Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:

Моделювання архітектури програмного продукту інтелектуальної системи «Фінансист»

Студентки 2м курсу, 2з групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Франчук Тамари Михайлівни

Науковий керівник доктор технічних наук, професор, завідувач кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Криворучко Олена Володимирівна

Гарант освітньої програми доктор економічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Токар Володимир Володимирович

КИЇВ – 2021

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«29» грудня 2020 р.

Завдання
на випускний кваліфікаційний проєкт студентові

Франчук Тамари Михайлівни
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проєкту Моделювання архітектури програмного продукту інтелектуальної системи «Фінансист»

Затверджена наказом ректора від «28» грудня 2020 р. № 3923

2. Строк здачі студентом закінченого проєкту 25 листопада 2021р.

3. Цільова установка та вихідні дані до проєкту

Мета дослідження облік та прогнозування процесів ведення особистого бюджету за допомогою інтелектуально-інформаційної системи, яка завдяки інтеграції у месенджери, має можливість запускатись на будь-якій сучасній платформі.

Об'єкт дослідження: процеси моделювання архітектури програмного забезпечення та когнітивні технології розпізнання мови за для фіксації витрат та доходів користувача у месенджері.

Предмет дослідження: розробка інтелектуально-інформаційної системи, що використовує когнітивні технології для розпізнання команд користувача.

4. Консультанти проєкту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проєкту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. КОНТРОЛЬ ВЛАСНИХ ФІНАНСІВ – СКЛАДНА ІНТЕЛЕКТУАЛЬНА ІНФОРМАЦІЙНА СИСТЕМА

1.1. Аналіз предметної області інтелектуальної інформаційної системи

1.2. Висновок до розділу 1

РОЗДІЛ 2. ОБГРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТАРІЮ ТА МОВИ UML ДЛЯ МОДЕЛЮВАННІ ПРОГРАМНОГО ПРОДУКТУ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ «ФІНАНСИСТ» 11

2.1. Огляд існуючого програмного забезпечення для керуванням власним бюджетом

2.2. Вибір процесу проектування програмного продукту інтелектуальної системи «Фінансист»

2.3. Мова UML в моделюванні програмних продуктів

2.4. Створення бази даних інтелектуальної системи «Фінансист»

2.5. Побудова UML-діаграми класів програмного продукту інтелектуальної системи «Фінансист»

2.6. Висновок до розділу 2

РОЗДІЛ 3. ПРОЦЕСИ МОДЕЛЮВАННЯ ТА ВПРОВАДЖЕННЯ АРХІТЕКТУРИ ПРОГРАМНОГО ПРОДУКТУ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ «ФІНАНСИСТ»

3.1. Моделювання архітектури інтелектуальної системи «Фінансист»

3.2. Вибір мови програмування та середовища розробки

3.3. Вибір СУБД та опис її фізичної моделі

3.4. Реалізація основних класів та методів

3.5. Розгортання програмної системи та системні вимоги

3.7. Опис типових схем використання системи

3.8. Огляд та реєстрація когнітивних сервісів LUIS

3.9. Висновок до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ТЕХНІЧНЕ ЗАВДАННЯ

ДОДАТКИ

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційногопроєкту*	*21.09.2020*	*21.09.2020*
2.	*Розробка та затвердження завдання на проєкт магістра*	*29.12.2020*	*29.12.2020*
3.	*Вступ та перелік літературних джерел*	*27.02.2021*	*27.02.2021*
4.	*Розробка технічного завдання*	*20.03.2021*	*20.03.2021*
5.	*Розділ 1. Контроль власних фінансів – складна інтелектуальна інформаційна система*	*16.04.2021*	*16.04.2021*
6.	*Розділ 2. Обґрунтування вибору інструментарію та мови UML для моделюванні програмного продукту інтелектуальної системи «фінансист»*	*24.05.2021*	*24.05.2021*
7.	*Розділ 3. Процеси моделювання та впровадження архітектури програмного продукту інтелектуальної системи «фінансист»*	*21.06.2021*	*21.06.2021*
9.	*Розробка програми та методики тестування*	*18.10.2021*	*18.10.2021*
10.	*Написання наукової статті*	*22.05.2021*	*22.05.2021*
11.	*Керівництво користувача*	*21.10.2021*	*21.10.2021*
12.	*Висновки та пропозиції*	*01.11.2021*	*01.11.2021*
13.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*03.11.2021*	*03.11.2021*
14.	*Підготовка автореферату та презентації доповіді*	*03.11.2021*	*03.11.2021*
15.	*Попередній захист випускного кваліфікаційного проєкту*	*22.11.2021 – 25.11.2021*	*22.11.2021*
16.	*Здача зброшурованої випускного кваліфікаційного проєкту*	*25.11.2021*	*25.11.2021*
17.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*	*26.11.2021*	*26.11.2021*
18.	*Підготовка до публічного захисту випускного кваліфікаційного проєкту*		

7. Дата видачі завдання «29» грудня 2020 р.

8. Науковий керівник випускного кваліфікаційного проєкту Криворучко О.В.

(прізвище, ініціали, підпис)

9. Гарант освітньої програми Токар В.В.

(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Франчук Т.М.

(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускного кваліфікаційного проєкту

(підпис, дата)

Відмітка про попередній захист _Криворучко О.В.

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проєкт

Випускний кваліфікаційний проєкт студента ___Франчук Т.М.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми_____Токар В.В.

(прізвище, ініціали, підпис)

Завідувач кафедри_____Криворучко О. В.

(підпис, прізвище, ініціали)

«___»_20_____ р.

АНОТАЦІЯ

Випускна кваліфікаційна робота на здобуття освітнього ступеня магістр за спеціальністю 121 – «Інженерія програмного забезпечення». – Київський національний торговельно-економічний університет, Київ, 2021 р.

Метою роботи є дослідження проблеми керування фінансами споживача та розробка архітектури програмного продукту, яка завдяки когнітивним технологіям дозволить спросити фіксацію та аналіз витрат.

В рамках даної випускної кваліфікаційної роботи виконується розробка архітектури програмного продукту (системи) для ведення розрахунку та аналізу особистих фінансів, а саме інформаційно-інтелектуальної системи !Фінансист» Об’єктом дослідження є проблема контролю особистого бюджету. Бюджет включає в себе детальну інформацію про перебіг коштів, за певний проміжок часу. Він може бути як паперовим, так і електронним. Також може відображати інформацію про прогноз-плани та заощадження.

В роботі запропонована архітектура програмного продукту для створення програмного забезпечення інформаційно-інтелектуальної системи «Фінансист», що інтегрується у месенджер, та завдяки когнітивним технологіям дає змогу розпізнати команди користувача, для можливості фіксування та аналізу витрат користувача.

В результаті аналізу предметної області було вирішено, що розроблюване рішення повинне становити собою дві серверні частини, для бота та безпосередньо для фінансового асистента.

Технічні вимоги – технології Docker, C#, PostgreSQL, Azure Services.

Ключові слова: програмний продукт, ефективність, об’єктивність, надійність системи, мова програмування, когнітивні технології, база даних, технічні вимоги, область застосування, інформаційні технології.

ANNOTATION

Graduation thesis for a master's degree in specialty 121 - "Software Engineering". - Kyiv National University of Trade and Economics, Kyiv, 2021.

The aim of the work is to study the problem of consumer financial management and develop a software architecture that, thanks to cognitive technologies, will require the fixation and analysis of costs.

Within the framework of this final qualification work, the architecture of the software product (system) for the calculation and analysis of personal finances, namely the information-intellectual system! Financier ”is being developed. The budget includes detailed information on the flow of funds over a period of time. It can be both paper and electronic. It can also display information about forecast plans and savings.

The paper proposes the architecture of the software product for creating software of the information-intelligent system "Financier", which is integrated into the messenger, and thanks to cognitive technologies allows to recognize user commands, to be able to record and analyze user costs.

As a result of the analysis of the subject area, it was decided that the developed solution should be two server parts, for the bot and directly for the financial assistant.

Technical requirements - Docker, C #, PostgreSQL, Azure Services technologies.

Keywords: software product, efficiency, objectivity, system reliability, programming language, cognitive technologies, database, technical requirements, scope, information technologies.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення.

ПС – програмна система, комплекс програмного забезпечення.

ПК – персональний комп’ютер, робоча машина для розробки та виконання програм.

C# – об’єктно-орієнтована мова програмування від компанії Microsoft.

ООП – (Об’єктно-орієнтований програмування) парадигма програмування, в якій основою є класи та об’єкти, які між собою взаємодіють.

UML – (Unified Modeling Language) уніфікована мова графічного представлення та об’єктного моделювання в області розробки програмного забезпечення парадигми об’єктно-орієнтованого програмування.

Алгоритм – набір інструкцій, які описують порядок виконання дій, що дозволяють досягти результату за скінченну кількість кроків.

Програмна бібліотека – пакет підпрограм або об’єктів, класів, що використовуються в розробці програмного забезпечення.

НФ – (нормальна форма) властивість відношення в реляційній моделі даних.

ЗМІСТ

					КНТЕУ 121 02з-16.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		27.02.21	*Моделювання архітектури програмного продукту інтелектуальної системи «Фінансист»*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Криворучко О.В.		*27.02.21*		*Зміст*	*3*	*64*
Гарант		Токар В.В.		*27.02.21*		*Факультет інформаційних технологій 2з курс, 2м група*		
Розробив		Франчук Т.М.		*27.02.21*	*Зміст*			

ВСТУП

Проблеми керування власним бюджетом є актуальними у наш час для усіх людей. Адже практично кожного дня ми витрачаємо кошти на придбання чогось в супермаркеті, оплату послуг та ін. Тому вміння керувати своїми коштами є необхідним у сучасних умовах та відіграє досить важливу роль.

Вважається, що контролювати свої фінанси є дуже клопіткою справою, яка може зайняти багато часу. Це правда, але лише у тому випадку, якщо використовувати класичні методи для обліку. Проте у сучасному світі є достатньо технологій, що дозволять спростити цей процес. Прикладом є спеціалізоване програмне забезпечення, яке дозволяє витрачати мінімум часу, для фіксації витрат. Для того, щоб використовувати його (ПЗ), достатньо мати комп'ютери чи мобільний телефон.

Згідно опитувань, майже 65% населення України користується смартфонами, з яких 91% мають встановлені месенджери [1].

На даний момент, месенджер можна використовувати на усіх актуальних операційних системах, тому розробка фінансового асистента, що інтегрується у месенджери, дозволить контролювати витрати, у будь-якому місці, де є доступ до смартфона, чи комп'ютера.

У наш час досить на високому рівні розвинута робота з нейронними мережами. Нейронні мережі активно допомагають людям у повсякденних справах. Вони застосовуються майже у кожній дії, що виконує програмне забезпечення. Від знімку фотографії на смартфон, до керування автомобілем, або літаком. Тому для того, щоб спростити роботу користувачів з інформаційно-інтелектуальною системою «Фінансист» застосовуються системи розпізнання мови.

В якості реалізації проекту було вирішено розробити інформаційно-інтелектуальну систему, що включає в себе бота, на прикладі месенджера «Telegram», та сервіс асистента, що дозволить розпізнавати, зберігати та аналізувати облік фінансів користувача. Для розпізнавання розмовної мови використано технології платформи Microsoft Azure, а саме когнітивний сервіс Luis. Дана програмна система призначена, насамперед, для молоді, які щодня використовують месенджери.

Мета дослідження облік та прогнозування процесів ведення особистого бюджету за допомогою інтелектуально-інформаційної системи, яка завдяки інтеграції у месенджери, має можливість запускатись на будь-якій сучасній платформі.

Об'єкт дослідження: процеси моделювання архітектури програмного забезпечення та когнітивні технології розпізнання мови за для фіксації витрат та доходів користувача у месенджері.

Предмет дослідження: розробка інтелектуально-інформаційної системи, що використовує когнітивні технології для розпізнання команд користувача.

У відповідності з метою дослідження поставлені наступні завдання:

- проаналізувати існуючі системи для обліку фінансів;
- дослідити можливість застосування когнітивних технологій;
- розробити програмну систему, яка інтегрується в обраний месенджер у вигляді бота;
- розробити серверну частину інтелектуально-інформаційної системи, яка за допомогою когнітивних технологій сервісу Luis буде аналізувати введений текст користувача та опрацьовувати його.
- розробити методи та алгоритми обробки та аналізу даних по витратах та доходах користувача

Методи дослідження: системний підхід для аналізу інформації, структурно-функціональний метод для здійснення постановки задачі, методи моделювання

архітектури програмного забезпечення, когнітивні технології.

Практичне значення дослідження: отримати програмне забезпечення яке дозволить користувачеві, використовуючи когнітивні технології розпізнання мови, фіксувати свої витрати та доходи у месенджері, та проводити їх аналіз та отримувати прогнозні рішення.

РОЗДІЛ 1

КОНТРОЛЬ ВЛАСНИХ ФІНАНСІВ – СКЛАДНА ІНТЕЛЕКТУАЛЬНА ІНФАРМАЦІЙНА СИСТЕМА

Слід зазначити, що контроль фінансів особистості не можна назвати складним процесом, але це здається на перший погляд. Для реалізації проєкту по контроля власних фінансів потрібно розуміти базові речі, з яких він складається. З часом, ці речі можна розширювати, в залежності від потреби. Одними з початкових частин ведення власного бюджету є фіксація витрат, яка в подальшому дозволить зробити їх оптимізацію. Також, проаналізувавши власні витрати, можна дізнатись, чи багато коштів йде на «непотрібні» речі, адже імпульсні придбання - один з варіантів для відстеження витрат - це може бути будь-який інтелектуальний програмний продукт (ПП). ПП можуть показати графіки та діаграми, щоб проілюструвати звички для витрат та інше. Такі ПП можуть функціонувати в додатку для телефону або за допомогою спеціального ПЗ, яке може синхронізуватися з ПЗ на вашому персональному комп'ютері.

Дослідженню моделювання архітектури програмного продукту, визначенню структури, основних характеристик присвячені праці вітчизняних та закордонних науковців . Ямпольського Л.С., Мельничук П.П., Самотокіна Б.Б., Мері Шоу і Дэвид Гарланд. Використанням когнітивних технологій в ІТ-сегменті займаються Осіпов Г.С., Максімов В.І., Корноушенко Є.К., Остапов С.Е. та інші.

1.1. Аналіз предметної області інтелектуальної інформаційної системи

Контроль фінансів не є складним процесом, як може здатися при першому знайомстві. Для його реалізації потрібно розуміти базові речі, з яких він складається. З часом, ці речі можна розширювати, в залежності від потреби. Одними з початкових частин ведення бюджету є фіксація витрат, яка в подальшому дозволить зробити їх оптимізацію. Також, проаналізувавши витрати, можна дізнатись, чи багато коштів йде на непотрібні речі, адже імпульсні покупки є витратами, від яких можна відмовитись.

Найбільш важливою частиною відстеження фінансів є систематичність. Незалежно від того, яким чином буде проводитись реєстрація транзакцій, завжди необхідно мати можливість легко і надійно посилатися на них. Обов'язково кожен запис потрібно доповнювати важливою інформацією, така як дата, витрачена або отримана сума і категорія витрат. Також потрібно вести записи послідовно. Наприклад, можна записувати транзакції, як тільки вони відбуваються, кожен раз, при поверненні додому, або навіть раз в тиждень.

Категорії витрат - це простий спосіб з'ясувати, на що витрачається найбільше грошей. Ці категорії можуть включати в себе такі речі, як житло, комунальні послуги, домашні витрати, продукти, охорону здоров'я, домашні тварини, особисті витрати і розваги. Ці категорії, звичайно, будуть відрізнятися у кожної людини, і можна розділяти категорії настільки конкретно або загально, наскільки це потрібно. Важливим є те, що класифікація витрат є узгодженою між транзакціями.

Найпростіший спосіб відслідковувати фінанси - записувати інформацію щодо кожної транзакції в блокнот. В кінці кожного періоду (тижня або місяця) також можна перенести інформацію в електронну таблицю, щоб вона була більш доступною. [2].

Щоб створити особистий бюджет, потрібно з перерахування фіксованих витрат щомісяця (наприклад, орендної плати та комунальних послуг) як витрати

в перший день кожного місяця разом з очікуваним доходом за цей місяць. Потім потрібно відняти інші витрати або додати інші доходи в міру необхідності протягом тижня або місяця.

Але у кожного способу є свої недоліки. У випадку з паперовим рішення контролю фінансів, не завжди є можливість взяти з собою блокнот, та у разі його втрати, потрібно буде почати все з початку. З таблицями теж не все так зручно. Доступ до особистого комп'ютера також буде не завжди. Ситуацію може змінити смартфон, який, у наш час, люди беруть з собою майже всюди. Для сучасних смартфонів є достатньо додатків, для керування таблицями, але навіть беручи до уваги розміри екрану смартфонів, для зручної роботи з таблицями його не достатньо.

Один з моментів вирішення пробелми відстеження витрат є будь-яка фінансова програма, в даному випадку Інтелектуальна система «Фінансист». Програми можуть показати графіки та діаграми, щоб проілюструвати звички для витрат. Вносити покупки в категорію нескладно, одночасно вводячи їх в поточний рахунок. Це можна зробити в додатку для телефону або за допомогою спеціального ПЗ, яке може синхронізуватися з ПЗ на вашому комп'ютері.

В період проведення дослідження не було виявлено цілком безкоштовних застосунків для ведення обліку особистих витрат, що дозволяють працювати на більшості популярних операційних системах смартфонів та комп'ютерів.

Майже усі додатки, що реалізовані під більшість ОС мають обмежений режим користування, а для розблокування усіх можливостей, потрібно купити підписку, або повну версію додатку. Також на різних платформах відрізняється інтерфейс додатку, що є не зручним рішенням, тому що змушує ще раз вивчати роботу додатку при зміні пристрою, з якого відбувається доступ до додатку.

1.2. Висновки до Розділу 1

ІУС мінімаркетом – це система яка спрощує та прискорює обслуговування клієнтів. Саме тому то цієї системи висувається ряд вимог, які повинні забезпечувати якісне та коректне функціонування програми. Цей Огляд наукових робіт та практичний досвід показує, що проблема контролю особистого бюджету включає в себе детальну інформацію про перебіг коштів, за певний проміжок часу. Як правило будь-хто не є об'єктивним фінансистом-аудитором власного бюджету і відслідковувати, контролювати та прогнозувати власний бюджет доволі складно. Облік витрат потрібен, насамперед, для визначення основних речей, на які найбільше витрачається коштів, та допоможе проаналізувати, на чому можна їх заощадити.

Відповідно контроль власних фінансів може бути, як паперовим, так і електронним. Враховуючи стрімкі процеси діджиталізації перевага все ж таки належить електронним порадникам, які можуть відображати інформацію про майбутні плани та заощадження. Одним з рішень даного напряму є когнітивні аспекти ухвалення рішень.

.

РОЗДІЛ 2

ОБГРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТАРІЮ ТА МОВИ UML ДЛЯ МОДЕЛЮВАННІ ПРОГРАМНОГО ПРОДУКТУ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ «ФІНАНСИСТ»

2.1. Огляд існуючого програмного забезпечення для керуванням власним бюджетом

Провівши дослідження існуючих програмних продуктів, які можуть вирішити проблему для споживача – особистого фінансового консультанта не було виявлено цілком безкоштовних застосунків для ведення обліку особистих витрат, що дозволяють працювати на більшості популярних операційних системах смартфонів та комп'ютерів.

Додаток Money Lover дозволяє контролювати свої фінанси. Має підтримку Android, iOS і також веб версію. Для збереження даних може використовувати хмарне сховище Dropbox. Користування цим додатком є частково безкоштовним. В безкоштовній версії є можливість активувати до 5 пристроїв. Платна версія дозволить обійти це обмеження, та дасть можливість експортувати дані у Excel формат.

Додаток Bills Monitor - для керування бюджетом. Має можливість сповіщеннями нагадати користувачеві про майбутню покупку, або оплату сервісу. Підтримує лише платформу IOS.

Додаток Monefy - підтримується на платформах Android, iOS, Windows. Дозволяє зберігати витрати у хмарних сховищах. Також реалізована підтримка ведення сімейного бюджету. Для розблокування повного функціоналу потрібно придбати платну версію. Додаток CoinKeeper дозволяє фіксувати витрати та доходи.

Має підтримку Android, iOS та веб версію. Також дозволяє будувати діаграми на основі витрачених коштів. Для повного функціоналу потребує платної підписки.

Додаток Goodbudget – платформа, яка розроблена для обліку особистих фінансів. Дозволяє проводити планування майбутніх витрат, та має можливість повідомити користувача, про досягнення ліміту по бюджету. Підтримує платформи Android, iOS, Web. Для розблокування повного функціоналу також вимагає платної підписки. [37]

Money Lover	Bills Monitor	Monefy	CoinKeeper

Рис. 2.1. Інтерфейси додатків

Goodbudget – платформа, яка розроблена для обліку особистих фінансів. Дозволяє проводити планування майбутніх витрат, та має можливість повідомити користувача, про досягнення ліміту по бюджету. Підтримує платформи Android, iOS, Web. Для розблокування повного функціоналу також вимагає платної підписки [7].

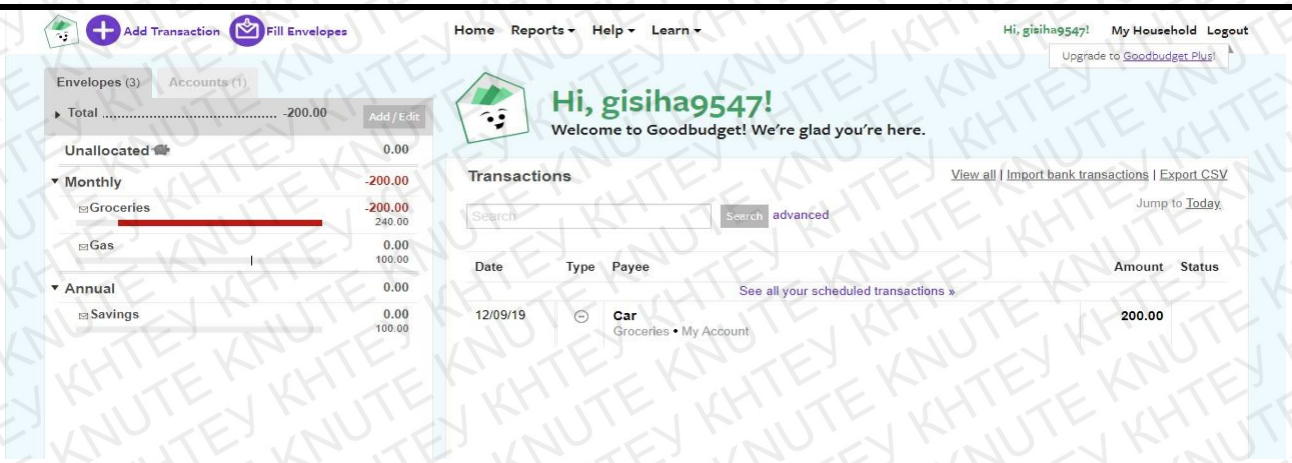

Рис. 2.2 Інтерфейс додатку Goodbudget

Проаналізувавши ринок додатків, можна сказати, що у більшості з них, для розблокування повного функціоналу, потрібно купувати додаток, або купувати підписку. Також існує відмінність інтерфейсів, залежно від платформи, що змушує кожного разу звикати до нового, при зміні клієнта. Також, у деяких додатків реалізована підтримка не усіх платформ.

2.2. Вибір процесу проектування програмного продукту інтелектуальної системи «Фінансист»

Ітеративний процес розробки був запропонований для заміни моделі водоспаду, у якому було виявлено низку проблем. Модифікований водоспад, раціональний єдиний процес (RUP) та більшість моделей засновані на ітераціях. Загальна ідея полягає у розробці системи за допомогою ітерацій (повторних циклів) та поступово (невеликими частинами часу). Завдяки ним розробники можуть аналізувати свої помилки і застосовувати ці знання на наступних ітераціях [8].

Деякі моделі можуть мати різні назви для ітерації, наприклад спринт. Ітерації можуть бути обмеженими в часі. Також вони закінчуються після узгодженого періоду незалежно від кількості задач, які були виконані. Альтернативний спосіб виконання ітерацій - обмежити їх за обсягом.

Вони тривають до повного завершення (розроблення та випробування) узгодженого обсягу.

Загальноприйнята практика полягає в тому, що кожна ітерація закінчується демонстрацією для зацікавлених сторін. Ця демонстрація використовується як процес аналізу помилок з метою виправлення їх у подальших ітераціях застосування (рис. 2.3). Оскільки працююча модель доступна набагато раніше, набагато простіше помітити проблеми, перш ніж буде пізно або занадто дорого робити виправлення.

Такий тип розробки схожий до моделі водоспаду, де кожна ітерація життєвого циклу розробки ПЗ повинна бути повністю завершена до наступного. Однією з головних переваг ітеративного розвитку є те, що він дозволяє з більшою гнучкістю адаптуватися до змін. На відміну від моделі водоспаду, де непередбачені проблеми часто виникають із запізненням у проекті та їх досить дорого виправити, ітеративний підхід, з іншого боку, проходить через короткі цикли, які дозволяють команді вчитися, адаптуватися та змінювати напрямок у наступній ітерації.

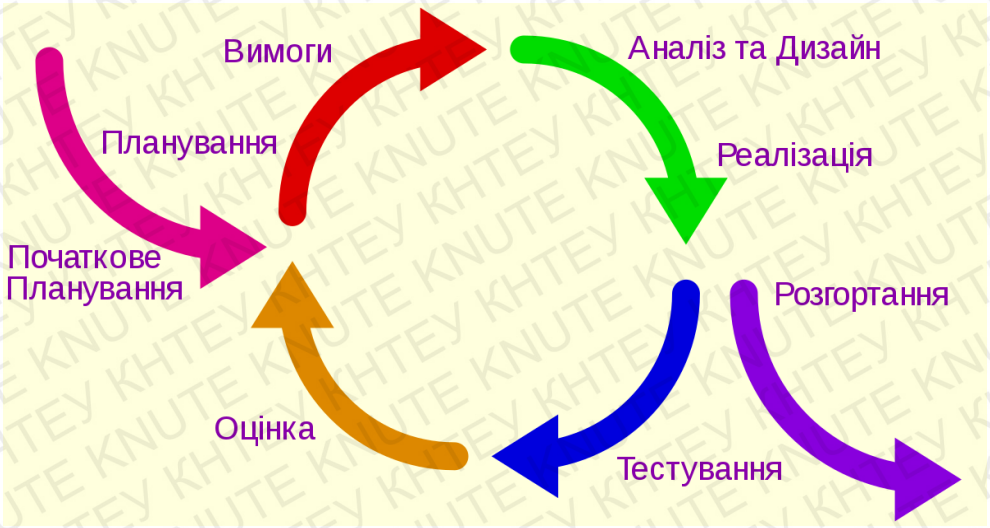

Рис. 2.3. Ітераційного підхід в розробці ПЗ

Не для усіх розробників підходить ітеративний підхід, тому що не кожен може ефективно використовувати його. Ітеративний розвиток набагато

складніший, ніж модель водоспаду. Це вимагає більш високого рівня технічної досконалості, більше дисципліни та розуміння процесу усім колективом. Часто потрібно, щоб члени команди могли одночасно виконувати декілька завдань. Наприклад, розробляти ПЗ і паралельно тестувати його.

Фаза інтеграції в ітераційному розвитку дуже коротка або, якщо це зроблено правильно, безперервна. Хоча в моделі водоспаду ця фаза може зайняти навіть кілька тижнів для великих проектів, ітерації вимагають, щоб вона була дуже короткою і робилася часто. Якщо, наприклад, тестувальникам потрібно перевірити певну функціональність, як тільки буде написано код, інтеграція до поточної системи та розгортання повинні бути майже миттєвими. Наразі існує багато інструментів, які полегшують інтеграцію та розгортання.

Часом недостатньо часу для повторного ручного тестування після завершення функціоналу, тому потрібен високий рівень автоматизації. Поки розробляється частина функціоналу, тестерам потрібно писати сценарії, які перевірять її. Автоматизація вимагає навичок програмування, якими не всі тестери володіють. Як результат, тестова автоматизація може бути залишена розробникам, поки тестери продовжують зосереджуватися на ручному тестуванні. Однак у цих випадках тестери можуть відчути, що частина їхньої роботи, яку вони повинні виконувати, переходить до розробників.

Кінцеві продукти часто узгоджуються з потребами клієнта завдяки перепланування, що виконується в кожній ітерації та коригується залежно від зворотного зв'язку. Більш високий рівень автоматизації, необхідний для успішних ітерацій, дозволяє швидше виявити проблеми та створити надійні та повторювані процеси. Ця ж автоматизація після початкових витрат часу призводить до скорочення витрат. Передача досвіду між членами команди збільшує спільні знання в колективі, що призводить до кращого розуміння процесу.

2.3. Мова UML в моделюванні програмних продуктів

Виходячи із поставленого завдання, можна виділити основні типи користувачів (акторів) даної системи: адміністратор та користувач. Опис акторів наведено у таблиці 2.1.

Таблиця 2.1

Актори інформаційної системи управління стадіоном

Актор	Короткий опис
Адміністратор	Особа, яка займається редагуванням сутностей та категорій.
Користувач	Особа, що використовує програму.

Завданнями адміністратора є підтримка інформації щодо категорій витрат та доходів в актуальному стані.

Таблиця 2.2

Опис прецедентів для адміністратора

Актори	Найменування	Формулювання
Адміністратор	Керування категоріями асистента	Дозволяє додати, видалити та відредагувати
Адміністратор	Опрацювання відгуків	Дозволяє переглядати відгуки користувачів
Адміністратор	Керування LUIS API	Дозволяє керувати LUIS параметрами за допомогою API

Користувач – звичайний користувач системи, що має можливість вести облік витрат та доходів. Також може будувати звіти, встановлювати інформаційні обмеження на витрати та керувати заощадженнями.

Таблиця 2.3

Опис прецедентів для користувача

Актори	Найменування	Формулювання
Користувач	Керування витратами та доходами	Можливість внести, видалити та редагувати дані
Користувач	Встановити інформаційне обмеження на витрати	Цей варіант використання дозволяє користувачеві задати ліміт на витрати на певний період часу
Користувач	Отримати текстовий звіт	Дозволяє користувачеві отримати звіт по витратах та доходах у наступних форматах: CSV файл, Excel таблиця, текст.
Користувач	Отримати графічний звіт	Можливість отримати звіт у вигляді діаграми або гістограми
Користувач	Провести аналіз витрат	Дозволяє користувачеві отримати аналіз витрат
Користувач	Керування заощадженнями	Можливість керування заощадженнями
Користувач	Встановити нагадування	Дозволяє встановити нагадування про майбутню подію або транзакцію

Актори та варіанти використання для користувача можна відобразити у вигляді діаграми (рис. 2.1), з якої ми бачимо який функціонал необхідно реалізувати в системі.

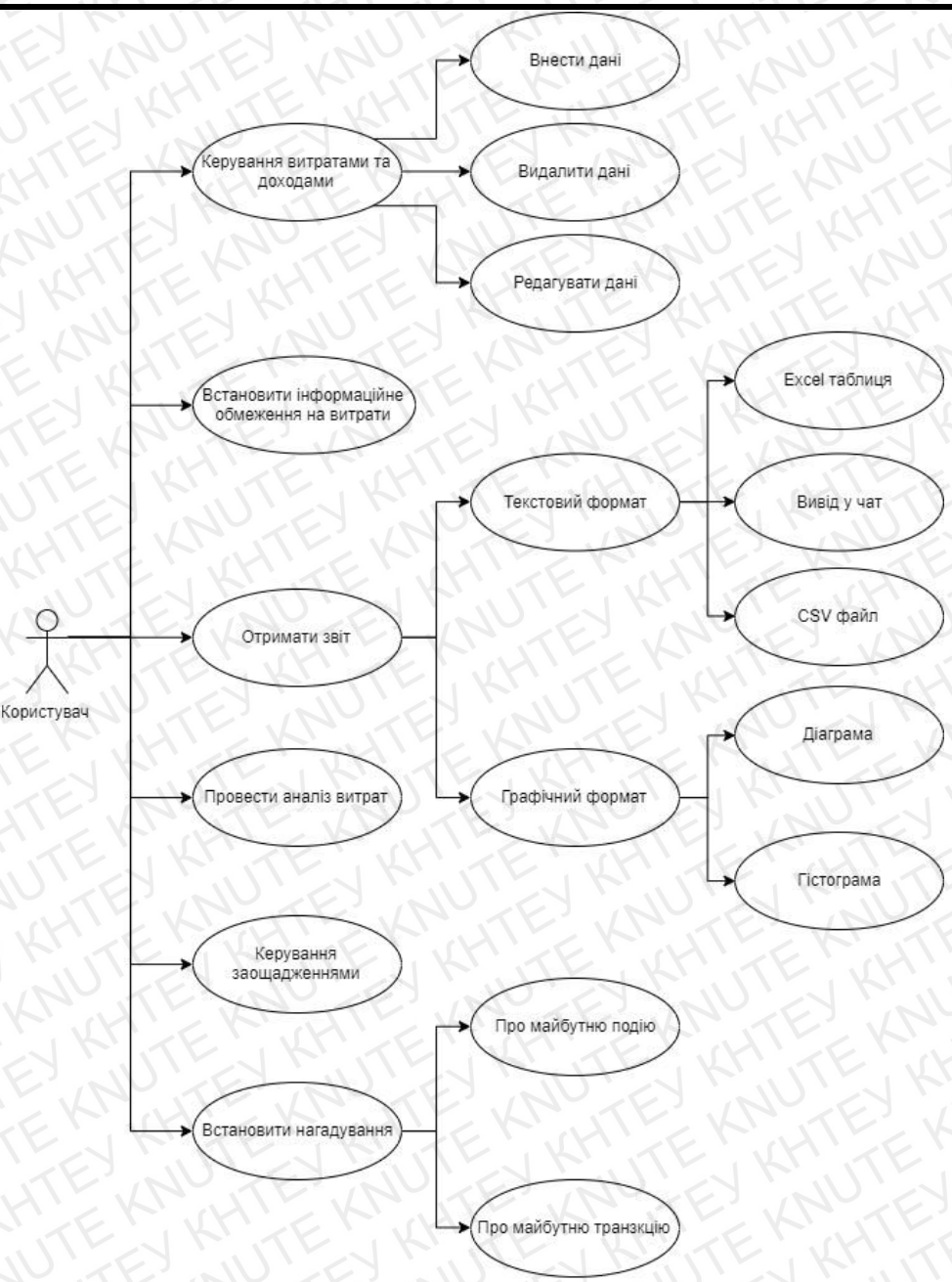

Рис. 2.1. Опис прецедентів для користувача

На рисунку 2.2 зображено функціонал, який повинен підтримувати система для ролі адміністратора.

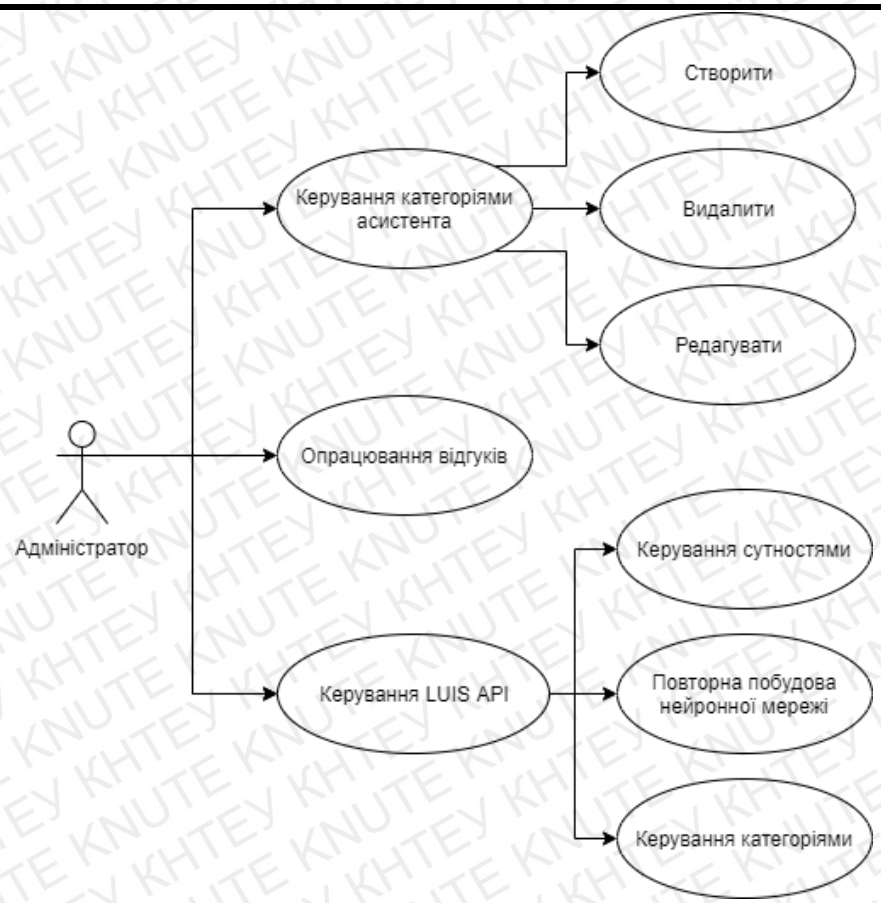

Рис. 2.2. Опис прецедентів

2.4. Створення бази даних інтелектуальної системи «Фінансист»

Для побудови схеми бази даних потрібно дослідити предметну область, визначити основні сутності, що будуть використовуватись у системі, та проаналізувати їх взаємовідношення.

Сутності – це певні об’єкти з реального світу, про яких потрібно зберігати інформацію у базі даних. Для правильної побудови моделі сутностей, потрібно виділити їх основні атрибути, та між якими сутностями будуть залежності. Сутність не завжди повинна бути матеріальною. Наприклад такі речі як черга, поїздка і т.д. теж можуть бути виділеними як сутності, з яких можливо визначити атрибути [9].

Атрибутами є параметри сутностей, такі як ім’я, опис, розмір, колір і т.д. Існують атрибути, що можуть містити або одне, або декілька значень. Якщо у

випадку, коли потрібне лише одне значення, не виникає проблем, тоді як для зберігання багатьох значень потрібно генерувати відношення, та виносити ці значення у окремі таблиці. Це потрібно, для нормалізації бази даних, та виконання першої нормальної форми.

Нормалізація бази даних – винесення усіх зв'язків, згідно з алгоритмом, у окремі таблиці, або у більш детальні відношення [10].

Більша частина СУБД орієнтована на досягнення ступеня нормалізації третьої нормальної форми (3НФ). Це пов’язано з тим, що зведення відношень до 3НФ цілком відповідає майже усім практичним задачам. При розробці винятково великих систем на надшвидкодіючих комп'ютерах, коли необхідно забезпечити максимальне зменшення обсягів даних, бажано виконати подальшу нормалізацію відношень.

Відношення буде зведено до третьої нормальної форми (3НФ) тоді й тільки тоді, коли воно є у другій нормальній формі і у ньому немає транзитивних залежностей між неключовими атрибутами, тобто значення будь- якого атрибута відношення, що не входить до первинного ключа, не залежить від значення іншого атрибута, що не входить до первинного ключа [13].

Побудована схема бази даних зображена на рисунку 2.3.

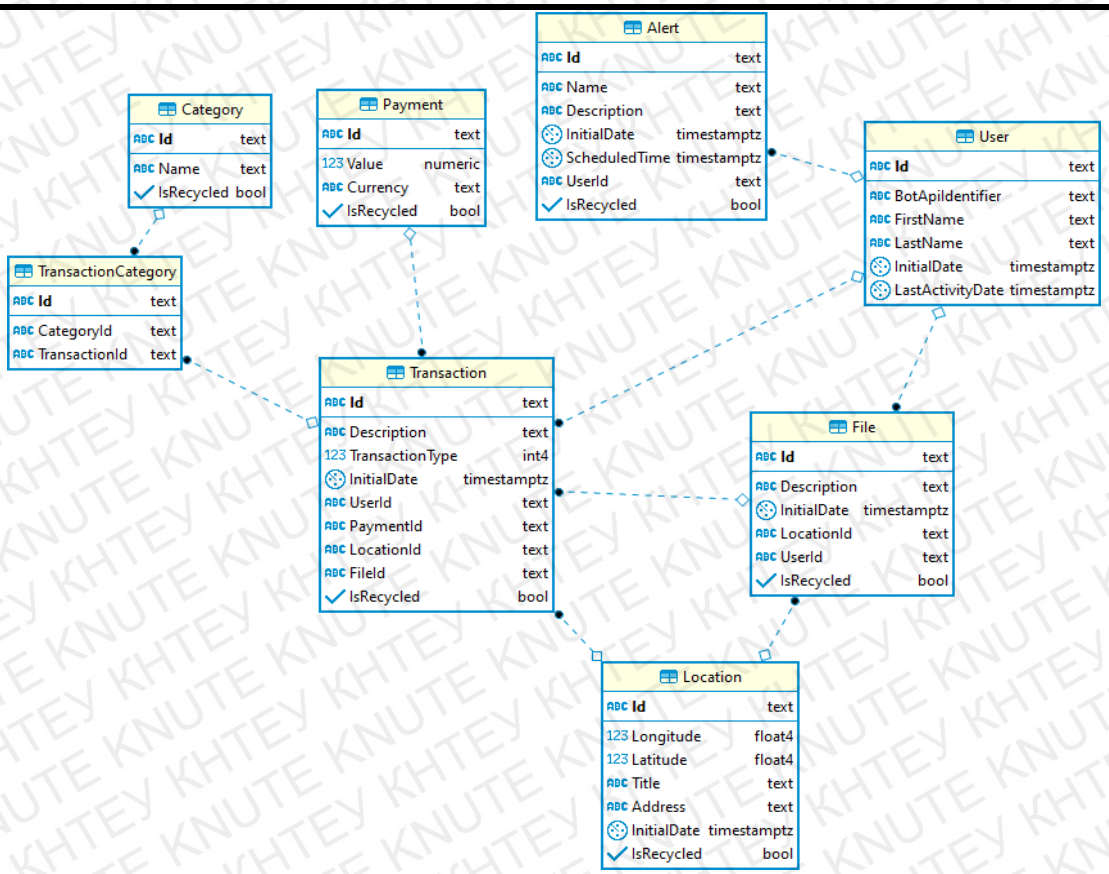

Рис. 2.3. Схема бази даних інтелектуальної системи «Фінансист»

2.5. Побудова UML-діаграми класів програмного продукту інтелектуальної системи «Фінансист»

Для опису архітектури програмного продукту інтелектуальної системи «Фінансист» використовуються UML діаграми класів.

UML – мова моделювання, що є уніфікованою, яка застосовується при проектуванні архітектури програмної системи. Також активно використовується при документуванні систем [14].

Як було сказано, при розробці даної системи використовується ітераційна модель. Після завершення кожної ітерації, потрібно оновлювати діаграми, для загального розуміння системи, що дозволить покращити загальну якість продукту, та дозволить новим розробникам швидше зрозуміти структуру проекту.

UML під час розробки зазвичай використовується наступними ролями:

Розробниками програмної системи;

Архітекторами;

Аналітиками;

Менеджерами.

Однією з частин системи є бот месенджера. Базова логіка для отримання та опрацювання повідомлень зображена у вигляді UML діаграми на рис. 2.4. Центральним об’єктом, що відповідає за робота безпосередньо з самим telegram API є Bot Manager. На подію, що сигналізує про нове повідомлення підписується Message Emitter, що вирішує, чи користувач є авторизованим, та передає повідомлення на опрацювання об’єкту Message Processor.

Для більшої компактності діаграми, параметри методів були приховані. Після того, як серверна частина бота провела базове опрацювання повідомлення, воно передається до наступного сервісу – асистента. На рис. 2.5 зображено UML діаграму реалізації обробки транзакцій. За обробку та валідацію кожної сутності відповідає свій менеджер. Для обробки транзакції в цілому відповідає об’єкт Data Provider.

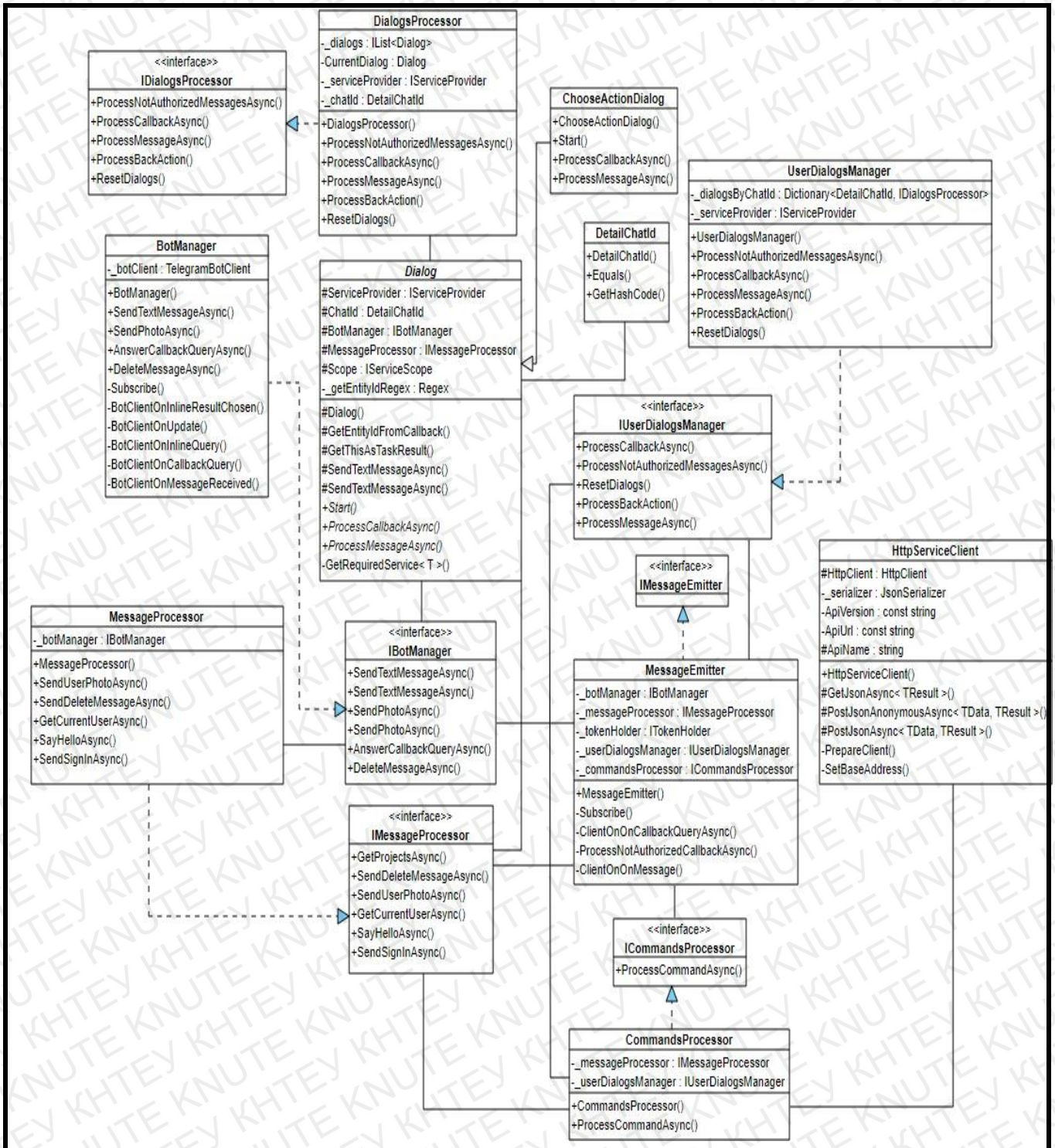

Рис. 2.4. UML діаграма серверної частини бота

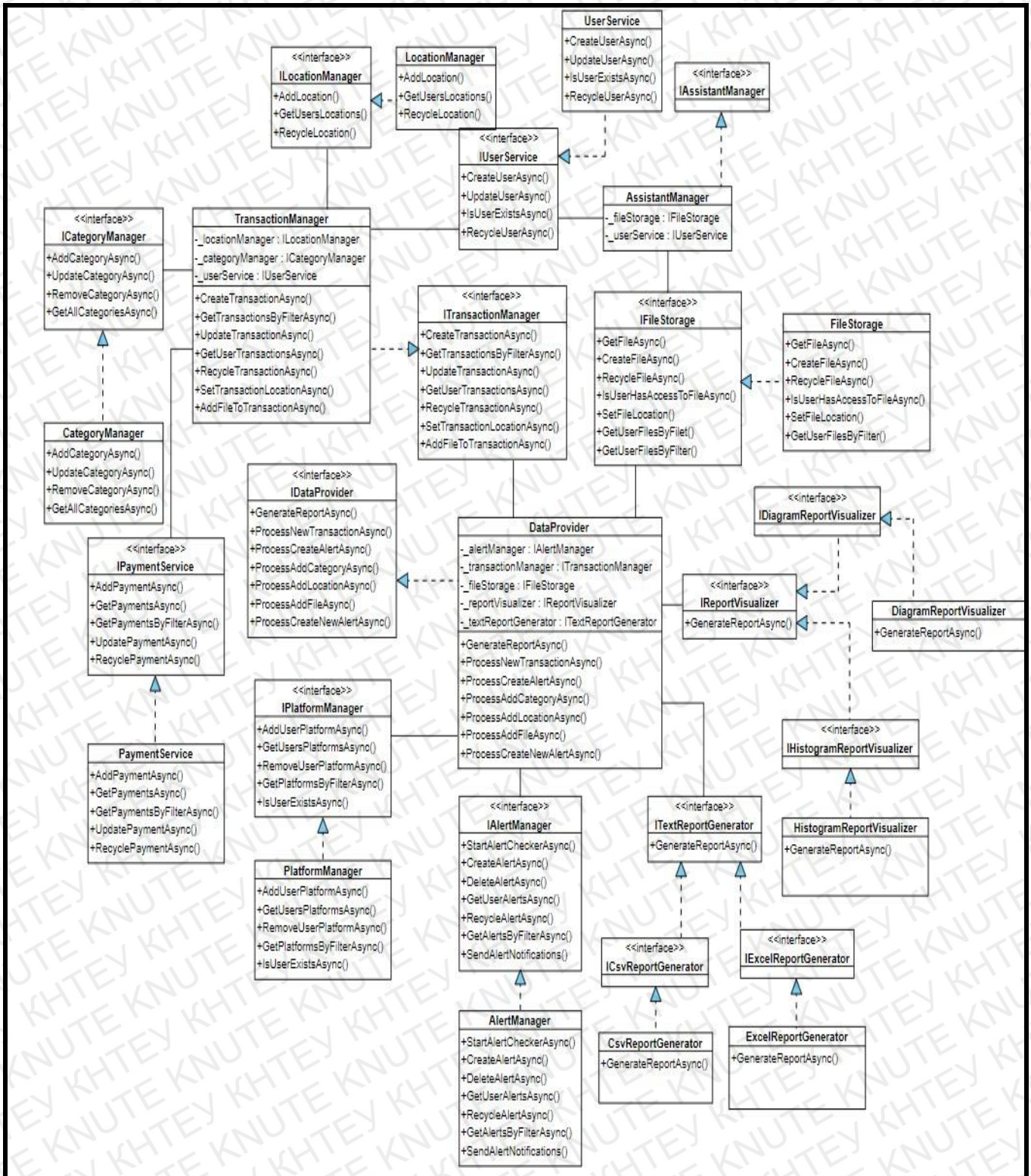

Рис. 2.5. UML діаграма серверної частини асистента

Для роботи з базою даних використовується ORM Entity Framework та підхід Code First. На рис. 2.6 зображено UML діаграму класів для реалізації CRUD операцій над таблицями.

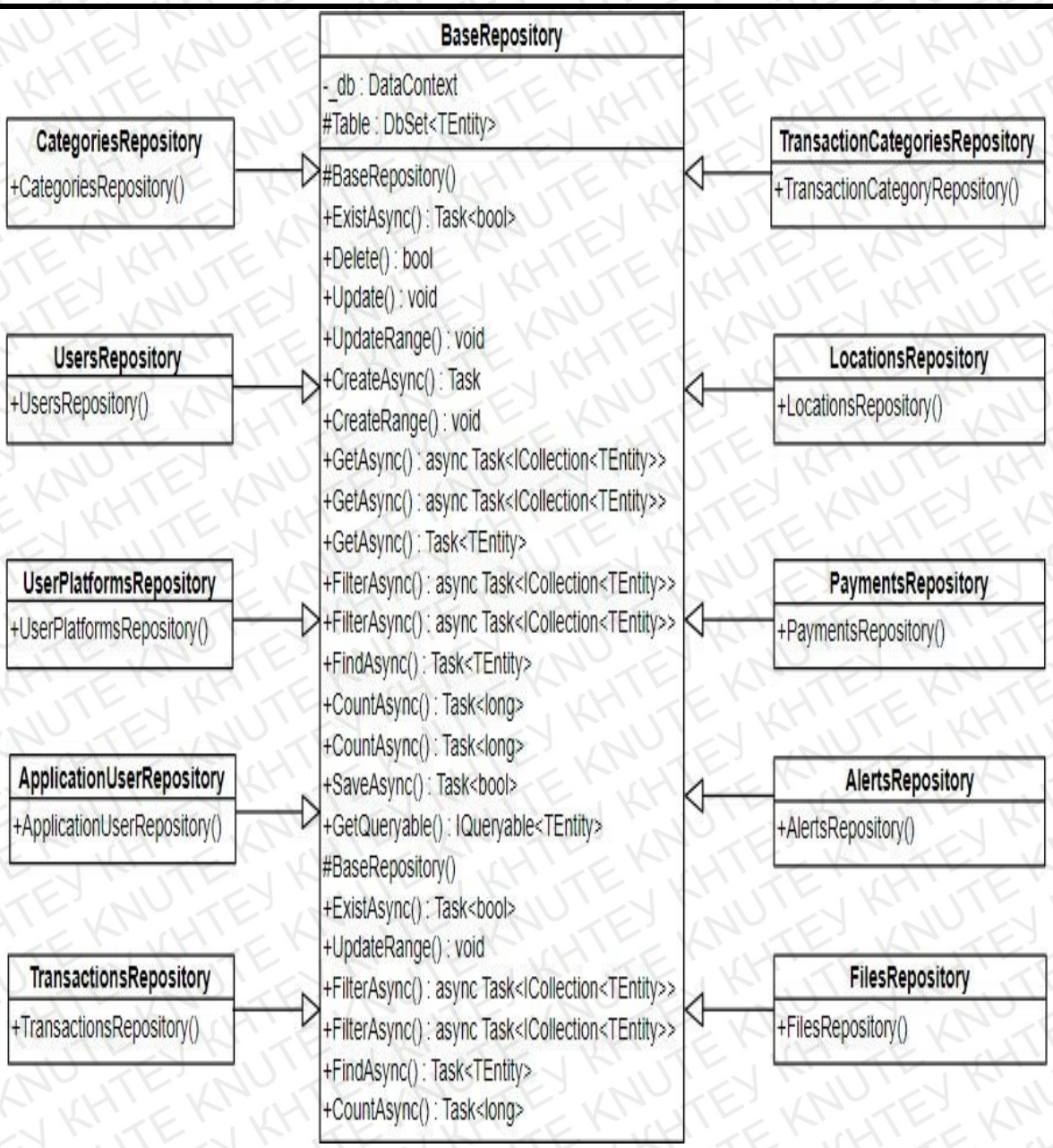

Рис. 2.6. UML діаграма роботи за базою даних

Спочатку створюється базовий репозиторій, з методами, для отримання, додавання, видалення, та редагування даних. Після цього додаються конкретні репозиторії для таблиць, що є у системі. Завдяки парадигмі ООП, що використовувалась при написанні коду, ці репозиторії наслідують усі методи від базового класу, та будуть працювати лише зі своїми таблицями.

2.6. Висновки до Розділу 2

Запропоновано структура та загальна концепція побудови інтелектуальної системи «Фінансист», ретельно проаналізовані функціональні вимоги та перелік варіантів використання.

UML – мова моделювання, яка є уніфікованою, застосована для опису та проєктування архітектури програмного продукту інтелектуальної системи «Фінансист». Після завершення кожної ітерації, потрібно оновлювати діаграми, для загального розуміння системи, що дозволить покращити загальну якість продукту, та дозволить новим розробникам швидше зрозуміти структуру проекту.

Обрано класичне середовище розробки та мовf програмування C#, проведена розробка проекту.

РОЗДІЛ 3
ПРОЦЕСИ МОДЕЛЮВАННЯ ТА ВПРОВАДЖЕННЯ АРХІТЕКТУРИ ПРОГРАМНОГО ПРОДУКТУ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ «ФІНАНСИСТ»

3.1. Моделювання архітектури інтелектуальної системи «Фінансист»

Для розробки програмної системи було обрано клієнт-серверну архітектуру. Клієнтом буде виступати бот месенджера, який буде відправляти запити до серверної частини, яка буде вести комунікацію з серверам платформи Microsoft Azure.

Клієнт-серверна архітектура поділяє комунікацію програмного забезпечення на дві частини:

- Клієнта що робить запит на опрацювання чи отримання даних;
- Сервер, який отримує запит, та належним чином його опрацьовує.

Для того, щоб зрозуміти, як працює дана архітектура, можна розглянути наступний приклад. Під час використанні браузера, коли користувач запитує певні ресурси з веб-сторінки, клієнтом виступає комп'ютер, на якому запущений браузер. Сервер отримує клієнтський запит, опрацьовує його, та повертає відповідь. Також прикладом є IP-телефонія, коли користувач розпочинає дзвінок, запит йде на віддалений сервер, який перенаправляє запит на іншого користувача

В усіх випадках клієнт генерує запит до сервісів та їх ресурсів, а сервер - в прослуховує мережу та очікує на запит, який він може опрацювати або передати на інший сервер (рис. 3.1).

Сервери, що використовуються в мережі, отримують можливість обслуговувати запити за допомогою встановленому програмному забезпеченні.

Оскільки віддалений сервер може запускати кілька служб або мати на ньому кілька серверних додатків, комп'ютер, призначений для ролі сервера, може виконувати кілька функцій в мережі. Наприклад, веб-сервер також можеодночасно бути в якості поштового сервера. Таким же чином SIP-сервери можуть надавати різні послуги. Реєстратор може реєструвати клієнтів, а також запускати службу визначення місцезнаходження, яка дозволяє клієнтам і іншим серверам визначати місцезнаходження інших користувачів, зареєстрованих в мережі. Таким чином, один сервер може буде багатофункціональним мережі [15]. Інша важлива функція сервера - він працює цілодобово, та його мережева адреса не змінюється, що дозволяє у будь-який момент часу до нього звернутись.

Це одна з його ключових функцій, для пошук інших комп'ютерів в мережі.

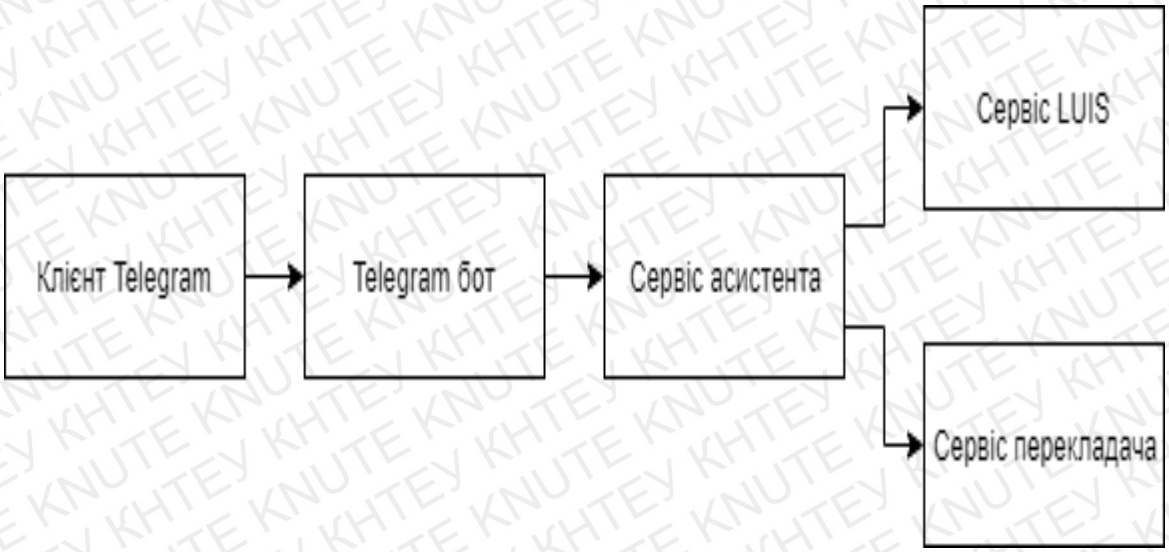

Рис. 3.1. Клієнт-серверна архітектури

Перевагами клієнт-серверної архітектури є спрощення локального обслуговування та покращення системи в цілому. Адже у будь-який момент можна збільшити продуктивність віддаленого сервера за рахунок додавання нових ресурсів, що у наш час відбувається майже без очікування. Також платформа, на якій розгорнуто сервер користувача відповідає за щоденні резервні копії, та безпеку даних [16].

3.2. Вибір мови програмування та середовища розробки

Для розробки програмної системи обрано мову програмування C#.

C# - є мовою об'єктно-орієнтованого програмування для мереж і веб-розробки [17]. Метою C# була розробка мови програмування, який не тільки проста у вивченні, але також підтримує сучасні можливості для всіх видів розробки програмного забезпечення.

C# надає функціональність для розробки програм з новим поколінням програмного забезпечення. C# дозволяє вести розробку для веб-технологій, мобільних пристроїв та додатків. Одними з переваг мови C# є те, що вона підтримує узагальнені типи, типи var, автоматичну ініціалізація типів і колекцій, лямбда-вирази, динамічне програмування, асинхронне програмування, кортежі, розширені налаштування і обробка помилок та багато іншого.

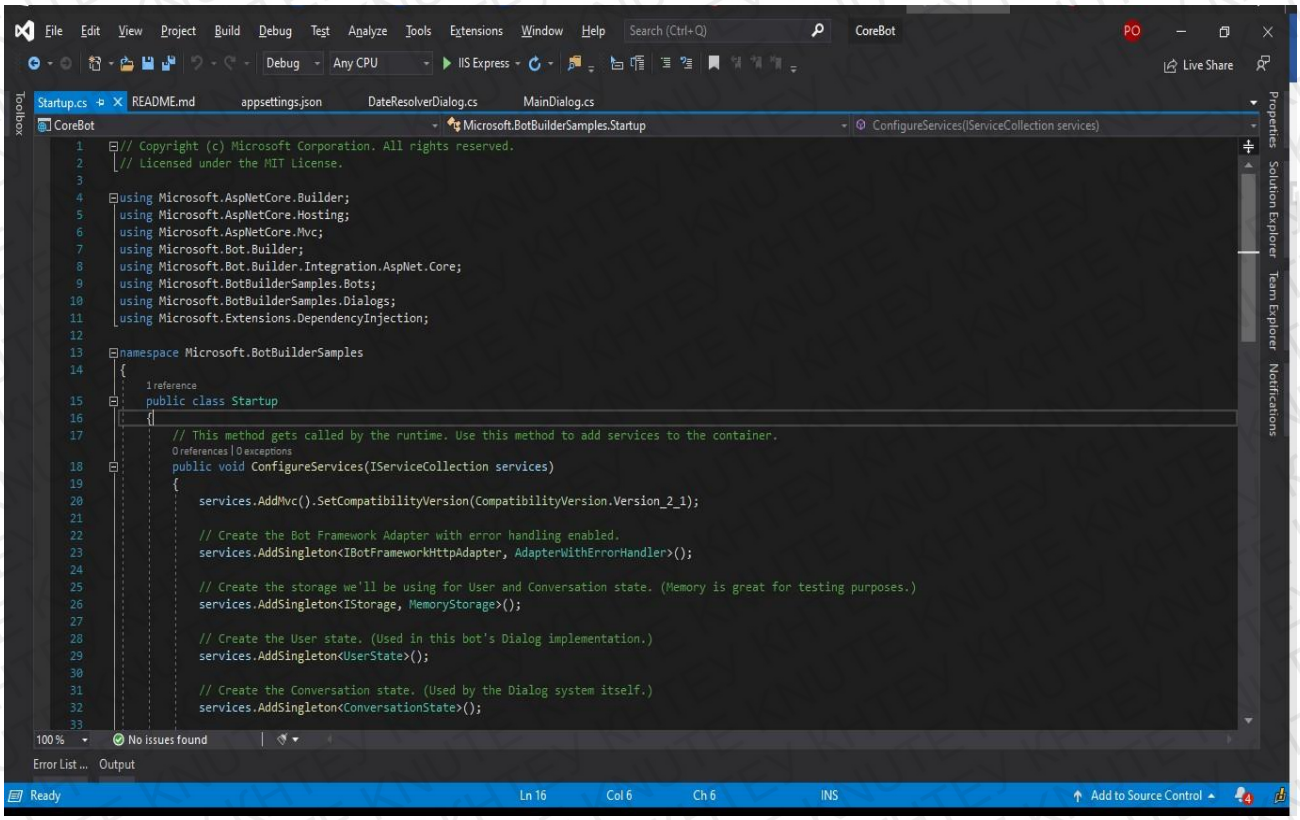

Рис. 3.2. Інтерфейс середовища Visual Studio 2019

Синтаксис мови C # схожий до C ++, Java, Pascal і декількох інших мов. Він є дуже простим, та дозволить швидко його вивчити. У порівнянні з C++, мова C# має строго типізовані логічні типи даних. Беручи до уваги, що C# працює усередовищі CLR, то для розробника спрощується керування пам'яттю, звільнення ресурсів при розробці на даній мові. Для середовища розробки було обрано Visual Studio 2019.

Visual Studio - це середовище розробки, яке розроблене Microsoft. Воно дозволяє створювати графічний інтерфейс користувача, консольний, веб або мобільний додаток, для хмарних і веб-служб. Також середовище дозволяє створювати застосунки під різні платформи, такі як Windows API, Windows store. Visual Studio підтримує написання коду на C#, C++, Visual Basic, Python, JavaScript і багато інших мовах. Середовище підтримується такими платформами як Windows, так і MacOS [18].

3.3. Вибір СУБД та опис її фізичної моделі

Для зберігання даних обрана СУБД PostgreSQL. PostgreSQL – являє собою систему управління БД. Дана система розміщена з відкритим вихідним кодом. PostgreSQL має у собі підтримку SQL і пропонує широкий функціонал: великі запити, різні типи зв'язків, тригери, функції, процедури. Крім того, існують можливості для розширення функціоналу завдяки реалізації можливості створювати власні типи даних, функції, індекси, продедурні мови. PostgreSQL є об'єктно-реляційно, що надає їй ряд переваг, порівняно з іншими СУБД [19].

PostgreSQL є досить надійною та гнучкою, що дозволяє опрацьовувати різноманітні структури даних. Список платформ, що підтримуються досить великий найпопулярнішими з них є Windows, Linux, UNIX, Mac OS, Solaris. PostgreSQL дозволяє зберігати графічні, аудіо та відео файли, а також містить інтерфейси підключення з багатьох мов програмування, та підтримку ODBC Open Database Connectivity для Java, C++, C# і ін.

При розробці інтелектуальної системи «Фінансист», було спроектовано модель бази даних що зображена на рис. 3.3.

База даних складається з наступних таблиць:

- User
- File
- Location
- Transaction
- Transaction category
- Category
- Payment
- Alert

Кожна з таблиць містить унікальний ідентифікатор, та помітку, чи запис є видаленим. Це потрібно для того, щоб не видаляти інформацію користувачів, а лише помітити її видаленою.

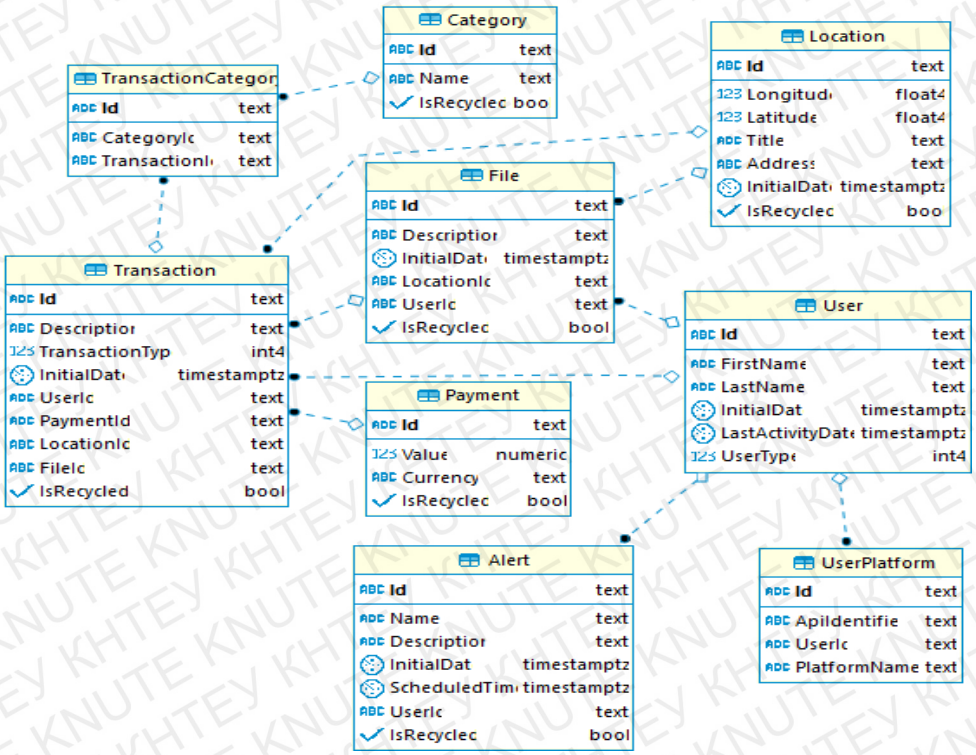

Рис. 3.3 – Фізична модель бази даних

У таблиці User міститься інформація про користувачів системи (Рис. 3.4)

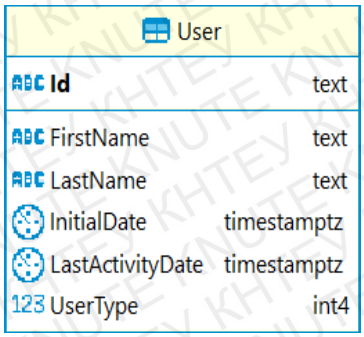

Рис. 3.4. Таблиця User

Поля FirstName та LastName слугують для збереження ім’я та прізвища. InitialDate та LastActivityDate вказують на дату реєстрації користувача та його останню активність . Для визначення типу користувача слугує поле UserType.

Таблиця Alert (рис. 3.5) містить інформацію про створені сповіщення користувачів.

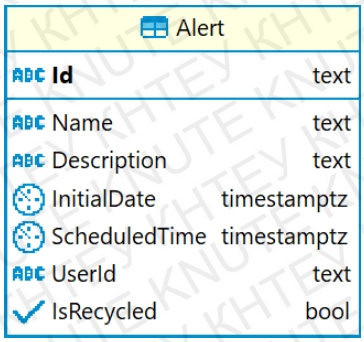

Рис. 3.5 Таблиця Alert

Таблиця Payment слугує для зберігання оплати користувача, де Value – сума, а Currency валюта (Рис. 3.6).

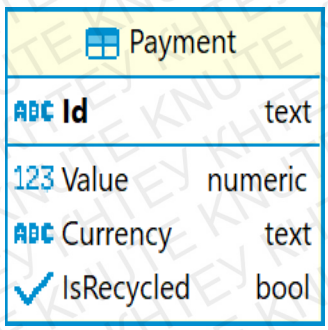

Рис. 3.6 Таблиця Payment

У таблиці File зберігається інформація про фізичні файли, завантажені користувачами (Рис. 3.7). Вони можуть бути прикріпленими до транзакції, або безпосередньо до користувача. Поле LocationId слугує для фіксації розташування, під час якого було завантажено файл.

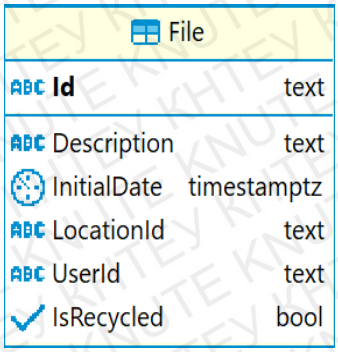

Рис. 3.7 Таблиця File

Таблиця Transaction містить інформацію про усі транзакції в системі. Вказує на локацію, оплату, файл, та користувача що здійснив транзакцію (Рис. 3.8). Також містить тип транзакції.

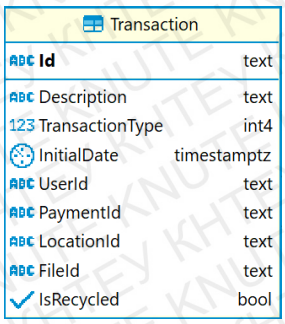

Рис. 3.8 Таблиця Transaction

Таблиця Category зберігає назви категорій витрат та доходів (Рис. 3.9).

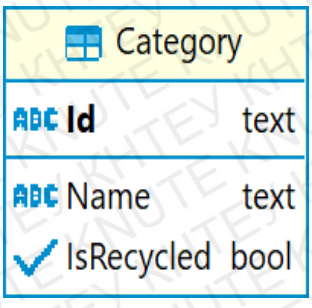

Рис. 3.9 Таблиця Category

У таблиці TransactionCategory фіксується множинний зв'язок між категорією та транзакцією. Адже безліч транзакцій можуть мати безліч категорій (Рис. 3.10).

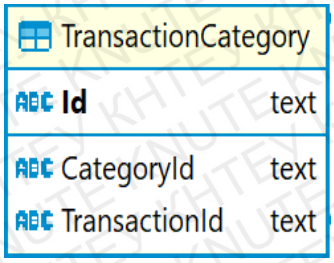

Рис. 3.10 Таблиця Transaction Category

Таблиця UserPlatform дозволяє зберігати інформацію про платформу, яку використовує користувач. Також містить унікальний ідентифікатор користувача з конкретної платформи. Це може бути ідентифікатор користувача у месенджері (Рис. 3.11).

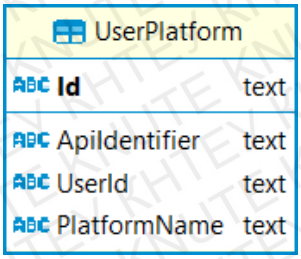

Рис. 3.11 Таблиця User Platform

У таблиці Location фіксуються координати користувача, та адреса, згідно цих координат (Рис. 3.12).

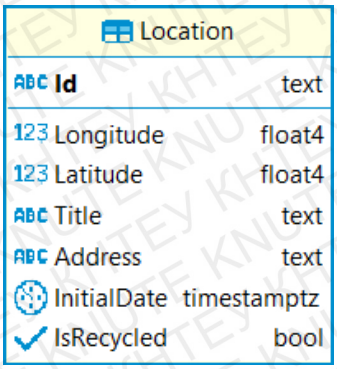

Рис. 3.12 - Таблиця Location

3.4. Реалізація основних класів та методів

Дана програмна система складається з двох серверних частин. Першою є бот до месенджера telegram. Іншою є реалізація фінансового асистента «Фінансисита».

У лістингу 3.1 зображено реєстрацію компонентів за допомого. DI, для подальшого їх використання у коді.

Лістинг 3.1 – Реєстрація компонентів системи:

```
    public class Startup
    {
    public void ConfigureServices(IServiceCollection services)
    {
    services.AddEntityFrameworkNpgsql();
services.AddDbContext<DataContext>( options => {
    options.UseNpgsql(ConnectionStringBuilder.Build()); },
ServiceLifetime.Transient, ServiceLifetime.Transient);

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion
.Vers ion_2_1);
    services.AddSingleton<IBotFrameworkHttpAdapter,
AdapterWithErrorHandler>();
    services.AddSingleton<IStorage, MemoryStorage>();
services.AddSingleton<UserState>();
services.AddSingleton<ConversationState>();
services.AddSingleton<ShoppingRecognizer>();
services.AddSingleton<BookingDialog>();
services.AddSingleton<ShoppingDialog>();
services.AddSingleton<MainDialog>(); services.AddTransient<IBot,
    DialogAndWelcomeBot<MainDialog>>();
    services.AddTransactionManagers();
    }

    public void Configure(IApplicationBuilder app,
IHostingEnvironment env)
    {

      app.UseHsts(); app.UseDefaultFiles(); app.UseStaticFiles();
                app.UseWebSockets(); app.UseMvc();
```

DI (Dependency injection) – шаблон проектування, який дозволяє додавати залежності класів у систему, та в потрібний момент отримати екземпляр цього класу [20]. Найчастіше даний принцип використовується для інверсії

залежностей завдяки абстракцій та інтерфейсів. За допомогою інтерфейсу IServiceCollection відбувається реєстрація нових залежностей. Існує 3 рівні, на яких можна зареєструвати залежності:

- Transient
- Scoped
- Singleton

Використовуючи перший рівень, при кожному звертанні до класу, буде створювати його новий екземпляр. Такий підхід буде гарним рішенням для сервісів, які не тримають у собі певний стан, або дані [21].

Другий випадок дозволить розробнику отримувати один і той же екземпляр класу при роботі в межах одного контексту [22].

В останньому випадку, сервіс реєструється лише як одна копія, і кожного разу буде повертатись єдиний екземпляр класу. Як видно з коду, усі діалогові вікна реєструються як singleton [23].

У лістингу 1.2 відображено конструктор головного діалогу. Як видно з коду, при створенні головного вікна, реєструються діалоги та задається їхній порядок виклику. Для діалогу типу «Водопад» зареєстровано вступний крок, основний, та крок для завершення діалогу з користувачем.

Лістинг 3.2 – Конструктор головного діалогу

```
    public MainDialog(ShoppingRecognizer luisRecognizer,
ShoppingDialog shoppingDialog, ILogger<MainDialog> logger)
    : base(nameof(MainDialog))
    {
    _luisRecognizer = luisRecognizer; Logger = logger;

    AddDialog(new TextPrompt(nameof(TextPrompt)));
AddDialog(shoppingDialog);
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog),
    new WaterfallStep[]
    {
    IntroStepAsync, ActStepAsync, FinalStepAsync,
    }));
```

```
        InitialDialogId = nameof(WaterfallDialog);
        }
```

Лістинг 3.3 відображає зразок конструктора діалогу, для визначення покупки. Тут також додаються кроки для діалогів, та для розпізнання дати. Для типу «Водопад» першим зареєстровано крок для визначення типу товару. Наступним йде крок для визначення ціни та дати транзакції. Після проходження цих кроків, наступним буде підтвердження інформації, що була розпізнана, та фінальний крок.

Процес виклику діалогів, відбувається наступним чином:

1. Головний діалог відправляє інформаційне повідомлення, для введення даних користувачем.

2. Після введення інформації, відправляється запит до сервісу Luis, який розпізнає сутності, та повертає відповідь.

3. У разі виявлення, що не вистачає інформації, запускається діалог типу «Водопад», який просить поетапно уточнити дані.

4. Після отримання усіх даних, відправляється запит для підтвердження виявлених сутностей.

5. Якщо все вірно, формується транзакція, та додається до бази даних.

6. При потребі, є можливість додати до транзакції файл, або локацію.

Лістинг 3.3 – Конструктор діалогу покупок

```
        public ShoppingDialog()
            : base(nameof(ShoppingDialog))
        {
            AddDialog(new TextPrompt(nameof(TextPrompt)));
            AddDialog(new ConfirmPrompt(nameof(ConfirmPrompt)));
            AddDialog(new DateResolverDialog());
            AddDialog(new WaterfallDialog(nameof(WaterfallDialog),
  new WaterfallStep[]
            {
                CommodityStepAsync,
                PriceStepAsync,
                PurchaseDateStepAsync,
                ConfirmStepAsync,
                FinalStepAsync,
            }));


            InitialDialogId = nameof(WaterfallDialog);
        }
```

Наприклад, користувач не уточнив ціну та час, коли відбулась транзакція. У цьому випадку для користувача буде відправлено інформаційне повідомлення, яке попросить ввести ціну, за якою було куплено товар. За це відповідає Shopping Dialog та крок для визначення ціни. Після цього ініціалізується наступний крок, для визначення часу. Цей крок очікує, що у параметри йому прийде ціна. Якщо буде виявлено, що ціну введено, буде відправлено повідомлення, для уточнення дати, та ініціалізація наступного кроку. Таким чином працюють діалоги у системі.

Для публічного API використовується BotController (див. лістинг 3.4), який завдяки адаптеру передає вхідний текст на подальшу обробку.

Лістинг 3.4 – API для повідомлень

```
    Route("api/messages")] [ApiController]
    public class BotController : ControllerBase
    {
    private readonly IBotFrameworkHttpAdapter _httpAdapter;
private readonly IBot _bot;

    public BotController(IBotFrameworkHttpAdapter httpAdapter,
IBot bot)
    {
    _httpAdapter = httpAdapter;
    _bot = bot;
    }

    [HttpPost, HttpGet]
    public async Task ProcessMessageAsync() => await
    _httpAdapter.ProcessAsync(Request, Response, _bot);
    }
```

Наступним фрагментом коду є опис частини, що відповідає за роботу бота (див. лістинг 3.5).

Bot Manager – клас, який відповідає за підтримання з'єдання клієнта з сервером месенджера. Для створення самого клієнта, йому потрібно передати в параметри конфігурацію, яка містить інформацію про токен бота, адресу та порт. Сам токен генерується при реєстрації бота. Також, під час створення екземпляру

клієнта відчувається підписка на нові події.

Лістинг 3.5 – Клієнт для telegram API

```
    private readonly TelegramBotClient _botClient;

    public event EventHandler<MessageEventArgs> OnNewMessage;
public event EventHandler<CallbackQueryEventArgs>
    OnCallbackQuery;

    public BotManager(IOptions<BotSettings> config)
    {
    _botClient = new TelegramBotClient(config.Value.BotToken);
    Subscribe();
    _botClient.StartReceiving(Array.Empty<UpdateType>());
    }

    private void Subscribe()
    {
    _botClient.OnMessage += BotClientOnMessageReceived;
    _botClient.OnCallbackQuery += BotClientOnCallbackQuery;

    _botClient.OnInlineQuery += BotClientOnInlineQuery;
    _botClient.OnUpdate += BotClientOnUpdate;
    _botClient.OnInlineResultChosen +=
BotClientOnInlineResultChosen;
    }
```

3.5. Розгортання програмної системи та системні вимоги

Так як програмний продукт написаний но мові програмування C# з використанням технології ASP NET Core, це дозволить розгорнути ПЗ на платформі Docker.

Docker - це технологія, яка дозволяє розробникам керувати програмним забезпеченням у будь-якому місці. Docker у своїй роботі застосовує контейнери, тому для подальшого розуміння його роботи, потрібно розглянути як вони працюють. [24].

Контейнер - це процес в операційній системі, який є ізольовано від інших процесів. Під час створення контейнера, йому виділяються ресурси, до яких доступ має лише він. В подальшій роботі ці ресурси можна відкрити і для інших

процесів. Процеси, що запущені не в межах контейнеру, можуть отримати доступ до файлів або процесів операційної системи, що було досить не надійним рішенням.

Правильно налаштований контейнер не зможе змінити щось в операційній системі, що додає безпеки та надійності. Перевагами використання Docker є те, що він дозволяє усунути залежності, для запуску програми. Під час побудови контейнера розробник вказує усі залежності та деталі, які повинні завантажитись в контейнер під час його розгортання. Це гарантує, що те, як працює програма у розробника локально, аналогічно буде працювати і на сервері.

Також, налаштувавши контейнер, ви можете відправити його іншій людині, яка відразу може розпочати його розгортання. Docker має відкритий код, та для початку роботи з ним, потрібно мати комп'ютер з підтримкою Virtualbox.

Контейнери використовуються для збільшення продуктивності комп'ютера, без збільшення кількості апаратного забезпечення. Раніше, масштабування сайту, потрібно було орендувати більшу кількість серверів, або покращувати поточне обладнання, але з приходом контейнері є можливість на тій самій апаратній складові запустити додаткові сервіси (див. рисунок 3.13).

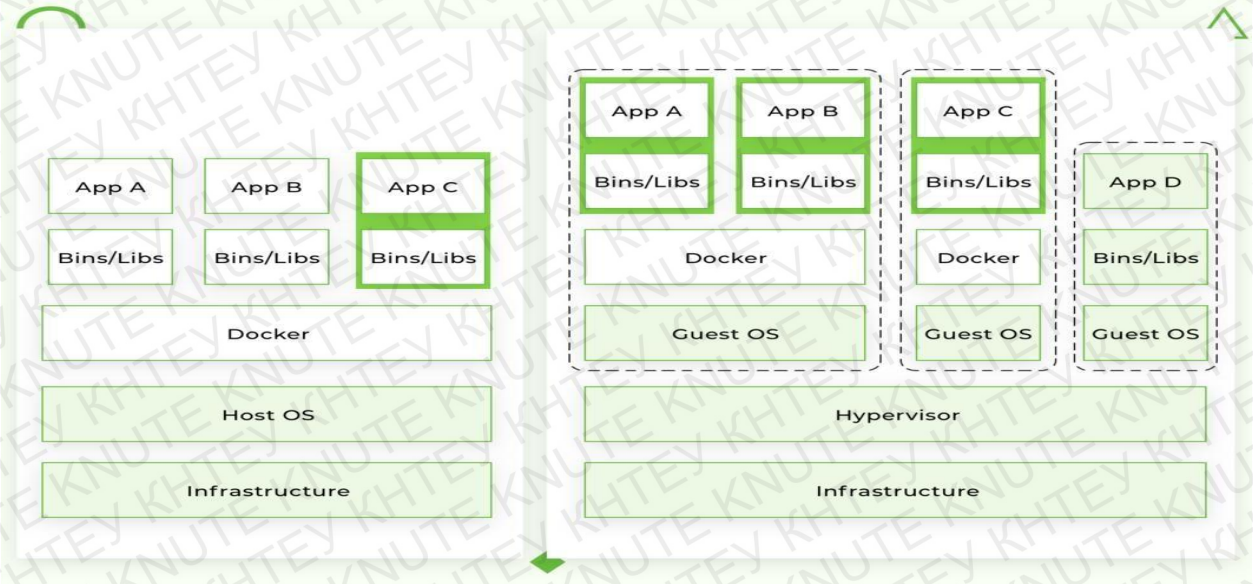

Рис. 3.13 – Схематичне зображення процесів у системі

Для побудови контейнерів використовується dockerfile з командами конфігурації.

Якщо потрібно розгортати та керувати великою кількістю контейнерів, то використання лише однієї docker платформи може виявитись недостанім. Тому що це може зайняти велику кількість часу. Для цього можна використати утиліту Docker compose. Вона призначена для керування багатьма контейнерами одночасно. Всі налаштування прописуються у файлі з розширенням yml. Ці правила є зрозумілими для розробника та оптимізовані для машин. Вони дають нам ефективний спосіб керувати всіма проектами за допомогою кількох рядків. Практично кожна команда інтерпретується до команди у платформі Docker.

Docker compose надає можливість використання імені проектів для ізоляції їх одне від одного, назва проекту - це ім'я каталогу, який містить проект [25]. У додатку для асистента, назва контейнера буде наступною: cash-assistant. Це і буде відповідати назві папки. Також є можливість вказати власну назву проекту, використовуючи атрибут -p, а потім власне ім'я.

Docker compose зберігає всі віртуальні диски, що використовуються у проектах, які є визначеними у файлі конфігурації, тому дані не втрачаються, коли контейнери перезавантажуються з використанням команд docker compose. Ще одна особливість полягає в тому, що ще раз створюються лише ті контейнери, які змінилися, а ті, стан яких не змінився, залишаються.

На Рисунку 3.14 зображено dockerfile для побудови релізної версії контейнерів. Він дозволяє знайти усі проекти в папці, та побудувати вихідні файли, що в подальшому, будуть використовуватись для розгортання на сервері.

```
FROM mcr.microsoft.com/dotnet/core/sdk:2.2 AS publish
WORKDIR /src
COPY . .
RUN for proj in $(find Server -name "*.csproj" -type f); do \
        if [ -f "$(dirname $proj)/Startup.cs" ]; then \
            dotnet publish $proj -c Release -o "/app/$(basename $proj .csproj)"; \
        fi; \
    done; \
```

Рис. 3.14 – Dockerfile для побудови релізної версії

Для побудови контейнерів, використовується наступна конфігурація (див. рисунок 3.15)

Де FROM – це налаштування базового контейнера, що буде використовуватись для розгортання. WORKDIR вказує робочий каталог. EXPOSE дозволяє вказати порти, що будуть відкриті для зовнішнього підключення. COPY це команда, буде виконувати копіювання готових файлів програми до контейнера. ENTRYPOINT дозволяє вказати, що саме буде виконувати контейнер після його запуску.

```
FROM microsoft/dotnet:2.2-aspnetcore-runtime-alpine AS base

WORKDIR /app
EXPOSE 23014
COPY --from=assistant.build /app/Assistant.Services.Telegam-bot .


ENTRYPOINT ["dotnet", "Assistant.Services.Telegam-bot.dll"]
```

Рис. 3.15 – Приклад конфігурації docker файлу

Приклад docker compose файлу зображено на рисунку 3.16. Де вказуються назви контейнерів, шлях до файлів конфігурацію docker, та усі залежності.

```
version: "3"

services:
  assistant.build:
    image: assistant.build
    build:
      context: .
      dockerfile: Dockerfile

  assistant.services.telegram-bot:
    image: assistant.services.telegram-bot
    build:
      context: .
      dockerfile: Server/Services/Assistant.Services.Bot/Dockerfile.txt
    depends_on:
      - assistant.build

  assistant.services.core-bot:
    image: assistant.services.core-bot
    build:
      context: .
      dockerfile: Server/Services/Assistant.Services.Core-bot/Dockerfile.txt
    depends_on:
      - assistant.build
```

Рис. 3.16. Конфігурація docker compose

До базових вимог системної програми можна віднести:

1. Надійність;

2. Продуктивність;

3. Обмеження проектування;

4. Структурність;

5. Безпеку.

Нижче перелічено детальний опис вимог, яким повинна відповідати система.

Підтримка різноманітних платформ. Програмна система повинна уміти проводити інтеграцію з усіма сучасними месенджерами, що дозволить запускати її на усіх сучасних операційних системах.

Можливість залишити відгук. Користувач повинен мати можливість залишити відгук, про вдосконалення, або технічні проблеми системи.

Безперебійна робота. Система повинна працювати на протязі повного робочого дня. Для оновлень системи та проведення технічних робіт не повинно затрачатись часу більше ніж 5% від загального часу роботи. Час для безвідмовної роботи повинен бути не менше ніж 30 робочих днів.

Одночасно працюючі користувачі. Програмна частина повинна підтримувати мінімум 50 одночасно працюючих користувачів.

Час відгуку. Час відгуку не повинен перевищувати 1 секунди, для опрацювання введених даних. Та до 5 секунд при генерації звіту.

Оновлення версій. Оновлення версій повинне проходити з використанням системи контролю версій та у автоматичному режимі.

Вимоги до апаратної частини. Система повинна підходити мінімальним параметрам сервера:

– Процесор 1.6 ГГц
– Оперативна пам’ять 4 ГБ
– Жорсткий диск 20 ГБ
– Підтримка віртуалізації

Також інформаційна система повинна бути реалізована у вигляді модульної структури, паролі в базі даних повинні зберігатися в захищеному вигляді.

Базою даних повинна бути СУБД PostgeSQL. Використовувана БД повинна бути приведена до третьої нормальної форми.

3.6. Опис типових схем використання системи

На Рисунку 3.17 зображено діаграму послідовностей для формування звіту по витратах. Спочатку користувач формує запит, у якому вказує, який тип звіту йому потрібен, та за який період часу. Після цього, за допомогою бота, команда відправляється на сервер асистента. Сервер передає вхідний текс для аналізу за допомогою LUIS. Після відповіді визначаються параметри, для формування звіту. З бази даних витягується відповідна інформація, на основі якої будується звіт вибраного формату. Наступним етапом є формування файлу зі звітом, який через бота передається до користувача.

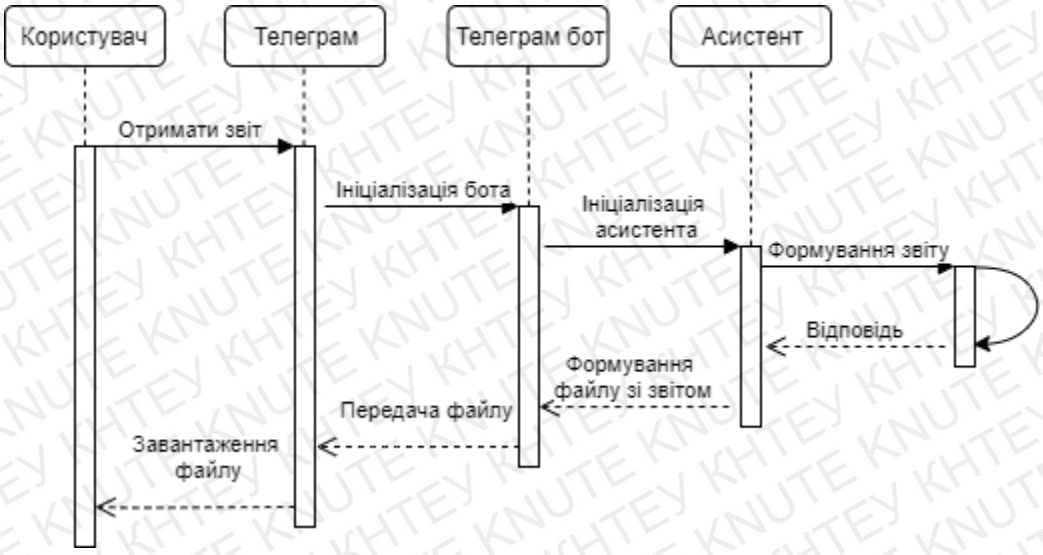

Рис. 3.17 – Діаграма послідовностей отримання звіту

На рисунку 3.18 зображена діаграма послідовності процесу внесення покупки.

Спочатку користувач у чаті месенджера вказує свою покупку. Після цього, телеграм відправляє повідомлення користувача до боту, який, в свою чергу, відправляє його до асистента. На стороні асистента визначається мова, на якій написано дане повідомлення. У разі виявлення мова, що не підтримується когнітивними сервісами, відбувається запит на переклад. Після отримання відповіді, текст перекладеного повідомлення відправляється до когнітивних сервісів luis. Після аналізу, результат повертається до асистента, який перевіряє, чи визначено усі потрібні дані для внесення запису до бази даних. У разі відсутності хоча б одного з них, до клієнта відправляється повідомлення, яке містить інформацію для уточнення даних. Після того, як користувач відправить доповнення, виконується аналогічний ланцюжок викликів. Якщо буде визначено усі потрібні дані, асистент додає новий запис у базу даних, та повертає відповідь користувачеві про успішну операцію.

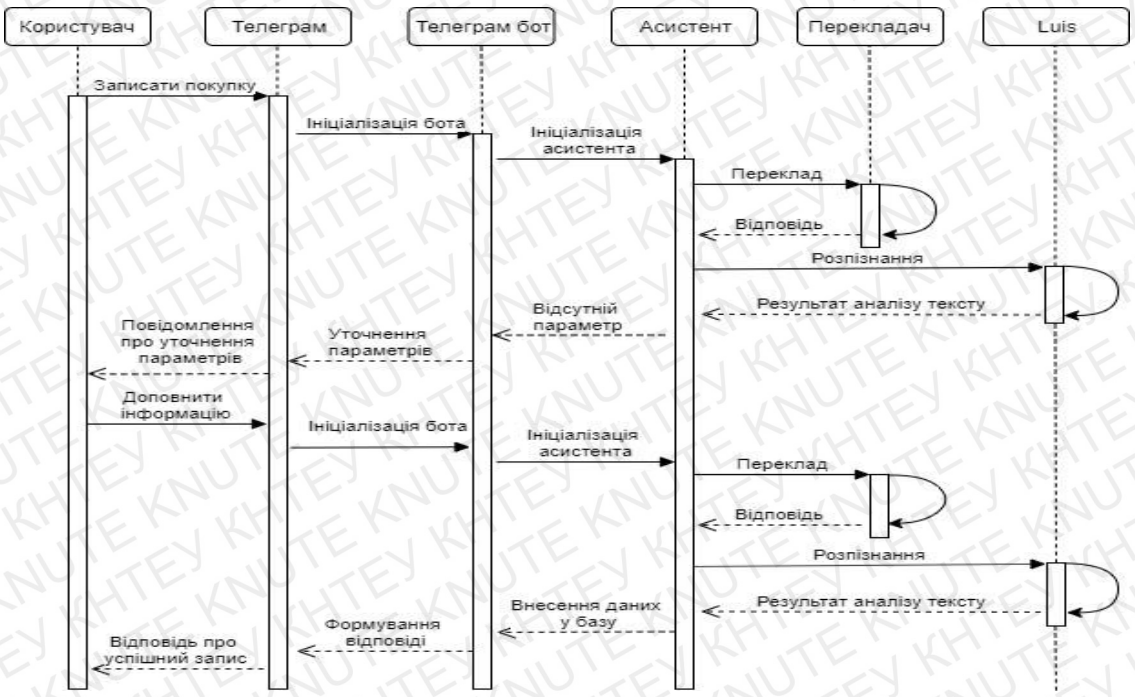

Рис. 3.18 Діаграма послідовностей для реєстрації покупки На рисунку

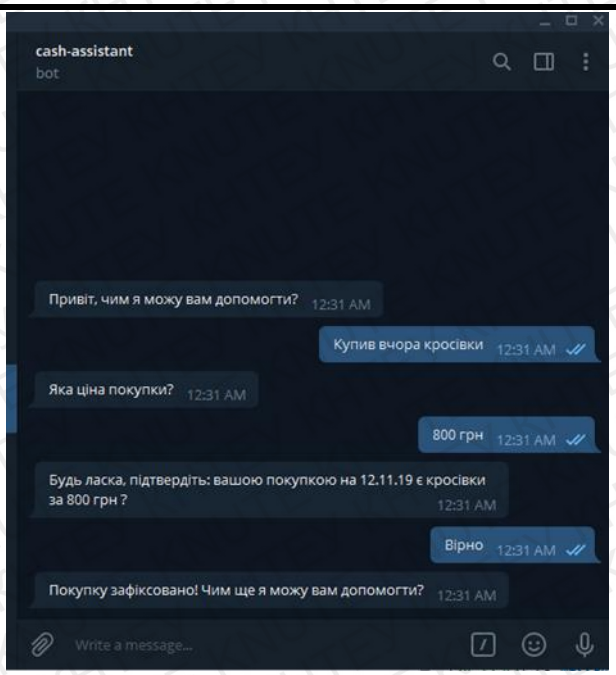

Рис. 3.19 Запис покупки

Для отримання звіту, потрібно ввести команду, як показано на рисунку 3.20. Після його генерації та отримання у повідомленні, його можна відкрити у ПЗ, що призначене для роботи з файлами .xlsx.

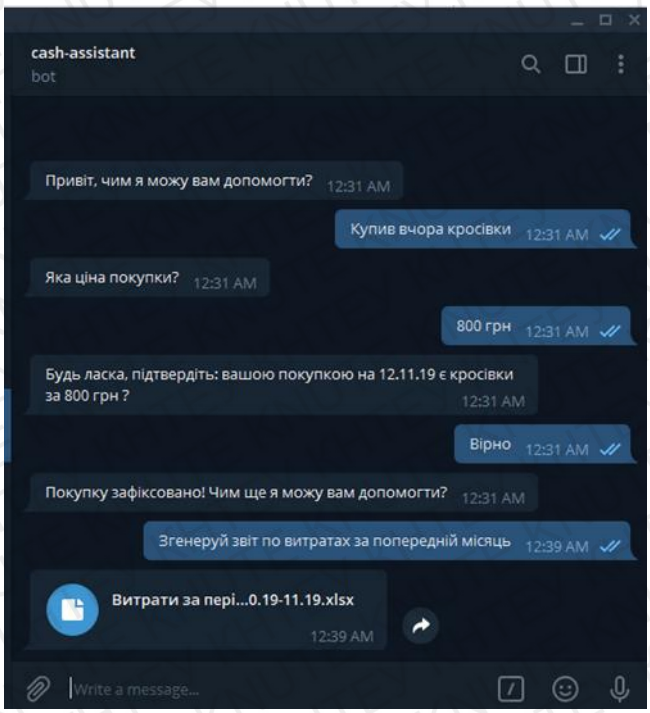

Рис. 3.20 Генерація звіту

Рисунок 3.21 містить інтерфейс при роботі з асистентом з смартфону. Порівнявши його з ПК версією, можна відмітити, що користування асистентом є однаковим з усіх платформ.

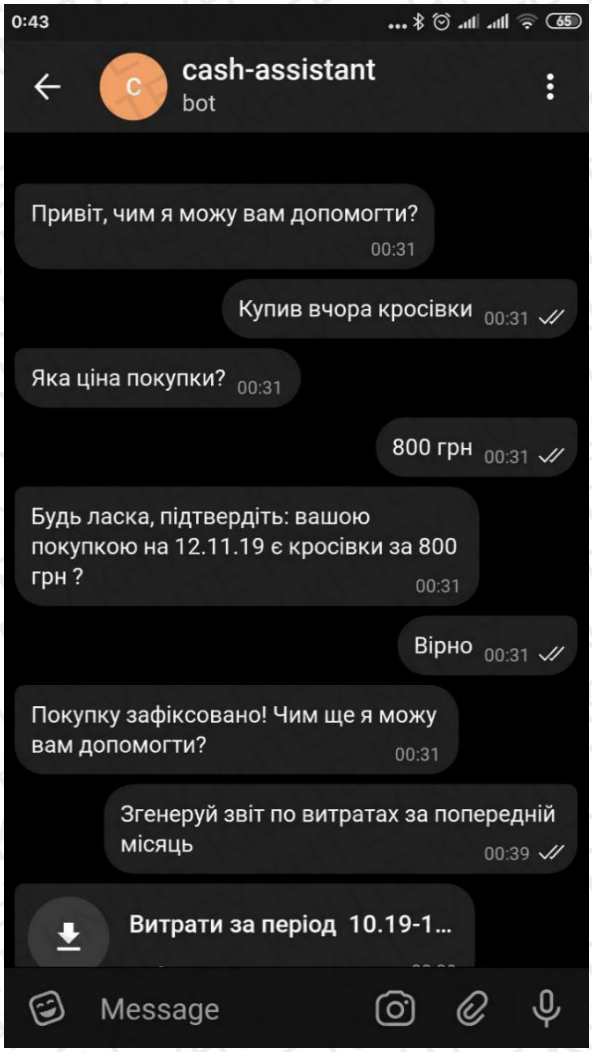

Рис. 3.21 Робота з асистентом у мобільному додатку

3.7. Тестування програмної системи

Для перевірки якості продукту застосовується процедура тестування. Враховуючи, що для розробки даної системи обрано ітераційний підхід, то тестування повинне проводитись після завершення кожної ітерації.

Тестування поділяється на ручне та автоматизоване.

Ручне тестування виконується членом команди, що відповідає за тестування особисто. Для цього потрібно виконати сценарії тестів вручну,

використовуючи додаток або його публічне API. Також для цього можуть використовуватись допоміжні інструменти. Зазвичай, таке тестування витрачає багато людських ресурсів. Для початку тестування потрібно налаштувати на розгорнути усі сервіси. Також під час тестування буде присутнім людський фактор, що може привести до помилки у сценарії, або до проблеми, що була не поміченою [26].

У свою чергу, автоматизоване тестування виконується на рівні коду, зазвичай, перед розгортанням нової версії. Автоматизовані тести можуть мати різноманітне призначення. Вони можуть тестувати окрему частину коду, або систему в цілому. Даний тип тестів застосовується для знаходження проблем як на серверній, так і на клієнтській частину. Це є дуже надійний спосіб перевірки загального стану системи, чи готова вона переходити на наступний етап, чи потрібен ще деякий час, для виправлення помилок. Але для написання автоматизованих тестів потрібен певний навичок, та розуміння процесу тестування. Тому в більшості випадків, самі розробники огортають свій код тестами. Автоматичне тестування застосовується в підході безперервної доставки коду.

Також тести поділяються на різні типи, щодо елементів, що підлягають перевірці.

Unit тестування написане та використовується для тестування модулів в самому коді розробника. За допомогою даного типу тестів можна буде впевненим, що конкретні методи працюють вірно. Зазвичай, даний тип тестування не вимагає витрати багатьох ресурсів при розробці та запуску.

Інтеграційне тестування перевіряє, чи модулі програми працюють вірно. Ці тести виконуються за рахунок API додатку. Наприклад, перевірка взаємодії з БД. Вимагають більше ресурсів для запуску, тому що вимагають повного розгортання модулів програмної системи [27].

Для створення тестів у середовищі Visual Studio потрібно в пошуку

параметри для пошуку, та обрати тип проекту для тестування (див. рисунок 3.22).

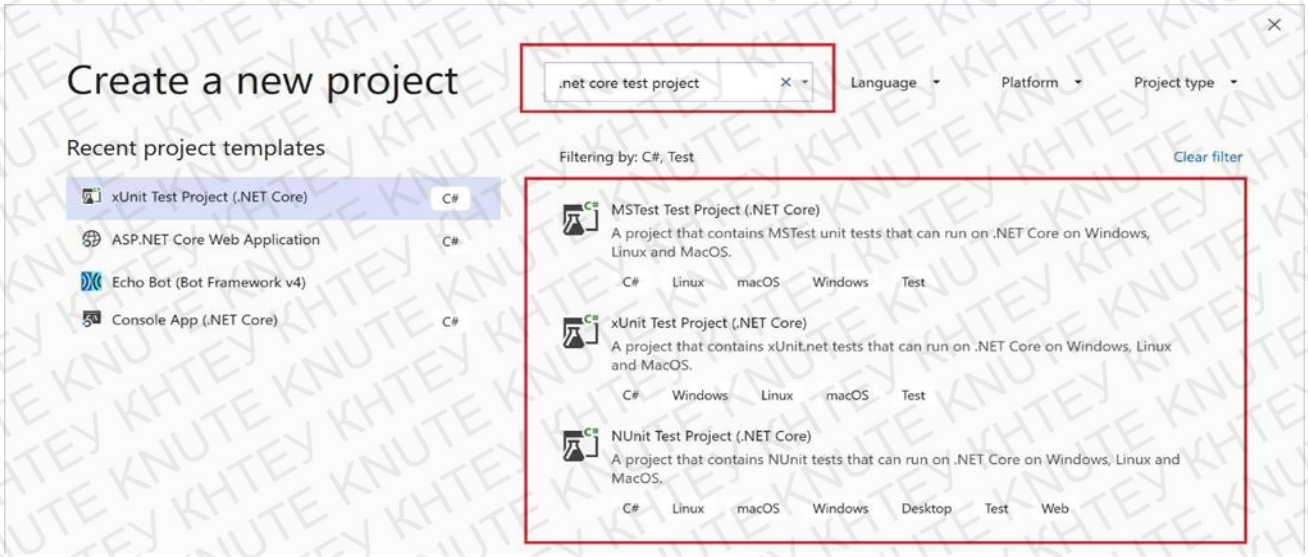

Рис. 3.22 Процес створення проекту для тестування

Після запуску тестів, можна спостерігати за їх прогресом виконання, та дізнатись, які не завершились успішно (Рис. 3.23).

Також тести можуть запукатись при розгортанні нових версій програмного забезпечення на віддалених серверах. При не успішному проходження, нова версія не буде розгорнута.

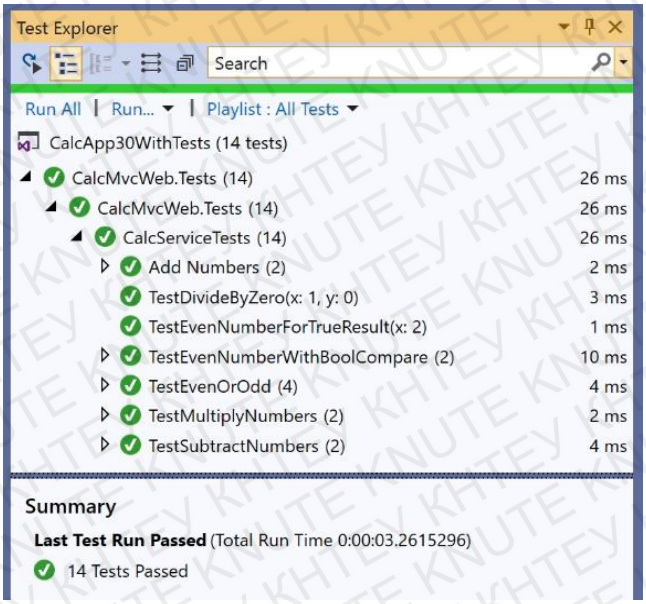

Рис. 3.23 Процес проходження тестів

У разі виявлення помилок, та після їх усунення є можливість ще раз запустити окрему групу тестів.

3.8. Огляд та реєстрація бота у месенджері Telegram

Як було сказано, у межах даної програмної системи було створено два додатки. Першим є бот для месенджера. Але для того, щоб отримувати дані на серверну частину бота, спочатку його потрібно зареєструвати.

Для прикладу було взято досить популярний месенджер telegram. Telegram – месенджер, що підтримує усі актуальні платформи, та дозволяє пересилати повідомлення та файли багатьох форматів. Окрім базового функціоналу месенджера підтримує створення та керування ботами, серверна частина яких перебуває у розпорядженні розробників [28].

Telegram є досить надійним месенджером, завдяки протоколу передачі даних MTProto він використовує декілька протоколів шифрування. RSA-2048 для авторизації, та DH-2048 для шифрування. Під час передачі по мережі, повідомлення шифруються алгоритмом AES, та генерується ключ, що є лише у клієнта та сервера.

Також у месенджері існують секретні чати. Відмінністю від звичайних є те, що повідомлення, під час передачі по секретному чату, не розшифровуються на сервері, а історія листування зберігається лише на клієнтських.

Бот у месенджері telegram являє собою різновид чат-боту.

Для реєстрації бота, потрібно знайти чат з ідентифікатором @botfather, в якому завдяки команд можна керувати вже створеними ботами, або додати новий (Рис. 3.24).

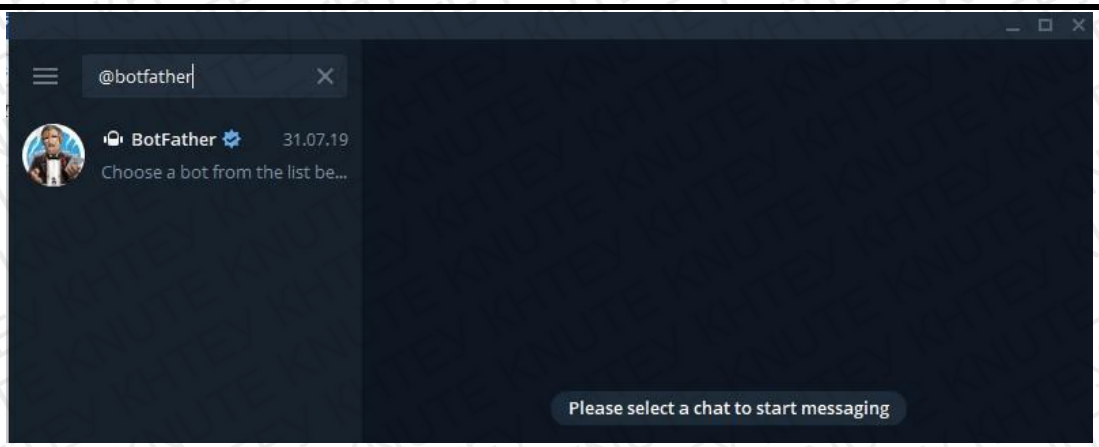

Рис. 3.24 Пошук каналу керування ботами

Далі потрібно вказати команду /newbot, ввести назву бота, та вказати його ідентифікатор. Весь процес створення відображено на рисунку 3.25.

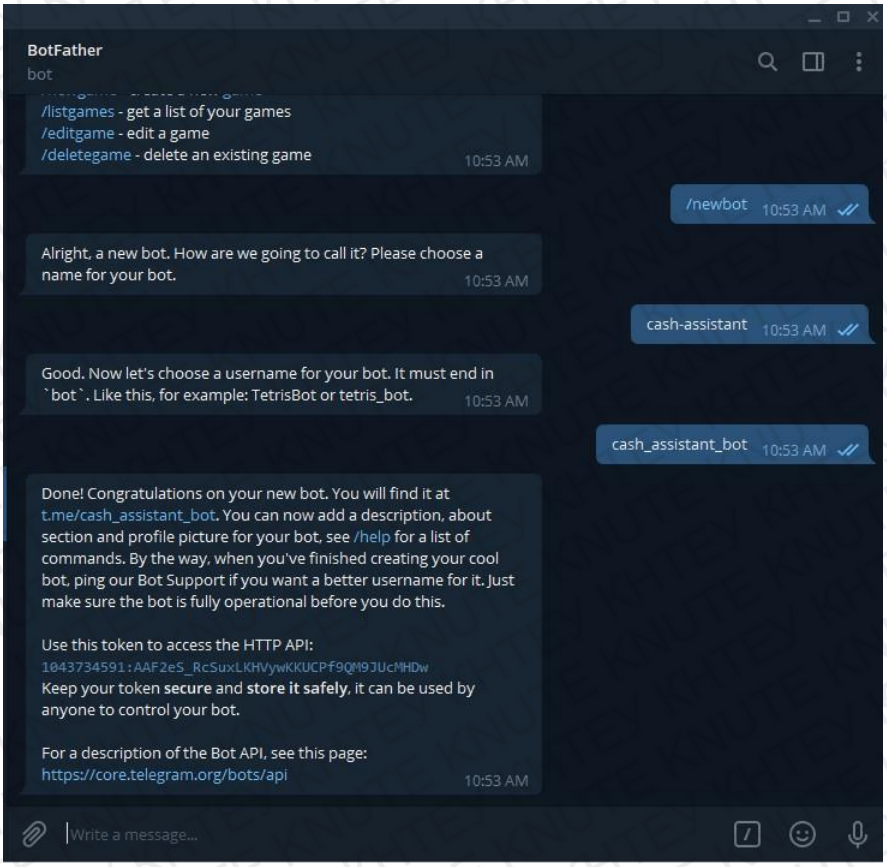

Рис. 3.25 Процес реєстрації бота

3.9 Огляд та реєстрація когнітивних сервісів LUIS

LUIS - це API сервіс, який знаходиться на серверах Microsoft Azure, та містить у собі спеціальний алгоритм машинного навчання, що дозволяє розпізнавати розмовну мову, та дозволяє розпізнати з повідомлення загальний зміст тексту, та отримати задану інформацію з нього [29].

Нейронна мережа – набір алгоритмів, що імітуються роботу людського мозку. Під час своєї роботи використовують кластеризацію або маркування вхідних елементів. Результати аналізу представляються у вигляді векторів які містять числа. В подальшому ці вектори переводяться у кінцевий тип даних.

Використання класифікаторів дозволяє встановити зв’язки між вхідними елементами, за допомогою вхідних параметрів. Чим більше точніших параметрів буде додано, тим точніші будуть результати. Також, можливо визначити зв’язок між минулими подіями, та майбутніми. Чим точніший буде результат, тим більше шансів запобігти настанню певної події.

Для визначення активності, LUIS використовує логістичні регресійні класифікатори (LRC). Для визначення самих об’єктів використовуються умовні випадкові поля (CRF). [30]

Для розуміння, з якими вхідними та вихідними параметрами працює LUIS, потрібно розглянути наступний приклад. Користувач відправляє команду:

«Book me a flight to Cairo», що розпізнає та повертає сервіс LUIS зображено на рисунку 3.26. Даний результат перебуває у JSON форматі, та містить активності, які зареєстровані у системі, та з якою вірогідністю дана команда належить до однієї з них.

```
{
    "query": "Book me a flight to Cairo",
    "topScoringIntent": {
        "intent": "BookFlight",
        "score": 0.9887482
    },
    "intents": [
        {
            "intent": "BookFlight",
            "score": 0.9887482
        },
        {
            "intent": "None",
            "score": 0.04272597
        },
        {
            "intent": "LocationFinder",
            "score": 0.0125702191
        },
        {
            "intent": "Reminder",
            "score": 0.00375502417
        },
        {
            "intent": "FoodOrder",
            "score": 3.765154E-07
        },
    ],
    "entities": [
        {
            "entity": "cairo",
            "type": "Location",
            "startIndex": 20,
            "endIndex": 24,
            "score": 0.956781447
        }
    ]
}
```


Рис. 3.26 JSON формат розпізнаних даних

Іншим модулем системи є серверна частина, яка відповідає за отримання, зберігання та опрацювання інформації по обліку фінансів. Так як було сказано, команди користувача, за допомогою бота будуть приходити на сервер, який повинен розпізнати з тексту потрібну інформацію, та опрацювати її. Для розпізнання тексту використовуються когнітивні служби LUIS на платформі Microsoft Azure, можливості яких схематично зображено на рисунку 3.27.

Рис. 3.27. Схематичне зображення можливостей LUIS та платформи Microsoft Azure

Для роботи з сервісами LUIS потрібно пройти реєстрацію на платформі Microsoft Azure Portal. Наступним кроком є, є пошук потрібних сервісів, що показано на рисунку 2.28. Після цього потрібно створити підписку та додати новий ресурс. Створення нового ресурсу зображено на рисунку 3.29.

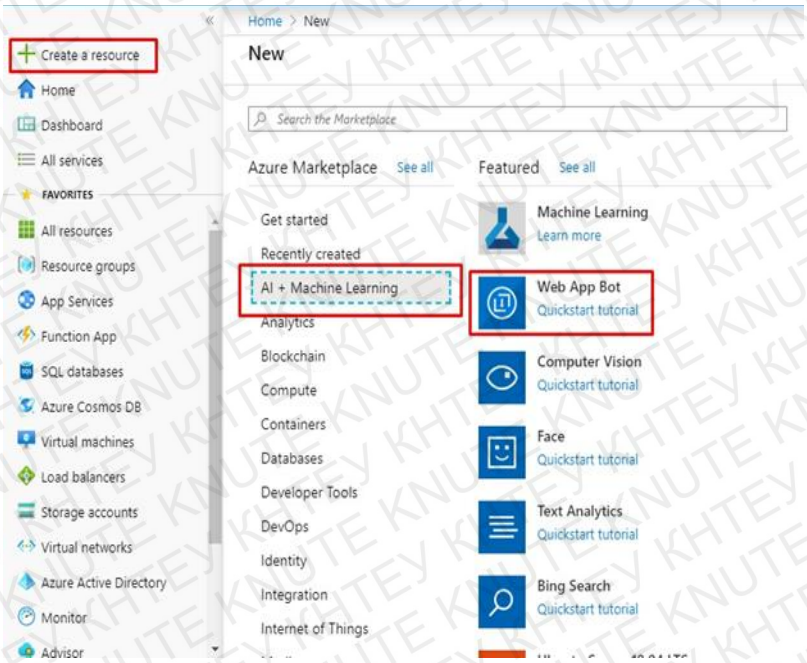

Рис. 3.28 Кроки для пошуку сервісів LUIS

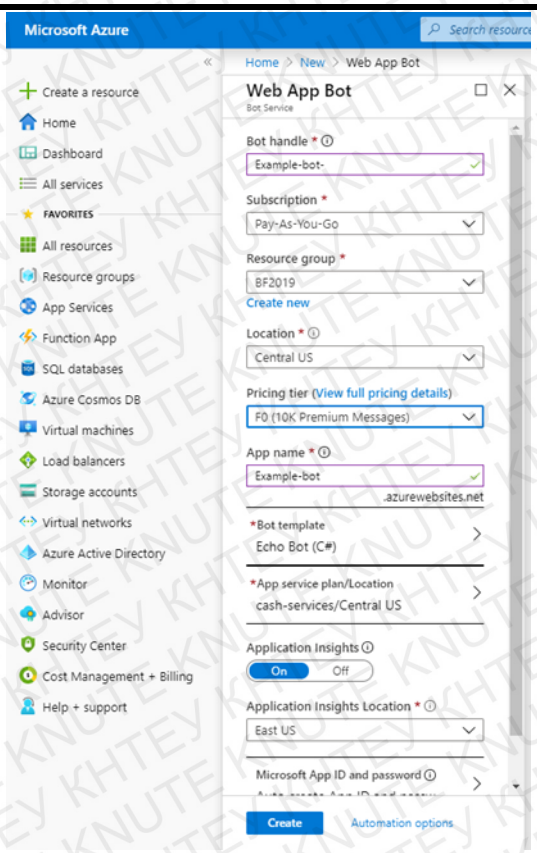

Рис. 2.29 Процес реєстрації нового ресурсу та додатку

Для того, щоб отримати, до якої активності відноситься вхідний текст, потрібно створити шаблони, на основі яких буде проводитись аналіз тексту. (Рисунок 3. 30).

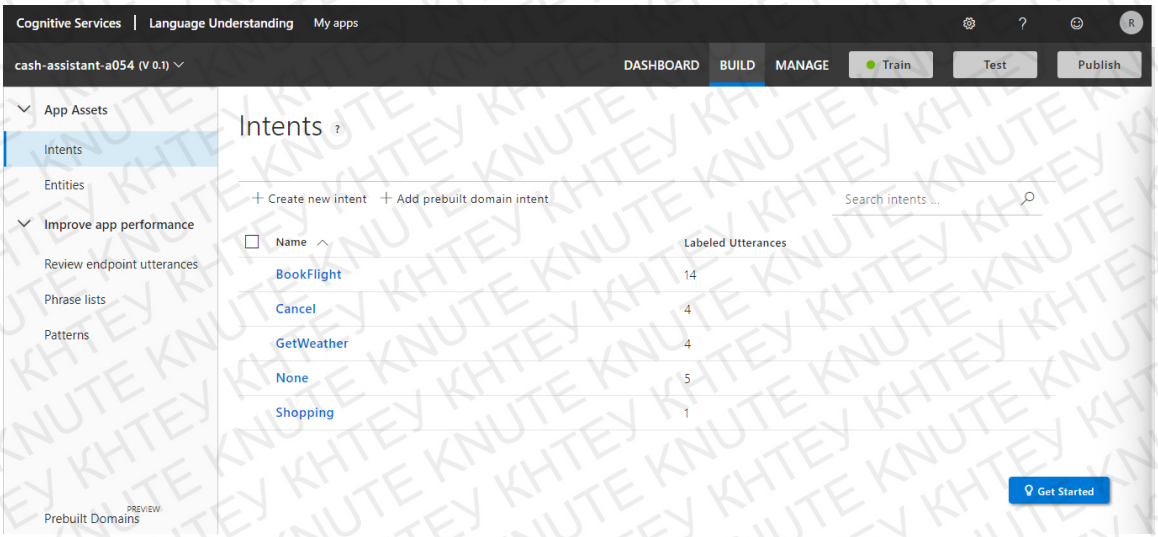

Рис. 3.30 Список налаштування активностей

На наступному етапі потрібно оголосити категорії, які будуть визначатись з контексту (Рисунок 3.31). У даному випадку це будуть категорії для можливих витрат, та доходів.

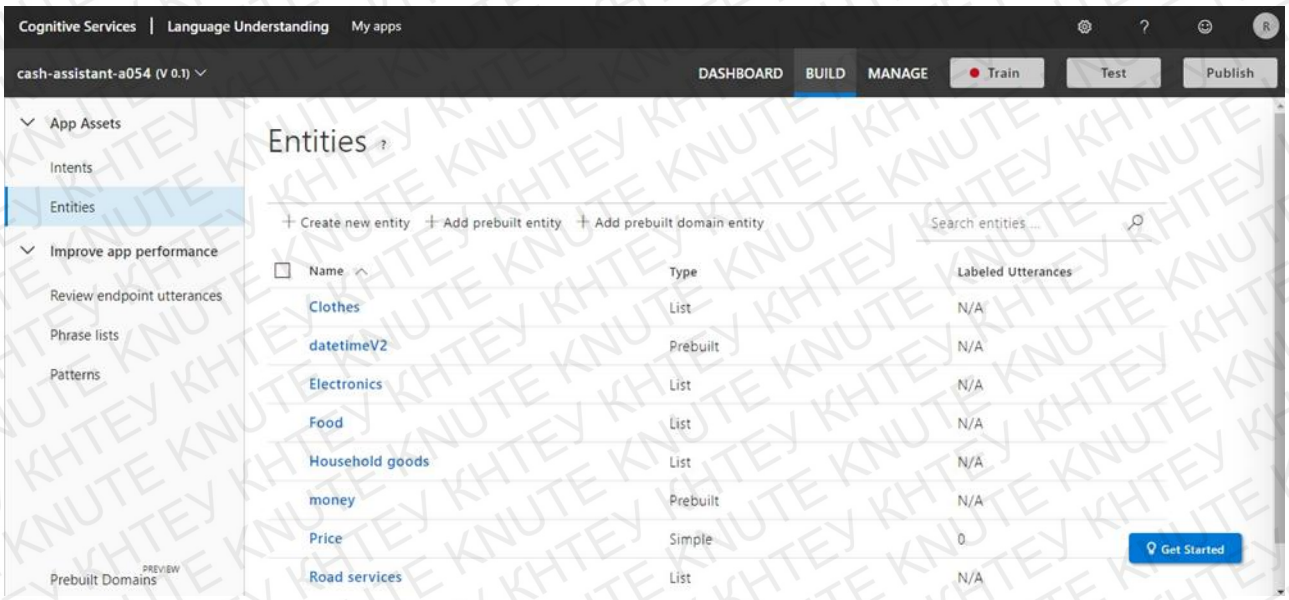

Рис. 3.31. Оголошення сутностей

Також кожну категорію потрібно заповнити сутностями з реального світу, для подальшого розпізнання. Їх перелік можна спостерігати на прикладі категорії «Електроніка» (Рисунок 3.32). Також, для спрощення налаштування, на основі вже введених сутностей система запропонує додати нові, що відповідають тематиці даній категорії.

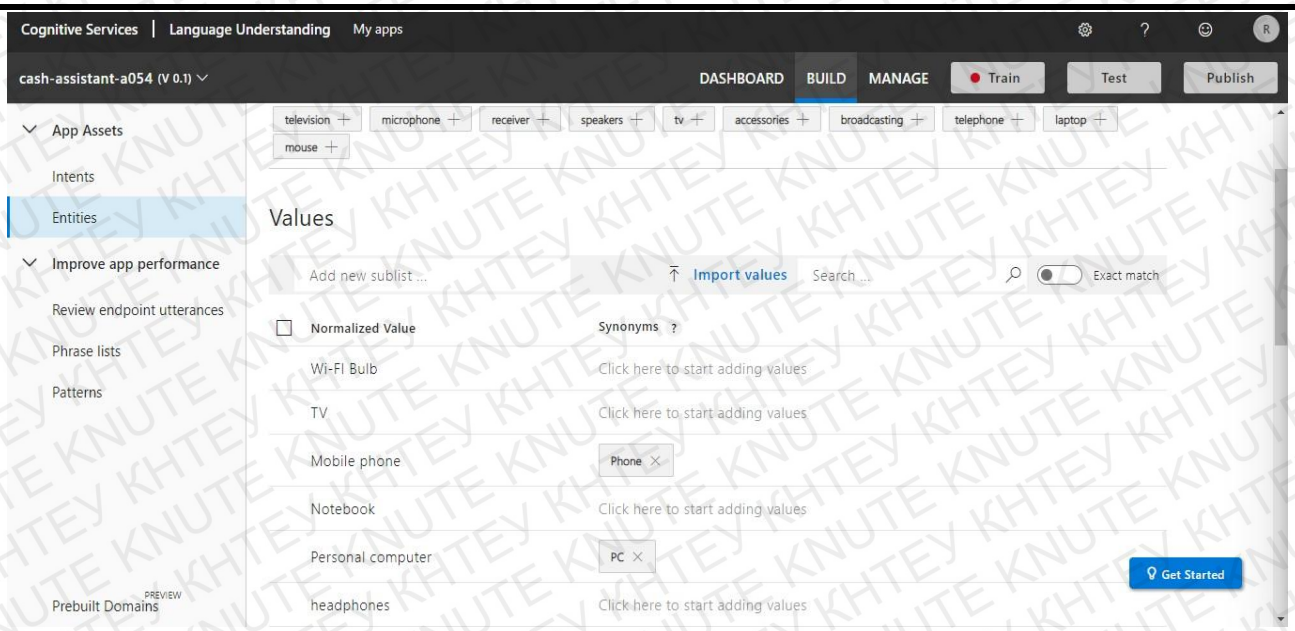

Рис. 3.32. Перелік об’єктів з шаблону «Електроніка»

Вказавши всі дані, потрібно описати приклади вхідних повідомлень, з використанням вказаних вище сутностей (див. рисунок 3.33). Когнітивні сервіси розпізнання тексту самі визначать, до якої з сутностей належить кожне слово.

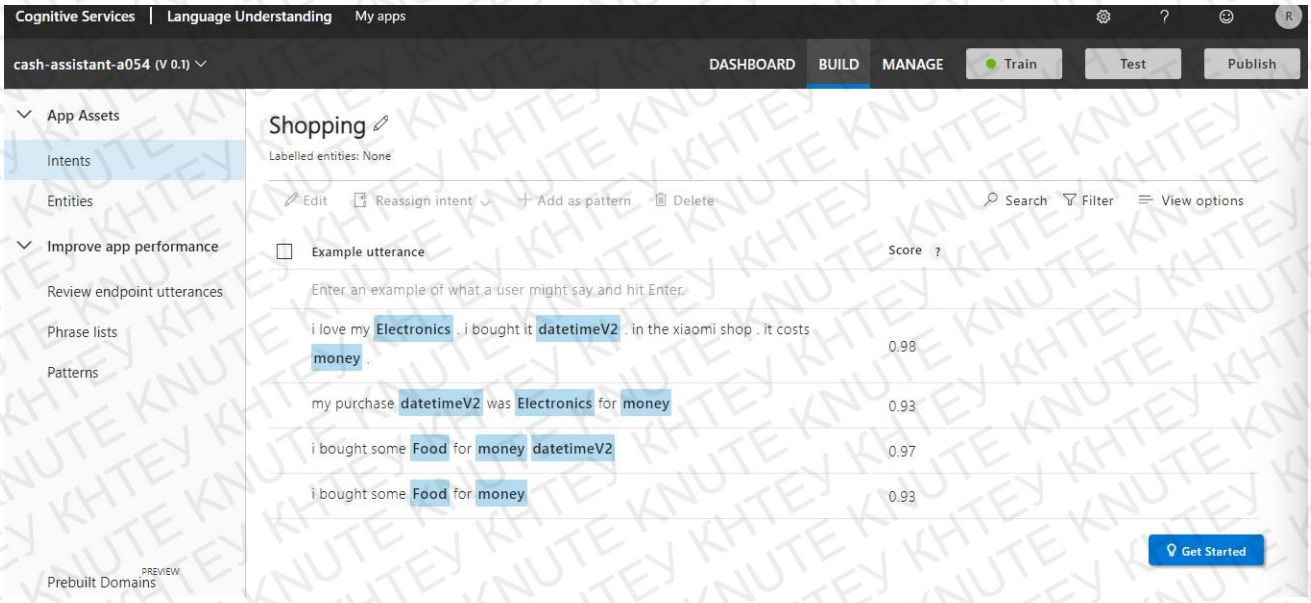

Рис. 3.33. Приклади вхідних даних у налаштуваннях активності

Після цього потрібно запустити процес навчання когнітивного сервісу (див. рисунок 3.34), який триватиме певний проміжок часу, та після завершення дозволить використовувати введені дані для визначення сутностей з вхідного тексту.

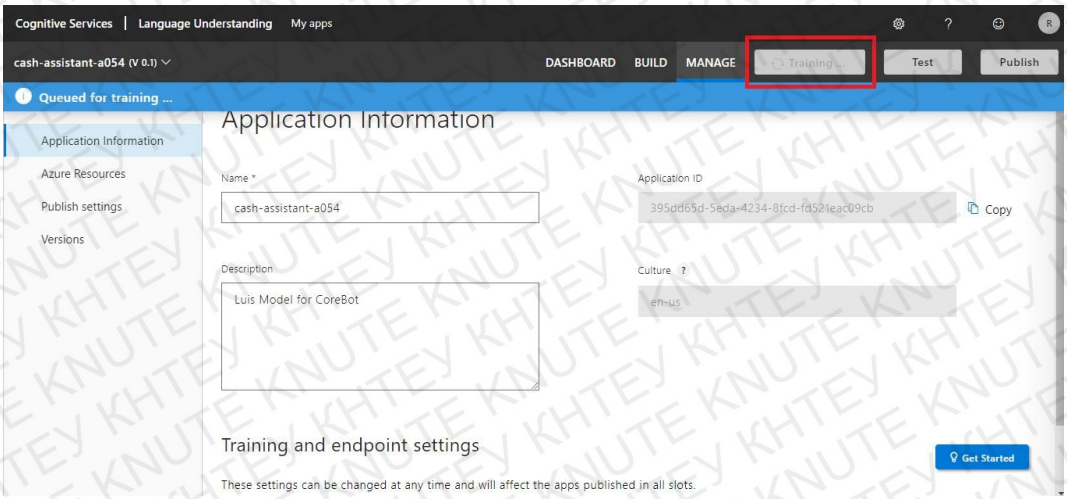

Рис. 3.34 Запуск процесу навчання когнітивних сервісів

Для перевірки результатів навчання на основі введених сутностей у платформі передбачено тестовий режим, що дозволяє аналізувати введений текст (див. рисунок 3.35). Даний режим визначить, до якого шаблону відноситься текст, та відобразить визначені сутності. Такий формат даних і буде повертатись у відповіді з серверу.

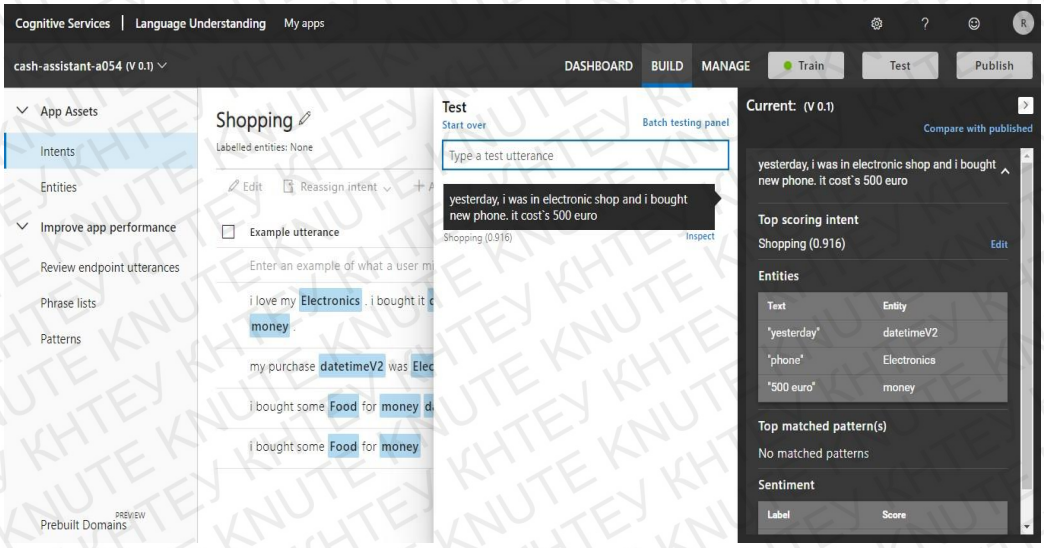

Рис. 3.35. Аналіз введених даних

На рисунку 3.36 графічно відображено кількість запитів до LUIS API.

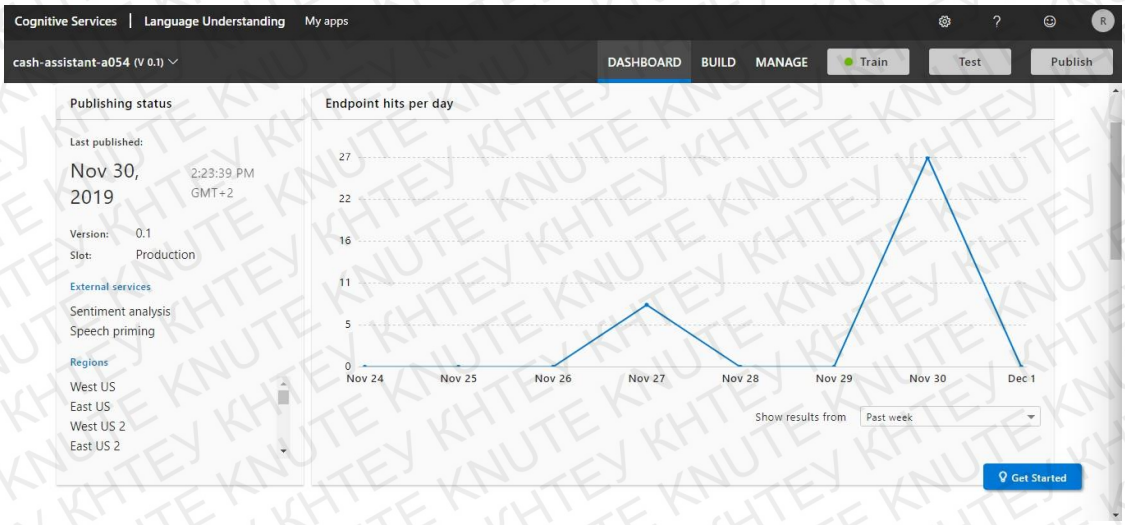

Рис. 3.36 Сторінка аналізу звертань до системи

3.10. Висновки до Розділу 3

В процесі моделювання та впровадження архітектури програмного продукту інтелектуальної системи «Фінансист» було зпроєктовану СУДБ та описано її фізичну модель, що дано можливість реалізувати основні класи та методи. Запропоновано та обґрунтовано яким чином має відбуватись розгортання програмної системи та описано системні вимоги. Наведено опис типових схем використання системи та показано, що одним з обов'язковим етапів створення програмного продукту є етап тестування. Щоб інтелектуальна система була працездатною її необхідно зареєструвати в одному з меседжерів та застосувати когнітивний сервісів LUIS.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

У наш час все більшої актуальності набуває проблема невміння контролювати свої витрати. Облік особистих фінансів відіграє важливу роль в сучасному житті. Це початкова і найважливіша ланка усіх етапів фінансового планування.

Облік особистих витрат потрібен для:

- оцінки коштів, що використовуються неефективно;
- виявити причини нестачі грошей, та знайти варіанти для їх вирішення;
- провести аналіз структури витрат з метою підвищення їх ефективності.

Щоденні витрати можна фіксувати у блокноті або ж у вигляді електронних таблиць. Але це не є зручним рішенням, адже не завжди є можливість носити їх з собою. У сучасному світі, альтернативою може виступати використання спеціальних електронних додатків. Проте, безкоштовні додатки мають обмежений функціонал та не мають підтримки усіх актуальних операційних систем.

Метою роботи є дослідження проблеми керування фінансами споживача та розробка програмної системи, яка завдяки когнітивним технологіям дозволить спросити фіксацію та аналіз витрат.

В ході виконання даної магістерської роботи було створено програмну систему для обліку фінансів користувача. Вдосконалено процес обліку та аналізу бюджету особи, за допомогою сучасного програмного забезпечення шляхом використання когнітивних технологій розпізнання розмовної мови. Запропонований підхід може бути інтегрований у відомі програмні системи, що призначені для спілкування користувачів та дозволить спросити процес облік і аналізу особистих витрат.

					КНТЕУ 121 02з-16.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		01.11.21	*Моделювання архітектури програмного продукту інтелектуальної системи «Фінансист»*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Криворучко О.В		*01.11.21*		*ВП*	*60*	*64*
Гарант		Токар В.В.		*01.11.21*		*Факультет інформаційних технологій 2з курс, 2м група*		
Розробив		Франчук Т.М.		*01.11.21*	*Висновки та пропозиції*			

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 45% всіх дорослих жителів України користуються тач-скрін смартфонами [Електронний ресурс]. – Режим доступу: https://sostav.ua/publication/kolichestvo-polzovatelej-smartfonov-v-ukraine- dostiglo-85-79227.html
2. Сімейний бюджет [Електронний ресурс]. https://simeinyi-budzhet.ua/moneyandlove/simeinyi-budzhet/
3. Money Lover, офіційний сайт [Електронний ресурс] - https://moneylover.me/
4. Bills Monitor [Електронний ресурс] - https://play.google.com/store/apps/details?id=com.swarooptech.billsfree&hl=en_US
5. Monefy, офіційний сайт [Електронний ресурс] - http://www.monefy.me/
6. CoinKeeper, офіційний сайт [Електронний ресурс] https://coinkeeper.me/
7. Goodbudget, офіційний сайт [Електронний ресурс] - https://goodbudget.com/
8. Ларман К. Ітеративна та інкрементальна розробка: коротка історія / К. Ларман, Ст. Базілю // Відкриті системи. — 2003.
9. An Entity Relationship Diagram Example. Demonstrates the crow's feet notation by way of an example [Електронний ресурс] - http://rapidapplicationdevelopment.blogspot.com/2007/06/entity-relationship-diagram-example.html
10. Head First SQL, O'Reilly Media, Inc. - https://learning.oreilly.com/library/view/head-first-sql/9780596526849/
11. ITS, University of Texas [Електронний ресурс] - ttps://web.archive.org/web/20100106115112/http://www.utexas.edu/its/archive/ windows/database/datamodeling/rm/rm7.htm

12. Базові знання з нормалізації баз даних- http://databases.about.com/od/specificproducts/a/normalization.htm
13. Третя нормальна форма [Електронний ресурс] - http://studepedia.org/index.php?vol=1&post=103485
14. Діаграма станів (UML) [Електронний ресурс] - http://doc.omg.org/formal/2009-02-02.pdf
15. Клієнт-серверні системи [Електронний ресурс] - https://stud.com.ua/97304/informatika/kliyent_serverni_sistemi
16. Клієнт-серверна архітектура [Електронний ресурс] - http://wikiinfo.mdpu.org.ua/index.php?title=Client-server-side
17. C# 8.0 in a Nutshell : The Definitive Reference, Joseph Albahari, 384 сторінки.
18. Mastering Visual Studio 2019, Kunal Chowdhury - 374 сторінки.
19. Офіційний сайт PostgreSQL [Електронний ресурс] - http://www.postgresql.org/
20. Dependency Injection in .NET — Mark Seemann, Manning, 2011
21. Dependency injection in ASP.NET Core [Електронний ресурс] - https://docs.microsoft.com/ru-ru/aspnet/core/fundamentals/dependency- injection?view=aspnetcore-3.1
22. Singleton-об’єкти і scoped-сервіси [Електронний ресурс] - https://metanit.com/sharp/aspnet5/6.5.php
23. Implementing the Singleton Pattern in C# [Електронний ресурс] - https://csharpindepth.com/articles/singleton
24. «Використання Docker» Едрієна Моуета, 2017, 354 сторінок
25. Docker Compose Starter Pack [Електронний ресурс] - https://dev.to/rohansawant/docker-compose-starter-pack-ubuntu-container-using- docker-and-docker-compose-4cl1
26. «Testing computer software» Cem Kaner, Jack L.Falk, 2005

27. Гнучке тестування: практичне керівництво для тестувальників ПЗ і гнучких команд Лайза Кріспін, Джанет Грегорі. М.: «Вільямс», 2010. - 464 сторінок.

28. Telegram, офіційний сайт - https://telegram.org/

29. LUIS, офіційний сайт - https://www.luis.ai/

30. What is Language Understanding (LUIS)? [Електронний ресурс] - https://docs.microsoft.com/en-us/azure/cognitive-services/luis/what-is-luis

31. Про авторське право і суміжні права [Електронний ресурс] - https://zakon.rada.gov.ua/laws/show/3792-12

32. International Valuation Standards [Електронний ресурс] - https://www.ivsc.org/standards/international-valuation-standards

33. СОБІВАРТІСТЬ ПРОДУКЦІЇ Юридична енциклопедія : [у 6 т.] / ред. кол. Ю. С. Шемшученко (відп. ред.) [та ін.] — К. : Українська енциклопедія ім. М. П. Бажана, 1998—2004.

34. Armstrong, Deborah J. (February 2006). The Quarks of Object-Oriented Development. Communications of the ACM 49 (2): 2006-08-08.

35. Правила безпечної роботи на комп'ютері [Електронний ресурс] - https://www.pedcollege.kiev.ua/index.php/77-robota-koledzhu/okhorona- pratsi/589-pravyla-bezpechnoi-roboty-na-kompiuteri

36. Стійкість функціонування об'єктів економіки в НС [Електронний ресурс] - https://studme.com.ua/1098120514395/bzhd/ustoychivost_funktsionirovaniya_ob ektov_ekonomiki.htm

37. Франчук Т. Використання когнітивних технологій в моделюванні архітектури програмного продукту"/ Збірник наукових статей студентів Software Engineering Комп'ютерних систем. КНТЕУ: Київ, 2021 - с. 175-181.

ТЕХНІЧНЕ ЗАВДАННЯ

1. ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до плану написання випускних кваліфікаційних робіт здобувачів освіти ос «магістр» спеціальності 121 «Інженерія програмного забезпечення» заочної форми навчання КНТЕУ.

Тема ВКР: «Моделювання архітектури програмного продукту інтелектуальної системи «Фінансист»

2. ПРИЗНАЧЕННЯ РОЗРОБКИ

Випускний кваліфікаційний рпроєкт присвячена створенню програмного забезпечення фінансового асистента, який інтегрується у месенджери, та дозволяє фіксувати витрати користувача.

Предметом дослідження є програмна система, що використовує когнітивні технології для розпізнання команд користувача.

Мета роботи спростити ведення особистого бюджету за допомогою програмної системи, яка завдяки інтеграції у месенджери, має можливість запускатись на будь-якій сучасній платформі.

За результатами виконаної роботи необхідно отримати програмне забезпечення яке дозволить користувачеві, використовуючи когнітивні технології розпізнання мови, фіксувати свої витрати та доходи у месенджері, та проводити їх аналіз.

					КНТЕУ 121 02з-16.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		18.10.21	*Моделювання архітектури програмного продукту інтелектуальної системи «Фінансист»*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Криворучко О.В.		*18.10.21*		*ТЗ*	*64*	*64*
Гарант		Токар В.В.		*18.10.21*		*Факультет інформаційних технологій 2з курс, 2м група*		
Розробив		Франчук Т.М.		*18.10.21*	*Технічне завдання*			

3. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Функціональні характерики.

Програмне забезпечення має виконувати наступні дії:

- Інтегруватись з месенджером у вигляді бота та отримувати команди від користувача;
- Дозволити вести керування витратами та доходами;
- Встановлювати інформаційний ліміт на витрати;
- Отримувати текстовий звіт за певний період;
- Отримувати графічний звіт за певний період;
- Мати функції керування заощадженнями;
- Мати можливість встановлення нагадування;
- Склад та параметри технічних засобів.

Система повинна підходити мінімальним параметрам сервера:

- Процесор 1.6 ГГц;
- Оперативна пам’ять 4 ГБ;
- Жорсткий диск 20 ГБ;
- Підтримка віртуалізації;
- Інформаційна та програмна сполучність.

Програмний продукт повинен коректно функціонувати в різних операційних системах. Розроблювана програмна система повинна бути пристосована до інтеграції у різні месенджери. Розробку виконувати з використанням мови C#, середовище розробки Visual Studio, базами даних PostgreSQL, docker.

4. СТАДІЇ РОЗРОБКИ

В ходів реалізації роботи проект повинен пройти крізь наступні стадії розробки:

- аналіз предметної області;
- проектування архітектури;

- реалізація класів і модулів;
- тестування результатів розробки;
- оформлення супровідної документації;
- здача роботи.

5. ПРОГРАМНА ДОКУМЕНТАЦІЯ

- Для програмного продукту повинні бути розроблені наступні документи:
- Пояснювальна записка;
- Технічне завдання;
- Презентаційний матеріал;
- Інструкція користувача;
- Додатки.

6. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Розроблений програмний продукт має виконувати всі вимоги, що складаються з перерахованих у п. 3.1 характеристик.

ДОДАТКИ

Додаток А

Лістинг кому для перекладу тексту

Лістинги коду

```
static public async Task<List<string>> TranslateTextRequest(string subscriptionKey, string endpoint, string route, string
inputText)
{
   using (var client = new HttpClient())
   using (var request = new HttpRequestMessage())
   {
      request.Method = HttpMethod.Post; request.RequestUri = new Uri(endpoint + route); request.Content = new
StringContent(inputText, Encoding.UTF8, "application/json");
      request.Headers.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

      HttpResponseMessage response = await client.SendAsync(request).ConfigureAwait(false);
      string result = await response.Content.ReadAsStringAsync();
      TranslationResult[] deserializedOutput = JsonConvert.DeserializeObject<TranslationResult[]>(result);
      // Iterate over the deserialized results.
      foreach (TranslationResult o in deserializedOutput)
      {
         Console.WriteLine("Detected input language: {0}\nConfidence score: { 1}\n ", o.DetectedLanguage.Language,
o.DetectedLanguage.Score);
         return o.Translations.Select(x => x.Text).ToList();
      }
   }
}
```

Лістинг налаштувань сервісів

```
public class Startup
{
   public void ConfigureServices(IServiceCollection services)
   {
      services.AddEntityFrameworkNpgsql(); services.AddDbContext<DataContext>(
      options => { options.UseNpgsql(ConnectionStringBuilder.Build()); },
      ServiceLifetime.Transient, ServiceLifetime.Transient);

      services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Vers ion_2_1);
      services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();
      services.AddSingleton<IStorage, MemoryStorage>(); services.AddSingleton<UserState>();
services.AddSingleton<ConversationState>(); services.AddSingleton<ShoppingRecognizer>();
services.AddSingleton<BookingDialog>(); services.AddSingleton<ShoppingDialog>(); services.AddSingleton<MainDialog>();
services.AddTransient<IBot,
      DialogAndWelcomeBot<MainDialog>>();
      services.AddTransactionManagers();
   }

   public void Configure(IApplicationBuilder app, IHostingEnvironment env)
   {
      app.UseHsts(); app.UseDefaultFiles(); app.UseStaticFiles(); app.UseWebSockets(); app.UseMvc();
   }
}

public class Startup
```

```
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Vers ion_2_1);
        services.AddScoped<IUpdateService, UpdateService>(); services.AddSingleton<IBotService, BotService>();
        services.Configure<BotSettings>(Configuration.GetSection("BotSettin gs"));
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseHttpsRedirection(); app.UseMvc();
        var botService = app.ApplicationServices.GetRequiredService<IBotService>();
    }
}
```

Лістинг налаштувань бота

```
public class BotSettings
{
    public string BotToken { get; set; }
    public string Socks5Host { get; set; }
    public int Socks5Port { get; set; }
}
```

Лістинг контексту бази даних

```
public sealed class DataContext : DbContext
{
    /// <inheritdoc />
    public DataContext(DbContextOptions contextOptions) : base(contextOptions)
    {
        ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;

        ChangeTracker.AutoDetectChangesEnabled = false;
    }

    public DbSet<User> Users { get; set; }
    public DbSet<Transaction> Transactions { get; set; }
    public DbSet<TransactionCategory> TransactionCategories
    {
        get; set;
    }
    public DbSet<Payment> Payments { get; set; }
    public DbSet<Category> Categories { get; set; }
    public DbSet<File> Files { get; set; }
    public DbSet<Location> Locations { get; set; }
    public DbSet<Alert> Alerts { get; set; }
    public DbSet<UserPlatform> UserPlatforms { get; set; }

    /// <inheritdoc />
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder
```

```
        .Entity<User>()
        .ToTable(nameof(User))
        .HasKey(x => x.Id);

        modelBuilder
        .Entity<Transaction>()
        .ToTable(nameof(Transaction))
        .HasKey(x => x.Id);

        modelBuilder
        .Entity<Payment>()
        .ToTable(nameof(Payment))
        .HasKey(x => x.Id);

        modelBuilder
        .Entity<Category>()
        .ToTable(nameof(Category))
        .HasKey(x => x.Id);

        modelBuilder
        .Entity<File>()
        .ToTable(nameof(File))
        .HasKey(x => x.Id);

        modelBuilder
        .Entity<Location>()
        .ToTable(nameof(Location))
        .HasKey(x => x.Id);

        modelBuilder

        .Entity<Alert>()
        .ToTable(nameof(Alert))
        .HasKey(x => x.Id);

        modelBuilder
        .Entity<TransactionCategory>()
        .ToTable(nameof(TransactionCategory))
        .HasKey(x => x.Id);

        base.OnModelCreating(modelBuilder);
    }
}
```

Лістинг сутностей бази даних

```
public class Alert
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public DateTimeOffset InitialDate { get; set; }
    public DateTimeOffset ScheduledTime { get; set; }

    public User User { get; set; }
    public string UserId { get; set; }

    public bool IsRecycled { get; set; }
}
```

```
public class Category
{
    public string Id { get; set; }
    public string Name { get; set; }
    public bool IsRecycled { get; set; }
}
public class File
{
    public string Id { get; set; }
    public string Description { get; set; }
    public DateTimeOffset InitialDate { get; set; }
    public Location Location { get; set; }
    public string LocationId { get; set; }
    public User User { get; set; }
    public string UserId { get; set; }

    public bool IsRecycled { get; set; }
}

public class Location
{
    public string Id { get; set; }
    public float Longitude { get; set; }
    public float Latitude { get; set; }
    public string Title { get; set; }
    public string Address { get; set; }
    public DateTimeOffset InitialDate { get; set; }
    public bool IsRecycled { get; set; }
}


public class Payment
{
    public string Id { get; set; }
    public decimal Value { get; set; }
    public string Currency { get; set; }
    public bool IsRecycled { get; set; }
}


public class Transaction
{
    public string Id { get; set; }
    public string Description { get; set; }
    public TransactionType TransactionType { get; set; }
    public DateTimeOffset InitialDate { get; set; }

    public virtual ICollection<TransactionCategory> Categories
    { get; set; }

    public User User { get; set; }
    public string UserId { get; set; }

    public Payment Payment { get; set; }
    public string PaymentId { get; set; }

    public Location Location { get; set; }
    public string LocationId { get; set; }

    public File File { get; set; }
```

```
    public string FileId { get; set; }

    public bool IsRecycled { get; set; }
}

public class TransactionCategory
{
    public string Id { get; set; }

    public Category Category { get; set; }
    public string CategoryId { get; set; }

    public Transaction Transaction { get; set; }
    public string TransactionId { get; set; }
}


public enum TransactionType
{
    Incoming = 0,
    Outgoing = 1,
    Future = 2,
    Debt = 3
}


public class User
{
    public string Id { get; set; }

    public string FirstName { get; set; }
    public string LastName { get; set; }

    public DateTimeOffset InitialDate { get; set; }
    public DateTimeOffset LastActivityDate { get; set; }
    public UserType UserType { get; set; }
    public virtual ICollection<Transaction> Transactions { get; set; }
    public virtual ICollection<UserPlatform> UserPlatforms { get; set; }
}

public class UserPlatform
{
    public string Id { get; set; }
    public string ApiIdentifier { get; set; }

    public User User { get; set; }
    public string UserId { get; set; }

    public string PlatformName { get; set; }
}


public enum UserType
{
    User = 0,
    Administrator = 1,
    Owner = 3
}


[Route("api/messages")]
```

```
[ApiController]
public class BotController : ControllerBase
{
    private readonly IBotFrameworkHttpAdapter _httpAdapter; private readonly IBot _bot;

    public BotController(IBotFrameworkHttpAdapter httpAdapter, IBot bot)
    {
        _httpAdapter = httpAdapter;
        _bot = bot;
    }

    [HttpPost, HttpGet]
    public async Task ProcessMessageAsync() => await
    _httpAdapter.ProcessAsync(Request, Response, _bot);
}

public enum Intent
{
    Shopping, Cancel, None
};

public class Shopping : IRecognizerConvert
{
    [JsonProperty("text")]
    public string Text { get; set; }

    //[JsonProperty("intents")]
    //public Intents Intents { get; set; }

    [JsonProperty("entities")]
    public Entities Entities { get; set; }

    [JsonProperty("sentiment")]
    public Sentiment Sentiment { get; set; }
    public Dictionary<Intent, IntentScore> Intents; public (Intent intent, double score) TopIntent()
    {
        var maxIntent = Intent.None; var max = 0.0;
        foreach (var entry in Intents)
        {
            if (entry.Value.Score > max)
            {
                maxIntent = entry.Key;
                max = entry.Value.Score.Value;
            }
        }
        return (maxIntent, max);
    }

    public decimal? GetPrice() => Entities.Money?.FirstOrDefault()?.Number;

    public string GetCommodity() => Entities.Commodity?.FirstOrDefault()?.FirstOrDefault();

    public DateTimeOffset? GetPurchaseDate() => Entities.Datetime?.FirstOrDefault()?.Timex?.FirstOrDefault();

    public void Convert(dynamic result)
    {
        var app = JsonConvert.DeserializeObject<Shopping>(JsonConvert.SerializeObject(result, new JsonSerializerSettings {
NullValueHandling = NullValueHandling.Ignore }));
        Text = app.Text;
        //AlteredText = app.AlteredText; Intents = app.Intents;
        Entities = app.Entities;
```

```
        //Properties = app.Properties;
    }
}

public class Entities
{
    [JsonProperty("$instance")]

    public Instance Instance { get; set; }

    [JsonProperty("datetime")]
    public Datetime[] Datetime { get; set; }

    [JsonProperty("money")]
    public Money[] Money { get; set; }

    [JsonProperty("Commodity")]
    public string[][] Commodity { get; set; }
}

public class Datetime
{
    [JsonProperty("type")]
    public string Type { get; set; }

    [JsonProperty("timex")]
    public DateTimeOffset[] Timex { get; set; }
}

public class Instance
{
    [JsonProperty("datetime")]
    public Commodity[] Datetime { get; set; }

    [JsonProperty("money")]
    public Commodity[] Money { get; set; }

    [JsonProperty("Commodity")]
    public Commodity[] Commodity { get; set; }
}

public class Commodity
{
    [JsonProperty("startIndex")]
    public long StartIndex { get; set; }

    [JsonProperty("endIndex")]
    public long EndIndex { get; set; }

    [JsonProperty("text")]
    public string Text { get; set; }

    [JsonProperty("type")]
    public string Type { get; set; }
}
public class Money

{
    [JsonProperty("number")]
    public decimal Number { get; set; }
```

```
    [JsonProperty("units")]
    public string Units { get; set; }
}

public class Intents
{
    [JsonProperty("Shopping")]
    public ShoppingClass Shopping { get; set; }
}

public class ShoppingClass
{
    [JsonProperty("score")]
    public double Score { get; set; }
}

public class Sentiment
{
    [JsonProperty("label")]
    public string Label { get; set; }

    [JsonProperty("score")]
    public double Score { get; set; }
}
```

Лістинг класу покупок, для LUIS моделі

```
public class Shopping : IRecognizerConvert
{
    public string Text;
    public string AlteredText;
    public Dictionary<Intent, IntentScore> Intents; public class _Entities
    {

        // Built-in entities
        public DateTimeSpec[] datetime;

        // Lists
        public string[][] Commodity;

        public class _InstancePrice
        {

            public InstanceData[] Commodity;
        }

        // Composites
        public class _InstancePurchase
        {
            public InstanceData[] Commodity;
        }
        public class PurchaseClass
        {
            public string[][] Commodity;[JsonProperty("$instance")]
            public _InstancePurchase _instance;
        }
        public PurchaseClass[] Purchase;
        // Instance
        public class _Instance
```

```
    {
        public InstanceData[] datetime; public InstanceData[] Commodity; public InstanceData[] Price; public InstanceData[]
Purchase;
    }
    [JsonProperty("$instance")] public _Instance _instance;
  }
  public _Entities Entities;

  [JsonExtensionData(ReadData = true, WriteData = true)] public IDictionary<string, object> Properties { get; set; }

  public void Convert(dynamic result)
  {
    var app = JsonConvert.DeserializeObject<Shopping>(JsonConvert.SerializeObject(result, new JsonSerializerSettings {
NullValueHandling = NullValueHandling.Ignore }));
    Text = app.Text;
    AlteredText = app.AlteredText; Intents = app.Intents; Entities = app.Entities; Properties = app.Properties;
  }

  public (Intent intent, double score) TopIntent()
  {
    Intent maxIntent = Intent.None; var max = 0.0;

    foreach (var entry in Intents)
    {
      if (entry.Value.Score > max)
      {
        maxIntent = entry.Key;
        max = entry.Value.Score.Value;
      }
    }
    return (maxIntent, max);
  }

  public (string Purchase, string Commodity) PurchaseEntity
  {
    get
    {
      var purchaseValue =
      Entities?._instance?.Purchase?.FirstOrDefault()?.Text; var commodityValue =
      Entities?.Commodity?.FirstOrDefault()?.FirstOrDefault(); return (purchaseValue, commodityValue);
    }
  }

  public (string Price, string PriceValue) PurchasePrice
  {
    get
    {
      var purchaseValue =
      Entities?._instance?.Purchase?.FirstOrDefault()?.Text; var froCommodityValue =
      Entities?.Purchase?.FirstOrDefault()?.Commodity?.FirstOrDefault()?.FirstOrDefault();
      return (purchaseValue, froCommodityValue);
    }
  }

  public string PurchaseDate=>Entities.datetime?.FirstOrDefault()?.Expressions.FirstOrDefault()?.Split('T')[0];
```

Лістинг головного діалогу

```
public class MainDialog : ComponentDialog
{
   private readonly ShoppingRecognizer _luisRecognizer; protected readonly ILogger Logger;

   public MainDialog(ShoppingRecognizer luisRecognizer, ShoppingDialog shoppingDialog, ILogger<MainDialog> logger)
   : base(nameof(MainDialog))
   {
      _luisRecognizer = luisRecognizer; Logger = logger;

      AddDialog(new TextPrompt(nameof(TextPrompt))); AddDialog(shoppingDialog);
      AddDialog(new WaterfallDialog(nameof(WaterfallDialog),
      new WaterfallStep[]
      {
IntroStepAsync, ActStepAsync, FinalStepAsync,
      }));

      InitialDialogId = nameof(WaterfallDialog);
   }

   private async Task<DialogTurnResult> IntroStepAsync(WaterfallStepContext stepContext, CancellationToken
cancellationToken)
   {
      if (!_luisRecognizer.IsConfigured)
      {
         await stepContext.Context.SendActivityAsync(
         MessageFactory.Text("NOTE: LUIS is not configured. To enable all capabilities, add 'LuisAppId', 'LuisAPIKey' and
'LuisAPIHostName' to the appsettings.json file.", inputHint: InputHints.IgnoringInput), cancellationToken);

         return await stepContext.NextAsync(null, cancellationToken);
      }

      var messageText = stepContext.Options?.ToString() ?? "What can I help you with today?\nSay something like \"I bought
some food for 5 euro yesterday";
      var promptMessage = MessageFactory.Text(messageText, messageText, InputHints.ExpectingInput);
      return await stepContext.PromptAsync(nameof(TextPrompt), new PromptOptions { Prompt = promptMessage },
cancellationToken);
   }

   private async Task<DialogTurnResult> ActStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
   {
      if (!_luisRecognizer.IsConfigured)
      {
         return await stepContext.BeginDialogAsync(nameof(ShoppingDialog), new ShoppingDetails(), cancellationToken);
      }

      var luisResult = await
      _luisRecognizer.RecognizeAsync<Shopping>(stepContext.Context, cancellationToken);
      switch (luisResult.TopIntent().intent)
      {
         case Intent.Shopping:
            var shoppingDetails = new ShoppingDetails()
            {
               PurchaseDate = luisResult.GetPurchaseDate(),
               Commodity = luisResult.GetCommodity(),
               Price = luisResult.GetPrice()
            };

            return await stepContext.BeginDialogAsync(nameof(ShoppingDialog), shoppingDetails, cancellationToken);
```

```
            default:
                var didntUnderstandMessageText = $"Sorry, I didn't get that. Please try asking in a different way (intent was
            { luisResult.TopIntent().intent})";
        var didntUnderstandMessage = MessageFactory.Text(didntUnderstandMessageText, didntUnderstandMessageText,
InputHints.IgnoringInput);
                await stepContext.Context.SendActivityAsync(didntUnderstandMessage, cancellationToken);
                break;
        }

        return await stepContext.NextAsync(null, cancellationToken);
    }

    private async Task<DialogTurnResult> FinalStepAsync(WaterfallStepContext stepContext, CancellationToken
cancellationToken)

    {
        if (stepContext.Result is BookingDetails result)
        {
            var timeProperty = new TimexProperty(result.TravelDate);
            var travelDateMsg = timeProperty.ToNaturalLanguage(DateTime.Now);
            var messageText = $"I have you booked to
        { result.Destination}
            from { result.Origin}
            on { travelDateMsg}
            "; var message = MessageFactory.Text(messageText,
        messageText, InputHints.IgnoringInput);
            await stepContext.Context.SendActivityAsync(message, cancellationToken);
        }

        var promptMessage = "What else can I do for you?"; return await
        stepContext.ReplaceDialogAsync(InitialDialogId, promptMessage, cancellationToken);
    }
}


public DateResolverDialog(string id = null): base(id ?? nameof(DateResolverDialog))
{
    AddDialog(new DateTimePrompt(nameof(DateTimePrompt),
    DateTimePromptValidator));
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new WaterfallStep[]
    {InitialStepAsync, FinalStepAsync,}));

    InitialDialogId = nameof(WaterfallDialog);
}

private async Task<DialogTurnResult> InitialStepAsync(WaterfallStepContext stepContext, CancellationToken
cancellationToken)
{
    var timex = (string)stepContext.Options;

    var promptMessage = MessageFactory.Text(PromptMsgText, PromptMsgText, InputHints.ExpectingInput);

    var repromptMessage = MessageFactory.Text(RepromptMsgText, RepromptMsgText, InputHints.ExpectingInput);

    if (timex == null)
    {
        return await stepContext.PromptAsync(nameof(DateTimePrompt),
        new PromptOptions
        {
            Prompt = promptMessage,
```

```
            RetryPrompt = repromptMessage,
        }, cancellationToken);
    }

    var timexProperty = new TimexProperty(timex); if
    (!timexProperty.Types.Contains(Constants.TimexTypes.Definite))
    {
        return await stepContext.PromptAsync(nameof(DateTimePrompt),
        new PromptOptions
        {
            Prompt = repromptMessage,
        }, cancellationToken);
    }

    return await stepContext.NextAsync(new List<DateTimeResolution> { new DateTimeResolution { Timex = timex }
}, cancellationToken);
}

private async Task<DialogTurnResult> FinalStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    var timex = ((List<DateTimeResolution>)stepContext.Result)[0].Timex;
    return await stepContext.EndDialogAsync(timex, cancellationToken);
}

private static Task<bool> DateTimePromptValidator(PromptValidatorContext<IList<DateTimeResolu tion>> promptContext,
CancellationToken cancellationToken)
{
    if (promptContext.Recognized.Succeeded)
    {

        var timex = promptContext.Recognized.Value[0].Timex.Split('T')[0];

        var isDefinite = new TimexProperty(timex).Types.Contains(Constants.TimexTypes.Definite);

        return Task.FromResult(isDefinite);
    }

    return Task.FromResult(false);
```