ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА

(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій

(повне найменування факультету)

Кафедра інформаційних технологій

(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «Бакалавр»

«Розробка додатку для моніторингу актуальної інформації ринку валют»

(тема кваліфікаційної роботи українською мовою)

«Development of an Application for Monitoring Current Currency Market Information»

(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач денної форми навчання

спеціальності 122 Комп'ютерні науки

(код, назва спеціальності)

Освітня програма Комп'ютерні науки

(назва)

Сенкевич Андрій Вікторович

(прізвище, ім'я, по-батькові здобувача)

Керівник к.т.н., доцент Гнатовська Г.А.

(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент к.т.н., доцент Волощук Л.А.

(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
Інформаційних технологій .
№ 1 від 09 червня 2024 р.

Завідувачка кафедри

(підпис) КАЗАКОВА Надія .

(прізвище,ім'я)

Захищено на засіданні ЕК № 13 .
протокол № 25 від 21 червня 2024 р.

Оцінка добре / С / 80 .

(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК

(підпис) КОПИЧЕНКО Іван .

(прізвище,ім'я)

Одеса 2024

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ТЕРМІНІВ І ПОЗНАЧЕНЬ

БД – база даних

СУБД – система управління базами даних

CSS – Cascading Style Sheets

CGI – Common Gateway Interface

HTML – HyperText Markup Language

ODBC – Open Database Connectivity

PHP – Hypertext Preprocessor

SQL – Structured Query Language

ВСТУП

Розробка та впровадження інформаційних систем стає все більш важливою для установ та організацій, які потребують отримувати актуальну інформацію у реальному часі. Сучасний стан розвитку інформаційних технологій, коли актуальність та швидкість доступу до інформації набувають першочергового значення, є актуальним застосування додатків, які відображають інформацію на будь-якому мобільному пристрої. У світі сьогодення, де потрібна швидка реакція на зміни, особливо в фінансовій сфері, моніторинг актуальної інформації ринку валют повинен забезпечувати відображення не лише поточних ставок, але й забезпечує можливість відстежувати їх у режимі реального часу. Це особливо цінно для тих, хто здійснює значні фінансові операції або просто бажає бути в курсі ринкової ситуації.

Зараз існує велике різноманіття мов програмування та технологій для розробки додатків, що пов’язано з великою кількістю різних мобільних пристроїв, які працюють на різних операційних системах. Процес розробки додатків для мобільних пристроїв вимагає врахування певних обмежень, пов'язаних з особливостями їхньої роботи в мобільних операційних системах. Вибір мови програмування часто залежить від цільової платформи.

Для отримання більш глибокого розуміння ринкових процесів необхідно аналізувати не лише поточні показники, але й їх динаміку з часом. Це може бути надзвичайно корисним для трейдерів, економістів та всіх, хто планує довгострокові інвестиції або займається міжнародним бізнесом. У сучасному світі розуміння взаємозв'язків між різними валютами є ключовим для успішного бізнесу та інвестицій. Порівняння різних валют дозволяє не лише оцінити актуальний курс однієї валюти відносно іншої, а й глибше зрозуміти їх взаємодію та вплив глобальних економічних подій на їхню вартість.

Розробка додатку для моніторингу актуальної інформації ринку валют є актуальним та доцільним завданням, який забезпечить оновлення важливої фінансової інформації у реальному часі, що гарантує отримання найсвіжішої

інформації. Такий додаток повинен мати інтуїтивно зрозумілий інтерфейс, що дозволить зручно та швидко знаходити актуальну фінансову інформацію з достовірних перевірених джерел, а застосування мобільних технологій розробки забезпечить користувачам доступність додатка з будь-якого мобільного пристрою у будь-якому місці.

Метою створення додатку для моніторингу актуальної інформації ринку валют – є відстеження фінансової інформації на валютних ринках та забезпечення швидкого доступу до даних для зручного перегляду курсів валют, фінансових новин та іншої фінансової інформації. Такий додаток адресований користувачам з будь-яких сфер діяльності, які потребують оперативної інформації про фінансовий ринок країни. Крім того, вбудований конвертер валют у додатку дозволить зручно перераховувати грошові суми з гривні в інші валюти. Застосування сучасних технологій розробки мобільних додатків має значні переваги і гарну перспективу розвитку у майбутньому. Додатки стають набагато могутніше по продуктивності і менш енерговитраті, завдяки удосконаленню самих мобільних пристроїв і отриманню нових методів та технологій розробки.

Використання додатку для моніторингу актуальної фінансової інформації ринку валют надасть зацікавленим користувачам доступ до своєчасної фінансової інформації шляхом здійснення моніторингу новин та актуальних курсів валют з фінансового порталу МінФін, а також дозволить проводити конвертацію різних видів валют.

Дана кваліфікаційна робота бакалавра містіть 70 сторінок, 5 таблиць, 19 рисунок та 14 посилань.

1 ВИЗНАЧЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО РОЗРОБКИ ДОДАТКУ

Для здійснення розробки та програмної реалізації додатку для моніторингу актуальної фінансової інформації ринку валют необхідним є визначення особливостей функціонування інформаційних систем надання та моніторингу фінансової інформації. Також доцільно провести огляд та аналіз існуючих систем для подальшого визначення недоліків та переваг застосування всіх необхідних функціональних можливостей аналогічних систем.

1.1 Поняття фінансової інформації

Фінансова інформація відображає економічний стан конкретного об'єкта та його економічні параметри. Опис об'єкта необхідний для проведення фінансових операцій з ним та зміни його економічних характеристик, що відображає значення самої фінансової інформації.

Головна мета отримання фінансової інформації полягає в можливості впливу на економічний стан об'єкта. Для того, щоб бути корисною, інформація повинна бути актуальною, тобто інформація повинна вчасно надходити, що забезпечить прийняття на її основі вірних рішень. Одна і та ж інформація може бути відповідною для одного конкретного вирішення, а для іншого – ні, так само як і може бути відповідною для однієї групи користувачів, а для іншої – ні. Відповідність визначається користувачем і прийнятим рішенням.

Останнім часом в Україні курс гривні до інших валют не є стабільним, і банківський ринок швидко змінюється. Люди, які займаються інвестуванням або фінансовою аналітикою, повинні бути в курсі змін на ринку та швидко реагувати на них. З огляду на те, що в країні діє багато малих підприємств, які здійснюють торговельні операції та ведуть розрахунки у валюті або гривні, важливим є відстеження швидкої зміни курсів валют та актуальної фінансової інформації. Крім того, зручним і доцільним є використання курсового

конвертера-калькулятора, який проводить конвертацію валюти прямо в мобільному додатку.

1.2 Особливості розробки додатку

Ключовим компонентом у системі моніторингу потоками фінансової інформації є технологічна інфраструктура, яка забезпечує збір, обробку і розподіл інформації. Особливу вагу має взаємодія між аналітичними підрозділами та службами моніторингу, аналізу та автоматизації. Система збору та обробки даних повинна враховувати як стратегічні, так і оперативно-тактичні потреби підрозділів моніторингу та аналізу. Важливо досягти балансу між забезпеченням стратегічних інтересів і підтримкою оперативних та актуальних потреб функціональних підрозділів.

У контексті нестабільності глобальних фінансових ринків, доступ до актуальної інформації про курси валют стає надзвичайно затребуваним. Валютний курс є результатом взаємодії традиційних факторів попиту і пропозиції, які в свою чергу піддаються впливу різноманітних факторів. Ця складність призводить до того, що реальна вартість валюти (яка визначається тим, скільки товарів і послуг можна купити за неї) та її курс (який формується балансом попиту і пропозиції на ринку) можуть значно розходитися. На котирування валют впливають економічні та політичні фактори. Основні з них - рішення центральних банків держав та міжнародних організацій щодо кредитно-грошової політики відносно валюти, яку вони випускають. Хоча управління курсом зазвичай не є прямою метою центральних банків, зміна кількості грошей в обігу - це постійний інструмент впливу, який використовують ці банки і який може призвести до змін у валютному курсі [1].

Останнім часом в Україні курс гривні до іноземних валют змінюється досить часто (іноді навіть кожні 5 хвилин). У такій обстановці підприємства, що займаються імпортом або експортом, ризикують втрачати значні суми, не встигаючи вчасно реагувати на зміни на ринку, а керівництво таких підпри-

ємств повинно бути оперативно та вчасно проінформовані про такі зміни, застосовуючи зручні засоби, якими може бути сучасний мобільний додаток.

Аналізуючи існуючи варіанти видів додатків для відображення фінансової інформації, доцільним було використання мобільного додатку, який включає в себе зручний віджет та додатковий додаток на смарт-годиннику.

Після аналізу актуальності інформації та наявності API на фінансових порталах, було вирішено використати інформаційний ресурс МінФін [2]. Вибір саме цього ресурсу обґрунтовано наявністю наступних переваг:

- наявність API з основними курсами валют (долар США, євро, румунські леї і т.д.), який дозволяє отримувати інформацію з Нацбанку, Міжбанківських торгів, комерційного ринку, та середнього курсу банків України;
- наявність фінансових новин і прогнозів експертів щодо валютного ринку, які постійно оновлюються;
- відображення рейтингу та контактної інформації банків щодо всіх акредитованих в Національному банку України.
- стабільна робота та надійність платформи.

Щодо недоліків, можна відзначити лише обмеження API, яке передбачає, що кожен запит повинен надходити з клієнта не частіше, ніж раз у 5 хвилин. Також є актуальним завдання відображення курсу криптовалюти у додатку для моніторингу актуальної фінансової інформації, у зв’язку зі стрімким зростанням за останні роки ринку криптовалюти не лише в Україні, але й у всьому світі. Криптовалюта є цифровою (віртуальною) валютою, одиницею якої є монета, захищена від підробки. Основна особливість криптовалюти полягає в тому, що вона представлена зашифрованою інформацією, яку неможливо скопіювати. Криптовалюта створюється безпосередньо в мережі і не пов'язана з будь-якою валютою чи державною валютною системою. Це відрізняє її від звичайних грошей, які спочатку мають бути внесені на рахунок у грошовому еквіваленті, наприклад, через банк або платіжний термінал, перш ніж стати доступними у електронному вигляді [3]. Одним з ключових

аспектів було відображення курсів популярних криптовалют. Лідером на цьому ринку є біржа kuna.io, перевагою якої є ідентичність API з порталом Мінфін, що дозволяє мати однакову модель даних, отриману з різних джерел.

1.3 Огляд та аналіз існуючих додатків

Звісно, додатки для відстеження актуальної фінансової інформації та курсів валют мають важливе значення в сучасному фінансовому світі. Вони забезпечують миттєвий доступ до актуальних даних та надають користувачам зручні та прості у використанні інтерфейси.

Огляд та аналіз існуючих додатків надасть можливість визначити необхідні функції, які повинен мати додаток для моніторингу актуальної фінансової інформації.

Першим було розглянуто найпопулярніший додаток XE Currency Converter (рис. 1.1).

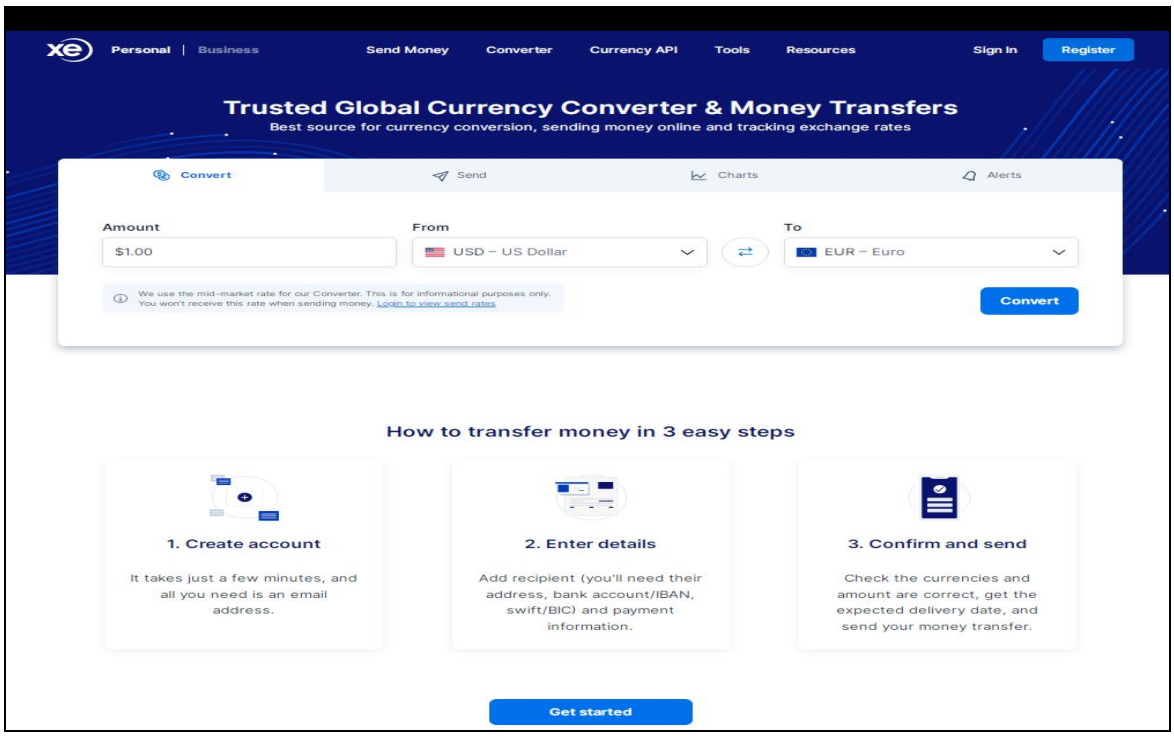

Рисунок 1.1 – Головна сторінка додатку XE Currency Converter

Цей додаток дозволяє перевіряти актуальні курси валют та конвертувати гроші між різними валютами. Він має простий інтерфейс і широкий спектр підтримуваних валют.

XE Currency надає користувачам актуальні котирування, графіки та конвертери для більш ніж 180 валют світу, а також забезпечує зручним механізмом надання графіків та аналітичних даних для подальшого прийняття оптимального фінансового рішення. XE Currency дозволяє отримувати миттєві оновлення курсів валют, використовуючи власний API.

Але серед недоліків користування додатком XE Currency можливо визнати, що цей додаток є платним ПЗ, всі API дозволено використовувати обравши необхідний платний пакет послуг. Користувачам дозволено ознайомлення з функціями додатку через взяття «пробного періоду» використання.

Наступним було розглянуто додаток Revolut (рис. 1.2). Revolut – це відома інноваційна платформа, яка спеціалізується на фінансових технологіях з 2015 року.

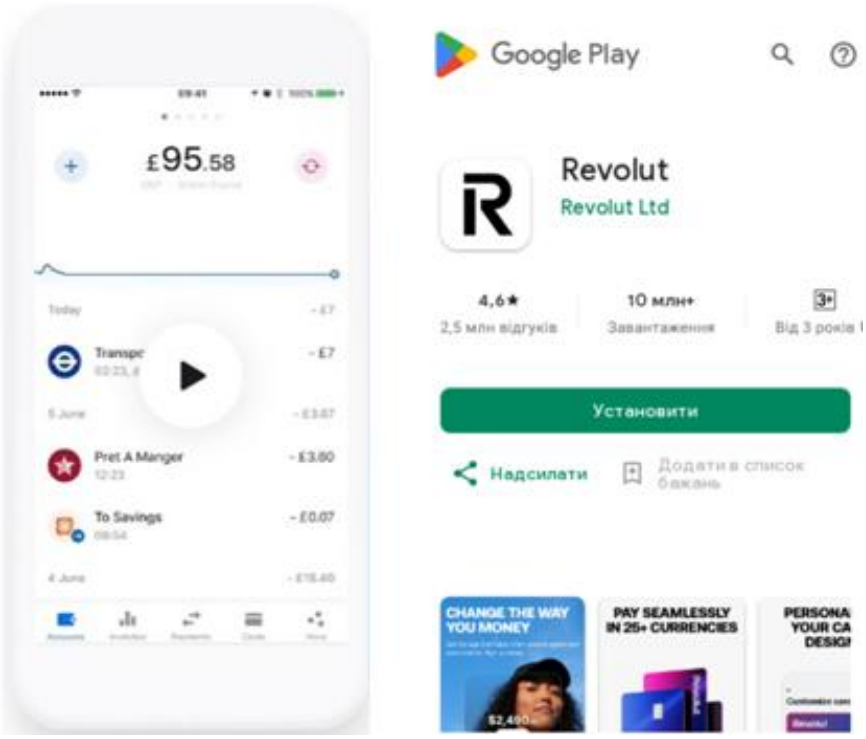

Рисунок 1.2 – Сторінка додатку Revolut

Цей різнобічний фінтех-проєкт був започаткований у Великобританії. Цей додаток не лише надає інформацію про курси валют, але також служить інструментом для міжнародних банківських операцій, що дозволяє здійснювати транзакції в різних валютах. Серед основних переваг можливо зазначити наступні:

- Revolut пропонує безкоштовні міжнародні транзакції за певних умов, що робить його привабливим для тих, хто часто взаємодіє з різними валютами;
- користувачі можуть обмінювати валюти без обмежень за реальним обмінним курсом;
- Revolut інформує користувачів про зміни курсів валют, що дозволяє оперативно реагувати на ринкові тенденції.

Безумовно, Revolut є одним з найкращих віртуальних фінансових сервісів, але він все ж має деякі недоліки. Перш за все, це платний додаток, який має обмежену функціональність у найдешевшому тарифі «Standard». Це повноцінний фінансовий додаток, що пропонує послуги обміну валюти, переказу грошей, інвестицій, страхування, аналітику витрат, що робить його надзвичайно функціональним, але не дозволяє використовувати обмежену кількість функцій налаштувавши під потреби конкретного користувача.

Ще одним з відомих додатків, який забезпечує відстеження актуальних курсів валют та надає конвертер з підтримкою понад 180 валют є додаток Easy Currency Converter. Він надає миттєвий доступ до курсів валют та простого конвертера для швидкого розрахунку та дозволяє користувачам налаштувати додаток відповідно до своїх вподобань, обравши валюти, які їх цікавлять, і встановивши автоматичні оновлення. Додаток дозволяє налаштувати свій особистий список валют, крім того надає актуальну інформацію щодо деяких металів та криптовалюти (Bitcoin, Ethereum, Litecoin). Додаток оновлює курси валют кожну секунду, що не завжди є зручним та актуальним і іноді заважає роботі та перевантажує додаток. Для конвертації валют у режимі офлайн додаток Easy Currency Converter використовує дані з останнього

з'єднання, що зобов’язує користувача бути уважним та контролювати час останнього з’єднання і оновлення інформації у додатку (рис. 1.3).

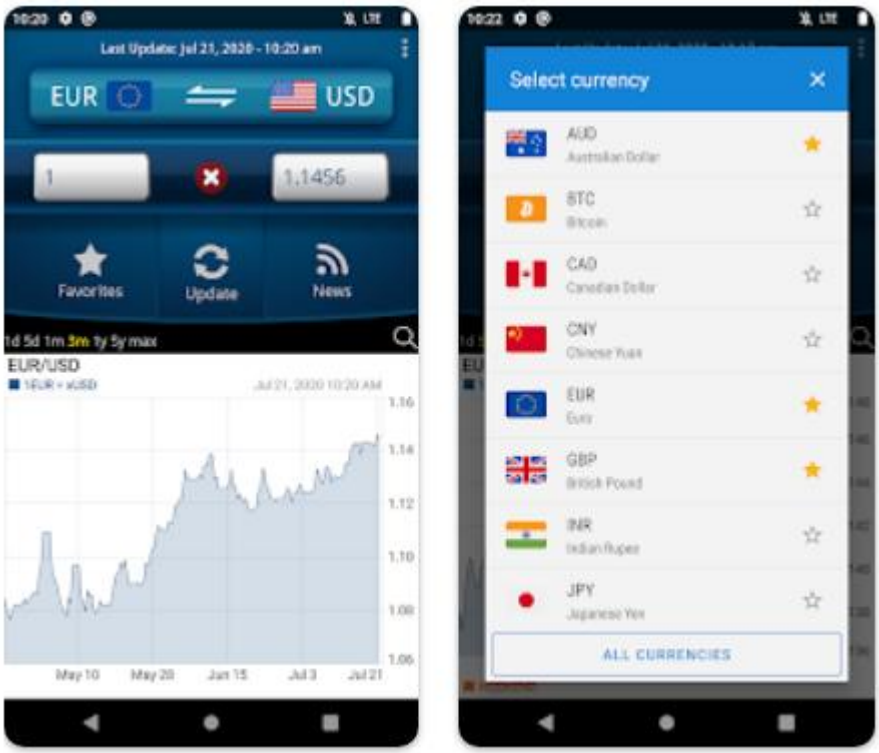

Рисунок 1.3 – Вигляд мобільного додатку Easy Currency Converter

Таким чином, огляд та аналіз існуючих додатків для моніторингу актуальної фінансової інформації ринку валют дозволив стверджувати, що вибір програми для відстеження курсів валют залежить від потреб і вподобань конкретних користувачів. Сучасний додаток потрібен обов’язково забезпечувати надійні дані, зручний інтерфейс та додаткові функції для детального моніторингу валютних ринків. Саме такий додаток надасть користувачу впевненість у керуванні своїми фінансами та приймати обдумані рішення.

Здійснивши дослідження та аналіз існуючих функцій у додатках для моніторингу фінансової інформації валютних ринків, можливо відзначити наступні необхідні функції, що повинен надавати мобільний додаток, що розробляється.

Метою розробки мобільного додатку для відстеження актуальної фінансової інформації на валютних ринках є створення зручного інструменту для користувачів з наступними можливостями:

- відстеження курсів валют. Забезпечення можливості швидкого отримання своєчасної інформацію про актуальні курси валют на основних ринках, що дозволить відстежувати поточні тенденції та приймати користувачам обґрунтовані фінансові рішення;
- зручний та швидкий конвертер валют. Забезпечення зручного та швидкого механізму конвертації різних валют, що надасть користувачам можливість ефективного управління фінансами в умовах змінливого ринкового середовища;
- актуальні новини фінансових ринків. Забезпечення користувачів своєчасним доступом до актуальних фінансових новин, що надасть можливість аналізувати поточний стан на фінансових ринках і приймати обґрунтовані рішення;

Практична цінність розробки такого мобільного додатку полягає в забезпеченні зацікавлених користувачів доступом до актуальної фінансової інформації та інструментами для ефективного управління своїми фінансами.

1.4 Постановка завдання

При виконанні кваліфікаційної роботи бакалавра необхідно здійснити проектування та розробити мобільній додаток, якій надає користувачам актуальну інформацію валютного ринка та найважливіші фінансові новини в Україні. Інформація буде завантажуватися з фінансового порталу МінФін за допомогою публічного API та технології парсингу HTML-сторінок.

Основним призначенням та метою розробки додатку для моніторингу актуальної фінансової інформації ринку валют є забезпечення зручного додатку для відстеження наступної фінансової інформації:

- курсів валют,

– швидкого механізму конвертації валют,
– актуальних фінансових новин,
– відстеження курсів валют в банках.

Для досягнення поставленої мети, в рамках виконання кваліфікаційної роботи бакалавра необхідно вирішити наступні завдання:

– визначити основні функціональні можливості додатку для моніторингу актуальної фінансової інформації ринку валют;
– здійснити огляд та аналіз існуючих додатків;
– здійснити постановку завдання з зазначенням всіх функцій додатку;
– визначити архітектуру, технологій та засоби розробки додатку;
– здійснити проектування додатку;
– здійснити програмну реалізацію та провести тестування додатку;

На підставі проведеного дослідження предметної області визначені основні розділи, яки повинні бути доступні користувачу додатку для моніторингу актуальної фінансової інформації ринку валют. Основні розділи додатку:

– інформер курсів валют;
– конвертор валют;
– фінансові новині;
– курси в банках.

Мобільний додаток для моніторингу актуальної фінансової інформації ринку валют повинен забезпечувати оперативний доступ та достовірність інформації для всіх учасників інформаційного обміну в системі.

2 ВИБІР ТЕХНОЛОГІЙ ТА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ ДОДАТКА

Для здійснення проектування та програмної реалізації додатку для моніторингу актуальної фінансової інформації ринку валют необхідним є здійснення огляду та вибору архітектури, технологій та засобів розробки додатку.

2.1 Вибір архітектурного рішення

Шаблони проектування роблять код додатку більш модульним і легким, коли справа доходить до виправлення помилок і змін. Для вибору архітектурного рішення для додатка моніторингу актуальної фінансової інформації ринку валют проаналізовано найпопулярніші шаблони проектування MVC (Model – View – Controller) і MVVM (Model – View – View – Model).

Хоча шаблони проектування (також відомі як архітектурні шаблони) є ключовими для розробки масштабованих додатків IOS, але існує багато суперечок про те, який архітектурний шаблон найкраще підходить для використання в додатку.

Розглянемо шаблон MVC (модель-вид-контролер). MVC це найбільш часто використовуваний шаблон в розробці IOS. Цей шаблон проектування ефективно організовує код за трьома категоріями (або шарами): Модель, Вид (або представлення) та Контролер. За допомогою цього шаблону можна вносити зміни в один шар, не зачіпаючи інші шари шаблону. Наприклад, якщо потрібно змінити базу даних, потрібно просто видалити модель і замін її, не редагуючи представлення або контролер. Або, якщо треба змінити зовнішній вигляд представлення, розробнику не потрібно турбуватися про плутанину в коді бази даних [4].

Таким чином реалізується «поділ завдань», і це полегшує налагодження, підтримку і повторне використання раніше написаного коду. Крім того, це шаблон дизайну, який рекомендується Apple, і це є норма в співтоваристві розробників iOS.

Шаблон проектування MVC розділяє кожну частину коду на одну з трьох частин: модель, вид і контролер (рис. 2.1).

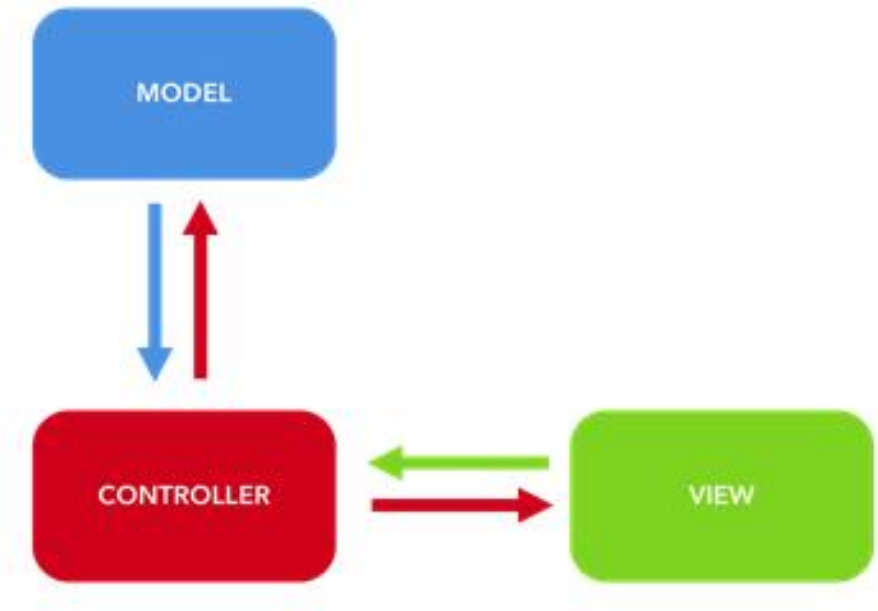

Рисунок 2.1 – Структура архітектурного шаблону MVC

Модель (англ. Model): цей шар відповідає тільки за дані і їх форму в тому вигляді, в якому вони відображаються в додатку. Шар Controller може вносити зміни в дані програми, повідомляючи модель. Модель не несе відповідальності ні за що, в тому числі за те, як дані показуються користувачеві.

Вид (англ. View): відповідає за подання даних – тільки за те, як користувач бачить і взаємодіє з даними. Це може включати в себе повторно використовувані осередки, таблиці та інші елементи призначеного для користувача інтерфейсу, які нічого не знають про дані або про те, як вони обробляються. Дані передаються в елементи уявлення контролером.

Контролер (англ. Controller): приносить дані з моделі, а потім передає їх в представлення для відображення користувачеві, відповідає за прослуховування введення і зміну моделі в міру необхідності [4].

Оскільки MVC не має суворої реалізації, то варіантів реалізації може бути безліч. Немає загальноприйнятого визначення, де повинна розташовуватися бізнес-логіка. Вона може знаходитися як в контролері, так і в моделі. В останньому випадку, модель буде містити всі бізнес-об'єкти з усіма даними і функціями. Деякі фреймворки жорстко задають де повинна розташовуватися бізнес-логіка, інші не мають таких правил. Також не вказано, де повинна знаходитися перевірка введених користувачем даних. Проста валідація може зустрічатися навіть у Виді, але частіше вони зустрічаються в виді Моделі. Інтернаціоналізація і форматування даних також не має чітких вказівок по розташуванню.

Серед iOS розробників часто називають MVC як Massive View Controller. Сильна зв'язаність між відображенням і контролером тягне за собою проблеми розширюваності, перевикористовування і велику кількість коду в класах контролерів. Контролер набуває відповідальності які йому не властиві. Єдиним виходом у такій ситуації для розробника є рознесення логіки, необхідно полегшити контролери, виділяти окремі класи для отримання і маніпуляції даними. Також необхідно дотримуватися одному з основоположних принципів ООП – оперувати інтерфейсами, а не реалізаціями [5].

Наступним було розглянуто шаблон MVVM (Модель-Вид-Вид-Модель). Архітектурний патерн MVVM, запозичений розробниками IOS з Microsoft, може допомогти у вирішенні проблеми з масивними контролерами уявлення. Хоча цей патерн не так поширений в розробці для iOS, як MVC, але він все частіше використовується для усунення недоліків MVC.

Використовується для поділу моделі і її представлення, що необхідно для їх зміни окремо один від одного. Наприклад, розробник задає логіку роботи з даними, а дизайнер працює з призначеним для користувача інтерфейсом. У MVVM View і Controller об'єднується в один шар (рис. 2.2).

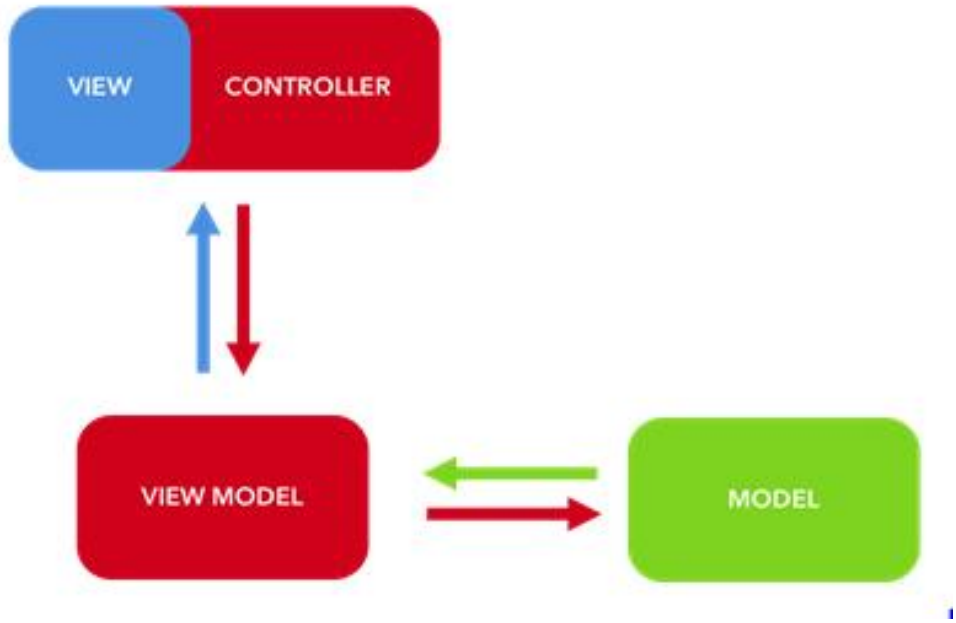

Рисунок 2.2 – Структура архітектурного шаблону MVVM

(Вид) Контролер (англ. ViewController): цей шар, зазвичай відомий як Вид, тісно пов'язаний з Контролером та настільки, що вони навіть не розділені на різні верстви. Вид зв'язується тільки з Контролером, але Контролер зв'язується з ViewModel та View. View і Controller виконують ті ж завдання, що і в MVC, але відмінність полягає в тому, що деякі завдання, які були передані Controller (в MVC), тепер обробляються проміжним рівнем, ViewModel, для запобігання неправильного використання Controller. Вид як і раніше обробляє відображення даних для користувача, а контролер відповідає на призначені для користувача події та пов'язується з іншими верствами шаблону [5].

Модель Вигляду (англ. ViewModel): цей шар насправді не містить будь-яких «уявлень» сам по собі, але замість цього обробляє логіку відображення представлення – зазвичай звану логікою представлення. Це включає в себе створення налаштованих форматуючих і оброблюваних рядків для відображення, а також фактичне скорочення чисел, які будуть показані користувачеві на основі даних на рівні моделі. Тобто, вона містить Модель, перетворену до Виду, а також команди, якими може користуватися Вид, щоб вплива-

ти на Модель. Модель (англ. Model): не дуже відрізняється від шару моделі в патерні MVC. Модель обробляє дані і вносить зміни в ці дані тільки в тому випадку, якщо вона отримує оновлення від шару ViewModel. Не правильно змішувати обов'язки будь-якого з цих рівнів, так як це може призвести до ускладнення коду програми, що робить використання будь-якого шаблону проектування надмірною [4].

Підсумком застосування патерну MVVM є функціональний розподіл програми на три компонента, які простіше розробляти і тестувати, а також в подальшому модифікувати і підтримувати.

Для зв'язку між View і ViewModel використовуються реактивні прив'язки (Reactive Bindings) які є основною ідеєю архітектури MVVM, де розробники можуть працювати з кодом ViewModel, а дизайнери можуть працювати з View в Interface Designer. Існує достатня кількість фреймворків для реалізації реактивних прив'язок таких як ReactiveCocoa, RxSwift, Bond та інші. Так само можна реалізовувати зв'язок використовуючи метод делегування. Реактивний фреймворк Combine, за допомогою якого можна використовувати реактивні прив'язки спільно з новим фреймворком для верстки у декларативному стилі SwiftUI буде сприяти використанню даного архітектурного шаблону багатьма розробниками.

MVVM – це дійсно хороший і простий спосіб передачі логіки подання іншого об'єкта, який допомагає розробникам уникнути великих View-Controller, спростити управління і охопити код юніт-тестами. Саме так розробник і отримує просту і тестовану архітектуру.

В якості архітектурного шаблону для створення додатку для моніторингу актуальної фінансової інформації валютних ринків було обрано архітектуру MVVM.

Наступним кроком при виконанні розробки додатку для моніторингу актуальної фінансової інформації валютних ринків було визначення середовища, яка надає розробнику засоби для створення додатків для iPhone, iPad, Mac, Apple Watch и Apple TV.

2.2 Огляд середовищ розробки IOS додатків

Найбільш популярні середовища нативної розробки IOS додатків, або IDE (Integrated Development Environment) – це Xcode від Apple та AppCode від JetBrains. Швидкий редактор, укомплектований повним набором інструментів для розробки під iOS, macOS та ін. Надаються розробникам безкоштовно. Оновлення можна безкоштовно скачати на офіційному сайті для розробників додатків або у магазині AppStore. Xcode IDE надає все, що потрібно для розробки: від професійних редакторів з функцією автозаповнення коду і Cocoa рефакторінга, до налаштування open-source компіляторів [6].

IB (Interface Builder) – це додаток з набором інструментів для розробки графічних інтерфейсів, інтегрований в Xcode. Усю верстку можна зробити в IB, а потім зв'язати візуальні елементи з файлом реалізації, в якому описана вся логіка взаємодії з ними. Перевагами застосування Interface Builder є наочна верстка, настроювання стилів, бекграунду, шрифтів і т. д., а мінусом є робота з анімацією, її можна виконати тільки кодом. Використання Interface Builder зберігає величезну кількість часу, що припадає на створення UI частини додатку. Interface Builder усуває написання коду, необхідного для створення, налаштування і позиціонування об'єктів, які використовуються для розробки графічного інтерфейсу.

Додаток Interface Builder з набором інструментів для розробки графічних інтерфейсів, має наступну низку переваг [6]:

- наявність IB;
- стимулятор; надає повний набір iOS-пристроїв, на яких можна запустити і протестувати свій додаток;
- відгадчик; вміє розбирати візуальну частину по-єлементно для пошуку помилок в верстці, також допоможе відловити баг і розібратися з проблемою витоку пам'яті. На додаток до всього відладчик може виробляти всі ці операції і UI-тести в «бездротовому» режимі;

- вбудована система контролю версій; крім стандартних функцій розгалуження, в режимі розділеного екрану дозволяє переглядати зміни в різних «гілках»;
- функція імітації геолокації; незамінна при роботі з картою, має «зашитий» набір міст, список яких можна доповнити;
- містить Swift Playground, так звану «пісочницю», яка дозволяє швидко перевірити новий алгоритм або графічну рутину (наприклад, кілька рядків коду), не створюючи цілий додаток;
- може збирати додаток відразу на iOS-пристрій;
- впроваджена інтелектуальна компіляція, яка прискорює час компіляції одного файлу;
- має просунуту програму перегляду і пошуку документації Apple;
- вбудовані інструменти для аналізу поведінки і продуктивності.

До недоліків Interface Builder можливо віднести: нестабільний – в процесі роботи може мимоволі закриватися з помилкою; найчастіше не працює функція доповнення тексту по введеній частині.

Наступним додатком, який було розглянуто став AppCode. Як і Xcode, містить повний набір необхідних засобів для ефективної та зручної роботи з мовою Objective-C, Swift, C ++ і на 100% сумісна з Xcode. Перша версія публічного превью AppCode стала доступна в квітні 2011 року. AppCode побудований на платформі IntelliJ IDEA, який написаний на Java. Можна розширити можливості IDE за рахунок установки плагінів, створених для IntelliJ Platform, а також є можлівисть написати свої власні плагіни. Серед переваг використання AppCode можливо визначити наступні [7]:

- стабільний; раптових закриттів, як в Xcode, немає;
- автодоповнення працює стабільніше і швидше, ніж в Xcode;
- інтеграція з баг/issue-трекерами, такими як JIRA, YouTrack, Lighthouse, Pivotal Tracker, GitHub, Redmine і т. д.;
- більш докладний опис помилок і попереджень;
- може збирати додаток відразу на iOS-пристрій, як і Xcode.

Недоліками застосування Xcode є:

- для збірки додатка все одно потрібен Xcode;
- немає Intreface Builder; відкриття IB-файлу з AppCode відбувається в Xcode;
- немає візуального відладчика, а також немає бездротового налагодження;
- запуск програми відбувається на Xcode-симуляторі;
- AppCode – платний продукт, для безкоштовного ознайомлення дається 30 днів;
- оновлення мови приходить з деякою затримкою. У Xcode доступні свіжі версії ще на етапі бета-тестування.

Іншими словами, для використання середовища розробки AppCode в якості основного IDE, для повноцінної розробки все одно потрібен Xcode.

Проведений аналіз переваг та недоліків застосування середовищ розробки для розробки додатку для моніторингу актуальної фінансової інформації валютних ринків дозволяє здійснити вибір середовища розробки на користь застосування Xcode.

Проаналізувавши особливості кожного з середовищ розробки, Xcode перемагає усіх вище представлених учасників. Це середовище гнучке, швидке, потужне і здатне завжди прийти на допомогу. Xcode стає все краще, незважаючи на складні заходи, які вживаються Apple з метою утримання повного контролю над iOS додатками і пристроями. Робочий простір в Xcode тримає розробника зосередженим. Під час написання коду, в реальному часі можна побачити помилки компілятора або проблеми, а також повідомлення з докладним описом корисної інформації. Відладчик працює плавно, а симулятор – швидкий і орієнтований на користувача. Отже, Xcode перевершує всі інші IDE. Тема розробки мобільних застосувань для платформи iOS є досить цікавою і представляє собою широке поле для дослідження в галузі розробки мобільного програмного забезпечення.

2.3 Вибір мови програмування

Для розробки додатку для моніторингу актуальної фінансової інформації ринку валют необхідним э завдання вибору мови програмування, що дозволить реалізувати необхідний функціонал додатку.

Розглянута була мова програмування Objective-C. Objective-C –мова програмування, створена на початку 1980-х років минулого століття шляхом схрещування C (Ci) з популярним в той час Smalltalk (зв'язок з об'єктами через повідомлення). Objective-C спочатку сприймався, як проста надбудова над мовою C, що модифікує його деякі синтаксичні конструкції, але після того, як за ліцензування взялася спочатку компанія Next Step, а потім на правах наступника і Apple, Objective-C став одним з найбільш популярних мов для розробки додатків. Тому багато типів даних в Objective-C успадкували префікс NS (Next Step). Це основна мова, що використовується компанією Apple, знання якої дозволяє писати під будь-які платформи Apple, в тому числі macOS. До переваг використання мови можливо віднести наступні [8]:

- високий ступінь підтримування коду: з кожним оновленням зміни в Objective-C мінімальні;
- велика кількість документації, технічної літератури та величезне співтовариство. Apple надає і регулярно оновлює офіційні книги і ресурси;
- швидкий перехід з однієї з мов сімейства C. Objective-C – це розширення мови C. Це означає, що будь-який код на C є також коректним кодом і для Objective-C, потрібно тільки звикнути до синтаксису;
- сумісність Objective-C всередині проектів, написаних на Swift, дозволяє застосовувати дві мови одночасно.

Але мова програмування Objective-C має і наступні недоліки:

- громіздкий і важко читаємий синтаксис;

- динамічна система типів даних, яка також є плюсом, передбачає можливість появи або пропуску помилок навіть під час компіляції. Зокрема, затягнути процес можуть помилки;
- низька в порівнянні з мовою Swift продуктивність;
- взаємодія з файлами Swift відбувається за допомогою «моста» (умовний адаптер, який переводить код на Swift в формат Objective-C), що сильно уповільнює процес складання.

Наступною була розглянута мова програмування Swift. Молода, могутня і відкрита, SWIFT – мова програмування загального призначення. Офіційно представлена компанією Apple 2 червня 2014 року. Поєднує в собі все краще від C і Objective-C, але позбавлена обмежень останньої, що накладаються на догоду сумісності з C. У Swift використовуються сувора типізація об'єктів, що зменшує кількість помилок ще на етапі написання коду. Також Swift-програми компілюються у машиний код, що дозволяє забезпечити високу швидкодію. В Swift додані сучасні функції, такі як дженерики, замикання, елементі функційного програмування, множинні повернені значення і багато іншого, що перетворюють створення додатка в більш гнучкий і захоплюючий процес [9]. Перевагами використання такої мови є:

- швидкість;
- спрощена навігація по файлах проекту; на відміну від Objective-C, який створює два файли для оголошення і реалізації, Swift обходиться всього одним. Крім того, імена методів і коментарі між файлами синхронізуються автоматично;
- легка читаність і лаконічність, оскільки дана мова не побудована на C. Наприклад, не потрібно ставити крапку з комою в кінці рядка і писати дужки для оточення вираження всередині if/else. Ніяких квадратних дужок, Swift нагадує звичайну англійську мову, є набагато чистішим і має спрощений синтаксис;
- великі можливості в порівнянні з Objective-C. Наприклад, дженерики (універсальні шаблони). Універсальний код дозволяє писати гнучкі,

загального призначення функції та типи, які можуть працювати з будь-якими іншими типами;

– підвищена безпека. Swift, на відміну від Objective-C, строго типізований, тобто при оголошенні іменованих параметрів потрібно явно вказувати тип даних, інакше при виконанні коду компілятор викликає помилку;
– підтримка динамічних бібліотек. Одне із значущих змін в Swift – перехід від статичних бібліотек до динамічних, які по суті є виконуваними шматками коду. Вони приєднуються до додатка і «зв'язуються» з новими версіями мови, що дозволяє програмі працювати стабільно.

Мова Swift постійно розвивається і змінюється. Наприклад, виклик методу може змінитися після оновлення. Благо Apple збудували цей процес таким чином, що код написаний на більш ранніх версіях не буде зламаний. Імплементовано автоматичний редактор який запропонує перехід на більш нову версію і допоможе виконати цей процес через підрядник. Взаємодія з файлами Objective-C відбувається за допомогою «моста», який сильно гальмує процес складання, що можливо віднести до незначних недоліків використання мови.

Проведений огляд основних можливостей мов програмування для здійснення розробки додатку для моніторингу актуальної фінансової інформації ринку валют, дозволяє обрати в якості мови програмування Swift, що забезпечить реалізацію всіх функціональних можливостей додатку.

Головними критеріями у виборі інструментів розробки для мобільного додатку стають простота у використанні та швидкість. Тому для розробки додатку для моніторингу актуальної фінансової інформації ринку валют будуть використовуватися сучасні продукти від компанії Apple – середовище розробки Xcode і мова програмування Swift. Адже вони увібрали найкращі особливості та переваги універсальних інструментів для розробки мобільного програмного забезпечення.

Найактуальнішою мовою розробки додатків для платформи iOS є Swift. Причина не тільки в її високої швидкості роботи та лаконічному синтаксисі і інтерактивності, але і в тому, що ця мова стає стандартом розробки де-факто і отримує все більше підтримки з боку розробників Apple. Це не скасовує необхідності знайомитися з Objective-C: на цій мові створені корисні бібліотеки, і на багатьох підприємствах потрібна підтримка старих проектів. Актуальність теми підкреслюється широким спектром можливостей для втілення ідей у вигляді мобільного додатку.

2.4 Визначення основних об’єктів і циклу життя IOS додатку

Розуміння того, як влаштован і працює додаток дуже важливо для більш осмисленого програмування. Додатки представляють дуже складну взаємодію між кодом і системними фреймворками. Системні фреймворки надають базову інфраструктуру, з якої всі програми повинні працювати, а розробник пише код для налаштування цієї інфраструктури.

Вхідною точкою входу в кожному додатку заснованому на C є функція main. В iOS додатках це правило так само дотримується. Єдине чим відрізняється, це те, що в iOS додатку розробник не повинен писати в функцію main самостійно. Xcode створює цю функцію як частину основи для проекту [10]. Правда, в мові Swift такої функції вже не спостерігається.

Робота передається під управління фреймворка UIKit. Функція UIApplicationMain передає процес створенням об'єкта ядра додатка, завантаженням призначеного для користувача інтерфейсу програми з файлів storyboard, викликом коду, що дає розробникам шанс налаштувати щось під час запуску, і вкладення додатка в цикл. Єдине, що необхідно надати, це файли storyboard і ініціалізаційний код. Під час запуску, функція UIApplicationMain вибирає кілька ключових об'єктів і запускає додаток. У серці кожного iOS додатка лежить об'єкт UIApplication, який сприяє взаємодії між системою та іншими об'єктами додатка. Як видно на рисунку 2.3, iOS

додаток використовує MVC архітектуру. Цей патерн відокремлює дані додатку від бізнес-логіки і візуальним представленням даних. Ця архітектура ключова в створенні програми, яка здатна запускатися на різних пристроях з різними розмірами екрану. На рис. 2.3 показані об'єкти для більшості додатків.

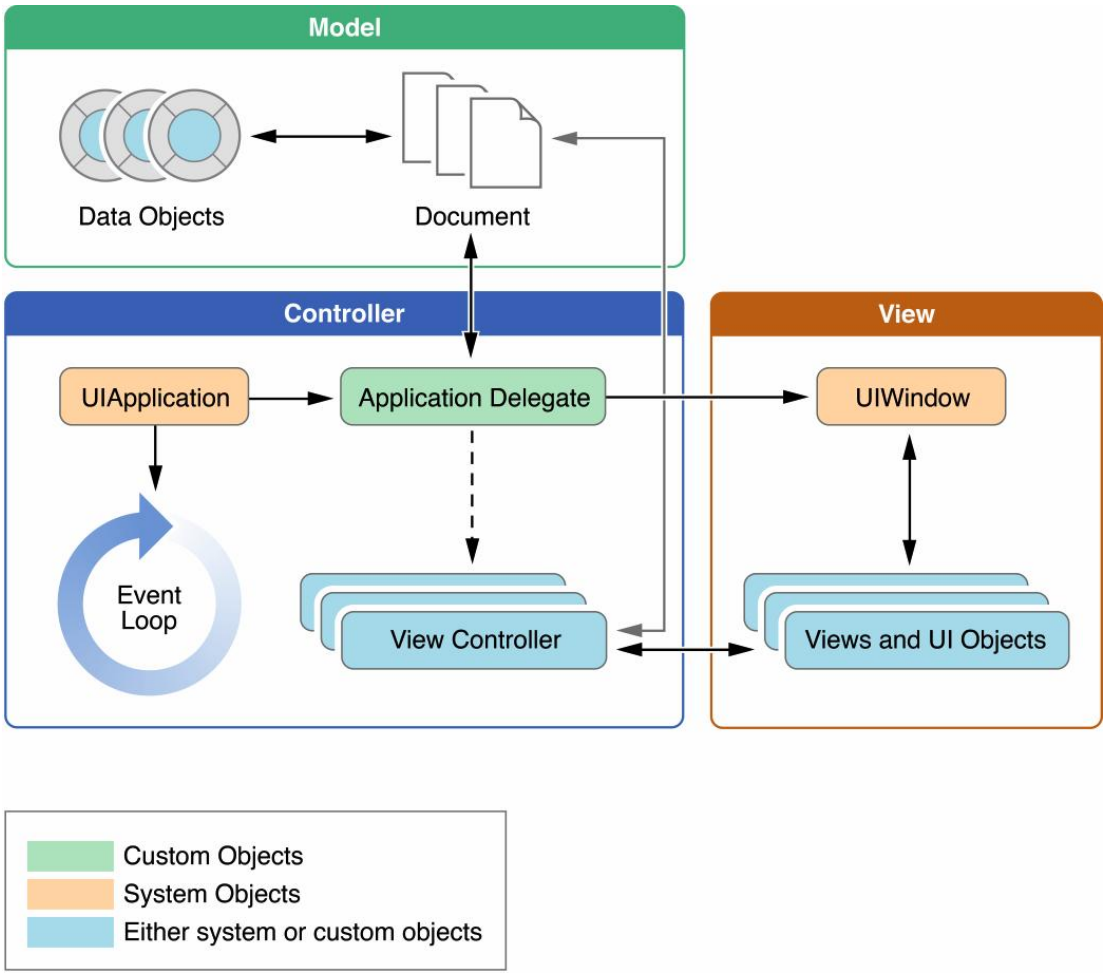

Рисунок 2.3 –Визначення об’єктів циклу життя IOS додатку

Зазначимо ролі, які виконує кожен об'єкт:

– об'єкт UIApplication управляє циклом обробки подій і різними типами поведінки додатків високого рівня. Він також повідомляє про ключові переходи додатку і деяких спеціальних подіях (наприклад, що входять повідомлення Push) своєму делегату, який представляє собою

користувальницький об'єкт. Використовуйте UIApplication «як є», без підкласів;

- об'єкт UIDelegate є серцем коду. Цей об'єкт працює в зв'язці з об'єктом UIApplication і служить для обробки ініціалізації програми, переходу між станами, а так само забезпечує безліч подій в додатках високого рівня;
- Documents і Data Model Objects. Модель даних може бути специфічною для додатка. Додатки можуть також використовувати об'єкти Documents (підкласи UIDocuments) для управління моделями даних. Об'єкти Documents необов'язкові, але пропонують зручний спосіб групувати дані в пакетах файлів;
- об'єкт View Controller служить для управління відображенням контенту додатка на екрані. Клас UIViewController є базовим класом для всіх об'єктів View Controller. Він надає функціональні можливості за замовчуванням для завантаження уявлення, показуючи їх, обертаючи їх у відповідь на поворот пристрою, а також кілька інших стандартних систем поведінки;
- об'єкт UIWindow виводить і координує один або кілька View на екрані. Більшість програм містять тільки один Window, в якому представлений весь контент програми на головному екрані. Але можуть знадобитися додатковий Window. Наприклад, для відображення допоміжних елементів на зовнішньому екрані;
- об'єкти View, об'єкти Controls, об'єкти Layer. Об'єкти View забезпечують візуально уявлення вмісті додатка. View є об'єктом, який малює зміст в призначенній прямокутній області і реагує на події в межах цієї області. Об'єкти Controls вдають із себе спеціальні типи View, що відповідають за реалізацію звичних об'єктів інтерфейсу: кнопки, текстові поля, перемикачі і т. д. Об'єкти Layer – це об'єкти даних, які вдають із себе візуальний контент.

Основний цикл роботи додатка обробляє всі призначені для користувача події. Об'єкт UIApplication запускає основний цикл з моменту запуску і використовує його для обробки подій і обробки змін в інтерфейсі. Як випливає з назви, основний цикл виконується в основному потоці додатка. Така поведінка гарантує, що події зв'язкові з користувачем будуть оброблятися послідовно в тому порядку в якому вони були отримані. На рисунку 2.4 зображена архітектура основного циклу і то як відбувається обробка користувальницьких подій після їх отримання в додатку.

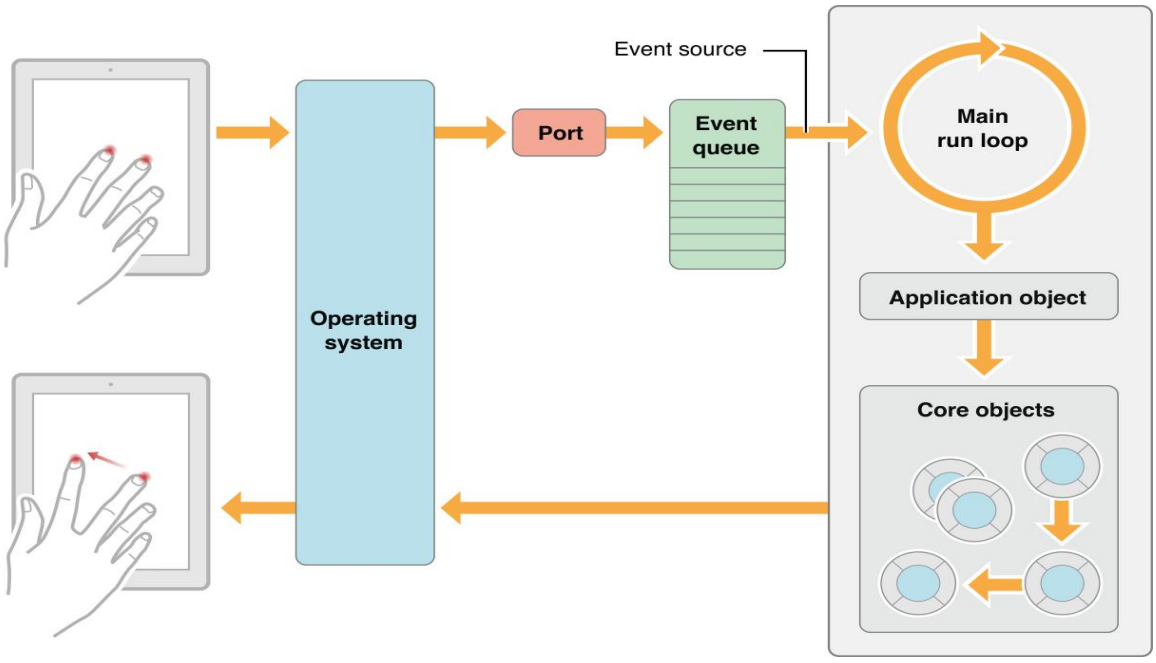

Рисунок 2.4 – Обробка подій користувача в основному циклі

Коли користувач взаємодіє з пристроєм, події, пов'язані з цим взаємодією генеруються системою через спеціальний порт створений UIKit. Події поміщаються в чергу всередині програми і направляються один за іншим в основний цикл для виконання. Об'єкт UIApplication це перший об'єкт який приймає події і приймає рішення про те, що з ним має бути зроблено. Події торкання (touch) зазвичай направляються на об'єкт головного вікна, яке в свою чергу пересилає це повідомлення у view, в якому відбувся цей дотик. Інші події можуть бути прийняті трохи іншим шляхами через різні об'єкти

додатка. У будь-який момент часу додаток знаходиться в одному з п'яти станів:

- не запущений – додаток не було запущено, або запущено але було призупинено;
- неактивний – додаток працює, але в даний час в ньому немає активних подій. Зазвичай в такому стані перебуває недовго, оскільки незабаром переходить в інший стан;
- активний – додаток запущено і в даний час в ньому відбуваються якісь події;
- фоновий режим – додаток перебуває у фоні і виконує певний код. Більшість додатків в цьому стані так само знаходяться недовго. Однак, додаток може запитати додатковий час, для роботи в цьому режимі. І цей стан замінить стан (неактивний);
- припинено – Додаток перебуває у фоні, але не виконує код. Система автоматично поміщає додаток в цей стан, без попередження. При зупинці, додаток перебуває в пам'яті, але воно нічого не виконує. Коли пам'яті буде не вистачати, то система може очистити цей додаток з неї, щоб дати пам'ять додатків з активним станів.

Додаток має бути завжди готовий до того, що його можуть відключити в будь-який час і не повинно чекати, поки збережуться якісь настройки або дані. Вимкнення це нормальна частина життєвого циклу програми. Система зазвичай вимикає додатки, для очищення пам'яті або підготовки до запуску інших додатків, які запущені користувачем, але система так само може вимкнути додатки, які працюють некоректно або не відповідають. Додатки в загальмованому режимі не отримують повідомлення про завершення. Система вбиває процес і відновлює відповідну пам'ять. Якщо додаток запущено у фоні і не відповідає, система викличе applicationWillTerminate: щоб додаток підготувався до вимикання. Система не викликає метод коли пристрій перезавантажується. На додаток, система вимикає додаток, коли користувач вимкнув його за допомогою інтерфейсу мультизадачності. Вимкнення викликане ко-

ристувачем викликає такий же ефект як при виключенні додатка в загальмованому режимі. Процеси додатка видаляються так само без попередження.

Система створює додаток в основному потоці і є можливість створювати окремі потоки, якщо це необхідно, для вирішення будь-яких завдань. Для додатків iOS, найкращим методом є використання Grand Central Dispatch (GCD), які оперують з об'єктами, і інший інтерфейс асинхронного програмування не створюючи і керуючи потоками власноруч. Такі технології як GCD дозволяють визначити роботу, яку необхідно зробити і в якому порядку, але нехай система вирішує як краще виконати цю роботу для CPU. Коли система управляє потоками розробнику легше писати код, йде краще розуміння коду, а так само збільшує загальну продуктивність програми [11].

2.5 Застосування технології управління моделлю даних в iOS

CoreData – це технологія для управління моделлю даних в IOS додатках. Бібліотека призначена для використання в додатках, в яких модель даних вже високо структурована. Замість визначення структур даних програмно, використовується графічний інструмент Xcode для побудови схеми подання моделі даних. Під час виконання екземпляри елементів моделі даних створюються, керуються і стають доступними за допомогою бібліотеки Core Data. Core Data – гнучкий фреймворк для роботи з збереженими на пристрої даними. Звичайно ж є й інші варіанти зберігання даних, які можуть краще підійти при вирішенні певних завдань, але зараз CoreData дуже добре вписується в розробку IOS додатків. Більшість деталей по роботі зі сховищем даних CoreData приховує, дозволяючи сконцентруватися на тому, що дійсно робить мобільний додаток унікальним і зручним у використанні.

CoreData застосовує 4 типа сховища даних: SQLite; Binary; In-Memory; XML (лише для Mac OS) (рис. 2.5).

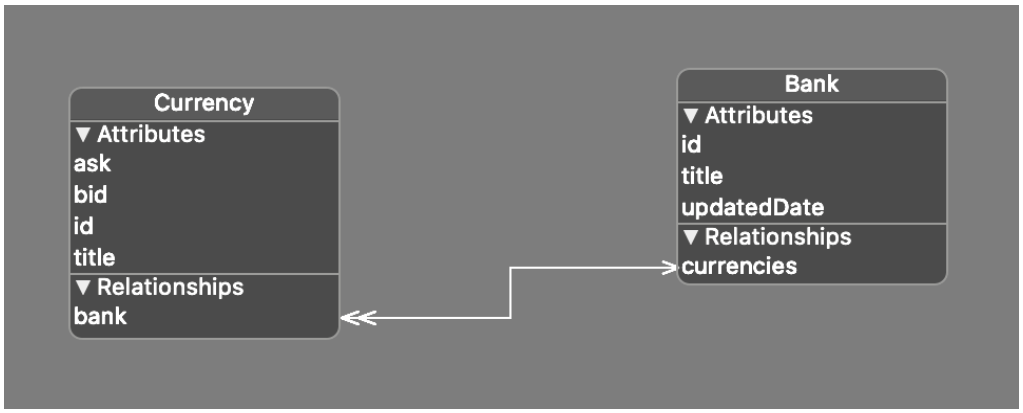

Рисунок 2.5 – Схема моделі даних CoreData

Якщо, наприклад, з міркувань безпеки не доцільно зберігати дані в файловому вигляді, але при цьому потрібно використовувати кешування протягом сесії і дані у вигляді об'єктів, цілком підійде сховище типу «In-Memory». Власне, не забороняється мати в одному додатку кілька сховищ різного типу. Керуючи моделлю даних мобільного застосування, бібліотека істотно скорочує кількість коду, яку необхідно написати. Фреймворк CoreData також надає наступні можливості:

- зберігання даних об'єктів в базі даних SQLite для оптимальної продуктивності;
- клас NSFetchedResultsController для управління результатами в полях-таблицях;
- управління скасуванням або повтором при редагуванні звичайного тексту;
- підтримка перевірки значень властивостей;
- підтримка передачі змін і забезпечення узгоджених зв'язків між об'єктами;
- підтримка угруповання, фільтрації і організації даних в пам'яті.

Ключові компоненти CoreData:

- managed object model (керована об'єктна модель) – фактично це модель (в парадигмі MVC), яка містить усі сутності, їх атрибути і взаємозв'язку;
- managed object contexts (контекст керованого об'єкта) – використовується для управління колекціями об'єктів моделі (в загальному випадку, може бути кілька контекстів);
- persistent store coordinator (координатор постійного сховища) – посередник між сховищем даних і контекстом, в яких ці дані використовуються, відповідає за зберігання даних і їх кешування.

Об'єкт NSManagedObjectContext. Apple дає вельми туманне формулювання для NSManagedObjectContext – середовище для роботи з об'єктами CoreData. Все це від бажання відмежуватися від асоціацій з реляційними базами, і уявити CoreData, як простий у використанні засіб, що не вимагає розуміння ключів, транзакцій та іншої атрибутики. Але на мові реляційних баз NSManagedObjectContext можна, в деякому сенсі, назвати менеджером транзакцій [12].

Для створення сховища в додатку використовуються класи NSPersistentStoreCoordinator або NSPersistentContainer. SPersistentStoreCoordinator створює сховище зазначеного типу на основі моделі, можна вказати розміщення і додаткові опції. NSPersistentContainer можна використовувати з IOS10, дає можливість створення з мінімальною кількістю коду.

Не дивлячись на те, що CoreData може зберігати дані в реляційній базі даних на зразок SQLite, Core Data не є СУБД. Насправді, в якості сховища CoreData може взагалі не використовувати реляційні бази даних. Core Data не є чимось на зразок Hibernate, хоча і надає деякі можливості ORM. CoreData швидше за все є оболонкою для роботи з даними, яка дозволяє працювати з сутностями і їх зв'язками, атрибутами, в тому вигляді, який нагадує роботу з об'єктним графом в звичайному об'єктно-орієнтованому програмуванні.

2.6. Вибір засобів розробки користувацького інтерфейсу додатку

Storyboard – це середовище для розробки користувацького інтерфейсу мобільного застосування [13]. Головна перевага Storyboard – не маючи уявлення, як працює дане мобільне застосування, розробник бачить, як виглядають його сторінки і як вони пов’язані між собою (рис. 2.6).

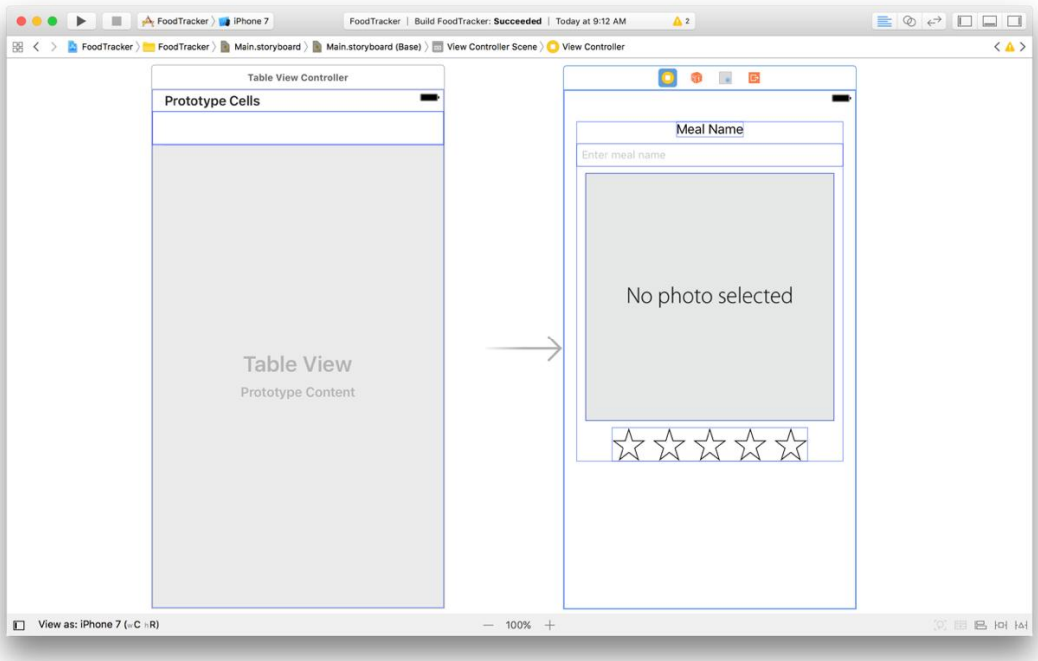

Рисунок 2.6 – Приклад UIView у середовищі Storyboard

Якщо створюється додаток з великою кількістю різних сторінок, то Storyboard дозволяють істотно зменшити кількість коду для їх з’єднання, щоб забезпечити перехід від однієї сторінки до іншої. Раніше розробникам доводилося створювати окремі файли (файли nib або xib) інтерфейсу для кожного окремого view controller (сторінки). Але тепер додатки використовують єдиний Storyboard, який включає в себе дизайн всіх сторінок і який визначає взаємодію між ними. Interface Builder дозволяє побачити результат роботи з Auto Layout без запуску додатка, і якщо якісь constraints не вписуються в загальну схему, Xcode відразу ж попередить про це. Звичайно, існують випад-

ки, коли Interface Builder не здатний забезпечити потрібну поведінку якогось дуже динамічного і складного інтерфейсу, тоді доводиться покладатися на код. Але навіть в таких ситуаціях можна зробити більшу частину в Interface Builder і доповнити це лише парою рядків коду. Так само у Xcode є режим попереднього перегляду, він дозволяє побачити відразу кілька розмірів екрану одночасно без запуску програми. Якщо Storyboard підтримує кілька мов, можна подивитися, як виглядатиме вибраний екран з кожним з них [14].

Створення призначеного для користувача інтерфейсу без Interface Builder супроводжується або великою кількістю шаблонного коду, або суперклас і розширеннями, які тягнуть за собою додаткову роботу з обслуговування. Цей код може проникати в інші частини додатку, ускладнюючи читання і пошук. Використання Storyboards і Xibs дозволяє розвантажити код, завдяки чому, він стає більш сфокусованим на логіці. Storyboards мають ряд переваг:

- у Storyboard зручно спостерігати загальну картину застосування і взаємозв'язків між його сторінками. Можна стежити за будь-чим, тому що загальний дизайн програми міститься в одному єдиному файлі, а не розподілений між декількома файлами nib;
- у Storyboards можуть описувати переходи між різними вікнами. Ці переходи називаються “segues”, які створюються шляхом з'єднання двох сторінок прямо в Storyboard. Завдяки цим segues використовується менше коду для користувацького інтерфейсу;
- у Storyboards полегшують вашу роботу з table view, з cell. Можна створювати таблиці практично повністю в Storyboards, але ж знову таки це саме те, що зменшує в рази код, який би довелося писати;
- Storyboards спрощують роботу при використання AutoLayout.

Але є і недоліки, Storyboard – це XML-файл. Він гірше піддається читанню, ніж код, тому вирішувати конфлікти у системі керування версіями в ньому складніше. Але ця складність також залежить і від того, як ми працю-

ємо зі Storyboard. Можна значно спростити завдання, якщо слідувати наведеним нижче правилам:

- не поміщати весь UI в один єдиний Storyboard, розділити його на кілька дрібніших. Це дозволить розподілити роботу над Storyboards між розробниками без ризику виникнення конфліктів, а в разі їх неминучості – спростить завдання по їх вирішенню;
- якщо потрібно використовувати один і той же View в декількох місцях – виділити його в окремий підклас з власним Xib-файлом;
- робити комміти частіше, так як набагато простіше працювати зі змінами, які надходять невеликими шматками.

Використання декількох Storyboards замість одного позбавляє можливості спостерігати всю карту додатку в одному файлі. Але найчастіше це і не потрібно – досить лише конкретної частини, над якою ми працюємо в даний момент.

Висновок Storyboards – потужна функція, яка дозволяє визначати математичні взаємозв’язки між елементами, які мають певні розміри і позиції, і так само спрощує роботу по відображенню мобільного застосування на різних пристроях з різними розширеннями екрану.

При розробці додатку для моніторингу актуальної фінансової інформації ринку валют доцільно використовувати механізм розташування елементів користувача на екрані AutoLayout (автомакет, автокомпановка).

AutoLayout використовується для побудови динамічних призначених для користувача інтерфейсів, масштабованих і адаптованих до різних форматів і розширень екранів пристроїв, а також до їх орієнтацій. Також AutoLayout робить інтернаціоналізацію більш простим завданням, розміщувати текст змінної довжини на екрані стає простіше, також підтримуються мови з напрямком письма справа наліво, такі як іврит і арабська. Робота AutoLayout заснована на зв'язках (constraints), які встановлюють геометричні відносини між уявленнями призначеного для користувача інтерфейсу. Наприклад, можна створити зв'язок, який говорить: «текстова мітка повинна бу-

ти закріплена на деякий відстані від лівого краю батьківського уявлення і з'єднана з лівим краєм кнопки, з додавання між ними проміжку в 10 px» [13].

AutoLayout бере задані зв'язки і математично обчислює ідеальні позиції і розміри для всіх уявлень. Розробнику не потрібно більше встановлювати розміри уявлень вручну, задавати їх координати розташування – AutoLayout бере цю роботу на себе. Створювати зв'язки можна як програмно, так і за допомогою Interface Builder.

Реактивне програмування в IOS розробці можна реалізувати за допомогою різних фреймворків, таких як ReactiveSwift і RxSwift. Вони змінюють рамки імперативного стилю мови Swift і у такого підходу до програмування є що запропонувати стандартної парадигми. Це, безумовно, і привертає увагу iOS розробників.

Реактивні прив'язки (англ. – Reactive bindingsr) прекрасно вирішують проблему поновлення користувальницького інтерфейсу і ідеально вписуються в архітектуру MVVM. Динамічне зв'язування ViewModel і View дозволяє описувати бізнес логіку у відриві від уявлення і мінімізує код у ViewController. За допомогою реактивних прив'язок можна легко дізнатися коли стан нашої ViewModel змінився і ця зміна буде відображена в UI.

Для реалізації додатку для моніторингу актуальної фінансової інформації ринку валют було обрано реактивний фреймворк ReactiveSwift і ReactiveCocoa для впровадження реактивних прив'язок.

3 ПРОЕКТУВАННЯ ДОДАТКУ

Наступним етапом розробки додатку для моніторингу актуальної фінансової інформації ринку валют є виконання проектування додатку, визначення функціональних можливостей додатку, розробка інтерфейсу. Додаток повинен забезпечувати зацікавлених користувачів актуальною інформацією щодо котирування валютного ринка та надавати можливість доступу до найважливіших фінансових новини в Україні. Інформація буде завантажуватися з фінансового порталу МінФін за допомогою публічного API та технології парсингу HTML-сторінок.

Основним призначенням та метою розробки додатку для моніторингу актуальної фінансової інформації ринку валют є забезпечення зручного додатку для відстеження наступної фінансової інформації:

- курсів валют,
- швидкого механізму конвертації валют,
- актуальних фінансових новин,
- відстеження курсів валют в банках.

3.1 Визначення функціональних можливостей користувачів додатку

Мобільний додаток для моніторингу актуальної фінансової інформації ринку валют (надаємо йому ім’я iKurs) забезпечує користувачам доступ до актуальної фінансової інформації. На етапі проектування важливо було розробити максимально зручний і простий у використанні інтерфейс, так як додаток розрахований для широкого кола користувачів, в тому числі для недуже досвідчених користувачів мобільних інтерфейсів. Визначимо основні функції додатку (рис. 3.1):

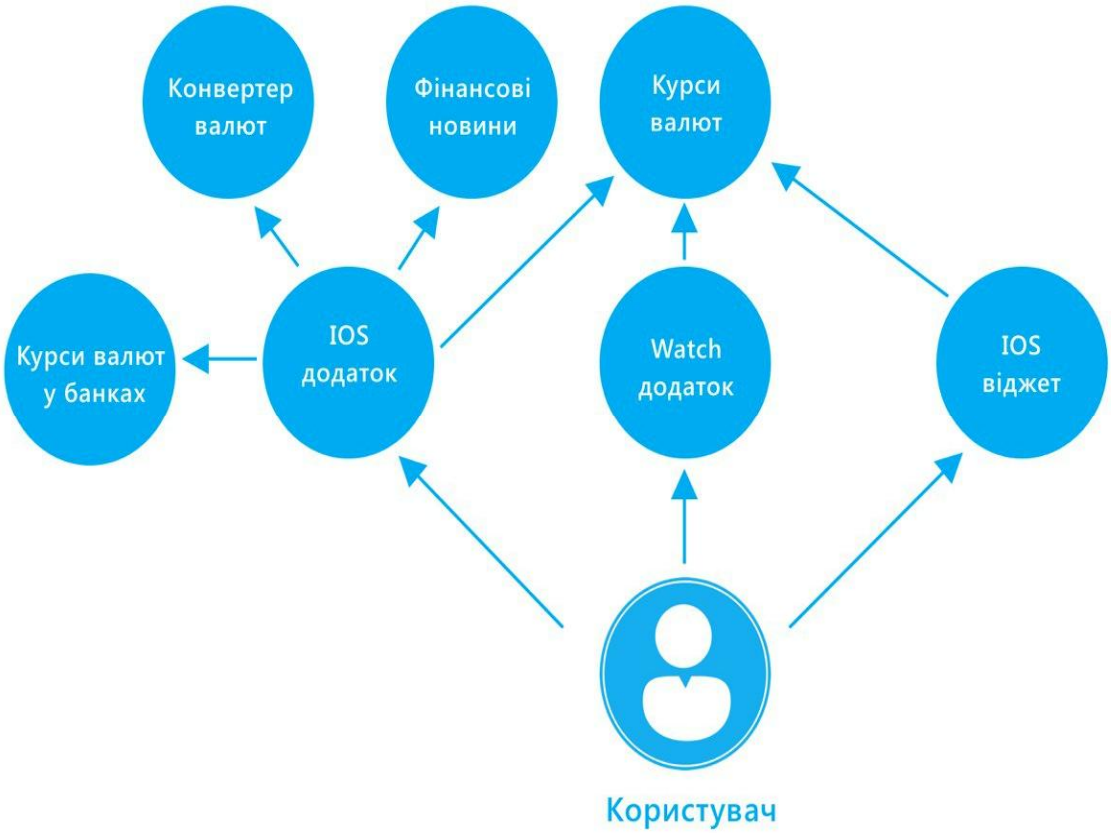

Рисунок 3.1 – Діаграма функціональних можливостей користувача

Користувачу мобільного додатку для для моніторингу актуальної фінансової інформації ринку валют доступні наступні функції у системі:

- відображення актуальних курсів основних валют, с можливістю ручного поновлення даних;
- зручний калькулятор валют;
- отримання важливих фінансових новин;
- курси валют в банках, а так само актуальна контактна інформація про банки;
- віджет курсів валют;
- додатковий модуль додатку на Apple Watch.

При завантаженні IOS програми відразу доступні курси продажу і покупки трьох валют: Долар США, Євро, і Польський злотий. Вибір даних ва-

лют був обумовлений наявністю їх в API порталу Мінфін. Схема навігації IOS програми відображена на рис. 3.2.

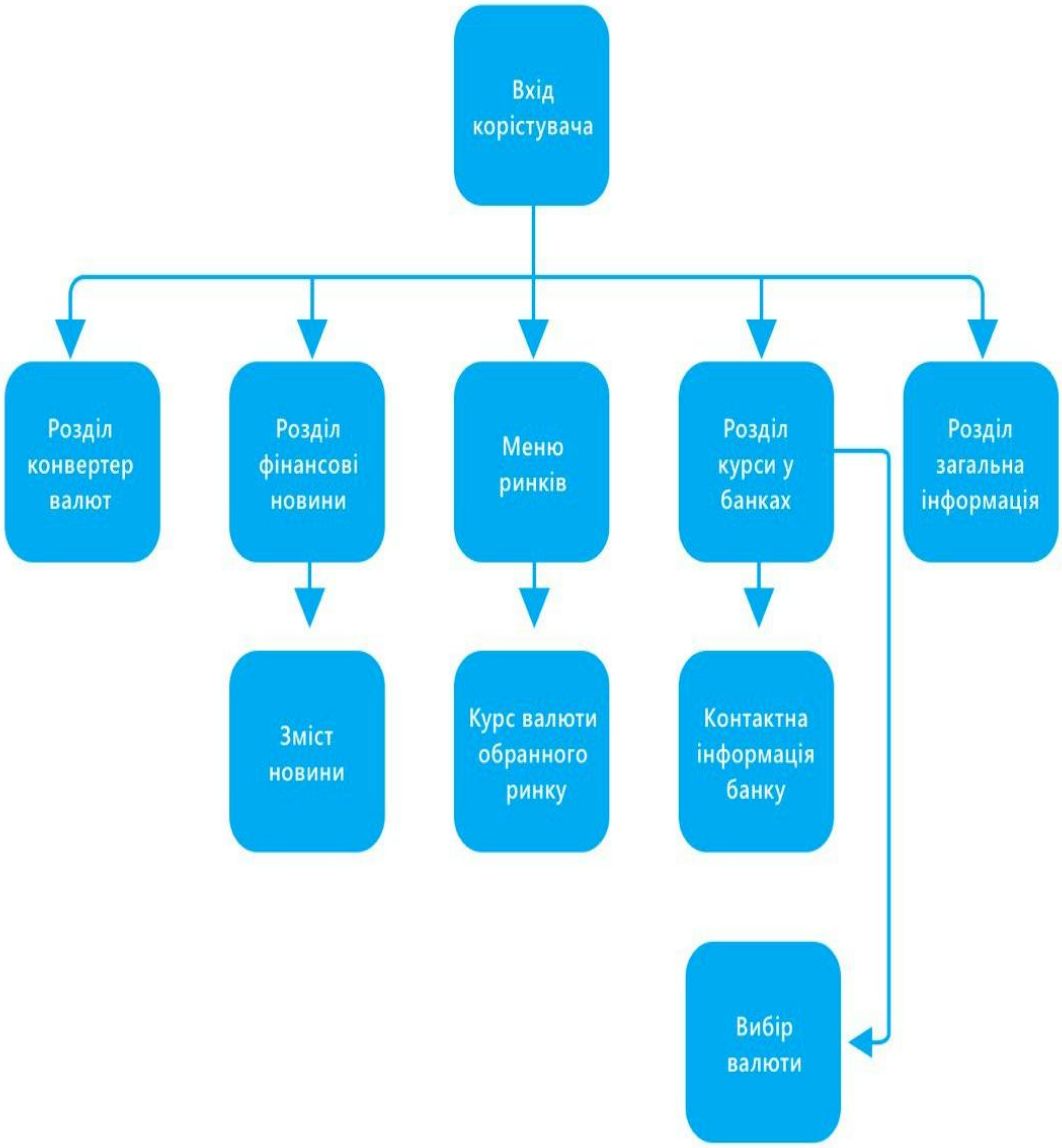

Рисунок 3.2 – Схема навігації та функціональні можливості IOS додатку

Інформація відображається по одній валюті з можливістю використання жесту свайп для перемикання на іншу валюту. Так само для ручного поновлення даних доданий функціонал: «Pull to refresh», який дозволяє користувачеві потягнувши вниз UI елемент і потім відпустити його оновивши дані про курси валют з сервера.

На головному екрані між меню навігації і інформером курсів, розташований рядок, що біжить з останньою рандомною новиною після натискання, на яку відкривається екран з контентом цієї новині.

У нижньому меню навігації доступні переходи на конвертер валют, список фінансових новин, меню перемикання валютних ринків, банківських курсів і на екран загальної інформації про додаток, де вказані посилання на джерела інформації та контакти розробників, а так само є функціонал поділитися додатком, який дозволяє поширити додаток серед контактів користувача через популярні месенджери і соціальні мережі.

Реалізоване у додатку Меню перемикання ринків включає наступні розділи:

– готівковий – відображає курси валют на чорному ринку;
– середній в банках – відображає середній банківський курс;
– міжбанк – відображає міжбанківський курс;
– НБУ – відображає офіційний курс Національного банку України;
– криптовалюта – відображає курс популярних крипто валют.

Обраний ринок валют передається на екран конвертера на якому користувач може легко конвертувати потрібну йому валюту в іншу, за курсом продажу або покупки даного ринку.

На екрані списку новин завантажується 40 найактуальніших фінансових новин України. При натисканні на конкретну новину відкривається екран з детальною інформацією що до цієї новини – картинка, текст посилання, графіки і т.п. Користувач зможе дізнатися про стан економіки країни, прогнози курсів валют від фінансових експертів, зміну в оподаткуванні та іншу важливу фінансову інформацію.

Екран курсів в банках відображає список банків України з курсами валют в касах і при використанні банківських карт. При натисканні на осередок з банком вона розсувається і відображається контактна інформація банку, користувач може перейти на сайт банку, зателефонувати або написати електро-

нного листа в банк. Так само є функція сортування списку, можна впорядкувати його за назвою банку і за кращим курсом покупки або продажу.

Функціонал додатку на Apple Watch і віджета досить простий. Віджет просто відображає курс трьох валют останнього обраного ринку в IOS додатку. Watch OS додаток дозволяє користувачу жестом свайп перемикатися між валютами імітуючи поведінку додатку на IOS. Так як Apple Watch має функцію «Force Touch» – ця технологія дозволяє користувачу при сильному натисканні на екран годин провести якусь дію в додатку. Було прийнято рішення реалізувати меню перемикання ринків використовуючи цю технологію. Так само при тривалому натисканні користувач може вибрати основну валюту, яка буде завжди відображатися першою при завантаженні додатка.

3.2 Проектування бази даних додатку

У додатку для моніторингу актуальної фінансової інформації ринку валют необхідно реалізувати механізм кєшування. Кєшування застосовується тоді, коли потрібно збільшити швидкодію клієнт-серверних додатків за рахунок тимчасового збереження результатів запитів до сервера і подальшого їх використання, тим самим знизивши навантаження на сервер і зменшивши в середньому швидкість отримання запиту на клієнті. Для зберігання запитів до сервера потрібна локальна база даних на мобільному пристрої. Використовуємо для цього інструмент CoreData, так як він входить до складу пакету XCode. CoreData є бібліотекою, що забезпечує інфраструктуру для управління, збереження і вилучення об'єктів зі сховища. За базу даних, керованої CoreData виступає SQLite.

При проектуванні бази даних потрібно виділити сутності які будуть зберігатися в ній. У головному розділі курсів валют основною сутністю є Ринок (англ. Market) у цієї сутності повинен бути унікальний ідентифікатор і назва. Так само, так як маємо враховувати обмеження від API, запитувати дані не частіше ніж 5 хвилин, потрібно зберігати час останнього запиту, щоб

була можливість порівнювати час перед наступним запитом. У кожної сутності Market повинний бути зв'язок з валютами. Кожна сутність валюта (англ. Currency) повинна зберігати курс покупки і продажу, мати унікальний ідентифікатор і назву, а також зв'язок з ринком.

У розділі фінансових новин єдиною сутністю є сутність Новина (англ. News). У якій повинні бути такі поля, як унікальний ідентифікатор, назва, дата створення, посилання на картинку або фото новини, посилання на саму новину на сайті джерела, текст новини, а також повинен бути атрибут за яким можна буде зрозуміти чи читав користувач цю новину.

У розділі курсів валют в банках головною сутністю є сутність Банк (англ. Bank). Дана сутність повинна мати унікальний ідентифікатор, дату актуальності даних, назва банку, логотип і контактні дані: адресу електронної пошти, посилання на сайт, телефон. Дані про курси валют в банках будемо отримувати способом парсингу html сторінок порталу Мінфін, а потім зберігати посилання на дану сторінку банку.

У кожного банку є зв'язок з сутностями BankCurrency, які зберігають інформацію курсу продажу і покупки в касі банку і при здійсненні операцій з банківськими картами (безготівкові операції). BankCurrency має унікальний ідентифікатор, назву валюти та мобільний банкінг.

Після визначення основних сутностей, необхідно провести проектування реляційної бази даних iOS додатку для моніторингу актуальної фінансової інформації ринку валют додатку. Xcode дозволяє легко і зручно створювати сутності та їх поля, а й так само відносини і зв'язки між ними, використовуючи графічний інтерфейс. Після чого будуть згенеровані файли з нативними Swift моделями даних і створені відповідні поля і таблиці в базі даних.

У кожної сутності спроектованої бази даних iOS додатку для моніторингу актуальної фінансової інформації ринку валют атрибут id є первинним ключем. Діаграму бази даних додатка «Сутність – Зв’язок» наведено на рисунку 3.3.

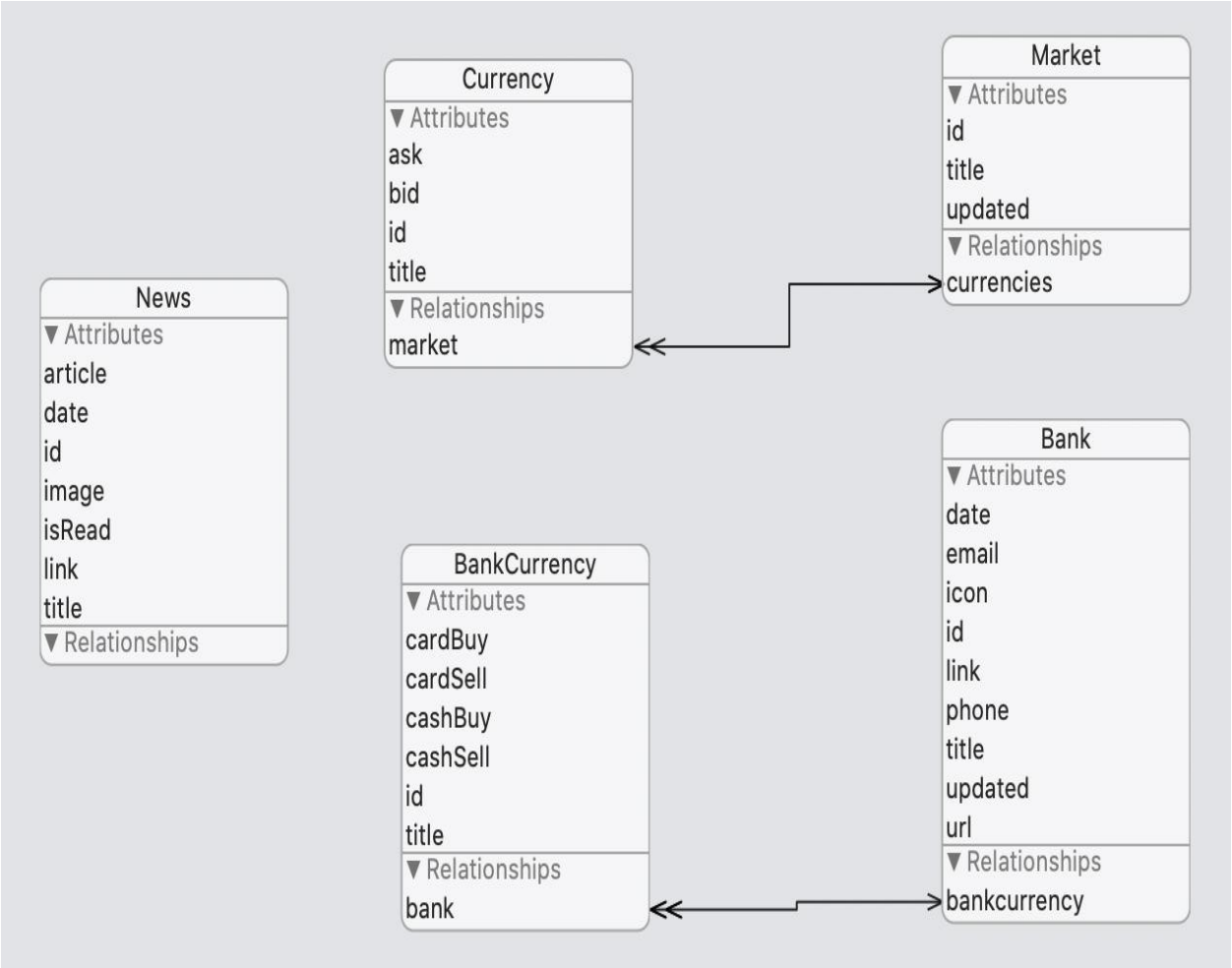

Рисунок 3.3 – Діаграма «Сутність – Зв’язок» бази даних додатку

Сутність Market використовує зв’язок один до багатьох з сутністю Currency. Сутність News не має зв'язків і відносин з іншими сутностями. Сутність Bank використовує зв’язок один до багатьох з сутністю BankCurrency. Сутність Market (табл. 3.1) має наступні атрибути: id – який є числовим значенням; title – зберігає назву ринку; updated – дата останнього запиту.

Таблиця 3.1 – Сутність «Market»

№	Атрибут	Тип
1	Id	int(11)
2	title	varchar(25)
3	updated	datetime

Сутність Currency (табл. 3.2) має наступні атрибути: id – який є числовим значенням; title – зберігає назву валюти; ask – який є числовим значенням; bid – зберігає назву ринку.

Таблиця 3.2 – Сутність «Currency»

№	Атрибут	Тип
1	Id	int(11)
2	title	varchar(25)
3	ask	int(15)
4	bid	varchar(25)

Сутність Currency (табл. 3.3) має наступні атрибути: id – який є числовим значенням; date – дата створення новини; article – зберігає зміст новини в текстовому полі; isRead – зберігає числове значення яке відповідає 1, якщо користувач читав цю новину і 0, якщо ні; link – текстове поле яке зберігає посилання на новину; title – зберігає заголовок новини; Image – посилання на картинку новини.

Таблиця 3.3 – Сутність «News»

№	Атрибут	Тип
1	Id	int(11)
2	date	datetime
3	article	text
4	isRead	int(2)
5	link	varchar(100)
6	title	varchar(25)
7	Image	varchar(100)

Сутність Currency (табл. 3.4) має наступні атрибути: id – який є числовим значенням; title – зберігає назву банку; updated – дата останнього запиту; URL – посилання на html сторінку банку; phone – зберігає номер телефону;

link – дата останнього запиту; icon – який є числовим значенням; email – зберігає електрону адресу банку; date – дата актуальності даних.

Таблиця 3.4 – Сутність «Bank»

№	Атрибут	Тип
1	Id	int(11)
2	title	varchar(50)
3	updated	datetime
4	URL	varchar(100)
5	phone	varchar(25)
6	link	varchar(100)
7	icon	int(10)
8	email	varchar(50)
9	date	datetime

Сутність BankCurrency (табл. 3.5) має наступні атрибути: id – який є числовим значенням; title – зберігає назву валюти; cardBuy; cardSell, ashBuy, cashSell.

Таблиця 3.5 – Сутність «BankCurrency»

№	Атрибут	Тип
1	Id	int(11)
2	title	varchar(50)
3	cardBuy	varchar(40)
4	cardSell	varchar(40)
5	cashBuy	varchar(40)
6	cashSell	varchar(40)

3.3 Розробка сервісів для роботи з даними

Для роботи з протоколом HTTP в мобільному додатку для моніторингу актуальної фінансової інформації ринку валют використовується бібліотека Alamofire. Вона надає зручні інструменти для спрощення доступу до API і

обробки відповідей сервера. Для парсингу відповіді з сервера в нативні об'єкти використовується інтерфейс Decodable, який з'явився з появою четвертої версії мови Swift. Для роботи з базою даних використовується фреймворк CoreData. Ще є потреба парсити html сторінки, тому для реалізації був обраний фреймворк Kanna.

Архітектура структури системи для роботи з даними складається з сервісів для отримання потрібних типів даних. На рисунку 3.4 відображена схема взаємовідносин між об'єктами створеної системи роботи з даними, яка дозволяє не дублювати, а перевикористовувати існуючий код.

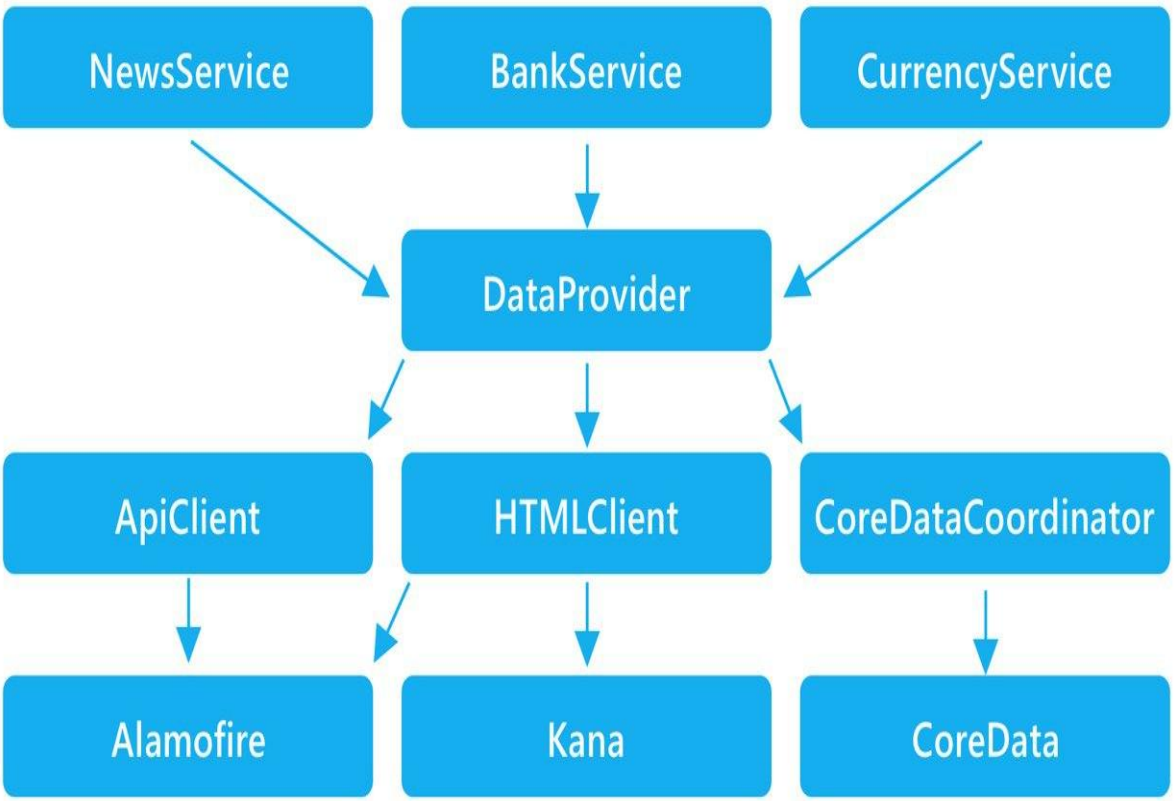

Рисунок 3.4 –Архітектура системи роботи з даними

Клас NewsService складається з статичних методів отримання новин та конкретної новини:

– static func fetchNews (cachePolicy: CachePolicy);

– static func fetchNews (by id: String, cachePolicy: CachePolicy);

Подібні методи для роботи з даними реалізують BankService і CurrencyService. Ці методи звертаються до класу DataProvider який в свою чергу несе відповідальність за роботу з даними.

DataProvider знає за якими даними потрібно звертатися до сервера використовуючи API, а за якими оперуючи засобами парсингу HTML сторінок. Щоб з'ясувати причину порушення даних був реалізований enum CachePolicy, який має два значення, які вказують звідки потрібно отримувати дані з сервера або з бази даних:

```
enum  CachePolicy {
case fromCache
case  fromNetwork
}
```

CachePolicy передається через параметри методів сервісу далі в DataProvider (дані будуть запитуватися з бази даних):

```
NewsService.fetchNews(by : id, cachePolicy: .fromCache)
//дані будуть запитуватися з сервера:
NewsService.fetchNews(by : id, cachePolicy: .fromNetwork)
```

DataProvider складається з методів отримання всіх потрібних сутностей і звертається до відповідного клієнту за ними. Для отримання основних курсів валют використовує ApiClient, йому віддана головна роль в механізмі запитів на сервер в проекті. Це клас, який реалізує шаблони «Одинак» і «Фасад», що приховує за собою настройки підключення до API сервера і методи запитів по ключам API. Фрагмент реалізації класу APIClient з універсальним методом `send request`:

```
final class APIClient {
static let shared = APIClient()
func send(request: URLRequestConvertible & NetworkRouter-
Params,
 qos: DispatchQoS.QoSClass = .default)
```

```
    -> SignalProducer<AnyObject, Error> {
    return SignalProducer {[unnowned self]  sink, disposable in
                sessionManager.request(request)
    response(queue: DispatchQueue.global(qos: qos)) { response
in
         guard response.error?.isConnectionError != true else {
    self.handleConnectionError(for: request, observer: sink);
return
                    }
          guard let statusCode = response.response?.statusCode
            else { sink.sendInterrupted(); return }
                    switch(statusCode) {
                    case 204:
    sink.complete(value: ("" as AnyObject))
        case 200...203, 205, 409:
     self.handleResponse(for: response.data!, observer: sink)
                    case 404:
                        sink.complete(value: ("" as AnyObject))
                    case 400...403, 405..<600:
    self.handleErrorResponse(for: response.data!, observer:
sink)
                    default:
       sink.send(error: Error.commonError(with: "unknown code"))
                    }
                }
            }
        }
    }
```

За отриманням новин і банківської інформації DataProvider звертається до HTMLClient, який займається парсингом html сторінки і декодує результат в нативні об'єкти. HTMLClient завантажує html сторінку використовуючи фреймворк Alamofire. Приклад завантаження html сторінки:

```
    func getNewsHTML(with url : String) {
    requestNews?.cancel()
    requestNews = Alamofire.request(url, method: .get)
    .validate(statusCode: 200..<404).responseString { [weak
self] response in
                if let value = response.result.value {
                    DispatchQueue.global().async {
                        self?.parseNews(value)
                    }
                } else if let error =  response.result.error {
                       self.send(error: error)
                }
    }
```

Потім за допомогою фреймворка Kanna виконується парсинг по потрібним тегам і зберігається в потрібний тимчасовий об'єкт:

```
let document = try? Kanna.XML(xml: response.result.value!,
   encoding: .utf8)
if let doc = document {
let titles = doc.xpath("//channel/item/title")
for title in titles {
self.news["title"]?.append(title.text!)
}
let dates = doc.xpath("//channel/item/pubDate")
for date in dates {
let tempDate = String().getDateWithString(date.text!)
                    let formatter = DateFormatter();
               formatter.dateFormat = "yyyy-MM-dd HH:mm:ss"
let defaultTimeZoneStr = formatter.string(from: tempDate)
self.news["date"]?.append(defaultTimeZoneStr
          }
          let links = doc.xpath("//channel/item/link")
                    for link in links {
   let slink = link.text!.components(separatedBy: "/")
   let id = slink[slink.count-2
self.news["id"]?.append(id)
self.news["link"]?.append(link.text!)
                     }
            }
```

Отриманий об'єкт декодується в нативну модель за допомогою протоколу Decodable. Протокол Decodable, який може декодувати себе з зовнішньої репрезентації, використовується для типів, які можуть бути декодовані. Також містить всього один метод: init (from :) створює новий екземпляр класу, декодуючи із заданого декодувальнику. Приклад декодування сутності News:

```
if let jsonData = jsonString.data(using: .utf8){
    let newsObject = try? JSONDecoder().decode(News.self,
from: jsonData)
```

За роботу з базою даних, а точніше з CoreData відповідає CoreDataCoordinator, який реалізує конфігурацію стека і містить інтерфейс для зручної

роботи з даними зберігаючи або запитуючи їх. Методи CoreDataCoordinator: методи які створюють об'єкти і записують в базу даних:

```
func createMarket() -> Market
func createCurrency() -> Currency
func createNews() -> News
func createBank() -> Bank
//приклад реалізації методу create
func createCurrency() -> Currency {
        let ucurrencyEntity =  NSEntityDescrip-
tion.entity(forEntityName: "Currency", in: managedObjectCon-
textPrivate)
   let ucurrency = NSManagedObject(entity: ucurrencyEntity!,
insertInto: managedObjectContextPrivate) as! Currency
        return ucurrency
```

Далі наведено методи для видалення об'єктів з бази даних:

```
func remove (news : News)
func remove (market: Market)
//приклад реалізації методу remove
func remove (bank : Bank) {
managedObjectContextPrivate.delete(news)
}
```

Методи для запиту об'єктів, що зберігаються в базі даних, реалізовані наступним чином:

```
func fetchAllNews (completion : @escaping (_ news :
[News]?) -> Void)
func fetchAllBanks (completion : @escaping (_ ubanks :
[Bank]?) -> Void)
func fetchAllBanksByCurrencyTitle (title : String, comple-
tion : @escaping (banks : [Bank]?) -> Void)
func fetchBanks (request : NSFetchRequest<Bank>, completion
: @escaping (banks : [Bank]?) -> Void)
func fetchNews (request : NSFetchRequest<News>, completion
: @escaping (news : [News]?) -> Void)
func fetchNews (by id : NSNumber, completion : @escaping
(news : News?) -> Void)
func fetchBank (by identifier: String, completion :
@escaping (bank : Bank?) -> Void )
```

Наведемо приклад реалізації методу fetch у системі:

```
    func fetchBank(by id : NSNumber, completion : @escaping
(bank : Bank?) -> Void ) {
            let entityBank = NSEntityDescrip-
tion.entity(forEntityName: "Bank", in: managedObjectContextPri-
vate)
            let request = NSFetchRequest<Bank>()
            request.entity = entityBank
          request.predicate = NSPredicate(format:"id == %@",id)
        self.managedObjectContextPrivate.perform {
                do {
                    let banks = try
self.managedObjectContextPrivate.fetch(request)
             completion(banks.count > 0 ? banks[0] : nil)
                }
        tch _ as NSError {
                    completion(nil)
                }
```

Метод зв'язування об'єктів: func addCurrencyToBank(bank: Bank, currency: Currency).

Таким чином CoreDataCoordinator реалізує можливість зручно працювати з сутностями, які зберігаються в базі даних і взаємодіють з класами нижчого рівня в системі.

3.4 Розробка навігації екранів додатка

Навігація між екранами – це те, з чого взагалі починається додаток ще на етапі створення прототипу, коли розробник навіть до кінця не вирішив: як будуть виглядати екрани, які будуть анімації, чи буде кешування даних. Екрани можуть бути порожніми або статичними картинками, але завдання навігації з'являється в додатку, як тільки цих екранів стає більше одного. Тобто фактично відразу. Найбільш поширені методи побудови архітектури iOS додатків: MVC і MVVM, описують те, як побудувати один екран-модуль. Ще там говориться про те, що модулі можуть знати один про одного, спілкуватися один з одним і т.д. Але зовсім мало уваги приділяється питанням, як відбуваються переходи між цими модулями, хто приймає рішення про ці переходи, і як передаються дані. iOS надає кілька способів показати наступний за сценарієм екран:

– UlStoryboard + segues, коли користувач позначає всі переходи між екранами в одному мета-файлі, і потім їх викликає;
– контейнери Containers – такі, як UINavigationController. UITabBar-Controller, UIPageController або контейнери створені розробниками, які можна використовувати як програмно, так і разом зStoryBoards;
– метод present (_:animated: completion:). Це просто метод класу UI-Controller.

У самих цих інструментах проблем немає. Проблема в тому, як саме вони зазвичай використовуються. UINavigationController, performSegue, prepareForSegue, метод presentViewController – це все property-методи класу UIViewController. Apple пропонує користуватися цими інструментами всередині самого UIViewController. Такий підхід веде до проблем в побудові навігації (рис. 3.5).

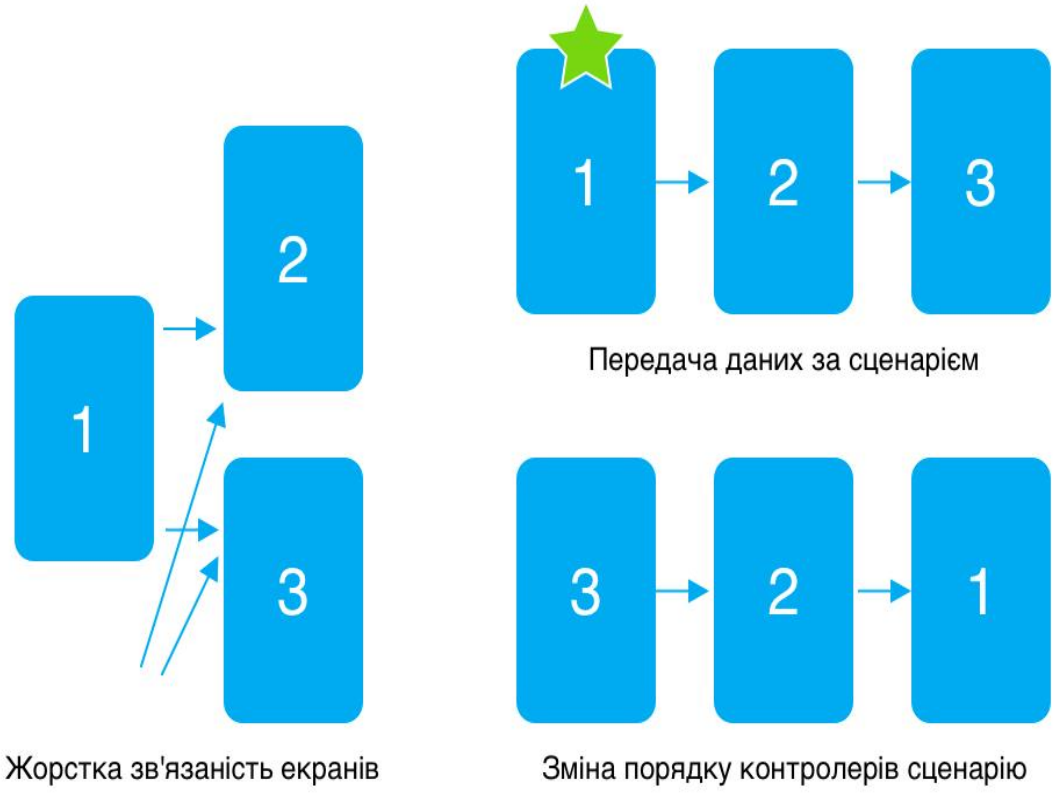

Рисунок 3.5 – Схема виникаючих проблем при побудові навігації

Розглянемо схему з жорсткою зв’язаністю екранів. Це означає, що екран 1 знає про існування екрану 2. Мало того, що він знає про його існування, він його ще й потенційно створює, або бере з segue, знаючи, якого він типу, і передає йому якісь дані. Якщо розробнику знадобиться при якихось обставинах показати замість екрану 2 екран 3, то доведеться знання про новий екрані 3 точно так же зашивати в контролер екрана 1. Все будується ще складніше, якщо контролери 2 і 3 можуть викликатися ще з кількох місць, не тільки з екрана 1. Виходить, що знання про екран 2 і 3 доведеться зашивати в кожне з цих місць.

Розглянемо схему, яка передбачає зміну порядку контролерів сценарію. Це теж не так просто через пов'язаності. Щоб поміняти місцями два ViewController, недостатньо буде зайти в UlStoryboard і поміняти місцями 2 картинки. Доведеться відкривати код кожного з цих пунктів меню, переносити його в налаштування наступного, міняти його місцями, що не дуже зручно.

Розглянемо схему, передачі даних за сценарієм. Наприклад, при виборі чогось на екрані 3, нам потрібно оновити View на екрані 1. Так як у розробника спочатку немає нічого, крім ViewController, доведеться якимось чином зв'язати ці два ViewController – не важливо як – через делегування або якось ще. Ще складніше буде, якщо за дією на екрані 3 потрібно буде оновити не один екран, а відразу декілька, наприклад, і перший, і другий. В такому випадку делегуванням обійтися не вдасться, тому що делегування – це зв'язок один до одного. Якщо використовувати нотифікацію або shared state, то це ускладнить налагодження і відстеження потоків даних в додатку.

Для того щоб вирішити перераховані вище проблеми потрібно:

– заборонити додатку всередині UIViewController звертатися до контейнерів, тобто до self.navigationController, self.tabBarController або ще до яких-небудь контейнерів, які були створені як property extension. Розробник тепер не зможе з коду екрану взяти свій контейнер і попросити його щось зробити;

- заборонити всередині UIViewController викликати метод performSegue і писати код в методі prepareForSegue, який би брав наступний за сценарієм екран і займався його налаштуванням. Тобто розробник більше не працює з segue (з переходами між екранами) всередині UIViewController;
- забороняємо будь-які згадки про інших контролерів всередині нашого конкретного контролера: ніяких ініціалізацій, передач даних.

Так як всі ці обов'язки прибираються з UIViewController, потрібна буде нова сутність, яка буде їх виконувати. Для цього був створений новий клас об'єктів – Coordinator. Координатор – це просто звичайний об'єкт, якому ми передаємо на старті NavigationController і викликаємо метод Start. Тепер він починається не з того, що ми готуємо перехід на якийсь конкретний екран NavigationController, а у координатора викликаємо метод Start, передавши йому перед цим в ініціалізатор NavigationController. Координатор розуміє, що в NavigationController пора показати перший екран, що він і робить.

Далі, коли користувач натискає наприклад, на елемент меню виклику конвертера валют, який цю подію передає наверх координатору. Тобто екран сам нічого не знає, він передає повідомлення координатору, і далі координатор реагує на це тим (так як у нього є NavigationController), що відправляє в нього наступний крок – це дані про курси валют поточного ринку.

Контролери тепер не спілкуються один з одним, вирішуючи, хто буде наступний, і не передають один одному ніякі дані. Більш того, вони взагалі нічого не знають про своє оточення. Координатор має два property: ViewModel потрібного UIViewController; NavigationController, який потрібно передати при старті.

Є простий init(), який заповнює ці property. Властивість NavigationController ініціалізується в батьківському класі BaseCoordinator. Далі є метод start(), який призводить до того, що викликається потрібний ViewController. Нижче наведено приклад реалізації Координатора для екрану конвертера валют:

```
    final class ConverterCoordinator:  BaseCoordinator {
        private let viewModel: ConverterViewModel

        init(viewModel: ConverterViewModel) {
            self.viewModel = viewModel
        }
        override func start() {
            let viewController = ConverterViewControl-
ler(viewModel: self.viewModel)

self.navigationController.pushViewController(viewController,
animated: true)    }
    }
```

Кожен сценарій розробник тепер загортає всередину свого координатора. Розробник повинен мати можливість, насправді, стартувати ці сценарії з будь-якого місця в додатку для моніторингу актуальної фінансової інформації ринку валют. У цьому повинна бути гнучкість – координатор повинен бути повністю самодостатній. Такий підхід в розробці дає додаткову зручність, яка полягає в тому, що якщо в даний момент ведеться робота з конкретним сценарієм, то не потрібно кожен раз при запуску до нього добиратися по стеку навігації.. Схема взаємовідносин координаторів приведена на рисунку 3.6.

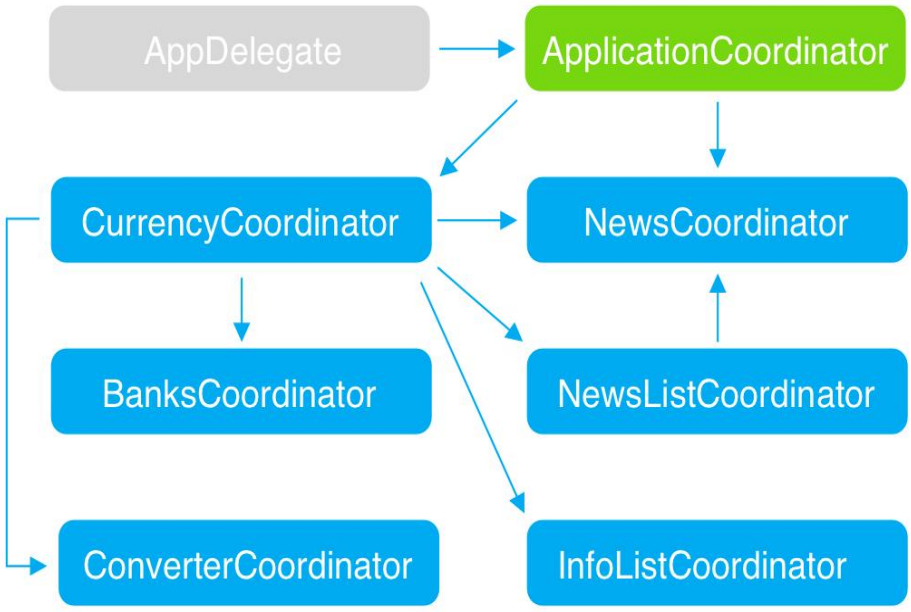

Рисунок 3.6 – Схема взаємовідносин координаторів

Далі потрібно визначити, який сценарій може призвести до старту другого, і з цих сценаріїв скласти дерево.

У нашому випадку дерево просте: координатор курсів валют може стартувати всі інші координатори. Тут майже все встає на свої місця, але залишається дуже важлива деталь – точка входу. Цією точкою входу буде особливий координатор – ApplicationCoordinator. Його створює і стартує AppDelegate, і далі він вже управляє логікою на рівні додатку, тобто тим, який координатор стартує прямо зараз. Приклад реалізації ApplicationCoordinator:

```
final class ApplicationCoordinator: BaseCoordinator {
private var appDelegate: AppDelegate!
    init(appDelegate: AppDelegate) {
        self.appDelegate = appDelegate
    }
    override func start() {
           self.showRootViewController()
    }
    func showRootViewController() {
        self.showCurrency(appDelegate) :
    }
}
```

У наведеному коді у ApplicationCoordinator передається AppDelegate, а при визові метода start викликається метод, який презентує потрібний CurrencyViewController. Підсумками реалізованого рішення є:

- незалежні екрани і сценарії, які нічого один про одного не знають, один з одним не спілкуються – що і потрібно було домогтися;
- легко змінювати порядок екранів в додатку без зміни кодів екранів. Якщо все зроблено, як треба, єдине, що має змінитися в додатку при зміні сценарію, це не код екранів, а код координатора;
- спрощується передача даних між екранами та інші завдання, які мають на увазі під собою зв'язок між екранами;
- щоб почати його застосовувати, не потрібно додавати в проект ніяких сторонніх залежностей і розбиратися в чужому коді.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКА

4.1 Розробка інтерфейсу IOS додатка

У Xcode розробка інтерфейсів можлива використовуючи Interface Builder (Storyboard і Xib) або безпосередньо кодом. Виходячи з проведеного аналізу було прийнято рішення використовувати IB, щоб спростити завдання і скоротити час розробки. У проекті створюється файл .storyboard і весь інтерфейс завантажується з нього. Усередині цього файлу описана головна сцена курсів валют (рис. 4.1).

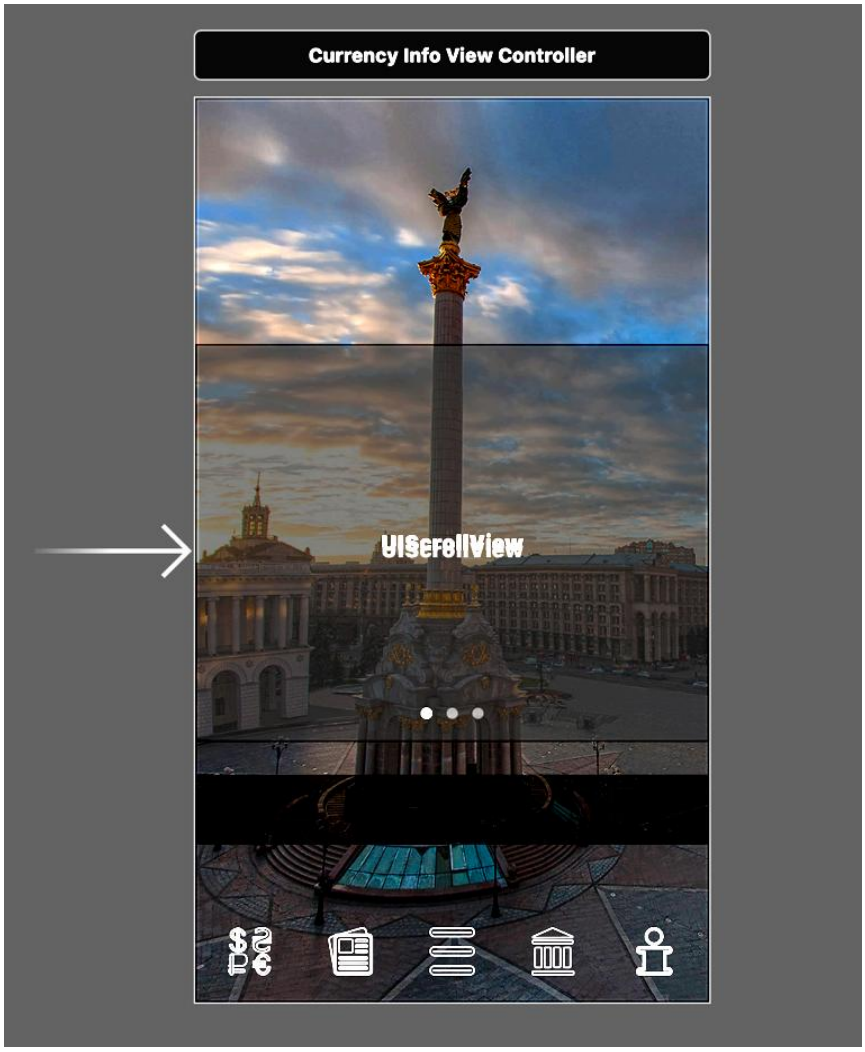

Рисунок 4.1 – Головний екран додатка у середовищі Storyboard

IB дозволяє перетягуванням потрібних UI елементів, швидко зверстати екран і розставити обмеження autolayout. На головному екрані внизу розташовані елементи UIButton після натискання на які відбувається перехід на потрібні екрани. Вище розміщений UILabel з напівпрозорим фоном який відображає випадкову новину, на нього доданий жест UITapGestureRecognizer для того щоб користувач зміг перейти на екран з контентом новини.

```
     // ініціалізація і додавання жесту
     let tap = UITapGestureRecognizer.init(target: self, action:
#selector(ewsLabelTapped (_:)))
     self.newsLabel.addGestureRecognizer(tap)

     // метод обробки натискання користувачем на елемент
     @IBAction func newsLabelTapped(_ sender: AnyObject) {
             let newsDetailVC = NewsDetailViewControl-
ler(nibName: "NewsDetailViewController", bundle: nil)
             let navigationVC = UINavigationControl-
ler(rootViewController: newsDetailVC)
             self.present(navigationVC, animated: true, comple-
tion: nil)
     }
```

Для відображення з можливістю перегортання UIView на яких відображається інформація про курси валют був обраний елемент UIScrollView в якому встановивши значення властивості isPagingEnabled = true, щоб реалізувати дану можливість. Всі елементи лежать на головному UIScrollView який реалізовує функцію «Pull to refresh». Щоб можна було звертатися до всіх цих елементів в коді, треба створити посилання на кожен елемент за допомогою атрибута @IBOutlet:

```
@IBOutlet weak var refreshScrollView: UIScrollView
@IBOutlet weak var bankTitleLabel: UILabel!
@IBOutlet weak var pageControl: UIPageControl!
@IBOutlet weak var scrollView: UIScrollView!
@IBOutlet weak var newsLabel: UILabel!
```

Інтерфейс наступних екранів розроблявся за допомогою xib файлів. При використанні Xib створюється два файли .swift де розміщений код і .xib

– xml файл, де знаходиться верстка. Перевагою такого підходу на багатофункціональних екранах є візуалізація елементів і те, що можна побачити результат роботи з AutoLayout без запуску програми, і якщо якісь обмеження не вписуються в загальну схему, Xcode відразу ж попередить про це. На рисунку 4.2 зображено приклад верстки екрану конвертера валют.

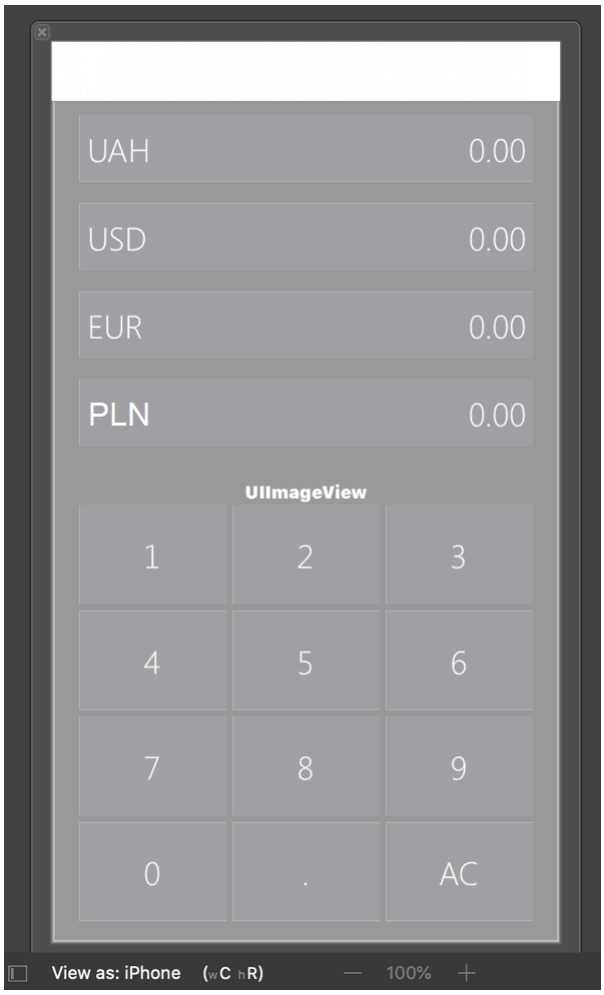

Рисунок 4.2 – Екран конвертера валют за допомогою Xib

Екран списку новин, який передбачено у інтерфейсі додатку для моніторингу актуальної фінансової інформації ринку валют реалізовувався за допомогою зручного елемента UITableView, який був розроблений спеціально для візуалізації списків. Для використання потрібно реалізувати основні ін-

терфейси UITableViewDataSource і UITableViewDelegate. Реалізація UITableViewData-Source складається з обов'язкових методів:

```
     // вказуємо скільки елементів буде відображено в таблиці
     func tableView(_ tableView: UITableView,
                    numberOfRowsInSection section: Int) -> Int {
     return viewModel.numberOfItems
     }
     // вказуємо який осередок буде відображатися і пов'язуємо
його з відповідною viewModel
     func tableView(_ tableView: UITableView,
                          cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
     let cell = tableView.dequeueReusableCell(withIdentifier:
"newsCell",
                               for: indexPath) as!  NewsTableView-
Cell
     if let cellViewModel = self.viewModel.cellViewModel(for:
indexPath) {
          cell.bind(viewModel: cellViewModel)
     }
     return cell
     }
```

Інтерфейс UITableViewDelegate реалізований опціональними методами: даний метод повідомляє, що користувач натиснув на осередок і викликає екран конкретної новини.

```
     func tableView(_ tableView: UITableView,
               didSelectRowAt indexPath: IndexPath) {
              self.presentNewsDetailVC((indexPath as NSIndex-
Path).row)
     }
     //також, так як заголовок кожної новини різний, потрібно
вважати висоту кожного осередку в залежності від контенту.
     func tableView(_ tableView: UITableView,
               heightForRowAt indexPath: IndexPath) -> CGFloat {
              return NewsTableViewCell.getCellHeght ()
     }
```

На рисунку 4.3 наведено приклад вигляду інтерфейсу додатку для моніторингу актуальної фінансової інформації ринку валют на IOS пристрої.

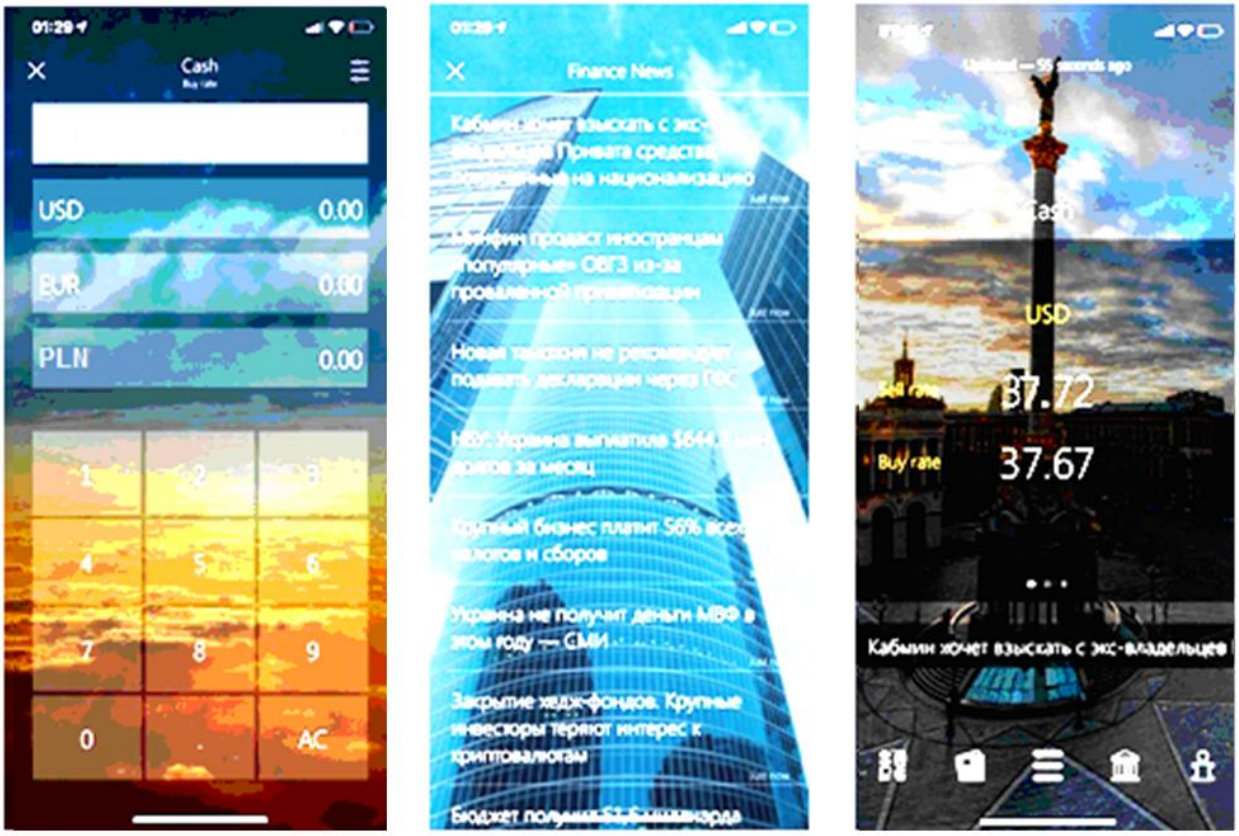

Рисунок 4.3 – Основні екрани інтерфейсу IOS додатку

4.2 Визначення локалізації додатку

Локалізація – це процес адаптації користувальницького інтерфейсу і ресурсів програми для відповідності культурним, мовним і іншим вимогам конкретного цільового ринку.

Xcode надає розробникам досить простий і гнучкий механізм локалізації строкових ресурсів додатку, достатній для реалізації простих завдань локалізації у відносно невеликих проектах.

У проектах зі статичним контентом, строкові дані цілком можна зберігати безпосередньо в інтерфейсі (файлах розмітки .storyboard і .xib, які в свою чергу є XML-файлами візуалізуючими за допомогою Interface Builder) або в коді. Перший підхід дозволяє спростити і прискорити процес розмітки екранів і окремих відображень, тому розробник може спостерігати більшість змін без збірки додатку. Однак в цьому випадку не складно створити надмі-

рності даних (якщо один і той же текст використовується декількома елементами, відображеннями). Другий підхід якраз позбавляє від проблеми надмірності даних, але призводить до необхідності заповнювати екрани вручну (шляхом встановлення додаткових IBOutlet і присвоєння їм відповідних текстових значень), що в свою чергу призводить до надмірності коду (зрозуміло, крім тих випадків, коли текст повинен встановлюватися безпосередньо кодом програми).

Крім цього Apple надає стандарт файлів з розширенням .strings. Даний стандарт регламентує формат зберігання строкових даних у вигляді асоціативного масиву («ключ-значення»): "key" = "value". Ключ чутливий до регістру символів, допускає використання прогалин, підкреслень, знаків пунктуації та спеціальних символів. Важливо зауважити, що незважаючи на легкий синтаксис, Strings-файли є регулярним джерелом помилок на етапі компіляції, збирання або експлуатації додатку. Причин тому декілька:

- по-перше, помилки в синтаксисі. Пропущені крапка з комою, знак рівності, зайві або незаекрановані лапки неминуче приведуть до помилки компілятора. Причому Xcode вкаже на файл з помилкою, але не підсвітить рядок, в якому щось не так. Пошук такої помилки може зайняти значний час, особливо якщо файл містить значний обсяг даних;
- по-друге, дублювання ключів. Додаток через нього, звичайно, не стане непрацездатним, але користувачеві можуть бути відображені некоректні дані. Вся справа в тому, що при зверненні до рядка по ключу, підтягується значення відповідне останньому входженню ключа в файлі.

У підсумку, проста конструкція вимагає від розробника значної скрупульозності і уважності при наповненні файлів даними.

Для роботи з локалізованими текстовими ресурсами фреймворк Foundation надає сімейство методів NSLocalizedStrings:

- NSLocalizedString(_ key: String, comment: String);

– NSLocalizedString(_ key: String, tableName: String?, bundle: Bundle, value: String, comment: String).

Параметр key – ключ рядка в Strings-файле; val (default value) – значення за замовчуванням, яке використовується в разі відсутності зазначеного ключа в файлі; comment – (менш очевидний) короткий опис локалізуючого рядка (по суті не несе в собі корисного функціоналу і призначений для пояснення мети використання конкретного рядка).

tableName (tbl) – це ім'я String-файла, в якому знаходиться необхідний нам рядок за зазначеним ключем; при його передачі розширення .string не вказується. Можливість навігації між таблицями дозволяє не зберігати рядкові ресурси в одному файлі, а розподіляти їх на власний розсуд. Це дозволяє позбутися від перевантаженості файлів, спрощує їх редагування, мінімізує шанс появи помилок. Параметр bundle розширює можливості навігації по ресурсах ще більше, bundle – це механізм доступу до ресурсів додатку, тобто ми можемо самостійно визначати джерело ресурсів. Для того, щоб не нагромаджувати код, було реалізовано розширення класу String у вигляді додавання властивості localized, яке в свою чергу звертається до вище зазначених методів:

```
extension String {
    var localized: String {
        return NSLocalizedString(self, comment: "")
    }
}
```

Як видно на рисунку 4.4 додаток підтримує дві мови: англійська як базова, та українська.

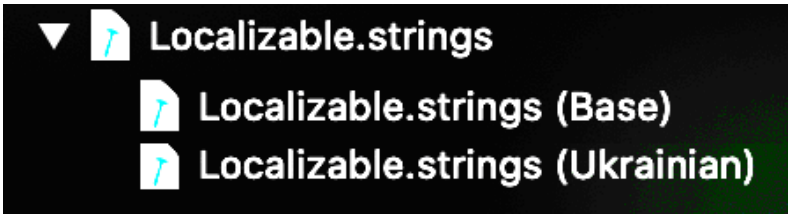

Рисунок 4.4 – Структура файлів локалізації.

Строкові файли локалізації заповнюються у форматі “key” = “value”; Строкові файли локалізації заповнюються у наступному форматі:

```
"Cash" = "Готівковий";
"Average in Banks" = "Середній в Банках";
"Interbank" = "Міжбанк";
"NBU" = "НБУ";
"NO DATA" = "Немає даних";
```

Тепер, щоб використовувати локалізований текст в коді, потрібно просто використовувати властивість localized у потрібний нам рядок:

```
      let title = viewModel.marketTitle(by: 0).localized
           let action = FloatingAction(title:title) {[weak self]
action in
                self?.startUpdate()
           }
```

У даному прикладі дізнаємося назву ринку за індексом, що робить viewModel і потім передаємо параметром в конструктор потрібного нам action. В залежності від мови системи назву ринку буде відображено на відповідній мові. У підсумку можна підкреслити ряд переваг виконання локалізації додатку:

- стандарт зберігання строкових даних в спеціалізованих файлах .string з простим і доступним синтаксисом;
- можливість локалізації інтерфейсу без додаткових маніпуляцій в коді;
- швидкий доступ до локалізованих ресурсів з коду;
- автоматична генерація файлів локалізації і структурування ресурсів директорії додатку засобами Xcode;

Локалізація, є важливим інструментом для просування додатка на багато ринків, а так само для збільшення кількості користувачів в країнах з багатомовним населенням.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра було проведено проектування та здійснено програмну реалізацію мобільного додатку для моніторингу актуальної фінансової інформації ринку валют, що забезпечує зацікавлених користувачів зручним програмним забезпеченням функціонуючим на платформі iOS. Розробка програмного забезпечення для компаній, які ведуть зовнішньоекономічну діяльність та здійснюють розрахунки у іноземній валюті, є актуальним завданням, що забезпечує відстеження швидкої зміни курсів валют та відстеження актуальної фінансової інформації.

В ході роботи були вирішені наступні завдання: здійснено аналіз предметної області, виконана постановка завдання; визначені вимоги до функцій додатку; проведено вибір архітектури, технологій та засобів реалізації додатку; проведено проектування структури, бази даних та інтерфейсів додатку; виконана програмна реалізація мобільного додатку для моніторингу актуальної фінансової інформації ринку валют.

Програмна реалізація додатку здійснена з використанням сучасних засобів розробки: мови програмування Swift у середивищі Xcode, використовуючи сучасне архітектурне рішення MVVM, що забезпечить у подальшому легку модифікацію та внесення змін при розширенні функцій додатку.

Основним призначенням розробленого мобільного додатку для моніторингу актуальної фінансової інформації ринку валют є надання користувачам швидкого та зручного доступу до фінансових новин, а також інформації що до швидкої зміни курсів валют. Додаток також забезпечує користувачів зручним інструментом конвертації будь-яких валют.

Практична цінність розробленого додатка полягає в тому, що застосування такого програмного забезпечення забезпечує користувачів актуальними фінансовими новинами, використовуючи інформацію з фінансового порталу МінФін та може бути використано для доступу к актуальній фінансовій інформації всіма зацікавленими особами та організаціями.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Equity. Як і ким визначається курс валют. Визначення та регулювання. URL: https://equity.today/kak-opredelyaetsya-kurs-valyut.html (дата звернення: 15.03.2024).
2. Мінфін – все про фінанси: новини, курси валют, банки. URL: https://minfin.com.ua/ (дата звернення: 17.04.2024).
3. Криптовалюта – Вікіпедія. URL: https://uk.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B2%D0%B0%D0%BB%D1%8E%D1%82%D0%B0 (дата звернення: 24.04.2024).
4. Model-View-Controller. URL: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html (дата звернення: 16.04.2024).
5. Adam Freeman. Pro Design Patterns in Swift: Apress; 1st ed. Edition 2015. – 592 p.
6. Matt Neuburg. iOS 12 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics 1st Edition. O'Reilly: 2018. – 652p. ISBN 978-1492044550.
7. AppCode: Smart Swift/Objective-C IDE для розробки iOS і macOS. URL: https://www.jetbrains.com/objc/ (дата звернення: 12.04.2024).
8. Stephen G. Kochan. Programming in Objective C. Wiley: 2003. – 556p.
9. Swift – Apple Developer. URL: https://developer.apple.com/swift/ (дата звернення: 15.04.2024).
10. Matt Neuburg. iOS 7 Programming Fundamentals. O'Reilly Media., 2013. – 422 p.
11. Grand Central Dispatch – Wikipedia. URL: https://en.wikipedia.org/wiki/Grand_Central_Dispatch (дата звернення: 02.05.2024).
12. Marcus S. Zarra. Core Data in Swift: Pragmatic Bookshelf, 2016. – 213p.

13. Auto Layout Guide: Understanding Auto Layout. URL: https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutolayoutPG/index.html (дата звернення: 04.05.2024).
14. Craig Clayton Learn iOS 11 Programming with Swift 4 – Second Edition: Packt Publishing; 2nd Revised edition. 2018. – 812 p.