнтських інтерфейсів Git для різних операційних систем, включаючи Android, iOS, Mac, Linux та Windows. Git - одна з найкращих та найбільш бажаних доступних систем контролю версій (VCS). У багатьох проектах реалізовані сховища Git для зберігання та управління кодами, незалежно від того, для великого підприємства це чи для малого проекту.

Саме тому з кожним днем розроблюється все більше GUI інтерфейсів та сервісів. Я вибрав 3 найбільш популярних сервіса або клієнта для роботи з GIT: GitHub, SourceTree, GitKraken.

3.1 Порівняння сервісів на основі різноманітності платформ

GitHub можна розглядати як хмарне сховище, він дозволяє програмістам обмінюватися кодами, кодами пошуку, кодами завантаження, переглядати проекти, активність розробника тощо через зручний веб інтерфейс.

Додатково до веб інтерфейсу існує GUI GitHub Desktop – офіційний продукт від GitHub, він створений для роботи розробників через API самого GitHub. Однак офіційного додатку для телефона немає, як для iOs так і для Android.

SourceTree має застосунок лише для Windows або MacOs і так само не має свого мобільного додатку. Аналогічно до SourceTree так само працює і GitKraken.

Тобто жоден з найпопулярніших сервісів для роботи з Git не мають свого додатку для телефона, тільки GitHub має свою мобільну версію сайту, що означає, якщо потрібно подивитись проект, код, активність тощо, з телефона, люди вже зараз використовують саме GitHub мобільну версію сайта. Так само GitHub має найбільшу базу користувачів та свій власний та швидкий API який ідеально підходить для створення свого застосунка для телефона.

Перераховуючи все вище сказане, я вибрав саме GitHub для розробки свого мобільного додатку.

Аналіз веб інтерфейсу GitHub

Проаналізуємо веб версію. Особистий обліковий запис GitHub є безкоштовним і має необмежене сховище. Ви можете створювати як загальнодоступні, так і приватні проекти, скільки вам буде потрібно. Крім того, група GitHub рекомендує зберігати сховища невеликими, в ідеалі менше 1 ГБ. Це тому, що менші сховища швидше клонуються і простіше працювати з ними та обслуговувати. Окремі файли у сховищі суворо обмежені обмеженням до 100 МБ. Цю панель можна побачити на рисунку 3.1.

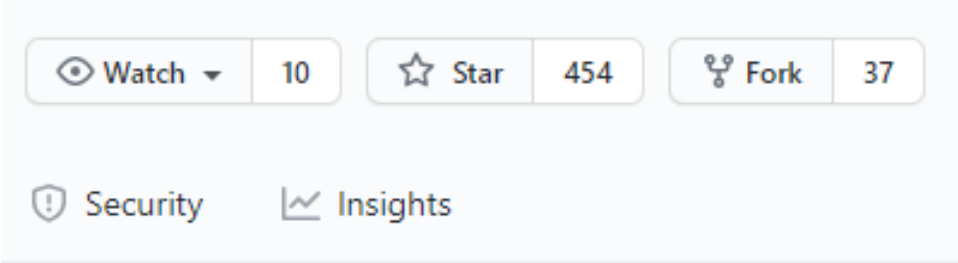

Рисунок 3.1 - Випадкова інформаційна панель проекту.

Watch - параметри сповіщення. Він пропонує "Не дивитись", "Лише релізи", "Дивитися" та "Ігнорувати".
Star – кількість людей яким сподобався цей проект.
Fork - скопіювати проект у свій обліковий запис. Ця функція використовується для модифікації проекту, і це важливо, коли ви кодуєте в команді.

Інформація про репозиторій

Репозиторій може розглядатися як основний файл проекту, в якому ви можете зберігати всі пов’язані папки та файли (включаючи зображення та відеокліпи розміром <100 Мб). Дану панель можна побачити на рисунку 3.2.

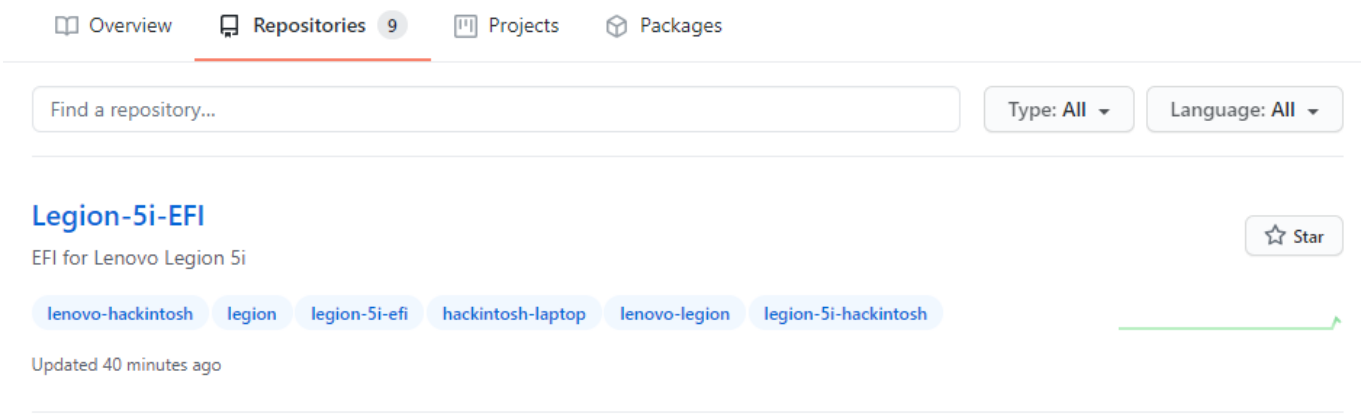

Рисунок 3.2 - Випадкова панель проекту.

OverView - сторінка, на якій відображаються ваші закріплені проекти, лічильники змін та діяльність щодо їх. Repositories або projects - ваші збережені папки проектів. На GitHub іх може бути безліч. Packages - місце для зберігання ваших створених пакетів NPM. NPM розшифровується як диспетчер пакетів Node.

Різниця між репозиторієм і проектом

В основному, вони однакові, просто інформаційна панель проекту має більше функцій управління, що дозволяє вам легше працювати над своїм проектом (проекти з відкритим кодом або проекти компанії).

Переваги використання GitHub

Ось кілька причин для цього:

- знання github - це плюс, якщо є бажання приєднатися до компаній, оскільки більшість використовує github як сховище проектів;
- github може бути однією з частин резюме. деякі інтерв'юери будуть відвідувати github, щоб краще зрозуміти здібності людини;
- github буде відстежувати кожен код, який ви передали в хмару, а це означає, що ви можете легко знати, які коди були змінені (функція до і після). це надзвичайно корисно, коли йде работа з командою;

– на рисунку 3.3 функція відслідковування частоти комітів. вона допомагає зрозуміти активність людини в проекті. на ній зображується діяльність розробника кожного дня і місяця.

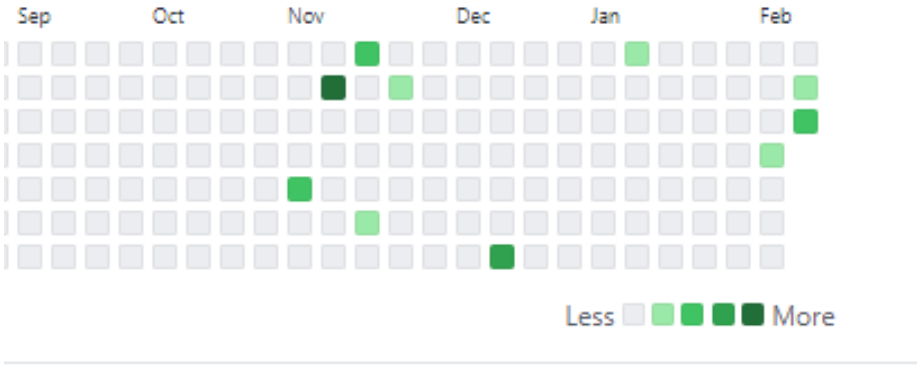

Рисунок 3.3 - Діяльність щодо внесків на сторінці огляду

3.2 Структура технології API

API - це сукупність ресурсів та інструментів в операційній системі, які необхідні в роботі розробників для створення різних програм та їх взаємодії з іншими службами. На рисунку 3.4 зображено схему ієрархії API в програмному продукті.

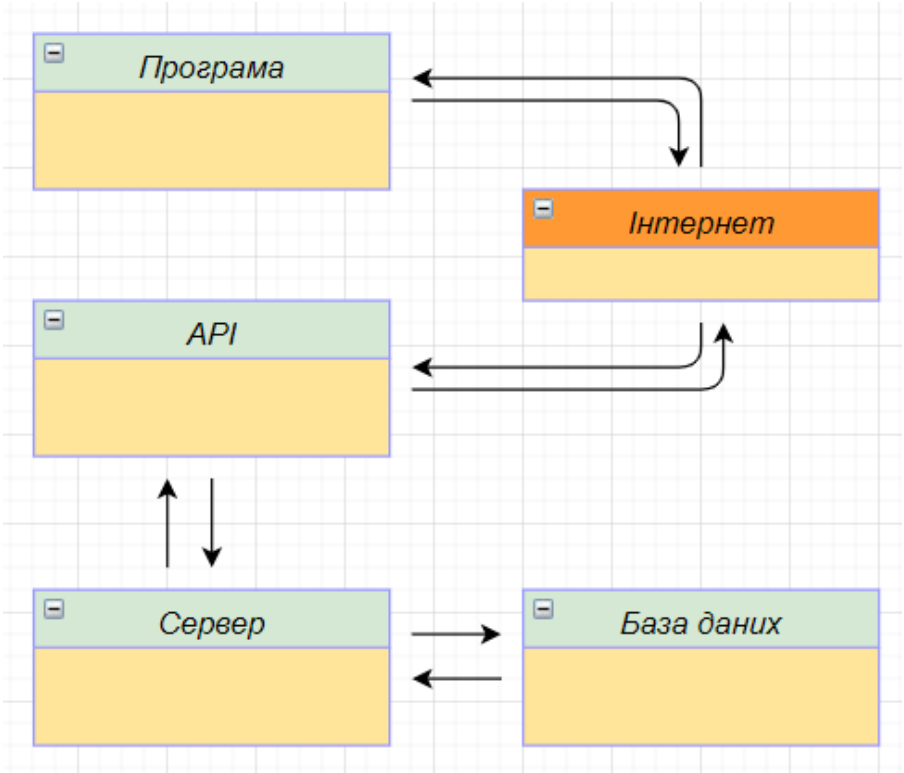

Рисунок 3.4– Позиція API в програмному проекті

API – ресурс, завдяки якому окремі програми можуть працювати та обмінюватись інформацією між собою. Важливість API важко переоцінити, адже без нього було б важко вивчити зв’язки між окремими програмними інструментами. API показує, які дії можна виконувати над різними обьектами та повну структуру роботи з ними.

Застосування API – наприклад, представимо такий додаток - банкомат. В той час як людина підходить до банкомату, вона хоче, щоб він дозволив їй отримати доступ до свого банківського рахунку та здійснити транзакцію, зняття готівки наприклад. Як і в звичайному банкоматі, програмний додаток надає функцію, але робить він це не сам по собі - йому потрібно спілкуватися як з користувачем, людиною, так і з “банком”, до якого він отримує доступ. Web-програма, програма для мобільного або внутрішня програма будь якого пристрою схожа на машину, яка вирішує задану проблему. В даному випадку програмним додатком може бути програма для клієнта, якот серверне забезпечення, яке і спрямовує усі запити до бази даних банку.

API робить можливим спілкування банкомату з банком. Програмування - це інженерна частина для програмного забезпечення програми, яка перетворює введені дані у вихідні. Інакше кажучи, він переводить запит користувача на отримання грошей в базу даних банку, перевіряє, чи достатньо готівки для зняття заданої суми, якщо банк надає дозвіл, банкомат повідомляє банку, скільки користувач зняв, щоб банк міг оновити баланс в базі даних.

Інтерфейс користувача - це спосіб взаємодії з додатком. У випадку з банкоматом це екран, клавіатура та слот для готівки - де відбувається вхід і вихід. Людина вводить свій PIN-код, потім скільки готівки вона хочемо зняти. Інтерфейси - це спосіб спілкування з машиною. З API-інтерфейсами це приблизно те саме, лише йде заміна користувачів програмним забезпеченням.

Якщо веб-сайт або програма пропонує API, це означає, що їх розробники створили набір URL-адрес, які повертають відповіді на чисті дані.

Саме через API від GitHub ми будемо спілкуватись з ним та отримувати дані користувача, інформацію про проекти, код програм тощо.

3.3 Опис різних видів API

Не всі API робляться на 100% відкритим для всіх. API для Партнерів та для приватних користувачів використовуються тільки внутрішніми або авторизованими користувачами. Такий формат дозволяє компаніям зберігати контроль над тим, хто використовує всю інформацію та інші ресурси, які пов'язані з її продуктами.

Види API:

- приватні арі створені лише для використання в середині компанії. такий варіант допомагає компаніям зберегти контроль над їх системним арі;
- партнерський арі . він ділиться з конкретними партнерами з якими ведуться ділові відносини. такий вид може забезпечити більші доходи без додаткової шкоди для якості;
- загальнодоступний арі який доступний для кожного. він дозволяє будь-якій особі розробити програму, яка взаємодіє з арі сервісу.

В випадку з GitHub ми будемо використовувати загальнодоступний API, оскільки я ніяк не відношусь до інших категорій.

3.4 Можливості API GitHub

GitHub API вважається єталоном того, як повинен взагалі виглядати доступ до сервісів компаній через API. Всього у компанії було 3 версії API: Rest API v1, Rest API v2, Rest API v3. Попередні версіях API вже не підгримуються, саме тому я буду використовувати версію v3.

До яких розділів є доступ через GitHub REST API v3: Дії, Діяльність, Застосунки, Виставлення рахунків, Перевірки, Сканування коду, Смайлики, Адміністрація GitHub Enterprise, База даних Git, Gitignore, Взаємодія, Проблеми, Ліцензії, Водяний знак, Міграції, Організації, Проекти, Завантаження, Обмеження лімітів, Реакції, Репозиторії, Пошук, Секретне сканування, Правила, Користувачі, Дозволи для додатку.

4 ВИБІР ТЕХНОЛОГІЙ ТА ПОРІВНЯННЯ С АНАЛОГАМИ

Оскілки зараз дуже багато різних платформ якими користуються люди, то спочатку потрібно вирішити під яку платформу буде розроблюватись програма.

4.1 Обґрунтування вибору розробки мобільного додатку

Мобільна ера настала. Сьогодні кількість мобільних користувачів перевищує кількість користувачів настільних ПК! Вибір між мобільними додатками та веб-сайтами залежить від їх вартості, зручності використання, необхідних функцій та аудиторії, яку вони обслуговують.

З огляду на це, дослідження показують, що користувачі більше віддають перевагу мобільним додаткам, ніж мобільним веб-сайтам. Це і є вагома причина для створення мобільних додатків для звернення до потенційних (і існуючих) клієнтів. Далі я розгляну декілька причин чому я вибрав розробку саме мобільного додатка.

1. Персоналізація - це пропонування індивідуального спілкування користувачам на основі їх інтересів, місцезнаходження, поведінки використання тощо. Мобільні програми можуть дозволити користувачам встановити свої уподобання на самому початку, виходячи з того, що користувачі можуть отримувати індивідуальний вміст.

   Додатки також можуть відстежувати та спостерігати за залученням користувачів та використовувати їх, щоб пропонувати користувацькі рекомендації та оновлення для користувачів.

   Крім того, вони можуть визначати місцезнаходження користувачів у режимі реального часу для надання географічного контенту. Однак покращення взаємодії з користувачем - не єдина мета, якою служить персоналізація. Це також може допомогти покращити коефіцієнт конверсії додатків.
2. Сповіщення бувають двох типів: push-повідомлення та сповіщення в додатку. Вони обидва є цікавими альтернативами для спілкування з користувачами додатків менш нав'язливо.

Сповіщення в додатку - це сповіщення, які користувачі можуть отримувати лише тоді, коли вони відкрили програму. Натомість push-сповіщення - це ті сповіщення, які користувачі можуть отримувати незалежно від будь-якої діяльності, яку вони виконують на своєму мобільному пристрої. Були випадки, коли push-носій сповіщень забезпечував коефіцієнт кліків 40%.

3. Мобільні програми мають перевагу використання таких функцій мобільного пристрою, як камера, список контактів, GPS, телефонні дзвінки, акселерометр, компас тощо.

   Ці функції можуть також зменшити зусилля, які доводиться докладати користувачам. Наприклад, авто заповнення форм, пошук друзів з телефонної книги і тд. Функції пристрою можуть значно скоротити час, який користувачі виконують для виконання певного завдання в програмі.
4. Можливість користуватися офлайн. Це, мабуть, найбільш принципова різниця між мобільним веб-сайтом та додатком. Хоча додатки теж можуть вимагати підключення до Інтернету для виконання більшості своїх завдань, вони все одно можуть пропонувати основний вміст та функціональність користувачам в автономному режимі.
5. Дизайн. У мобільних веб версіях присутній тільки один варіант, який нам дає розробник самого сайту. В той час як у мобільному додатку ми можемо зробити будь-який персоналізований дизайн, оскільки ми отримуємо інформацію по API, а в якому форматі вона буде виведена на екран вирішуємо ми.
6. Згідно статистики, користувачі мобільних пристроїв витрачають 86% свого часу на мобільні програми та лише 14% часу на мобільних веб-сайтах. Більше того, середній час, який користувачі витрачають на мобільні додатки, також збільшується - за 2020 рік він виріс на 21%.
7. Добре розроблений мобільний додаток може виконувати дії набагато швидше, ніж мобільний веб-сайт. Програми зазвичай зберігають свої дані локально на мобільних пристроях, на відміну від веб-сайтів, які зазвичай використовують веб-сервери. З цієї причини отримання даних відбувається швидше в мобільних

додатках. Також програми можуть додатково економити час користувачів, зберігаючи їхні уподобання та використовуючи їх для здійснення активних дій від імені користувачів.

З огляду на всі ці пункти, я зробив висновок, що мобільній додаток найкращий варіант для більшої частини користувачів.

4.2 Обґрунтування вибору мобільної системи iOs

Останнім кроком до початку розробки залишилось вибрати для якої саме системи краще розроблювати програму. Вибір між iOs або Android. Спочатку вибір платформи залежить від того яку аудиторію потрібно охопити. Оскільки це додаток для розробників, то моя аудиторія – це середній вік. І саме користувачі iOS, як правило, молодші, з більшою часткою ринку віком від 18 до 24 років. Додатково до цього, згідно статистики, користувачі Apple частіше мають вищу освіту.

Розробити для iOS швидше, простіше і дешевше - за деякими оцінками, час розробки Android на 30–40% довший. Однією з причин, чому iOS легше розробляти, є код. Програми для Android, як правило, написані на Java, мовою, яка передбачає написання більше коду, ніж Swift, офіційною мовою програмування Apple. Інша причина полягає в тому, що Android - це платформа з відкритим кодом. Відсутність стандартизації означає збільшення кількості пристроїв, компонентів та фрагментації програмного забезпечення. Закрита екосистема Apple означає, що ви розробляєте для кількох стандартизованих пристроїв та операційних систем.

Тобто, платформа iOs і мова програмування SWIFT найкраще підходять для моїх цілей.

4.3 Аналіз аналогів на iOs

В магазині додатків для iOs я знайшов лише 2 аналоги для сервісу GitHub. Проаналізуємо їх переваги та недоліки, щоб зрозуміти, що можна запозичити для своєї програми, а що потрібно переробити.

Недоліки аналогів:

- перша причина чому вони потребують переробки, вони написані 2 роки тому – це означає, що версія мови якою вони були написані 4.0-4.2. проте в минулому році в реліз вже вийшла нова версія мови програмування swift 5.1;
- тільки 2 роки тому вийшла версія github rest api v3 і одразу були написані ці додатки, в той же час після того яка ця сира версія доповнювалась та видо змінювалась, оновлень у програм не було;
- в них відсутні такі розділи як: спільний доступ до посилань, історія переглядів, сучасний пошук с фільтрами мови для репозиторіїв та користувачів, різні теми які підлаштовуються під головну тему смартфона.

При використанні були помічені такі баги як:

- запит на завантаження часто не вдається з першої спроби, далі все працює;
- надсилаючи коментар, іноді виникає помилка, що операція не вдалася, але коли сторінка оновлюється, коментар насправді присутній;
- вікно огляду закривається само по собі із за чого може бути втрачений коментар від користувача який не був відправлений;
- один з них чомусь потребує останньої версії операційної системи, при тому что близько 20% користувачів не оновлюються до неї, при тому немає жодної причини не дозволити користуватись додатком на старіших версіях системи;
- вони не оптимізовані під будь-які розміри дисплеїв;
- побудовані на react native, із-за цього витрачається набагато більше часу для оптимізації проекту під різні смартфони, тому що він розроблюється одразу

для декількох платформ. саме тому втрачається швидкість, присутня недостатня оптимізація під смартфони Apple.

Переваги аналогів:

- в одному з них присутня модель розробки MVVM (Model-View-ViewModel) що дозволить в майбутньому простіше розроблювати, тестувати та модифікувати додаток.

4.4 Обґрунтування вибору архітектури MVVM

Архітектурний шаблон є загальним рішенням для багаторазового використання часто зустрічається проблеми в архітектурі програмного забезпечення в заданому контексті. Архітектурні зразки схожі на зразки програмного дизайну, але мають більш широкий спектр.

На використання слова "шаблон" в індустрії програмного забезпечення вплинули концепції, що виражаються в традиційній архітектурі, наприклад, де обговорювалась практика з точки зору створення лексикону шаблонів, що спонукало практикуючих працівників розроблювати свої шаблони проектування програмного коду. Найбільш використовувані паттерни при розробці GUI клієнтів: MVC та MVVM. Саме тому я вирішив проаналізувати кожний з них та вибрати найбільш зручний варіант для свого проекту.

Паттерн MVC

Фреймворк MVC - це архітектурний шаблон, який розділяє програми на три основні логічні компоненти Model, View та Controller. Звідси і абревіатура MVC. У цій архітектурі компонент побудований для обробки конкретних аспектів розробки програми.

MVC відокремлює бізнес-логіку та рівень презентації один від одного. Цей архітектурний шаблон в основному використовується для графічних користувальницьких інтерфейсів (GUI). На рисунку 4.1 зображено модель MVC.

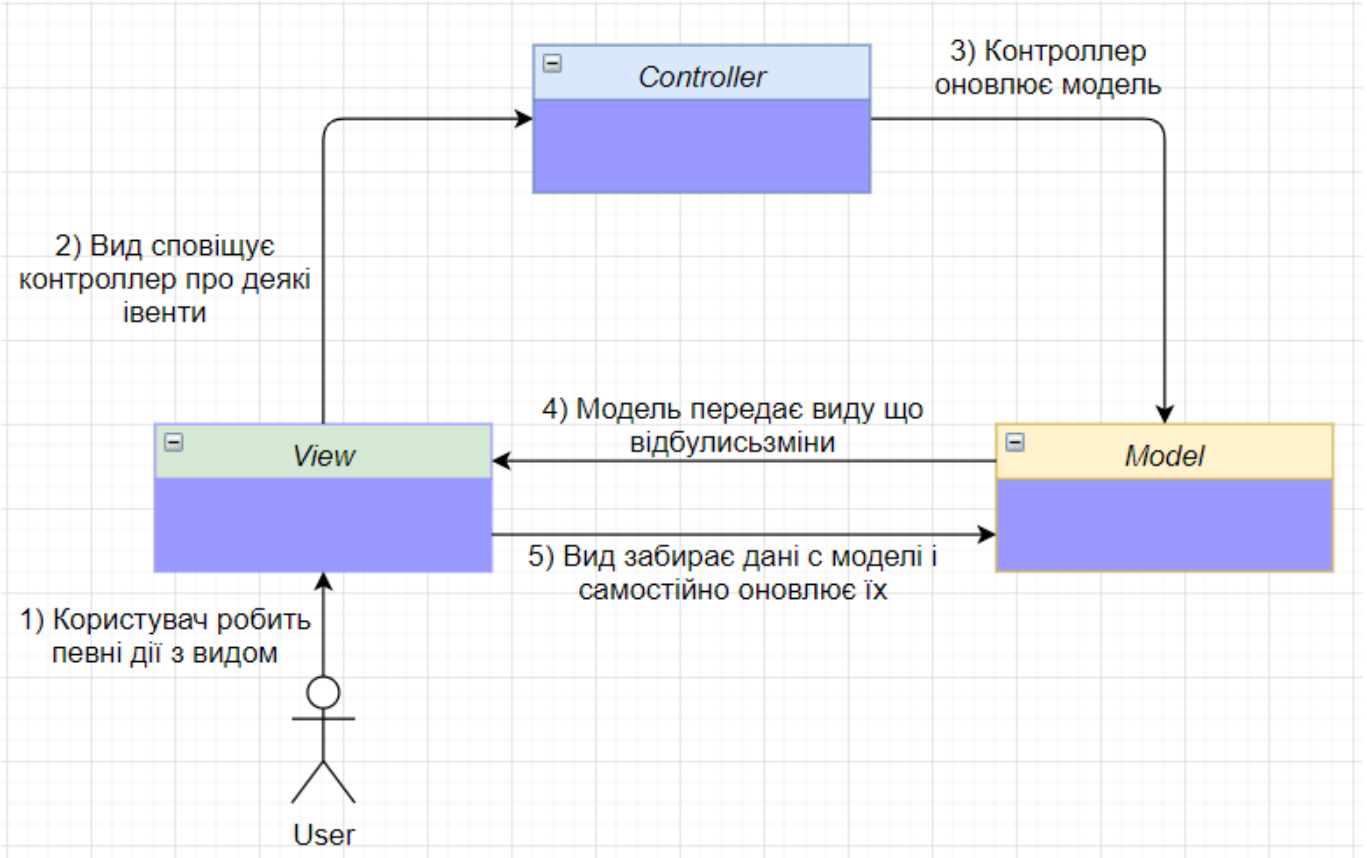

Рисунок 4.1 – Модель MVC архітектури

Три важливі компоненти MVC:

- модель: вона включає всі дані та пов'язану з ними логіку;
- перегляд: подання даних користувачеві або обробка взаємодії з користувачем;
- контролер: інтерфейс між компонентами model і view.

Розглянемо кожний з них:

Компонент моделі зберігає дані та пов'язану логіку. Він представляє дані, які передаються між компонентами контролера. Наприклад, об’єкт Controller допомагає отримати інформацію про клієнта з бази даних. Він маніпулює даними та відправляє їх назад до бази даних або використовує для рендеру тих самих даних.

Перегляд - це та частина програми, яка представляє подання даних. Представлення створюються за даними, зібраними з даних моделі. Представлення вимагає від моделі надати інформацію, щоб вона повторно надіслала вихідні дані користувачеві. Представлення також представляє дані з діаграм, діаграм та таблиць. Наприклад,

будь-яке представлення клієнта включатиме всі компоненти інтерфейсу, такі як текстові поля, випадаючі меню тощо.

Контролер - це та частина Програми, яка обробляє взаємодію користувача. Контролер інтерпретує введення миші та клавіатури від користувача, повідомляючи Модель та Вигляд про необхідність зміни. Контролер надсилає команди Моделі для оновлення її стану (наприклад, Збереження конкретного документа). Контролер також надсилає команди до пов'язаного подання для зміни презентації представлення (наприклад, прокручування певного документа).

Важливі особливості MVC:

- високотестуваний, розширюваний і широко підключаємий фреймворк;
- можна використовувати існуючі функції, пропоновані asp.net, django, jsp тощо;
- повний контроль над html, а також url-адресами;
- підтримує тест-драйв розробку (tdd);
- ця архітектура пропонує розділення логіки;
- дозволяє маршрутизацію url-адрес, зручних для seo.

Переваги використання MVC:

- простіша підтримка клієнтів нового типу;
- розробка різних компонентів може виконуватися паралельно. це дозволяє уникнути складності шляхом поділу програми на окремі (mvc) блоки;
- використовує лише шаблон переднього контролера, який обробляє запити веб-додатків за допомогою одного контролера;
- пропонує найкращу підтримку для тестової розробки;
- добре працює з веб-програмами, які підтримуються великими командами веб-дизайнерів та розробників;
- всі класифіковані об’єкти не залежать один від одного, так що ви можете перевірити їх окремо;
- mvc дозволяє логічно групувати пов'язані дії на контролері разом.

Недоліки використання MVC:

- бізнес-логіка змішується з інтерфейсом користувача;

- важко повторно використовувати та впроваджувати тести;
- відсутність формальної підтримки;
- підвищена складність та неефективність даних;
- складність використання mvc із сучасним інтерфейсом користувача;
- існує потреба в тому, щоб кілька програмістів проводили паралельне програмування;
- потрібні знання багатьох технологій.

Паттерн MVVM

Архітектура MVVM сприяє відокремленню розвитку графічного інтерфейсу користувача за допомогою мови розмітки або коду графічного інтерфейсу. Повною формою MVVM є Model – View – ViewModel. Модель подання MVVM є перетворювачем значень, що означає, що відповідальність моделі подання полягає у виведенні об'єктів даних із моделі таким чином, що об'єктами легко керувати та відображати. На рисунку 4.2 зображено модель MVVM.

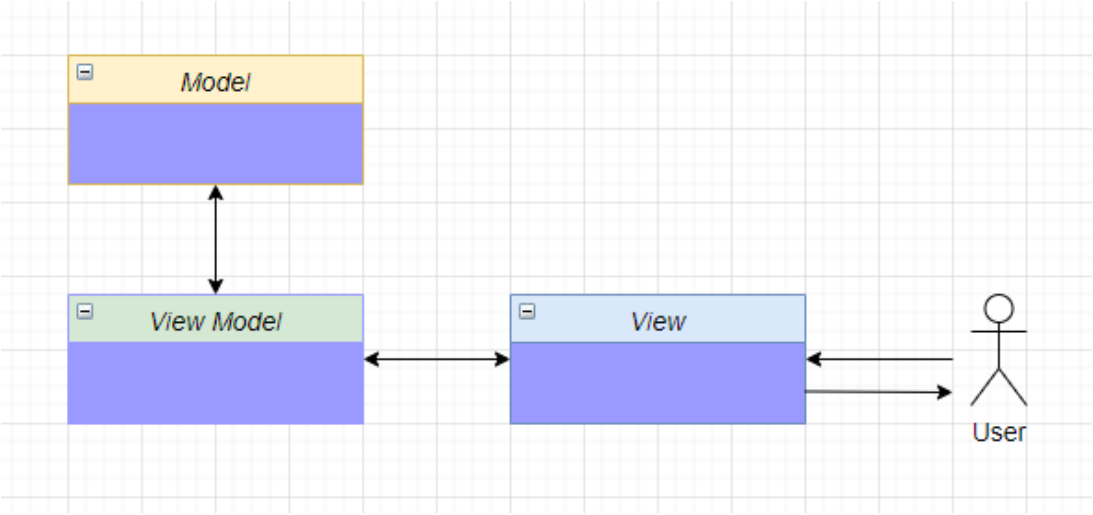

Рисунок 4.2 – Модель MVVM архітектури

Архітектура MVVM пропонує двостороннє прив'язку даних між видом та моделлю перегляду. Це також допоможе автоматизувати розповсюдження модифікацій усередині View-Model до подання. Модель перегляду використовує шаблон спостерігача для внесення змін у модель перегляду.

Розглянемо кожний з компонентів:

- модель зберігає дані та пов'язану логіку. вона представляє дані, які передаються між компонентами контролера або будь-якою іншою пов'язаною бізнес-логікою. наприклад, об’єкт controller отримає інформацію про учня зі шкільної бази даних. він маніпулює даними та відправляє їх назад до бази даних або використовує для рендеру тих самих даних.
- подання розшифровується як компоненти інтерфейсу, такі як html, css, jquery тощо. у mvvc подання шаблону несе відповідальність за відображення даних, отриманих від контролера як результат. цей вигляд також перетворює моделі (моделі) в інтерфейс користувача (ui).
- модель подання відповідає за представлення функцій, команд, методів для підтримки стану подання. також відповідальним є використання моделі та активація подій у поданні.

Важливі особливості MVVM:

- MVVM створений для настільних додатків з можливостями прив'язки даних - XAML та інтерфейсом INotifyPropertyChanged;
- якщо ви хочете зробити модифікацію в View-Model, View-Model використовує шаблон спостерігача;
- шаблон MVVM в основному використовується WPF, Silverlight, nRoute тощо.

Переваги використання MVVM:

- бізнес-логіка відокремлена від інтерфейсу користувача;
- легкий в обслуговуванні та тестуванні;
- легко повторно використовувати компоненти;
- тісно пов'язана архітектура;
- ви можете писати модульні тестові приклади як для моделі перегляду, так і для рівня моделі, без необхідності посилання на вигляд.

Недоліки використання MVVM:

- забагато коду у контролерах;
- деякі користувачі вважають, що для простих інтерфейсів архітектура mvvm може бути надмірною;

- не пропонує щільного зв’язку між видом та моделлю виду.

Головна різниця та висновки

Важлива різниця між MVVM та MVC.

Архітектура MVC:

- контролер - це точка входу до програми;
- один до багатьох, відношення між controller і view;
- перегляд не має посилання на контролер;
- mvc - стара модель;
- важко прочитати, змінити, протестувати та повторно використовувати цю модель;
- компонент mvc model можна протестувати окремо від користувача.

Архітектура MVVM:

- вигляд є точкою входу до програми;
- один до багатьох, відношення між view та viewmodel;
- у поданні є посилання на модель перегляду;
- mvvm - відносно нова модель;
- процес налагодження буде складним, коли ми маємо складні прив'язки даних;
- легко для окремого модульного тестування, а код керується подіями.

Аналізуючи все вищесказане я вибрав модель MVVM для написання свого додатку із-за її сучасності, простоті в підтримці, модульності тощо.

4.5 Обґрунтування вибору мови програмування SWIFT

Писати для iOS можна двома мовами програмування: Swift або Objective-C, в моєму випадку я вибрав Swift, оскільки мова програмування Swift швидко стала однією з найбільш швидкозростаючих мов в історії. Swift дозволяє легко писати програмне забезпечення, яке неймовірно швидке та безпечне. А найголовніше дуже добре

оптимізоване для техніки від компанії Apple, оскільки ця мова і є продукт від компанії. Так само, Swift можна застосовувати на ще більш широкому діапазоні платформ, від мобільних пристроїв до робочого столу або хмари.

Переваги використання мови Swift:

- swift працює дуже швидко - майже так само швидко, як c++. а з найновішими версіями xcode році він ще краще оптимізований для техніки від компанії apple;
- swift легше читати і легше вчити, ніж objective-c. objective-c старше на майже 30 років, і це означає, що він має більш незграбний синтаксис. swift впорядковує код і більше нагадує читабельну англійську мову, подібну до таких мов, як c #, c ++, javascript, java та python. розробники, які вже вміють програмувати на цих мовах, можуть розраховувати на те, що досить швидко розберуться з swift. крім того, swift вимагає менше коду. у той час як objective-c є багатослівним, коли мова йде про маніпулювання рядками, swift використовує інтерполяцію рядків, без заповнювачів чи токенів;
- кращі компілятори – це те саме, що кращий досвід кодування для програмістів. swift побудований на віртуальній машині низького рівня (llvm), компілятори, який використовується такими мовами, як scala, ruby, python, c # та go. llvm швидший та розумніший, ніж попередні компілятори c, тому більше робочого навантаження передається від програміста до xcode та компілятора;
- покращене управління пам’яттю. „витік пам’яті” може відбуватися в об’єктно-орієнтованому програмуванні та додатках, і вони зменшують доступну пам’ять для запуску програми, що спричиняє збій програми.. якщо під час витоку пам’яті використовується занадто багато пам’яті, операційна система може закрити програму. щоб виправити це, swift підтримує arc у всіх api, і ця стабільність означає, що програмістам доводиться витрачати менше часу, зосереджуючись на управлінні пам'яттю.

4.6 Висновки до розділу

Отже, для написання програми буде використовуватися мова Swift 5.1., що надає можливості у майбутньому дуже активно модифікувати свій продукт.

Також буде використовуватись модель MVVM та RxSwift оскільки така архітектура додатку краще підходить для модульних програм. REST API v3 буде використовуватись для спілкування с сервісом GitHub. Звичайно для такого додатку потрібен гарний рівень безпеки, тому без OAuth2 аунтефікації не обійтись, для неї я буду використовувати GraphQL API v4.

5 ВИКОРИСТАНІ ПРОГРАМНІ ЗАСОБИ РОЗВ'ЯЗАННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

Настав час коли вже не достатньо просто написати програмний код, потрібно розробити дизайн продукту, зробити програму для декількох платформ. І щоб спростити собі життя як у написанні так і у додаванні різних модулів, можна використовувати готові бібліотеки які були розроблені іншими людьми для використання у майбутньому.

Так само із-за масштабності усіх сучасних проектів для них потрібна спеціальна структура, щоб самі розробники не заплутались у великій кількості файлів там структур.

5.1 Опис основних технології

В своєму проекті я використав максимальну кількість нових для себе технологій та методик розробки програмного продукту, проаналізуємо кожну з них.

1) Архітектура проекту MVVM з використанням RxSwift.

Model-View-ViewModel (MVVM) - це структурний шаблон дизайну, який розділяє об'єкти на три окремі групи:

- моделі містять дані програми;
- подання відображають візуальні елементи та елементи керування на екрані. Зазвичай вони є підкласами UIView;
- моделі подання перетворюють дані про дану модель у значення, які після цього відображаються на екрані. Зазвичай це класи, тому їх можна передавати як посилання.

Наприклад, ми можемо використовувати модель, щоб перетворити дату в рядок у форматі дати, десяткову - у рядок у форматі валюти чи багато інших корисних перетворень. Цей шаблон особливо добре доповнює MVC. MVVM - це чудовий спосіб

зменшити масштабні контролери перегляду, які потребують декількох перетворень моделі в перегляд.

RxSwift є частиною набору мовних інструментів ReactiveX (Rx), які охоплюють різні мови програмування та платформи. RxSwift - це рамка для взаємодії з мовою програмування Swift. Основні переваги RxSwift: асинхронність спрощена Декларативним кодом, спрощена багато потоковість, чистіший код та архітектура, багатоплатформність та можливість компонування компонентів.

2) Координатори потоку.

Координатор - це звичайний об'єкт, який обробляє навігаційний потік. Кожен координатор має поле дочірнього координатора, яке використовується для збереження під потоків навігації.

Контролер потоку - це підклас UIViewController, який обробляє навігаційний потік за допомогою функції стримування ViewController.

3) REST API v3 – Moya and ObjectMapper.

При використанні Alamofire щоб абстрагувати доступ до URLSession та всіх тих неприємних деталей потрібно писати спеціальні рівні абстракції мережі. Щоб уникнути цього та спростити процес розробки можна використовувати Moya бібліотеку. Основна її ідея полягає в тому, що нам потрібен рівень абстракції мережі, який достатньо інкапсулює фактично виклик Alamofire безпосередньо.

Особливості Moya: перевірка часу компіляції щодо правильності доступу до кінцевої точки API, дозволяє визначити чітке використання різних кінцевих точок із пов'язаними значеннями перерахування, дуже легко реалізує модульне програмування.

ObjectMapper - це фреймворк, написаний на Swift, що полегшує вам перетворення об'єктів моделі (класів та структур) у JSON та з нього, приклад вкладеного 3х рівневого JSON файлу можна побачити на рисунку 5.1. Така структура вважається найпростішої у використанні та найбільш оптимізованою для відправки по інтернету.

Рисунок 5.1 – Приклад JSON структури

4) API GraphQL v4 Apollo. Apollo – це кешуючий клієнт GraphQL для iOS, написаний Swift. Він дозволяє виконувати запити до сервера GraphQL і повертати результати у вигляді специфічних типів Swift. Це означає, що не потрібно мати справу з синтаксичним розбором JSON або переходом словників і змушувати клієнтів вручну передавати значення до потрібного типу. Також не потрібно писати типи моделей самостійно, оскільки вони генеруються з визначень GraphQL, які використовує розроблений інтерфейс.

Оскільки згенеровані типи специфічні для запиту, можна отримати доступ лише до даних, які фактично вказали як частину запиту. Якщо не запитати поле, буде не можливо отримати доступ до відповідної властивості. По суті, це означає, що тепер можна покладатися на перевірку типу Swift, щоб переконатися, що помилки в доступі до даних відображаються під час компіляції. Завдяки інтеграції Xcode можна зручно працювати з кодом інтерфейсу та відповідними визначеннями GraphQL поруч.

Apollo нормалізує результати запитів для побудови кеш-пам’яті даних на стороні клієнта, яка постійно оновлюється при запуску подальших запитів. Це означає, що

користувальницький інтерфейс завжди внутрішньо узгоджений і може бути в курсі всього стану на сервері з мінімальною кількістю необхідних запитів.

Ця комбінація моделей із семантикою значення, одностороннім потоком даних та автоматичним управлінням послідовністю приводить до дуже потужної та елегантної моделі програмування.

5) Спеціальна анімація переходу Hero. Hero - це бібліотека для побудови переходів контролера iOS. Вона забезпечує декларативний рівень поверх громіздких API переходів UIKit - що робить власні переходи простим для розробників.

6) Програмний інтерфейс SnapKit. SnapKit - полегшений DSL, щоб робить автоматичне розміщення об'єктів та обмежень легким для роботи. Мова для конкретного домену (DSL) - це мова, створена для вираження конкретного домену або вирішення певної проблеми. У випадку з SnapKit він має на меті створити синтаксис, який є більш інтуїтивними і простим у використанні, спеціально для обмежень Auto Layout.

7) Події аналітики Mixpanel та Firebase (Umbrella). Mixpanel - це інструмент, який дозволяє аналізувати взаємодію користувачів із підключеним до Інтернету. Він розроблений для підвищення ефективності команд, дозволяючи кожному аналізувати дані користувача в режимі реального часу для виявлення тенденцій, розуміння поведінки користувачів та прийняття рішень щодо продукту. Firebase - це платформа для розробки програмного забезпечення, заснована у 2011 році Firebase inc і придбана Google у 2014 році. Заснована як база даних у режимі реального часу, тепер вона має 18 служб та спеціальні API. Вся платформа - це рішення Backend service як для мобільних, так і для веб-додатків, що включає послуги зі створення, тестування та управління додатками.

Umbrella - Шаблон абстракції Analytics для Swift.

8) Звітування про аварійне завершення роботи Crashlytics. Crashlytics, інструмент звітування про збої в реальному часі, допомагає визначити пріоритети та виправити найпоширеніші збої на основі впливу на реальних користувачів. Crashlytics також легко інтегрується у програми для Android, iOS, macOS, tvOS і watchOS.

9) Ведення журналу CocoaLumberjack. CocoaLumberjack - одна з найпопулярніших систем реєстрації, доступна для розробки та тестування iOS. Вона спеціально

розроблена для передових практик розробки, і тому це не найпростіший варіант для розробки реєстрації користувачів. Хоча фреймворк має безліч дивовижних особливостей, він в основному відомий своєю можливістю підключається архітектурою та детальною конфігурацією. Процес установки дуже простий із виконується за допомогою CocoaPods.

Опис головних особливостей програми

Нижче приведено список головних особливостей які були розроблені та додані до програмного забезпечення:

– базовий, особистий токен доступу та автентифікація oauth2. який дозволить безпечно підключатися, шифрувати дані та підтримувати зв’язок с аккаунтом github;

– перегляд популярних сховищ та користувачів в окремому вікні.

– розширений пошук та сортування сховищ та користувачів, фільтр за мовою;

– перегляд деталей сховища та користувачів, події, проблеми, коміти, запити на витягування, співавтори тощо;

– перегляд повідомлень про проблеми;

– інструмент для підрахунку рядків коду зі сховищ github, кодові вкладки з вибором мови програмування та підсвіткою коду;

– історія змін файла з будь-якого сховища;

– інструмент для візуалізації профілів github для наглядності та легкості читання;

– перегляд моделі вашого внеску на github. на рисунку 5.2 зображено його представлення на сайті github.

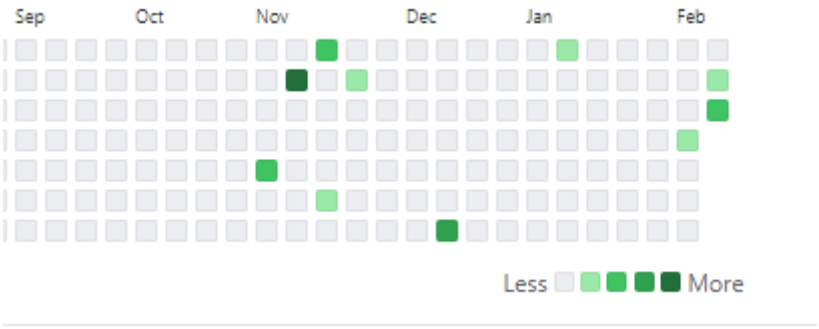

Рисунок 5.2 - Діяльність щодо внесків аккаунта

– переглядач вихідного файлу та підсвічування синтаксису;
– кольорові теми в світлому та темному режимах;
– переключення мови в програмі;
– нові функції у проекті.

Перелік використаних інструментів

В кожному проекті для поліпшення його роботи та покращення процесу розробки використовують сторонні бібліотеки, нижче приведено список бібліотек які додані до мого проекту:

– Brew - менеджер пакунків для macOS;
– Bundler – бібліотека для керування залежностями Ruby;
– Fastlane - найпростіший спосіб автоматизувати створення та випуск додатків для iOs та Android;
– Jazzy - документація для Swift & Objective-C;
– JSONExport - це настільна програма, яка дозволяє експортувати об'єкти JSON як класи моделей;
– R.swift - зображення, шрифти та сегменти в проектах Swift;
– Flex - інструмент налагодження та дослідження в додатку для iOS;
– Sourcetree - безкоштовний клієнт Git для Windows та Mac;
– Postman - потужний HTTP-клієнт для тестування веб-служб;
– Sketch - Додаток для цифрового дизайну для iMac.

5.2 Структура проекту

Надалі розглянемо структуру деяких екранів та всього проекту в цілому. На рисунку 5.3 можна побачити які папки (розділи) містить проект.

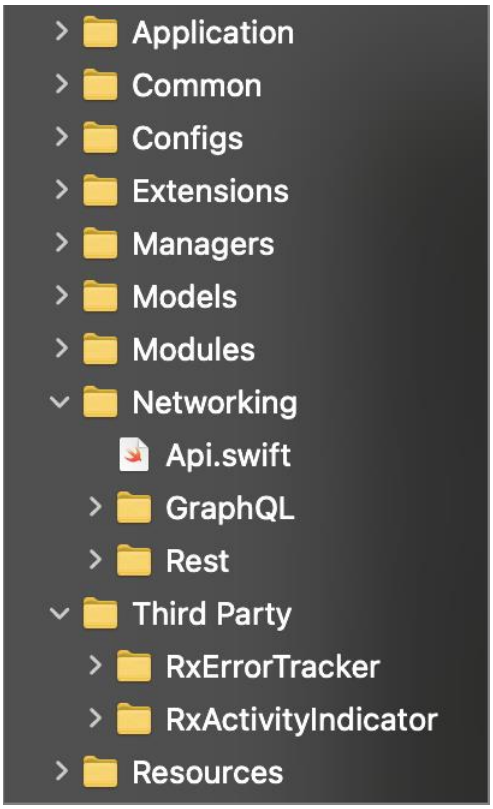

Рисунок 5.3 – Структура проекту

- Application – головні конфігураційні файли проекту;
- Common – усі кастомні UI компоненти;
- Configs – деякі базові конфігурації;
- Extensions – розширення базових класів;
- Managers – основна бізнес логіка додатку;
- Models – моделі даних;
- Modules – моделі UI компонентів та екранів;
- Networking – сервіс спілкування з бекендом;
- Third Party – сторонні бібліотеки;
- Resources – зображення.

Кожна папка містить файли потрібної категорії для зручності пошуку та вірної структуризації.

Структура екрану репозиторіїв

На рисунку 5.4 можна побачити блок схему залежностей моделей і сервісів на екрані пошуку.

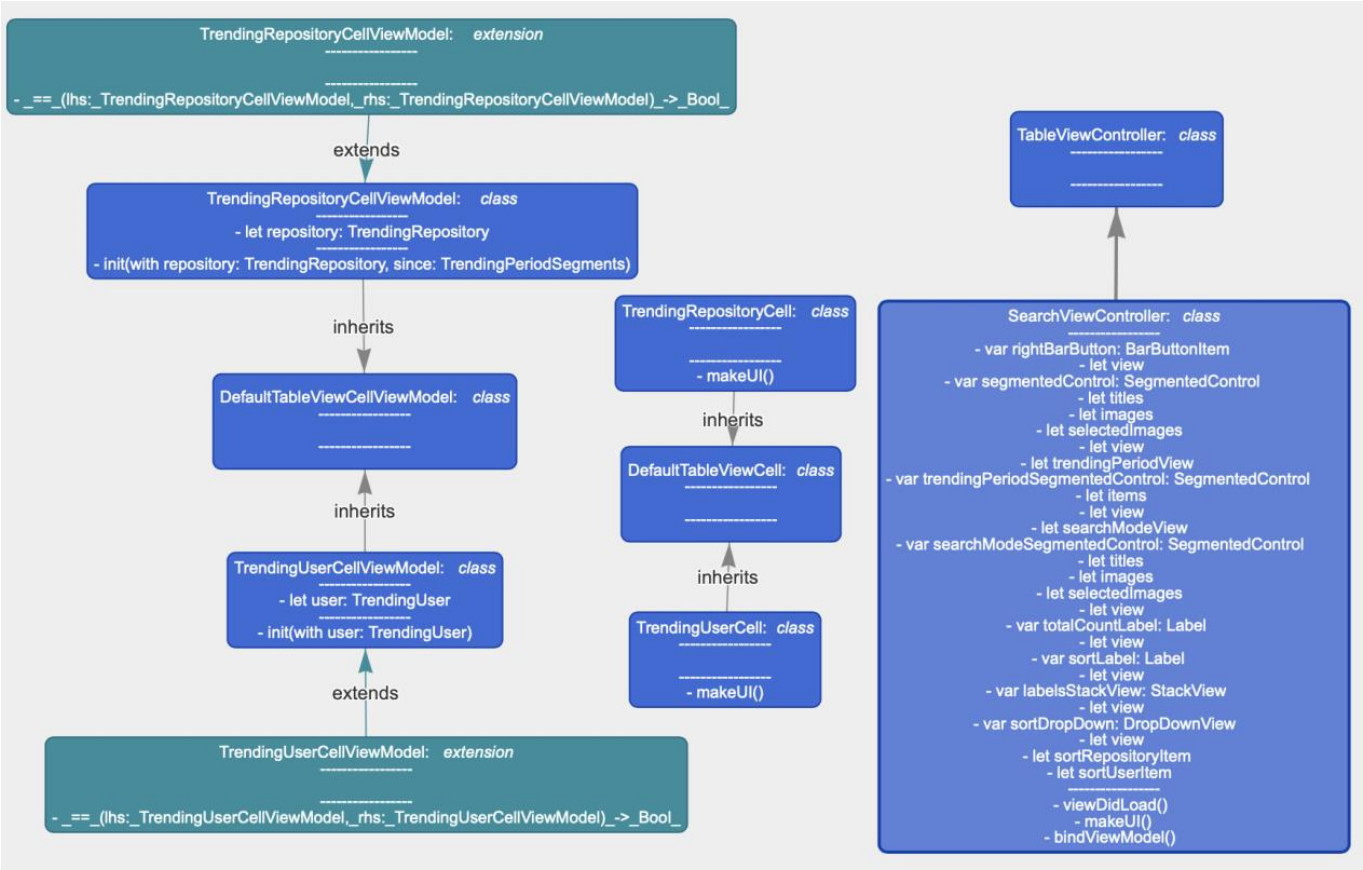

Рисунок 5.4 – Блок схема залежностей екрану пошуку

На схемі можна побачити что SearchViewController успадковує TableViewController. Це успадкування дозволяє SearchViewController мати метод скролу для списку. TrendingRepositoryViewCell успадковує DefaultTableViewCell і TrendingUserCell. Спадкування від TrendingUserCell дає комірці необхідний дизайн, DefaultTableViewCell дає базову поведінку для цієї комірки. Модель TrendingUserCellViewModel має розширення яке дозволяє порівнювати комірки даного типу. TrendingRepositoryCellViewModel аналогічно до TrendingUserCellViewModel.

Структура екрану налаштувань

На рисунку 5.5 можна побачити блок схему залежностей моделей і сервісів на екрані налаштувань.

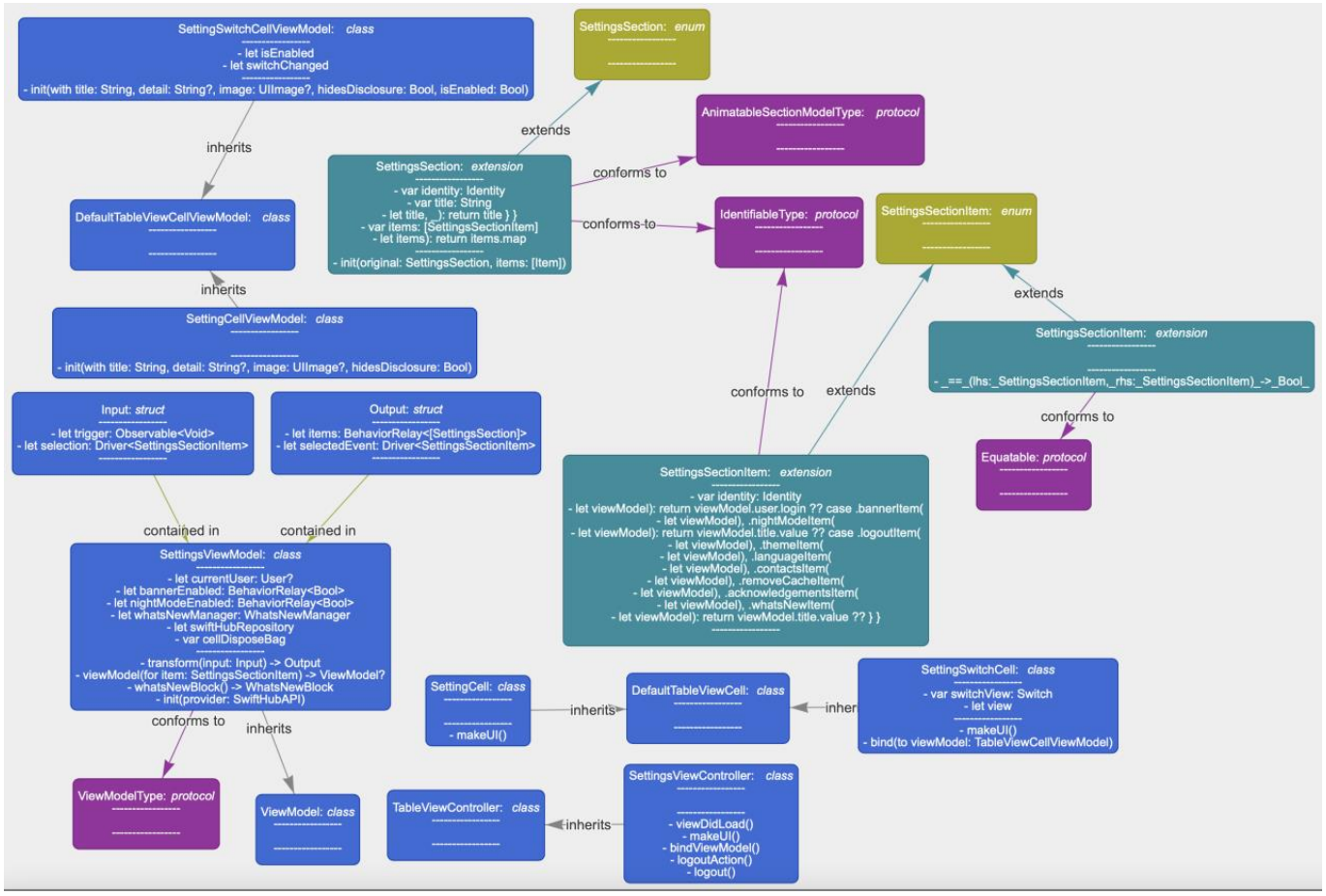

Рисунок 5.5 – Блок схема залежностей екрану налаштувань

На цій схемі бачимо, що SettingsViewController успадковує TableViewController. Це успадкування дозволяє SettingsViewController мати метод скролу для списку, аналогічно як було в SearchViewController. Модель SettingsViewModel має у своєму інтерфейсі 2 структури: input і output, ці структури створені для більш зручного спілкування між моделями. Модель SettingsSectionItem має розширення яке дозволяє порівнювати комірки даного типу. SettingsCell успадковується від DefaultTableViewCell і дає базову поведінку для цієї комірки. Розширення SettingsSection підписується до протоколу AnimatableSectionModelType, ця підписка дозволяє секції мати анімацію коли користувач взаємодіє з нею.

UML діаграма використання

На рисунку 5.6 можна побачити UML діаграму використання з одним користувачем та 26 use-case, які мають розширення та включення.

Рисунок 5.6 – UML діаграма використання додатку

6 РОБОТА КОРИСТУВАЧА ІЗ ПРОГРАМОЮ

Для забезпечення правильної роботи програми необхідно додержуватись головних вимог при установці та рекомендацій щодо її використання.

6.1 Системні вимоги та інсталяція додатку

Щоб встановити мобільний додаток потрібно мати смартфон виробництва компанії Apple. Цей смартфон повинен мати операційну систему iOs 11 та вище. Оскільки цього додатку немає в магазині App Store, то аби встановити його на свій телефон потрібно також мати настільник ПК від компанії Apple, середовище розробки XCode та сертифікат розробника від компанії Apple, який дозволяє встановлювати свої програми на смартфони.

Спочатку потрібно вибрати де буде встановлюватися і запускатися програма, на вашому телефоні чи на симуляторі від XCode. На рисунку 6.1 вікно вибору місця запуску додатка.

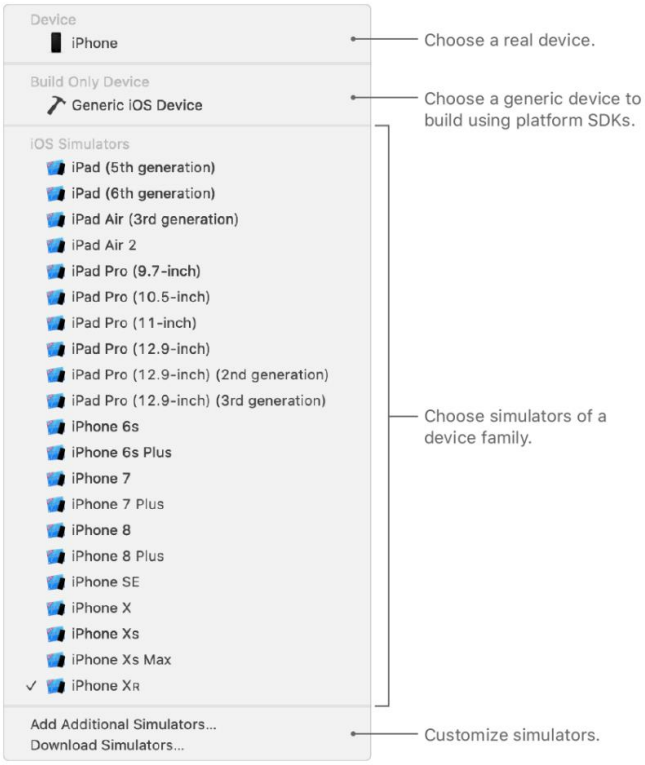

Рисунок 6.1 – Вибір платформи запуску

На рисунку 6.2 можна побачити зелений трикутник який запустить інсталяцію, якщо виконані всі системні вимоги.

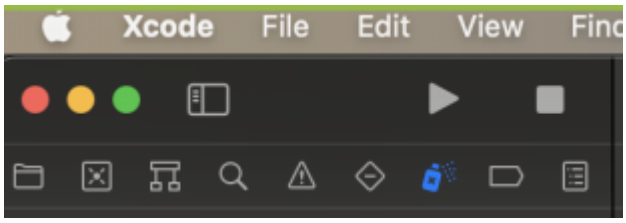

Рисунок 6.2 – Запуск та інсталяція програми

Після цього програма автоматично запуститься на заданому девайсі і буде готова до використання.

6.2 Опис інтерфейсу програми

Оскільки в програмі дуже багато вікон с різним призначенням, то потрібно розібрати кожна з низ для правильної роботи із додатком. Присутні екрани: вікно авторизації (3 варіанти), вікно налаштувань, вікно пошуку, вікно активностей, декілька вікон для налаштувань під конкретного користувача.

Вікно авторизації

У програмі присутні 3 варіанти авторизації на сервісі GitHub:

- авторизація через OAuth2, використовується сайт GitHub та браузер смартфона, це найшвидший та найзручніший спосіб авторизації для користувачів смартфона;
- авторизація через персонально створений токен (цей токен створюється людиною яка володіє аккаунтом у своєму кабінеті на сайті);
- проста авторизація через логін і пароль прямо у додатку, після цього йде авторизація по API.

На рисунку 6.3 зображено вікно авторизації через OAuth2 аунтифікатор, на рисунку 6.4 показана авторизація через персональний токен, на рисунку 6.5 зображено аунтефікація через логін та пароль у додатку.

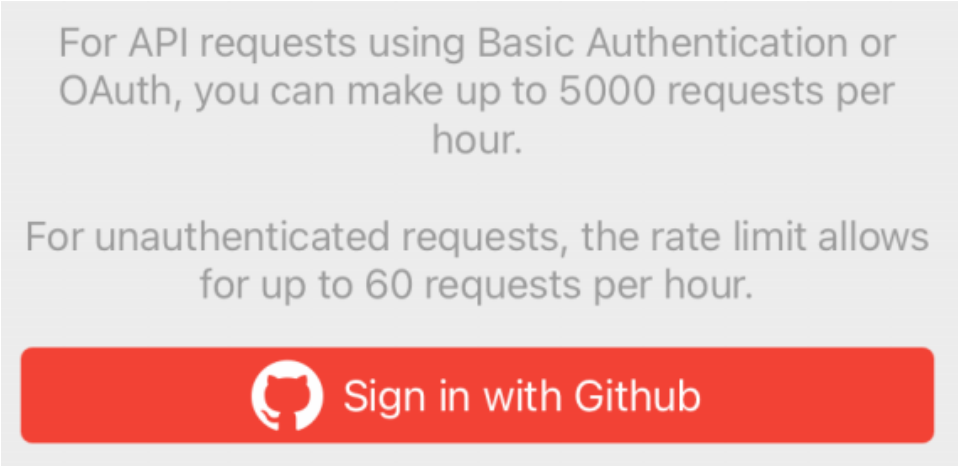

Рисунок 6.3 – Авторизація через OAuth2

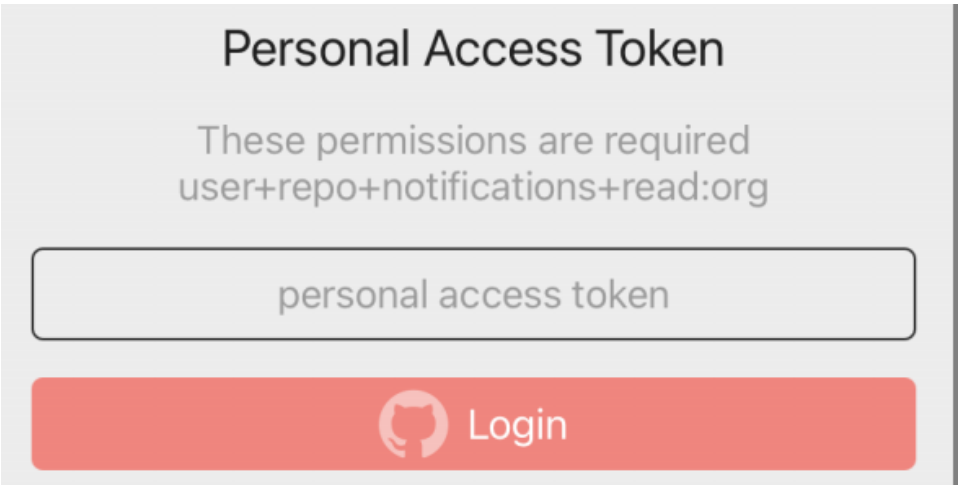

Рисунок 6.4 – Авторизація через персональний токен

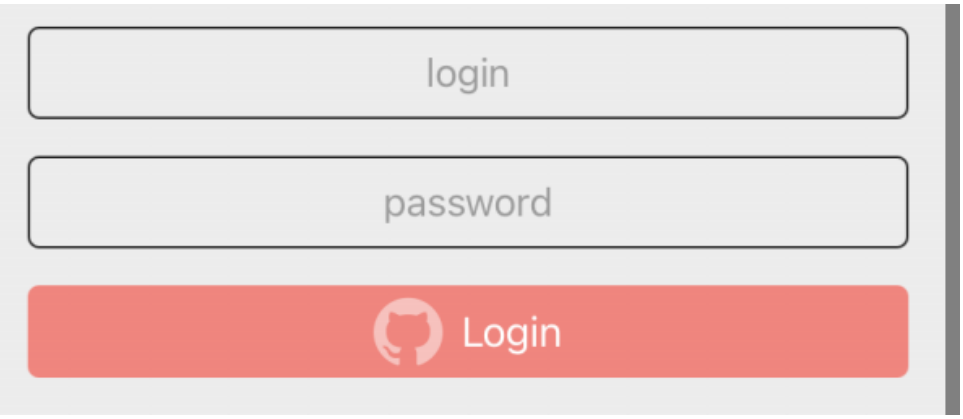

Рисунок 6.5 – Авторизація через через логін і пароль

Кожен із цих методів є безпечним та використовує стандартне шифрування, але OAuth2 біль швидкий, оскільки якщо користувач все входив до свого аккаунту в браузері, то він автоматично зробить логін до програми та не потрібно буде вводити свої персональні дані.

Вікно налаштувань

Кожна програма має вікно де користувач може налаштувати програмний додаток під себе, оскільки немає інтерфейсу який сподобається одразу всім та буде зручним для кожного юзера. На рисунку 6.6 скрін шот вікна налаштувань.

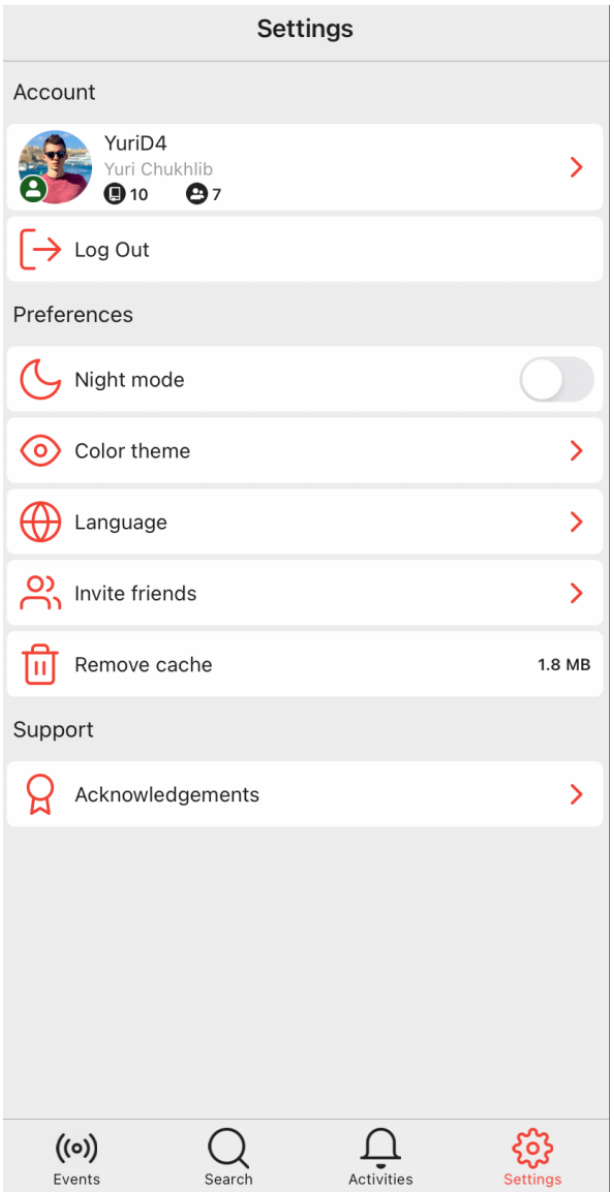

Рисунок 6.6– Вікно налаштувань

- Log Out – вийти з поточного аккаунта для зміни користувача;
- Night Mode – нічний режим;
- Color Theme – вибір теми кольорів на рисунку 6.7;

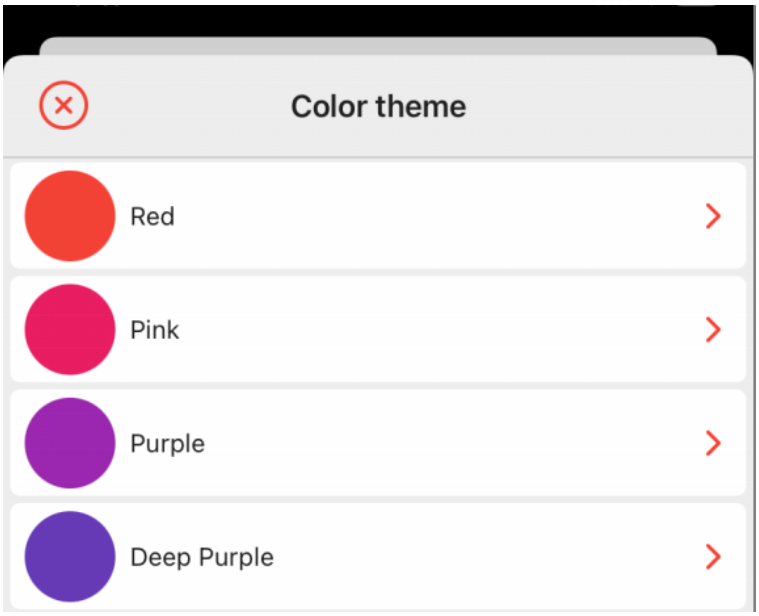

Рисунок 6.7– Вибір кольорової теми

- Language – вибір мови для всіх надписів на рисунку 6.8;

Рисунок 6.8– Вибір мови надписів

- Invite friends – як швидко додати друга через книгу контактів на рисунку 6.9;

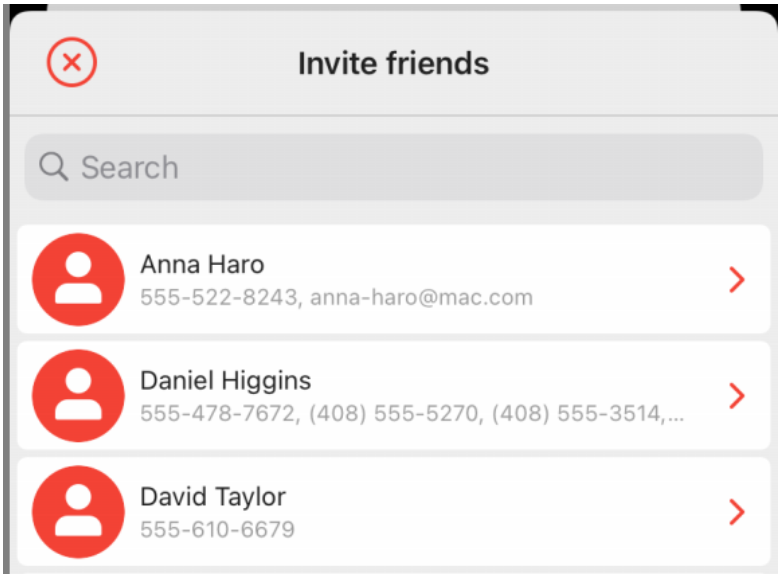

Рисунок 6.9– Додавання друзів через книгу контактів

- Remove Cash – очистити кеш який зберіг додаток;
- Acknowledgements – усі бібліотеки які використовувались з описом на рисунку 6.10 (частина екрану).

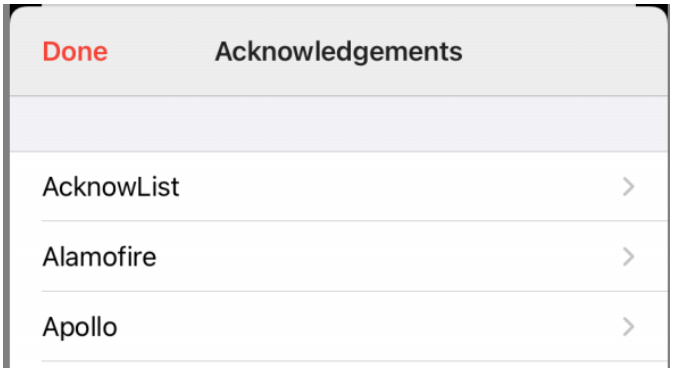

Рисунок 6.10– Список бібліотек

Огляд свого аккаунта

Для перегляду всієї інформації по аккаунту користувача створено окремо сторінку с різними розділами, яка зображена на рисунку 6.11.

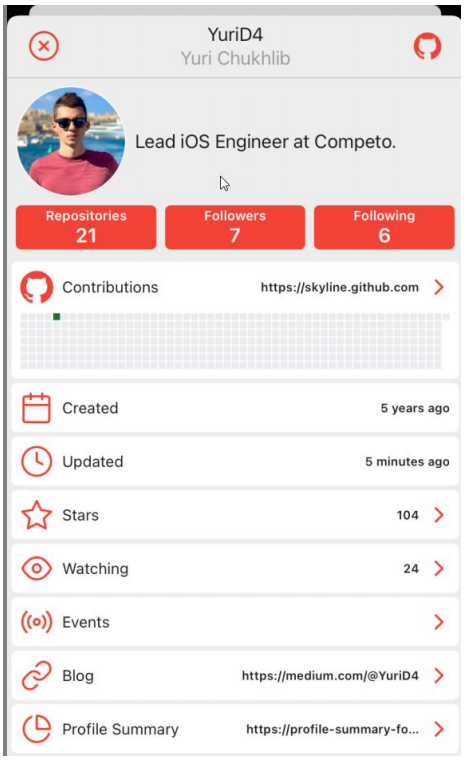

Рисунок 6.11 – Сторінка користувача з категоріями

На ній можна побачити кількість створених репозиторіїв, підписників та аккаунтів за якими спостерігає користувач. Список репозиторіїв можна побачити на малюнку 6.12. 2(частина екрану).

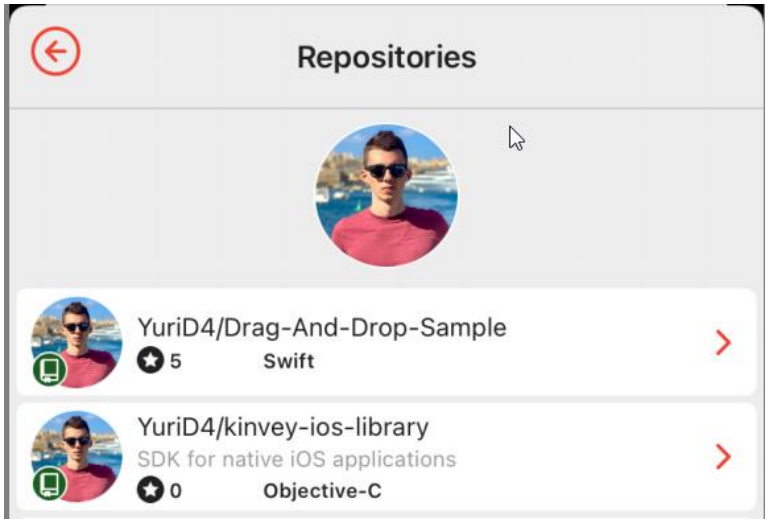

Рисунок 6.12– Репозиторії користувача

Список підписників на аккаунт також відображається на окремому вікні як зображено на рисунку 6.13, аккаунти за якими слідкує юзер аналогічні.

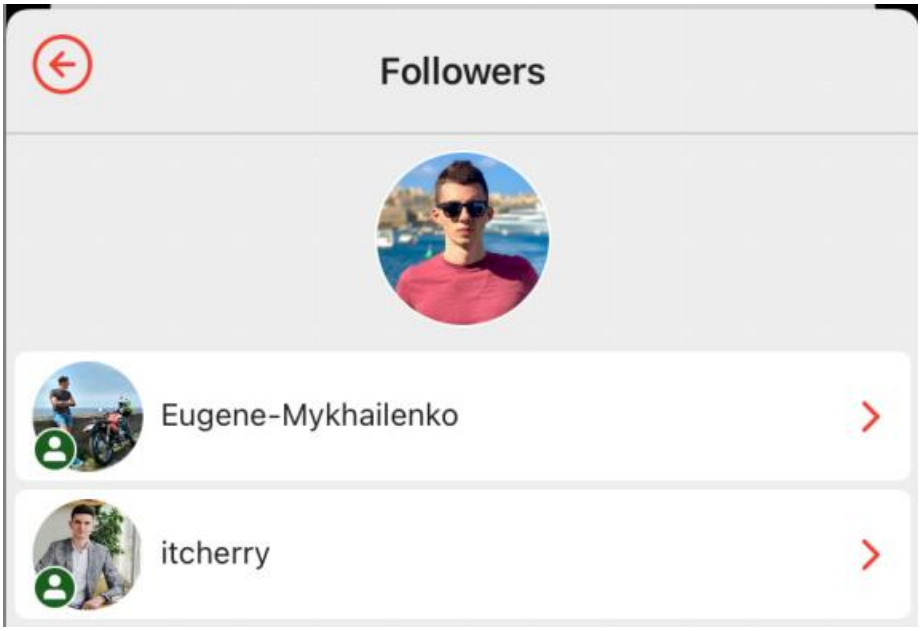

Рисунок 6.13 – Список підписників

За допомогою спеціальної бібліотеки для створення публічних профілів створено окреме вікно з повною деталізацією аккаунта та з графічним зображенням всієї інформації.

Вкладка активностей

На наступному екрані зображено усі активності які поступили на аккаунт, окремо непрочитані та бесіди в яких користувач зараз знаходиться. Верхню частину екрану можна побачити на рисунку 6.14.

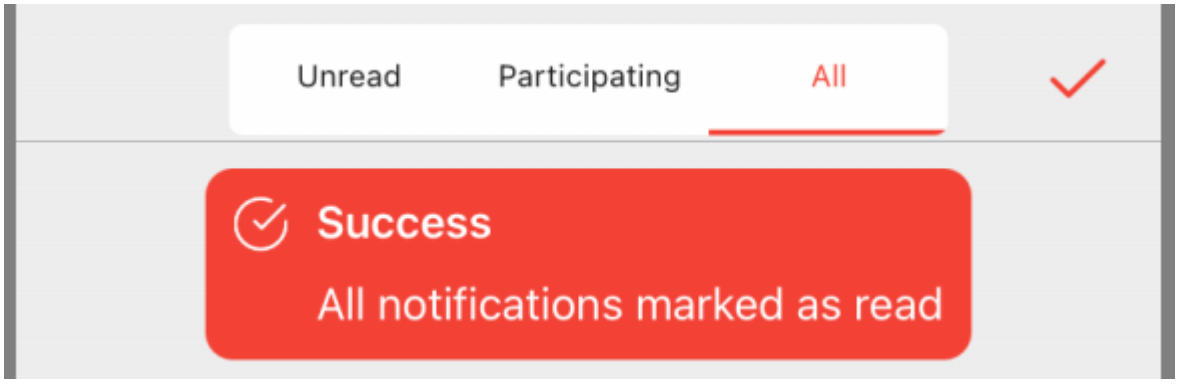

Рисунок 6.14 – Усі прочитані/непрочитані вхідні сповіщення

Розділ пошуку

Більша частина інформації знаходиться не в особистому аккаунту, а в аккаунтах інших користувачів. Для цього було створено екран пошуку де відображаються: найбільш популярні проекти за день / тиждень / місяць, найбільш популярні користувачі за день / тиждень / місяць, цей екран зображено на рисунку 6.15.

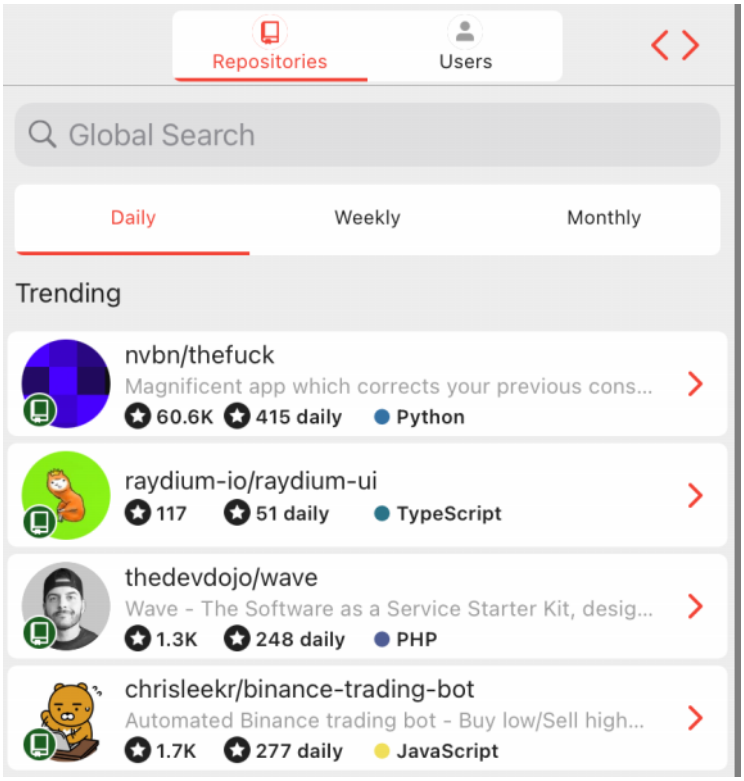

Рисунок 6.15 – Екран пошуку інформації

Біля кожного проекту зображено кількість зірочок які поставили користувачі, кількість зірок за останній день та мова програмування цього проекту. Сторінка автоматично завантажує інформацію наперед, тому при швидкому перегляданні немає зупинки на оновлення та доповнення інформації. Також біля кожного проекту в кожній комірці відображається короткий опис проекту. На рисунку 6.16 зображено найбільш популярні аккаунти в GitHub (частина екрану).

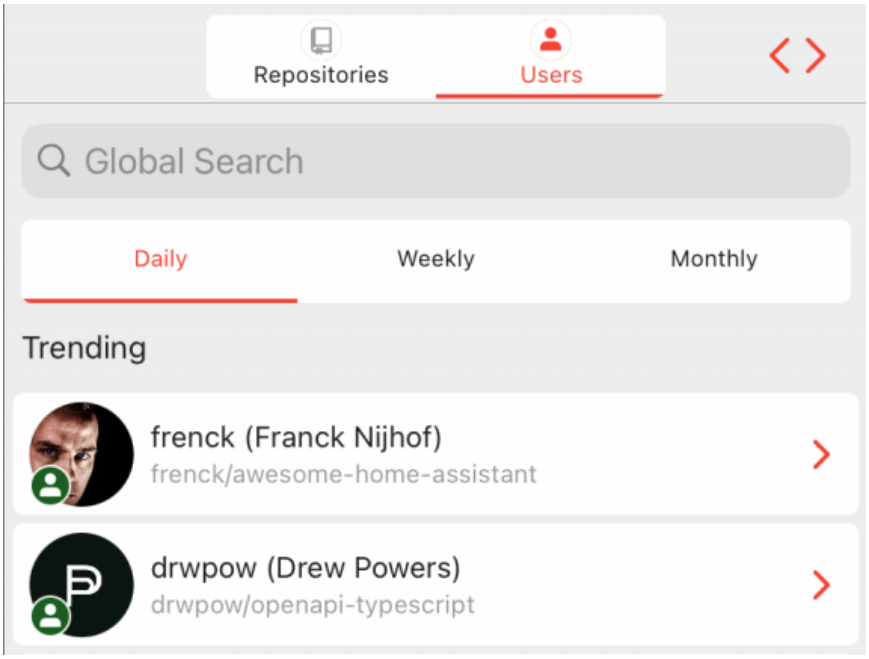

Рисунок 6.16 – Тренди серед аккаунтів

Для зручного пошуку потрібної інформації створено фільтри пошуку для всіх існуючих мов програмування, їх вибір зображено на рисунку 6.17 (частина екрану).

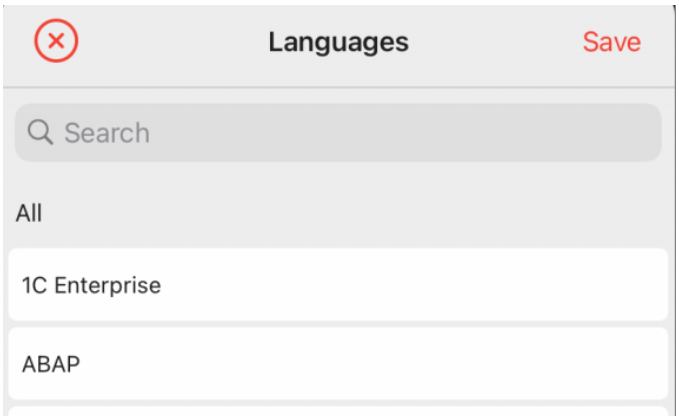

Рисунок 6.17 – Фільтр мов програмування

Якщо по фільтру не знайдено жодного проекту, то буде пусте вікно с повідомленням що таких проектів не знайдено. Для кожного проекту створено розділ з категоріями і всією інформацією. Де зображено: розмір проекту, дату створення, дату останнього оновлення, кількість помилок що знайшли інші користувачі, кількість комітів, кількість гілок проекту, кількість людей які допомагали розроблювати цей додаток тощо. Це вікно зображено на рисунку 6.18.

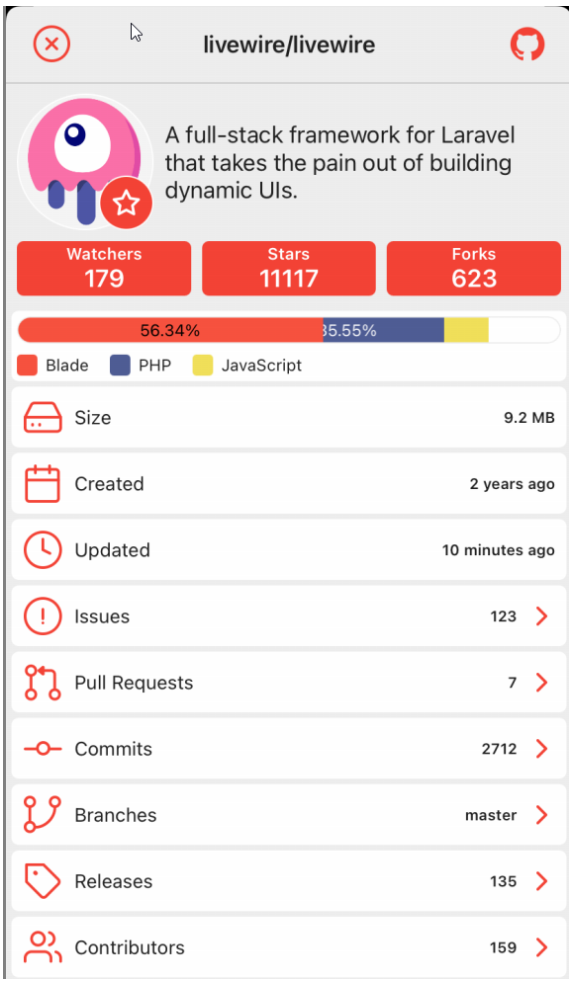

Рисунок 6.18 – Прикладу екрану з інформацією про проект

Коли користувач знаходить помилку у проекті він створює окрему гілку по цій помилці і там можете спілкуватись з іншими розробниками щодо цієї помилки, тобто зображення всіх помилок обов'язкове для користувача GitHub і цей екран можна побачити на рисунку 6.19. Так само якщо користувач може переглядати помилки які знайшли інші юзери та їх коментарі, то йому так само потрібно відповідати на ці коментарі та вести бесіди, сповіщення про нові повідомлення приходять до розділу активностей, а на рисунку 6.20 зображено екран з коментарями від інших користувачів.

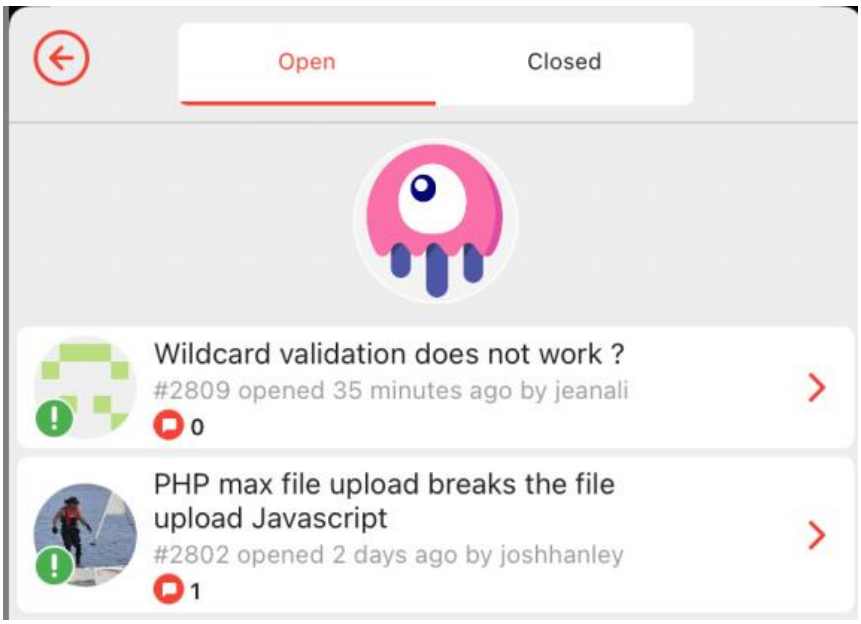

Рисунок 6.19 – Інформація про помилки у проекті

На кожному блоці відображено назву помилки, номер помилки, коли вона була створена та кількість коментарів інших користувачів.

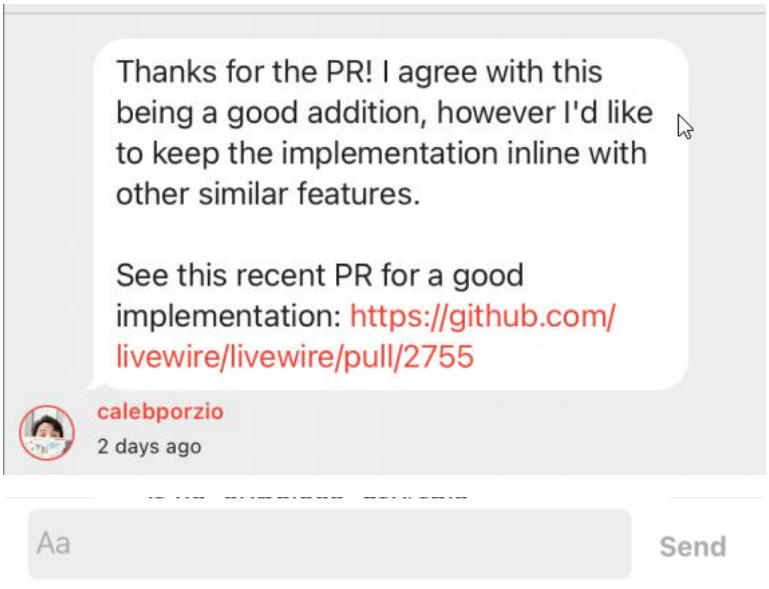

Рисунок 6.20 – Друга частина екрану з коментарями від інших користувачів

В кожному проекті велика структура файлової системи та багато різних файлів з кодом які потрібно переглядати, всі файли розбиті по папкам зі зручним пошуком інформації та у кожного файла підписано який розмір він займає. На рисунку 6.21 зображено приклад файлової системи одного з проектів.

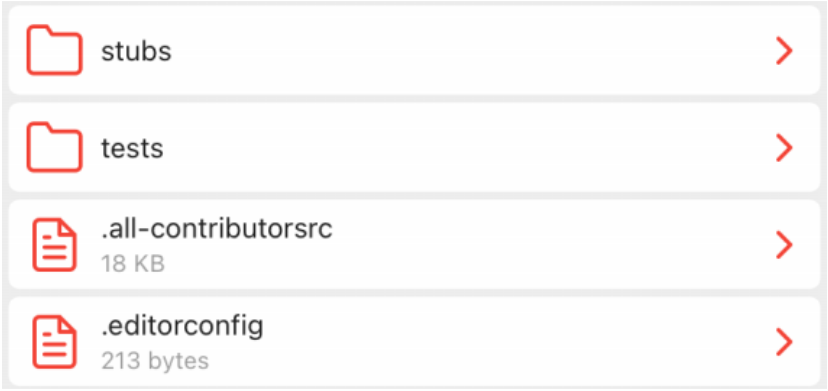

Рисунок 6.21 – Файлова система проекту

Кожен файл можна відкрити та передивитися його код, для зручного читання створено фільтр вибору мов, для підсвітки ключових слів з кожної мови тощо. На рисунку 6.22 приклад файлу з кодом.

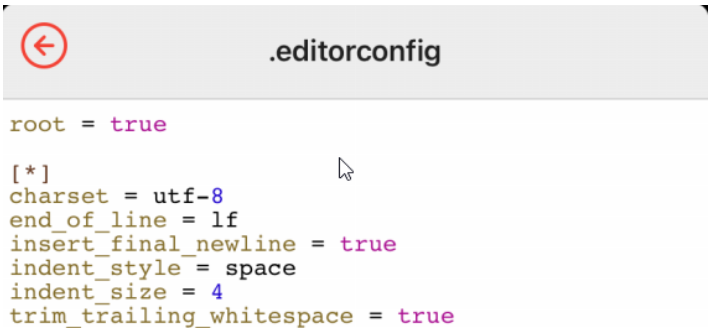

Рисунок 6.22 – Приклад зображення коду

Для вибори IDE та мови знизу створено зручну панель яку зображено на рисунку 6.23.

Рисунок 6.23 – Панель вибору зображення коду

За допомогою бібліотеки яка відображала графічне представлення користувача на екрані налаштувань створено графічне представлення вмісту проекту по файлам, видів

мови програмування та рядків коду. Приклад цієї кругової діаграми зображено на рисунку 6.24. Підрахунок відбувається автоматично за допомогою бібліотеки.

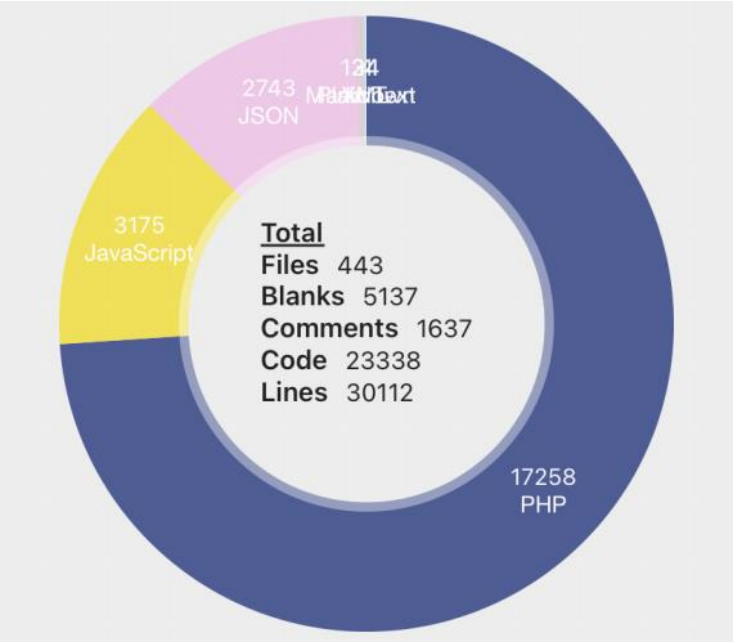

Рисунок 6.24 – Відображення вмісту проекту

І останній екран відображає оновлення проектів де користувач приймає участь, оновлення останніх подій він людей на яких він підписан, створення нових репозиторіїв тощо. Частину екрану зображено на рисунку 6.25.

Рисунок 6.25 – Оновлення від користувачів

ВИСНОВКИ

В ході розробки даного програмного продукту було створено мобільний додаток для роботи з сервісом контролю версій GitHub, який:

– дозволяє переглядати всі публічні репозиторії та інформацію по ним: структуру проекту, код кожного файлу, людей, які працювали над проектом тощо;

– отримувати сповіщення на телефон щодо нових повідомлень, інформацію про нові помилки у своїх проектах, зустрічі, на які запрошено користувача та будь-які оновлення, які пов’язані з особистим аккаунтом;

– кастомізувати зовнішний вигляд додатку для покращення сприйняття інформації та роботи з ним;

– переглядати програмний код, в тому числі код, де були знайдені помилки від інших користувачів і одразу відповісти їм.

Порівняння розробленого додатку з аналогами показало, що він має суттєві переваги при роботі з смартфоном завдяки таким функціям, як сповіщення, чат з іншими людьми, нагадування про події, розширена статистика, якою можна ділитись в соц-мережах тощо. Розроблений додаток може бути використано розробниками програмних продуктів, студентами, спеціалістами з пошуку персоналу для IT компаній.

Програмний засіб прийнято для використання при розробці спеціалізованого програмного забезпечення ТОВ “Амронбудторг”, що підтверджено Актом про впровадження результатів бакалаврської роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The powerful programming language [Електронний ресурс]. – Режим доступу: https://developer.apple.com/swift/
2. Craig Grummitt. iOS Development with Swift in Motion. – 50 с.
3. About Swift [Електронний ресурс]. – Режим доступу: https://docs.swift.org/swift-book/
4. Mobile Apps or Desktop [Електронний ресурс]. – Режим доступу: https://transform.makeen.io/2016/08/19/desktop-apps-vs-web-apps-vs-mobile-apps/
5. iOs or Android [Електронний ресурс]. – Режим доступу: https://www.tomsguide.com/face-off/iphone-vs-android
6. GitHub Rest API v3 [Електронний ресурс]. – Режим доступу: https://docs.github.com/en/rest
7. Introduction to Github APIs [Електронний ресурс]. – Режим доступу: https://www.loginradius.com/blog/async/github-api
8. The Swift Programming Language (Swift 5.4) [Електронний ресурс]. – Режим доступу: https://books.apple.com/ru/book/the-swift-programming-language-swift-5-4/id881256329
9. Git vs. SVN – What Is The Difference? [Електронний ресурс]. – Режим доступу: https://www.perforce.com/blog/vcs/git-vs-svn-what-difference
10. iOS MVVM Tutorial: Refactoring from MVC [Електронний ресурс]. – https://www.raywenderlich.com/6733535-ios-mvvm-tutorial-refactoring-from-mvc
11. What Is GitHub, and What Is It Used For? [Електронний ресурс]. – https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/
12. Друга Всеукраїнській науково-практична інтернет-конференція молодих вчених та здобувачів вищої освіти [Електронний ресурс]. – Режим доступу: http://www.ksau.kherson.ua/files/konferencii/20210514

ДОДАТОК А

Мобільний додаток для роботи з GitHub

Специфікація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ ТР71242_21Б

Аркушів 2

Київ – 2021

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР71242_21Б 81-1	Пояснювальна записка.docx	Пояснювальна записка
Комплекс		
Компоненти		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР71242_21Б 13-1	Додаток В	Опис програмного коду
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР71242_21Б 12-1	AuthManager.swift	Менеджер вікна авторизації
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР71242_21Б 12-2	LibsManager.swift	Менеджер бібліотек
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР71242_21Б 12-3	ContactsManager.swift	Менеджер контактів

ДОДАТОК Б

Мобільний додаток для роботи з GitHub

Лістинг програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ ТЕФ_АПЕПС_ ТР71242_21Б

Аркушів 10

Київ – 2021

```
//  AuthManager.swift
import Foundation
import KeychainAccess
import ObjectMapper
import RxSwift
import RxCocoa

let loggedIn = BehaviorRelay<Bool>(value: false)

class AuthManager {

    /// The default singleton instance.
    static let shared = AuthManager()

    // MARK: - Properties
    fileprivate let tokenKey = "TokenKey"
    fileprivate let keychain = Keychain(service: Configs.App.bundleIdentifier)

    let tokenChanged = PublishSubject<Token?>()

    init() {
        loggedIn.accept(hasValidToken)
    }

    var token: Token? {
        get {
            guard let jsonString = keychain[tokenKey] else { return nil }
            return Mapper<Token>().map(JSONString: jsonString)
        }
        set {
            if let token = newValue, let jsonString = token.toJSONString() {
                keychain[tokenKey] = jsonString
            } else {
                keychain[tokenKey] = nil
            }
```

```
            tokenChanged.onNext(newValue)
            loggedIn.accept(hasValidToken)
        }
    }

    var hasValidToken: Bool {
        return token?.isValid == true
    }

    class func setToken(token: Token) {
        AuthManager.shared.token = token
    }

    class func removeToken() {
        AuthManager.shared.token = nil
    }

    class func tokenValidated() {
        AuthManager.shared.token?.isValid = true
    }
}

//  ContactsManager.swift
import Foundation
import RxSwift
import RxCocoa
import Contacts

typealias ContactsHandler = (_ contacts: [CNContact], _ error: NSError?) -> Void

enum ContactsError: Error {
    case accessDenied
}

class ContactsManager: NSObject {
```

```
    static let `default` = ContactsManager()

    let contactsStore = CNContactStore()

    // MARK: - Contact Operations

    func getContacts(with keyword: String = "") -> Observable<[Contact]> {
        return Single.create { single in
            switch CNContactStore.authorizationStatus(for: CNEntityType.contacts) {
            case CNAuthorizationStatus.denied, CNAuthorizationStatus.restricted:
                // User has denied the current app to access the contacts.
                single(.error(ContactsError.accessDenied))

            case CNAuthorizationStatus.notDetermined:
                // This case means the user is prompted for the first time for allowing contacts
                self.contactsStore.requestAccess(for: CNEntityType.contacts, completionHandler: { (granted, error) -> Void in
                    // At this point an alert is provided to the user to provide access to contacts. This will get invoked if a user responds to the alert
                    if granted {
                        self.getContacts().subscribe(onNext: { (newContacts) in
                            single(.success(newContacts))
                        }).disposed(by: self.rx.disposeBag)
                    } else if let error = error {
                        single(.error(error))
                    }
                })

            case  CNAuthorizationStatus.authorized:
                // Authorization granted by user for this app.
                var contactsArray = [CNContact]()
                let contactFetchRequest = CNContactFetchRequest(keysToFetch: self.allowedContactKeys())
                contactFetchRequest.sortOrder = .givenName
                do {
```

```
            try self.contactsStore.enumerateContacts(with: contactFetchRequest, usingBlock: { (contact, stop) -> Void in
                contactsArray.append(contact)
            })

            single(.success(contactsArray.map { Contact(with: $0) }.filter({ (contact) -> Bool in
                if let name = contact.name, !keyword.isEmpty {
                    return name.contains(keyword, caseSensitive: false)
                }
                return true
            })))
        }
            // Catching exception as enumerateContactsWithFetchRequest can throw errors
        catch {
            single(.error(error))
            logError(error.localizedDescription)
        }
    @unknown default: break
    }
    return Disposables.create { }
    }.asObservable()
}

/// We have to provide only the keys which we have to access. We should avoid unnecessary keys when fetching the contact. Reducing the keys means faster the access.
///
/// - Returns: The allowed keys
func allowedContactKeys() -> [CNKeyDescriptor] {
    return [CNContactNamePrefixKey as CNKeyDescriptor,
            CNContactGivenNameKey as CNKeyDescriptor,
            CNContactFamilyNameKey as CNKeyDescriptor,
            CNContactOrganizationNameKey as CNKeyDescriptor,
            CNContactBirthdayKey as CNKeyDescriptor,
            CNContactImageDataKey as CNKeyDescriptor,
            CNContactThumbnailImageDataKey as CNKeyDescriptor,
```

```
            CNContactImageDataAvailableKey as CNKeyDescriptor,
            CNContactPhoneNumbersKey as CNKeyDescriptor,
            CNContactEmailAddressesKey as CNKeyDescriptor
        ]
    }
}

//  LibsManager.swift

import Foundation
import RxSwift
import RxCocoa
import SnapKit
import IQKeyboardManagerSwift
import CocoaLumberjack
import Kingfisher
#if DEBUG
import FLEX
#endif
import FirebaseCrashlytics
import NVActivityIndicatorView
import NSObject_Rx
import RxViewController
import RxOptional
import RxGesture
import SwifterSwift
import SwiftDate
import Hero
import KafkaRefresh
import Mixpanel
import Firebase
import DropDown
import Toast_Swift

typealias DropDownView = DropDown
```

```
/// The manager class for configuring all libraries used in app.
class LibsManager: NSObject {

    /// The default singleton instance.
    static let shared = LibsManager()

    let bannersEnabled = BehaviorRelay(value: UserDefaults.standard.bool(forKey: Configs.UserDefaultsKeys.bannersEnabled))

    private override init() {
        super.init()

        if UserDefaults.standard.object(forKey: Configs.UserDefaultsKeys.bannersEnabled) == nil {
            bannersEnabled.accept(true)
        }

        bannersEnabled.skip(1).subscribe(onNext: { (enabled) in
            UserDefaults.standard.set(enabled, forKey: Configs.UserDefaultsKeys.bannersEnabled)
            analytics.set(.adsEnabled(value: enabled))
        }).disposed(by: rx.disposeBag)
    }

    func setupLibs(with window: UIWindow? = nil) {
        let libsManager = LibsManager.shared
        libsManager.setupCocoaLumberjack()
        libsManager.setupAnalytics()
        libsManager.setupAds()
        libsManager.setupTheme()
        libsManager.setupKafkaRefresh()
        libsManager.setupFLEX()
        libsManager.setupKeyboardManager()
        libsManager.setupActivityView()
        libsManager.setupDropDown()
        libsManager.setupToast()
```

```
}

func setupTheme() {
    themeService.rx
        .bind({ $0.statusBarStyle }, to: UIApplication.shared.rx.statusBarStyle)
        .disposed(by: rx.disposeBag)
}

func setupDropDown() {
    themeService.attrsStream.subscribe(onNext: { (theme) in
        DropDown.appearance().backgroundColor = theme.primary
        DropDown.appearance().selectionBackgroundColor = theme.primaryDark
        DropDown.appearance().textColor = theme.text
        DropDown.appearance().selectedTextColor = theme.text
        DropDown.appearance().separatorColor = theme.separator
    }).disposed(by: rx.disposeBag)
}

func setupToast() {
    ToastManager.shared.isTapToDismissEnabled = true
    ToastManager.shared.position = .top
    var style = ToastStyle()
    style.backgroundColor = UIColor.Material.red
    style.messageColor = UIColor.Material.white
    style.imageSize = CGSize(width: 20, height: 20)
    ToastManager.shared.style = style
}

func setupKafkaRefresh() {
    if let defaults = KafkaRefreshDefaults.standard() {
        defaults.headDefaultStyle = .replicatorAllen
        defaults.footDefaultStyle = .replicatorDot
        themeService.rx
            .bind({ $0.secondary }, to: defaults.rx.themeColor)
            .disposed(by: rx.disposeBag)
```

```
        }
    }

    func setupActivityView() {
        NVActivityIndicatorView.DEFAULT_TYPE = .ballRotateChase
        NVActivityIndicatorView.DEFAULT_COLOR = .secondary()
    }

    func setupKeyboardManager() {
        IQKeyboardManager.shared.enable = true
    }

    func setupKingfisher() {
        // Set maximum disk cache size for default cache. Default value is 0, which means no limit.
        ImageCache.default.diskStorage.config.sizeLimit = UInt(500 * 1024 * 1024) // 500 MB

        // Set longest time duration of the cache being stored in disk. Default value is 1 week
        ImageCache.default.diskStorage.config.expiration = .days(7) // 1 week

        // Set timeout duration for default image downloader. Default value is 15 sec.
        ImageDownloader.default.downloadTimeout = 15.0 // 15 sec
    }

    func setupCocoaLumberjack() {
DDLog.add(DDOSLogger.sharedInstance)
        let fileLogger: DDFileLogger = DDFileLogger() // File Logger
        fileLogger.rollingFrequency = TimeInterval(60*60*24)  // 24 hours
        fileLogger.logFileManager.maximumNumberOfLogFiles = 7
        DDLog.add(fileLogger)
    }

    func setupFLEX() {
        #if DEBUG
        FLEXManager.shared.isNetworkDebuggingEnabled = true
        #endif
```

```
    }

    func setupAnalytics() {
        FirebaseApp.configure()
        Mixpanel.initialize(token: Keys.mixpanel.apiKey)
        FirebaseConfiguration.shared.setLoggerLevel(.min)
    }

    func setupAds() {
        GADMobileAds.sharedInstance().start(completionHandler: nil)
    }
}

extension LibsManager {

    func showFlex() {
        #if DEBUG
        FLEXManager.shared.showExplorer()
        analytics.log(.flexOpened)
        #endif
    }

    func removeKingfisherCache() -> Observable<Void> {
        return ImageCache.default.rx.clearCache()
    }

    func kingfisherCacheSize() -> Observable<Int> {
        return ImageCache.default.rx.retrieveCacheSize()
    }
```

ДОДАТОК В

Мобільний додаток для роботи з GitHub

Опис програмного модулю

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ ТЕФ_АПЕПС_ ТР71242_21Б

Аркушів 8

Київ – 2021

АНОТАЦІЯ

Програма являє собою мобільний додаток для роботи із веб сервісом GitHub. Додаток має роль – користувач веб версії GitHub. У функціонал входить: перегляд всіх репозиторіїв та інформації по ним, структуру проекту та код кожного файлу, отримувати сповіщення на телефон щодо нових повідомлень та інформацію про нові помилки у своїх проектах, кастомізувати зовнішний вигляд додатку, переглядати програмний код.

Проект було розроблено з використанням мови програмування SWIFT, технології MVVM, середовище розробки XCode.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Даний проект було розроблено з використанням мови програмування SWIFT, технології MVVM, середовище розробки XCode.

Програма призначена для покращення процесу взаємодії використання сервісу GitHub шляхом створення спеціалізованого мобільного додатку.

До основних переваг можна віднести: сповіщення, чат з іншими людьми, нагадування про події, розширена статистика якою можна ділитись в соц-мережах тощо. Розроблений додаток може бути використано розробниками програмних продуктів, студентами, спеціалістами з пошуку персоналу для IT компаній.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

При створенні програмного продукту була поставлена задача покращення процесу взаємодії використання сервісу GitHub для зручного використання сервісу у смартфоні. За допомогою архітектури MVVM програму можна дуже легко модифікувати. Програмний продукт містить велику кількість екранів взаємодії та багато методів які виконують наступні функції:

- відображають вікна перегляду репозиторіїв та користувачів;
- дають доступ до інформації яка пов’язана з репозиторієм: деталі, події;
- відображають інформацію по особистому аккаунту GitHub;
- міняють дизайни додатку;
- відображають перегляд програмного коду з підсвіткою;
- взаємодіють с іншими людьми.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Створена програма складається з декількох частин:

– моделей всіх об’єктів;

– бізнес логіки (модуль менеджерів, який працює з ViewModel який працює з View та Model);

– представлення (робота користувача з представленням у вигляді екрану смартфону).

Робота програми побудована таким чином, що відкривши мобільний додаток користувач потравляє на форму авторизації. Авторизувавшись перед користувачем відкривається повний функціонал додатку.

ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Для створення даної програми було використано середовище розробки Xcode та мова програмування SWIFT 5.1 з використанням технології MVVM та великої кількості сторонніх бібліотек для поліпшення роботи програми та більшої кастомізації.

Для запуску даної програми користувачу необхідно мати доступ до мережі Інтернет, мати смартфон від компанії Apple, комп’ютер від компанії Apple та сертифікат розробника.

ВХІДНІ ТА ВИХІДНІ ДАНІ

Вхідними даними є:

– логін та пароль, особистий токен для входу в мобільний додаток.

Вихідними даними є:

– робота з доступною інформацією.

ДОДАТОК Г

Мобільний додаток для роботи з GitHub

Апробація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ ТЕФ_АПЕПС_ ТР71242_21Б

Аркушів 4

Київ – 2021

Було подано тези на конференцію «Матеріали Другої Всеукраїнській науково-практичній інтернет-конференції молодих вчених та здобувачів вищої освіти». Тези надруковано в 5 секції “ Впровадження інновацій та сучасних технологій ” на сторінці 100. Текст тез наведено нижче.

УДК 004.42

К.В. Чухліб, О.М. Шушура
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
kirill.chukhlib@gmail.com

МОБІЛЬНИЙ ДОДАТОК ДЛЯ РОБОТИ З GITHUB

Git – одна з найкращих та найбільш бажаних доступних систем контролю версій програмного забезпечення. Сучасному розробнику потрібно швидко отримувати доступ до коду бібліотек, програм та open-source проектів для обговорення, пошуку потрібної для себе інформації тощо, яка представлена на Git. Сьогодні присутній різноманітний ряд інтерфейсів Git для різних операційних систем таких як: Android, iOS, Mac, Linux тощо.

Вибір між мобільними додатками або веб-сайтами як варіантами доступу до інформації залежить від вартості їх розробки, зручності використання всіх інтерфейсів та необхідних функцій для аудиторії, для якої вони розроблюються.

З огляду на це, дуже часто користувачі більше віддають перевагу мобільним додаткам, ніж їх мобільним веб-сайтам. Це і є вагома причина для створення мобільних додатків для звернення до потенційних (і існуючих) клієнтів з тією чи іншою послугою. Так само у мобільних додатках набагато більше варіантів взаємодії з самим користувачем через push повідомлення, геолокація, вподобання які дозволяє бачити система смартфона, активність користувача тощо.

Саме тому метою дослідження є створення мобільного додатка для web сервісу GitHub з використанням Rest API v3 від самого GitHub, так само використання великої кількості бібліотек для поліпшення роботи самого додатка для задля більшої кастомізації під кожного користувача з подальшою публікацією цього продукту до магазину додатків від компанії Apple, який називається App Store.

Для реалізації додатку обрана мова програмування Swift 5.1 з використанням моделі MVVM – Model-View-ViewModel, що у майбутньому дозволить простіше розроблювати, тестувати та модифікувати мобільний додаток. В такому масштабному проекті, звичайно, не обійтися без менеджера бібліотек для Xcode.

Предметом дослідження є мобільний додаток, як допомога для розробників при пошуку інформації та пришвидшення програмного процесу. З подальшим його використанням для пришвидшення пошуку потрібної інформації.

Новизна дослідження аргументується відсутністю сучасних аналогів, програмних продуктів, які були написані з використанням останніх оновлень API GitHub та мови програмування Swift 5.1.

Для досягнення поставленої мети було сформульовано наступні задачі:

- вивчити API документації GitHub
- розібрати Access Token and OAuth2 аунтефікація у додатках
- розробити фільтр користувачів, мов програмування тощо
- розробити пошук та перегляд історії будь-якого репозиторію
- створити декілька дизайнів додатку
- реалізувати перегляд коду файлів з вибраною мовою
- реалізувати перегляд найближчих подій
- оптимізувати програму під планшети
- забезпечити спілкування щодо помилок у комітах тощо.
- створити пошук серед проектів с заданими параметрами
- реалізувати модульну структуру з використанням RxSwift
- реалізувати захищений зв'язок програми с сервісом GitHub
- викласти програму до магазину App Store
- розробити варіанти монетизації програми
- розібрати інші системи та, можливо, підключити якісь з них.

Запропонований додаток дозволить користувачам переглядати великий збірник різних бібліотек та програм з повним кодом прямо у смартфоні. Можна публікувати та передивлятись свої роботи як готові продукти для майбутнього працевлаштування, оскільки популярні аккаунти GitHub дуже добре оцінюються роботодавцями.

В тому числі кожен користувач може допомагати з існуючими проектами, оскільки це open-source система, тому всі проекти які там розміщені дуже добре написані та не мають багів чи недоліків. Використання таких програм чи бібліотек тільки поліпшить роботу будь-якого іншого програмного додатку де вони будуть використовуватись.

А отже – це все буде супроводжуватися підвищенням якості розроблення програмних продуктів з залученням сучасних технологій.

ЛІТЕРАТУРА:

1. The powerful programming language. URL: https://developer.apple.com/swift (дата звернення 30.04.21).
2. About Swift URL: https://docs.swift.org/swift-book/ (дата звернення 29.04.21).
3. Mobile Apps or Desktop. URL: https://transform.makeen.io/2016/08/19/desktop-apps-vs-web-apps-vs-mobile-apps/ (дата звернення 30.04.21).
4. iOs or Android. URL: https://www.tomsguide.com/face-off/iphone-vs-android (дата звернення 30.04.21).
5. GitHub Rest API v3. URL: https://docs.github.com/en/rest (дата звернення 28.04.21).
6. Introduction to Github APIs. URL: https://www.loginradius.com/blog/async/github-api/ (дата звернення 30.04.21).
7. The Swift Programming Language (Swift 5.4). URL: https://books.apple.com/ru/book/the-swift-programming-language-swift-5-4/id881256329 (дата звернення 28.04.21).
8. Git vs. SVN – What Is The Difference? URL: https://www.perforce.com/blog/vcs/git-vs-svn-what-difference (дата звернення 30.04.21).