ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ФРАНКА
Кафедра оптоелектроніки та інформаційних технологій

КУРСОВА РОБОТА

на тему: Розробка системи підтримки прийняття рішень. Система управління витратами.

Студента I курсу Феім-13с групи
Напряму підготовки 122 компю'терні науки
Спеціальності системи штучного інтелекту
Агафонова Сергія Ігоровича
(прізвище, ім'я, по-батькові)
Керівник доц. Дуфанець М. В.
_____
(посада, науковий ступінь, прізвище та ініціали)

Національна шкала_____
Кількість балів _____Оцінка ECTS_____
Члени комісії: _____ _____
(підпис) (прізвище, ініціали)
_____ _____
(підпис) (прізвище, ініціали)
_____ _____
(підпис) (прізвище, ініціали)

Львів 2023

Зміст

ВСТУП 3
РОЗДІЛ 1. НАУКОМЕТРІЯ ТА АНАЛІЗ РОСТУ СППР 5
1.1. Огляд публікацій та досліджень у галузі систем підтримки прийняття рішень 5
1.2. Аналіз тенденцій розвитку систем підтримки прийняття рішень 10
РОЗДІЛ 2. ЗАСОБИ РЕАЛІЗАЦІЇ СППР 13
2.1. Визначення вимог до системи 13
2.2. Визначення головних технологій для розробки СППР 15
2.3. Проектування архітектури додатка 19
РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ УПРАВЛІННЯ ВИТРАТАМИ 22
3.1. Опис системи для користувача 22
3.2. Опис бази даних 24
3.3. Інтерфейс мобільного додатку 25
ВИСНОВКИ 29
ДОДАТКИ 31

ВСТУП

Сучасний світ характеризується постійним потоком інформації та швидкими змінами, що вимагають від нас прийняття швидких та обґрунтованих рішень. У багатьох сферах життя, включаючи бізнес, фінанси, управління та особисте планування, ефективне прийняття рішень є ключовим фактором успіху.

Розробка систем підтримки прийняття рішень здатних аналізувати складну інформацію, надавати користувачам рекомендації та підтримувати їх у процесі прийняття рішень, стала актуальною проблемою в галузі комп'ютерних наук. Ці системи поєднують у собі методи та інструменти комп'ютерних наук, статистики, штучного інтелекту та інших дисциплін для створення інформаційних систем, які допомагають ухвалювати обґрунтовані рішення на основі наявних даних [1].

Метою даної курсової роботи є розробка системи підтримки прийняття рішень, зокрема мобільного додатку, який надає користувачам можливість підрахунку та аналізу особистих витрат. Цей додаток стане важливим інструментом для кожної особи, оскільки дозволить ефективно контролювати витрати, розуміти, як розподіляються фінансові ресурси та приймати обґрунтовані рішення щодо планування особистих фінансів.

Відповідно до мети в курсовій роботі послідовно вирішуються такі завдання:

дослідити наукометрію та проаналізувати ріст СППР;

розглянути засоби реалізації СППР;

описати програмну реалізацію системи управління витратами.

Об'єктом дослідження є сама система, її функціональність, архітектура, інтерфейс та його вплив на ефективність прийняття рішень користувачами щодо управління своїми фінансами.

Предметом дослідження є сам процес розробки системи підтримки прийняття рішень та його вплив на управління особистими фінансами користувачів.

При написанні курсової роботи використано традиційні способи і прийоми розробки програмного продукту, проектування, реалізація, тестування та аналіз результатів.

Інформаційною основою курсової роботи є наукові статті та публікації за темою роботи, власні дослідження та інтернет ресурси.

РОЗДІЛ 1. НАУКОМЕТРІЯ ТА АНАЛІЗ РОСТУ СППР

1.1. Огляд публікацій та досліджень у галузі систем підтримки прийняття рішень

Для дослідження було обрано такі галузі як системи прийняття рішень, комп'ютерні науки та штучний інтелект. Відповідно було обрано ключові слова і проведено пошук статей Львівського національного університету імені Івана Франка у пошуковій системі Scival . На основі отриманих даних для кожної з галузей та для всіх разом було сформовано графік кількості нових статей починаючи з 2012 по 2021 роки (рис. 1.1), а також графік приросту кількості статей (рис. 1.2).

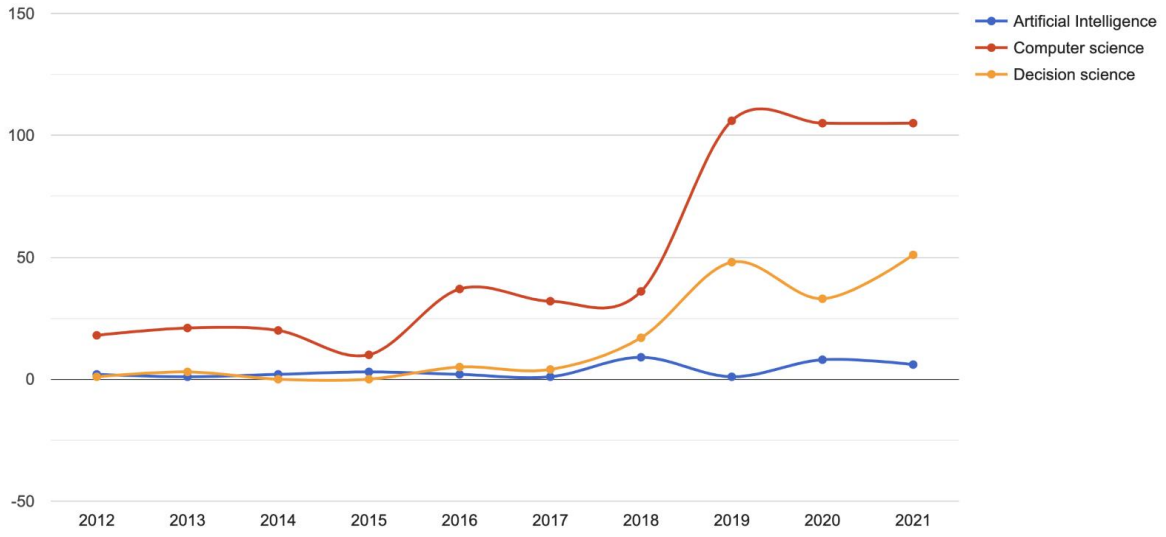

Рис.1.1 Графік кількості нових статей

На наведеному вище графіку показана кількість статей на певний момент часу (оранжевим – галузь систем прийняття рішень, червоним – галузь комп'ютерних наук, синім – галузь штучного інтелекту.)

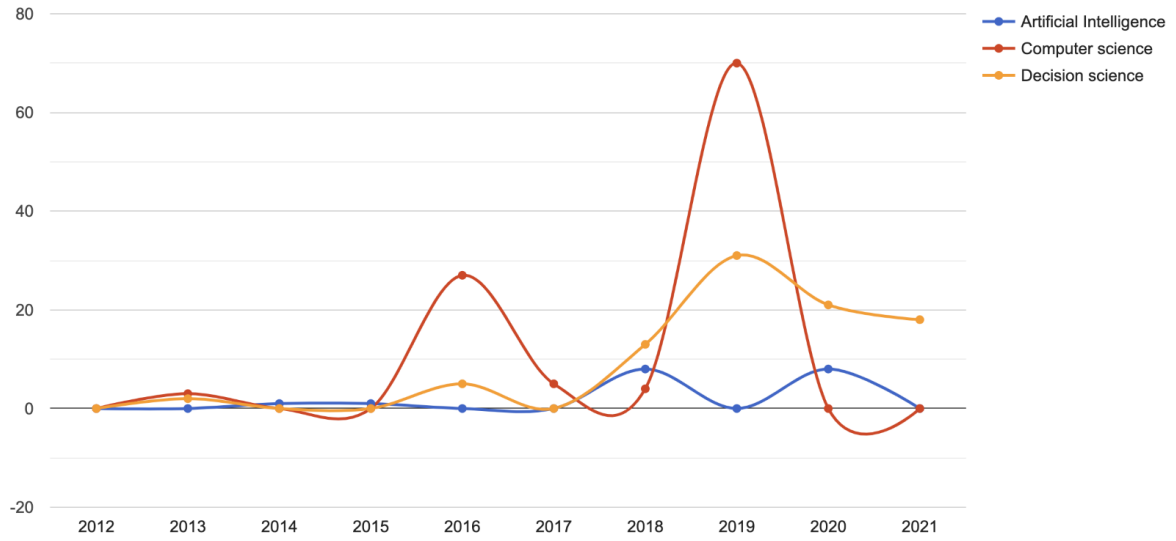

Рис.1.2 Графік приросту кількості статей.

Розглянемо проміжки часу, коли кількість статей зростала рівномірно.

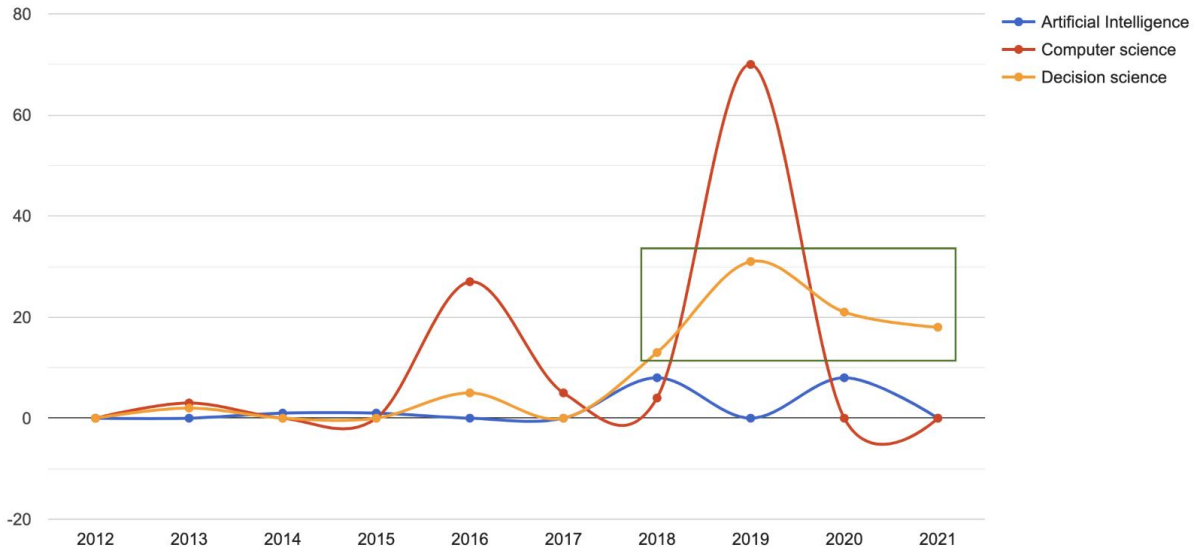

Рис.1.3 Частини рівномірного приросту.

З графіку можемо побачити, що тільки у галузі систем прийняття рішень, був проміжок рівномірного приросту у вигляді 37 статті 2018-2021рр., в той час як всі інші мають переривчасту тенденцію приросту. Таким чином значення константи будуть різні для різних проміжків.

Варто побудувати часову залежність росту числа публікацій та визначити час подвоєння числа публікацій. Розглянемо проміжки часу, коли значення приросту статей зростало.

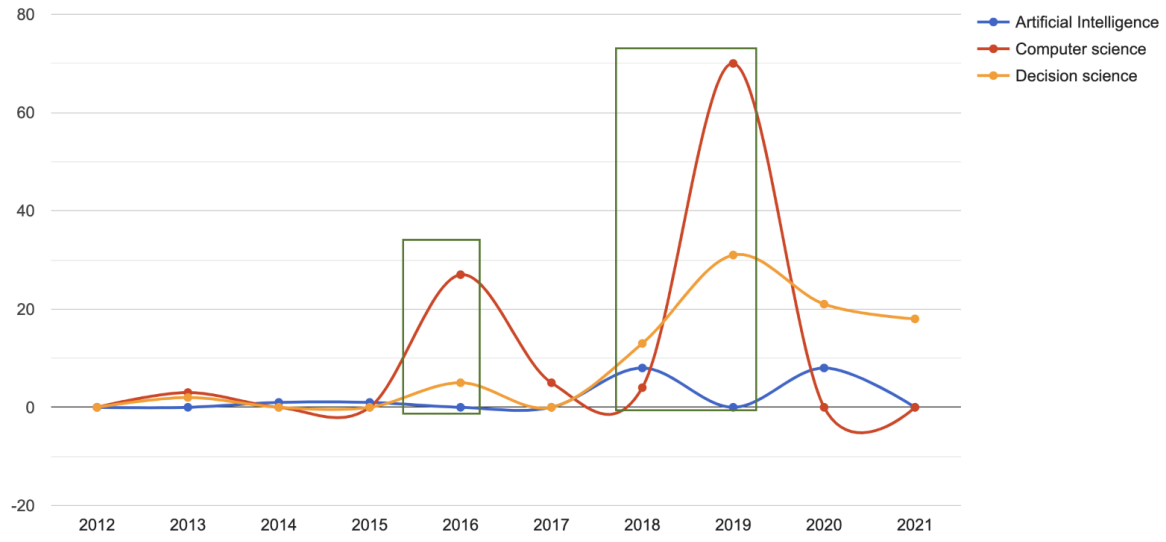

Рис.1.4 Частини стрімкого приросту.

2015 - 2016рр.

2 -- 32 (27 + 5 комп'ютерні науки + системи прийняття рішень відповідно) статті.

k = 15 статті на рік[2]

T = 0.046

2018-2019рр

21 -- 101 (31 + 70 комп'ютерні науки + системи прийняття рішень відповідно) статті .

k = 40 статей на рік[2]

T = 0.0173

Таким чином, було обчислено час подвоєння кількості статей для обох розглянутих ділянок.

Оцінка середнього відгуку на публікації є важливим критерієм для визначення якості та значущості цих публікацій. Відгуки відображають реакцію наукової спільноти на проведені дослідження та розробки в цій галузі.

Значення середнього відгуку полягає у визначенні рівня прийняття та визнання результатів досліджень. Високий середній відгук свідчить про важливість та інноваційність досліджень, а також про їхню потенційну впливовість на розвиток галузі та може сприяти подальшому розширенню знань у галузі. Оцінка середнього відгуку також має практичне значення. Вона може служити орієнтиром для дослідників, що визначають напрямки своїх подальших досліджень, а також для фахівців та практиків, які цікавляться застосуванням систем підтримки прийняття рішень у своїх сферах діяльності.

Константи, яка характеризують середній відгук на публікації, (дані взяти з інформаційного інтернет ресурсу Scival):

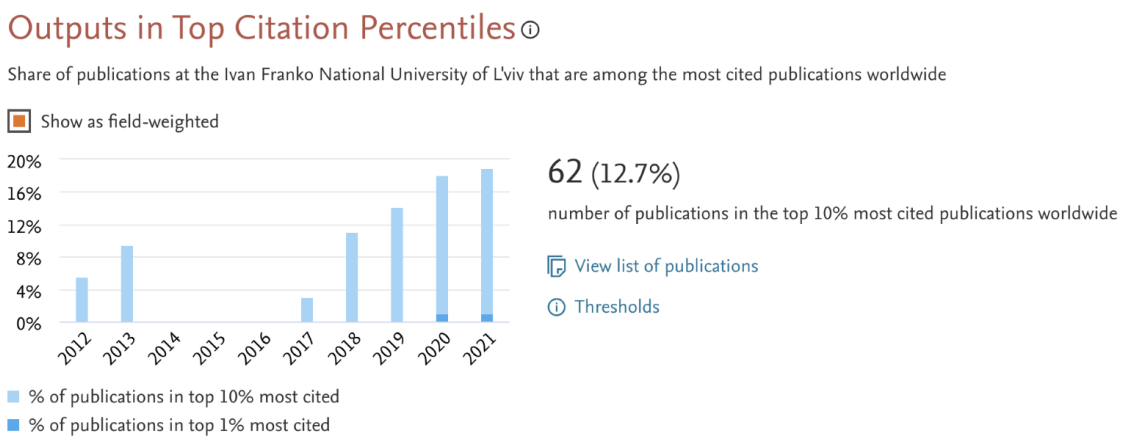

Рис.1.5 Системи підтримки прийняття рішень, c = 62.

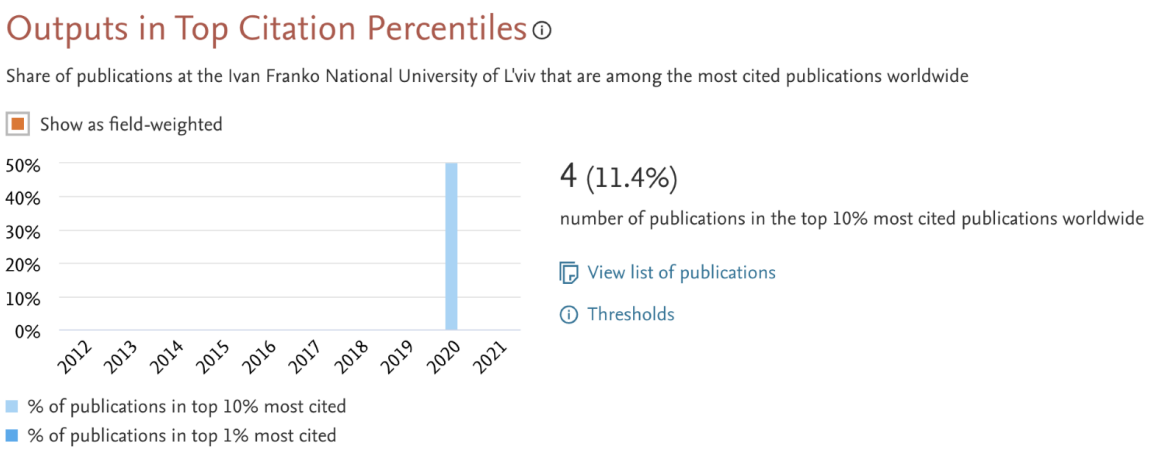

Рис.1.6 Штучний інтелект с = 4.

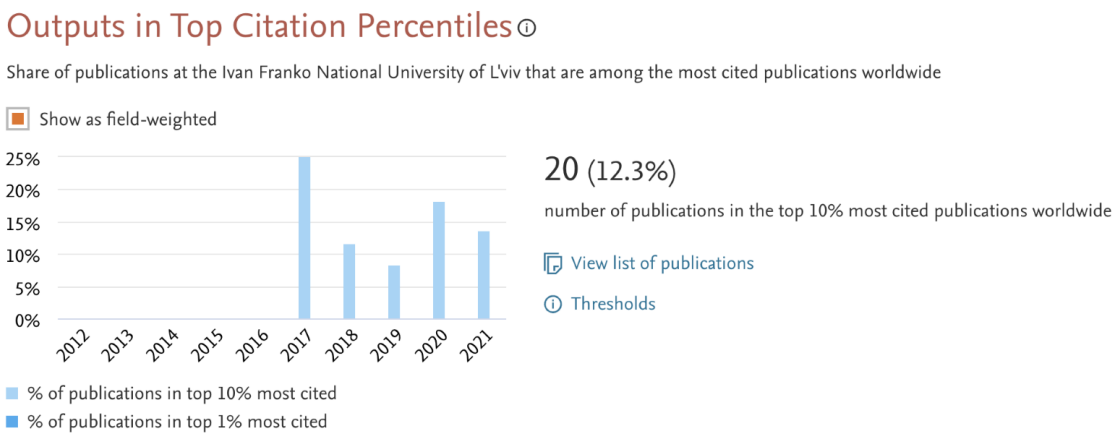

Рис.1.7 Комп'ютерні науки, с = 20.

Огляд публікацій та досліджень у галузі систем підтримки прийняття рішень є важливим для визначення поточного стану цієї галузі, виявлення тенденцій, ідентифікації невирішених проблем та напрямків подальших досліджень. Він допомагає науковій спільноті, фахівцям та практикам вдосконалювати і розвивати системи підтримки прийняття рішень з метою досягнення кращих результатів у прийнятті рішень у різних сферах діяльності.

1.2. Аналіз тенденцій розвитку систем підтримки прийняття рішень

Аналіз тенденцій розвитку систем підтримки прийняття рішень відображає напрямки, у яких ця галузь рухається і розвивається. Основні тенденції, які можна визначити, включають:

1. Застосування штучного інтелекту (ШІ): ШІ стає важливим інструментом у системах підтримки прийняття рішень. Використання методів машинного навчання, нейронних мереж, генетичних алгоритмів та інших технік ШІ дозволяє системам підтримки прийняття рішень робити більш точні прогнози, аналізувати великі обсяги даних та забезпечувати автоматизовану підтримку прийняття рішень.

Найбільш відомими прикладами використання систем штучного інтелекту в прийнятті рішень є:

- системи контекстуальної реклами, що базуються на алгоритмах прецедентного зв'язку та генерується системами інтелекту «на базі подібного» і тим самим збільшує ефективність сповіщень, оскільки кожен реципієнт отримує інформацію, що є цікавою саме йому;
- системи автоматичного розпізнавання та протидії щодо кібервторгнень, що дозволяють співробітникам відділів технологій та служб інформаційної безпеки своєчасно реагувати на ситуації, що вимагають якомога швидшого розв'язання. В окремих випадках такі системи можуть бути повністю автономними та взагалі не вимагати реакції адміністративного технічного персоналу;
- системи прийняття рішень щодо узгодження-неузгодження рішень відносно надання фінансування, що використовуються в банківських та приватних фінансових установах та використовуються для обробки великого обсягу даних (кредитні історії, фактори ризику, обсяги доходів реципієнта);
- пошукові системи, що використовують штучний інтелект для контекстуального пошуку, а також пошуку за подібними запитами. В

тому 68 числі - локальні системи пошуку в умовах великої кількості даних, що використовуються на підприємствах;

- системи безпеки (антивірусне програмне забезпечення), що приймають рішення відносно змісту файлів на основі їх поведінки в синтетичних умовах та самонавчаються на існуючому шкідливому програмному забезпеченні;
- автоматизовані симулятори присутності людини (т.з. «боти»), в тому числі - в комп'ютерних програмах та іграх (наприклад, в шахах), на біржових операціях, а також в інших сферах людської діяльності, що вимагають глибокого аналізу поведінки зовнішньої середи для прийняття правильного рішення [2].

2. Використання аналітики даних: Зростання обсягів даних та доступ до розширених аналітичних інструментів сприяють розвитку систем підтримки прийняття рішень. Аналітика даних дозволяє виявляти патерни, тренди та взаємозв'язки в даних, що сприяє кращому розумінню ситуацій та прийняттю кращих рішень.

3. Інтерактивність та візуалізація даних: Розвиток інтерактивних інтерфейсів та візуалізаційних засобів дозволяє користувачам взаємодіяти з системою підтримки прийняття рішень в більш зручний та зрозумілий спосіб.

Візуалізація сценаріїв та альтернатив допомагає представити можливі варіанти та наслідки рішень у зрозумілій формі. Вона дозволяє візуалізувати різні сценарії розвитку подій та їх вплив на ключові параметри. Це може бути представленням у вигляді дерева рішень, схеми потоку даних або іншої графічної моделі, яка допомагає легше порівнювати альтернативи та визначати оптимальний варіант.

4. Інтеграція з розподіленими та хмарними системами: Зростання значення розподілених та хмарних систем впливає на розвиток систем підтримки прийняття рішень. Інтеграція з цими системами дозволяє

отримувати доступ до розподілених даних, виконувати обчислення у хмарі та сприяє підвищенню ефективності та масштабованості систем підтримки прийняття рішень.

Технології інтеграції: Для забезпечення ефективної взаємодії між системами підтримки прийняття рішень та розподіленими/хмарними системами використовуються різні технології. Наприклад, сервіс-орієнтована архітектура (SOA) дозволяє реалізовувати взаємодію через веб-служби, архітектура мікросервісів дозволяє створювати незалежні компоненти, що можуть бути розгорнуті в різних середовищах. Інші технології, такі як API (Application Programming Interface) та ETL (Extract, Transform, Load), також використовуються для обміну даними та інтеграції систем.

Інтеграція з розподіленими та хмарними системами не є безпроблемною задачею. Виникають виклики щодо безпеки та конфіденційності даних при передачі їх між системами. Також потрібно вирішувати питання сумісності між різними технологіями та стандартами, а також забезпечувати стабільну та надійну взаємодію між системами.

Аналіз тенденцій розвитку систем підтримки прийняття рішень допомагає уявити майбутнє цієї галузі та спрямувати дослідження та розробки у відповідних напрямках. Врахування цих тенденцій дозволяє покращити ефективність, точність та придатність систем підтримки прийняття рішень для різних сфер діяльності [3].

РОЗДІЛ 2. ЗАСОБИ РЕАЛІЗАЦІЇ СППР

2.1. Визначення вимог до системи

Визначення вимог до системи є критичним кроком у процесі розробки системи підтримки прийняття рішень. Вимоги визначають функціональні та нефункціональні характеристики системи, які вона повинна виконувати для задоволення потреб користувачів та досягнення поставлених цілей. Основні етапи визначення вимог включають:

1. Збір і аналіз вимог: Цей етап включає вивчення потреб користувачів та стейкхолдерів системи. Збирання вимог проводиться через спілкування зі зацікавленими сторонами, опитування, інтерв'ю, аналіз існуючих документів та досліджень. Отримані вимоги аналізуються для визначення їх суттєвості, пріоритетності та взаємозв'язку.
2. Формулювання вимог: На основі зібраних вимог формулюються конкретні та вимірювані вимоги до системи. Це можуть бути функціональні вимоги, які описують, що система повинна робити, і нефункціональні вимоги, які описують характеристики системи, такі як надійність, швидкодія, безпека та інтерфейс користувача.
3. Валідація вимог: На цьому етапі перевіряється правильність, повнота та прийнятність вимог. Валідація вимог включає перевірку відповідності вимог реальним потребам користувачів, їх здійсненність та зв'язок з цілями проекту.
4. Документування вимог: Остаточні вимоги фіксуються у вигляді документа, який включає детальний опис кожної вимоги. Цей документ служить основою для подальшого проектування, розробки та тестування системи.

За сучасним станом розвитку система СППР має відповідати таким вимогам [4]:

– використовує слабоструктуровані та нечіткі дані;

– оперує зі слабоструктурованими рішеннями;

– підтримує як взаємозалежні, так і послідовні рішення;

– може застосовувати знання;

– підтримує моделювання та прогнозування;

– може бути легко побудована, якщо може бути сформульована логіка конструкції СППР;

– повинна бути простою у застосуванні та модифікації;

– повинна підтримувати три фази процесу прийняття рішень: інтелектуальну частину, проектування та вибір;

– система призначена для ОПР різного рівня;

– система може бути адаптована до індивідуального та групового застосування; – СППР підтримує різні стилі та методи рішень, що можуть бути корисними при застосуванні групою ОПР;

– система виявляє гнучкість і адаптується до змін в організації та в її оточенні;

– СППР дозволяє людині керувати процесом прийняття рішення за допомогою комп'ютера, а не навпаки;

– система підтримує еволюційне застосування та легко адаптується до мінливих вимог;

– застосування СППР підвищує ефективність процесу прийняття рішень.

Але з даного переліку я б виділив саме 5 основних вимог на котрі в першу чергу потрібно брати до уваги на стадії визначенні вимог.

1. Функціональність: СППР повинні мати широкий спектр функціональних можливостей, що дозволяють виконувати аналіз даних, моделювання сценаріїв, проводити оптимізацію та робити прогнози.

2. Візуалізація та інтерактивність: Системи повинні мати зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє візуалізувати дані, моделі та результати аналізу. Інтерактивність системи дозволяє користувачам

взаємодіяти з даними, вносити зміни, виконувати аналітичні запити та дослідження у реальному часі.

3. Швидкодія: Системи повинні бути швидкими та ефективними в обробці великих обсягів даних. Це важливо для забезпечення оперативності аналізу, прийняття рішень та взаємодії з системою. Оптимізація алгоритмів та використання потужностей обчислювальної техніки допомагають досягти швидкості та ефективності.

4. Надійність та безпека: Системи повинні бути надійними, щоб забезпечити безперебійну роботу та уникнути втрати даних. Забезпечення захисту від несанкціонованого доступу, шифрування даних та інших механізмів безпеки є важливими аспектами для забезпечення конфіденційності та цілісності інформації.

5. Мобільність: З огляду на зростання використання мобільних пристроїв, системи підтримки прийняття рішень повинні бути доступними і адаптованими для роботи на різних платформах та пристроях. Можливість доступу до системи через мобільні додатки або веб-браузери забезпечує гнучкість та зручність в роботі.

Ці вимоги допомагають забезпечити ефективну та успішну роботу систем підтримки прийняття рішень, дозволяючи користувачам аналізувати дані, моделювати сценарії та приймати обґрунтовані рішення для досягнення поставлених цілей [5].

2.2. Визначення головних технологій для розробки СППР

React Native був представлений компанією Facebook у 2015р. що застосовує React архітектуру до нативних IOS, Android та UWP додатків.

Це сучасний фреймворк для написання кроссплатформенних мобільних додатків. Він дозволяє створювати єдиний проєкт під IOS і Android. Facebook, Instagram, Skype, Pinterest, Uber і інші великі компанії використали цей фреймворк для розробки своїх мобільних додатків за

допомогою цієї технології. Дуже величезним плюсом є використання однієї бази ,писати один код, а отримати результат, який буде працювати відразу на двох платформах, iOS і Android. Цей фреймворк швидко розвивається та покращується, підтримується компаніями Facebook та Microsoft. Саме головне, що цей продукт поширюється під ліцензією відкритого програмного забезпечення ,що дозволяє повністю безкоштовно реалізувати свої мобільні нативні додатки.

Для написання додатку було використано редактор коду WebStorm. WebStorm є інтегрованою середовищем розробки (Integrated Development Environment, IDE) для веб-розробки. Це програмний продукт, розроблений компанією JetBrains, який надає широкі можливості для покращення продуктивності та спрощення процесу розробки веб-додатків. WebStorm пропонує розширену підтримку веб-технологій, таких як HTML, CSS, JavaScript і TypeScript. Він включає інтелектуальне автодоповнення коду, підсвічування синтаксису та форматування коду, що допомагає розробникам писати чистий та безпомилковий код. Інтегроване середовище також підтримує популярні веб-фреймворки, такі як React, Angular та Vue.js, надаючи розширену підтримку розробки специфічну для цих фреймворків. Графічний інтерфейс програмного середовища WebStorm можемо побачити на рис. 2.1.

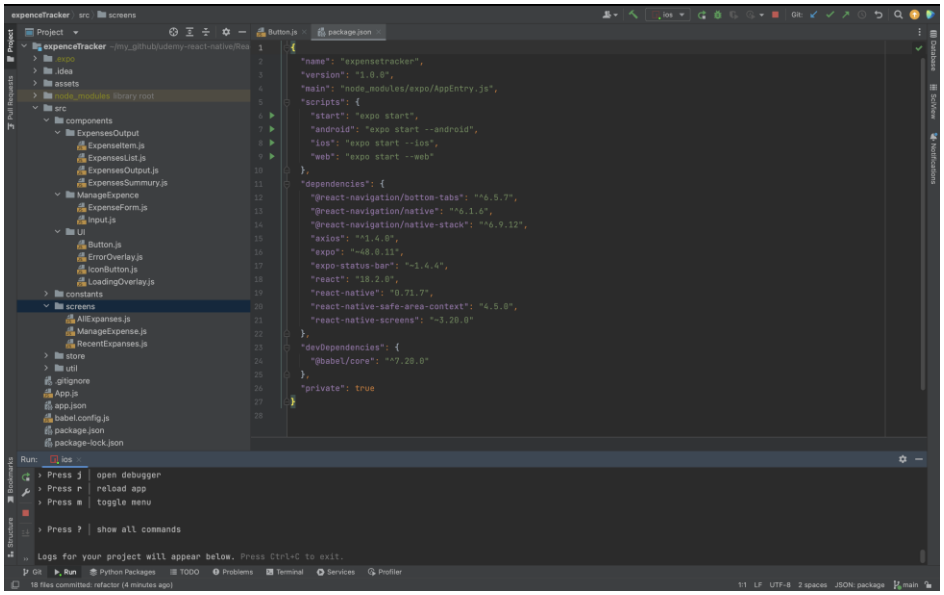

Рис 2.1 – Інтерфейс WebStorm

Редактор WebStorm має дуже багато розширень редактора, які полегшують життя розробника. Під час розробки були використані такі розширення:

- Prettier – полегшує читабельність коду в редакторі, дозволяє вільно писати код та під час зберігання приводить його в нормальний вигляд.

- Expo Tools – при роботі з файлами конфігурацій допомагає автоматично підтягнути потрібний пакет під час початку введення імені пакету.

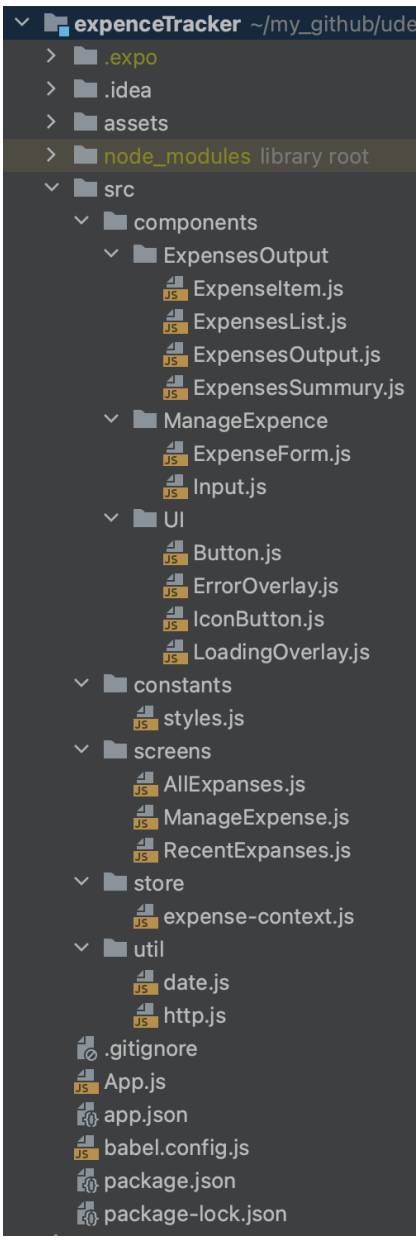

Рис 2.2 Вигляд структури додатку в редакторі коду WebStorm

2.3. Проектування архітектури додатка

В новій архітектурі React Native виконується поступова відмова від функцій моста, на зміну якій приходить новий механізм, званий JavaScript Interface (JSI).Застосування JSI відкриває нові можливості для деяких чудових поліпшень в розробці.

Перше таке поліпшення полягає в тому, що JS-бандл більше не покладається на JSC. Іншими словами, двигун JSC тепер легко можна замінити на щось інше, цілком можливо, що відрізняється більш високою продуктивність.

Друге поліпшення - це те, що лежить в основі нової архітектури React Native. Воно полягає в тому, що, завдяки використанню JSI, в JavaScript можна зберігати посилання на C ++ - об'єкти (Host Objects) і викликати методи цих об'єктів. В результаті JavaScript-частина програми та нативні механізми будуть знати один про одного набагато більше, ніж раніше.[6]

На рис. 2.3 ми можемо побачити структуру нової архітектури React Native

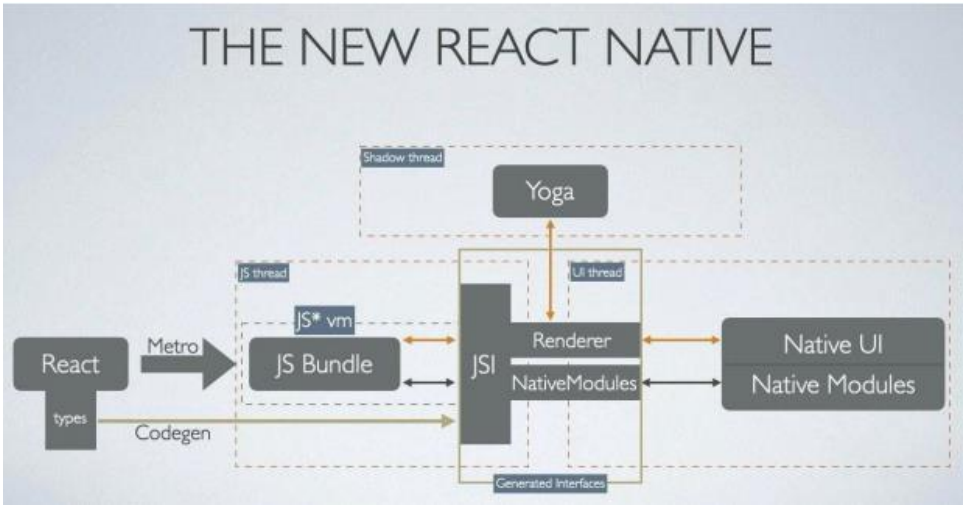

Рис. 2.3 – Нова архітектура React Native

Іншими словами, JSI дозволяє забезпечити повну взаємодію між всіма потоками. Завдяки використанню концепції спільного володіння (shared

ownership), JavaScript-код може запускати нативні методи безпосередньо з JS-потоку. При цьому немає необхідності в тому, щоб використовувати JSON для передачі даних. Це дозволяє позбутися від проблем, характерних для використання моста, пов'язаних з можливим переповненням черги і з асинхронною передачею даних.

Збереження даних користувача додатка відбувається на платформі FireBase. Це дає в режимі реального часу базу даних. Ця служба надає розробникам мобільних чи веб-додатків API, який дозволяє синхронізувати дані застосунків між клієнтами та зберігати їх у хмарному середовищі Firebase в виді таблиць. Приклад списку даних можна побачити на рис. 2.4.

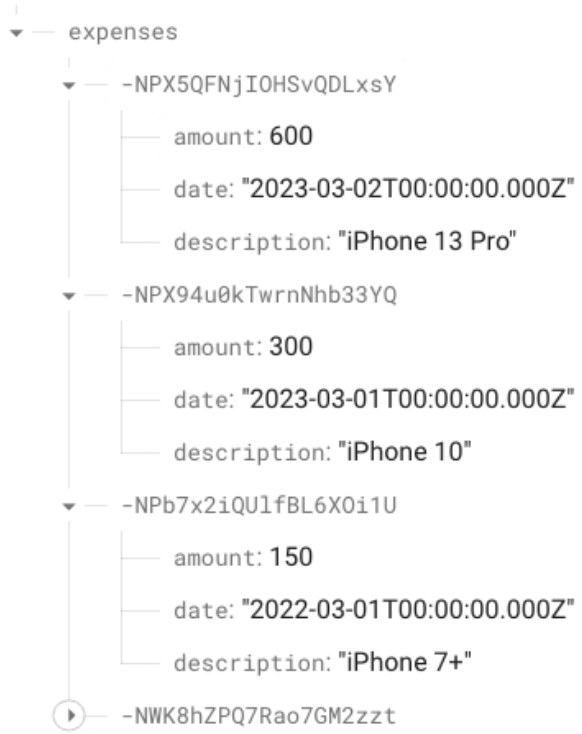

Рис 2.4 – Приклад об’єкту бази даних Realtime Database

Компанія надає всі потрібні клієнтські бібліотеки, які дозволяють інтегрувати Firebase в додатки на Android, iOS, JavaScript / Node.js, Java, Objective-C, Swift. База даних також доступна через REST API та прив'язки

до декількох сценаріїв JavaScript, таких як AngularJS, React, Ember.js та Backbone.js.

В розробці мобільного додатку було використано набір інструментів Expo тому, що він має переваги при розробці.

Основні переваги:

- Expo розширює можливості роботи з API React Native Звичайний React Native не дає вам усіх можливостей при роботі з API в JS, а лише примітивні функції. В той час, коли Expo забезпечує всіми можливими для роботи з API та навіть більше.[7]

- Expo надає змогу оновлювати JS код в режимі реального часу, що набагато пришвидшує розробку додатків.

- Клієнт Expo надає можливість використовувати будь -які додатки сумісні з ним.

- Надає можливість зібрати фінальний результат в працюючий додаток однією командою в терміналі.

РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ УПРАВЛІННЯ ВИТРАТАМИ

Проаналізувавши поставлену задачу, почнемо розробку нашого додатку. Перше з чого ми почнемо - це буде головна панель управління користувача витратами та їх збереження, так як вона служить елементом, який наповнює нашу базу даних FireBase, звідки дані будуть записуватись та виводитися на додаток мобільного телефону.

3.1. Опис системи для користувача

UI частина реалізована на React Native, що дозволило нам використати вже готові деякі залежності, які скоротили час розробки. Використання залежностей - це хороша практика, адже вони не несуть ніякої шкоди, а також є повністю безкоштовними. Залежності інсталюються за допомогою команд npm менеджера. Основними використані залежності є : "react-native-svg-charts": "^5.4.0", "react-native": "^0.71.7".

Всі використані залежності:

```
"dependencies": {
  "@react-navigation/bottom-tabs": "^6.5.7",
  "@react-navigation/native": "^6.1.6",
  "@react-navigation/native-stack": "^6.9.12",
  "axios": "^1.4.0",
  "expo": "~48.0.11",
  "expo-status-bar": "~1.4.4",
  "react": "18.2.0",
  "react-native": "0.71.7",
  "react-native-safe-area-context": "4.5.0",
  "react-native-screens": "~3.20.0",
  "react-native-svg-charts": "^5.4.0"
},
```

Рис 3.1. – Залежності

Перейдемо до функцій ,які може виконувати користувач:

- має можливість бачити всі раніше додані витрати, у вигляді списку для зручності;

- можливість детально переглядати окремо додані витрати;

- можливість редагувати окремі витрати, та зберігати зміни;

- змога повністю видалити витрати зі списку і тим самим з бази даних.

На даний момент функцій буде достатньо для управління витратами. Але в майбутньому для покращення зручності можна доповнювати різними цікавими функціями.

На рис. 3.2 ми можемо бачити схему системи управління витратами.

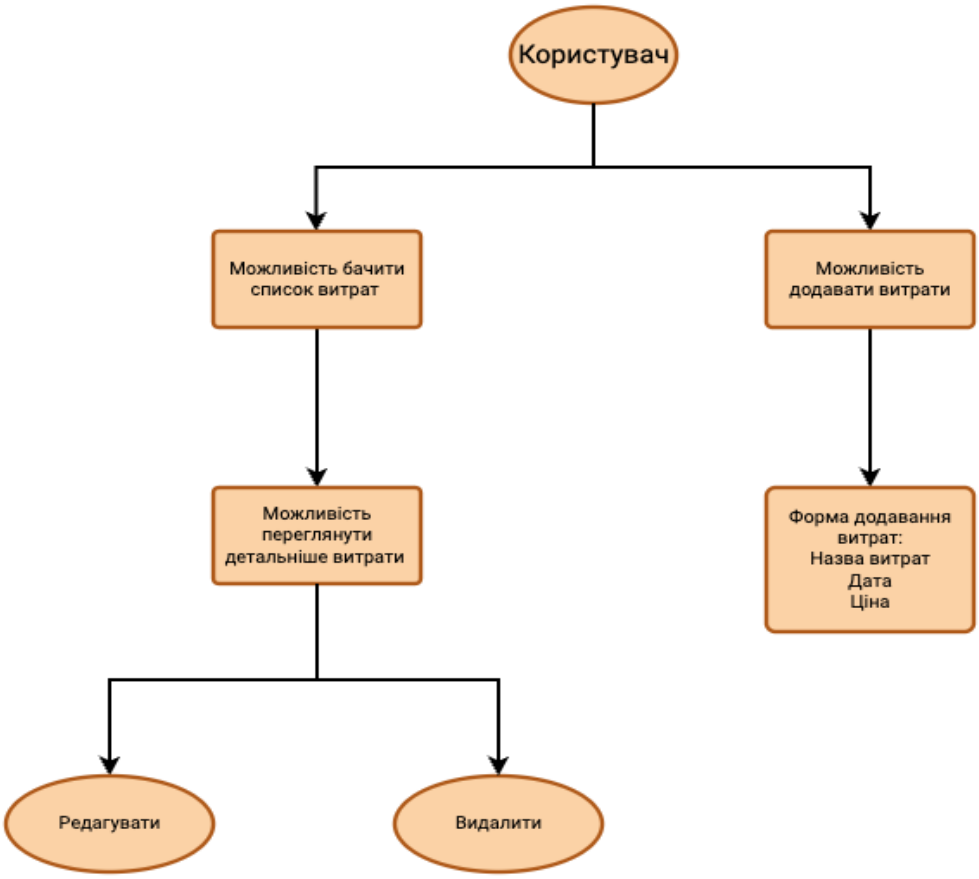

Рис 3.2 – Схема системи управління витратами

3.2. Опис бази даних

Firebase Realtime Database - це хмарна база даних, яка пропонує реальний час синхронізації даних між сервером та клієнтськими пристроями. Вона є частиною платформи Firebase від Google і спеціально призначена для розробки мобільних та веб-додатків.

За сутність Firebase Realtime Database для збереження даних в об'єктах витрат мобільного додатку включає наступні переваги:

1. Реальний час оновлення: Realtime Database надає миттєву синхронізацію даних між різними пристроями та користувачами. Це означає, що якщо один користувач змінює дані в базі даних, зміни автоматично оновлюються на всіх інших підключених пристроях.
2. Простота використання: Firebase Realtime Database має простий та зрозумілий API, який дозволяє легко взаємодіяти з базою даних, рис. 3.3. Ви можете додавати, оновлювати та видаляти дані за допомогою кількох рядків коду.

```
const BACKEND_URL : string  = 'https://expense-app-ffb19-default-rtdb.firebaseio.com/';

2 usages  serhii.ahafonov +1
export async function storeExpense(expenseData) : Promise<any>  {
    const response : AxiosResponse<any>  = await axios.post(
        url: `${BACKEND_URL}/expenses.json`,
        expenseData
    );

    const id = response.data.name;
    return id;
}
```

Рис 3.3 – Приклад використання Firebase API

3. Масштабованість: Firebase Realtime Database автоматично масштабується для вам, забезпечуючи швидку та надійну роботу з даними, навіть якщо ваш додаток отримує велику кількість запитів.

4. Зручність роботи з об'єктами: Ви можете зберігати дані у вигляді об'єктів JSON. У вашому випадку, об'єкти витрат можуть містити необхідну структуру даних, наприклад, поля "назва", "сума", "дата" тощо.
5. Рівень доступу та аутентифікація: Firebase Realtime Database дозволяє налаштувати права доступу до даних, визначаючи, хто може читати або записувати в базу даних. Ви можете обмежувати доступ лише для авторизованих користувачів, використовуючи механізми аутентифікації Firebase.
6. Інтеграція з іншими сервісами Firebase: Firebase пропонує інші корисні сервіси, такі як аутентифікація користувачів, зберігання файлів, аналітика та багато іншого. Це дозволяє вам легко розширити функціональність вашого додатку за допомогою інших сервісів Firebase [8].

3.3. Інтерфейс мобільного додатку

Перейдемо до розгляду мобільного додатку. При вході в мобільний додаток бачимо головний екран додатку, який можна бачити на рис. 3.4. На екрані відкрита дефолтна вкладка витрат за останні сім днів. Внизу екрана можна побачити панель вкладок, які насамперед мають активний стан.

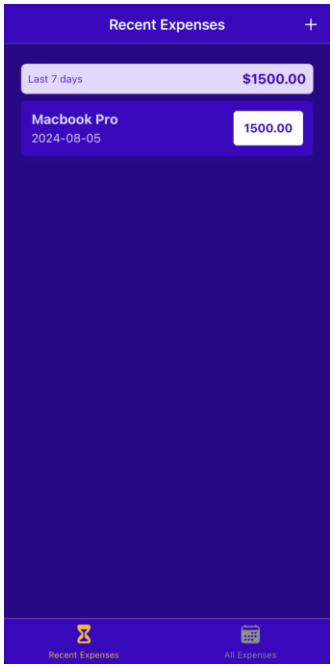

Рис 3.4 – Головний екран мобільного додатку, “Recent Expenses”

Перейшовши на вкладу “All expenses” можемо побачити всі витрати які були здійснені користувачем, рис. 3.5.

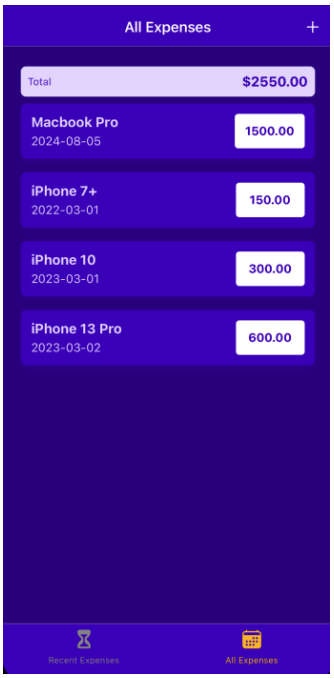

Рис 3.5 – Головний екран мобільного додатку, “All Expenses”

В кожного елемента списку витрат присутня назва, ціна і дата здійснення витрати, рис. 3.6.

Рис 3.6 – Елемент витрати

Користувач додатку має можливість додавати, редагувати та видаляти витрати. Розглянемо детальніше кожен з варіантів рис. 3.7 та рис. 3.8.

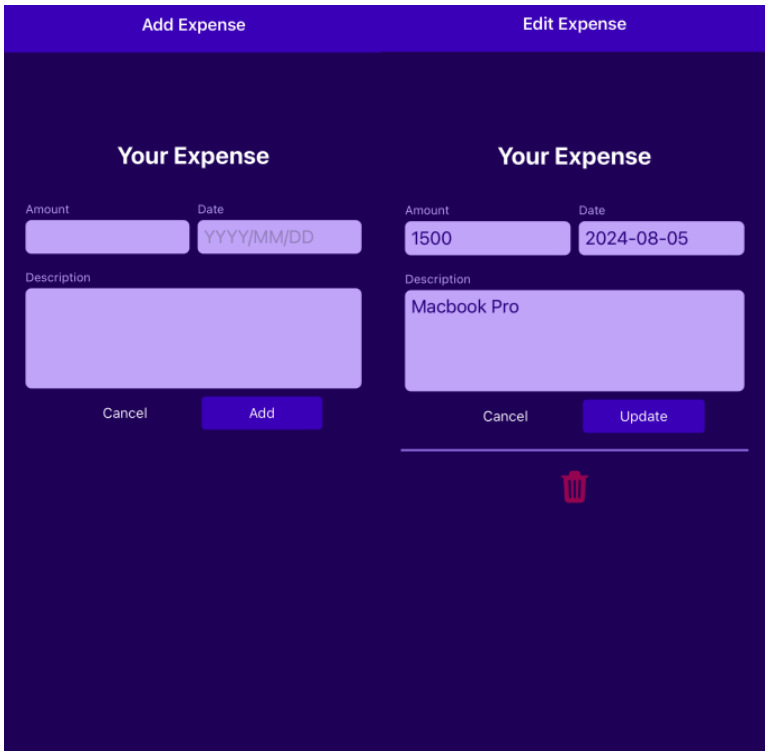

Рис. 3.7, 3.8 – Модальні вікна “Add expense” та “Edit expense”

Також користувач може переглянути кругову діаграму для відображення часток або пропорцій його витрат в цілому, рис. 3.9.

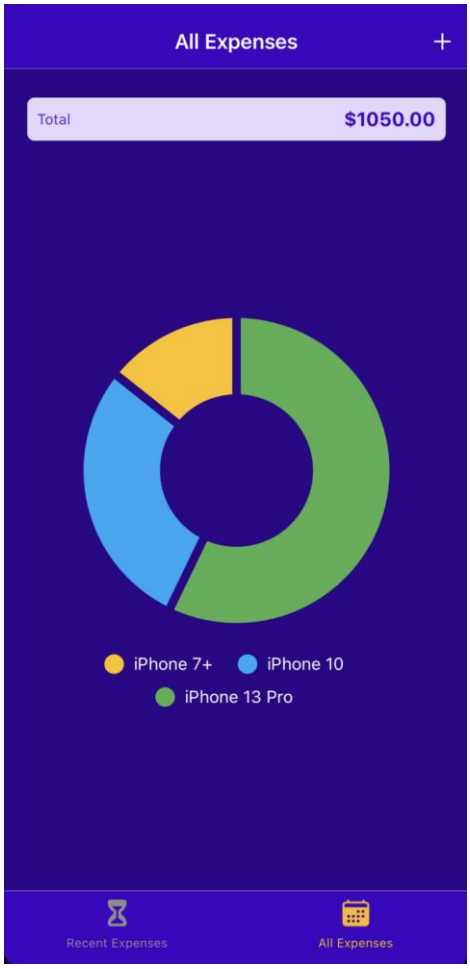

Рис. 3.9 – Кругова діаграма витрат

ВИСНОВКИ

У даній курсовій роботі досягнуто поставленої мети і вирішено всі необхідні завдання. За підсумками зробленого необхідно зробити кілька висновків, і узагальнень, а саме:

досліджено наукометрію та проаналізувати ріст СППР;

розглянуто засоби реалізації СППР;

описано програмну реалізацію системи управління витратами.

Для представлення даних було розроблено мобільний додаток на платформі React Native з використанням бази даних Firebase для ведення обліку витрат. Firebase Realtime Database було обрано як засіб для збереження даних про витрати в об'єктах витрат. Це рішення має декілька переваг, зокрема реальний час оновлення даних, простоту використання та масштабованість бази даних Firebase.

Як підсумок в курсовій роботі було вирішено поставлену задачу, а також вирішено всі проблеми, які виникли під час виконання завдання.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Інтелектуальні системи підтримки прийняття рішень [електронний ресурс]. – Режим доступу: http://nam.kiev.ua/files/publications/978-966-8406-94-2-(1-13).pdf
2. ВИКОРИСТАННЯ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ПРИЙНЯТТЯ УПРАВЛІНСЬКИХ РІШЕНЬ [електронний ресурс]. – Режим доступу: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/da94f422-dc27-4b78-8be1-a046cd06b3ef/content
3. СИСТЕМИ І МЕТОДИ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ / П. І. Бідюк, О. Л. Тимощук, А. Є. Коваленко, Л. О. Коршевнюк, 2022, КПІ ім. Ігоря Сікорського, 2022
4. Holsapple C.W. Decision Support Systems (a knowledge based approach) / Holsapple C.W., Whinston A.B. – New York: West Publishing Company, 2013. – 860 p.
5. МАТЕМАТИЧНІ МОДЕЛІ РИЗИКУ ДЛЯ СИСТЕМ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ / В.Я. Данилов, О.В. Мороз, 2003, УНІВЕРСУМ-Вінниця
6. Книга - Fullstack React Native - https://coursehunter.net/book/kniga-fullstack-react-native-polnoe-rukovodstvo-po-react-native
7. База Expo.io [електронний ресурс] - https://docs.expo.io/introduction/walkthrough/
8. База документації Firebase [електронний ресурс] - https://firebase.google.com/docs

ДОДАТКИ

Файл ManageExpense.js

```
import { useState, useContext, useLayoutEffect } from "react";
import IconButton from "../components/UI/IconButton";
import { GlobalStyles } from "../constants/styles";
import { View, StyleSheet } from "react-native";
import { ExpenseContext } from "../store/expense-context";
import { storeExpense, updateExpense, deleteExpense } from "../util/http";
import LoadingOverlay from "../components/UI/LoadingOverlay";
import ErrorOverlay from "../components/UI/ErrorOverlay";
import ExpenseForm from "../components/ManageExpence/ExpenseForm";

function ManageExpense({route, navigation}) {
 const [isSubmitting, setIsSubmitting] = useState(false);
 const [error, setError] = useState(); // [error, setError
 const expensesContext = useContext(ExpenseContext);
 const editedExpenseId = route.params?.expenseId;
 const isEditing = !!editedExpenseId;

 const selectedExpense = expensesContext.expenses.find(expense => expense.id === editedExpenseId);

useLayoutEffect(() => {
	navigation.setOptions({ title: isEditing ? 'Edit Expense' : 'Add Expense',});
}, [navigation, isEditing])

async function deleteExpenseHandler() {
	setIsSubmitting(true);
	try {
		await deleteExpense(editedExpenseId);
		expensesContext.deleteExpense(editedExpenseId);

		 navigation.goBack();
	} catch (error) {
		setError('Could not delete expense');
		setIsSubmitting(false);
	}
}

function cancelHandler() {
	navigation.goBack();
}

async function confirmHandler(expenseData) {
	setIsSubmitting(true);
	try {
		if (isEditing) {
			expensesContext.updateExpense(editedExpenseId, expenseData);
			await updateExpense(editedExpenseId, expenseData);
		} else {
			const id = await storeExpense(expenseData);
			expensesContext.addExpense({...expenseData, id});
		}
	navigation.goBack();
```

```
        }                    catch                    (error)                    {
                setError('Could not save expense, please try again');
                setIsSubmitting(false);
        }
}

function                    errorHandler()                    {
        setError(null);
}

if                    (error                    &&                    !isSubmitting)                    {
        return                <ErrorOverlay                message={error}                onConfirm={errorHandler}                />
}

if                    (isSubmitting)                    {
        return                    <LoadingOverlay                    />
}


return                    <View                    style={styles.container}>
        <ExpenseForm
                onSubmit={confirmHandler}
                onCancel={cancelHandler}
                submitButtonLabel={isEditing                ?                'Update'                :                'Add'}
                defaultValues={selectedExpense}                    />
        {isEditing &&

        <View                    style={styles.deleteContainer}>
                <IconButton
                        icon="trash"
                        color={GlobalStyles.colors.error500}
                        size={36}
                        onPress={deleteExpenseHandler}                    />
        </View>}
</View>
}

const                    styles                    =                    StyleSheet.create({
 container:                    {
 flex:                    1,
 padding:                    24,
 backgroundColor:                    GlobalStyles.colors.primary800,
 },
 deleteContainer:                    {
 marginTop:                    16,
 padding:                    8,
 borderTopWidth:                    2,
 borderTopColor:                    GlobalStyles.colors.primary200,
 alignItems:                    'center',
 }
});

export default ManageExpense;
```

Файл AllExpanses.js

```
import                    {                    useContext                    }                    from                    "react";
import                    ExpensesOutput                    from                    "../components/ExpensesOutput/ExpensesOutput";
```

```
import { ExpenseContext } from "../store/expense-context";

function AllExpenses() {
 const expensesCtx = useContext(ExpenseContext);
 return <ExpensesOutput expenses={expensesCtx.expenses} expensesPeriod="Total" fallbackText="No Registered
Expenses Found" />
}

export default AllExpenses;
```

Файл RecentExpanses.js

```
import ExpensesOutput from "../components/ExpensesOutput/ExpensesOutput";
import { ExpenseContext } from "../store/expense-context";
import { useContext, useEffect, useState } from "react";
import { getDateMinusDays } from "../util/date";
import { fetchExpenses } from "../util/http";
import LoadingOverlay from "../components/UI/LoadingOverlay";
import ErrorOverlay from "../components/UI/ErrorOverlay";

function RecentExpenses() {
 const [isFetching, setIsFetching] = useState(true);
 const [error, setError] = useState(null); // [error, setError
 const expensesCtx = useContext(ExpenseContext);

 useEffect(() => {
	async function getExpenses() {
		setIsFetching(true);

		try {
			const expenses = await fetchExpenses();
			expensesCtx.setExpenses(expenses);
		} catch (error) {
			setError('Cound not fetch expenses');
		}

	setIsFetching(false);
	}

	getExpenses();
 }, []);

 function errorHandler() {
	setError(null);
 }

 if (error && !isFetching) {
	return <ErrorOverlay message={error} onConfirm={errorHandler} />
 }

 if (isFetching) {
	return <LoadingOverlay />
 }
```

```
const recentExpenses = expensesCtx.expenses.filter(expense => {
    const today = new Date();
    const date7DaysAgo = getDateMinusDays(today, 70);

r    eturn expense.date > date7DaysAgo;
});
return <ExpensesOutput expenses={recentExpenses} expensesPeriod="Last 7 days" fallbackText="No Expenses
Registered for the last 7 days" />
}

export default RecentExpenses;
```

Файл PieChart.js

```
import React from 'react';
import { View, StyleSheet, Text } from 'react-native';
import { PieChart } from 'react-native-svg-charts';

const PieChartComponent = ({ data }) => {
const colors = ['#FFC107', '#03A9F4', '#4CAF50', '#FF5722'];

// Calculate the total sum of amounts
const totalAmount = data.reduce((sum, item) => sum + item.amount, 0);

// Create an array of pie chart data
const pieChartData = data.map((item, index) => ({
    value: item.amount,
    svg: { fill: colors[index % colors.length] },
    key: `pie_${index}`,
}));

return (
<View style={styles.container}>
    <PieChart style={styles.chart} data={pieChartData} />
    <View style={styles.legendContainer}>
        {data.map((item, index) => (
        <View key={`legend_${index}`} style={styles.legendItem}>
        <View style={[styles.legendColor, { backgroundColor: colors[index % colors.length] }]} />
        <Text style={styles.legendText}>{item.description}</Text>
        </View>
        ))}
    </View>
</View>
);
};

const styles = StyleSheet.create({
container: {
flex: 1,
alignItems: 'center',
justifyContent: 'center',
},
chart: {
height: 250,
width: 250,
```

```
},
legendContainer: {
marginTop: 20,
flexDirection: 'row',
flexWrap: 'wrap',
alignItems: 'center',
justifyContent: 'center',
},
legendItem: {
flexDirection: 'row',
alignItems: 'center',
marginRight: 20,
marginBottom: 10,
},
legendColor: {
width: 16,
height: 16,
borderRadius: 8,
marginRight: 8,
},
legendText: {
fontSize: 14,
color: 'white',
},
});

export default PieChartComponent;
```

Файл App.js

```
import { StatusBar } from 'expo-status-bar';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import ManageExpense from './src/screens/ManageExpense';
import RecentExpenses from './src/screens/RecentExpanses';
import AllExpances from './src/screens/AllExpanses';
import { GlobalStyles } from './src/constants/styles';
import {Ionicons} from '@expo/vector-icons';
import IconButton from './src/components/UI/IconButton';
import ExpensesContextProvider from './src/store/expense-context';

const Stack = createNativeStackNavigator();
const BottomTabs = createBottomTabNavigator();

function ExpancesOverview() {
  return <BottomTabs.Navigator screenOptions={({navigation}) => {
    return {
      headerStyle: {
      backgroundColor: GlobalStyles.colors.primary500,
    },
    headerTintColor: 'white',
    tabBarStyle: {
      backgroundColor: GlobalStyles.colors.primary500,
    },
```

```
                tabBarActiveTintColor:                                    GlobalStyles.colors.accent500,
                headerRight: ({tintColor}) => <IconButton icon="add" size={24} color={tintColor} onPress={() =>
{                                navigation.navigate('ManageExpense')}}                                    />,}
                }}>
        <BottomTabs.Screen        name="RecentExpenses"        component={RecentExpenses}        options={{
                title:                                    'Recent                                    Expenses',
                tabBarLabel:                                    'Recent                                    Expenses',
                tabBarIcon:                    ({color,                    size})                    =>                    {
                return        <Ionicons        name="hourglass"        size={size}        color={color}        />
                }
        }}                                                                                                    />
        <BottomTabs.Screen        name="All        Expenses"        component={AllExpances}        options={{
                title:                                    'All                                    Expenses',
                tabBarLabel:                                    'All                                    Expenses',
                tabBarIcon:                    ({color,                    size})                    =>                    {
                return        <Ionicons        name="calendar"        size={size}        color={color}        />
                }
        }}/>
        </BottomTabs.Navigator>
}

export                default                function                App()                    {
 return                                                                                                    (
        <>
                <StatusBar                                    style="light"                                    />
                <ExpensesContextProvider>
                <NavigationContainer>
                <Stack.Navigator                                                                    screenOptions={{
                        headerStyle:                                                                        {
                        backgroundColor:                                    GlobalStyles.colors.primary500,
                },
                headerTintColor:                                                                    'white',
                }}>
                <Stack.Screen    name="ExpancesOverview"    component={ExpancesOverview}    options={{
                        headerShown:                                                                    false
                }}                                                                                        />
                <Stack.Screen        name="ManageExpense"        component={ManageExpense}        options={{
                        presentation:                                                                    'modal',
                }}                                                                                        />
        </Stack.Navigator>
        </NavigationContainer>
        </ExpensesContextProvider>
 </>
 );
}
```