Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ПІД ОПЕРАЦІЙНУ СИСТЕМУ IOS ДЛЯ КОНТРОЛЮ ВЛАСНИХ ФІНАНСІВ

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
с.в. _Борозенний С.О.__
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2021 р.

Виконав студент _____
___Ландяк А.П._____
(прізвище та ініціали)
“____” _____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем,
доцент, к.ф-м.н.
_____О. П. Жежерун (підпис)
„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студента Ландяка Андрія Петровича факультету інформатики 3-го курсу

ТЕМА: Розробка мобільного додатку під операційну систему IOS для контролю власних фінансів

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
Частина 1: Аналіз предметної області. Постановка завдання курсової роботи
Частина 2: Теоретичні відомості
Частина 3: Опис реалізації програмного продукту
Висновки
Список літератури
Додатки

Дата видачі „___” _____ 2020 р. Борозенний С.О. _____
(підпис)

Завдання отримав _____
(підпис)

Календарний план виконання курсової роботи

Тема: Розробка мобільного додатку під операційну систему IOS для контролю власних фінансів

Календарний план виконання курсової роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	01.11.2020	
2.	Аналіз матеріалів за темою	14.01.2021	
3.	Розробка та програмування програми	01.03.2021	
4.	Написання текстової частини до курсової роботи	05.04.2021	
5.	Коригування виконаної роботи	10.05.2021	
6.	Створення слайдів для доповіді та написання доповіді.	11.05.2021	
7.	Остаточне оформлення роботи та слайдів	16.05.2021	
8.	Захист курсової роботи	24.05.2021	

Ландяк А. П. _____

Борозенний С. О. _____

“_____” _____

Зміст

Перелік умовних позначень

EAV – Entity-attribute-value model

User – користувач

iOS – мобільна операційна система від Apple

macOS – графічна операційна система

tsOS – операційна система компанії Apple для цифрового медіаплеєра

watchOS – операційна система розумних годинників iWatch (Apple Watch)

ОС – операційна система

SwiftUI – оновлений UI-фреймворк для мови програмування Swift

UIKit – UI-фреймворк для мови програмування Swift, презентований в 2013

Фреймворк – це комплекс програмних рішень для побудови програмного забезпечення

Realm – система управління об'єктами баз даних з відкритим кодом

Core Data – фреймворк для взаємодії з базою даних

SQLite – система керування баз даних

Firebase – платформа Alphabet Inc. для розробки мобільних чи веб-застосунків

СУБД – системи управління базами даних

API – application programming interface

View – представлення або вікно, що є основним будівельним блоком користувацької частини програми

IDE – Integrated Development Environment

Анотація

Робота присвячена розробці інструмента, що стане в нагоді тим, хто хоче навести порядок в особистих фінансах, відслідковувати всі грошові потоки та структурувати витрати за категоріями. Це мобільний додаток під операційну систему iOS, який дозволяє записувати, зберігати доходи та витрати, проводити аналітику за витратами по категоріям за весь час або ж за конкретний період, завдяки якому ви зможете налагодити вашу фінансову ситуацію як в особистих рахунках, так і в сімейних.

Вступ

Фінансова грамотність є достатньо трендовим поняттям сьогодні та багато людей все більше і більше намагаються опанувати дану галузь. Існує багато книжок та семінарів, які навчають цьому поняттю та демонструють його важливість, а також як це може вплинути на подальше життя, а також що з цим досвідом безпосередньо можна робити. Саме тому трейдинг як професія для фінансово грамотних набуває все більших масштабів: покупка пакетів акцій, валюти, цінних паперів[1].

Трейдер – це людина, яка вміє користуватися власними грошима. Можна сказати, що кожна людина є трейдером, адже ми щодня обмінюємо гроші на те, що нам потрібно: продукти, одяг, похід в ресторан. Першим кроком досягнення фінансової грамотності є контроль власних фінансів: доходів, витрат та боргів. Задля досягнення даного кроку потрібен зручний інструмент. Це може бути звичайний блокнот з календарем, проте зручним рішенням було б все таки використання передових технологій, ресурси яких не такі вичерпні, як кількість сторінок у блокноті: програма на робочому столі комп’ютера або ж мобільний додаток.

Питання про актуальність таких застосунків взагалі не постає – нові технології переживають свій підйом та розвиваються щодня. Проте постає запитання – десктопна програма чи мобільний додаток. Згідно статистики дослідницької фірми “BroadbandSearch” за 2020-2021 роки близько 55% глобального трафіку було здійснено з мобільних пристроїв[3] .

За мету курсової роботи було поставлено дослідити основні можливості додатків для контролю фінансів та проаналізувати сучасні підходи для розробки застосунків під операційну систему iOS.

Завдання курсової роботи є створення додатку під операційну систему iOS для контролю власних фінансів.

Об’єктом дослідження є створення додатку для зручного додавання транзакцій доходу та витрат, та здійснення аналітики за витратами.

Предметом дослідження є існуючі аналоги – мобільні додатки під iOS, основою яких є контроль фінансів та синхронізація з банківськими рахунками.

Мовою програмування було обрано Swift – як єдиний інструмент для розробки додатків під операційну систему iOS від компанії Apple, а саме використання нового фреймворку SwiftUI, який дозволяє проектувати та розробляти користувацький інтерфейс декларативним способом.

Для зберігання та використання даних було обрано фреймворк Core Data, який зберігає дані в додатку та дозволяє отримувати дані, які організовані в вигляді сутність-атрибут-значення - EAV.

Текстова частина курсової роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

Перший розділ буде включати в себе опис розвитку обраної галузі, проаналізовано аналоги, буде дано аналіз основним перевагам та недолікам існуючим рішенням.

Другий розділ буде містити основні теоретичні відомості даної області реалізації та засобів її розробки.

Третій розділ буде присвячено опису реалізації мобільного додатку, інтерфейсу користувача, конкретизації постановки задачі, пояснення структури.

Висновок буде містити підсумки роботи, результати виконання кожного кроку в поставлених завданнях і результат в загальному.

Після висновку буде прикріплено список використаних джерел та вихідний код.

Основна частина

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАННЯ КУРСОВОЇ РОБОТИ

1.1. Аналіз предметної області та обґрунтування теми

Фінансова грамотність – це не тільки здатність розуміти, володіти та ефективно використовувати різні фінансові навички, які пов’язанні з плануванням бюджету, управління фінансами, інвестування, а ще й багато інших навичок та концепцій. Це основа співіснування в світі, де гроші є визначальними[1].

Чимало книжок, наукових публікацій та статей можна знайти на тему фінансової грамотності та в чому її необхідність. Проте можна навести приклади, до чого може привести фінансова неграмотність: накопичення тягаря боргу, що в свою чергу може призвести до банкрутства, вилучення житла чи інших негативних наслідків. Саме тому багато книжок, наукових публікацій та доповідей на цю тему пропонують ряд ключових кроків для покращення потоку фінансових транзакцій, серед яких: створення бюджету та дотримання його, щомісячний перегляд витрат.

Таким чином, якщо виділити час і відслідковувати всі фінансові потоки вкінці місяця, то можна зрозуміти чи раціонально витрачаються гроші на продукцію певних категорій. Дуже часто відстеження витрат може не надто приємно здивувати, адже інколи щоденні невеликі витрати можуть зробити достатньо велику суму вкінці місяця. Саме через всі вищезазначені моменти можна зрозуміти, що контроль фінансів – це дуже важлива і необхідна процедура, а суворий контроль витрат – це відмінна риса хороших фінансово-розумних людей.

Більшість мобільних фінансових додатків-менеджерів під операційну систему iOS пропонують непогані рішення для контролю власних надходжень, витрат, а деякі навіть і з можливість задавання бюджету. Таким чином, можна проводити щомісячну аналітику по витратах в категоріях, створювати план та намагатися дотримуватися його. Завданням даного додатку є зручний інтерфейс з можливістю додавати транзакції, контролю бюджету згідно визначеного плану (популярними варіантами є 50/20/30, де 50 – це обов'язкові витрати, 20 – заощадження, 30 – необов'язкові витрати), або навіть підтягувати інформацію з банківської карти за згодою користувача.

1.2. Опис існуючих рішень-аналогів для мобільного додатку

Для розуміння та визначення необхідних вимог для зручного і корисного мобільного додатку під операційну систему iOS, які вже існують на ринку та користуються певною популярністю. Комплексний аналіз має дозволити сформувати чітке бачення важливого функціоналу, а також виділити переваги та недоліки, які трапляються в аналогічних застосунках.

1.2.1. Money Flow

Даний продукт містить близько однієї тисячі оцінок від користувачів та має середній показник рейтингу – 4.8/5. Наявний достатньо приємний інтерфейс для аналітики по витратах за кожною категорією, присутні діаграми, які можна обирати для кожного місяця, проте інтерфейс додавання нових транзакцій не є надто приємним і простим. Щодо основного функціоналу, то він включає в себе:

- додавання транзакції витрат;
- додавання транзакції доходу;
- кругова аналітика витрат за категоріями;

- список суми витрат за кожною категорією;

- перерахунок коштів за курсом банків.

Після проведення дослідження користувацького інтерфейсу можна зробити кілька висновків. Функціонал даного додатку є достатньо обмежений, інтерфейс додавання транзакцій є не надто очевидним та легким в користуванні, великий контраст між яскравою аналітикою та чорно-білим екраном додавання транзакцій.

Меню додавання транзакції зображено на рисунку 1.1

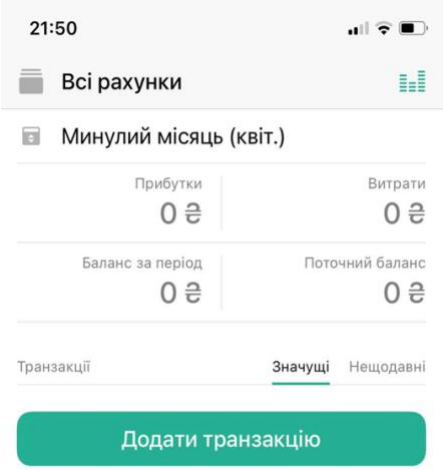


Рисунок 1.1 – Меню транзакцій додатку Money Flow

Діаграма витрат за категоріями зображена на рисунку 2.2

Рисунок 1.1 – Меню транзакцій додатку Money Flow

З переваг даного мобільного додатку можна виділити:

- хороший інтерфейс для аналітики;
- достатній функціонал в платній версії (даний продукт є повністю платним, мінімально безкоштовно можна користуватися тільки додавання транзакцій, аналітика присутня тільки у версії Pro);
- можливість внесення суми в валюті та її перерахунок;
- можливість додавання власної категорії.

З недоліків варто зазначити:

- мінімальний функціонал в безкоштовній версії;
- перезавантажений інтерфейс додавання транзакції;
- надто простий інтерфейс показу транзакції;
- немає інтеграції з банковими карточками.

1.2.2. Money Manager

Продукт Money Manager з рейтингом 4.9 займає 129 місце в розділі “Фінанси” та налічує близько чотирьохсот оцінок. Простий за розумінням інтерфейс допомагає легко ознайомитися з основними можливостями додатку. Основний функціонал включає в себе:

- створення транзакцій;
- аналітика за рахунком;
- створення різних рахунків;
- налаштування бюджету;
- надсилання виписки рахунку у форматі .xlsx на e-mail.

Після проведення аналітики даного продукту можна зазначити, що він є непоганим прикладом додатку для контролю фінансів, проте з певними недоліками, які можна було б суттєво покращити. Є певна кількість корисного функціоналу, який можна було б додати та зробити інтерфейс більш користувацьким.

Діаграма витрат Money Manager зображена на рисунку 1.3

Рисунок 1.3 – діаграма витрат Money Manager

Список транзакцій Money Manager зображено на рисунку 1.4

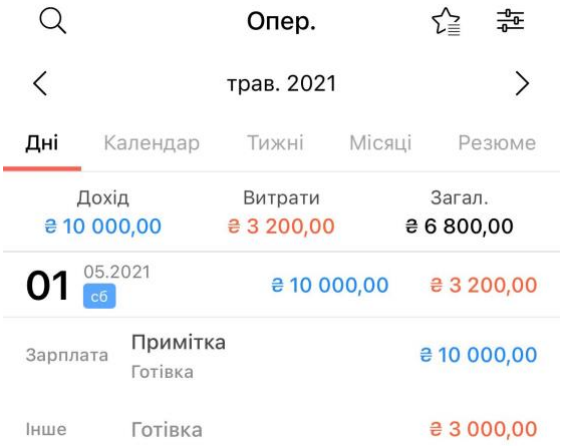


Рисунок 1.4 – список транзакцій додатку Money Manager

З переваг даного мобільного додатку можна виділити:

- достатній функціонал в безкоштовній версії;
- аналітика (діаграма з можливістю анімації) за категоріями;
- створення власних категорій.

З недоліків варто зазначити:

- перезавантажений інтерфейс додавання транзакції;
- неінтуїтивний інтерфейс додавання транзакцій;
- немає аналітики за весь час;
- перегляд транзакцій за весь час неможливо (тільки безпосередньо за місяць).

1.2.3. Zen-money

Даний продукт містить близько однієї з половиною тисячі оцінок від користувачів та має середній показник рейтингу – 4.8/5. Додаток не тільки дозволяє слідкувати за витратами та доходами, а також здійснює автоматичний підтягування інформації з попереднім підключенням банківських карт. За статистикою з офіційного сайту Дзен-мані в них є близько сімдесяти тисяч активних користувачів, 152 мільйони транзакцій, а також 900 мільйонів доларів, які керуються за допомогою даного додатку. Основний функціонал включає в себе:

- підключення синхронізації з банками;
- аналітика бюджету і витрат за категоріями;
- створення транзакцій: борг, переказ, прибуток, витрати;
- створення і користування бюджету кожного місяця згідно визначеного плану;
- можливість додавання власної категорії з вибором зображення для неї.

Після проведення аналітики даного продукту можна зазначити, що він є найкращим прикладом зручного додатку для контролю фінансів. Адже наявний зручний і простий інтерфейс додавання нових транзакцій, можливість перегляду аналітики конкретного місяця. Присутня достатня кількість можливого доступного функціоналу і без платної версії.

Також є можливість підключити бота в Telegram, який здійснює підтримку, з можливістю переглянути останні транзакції. Тобто разом з додатком це є дуже зручною перспективою в користуванні.

Меню синхронізації з банками зображено на рисунку 1.5

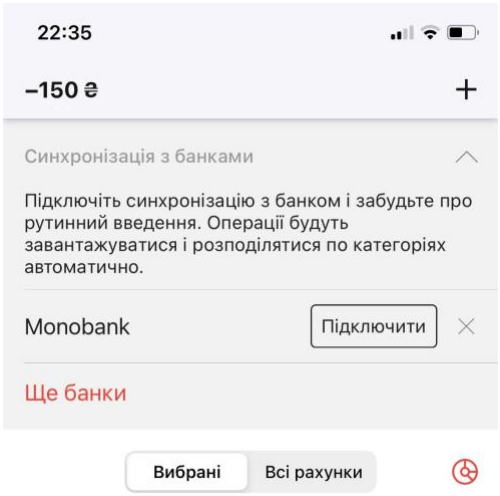


Рисунок 1.5 – меню синхронізації з банками

Список відображення транзакцій зображено на рисунку 1.6

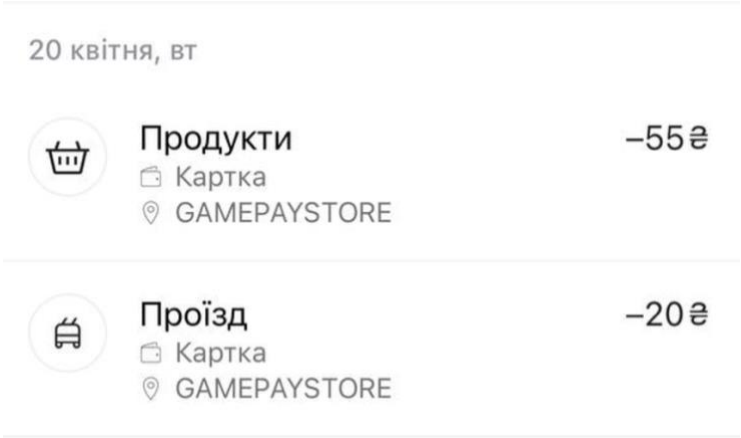


Рисунок 1.6 – список транзакцій

Діаграма витрат за категоріями зображена на рисунку 1.7

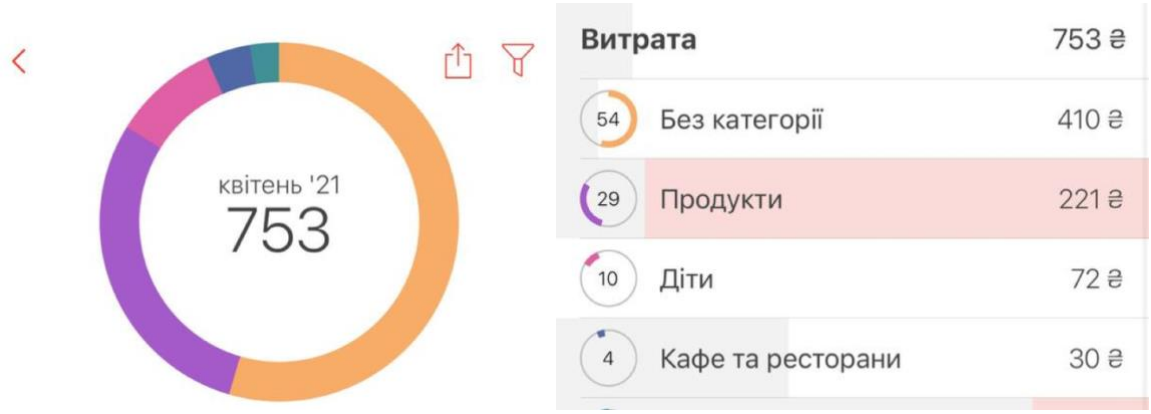


Рисунок 1.7 – діаграма витрат за категоріями

З переваг даного мобільного додатку можна виділити:

- достатній функціонал в безкоштовній версії (і перегляд аналітики, і створення транзакцій, і синхронізація з банками)
- зручний показ транзакцій;
- синхронізація з банками;
- можливість створення власних категорій;
- аналітика за конкретний місяць.

З недоліків варто зазначити:

- перезавантажений інтерфейс додавання транзакції;
- перезавантажений інтерфейс аналітики;
- немає аналітики за весь час.

1.3. Постановка завдання курсової роботи

Задачею дослідження є аналіз існуючих рішень, визначення та виділення переваг та недоліків даних застосунків, побудова власного мобільного додатку під операційну систему iOS з урахуванням можливостей аналогічних продуктів.

Таким чином, створений застосунок має містити такі можливості:

- Авторизація користувачів та способи доступу до втрачених аккаунтів (відновлення паролю)
- Додавання транзакцій за доходами користувача з потрібними полями: сума, дата здійснення транзакції та коментар, за бажанням користувача з можливістю видалити транзакцію чи редагувати.
- Додавання транзакцій за витратами користувача з потрібними полями: сума, вибір категорії у якій була здійснення транзація, дата здійснення транзакції та коментар, за бажанням користувача, з можливістю видалити транзакцію чи редагувати.
- Створення, видалення чи редагування категорій відповідно до тих, які потрібні користувачеві.
- Здатність здійснювати аналітику витрат за категоріями, з можливістю фільтрації за місяцем і роком.
- Оперування фотографіями для категорій, з можливістю обрання користувачем зі списку доступних

Додатково необхідно:

- - розробити інтуїтивно-зрозумілий користувацький інтерфейс для зручного користування додатком
- створити адаптивний інтерфейс, щоб в залежності від моделі телефону вигляд не погіршувався.
- додавати фотографію для профіля користувача за його бажанням.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1. Огляд особливостей mobile та desktop підходу для вирішення проблеми

У сучасних реаліях немає сенсу обирати між мобільними додатками чи десктопними рішення, адже світ програмного забезпечення пропонує чимало цікавих рішень в обох варіантах, які можуть прекрасно доповнювати одне одного. Сперечатися про явні переваги кожного з підходів недоречно, адже хоча часто вони й вирішують певну проблему, але мають певні переваги, недоліки і особливості. Не виключено, що сьогодні стане у пригоді мобільний додаток, а завтра для посилення зручності треба буде використати веб-застосунок. Щоб точно не помилятися у їх корисності, варто навести переваги та недоліки.

Переваги розробки веб-застосунку:

- доступність: веб-програми дозволяють здійснювати доступ користувачам, незалежно від їхніх пристроїв та операційних систем, за допомогою браузера;
- легкість здійснення оновлення: можна легко здійснювати посилання оновлень на сервер для миттєвої зміни видимості на будь-яких пристроях;
- вартість реалізації: розробка веб-застосунків вимагає менше часу та ресурсів, ніж реалізація аналогічного мобільного додатку;
- вихід на ринок: веб-застосунки не потрібно схвалювати власниками магазинів-завантаження, тому їх легше випустити;
- пошук у павутині: пошук веб-рішення релевантніший, ніж мобільного-додатку у пошукових системах (наприклад, Google)[4].

Недоліки розробки веб-застосунку:

- обмеженість використання функцій мобільних пристроїв: камера, локація та інший вбудований функціонал мобільного пристрою є недоступним у веб-застосунку;
- стабільність: оновлення чи різноманітність веб-браузерів можуть спричиняти труднощі діяльності веб-застосунку, тому варто постійно оновлювати бюджет;
- офлайн, а точніше відсутність роботи в офлайні: веб-програма дуже залежна від Інтернету, тому користувач не матиме доступ до застосунку без доступу до мережі;
- популярність та зручність App Store та Google Play: користувачі звикли знаходити необхідні їм додатки на цих платформах, адже це надійні джерела. Тому додатковий пошук в Інтернеті може викликати певний дискомфорт, тому людина надасть перевагу мобільному аналогу[4].

Переваги розробки мобільного додатку:

- зручність функціоналу: мобільні додатки працюють із можливістю використання вбудованого функціоналу мобільних пристроїв, таких як камера, мікрофон і так дальше, тому дані програми мають можливість працювати швидше;
- push-сповіщення: можливість надсилати нагадування користувачеві, що значно допомагає user;
- офлайн, а точніше здатність працювати в режимі офлайну: даний пункт залежить від реалізації програми та вбудованого в неї функціоналу, проте багато програм є корисними і без з’єднання з мережею Internet;
- кожному за потребами: оскільки розробка здійснюється окремо для кожної ОС – iOS та Android, тому є краща взаємодія з користувачем та його потребами.

- розміщення: App Store та Google Play збільшує можливості рекламної кампанії для мобільного додатку.

Недоліки розробки мобільного додатку:

- вартість розробки для обох платформ: Android та iOS не підтримують одне одного, тому розробка додатку на дві окремі платформи обійдеться дещо дорожче, ніж розробка веб-застосунку (можна згадати про міжплатформну розробку, проте вона ще розвивається, тому є певні недоліки та баги);
- обслуговування: вартість обслуговування мобільних програм є більшою, адже вони вимагають регулярного оновлення для виправлення помилок, проблем з безпекою тощо;
- відповідь-схвалення App Store: схвалення від Apple можна очікувати довго, а також вони видаляють додаток при будь-якому мінімальною порушенні;
- пам’ять: оскільки додаток потрібно завантажити, то на нього потрібно виділити пам’ять.

Таким чином, не можна однозначно сказати, а потрібно дивитися на конкретний приклад для того, щоб обрати між розробкою мобільного додатку та веб застосунку. Якщо потрібна робота в офлайн, підключення функцій мобільного пристрою, якщо важлива швидкість, обробка рахунків покладається на завантаження користувачами – тоді правильним вибором буде мобільний додаток, якщо ж навести інший приклад, такий як обмеженість бюджету, підтримка різних пристроїв, обмеженість часу, то варто обрати веб-застосунок. Ну, а розробникам програмного забезпечення на мові Swift найкраще стане в пригоді розробка додатку під операційну систему iOS, macOS, watchOS, або тим, що пов’язане з Apple Inc[5].

2.2. Огляд особливостей фреймворків SwiftUI та UIKit для побудови мобільних додатків

Чимало нових програмістів кидають погляд у бік розробки додатків для мобільних пристроїв, а оскільки Apple є компанією з найбільшим капіталом та славиться своїм брендом, тому є хорошою платформою для амбіційних кодерів. Саме тому Apple Inc. розробила власну мову, яку сама й підтримує – Swift, що слугує інструментом для розробки додатків під macOS, iOS, watchOS, tvOs. Нещодавно альтернатив за якістю та популярністю UIKit не було, проте Apple вирішила змінити ситуацію на ринку розробки під власну продукцію і випустила абсолютно новий фреймворк – SwiftUI, який дозволяє декларативно розробляти програми, а також з меншим кодом, ніж на UIKit.

Яка ж основна ідея SwiftUI та в чому відмінність?

Перш за все, SwiftUI був створений для створення додатків з тією ж складністю, проте меншим та легшим для читання кодом. На відміну від UIKit, який працює на основі storyboard – місце додавання та впровадження інтерфейсу в програмний продукт, SwiftUI, синтаксис якого дуже зрозумілий, абсолютно повністю заснований на програмному забезпеченні[6]. Безумовно дані фреймворки можуть співпрацювати та доповнювати одне одного, а оскільки UIKit є попередником SwiftUI та більшість продуктів на ринку написані саме на ньому, тому ще не так багато програмістів повністю освоїли та переключилися на його нову альтернативу. Це справа не тільки в привичці, а й в тому, що UIKit має ряд позитивних особливостей. Розглянемо основні переваги та недоліки кожного з фреймворків.

Переваги UIKit:

- стабільний та перевірений досвідом використання на проектах, які досі користуються популярністю;
- надає зручну можливість розширити базовий функціонал та стиль;

- набір компонентів, що реагують та дозволяють узгоджувати безконфліктні правила імен;
- зручне додавання компонентів на storyboard та позиціонування їх;
- простий в обслуговуванні, розроблений для легкого написання структурованого коду з можливість зручного розширювання[7].

Недоліки UIKit:

- простий користувацький інтерфейс має достатньо багато нюансів у налаштуваннях, в тому числі з обмежувачами;
- пропонує набір тільки тих компонентів, які відповідають обраному інтерфейсу користувача;
- довгий час очікування на підтвердження використання певних стилів та інтерфейсу, що може збільшити процес отримання згоди на публікацію від команди Apple;
- кожен проект є частково унікальний і займає додаткового часу на ознайомлення з деталями[7].

Чи справді декларативний фреймворк є таким прогресивним?

Переваги SwiftUI:

- функція попереднього перегляду для певного вікна без необхідності запуску основного симулятора, що зменшує час, витрачений на перевірку;
- не вимагає Interface Builder, замінюючи його компілятором Canvas, що надає можливість автоматизувати формування частини візуалізації програми;
- використання таких механізмів, як реактивне програмування, ObjectBinding, BindableObject та навіть використання фреймворку Combine;

- більша надійність у роботі з оновлення посилання, яке є в @IBOutlet в UIKit, адже SwiftUI відображає storyboard з кодом, що усуває конфілкти через синхронне використання тієї ж частини прикладної програми кількома членами команди розробників продукту;
- написання меншого коду для досягнення бажаного результату;
- зручність у відображенні сім'ї елементів на певному екрані чи частині екрану[8].

Недоліки SwiftUI:

- підтримуваність: основною проблемою є те, що фреймворк підтримується тільки з iOS версії 13 та Xcode(середовище розробки під Swift) версії 11, не працюючи на попередніх версіях;
- новий фреймворк: оскільки даний фреймворк є новим, тому виникають певні труднощі у вивченні та вирішенні складних проблем, у нагоді яких на допомогу міг прийти досвід;
- важко бачити чітку ієрархію в попередніх переглядах Xcode;
- недоступність певного функціоналу, який доступний тільки у UIKit;
- новий набір інструментів, який займе час у вивченні програмістами, що вже добре володіють UIKit;
- менша кількість професіоналів, тому важче знайти команду для розробки;
- кращі практики та засади фреймворку ще не сформовані[8].

Підсумовуючи, SwiftUI має дещо іншу ідеологію та підхід до написання коду, що ускладнює перехід для прихильників та спеціалістів фреймворку UIKit, проте SwiftUI є легшим в розумінні, простіший у читанні та зручний у використанні. Кожен з фреймворків має свої переваги та недоліки, проте це не помішає хорошому програмісту реалізувати запланований продукт[7].

2.3. Огляд особливостей можливих баз даних для мобільних додатків під операційну систему iOS

Успішність Swift як мови програмування можна оцінити, проаналізувавши ринок попиту, тобто користувачів. За статистикою, близько 900 мільйонів людей в світі володіють iPhone[15]. Згідно з результатів Forbes, Apple Inc. очолює список найвпливовіших та найдорожчих компаній у 2020 році та не тільки, а продаж мобільних телефонів даного бренду генерує близько 50% їхніх доходів, то можна зрозуміти, що користувачі Apple формують ринок. Таким чином, App Store можна вважати найкращою крамницею для продажу мобільних застосунків. Станом на 2018 рік він налічував 2.2 мільйони додатків, доступних для користувачів[14]. Важливим елементом в розробці мобільного застосунку є вибір інструментарію. Незважаючи на те, що при згадуванні про Swift поняттю баз даних приділяється не так багато уваги, адже є додатки, які не потребують цього і все відбувається на бекенді. Проте крім красивого інтерфейсу, який розважає користувача дуже часто важливо, щоб було місце, куди можна було б зберігати інформацію, щоб користувачі справді мали інтерактивний досвід роботи з застосунком.

Саме тому постає питання у тому, щоб проаналізувати та обрати найкращу базу даних для мобільного застосунку під операційну систему iOS. Те, що можна побачити в додатку в більшості випадків є безпосередньо результатом збереження, фільтрації, завантаження та пошуку, чи інших можливих операцій, що може знадобитися при роботі з певними даними, тобто є неможливою без бази даних, а ці маніпуляції є вирішальними для функціональної програми[14]. Тому обрання правильної бази даних для вашого застосунку є важливою частиною успішного застосунку. Кожен розробник під операційну систему iOS або просто продукцію Apple знає про існування багатьох можливих баз, проте

основними є SQLite, Realm та Core Data. Варто детальніше зупинитися на кожному з них.

SQLite є одним з найбільш використовуваним механізмом баз даних, які по суті є СУБД. У ній дані зберігаються безпосередньо в таблицю. Дуже зручно інтегрувати цю базу в додаток у вигляді окремої служби або у фоновому режимі, що робить її, мабуть, найулюбленішою базою даних для ОС iOS через легку реалізацію. Не менше важливою перевагою є сумісність SQLite, адже він доступний практично на будь-якій платформі: Windows, macOS, Linux, ну і звичайно Android та iOS[10]. Даний пункт є хорошим вибором для нових розробників мови Swift, які щойно змінили свій напрям і перейшли на розробку під продукцію Apple. Ключовою особливістю SQLite є те, що вона зберігається локально в iOS застосунку, тобто у пам’яті пристрою, що означає, що кожен пристрій має індивідуальну SQLite[9]. Основними перевагами SQLite є те, що можна зручно та легко здійснити конфігурацію, просто та зручно зберігати дані у таблицю з певними стовпцями, захищений доступ до даних у кілька потоків.

Realm – це СУБД з відкритим кодом. Фактично вона є MongoDB Realm під злиттям 2019 року. Ця база є об’єктно-орієнтованою, завдяки цьому розробники мають можливість здійснювати кодування зв’язків між об’єктами. На відміну від SQLite, який є представником SQL, Realm використовує C++ у своїй основі. Даний продукт у своїй меті має інтегрування в мобільні застосунки як під Android, так і під iOS, хоча він і є відносно новим конкурентом для ваших додатків, проте зарекомендував себе непогано. Мабуть основною перевагою Realm є швидкість, адже він працює швидше за свої аналоги – SQLite та CoreData. Не менш важливим моментом є те, що Realm має хорошу масштабованість, тобто може легко підтримувати велику кількість користувачів та значний обсяг даних, що є дуже хорошою перспективною можливістю для популярності будь-якої бази даних. Завдяки послузі MongoDB Realm Sync можна

значно спростити синхронізацію даних між користувачами, пристроями в режимі реального часу[13]. Не менш позитивним моментом є документація, яка ,як і документація Swift від Apple, є дуже зручною, чітко структурованою, легкодоступною та значно пришвидшує етап розуміння проблеми та розробки мобільного застосунку.

Останнє, але не менш важливе – CoreData. Особливістю даного фреймворка є те, що він розробляється, фінансується та популяризується компанією Apple. Чому фреймворк, а не база даних? Core Data має завдання не зберігати інформацію, а керувати графіком об’єктів, а функція бази даних – це просто одна з особливостей даного фреймворку[12]. Core Data може бути корисна для збереження даних для використання в автономному режимі, кешування тимчасових даних та визначення типів даних і взаємозв’язків між ними. По суті, її не можна віднести до баз даних, а скоріше до систем стійкості – проміжне програмне забезпечення, яке зберігає дані в базі, які виконують функцію шару між базою даних та даними мобільних застосунків.

Структура CoreData матиме вигляд:

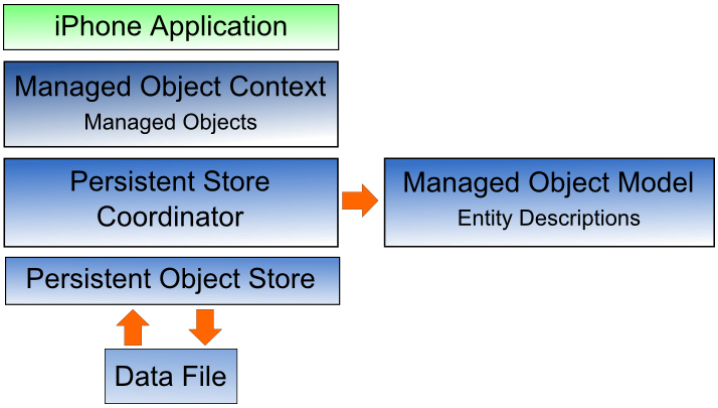


Рисунок 2.1 – структура Core Data.

Цікавою особливістю є те, що CoreData може використовувати SQLite як базу даних. Перевагою CoreData є те, що це продукт Apple Inc.. Таким чином,

вони будуть розвивати та синхронізувати його разом з розвитком інших фреймворків, як SwiftUI тощо. Зараз можна виділити певні моменти, які існують вже. Даний фреймворк є швидшим, ніж SQLite при отриманні даних. Крім того, вбудовані функції зменшують загальну кількість створеного розробниками коду для взаємодії з базою даних. Дані можуть бути організовані в бінарні, SQLite чи XML-сховища, що цілком полегшує розробку програмного забезпечення.

Таким чином, можна зробити певну структуризації у перевагах чи недоліках певних функції. Якщо розглядати використання бази даних, то SQLite є найзрозумілішій та найдоступніший, а CoreData використовувати найважче, адже вона явно управляє об'єктами в ManagedObjectContext, тому для її використання потрібне глибше розуміння API. Realm в свою чергу робить всі зміни негайно в самих блоках запису, тому при зупинці симулятора можна легко перевірити базу даних, що спрощує розуміння. У свою чергу, створення схеми в CoreData є мабуть найзручніше, адже модель, яка генерує класи, що використовуються для створення даних, створюється безпосередньо у Xcode, хоча у SQLite та Realm також немає жодних проблем, проте CoreData переважає саме у роботі з відношеннями "1 до n" та "m до n". Проте підтримка між платформами – це бонус для SQLite та Realm, адже CoreData, на відміну від попередніх двох, не є міжплатформенною базоб даних, адже як було сказано вище – це фреймворк Apple Inc. У свою чергу, Realm має базу для Android, тому логіка роботи з базами не зміниться, як і у SQLite. Realm також переважає CoreData та SQLite у швидкості, адже використовує власний двигун, який є простим та в той ж час швидким. Та ключовим моментом у виборі бази даних є безпека особистої інформації користувачів, проте як CoreData, так і SQLite та Realm забезпечують достатньо хороший захист даних[11].

2.4. Правила публікації iOS-додатку в App Store

Компанія Apple та відділ App Store достатньо чітко та прискіпливо працюють з продуктами, які пропонують розробники для публікації в крамниці. Тому потрібно, щоб додаток проходив під ряд вимог, запропонованих компанією. Хорошим моментом у цьому є те, що Apple у власній документації виставив всі вимоги, яких потрібно дотримуватися, щоб пройти етап публікації або ж, яких варто дотримуватися вже після публікацій, щоб мобільний застосунок продовжував існувати в Apple Store.

Всі правила розділені на п’ять категорій: безпека, ефективність, бізнес, дизайн та законність. Проте будь-яке правило може змінитися з метою вдосконалення, тому варто вдосконалювати власний продукт разом з Apple. Цікавим моментом є те, що оскільки до ринку мобільних застосунків є доступ і у дітей, а батьківський контроль не завжди може захистити дитину від будь-якого негативного впливу, то компанія повідомляє, що вони слідкують за дітьми, а тому й за тим, що діти встановлюють. Будь-які спроби обманути систему контролю будуть суворо покарані: всі продукти власника будуть видалені, а розробники будуть заблоковані у подальшій можливості публікувати інші мобільні застосунки в App Store.

Перед офіційним поданням в App Store варто самостійно ознайомитися з ключовими моментами, серед яких:

- перевірка додатку на справність, адже помилки є неприпустими для підтримання якості бренду Apple;
- перегляд актуальності та якості інформації, адже інформація про програму та метадані повинні бути повними;
- детальне пояснення неочевидних функцій та покупок у програмі до приміток огляду додатків;
- дотримання всіх правил з п’яти категорій.

Безпека є основною категорією, адже захист особистої інформації користувача є ключовим моментом, тому поширення особистої інформації третім обличчям є неприпустимий та карається вилученням додатку з App Store. Застосунок не повинен містити образливого, нечутливого вмісту та не повинен пошкоджувати пристрій користувача. Також мобільний застосунок не повинен загрожувати фізичною шкодою для людини, таких як заохочування споживання тютюну чи подібної продукції, наркотиків чи алкоголю. Не можна порушувати права інтелектуальної власності. Суворі правила накладаються на мобільні застосунки, які вказані в дитячу категорію, проте порушують їхні правила.

Для перевірки продуктивності варто зразу переконуватися у тому, що додаток є остаточною версією з усіма метаданими, тому клієнти повинні мати можливість переглянути їх точність, а також варто здійснити перевірку на працездатність, адже App Store – не місце для тестування програмного забезпечення, тому для бета-тестування пропонується TestFlight. Достатньо багато правил накладаються на вимоги до програмного забезпечення.

Розгляд коректності додатку може бути затриманий через неочевидність бізнес-моделі. Не менш важливим пунктом є право відкинути дорогий застосунок, який може мати на меті обдурити користувачів надто високими цінами, тому потрібно розумно підходити до монетизації та категорії бізнес.
Не варто маніпулювати відгуками, завищувати рейтинги графіків фальшивими відгуками чи взаємодіяти зі сторонніми службами, які можуть здійснити це від вашого імені.

Не менш важливим моментом є підхід до дизайну, адже ще Стів Джобс навчив клієнтів цінувати красоту та зручність продуктів. Тому Apple не втручається у дизайн, проте мобільний застосунок повинен підходити під мінімальні стандарти для схвалення. Варто пам’ятати, що потрібно

забезпечувати функціонування додатку та залучення нових і існуючих клієнтів, адже додатки, які пропонують поганий досвід та показують незадовільні цифри у користувачах можуть бути видалені з App Store у будь-який час. Важливою складовою є унікальність дизайну, адже Apple надихає на створення та реалізацію нових ідей, а не копіювання існуючих популярних аналогів, тому порушення інтелектуальної власності карається вилученням застосунку. Функції додатку повинні виводити його за межі перепакованого веб-сайту, тому некорисні додатки можуть бути видалені. Застосунки, які виконують існуючий функціонал, наприклад, додаток-книжка повинен бути надісланий до Apple Books Store. Також варто уникати спаму у власних додатках, а різні версії для певних специфічних областей повинен подаватися як один. Якщо застосунки містять сторонні служби входу, такі як Google чи Facebook, то він також повинен мати аналог – Sign in with Apple.

Додатки повинні не тільки задовольняти правила усіх попередніх розділам, а ще й відповідати законодавчим вимогам у будь-якому місці, де додаток може бути чи є наданим. Основними є: конфіденційність – захист користувача є головним в екосистемі Apple, інтелектуальна власність – точніше не порушення її, використовувати тільки власні доробки, або ліцензію на використання. Ігри, азартні ігри повинні проходити ряд вимог. Управління мобільними пристроями можна здійснювати тільки за згодою користувача та компанії Apple, тому про це потрібно запитувати. Це є програмою, які в основному пропонують комерційні підприємства. Останнє, але не менш важливе – це повага до всіх людей: клієнтів, працівників Apple тощо.

Після подачі додатку та подальшого відхилення можна подати апеляцію, або виправити деякі моменти і подати повторне звернення. У разі схвалення варто пам’ятати, що контроль здійснюється постійно і дотримуватися правил потрібно завжди, адже Apple турбується про своїх клієнтів[16].

РОЗДІЛ 3. ОПИС РЕАЛІЗАЦІЇ ПРОГРАМНОГО ПРОДУКТА

3.1. Аналіз технічного завдання

Основною ціллю даного продукту є здійснення контролю над власними або ж сімейними фінансами. Для спільного сімейного контролю потрібно здійснити авторизацію, щоб можна було маніпулювати, додавати чи змінювати інформацію одного аккаунта з різних мобільних пристроїв iPhone. Без авторизації немає доступу до додатку, адже він не працює з анонімними користувачами, тому тільки авторизований користувач може додавати свої доходи, фінанси, здійснювати аналітику, побачити залишок на рахунку. Помилково додані транзакції можна видалити, проте після видалення вони не відновлюються. Додаток працює лише на мобільних пристроях компанії Apple – iPhone та може бути доступним тільки у App Store (дана можливість не є реалізованою).

Для контролю версій та зручного маніпулювання застосунку використано веб-сервіс для розробки програмного забезпечення – GitHub, де створений закритий репозиторій. Здійснення авторизації відбувається завдяки платформі Firebase, яке призначене для розробки мобільних та веб-застосунків. За безпечне збереження інформації про користувачів та хешування паролів відповідає безпосередньо Firebase. За допомогою даної платформи можна не тільки здійснювати само-авторизацію, але й вхід через Google, Facebook, Apple та інші аккаунти.

3.2. Обґрунтування алгоритму та структури програми

У даному проекті багато view взаємодіють між собою і реалізована достатньо багато окремих представлень: авторизація, вікно-аккаунта, вікно-транзакцій, вікно-діаграми тощо. Вся структура класів проекту побудована навколо основного класу View. Наприклад, рисунок 3.1, який демонструє UML-діаграму структур для представлення вікон авторизацій.

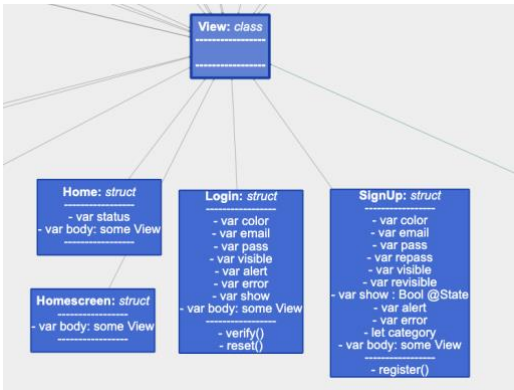


Рисунок 3.1 – діаграма представлення авторизації

Загальна діаграма доступна в додатку А.

У проекті використано дизайн патерн MVVM, що одним з найпопулярніших для застосунків під операційну систему iOS, який передбачає концепцію view-model. Шаблон використовує три шари: model, view і view-model. У свою чергу модель – це дані програми, на яких та завдяки яким працює додаток. Вікно або ж представлення – це те, що бачить користувач, тобто його інтерфейс. Останній шар – view-model – є своєрідним зв'язком між цими компонентами. Основними перевагами даного патерну проектування над іншим популярним аналогом MVC – це зменшена складність, виразність та зручність у тестуванні[17]. Адже багато ділової логіки вилучається з контролера, а в свою чергу модель краще її подає, відповідно здійснювати тестування є значно простішим. Структуру папок можна побачити на рисунку 3.2

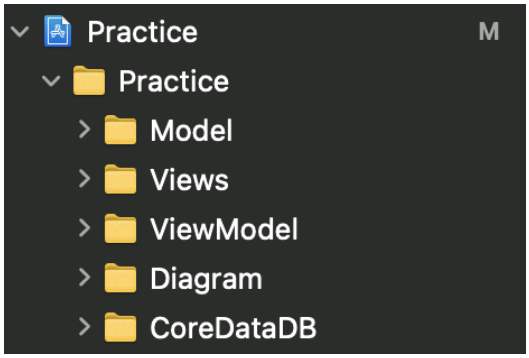


Рисунок 3.2 – структура папок проекту

Для зберігання інформації було обрано фреймворк від Apple – CoreData. Незважаючи на те, що у проекті не було складних сценаріїв використання баз даних, адже відбувається взаємодіями з кількома таблицями, проте за мету дослідження було покладено ознайомитися саме з такою можливістю зберігання інформації як використання CoreData. Вся інформація зберігається у таблицю (рисунок 3.3) і відбувається взаємодія за допомогою класу ManageData(рисунок 3.4) через NSManagedObjectContext – частина CoreData, яка відповідає за загрузку даних з NSPersistenceStore і NSManagedObjectModel, а також здійснює зберігання та хешування. Результатом дослідження стало використання CoreData у проекті та формування висновку, що у таких прикладах продуктів достатньо використовувати SQLite, проте воно не є доробком Apple, тому не входить у стандартний пакет Xcode.

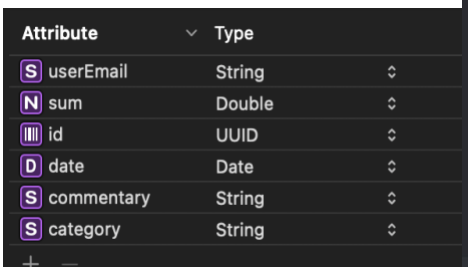


Рисунок 3.3 – таблиця транзакції

```
public class ManageData {

    static let shared = ManageData(moc: NSManagedObjectContext.current)

    var managedContext: NSManagedObjectContext

    private init(moc: NSManagedObjectContext) {
        self.managedContext = moc
    }
}
    extension NSManagedObjectContext {
        static var current: NSManagedObjectContext {
            let appDelegate = UIApplication.shared.delegate as! AppDelegate
            return appDelegate.persistentContainer.viewContext
        }
    }
```

Рисунок 3.4 – робота з CoreData

Як було зазначено вище, у проекті використовується платформа Firebase. Для її підключення знадобився CocoaPods – менеджер залежностей для Swift, за допомогою якого можна швидко і легко підключати будь-які бібліотеки чи утиліти. Всі залежності, які необхідні для проекту визначаються у Podfile, який має вигляд: 'Practice' **do** 'Firebase/Auth' 'Firebase/Analytics' **end.**

Алгоритм підключення наступний: спершу потрібно підключити CocoaPods, у podfile написати залежність, через командну стрічку здійснити синхронізацію проекту з підключеними залежностями.

3.3. Опис та пояснення вибору середовища розробки

Середовищем для розробки програмного забезпечення було обрано Xcode, продукт створений Apple Inc. Це IDE, за допомогою якого можна створювати проекти під продукцію Apple, а також це єдине засіб написання програмного забезпечення під macOS. Xcode пропонує комплексне рішення для будь-якої задачі, від створення безпосередньо проекту до публікації у крамниці – App Store. Основним позитивним моментом є те, що Apple дотримується ідеології зручності та комфорту для користувача і в цьому продукті, тому дизайн є дуже привабливим та зрозумілим навіть для нових користувачів. Створення проекту можна здійснити таким чином, щоб все відбулося автоматично, а в цього буде готовий до запуску проект на будь-якому симуляторі iPhone.

Уся інформація зберігається на одному вікні, тому інтерфейс можна охарактеризувати як одновіконний: простір з кодом, дебаг та інші особливості, як структура файлів проекту, детальна інформація обраного елементу чи елементи, які можна прикріплювати на view – усі ці компоненти знаходяться в одному фреймі. Завдяки цьому легко можна здійснювати навігацію потрібними компонентами та самостійно масштабувати екран частини кожного з цих компонентів. Не можна не згадати про підтримування utf-8 кодування, завдяки якому можна програмувати за допомогою емодзі – мовою ідеограм там смайликів.

Контроль версій. Як і інші IDE, наприклад IntelliJ IDEA чи інші продукції Jet Brains, або ж можна продукт IBM – Eclipse, Xcode підтримує Git, але більше того, розробники компанії Apple створили дуже приємний користувацький інтерфейс для роботи з ним. Можна здійснювати коміти, пуші чи пули, або ж переглянути існуючі гілки проекти безпосередньо у Xcode, тому не потрібно буде використовувати ні консоль, ні інших сторонніх ресурсів для роботи з Github.

Проте мабуть найулюбленіша особливість Xcode для більшості програмістів – це можливість здійснювати тестування власними очима. Адже завдяки симуляторам і підключенню проекту навіть до власного телефону, можна здійснювати будь-які перевірки на коректну роботу. Симулятор є точною копією справжніх мобільних пристроїв, відрізняються тільки кількістю вбудованим функціоналу: камеру чи деякі інші вбудовані застосунки не можна використовувати на емульованому девайсі. Саме тому це не тільки можливість зручно та доступно здійснювати написання користувацького інтерфейсу, але й можливість тестувати власні доробки.

Останнє, проте не менш важливе, Xcode – це продукт Apple. Оскільки компанія всіма своїми вчинками та нововведеннями позиціонує себе як незалежна організація, яка не зважає на продукцію інших фірм, то працювати з продукцією Apple варто безпосередьо в середовищі програмного забезпечення, розроблене Apple. Це забезпечить впевненість в підтримці в користувачів, а також постійному оновленні. Власне саме через цю причину варто обирати Xcode для розробки додатків під операційну систему iOS. Варто тільки пам’ятати, що для нього знадобиться близько 20Гб пам’яті для встановлення, у разі використанні певних розширень, в чому можна переконатися на рисунку 3.5

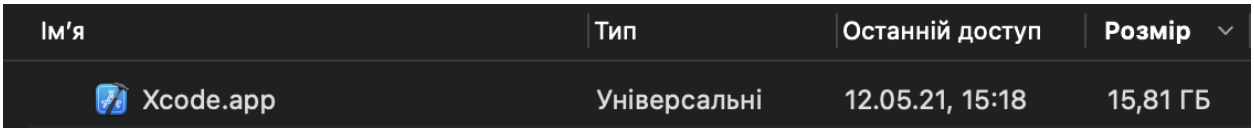


Рисунок 3.5 – розмір Xcode.app

3.4. Опис розробки мобільного застосунку та принципи роботи

Робота над застосунком розпочалась з створення логотипу та назви, адже саме завдяки цим компонентам люди зможуть знайти додаток у AppStore, тому не можна нехтувати важливістю цього. Оскільки є багато аналогів таким додаткам, а предметною областю даного застосунку є фінанси, то потрібно було придумати оригінальний логотип та назву. Поточна назва проекту – Expense

Manager або ж Expense Mngr. для публікації в AppStore у зв'язку з обмеженістю символів у назві, оскільки ідея полягає у тому, щоб не тільки відстежувати загальний баланс, а детальніше можна було б здійснювати саме перегляд по категоріям за витратами, адже дохід цікавить нас меншою мірою. Логотип повинен бути яскравим та таким, щоб запам'ятався, тому було прийнято рішення надати йому яскравого кольору, а також відобразити руку, яка тримає цент і додати назву продукту, незважаючи на те, що додаток розроблений у чорно-білих кольорах, кольором логотипу визначено #FF5976 (див. рис 3.6)

Рисунок 3.6 – фінальний варіант логотипу

Наступний етап полягав у визначенні архітектури проекту та структури основної програми. Застосунок побудований з використанням дизайн патерну MVVM, про який вже було згадано вище. Він обраний через гнучкість архітектури та зручність у додаванні нового коду. Безумовно, не можна назвати даний патерн ідеальним, проте він пропонує певну альтернативу недосконалому патерну MVC. У застосунку містяться як файли, які відповідають за відображення, тобто вони є шаром View в архітектурі MVVM, а також ViewModel, які наші представлення використовують завдяки @Binding та @ObservedObject, використання яких можна побачити на рисунку 3.7.

```
struct Edit: View {
    @Environment(\.presentationMode) var presentationMode: Binding<PresentationMode>

    @ObservedObject var transactionVM = TransactionListViewModel()


    @State var addNewCategory: Bool
    @State var checkExpense: Bool = true
```

Рисунок 3.7 – обгортки @Binding та @ObservedObject як приклад MVVM

Дане структура-представлення, що є View в архітектурі використовує клас TransactionListViewModel як свій ViewModel, тобто клас, який виконує усю логіку даного вікна. State – це стани, які відслідковують поведінку нашої змінної та можуть бути передаваними з view на view. Фактично SwiftUI і є MVVM в явному вигляді, адже навіть без належної архітектури в більшості випадків використовують обгортки @Published, @Binding, які свого роду виконують функції патерну. Обмін інформації між різними вкладками-вікнами здійнюється завдяки @ObservedObject. Це ж можна сказати і про CoreData, яка має обгортку @FetchRequst для отримання інформації з бази. Для цього варто розглянути запропоновану мною функцію, яка працює з CoreData – getAllTransaction(), запропонований на рисунку 3.8.

```
func getAllTrasaction() -> [Transaction] {
    var transfer = [Transaction]()
    let bdRequest: NSFetchRequest<Transaction> = Transaction.fetchRequest()
    let sdSortDate = NSSortDescriptor.init(key: "date", ascending: false)
    bdRequest.sortDescriptors = [sdSortDate]
    do { transfer = try self.managedContext.fetch(bdRequest)}
    catch {print(error)}
    return transfer
}
```

Рисунок 3.8 – отримання інформації з CoreData

Дана функція повертає масив об’єктів класу Transaction, який є автоматичним класом-генерацією таблиці Transaction, тобто містить всі поля, що є в таблиці. Завдяки fetchRequst() здійснюється отримання інформації з таблиці.

Інформація з бази зберігається в об’єкті TransactionViewModel, який містить усі поля, необхідні для внесення в базу даних, є два способи ініціалізації: передаючи туди транзакцію або ж передаючи всі поля, які необхідні для створення цієї транзакції. Даний код можна переглянути в додатку Є – TransactionViewModel.

Запуск додатку для неавторизованих користувачів направляє користувачів на LoginView (див. Додаток Б), яка надає можливість увійти в систему, або ж використати кнопку зареєструватися і перейти на нове вікно SignUpView (див. Додаток В), де користувач вводить необхідну інформацію та здійснює реєстрацію в системі. Будь-які помилки вирішуються за допомогою структури ErrorView(див. Додаток Г), за допомогою якої користувач отримує сповіщення.

База користувачів зберігається в хмарі, з використанням платформи Firebase, про яку вже було згадано вище. Саме за допомогою нею відбувається шифрування та відправлення особистих даних на сервер, тому авторизація є безпечною для користувачів.

У разі, якщо користувач вже зареєстований, проте забув пароль, йому на пошту приходить повідомлення з підтвердженням зміни паролю та введенням нового (див. рис 3.10).

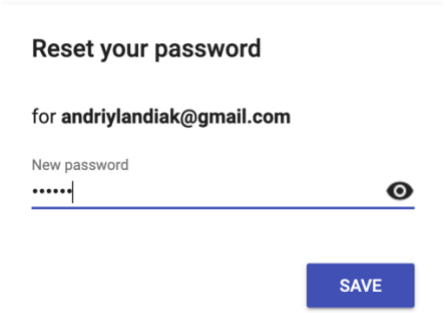


Рисунок 3.10 – зміна паролю

Після успішної авторизації користувача не викидатиме з системи автоматично, тому не буде потреби здійснювати авторизацію ще раз. Програма складається з трьох основних вікон:

- Аккаунт: відображення особистої інформації про користувача, з можливістю обрати фотографію з галереї або ж видалити наявну, кнопкою категорій, які відповідають за редагування наявних або ж додавання нових категорій та переглянути баланс(див. Додаток Ґ).

- Список транзакцій: відображення доступних транзакцій за датою(див. рисунок 3.10), а також можливість додавання нового доходу чи витрати(повний екран - див. додаток Д)

Рисунок 3.11 – відображення транзакцій

- Аналітика: відображення діаграми витрат за категоріями з можливістю здійснювати фільтрацію за місяцями, за роками чи за весь час(див. рисунок 3.12, для повного екрану див. додаток Е).

Рисунок 3.12 – фільтрація за місяцем

Дані три основні блоки розміщуються у вигляді кнопок в нижній панелі екрану – TabView , код виконання наведений на рисунку 3.12

```
struct Homescreen : View {
    var body: some View{
            TabView {
             AccountView(addNewCategory: false).tabItem { Label("Account", systemImage: "house")}
             ContentView().tabItem { Label("Transaction", systemImage:
                 "arrow.left.arrow.right.circle")}
             DiagramView(changeFilter: false, value: 2).tabItem { Label("Analytics", systemImage:
                 "banknote") }
            }.colorMultiply(.white).edgesIgnoringSafeArea(.top).accentColor(.black)
    }
}
```

Рисунок 3.12 – код TabView

ВИСНОВКИ

Отже, результатом роботи стало дослідження можливостей нового фреймворку SwiftUI від Apple Inc., платформи розробки мобільних та веб-застосунків Firebase від Alphabet Inc і фреймворком для маніпулювання базами даних та не тільки - Core Data для розробки мобільного застосунку під операційну систему iOS. Поставлена задача була реалізована та виконана, продукт готовий до використання.

Як результат дослідження, яке було описане в даній курсовій роботі, можна виділити декілька важливих моментів, які сформувалися шляхом обрання даного підходу до реалізації мобільного застосунку, серед яких варто зазначити переваги та недоліки кожного обраного інструментарію.

Бонусом для використання SwiftUI у даному проекті є велика кількість різних View, які взаємодіють між собою та обмінюються інформацією, яку потім записують у базу даних. Адже головною перевагою SwiftUI є саме декларативний стиль, тому дуже зручно і легко відображати велику кількість елементів на екрані мобільного телефону, а недоліком – його новизна, тобто є ймовірність стикнутися з проблемою вперше і не буде готових способів її вирішення, що вимагає від розробника чітке та глибоке розуміння всіх процесів.

Firebase гарно себе проявив для безпечного збереження користувачів у хмарі компанії Google, проте взаємодія з базами даних, у цьому випадку з Core Data для розробки під iOS не є зручною, тому варто використовувати бази даної платформи, що може призвести до значного переписування вже існуючого функціоналу. Перевагою використання баз даних від Firebase над використанням вбудованих в пам’ять телефону баз є той факт, що у випадку

видалення користувачем додатку ще певний час інформація може зберігатися в хмарі, а у Core Data чи її аналогів немає такої можливості.

Даний проект також має багато можливостей для подальшого розвитку, адже ще є велика кількість додаткового функціоналу, який був би корисний користувачам, наприклад, синхронізація даних з банківських карт, що вимагає використання корпоративного API, яке можна отримати безпосередньо в банку, тому це ускладнює та збільшує процес реалізації, проте значно полегшує життя користувачеві. Також можна додатково впровадити створення боргу, як транзакції, що вимагає відмінної таблиці, від вже наявних. Ще одним варіантом розвитку є створення категорій для доходів та відповідно аналітики за новими категоріями.

Як висновок, можна з впевненістю сказати, що ринок мобільних додатків є на великому підйомі, проте ще є багато аспектів в існуючих програмах, які можна покращити, тому необхідно ефективно підходити до втілення ідей як малого, так і великого бізнесу і з розумом використовувати всі можливості даних фрейворків чи платформ для розробки та успішного функціонування їх у складі єдиної системи – мобільного додатку під операційну систему iOS.

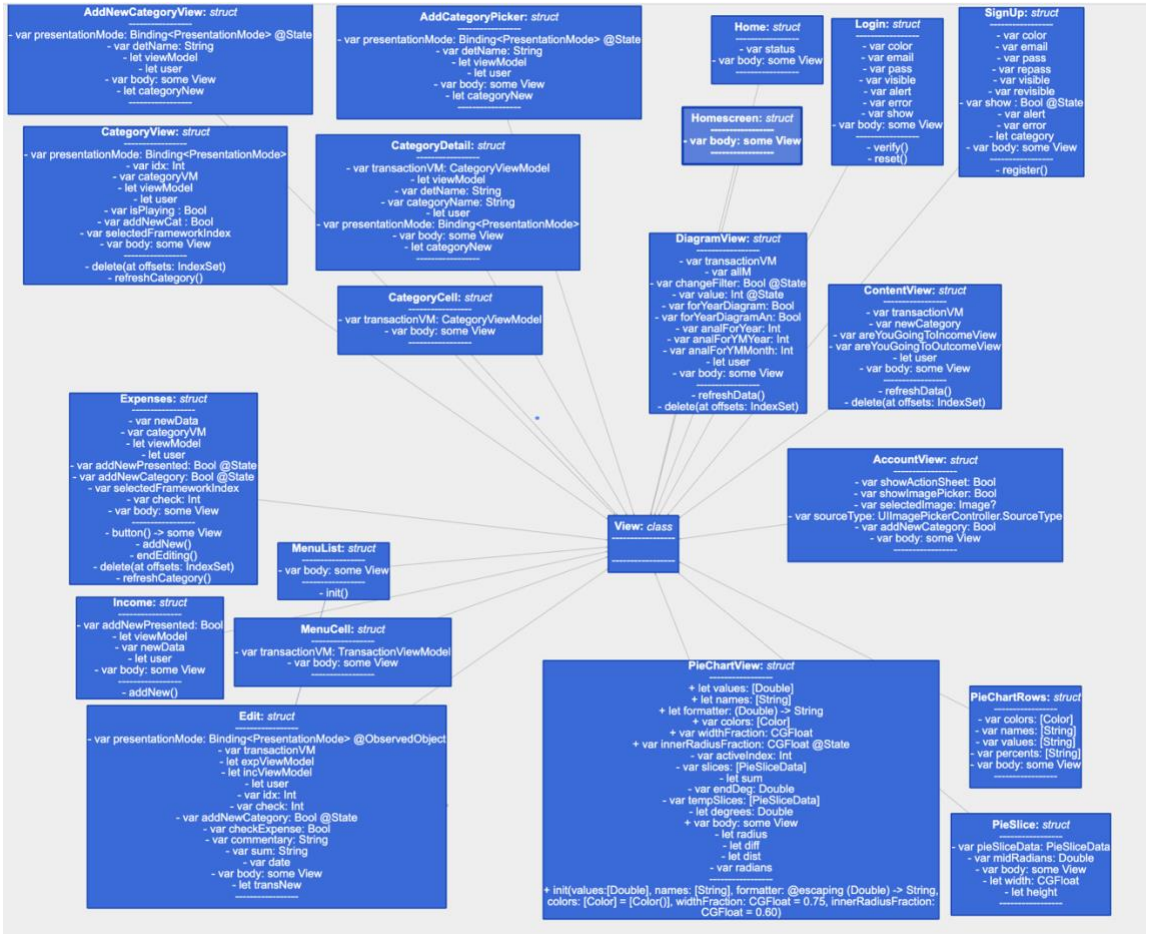
Список використаної літератури

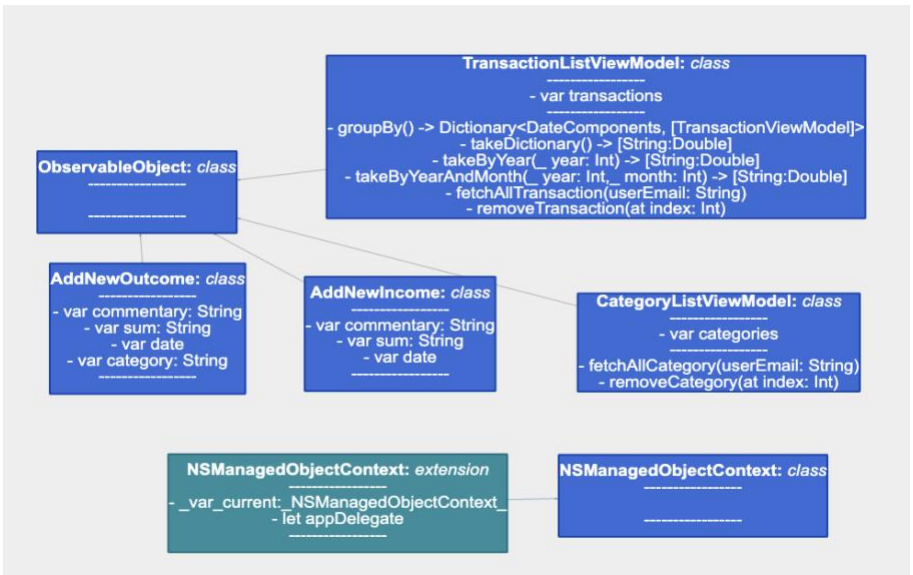
1. [Електронний ресурс] Why Financial Literacy is So Important - https://www.investopedia.com/articles/investing/100615/why-financial-literacy-and-education-so-important.asp
2. [Електронний ресурс] What is a trader? - https://www.investopedia.com/terms/t/trader.asp
3. [Електронний ресурс] Mobile Vs. Desktop Usage - https://www.broadbandsearch.net/blog/mobile-desktop-internet-usage-statistics
4. [Електронний ресурс] The benefits of using web-based applications - https://www.geeks.ltd.uk/about-us/blog/details/eQU5Ip/the-benefits-of-using-web-based-applications
5. [Електронний ресурс] The pros and cons of building a Mobile app vs a Web app - https://designli.co/blog/the-pros-and-cons-of-building-a-mobile-app-vs-a-web-app/
6. [Електронний ресурс] SwiftUI - https://developer.apple.com/xcode/swiftui/
7. [Електронний ресурс] Will SwiftUI kill UIKit?- https://topdevs.org/blog/will-swiftui-kill-uikit
8. [Електронний ресурс] SwiftUI in Production? - https://betterprogramming.pub/swiftui-in-production-6-pros-and-cons-you-need-to-consider-69ace40a1b46
9. [Електронний ресурс] Experimenting With SQLite in iOS - https://medium.com/capital-one-tech/experimenting-with-sqlite-in-ios-ae9dec92dbaf
10. [Електронний ресурс] How to Use SQLite to Manage Data in iOS Apps - https://www.appcoda.com/sqlite-database-ios-app-tutorial/

11. [Електронний ресурс] Core Data vs Realm. What to choose as a Database? - https://agilie.com/en/blog/coredata-vs-realm-what-to-choose-as-a-database-for-ios-apps
12. [Електронний ресурс] Core Data - https://developer.apple.com/documentation/coredata
13. [Електронний ресурс] Realm's mobile database - https://realm.io/
14. [Електронний ресурс] What is the Best Database for iOS Apps? - https://trio.dev/blog/best-ios-database
15. [Електронний ресурс] The Economics of the Iphone - https://www.investopedia.com/articles/investing/022316/economics-iphone-aapl.asp
16. [Електронний ресурс] App Store Review Guidelines - https://developer.apple.com/app-store/review/guidelines/
17. [Електронний ресурс] iOS MVVM Tutorial https://www.raywenderlich.com/6733535-ios-mvvm-tutorial-refactoring-from-mvc
18. [Електронний ресурс] Introducing XCode 12 https://developer.apple.com/xcode/

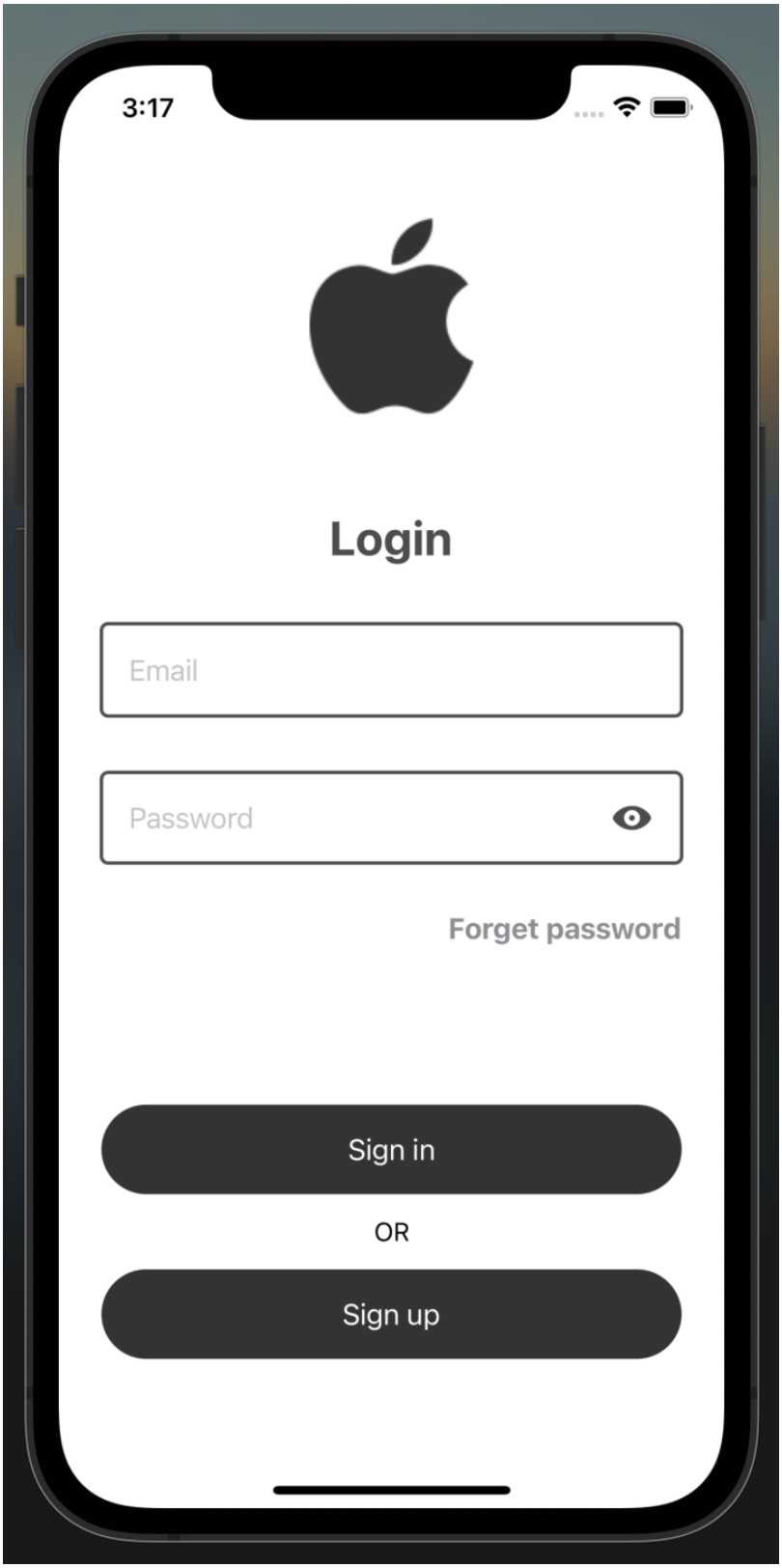
Додатки

Додаток А. Діаграма проекту

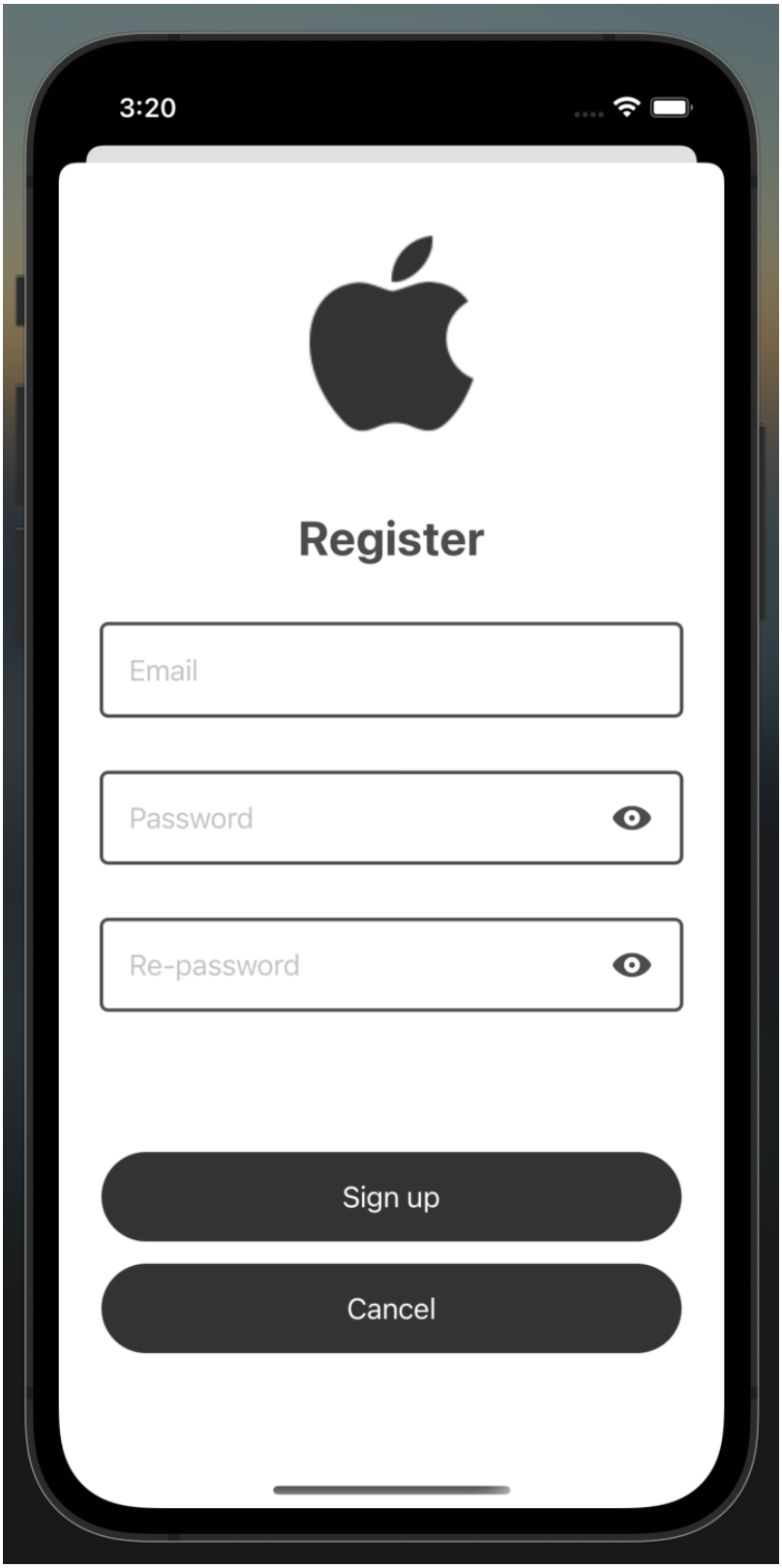




Додаток Б. LoginView



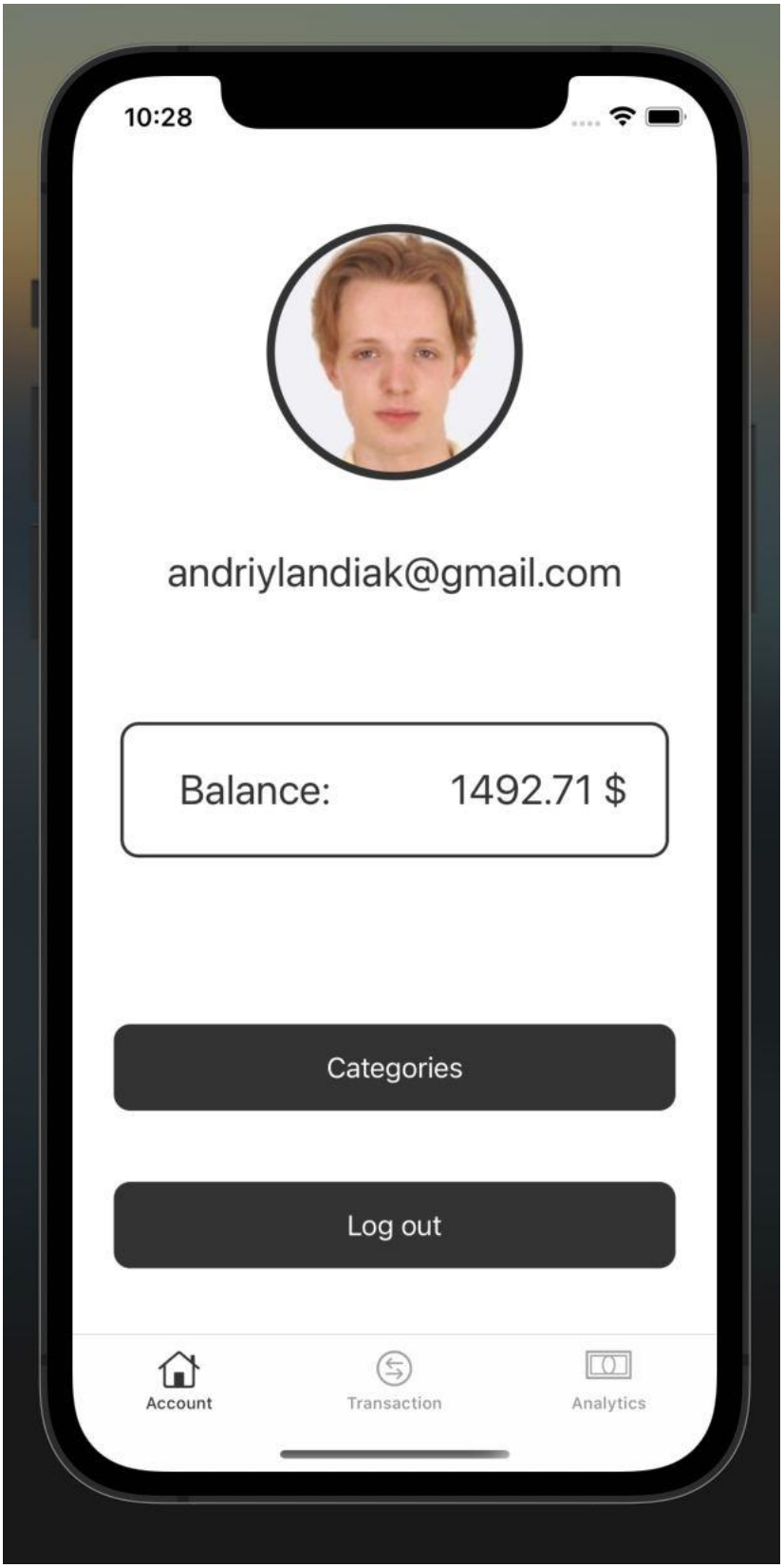
Додаток В. SignUpView



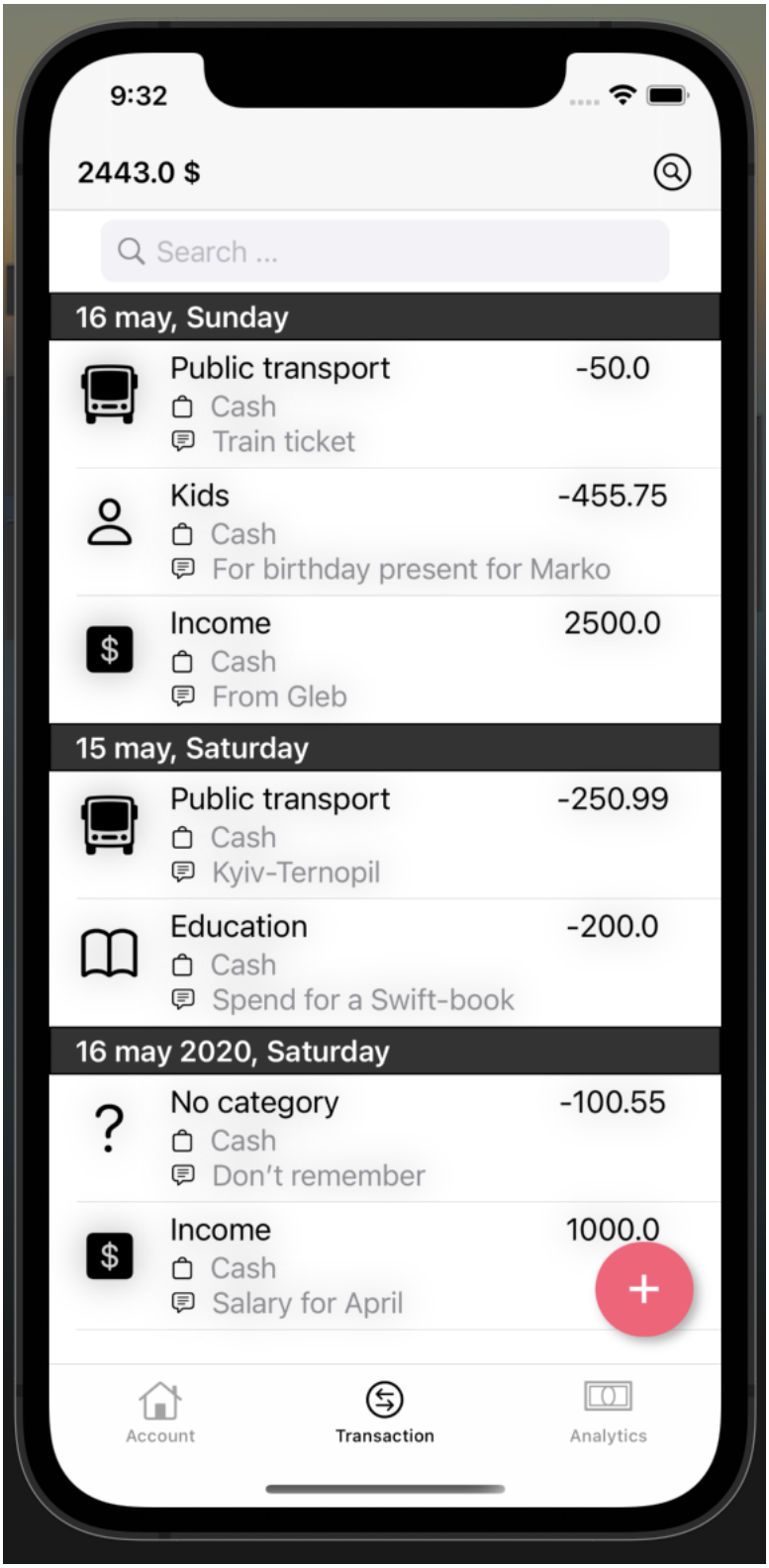
Додаток Г. Код структури опрацювання помилок

```
import SwiftUI

struct ErrorView : View {
    @State var color = Color.black.opacity(0.7)
    @Binding var alert : Bool
    @Binding var error : String
    var body: some View{

        GeometryReader{_ in
            VStack{
                HStack{
                    |
                    Text(self.error == "RESET" ? "Message" :
                        "Error").font(.title).fontWeight(.bold).foregroundColor(self.color); Spacer()}
                .padding(.horizontal, 25)

                Text(self.error == "RESET" ? "Password reset link has been sent successfully" :
                    self.error).foregroundColor(self.color).padding(.top) .padding(.horizontal, 25)

                Button(action: {self.alert.toggle()})
                {
                    Text(self.error == "RESET" ? "Ok" :
                        "Cancel").foregroundColor(.white).padding(.vertical).frame(width:
                        UIScreen.main.bounds.width - 120)
                }
                .background(Color("LoginColor"))
                .cornerRadius(10)
                .padding(.top, 25)

            }
            .padding(.vertical, 20)
            .frame(width: UIScreen.main.bounds.width - 70)
            .background(Color.white)
            .cornerRadius(15)
        }
        .padding(.horizontal, 40)
        .padding(.vertical, 50)
        .background(Color.black.opacity(0.35).edgesIgnoringSafeArea(.all))
    }
}
```

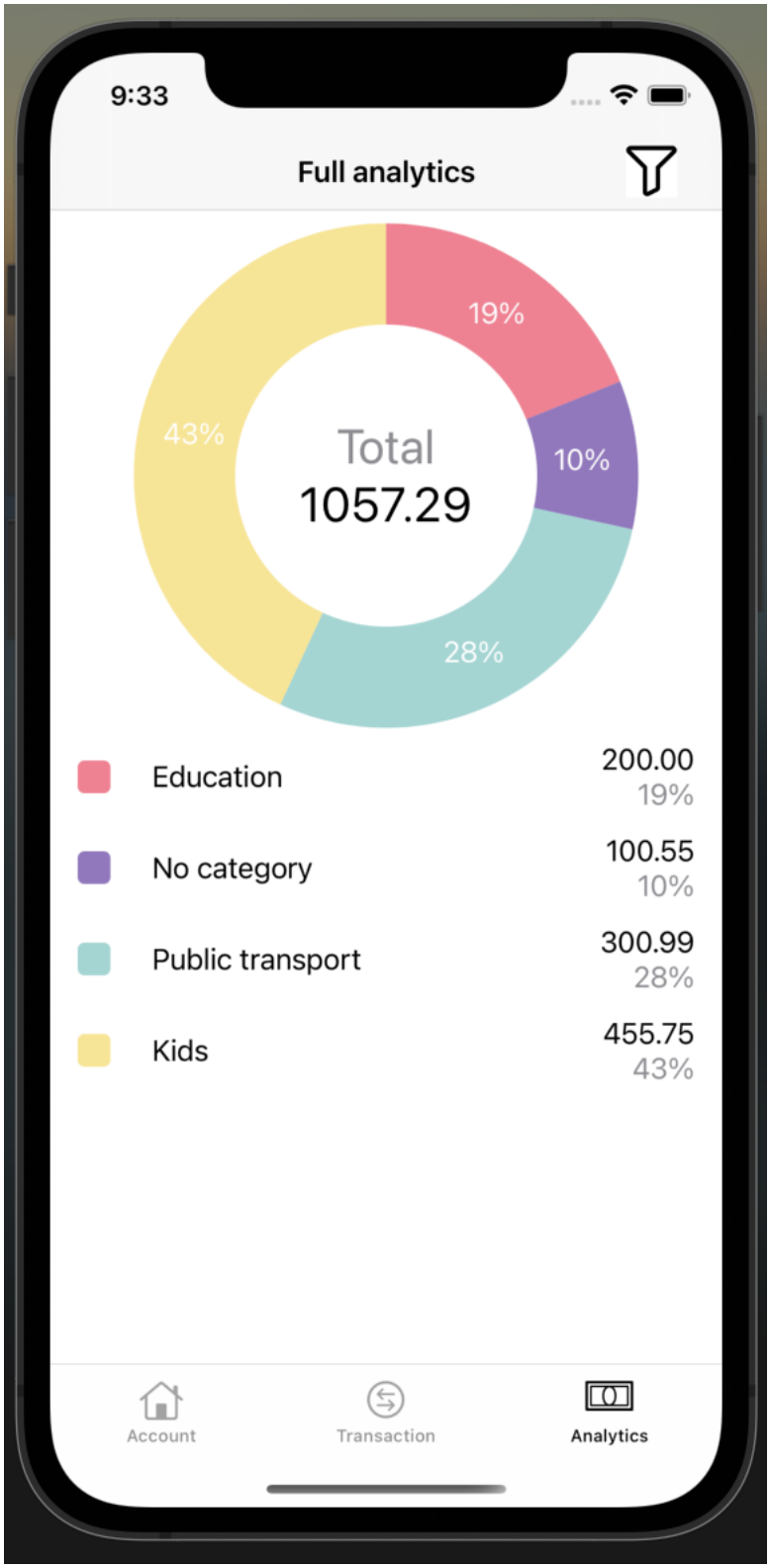
Додаток Ґ. Вікно аккаунта користувача



Додаток Д. Вікно списку транзакцій

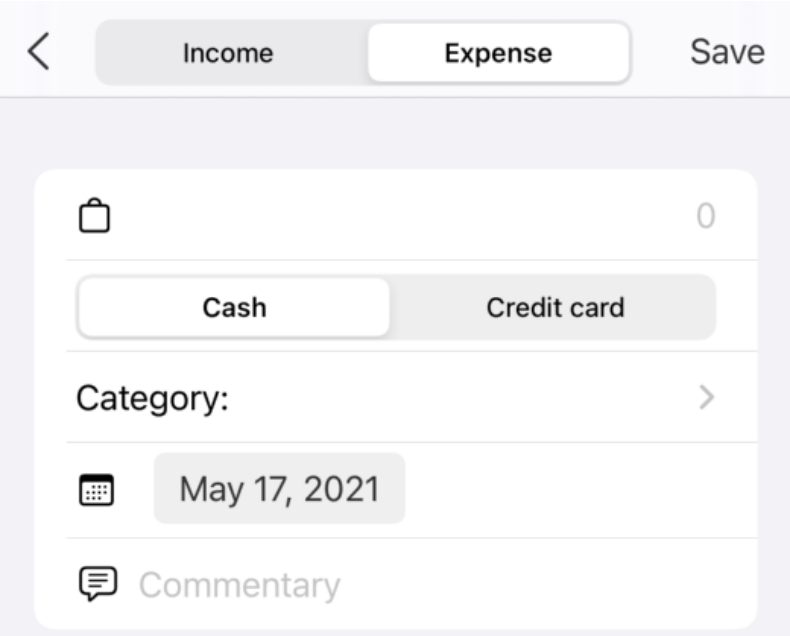


Додаток Е. Вікно аналітики

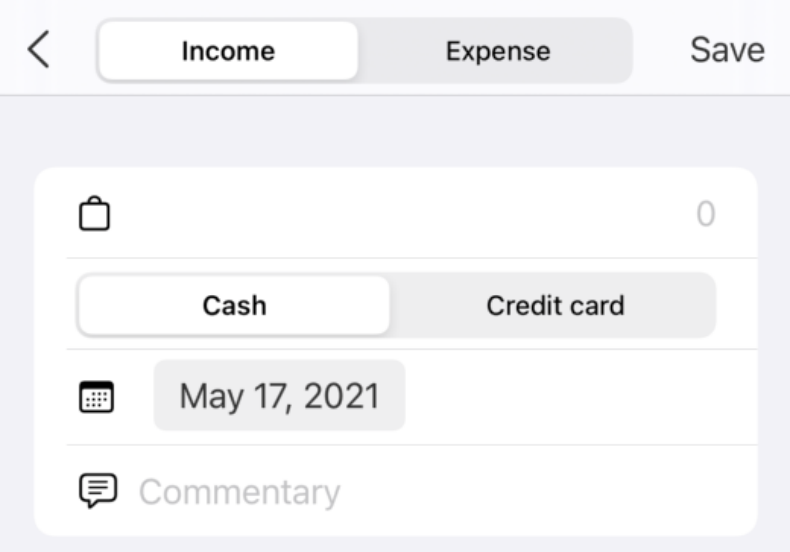


Додаток Є. Меню додавання транзакцій

a) Витрати



b) Дохід



Додаток Ж. TransactionViewModel

```
class TransactionViewModel: Identifiable {
    var id: UUID
    var sum: Double
    var date: Date
    var category: String
    var commentary: String
    var userEmail: String

    init(transaction: Transaction) {
        self.id = transaction.id ?? UUID()
        self.sum = transaction.sum
        self.date = transaction.date ?? Date()
        self.category = transaction.category ?? ""
        self.commentary = transaction.commentary ?? ""
        self.userEmail = transaction.userEmail ?? ""
    }

    init(id: UUID, sum: Double, date: Date, category: String, commentary: String, userEmail:
        String) {
        self.id = id
        self.sum = sum
        self.date = date
        self.category = category
        self.commentary = commentary
        self.userEmail = userEmail
    }
}
```

Додаток 3. Додавання та зміна транзакції

```
class AddIncomeViewModel {

    func addTransaction(transaction: TransactionViewModel) {
        ManageData.shared.addTrasaction(id: transaction.id, sum: transaction.sum, date:
            transaction.date, category: transaction.category, commentary: transaction.commentary,
            userEmail: transaction.userEmail)
    }

    func updateTransaction(transaction: TransactionViewModel) {
        ManageData.shared.updateTrasaction(id: transaction.id, sum: transaction.sum, date:
            transaction.date, commentary: transaction.commentary)
    }
}
```

```
class AddExpensesViewModel {

    func addTransaction(transaction: TransactionViewModel) {
        ManageData.shared.addTrasaction(id: transaction.id, sum: -transaction.sum, date:
            transaction.date, category: transaction.category, commentary: transaction.commentary,
            userEmail: transaction.userEmail)
    }

    func updateTransaction(transaction: TransactionViewModel) {
        ManageData.shared.updateTrasaction(id: transaction.id, sum: -transaction.sum, date:
            transaction.date, commentary: transaction.commentary)
    }
}
```

Додаток І. Робота з категоріями

```
struct CategoryViewModel {
    var id: UUID
    var name: String
    var userEmail: String


    init(category: CatEntity) {
        self.id = category.id ?? UUID()
        self.name = category.name ?? ""
        self.userEmail = category.userEmail ?? ""
    }

    init(id: UUID, name: String, userEmail: String) {
        self.id = id
        self.name = name
        self.userEmail = userEmail
    }
}
```

```
class AddUpdateCategory {

    func addCategory(category: CategoryViewModel) {
        ManageData.shared.addCategory(id: category.id, name: category.name, userEmail:
            category.userEmail)
    }

    func updateCategory(category: CategoryViewModel) {
        ManageData.shared.updateCategory(id: category.id, name: category.name)
    }

    func addDefaultCategoryForUser(userEmail: String) {
        let array:[String] = ["Cafe & Restaurant", "Product", "Education", "Public transport",
                    "Store", "Fitness", "Kid", "Else"]
        for el in array {
            let c = CategoryViewModel(id: UUID(), name: el, userEmail: userEmail)
            addCategory(category: c)
        }
    }
}
```

```
class CategoryListViewModel: ObservableObject {
    @Published var categories = [CategoryViewModel]()

    func fetchAllCategory(userEmail: String) {
        self.categories = ManageData.shared.getAllUserCategory(userEmail:
            userEmail).map(CategoryViewModel.init)
    }

    func removeCategory(at index: Int) {
        let bday = categories[index]
        ManageData.shared.removeCategory(id: bday.id)
    }
}
```

Додаток К. Робота з Firebase:

c) реєстрація в системі

```
func register(){
    if self.email != ""{
        if self.pass == self.repass{
                    Auth.auth().createUser(withEmail: self.email, password: self.pass) { (res,
                        err) in
                        if err != nil{
                            self.error = err!.localizedDescription
                            self.alert.toggle()
                            return
                        }
                        category.addDefaultCategoryForUser(userEmail: self.email)

                        UserDefaults.standard.set(true, forKey: "status")
                        NotificationCenter.default.post(name: NSNotification.Name("status"),
                            object: nil)
                    }
        }

        else{
            self.error = "Password mismatch"
            self.alert.toggle()
        }
    }
    else{
        self.error = "Please fill all the contents properly"
        self.alert.toggle()
    }
}
```

d) зміна паролю за допомогою пошти

```
 func reset() {

 if self.email != ""{
     Auth.auth().sendPasswordReset(withEmail: self.email) { (err) in
         if err != nil{
             self.error = err!.localizedDescription
             self.alert.toggle()
             return
         }
         self.error = "RESET"
         self.alert.toggle()
     }
 }
 else{

     self.error = "Email Id is empty"
     self.alert.toggle()
 }
}
```