Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

магістра
(освітній ступінь (освітньо-кваліфікаційний рівень))

на тему: **Розробка персонального фінансового асистента з використанням когнітивних технологій**

Виконав: студент (ка) VI курсу, групи СПм-62

спеціальності (напряму підготовки) 121
Інженерія програмного забезпечення
(шифр і назва спеціальності (напряму підготовки))

(підпис) Олещук Р. С. (прізвище та ініціали)

Керівник (підпис) Михалик Д. М. (прізвище та ініціали)

Нормоконтроль (підпис) Бойко І.В. (прізвище та ініціали)

Рецензент (підпис) Дмитроца Л.П. (прізвище та ініціали)

м. Тернопіль – 2019

АНОТАЦІЯ

Дипломна робота на здобуття освітнього ступеня магістр за спеціальністю 121 – «Інженерія програмного забезпечення». – Тернопільський національний технічний університет імені Івана Пулюя, Тернопіль, 2019 р.

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

В рамках цього проекту виконується розробка програмної системи для ведення розрахунку та аналізу особистих фінансів. 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 майбутні плани та заощадження.

Суть роботи полягає у створенні програмного забезпечення, 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. мова програмування, когнітивні технології, база даних, технічні вимоги, область застосування, інформаційні технології.

ANNOTATION

The master thesis for the qualification level of magistr on specialty 121 — Software Engineering. – Ternopil Ivan Pului National Technical University, Ternopil, 2019.

The purpose of the study is to investigate the problem of consumer financial management and to develop a software system that, through cognitive technologies, will allow to ask for cost fixation and analysis.

Within this project, a software system for calculating and analyzing personal finances is being developed. The object of the study is the problem of controlling one's personal budget. 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 future plans and savings.

The essence of the job is to create software that integrates with the messenger, and with the help of cognitive technologies allows to recognize the commands of the user, for the possibility of recording and analyzing the cost of the user.

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.

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

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.

ДНАОП – державний нормативний акт з охорони праці.

ОП – охорона праці.

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

Зміст

ВСТУП

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

Облік витрат потрібен, насамперед, для визначення основних речей, на які найбільше витрачається коштів, та допоможе проаналізувати, на чому можна заощадити.

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 покупки. Для того, щоб використовувати його, достатньо мати комп’ютери чи мобільний телефон.

Згідно опитувань, майже 45% 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 фінансовим асистентом застосовуються системи розпізнання мови.

В якості реалізації проекту було вирішено розробити програмну 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 месенджер.

1 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

1.1 Аналіз вимог до програмної системи

1.1.1 Аналіз предметної області

В рамках даного проекту виконується розробка програмної системи для ведення розрахунку та аналізу особистих фінансів. 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 майбутні плани та заощадження.

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]

Також можна використовувати електронну таблицю на комп'ютері. Використовуючи просту електронну таблицю в такій програмі, як Microsoft Excel, можна чітко організувати витрати і легко створювати графіки, щоб краще зрозуміти свої витрати. Є багато конкретних способів зробити це, але хорошим початком може стати створення особистого бюджету. Це буде зроблено на тижневої або щомісячній основі і буде включати, як було сказано, таку інформацію, як сума, категорія і дата кожної транзакції.

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 варіантів для відстеження витрат - це є будь-яка фінансова програма. 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.

1.1.2 Постановка задачі

Після виконання аналізу предметної області і виявлення основних проблем, визначено мету та завдання для виконання проектного рішення.

За мету було поставлено розробити програмну систему, яка дозволить спростити облік особистих фінансів.

На основі даних, зібраних під час виконання аналізу предметної області, виділено основні завдання для виконання:

- 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 асистента, 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;

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

Для здійснення всіх поставлених задач, програмний продукт повинен:

1. Інтегруватись з месенджером у вигляді бота та отримувати команди від користувача.
2. Дозволити вести керування витратами та доходами
    - Вносити дані
    - Видалити дані
    - Редагувати дані
3. Встановлювати інформаційний ліміт на витрати
4. Отримувати текстовий звіт за певний період
    - У вигляді Excel таблиці
    - Вивід у чат
    - У вигляді CSV файлу
5. Отримувати графічний звіт за певний період
    - У вигляді діаграми
    - У вигляду гістограми
6. Мати функції керування заощадженнями
7. Мати можливість встановлення нагадування
    - Про майбутню транзакцію

- Про майбутню подію

1.1.3 Пошук актантів та варіантів використання

Виходячи із поставленого завдання, можна виділити основні типи користувачів (акторів) даної системи: адміністратор та користувач. Опис акторів наведено у таблиці 1.1.

Таблиця 1.1 – 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 1.2 - 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 1.2 - 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 (див. рисунок 1.1), з якої ми бачимо який функціонал необхідно реалізувати в системі.

Рисунок 1.1 – Опис прецедентів для користувача

На рисунку 1.2 зображено функціонал, який повинен підтримувати система для ролі адміністратора.

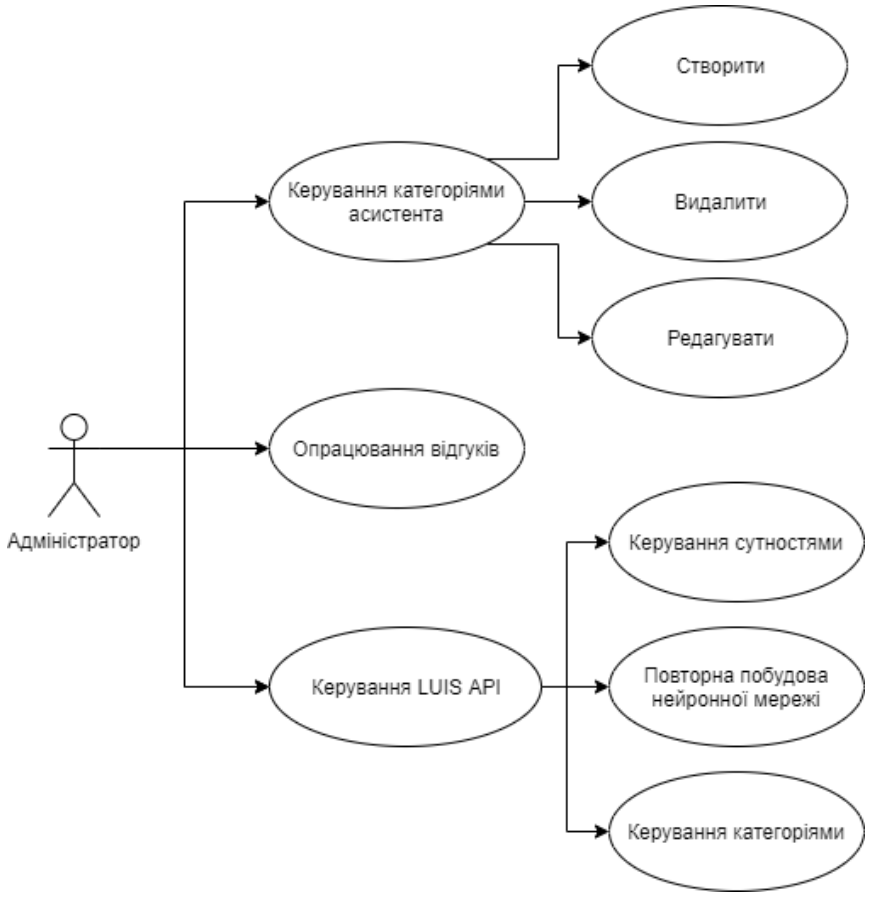

Рисунок 1.2 – Опис прецедентів

1.1.4 Огляд існуючого програмного забезпечення

Money Lover - додаток, який 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 [3].

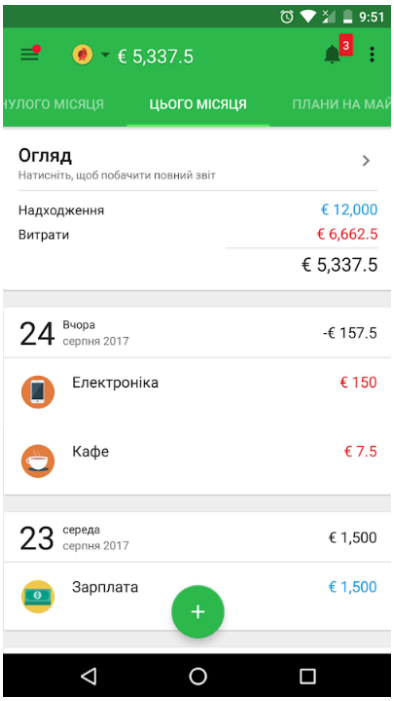

Рисунок 1.3 – Інтерфейс додатку Money Lover

Bills Monitor, ще один додаток для керування бюджетом. Має можливість сповіщеннями нагадати користувачеві про майбутню покупку, або оплату сервісу. Підтримує лише платформу IOS [4].

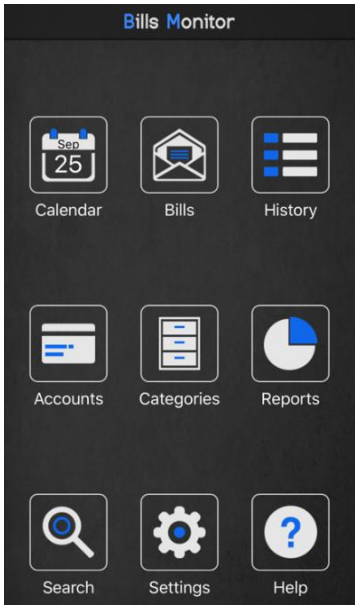

Рисунок 1.4 – Інтерфейс додатку Bills Monitor

Monefy - додаток, який 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. [5].

Рисунок 1.5 - Інтерфейс додатку Monefy

CoinKeeper додаток, що 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 [6].

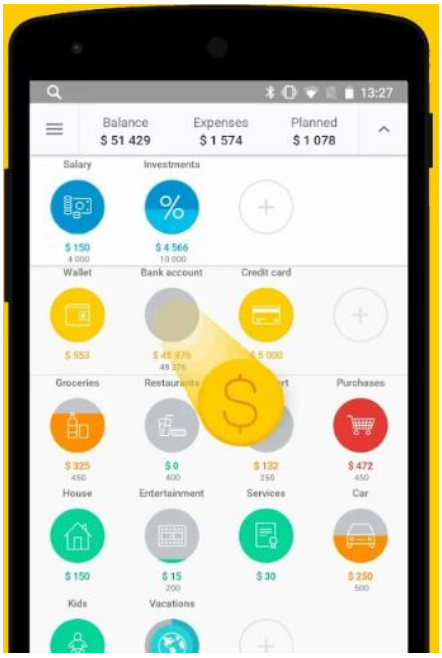

Рисунок 1.6 - Інтерфейс додатку CoinKeeper

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].

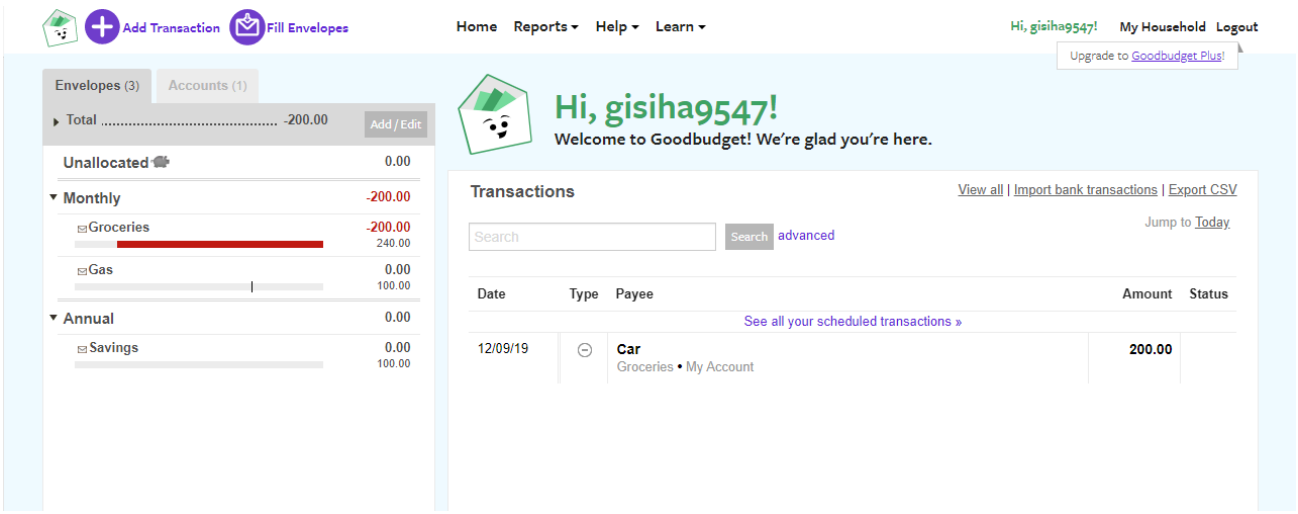

Рисунок 1.7 - Інтерфейс додатку Goodbudget

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.

1.2 Проектування програмної системи

1.2.1 Вибір процесу розробки

Ітеративний процес розробки був розроблений для заміни моделі водоспаду, у якому було виявлені ряд 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].

Робота за допомогою ітерацій являє собою поділ розробки програмного забезпечення на менші частини. Кожна ітерація дозволяє отримати завершену частину проекту, яку можна тестувати, та розгортати, для тестового користування клієнтами. Цикли ітерації повторюються, поки програмне забезпечення не буде завершеним. Процес розробки не розпочинається з повного набору вимог та дизайну, а лише з тієї частини даних, яка потрібна для роботи в даній ітерації.

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 (див. рисунок 1.8). 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.

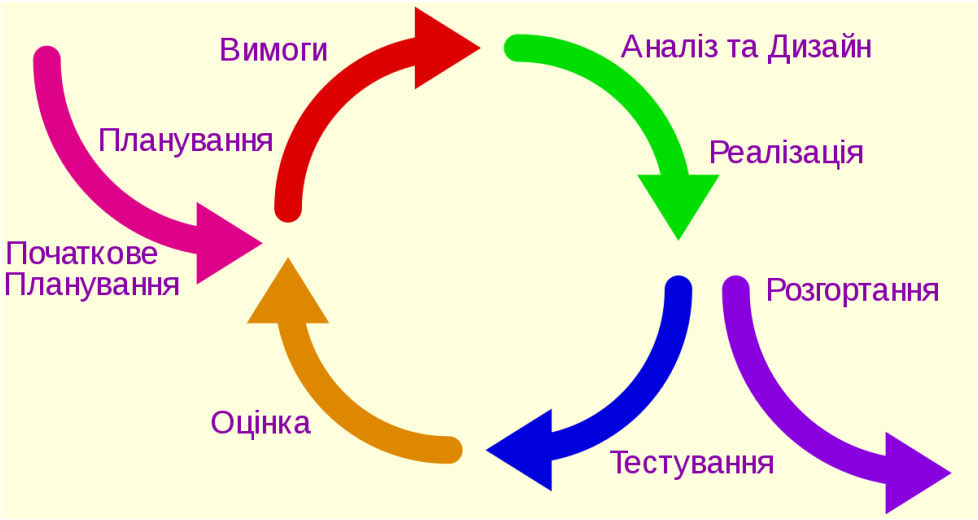

Рисунок 1.8 – Ілюстрація ітераційного процесу

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, тестерам 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.

Тестувальники, особливо якщо вони виконують ручне тестування, є одними з тих, у кого виникають найбільші труднощі з адаптацією до ітеративного процесу, коли вони переходять із моделі водоспаду, особливо якщо вони тестували раніше готовий продукт. Перехід до ітерацій змушує їх діяти по-іншому і мислити у формах конкретних функціональних можливостей, які слід перевірити замість повністю розробленої системи. Вони повинні працювати паралельно з розробниками, щоб дотримуватися термінів ітерації.

Часто немає 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.

1.2.2 Побудова схеми бази даних

Для побудови схеми бази даних потрібно дослідити предметну область, визначити основні сутності, що будуть використовуватись у системі, та проаналізувати їх взаємовідношення. Під цим розуміється визначення первинних та вторинних ключів.

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].

Нормальна форма – вимоги для відношень між сутностями, що оптимізуються за певним алгоритмом, та допомагають уникнути надлишковості даних, та можливих помилок у вибірках.

Згідно до поставлених вимог, база даних, що використовується повинна бути приведена до третьої нормальної форми. Для цієї здачі спочатку потрібно привести БД до другої та першої нормальної форми.

Перша нормальна форма означає, що усі атрибути відношення є простими, та усі таблиці мають основний ключ, який ідентифікує запис. Також необхідно, щоб дотримувалась атомарність, а саме, щоб кожен з атрибутів містив лише одне значення [11].

Для другої нормальної форми необхідно, щоб виконувалась перша нормальна форма, та усі дані, що знаходяться у стовпцях, що не є ключовими залежали від первинного ключа. Якщо стається, що поле не має залежності від первинного ключа, то потрібно внести в ключ додаткові таблиці [12].

Загалом 1НФ і 2НФ розглядаються як проміжні етапи в процесі нормалізації БД. Більша частина СКБД орієнтована на досягнення наступного 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 1.9.

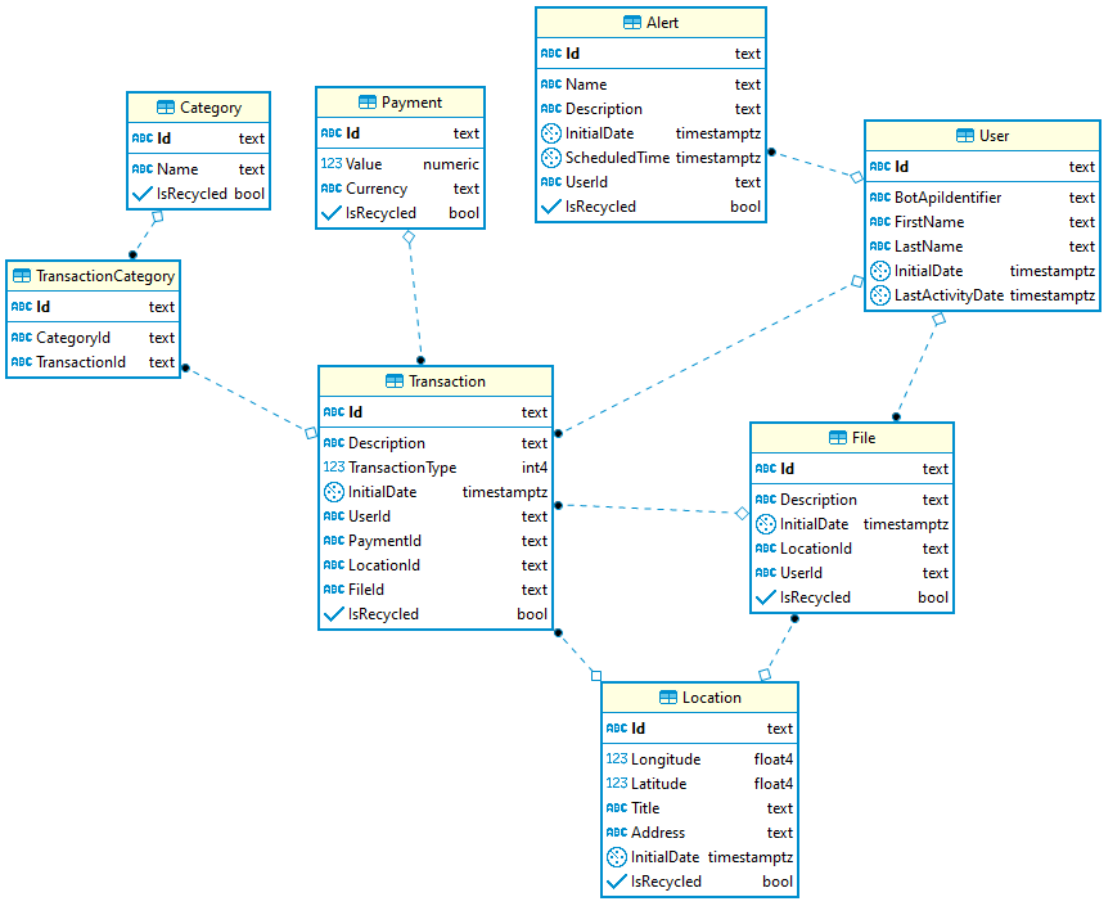

Рисунок 1.9 - Схема бази даних

1.2.3 Побудова UML-діаграми класів

Для опису архітектури програмної системи 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 Рисунку 1.10. 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 рисунку 1.11 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.

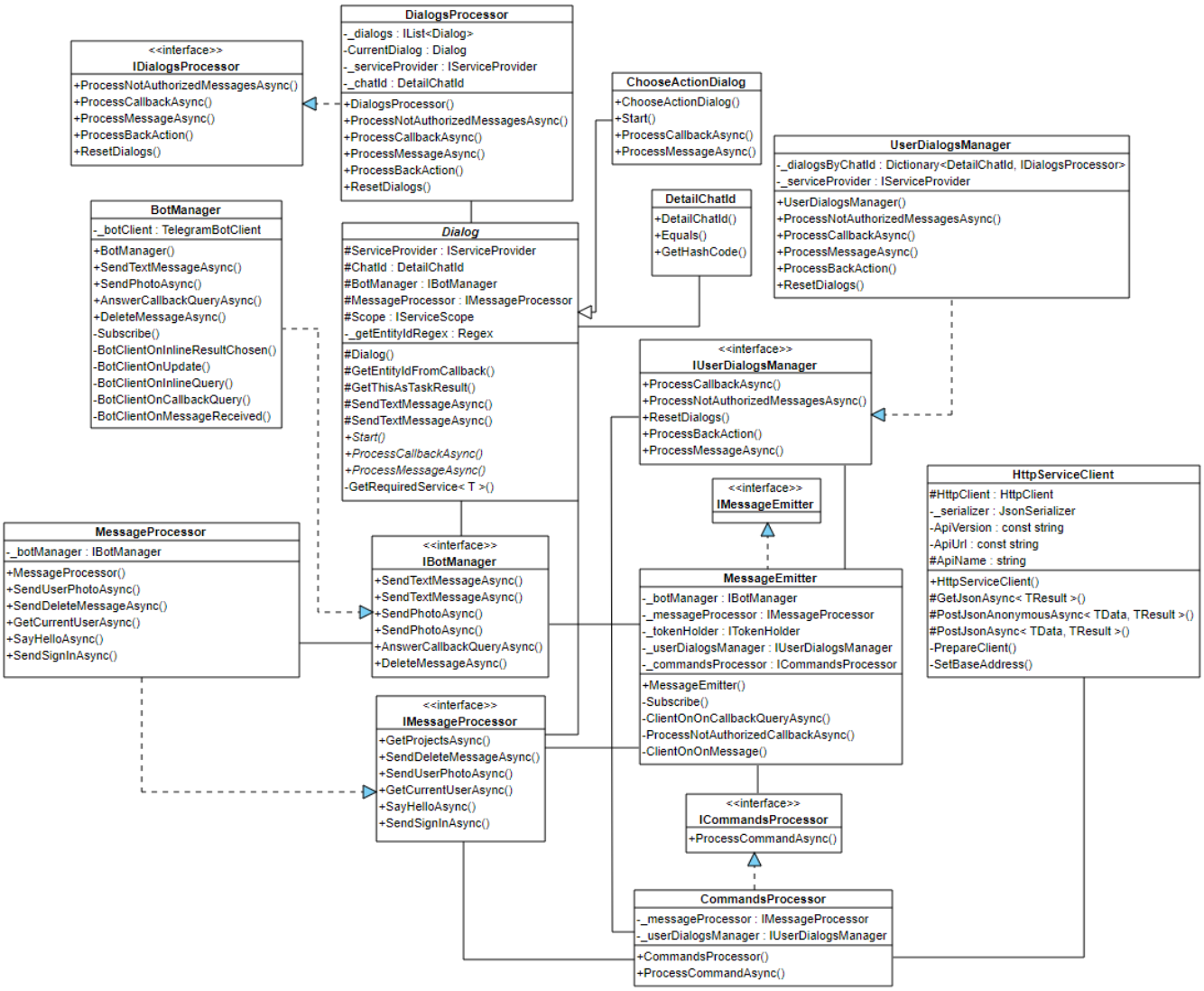

Рисунок 1.10 – UML діаграма серверної частини бота

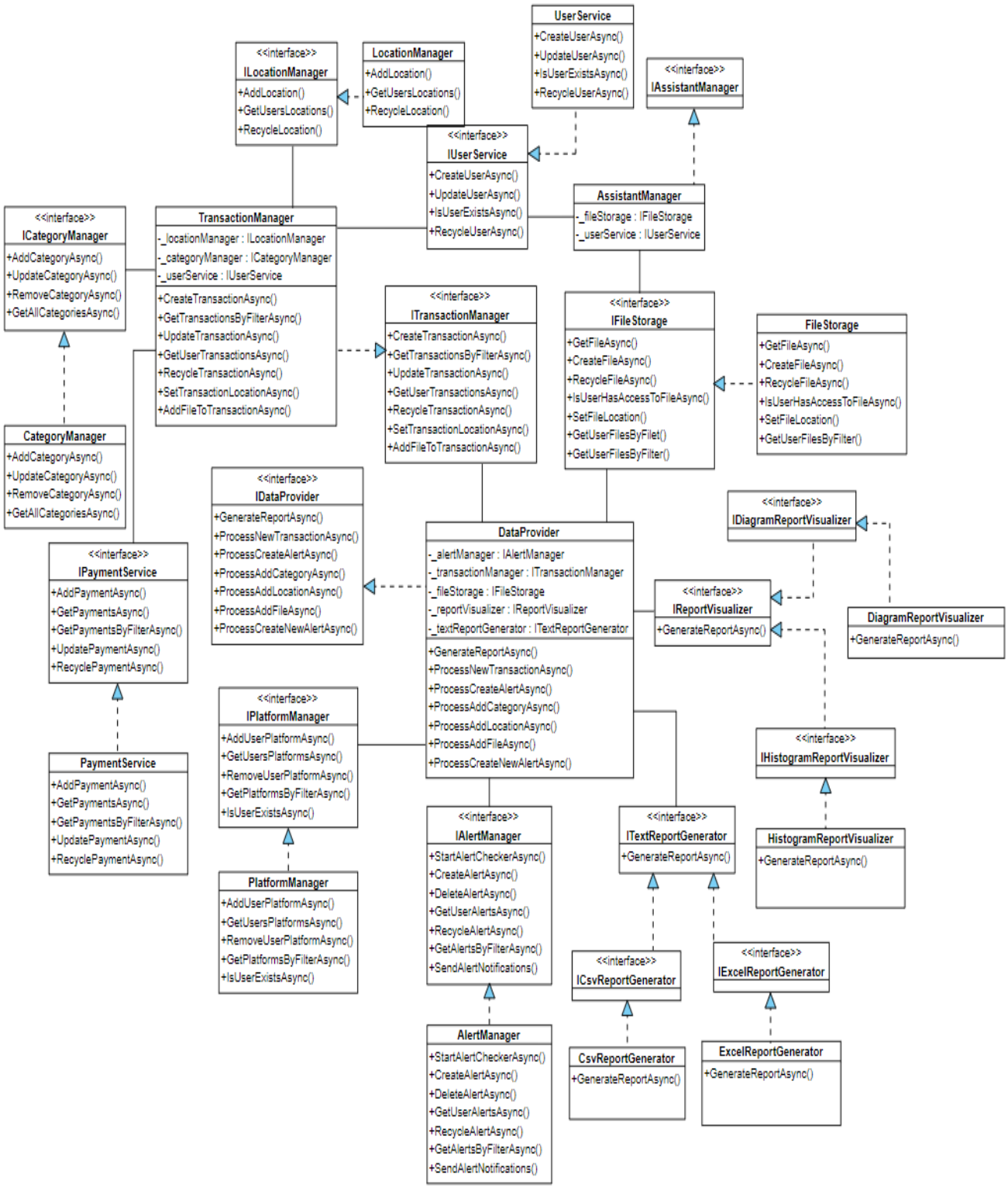

Рисунок 1.11 - UML діаграма серверної частини асистента

Для роботи з базою даних використовується ORM Entity Framework та підхід Code First. На рисунку 1.12 зображено UML діаграму класів для реалізації CRUD операцій над таблицями.

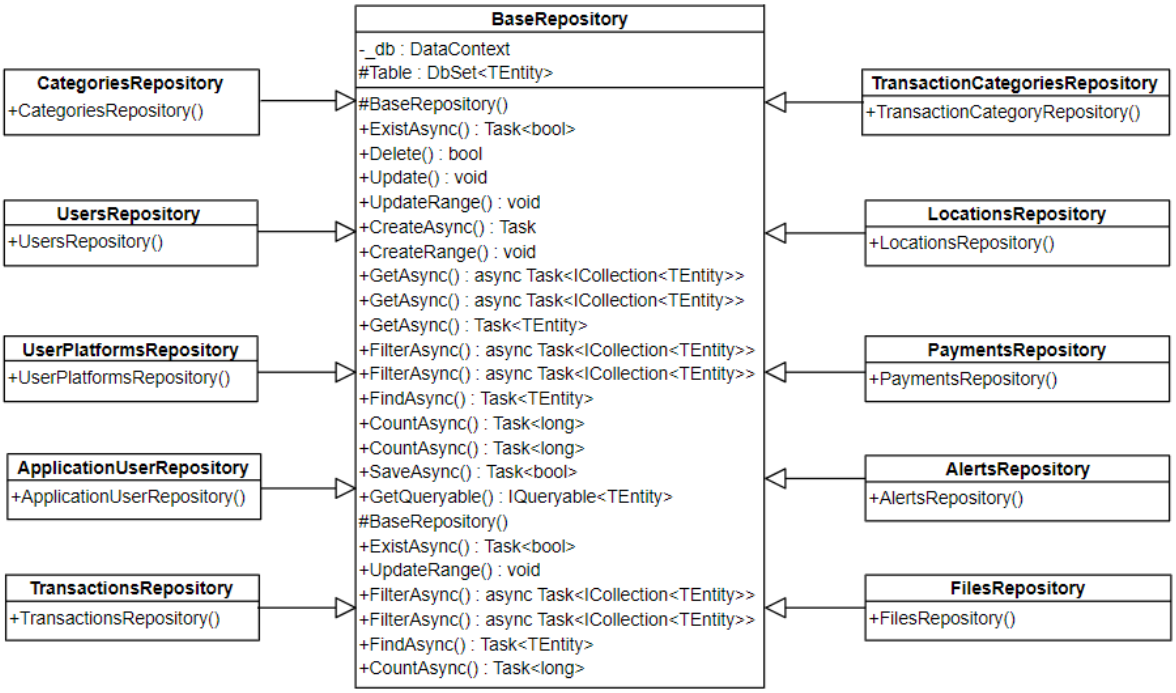

Рисунок 1.12 – UML діаграма роботи за базою даних

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.

1.2.4 Моделювання архітектури системи

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 (див. рисунок 1.13).

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 може одночасно 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.

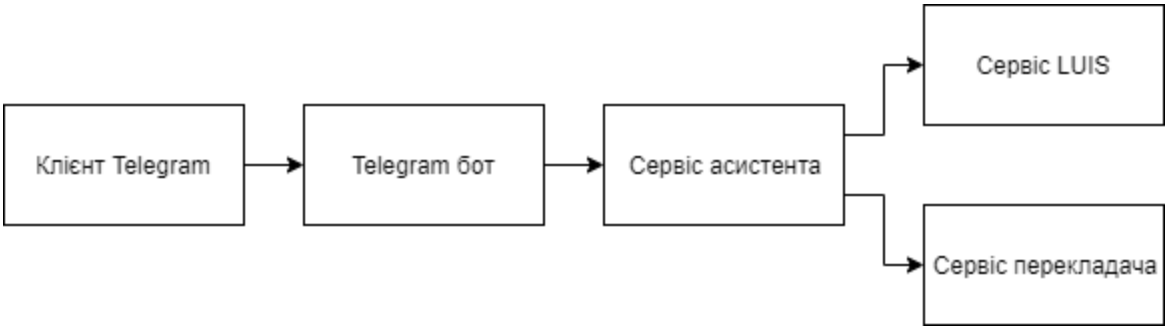

Рисунок 1.13 – Схематичне зображення клієнт-серверної архітектури

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].

1.3 Конструювання програмної системи

1.3.1 Вибір мови 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. Якщо поглянути на історію мов

програмування і їх особливості, то кожна мова була розроблений для певної мети, щоб вирішити конкретну потребу в той час.

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.

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 у середовищі CLR, то для розробника спрощується керування пам’яттю, звільнення ресурсів при розробці на даній мові.

Для середовища розробки було обрано Visual Studio 2019.

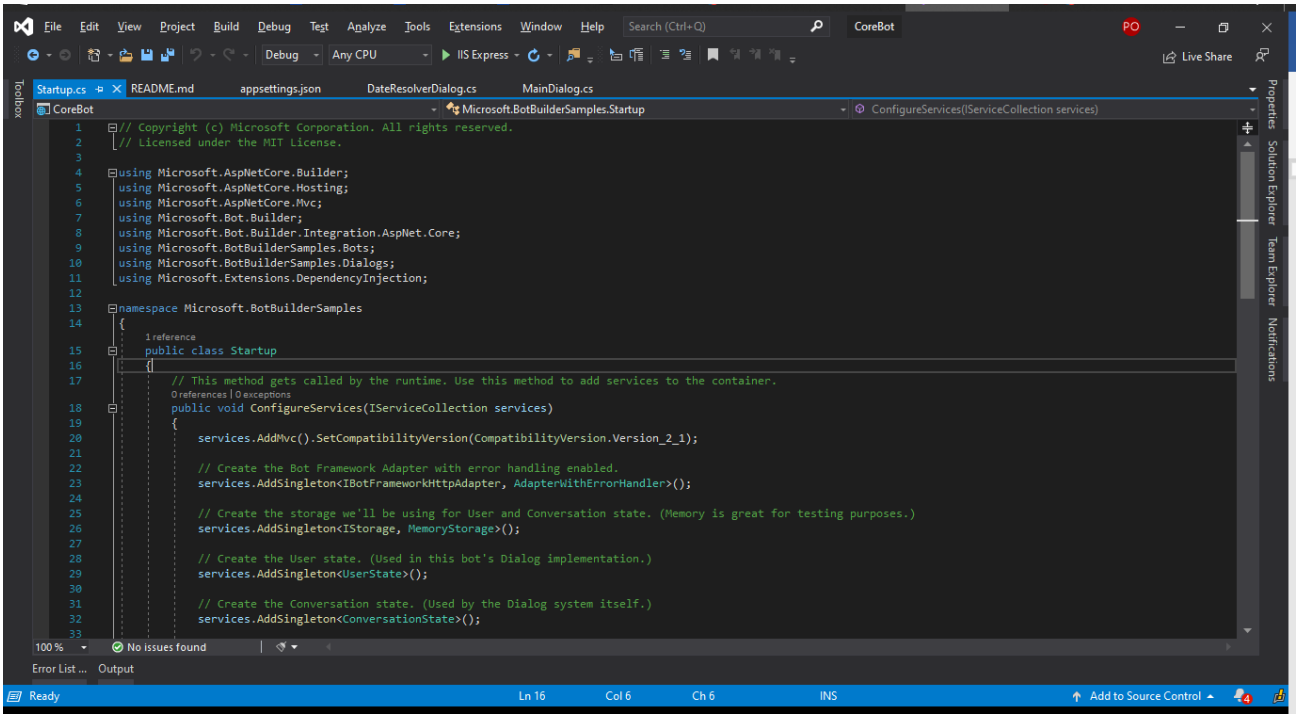

Рисунок 1.14 – Інтерфейс середовища Visual Studio 2019

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].

Перевагами C#, порівняно з Java є те, що він активно розвивається завдяки підтримці Microsoft. Також дана мова програмування має програмний код у вільному доступі, а середовища розробки є безкоштовними, для особистого використання, або невеликих компаній. Для спрощення розробки, у C# додано багато синтаксичних спрощень, що дозволять спростити розробку, та розуміння коду в цілому. Синтаксис C# схожий до Java, C та C++, але має досить низький поріг для входження. Також підтримується розробка для багатьох платформ, включаючи мобільні.

1.3.2 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 програмної системи, було спроектовано модель бази даних що зображена на рисунку 1.15.

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.

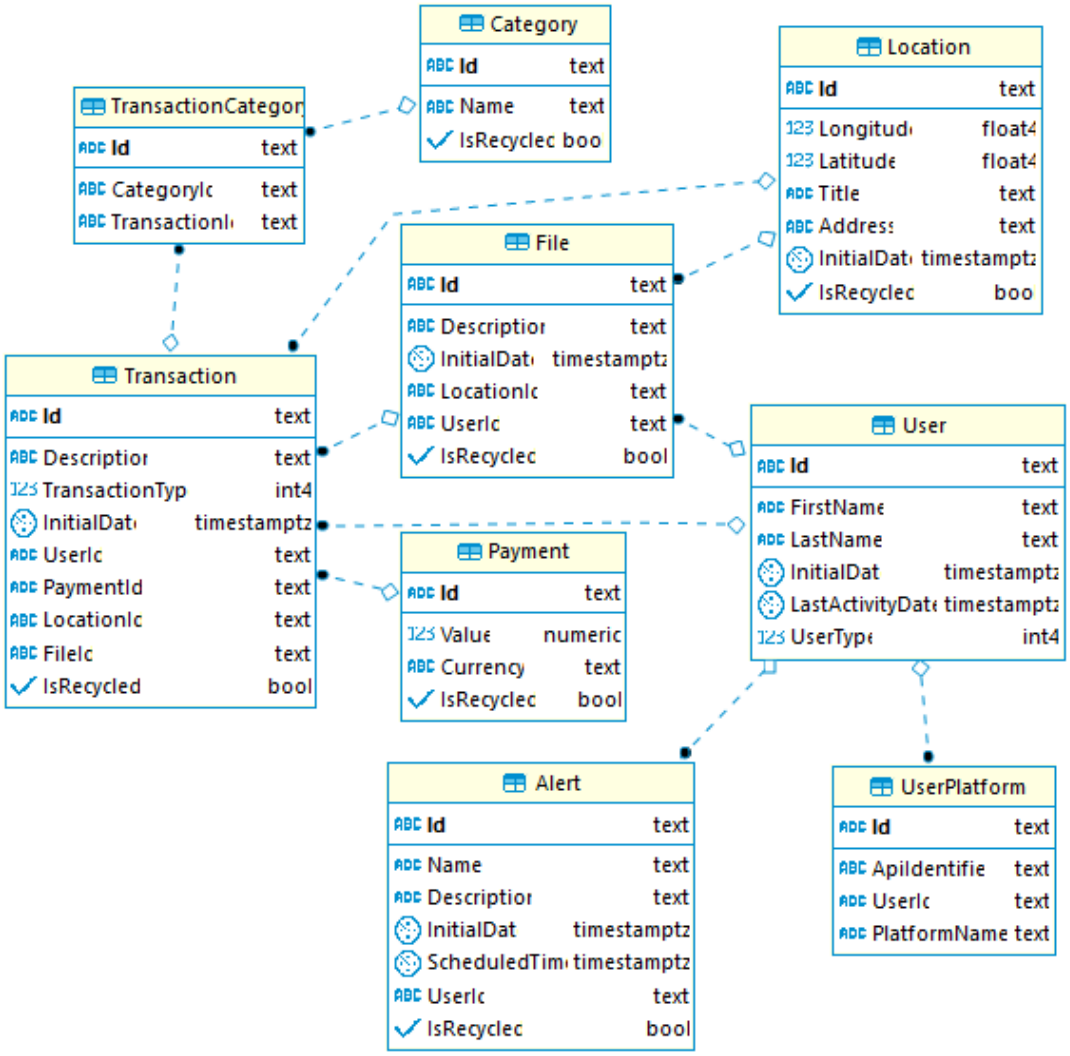

Рисунок 1.15 – Фізична модель бази даних

У таблиці User міститься інформація про користувачів системи (Рисунок 1.16).

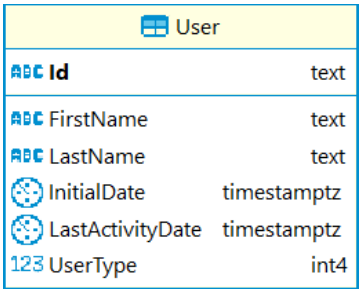

Рисунок 1.16 – Таблиця User

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 (див. рисунок 1.17) містить інформацію про створені сповіщення користувачів.

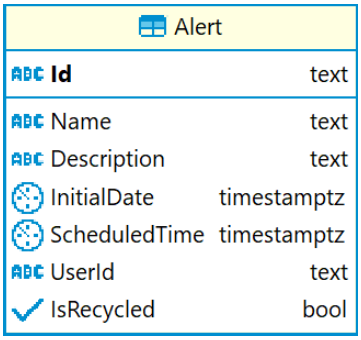

Рисунок 1.17 - Таблиця Alert

Таблиця Payment слугує для зберігання оплати користувача, де Value – сума, а Currency валюта (Рисунок 1.18).

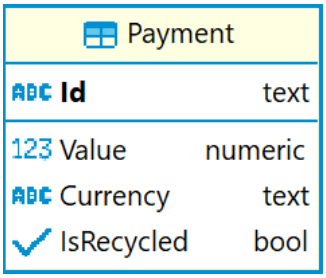

Рисунок 1.18 - Таблиця Payment

У таблиці File зберігається інформація про фізичні файли, завантажені користувачами (Рисунок 1.19). Вони можуть бути прикріпленими до транзакції, або безпосередньо до користувача. Поле LocationId слугує для фіксації розташування, під час якого було завантажено файл.

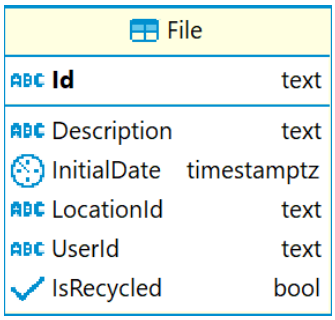

Рисунок 1.19 - Таблиця File

Таблиця Transaction містить інформацію про усі транзакції в системі. Вказує на локацію, оплату, файл, та користувача що здійснив транзакцію (Рисунок 1.20). Також містить тип транзакції.

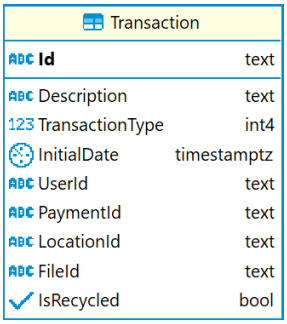

Рисунок 1.20 - Таблиця Transaction

Таблиця Category зберігає назви категорій витрат та доходів (Рисунок 1.21).

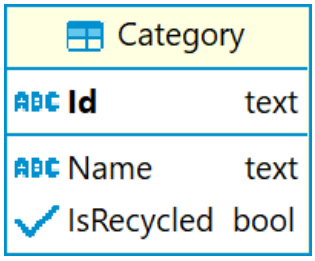

Рисунок 1.21 - Таблиця Category

У таблиці TransactionCategory фіксується множинний зв'язок між категорією та транзакцією. Адже безліч транзакцій можуть мати безліч категорій (Рисунок 1.22).

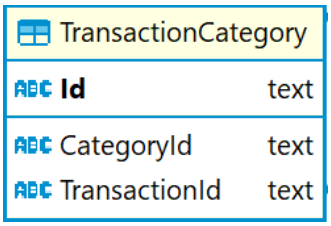

Рисунок 1.22 - Таблиця Transaction Category

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 (Рисунок 1.23).

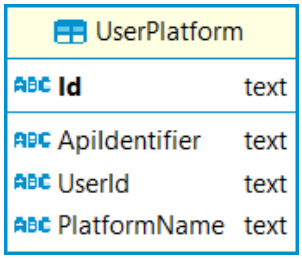

Рисунок 1.23 - Таблиця User Platform

У таблиці Location фіксуються координати користувача, та адреса, згідно цих координат (Рисунок 1.24).

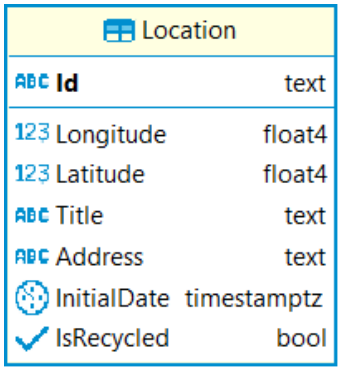

Рисунок 1.24 - Таблиця Location

1.3.4 Реалізація основних класів та методів

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

У лістингу 1.1 зображено реєстрацію компонентів за допомого. DI, для подальшого їх використання у коді.

Лістинг 1.1 – 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 1.2 – 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 1.3 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 1.3 – 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 1.4), який завдяки адаптеру передає вхідний текст на подальшу обробку.

Лістинг 1.4 – 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 1.5).

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 1.5 – 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;
        }
```

1.4 Використання програмної системи

1.4.1 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 1.25).

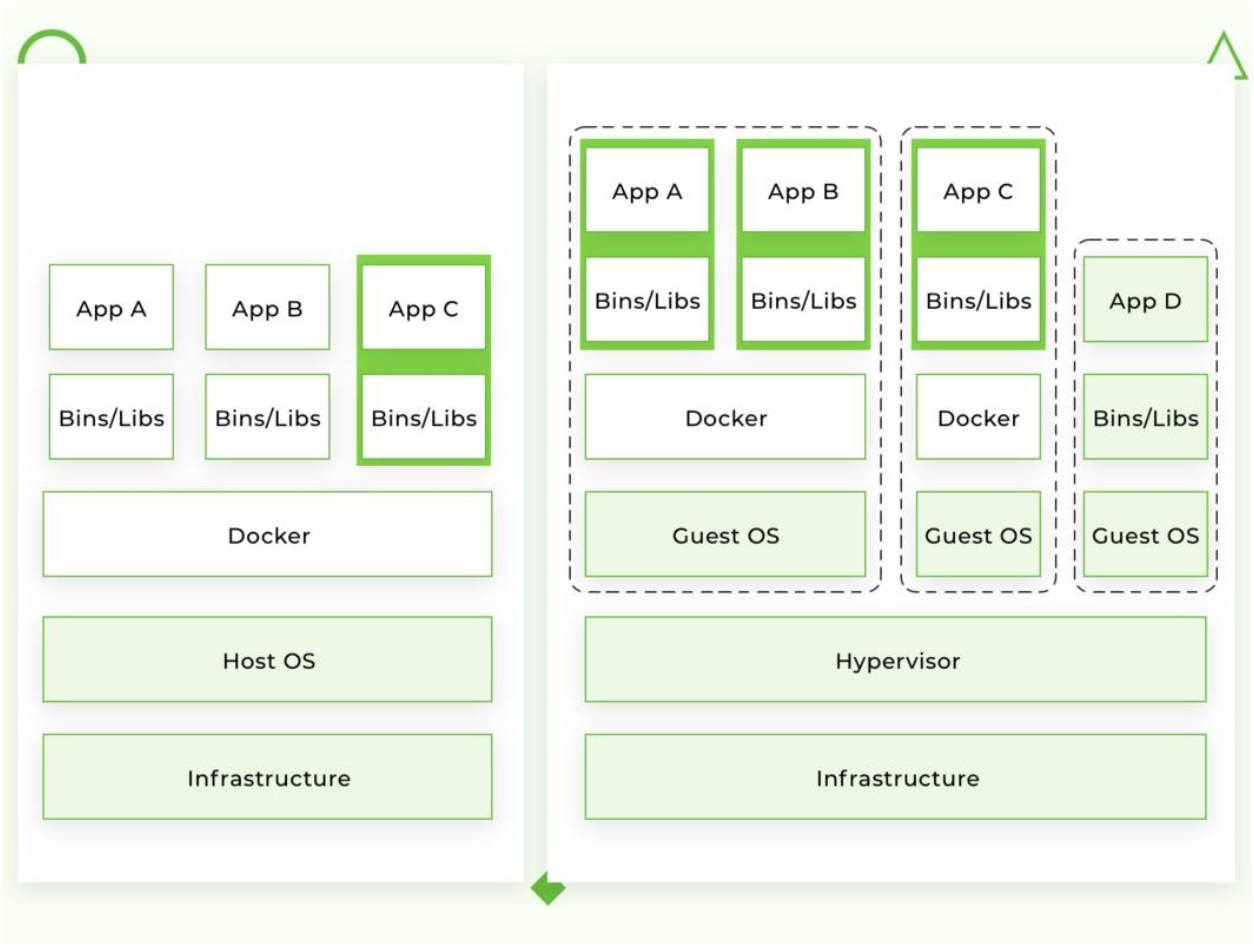

Рисунок 1.25 – Схематичне зображення процесів у системі

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 1.26 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; \
```

Рисунок 1.26 – Dockerfile для побудови релізної версії

Для побудови контейнерів, використовується наступна конфігурація (див. рисунок 1.27)

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"]
```

Рисунок 1.27 – Приклад конфігурації docker файлу

Приклад docker compose файлу зображено на рисунку 1.28. 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
```

Рисунок 1.28 – 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.

1.4.2 Опис типових схем використання системи

На Рисунку 1.29 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.

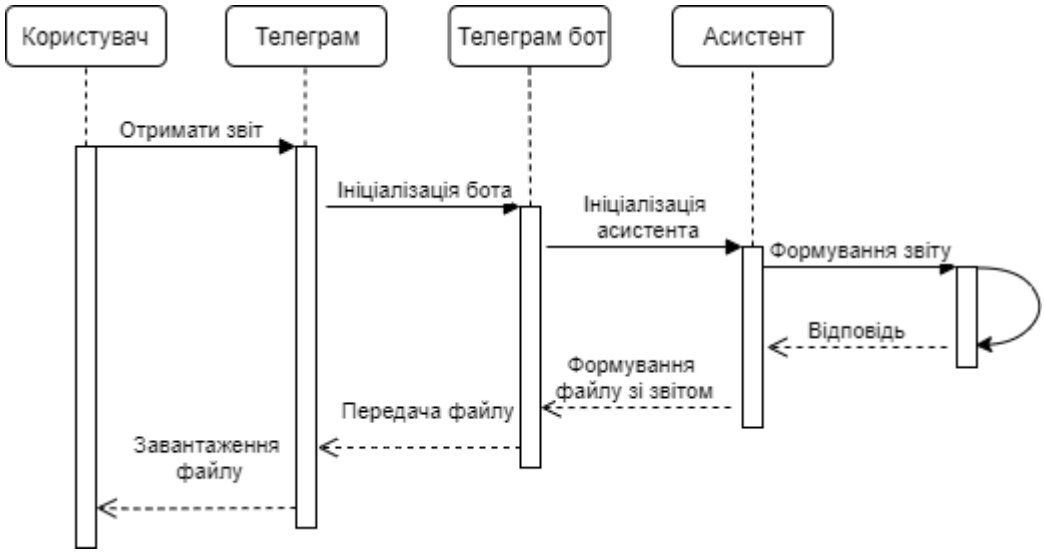

Рисунок 1.29 – Діаграма послідовностей отримання звіту

На рисунку 1.30 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.

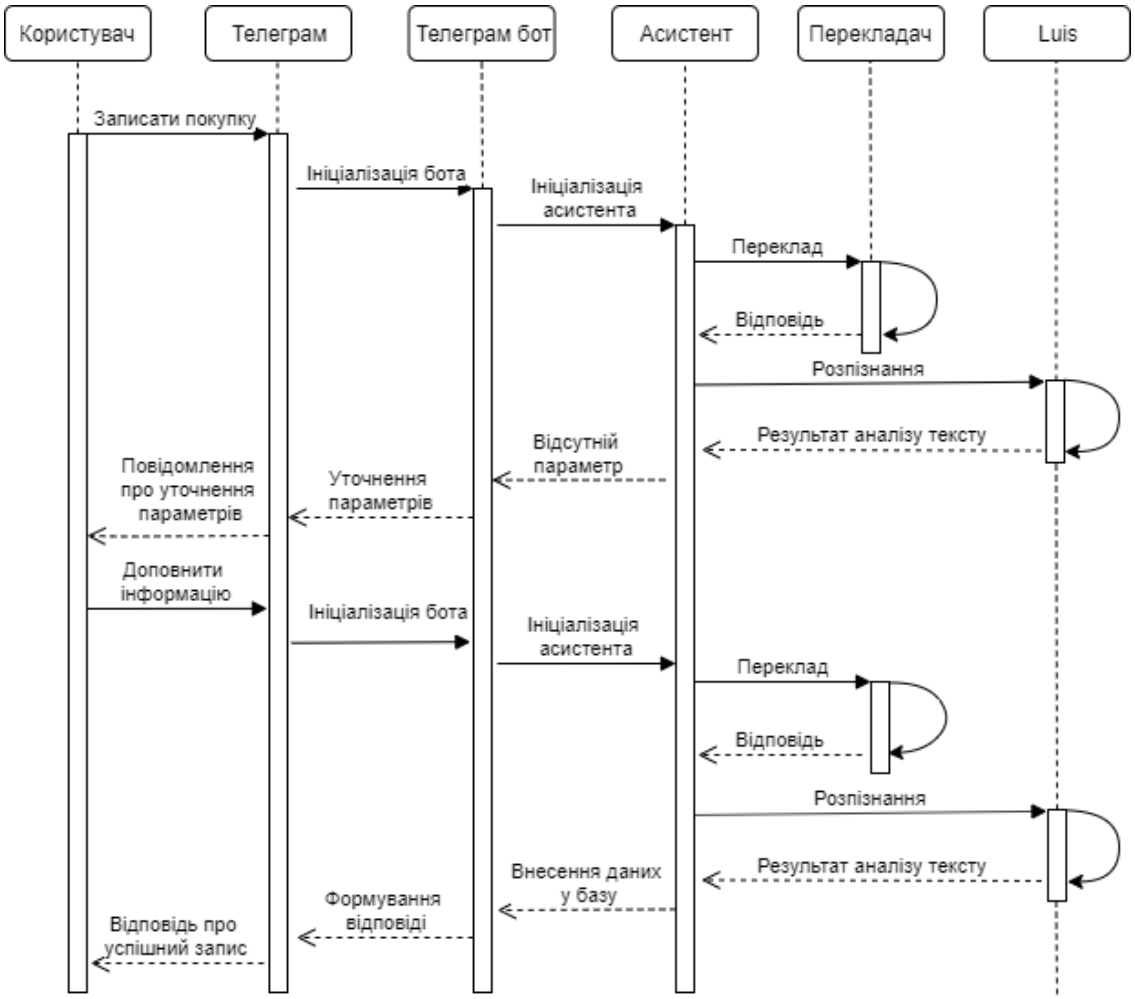

Рисунок 1.30 – Діаграма послідовностей для реєстрації покупки

На рисунку 1.31 зображено процес внесення нової покупки.

Рисунок 1.31 – Запис покупки

Для отримання звіту, потрібно ввести команду, як показано на рисунку 1.32. Після його генерації та отримання у повідомленні, його можна відкрити у ПЗ, що призначене для роботи з файлами .xlsx.

Рисунок 1.32 – Генерація звіту

Рисунок 1.33 містить інтерфейс при роботі з асистентом з смартфону. Порівнявши його з ПК версією, можна відмітити, що користування асистентом є однаковим з усіх платформ.

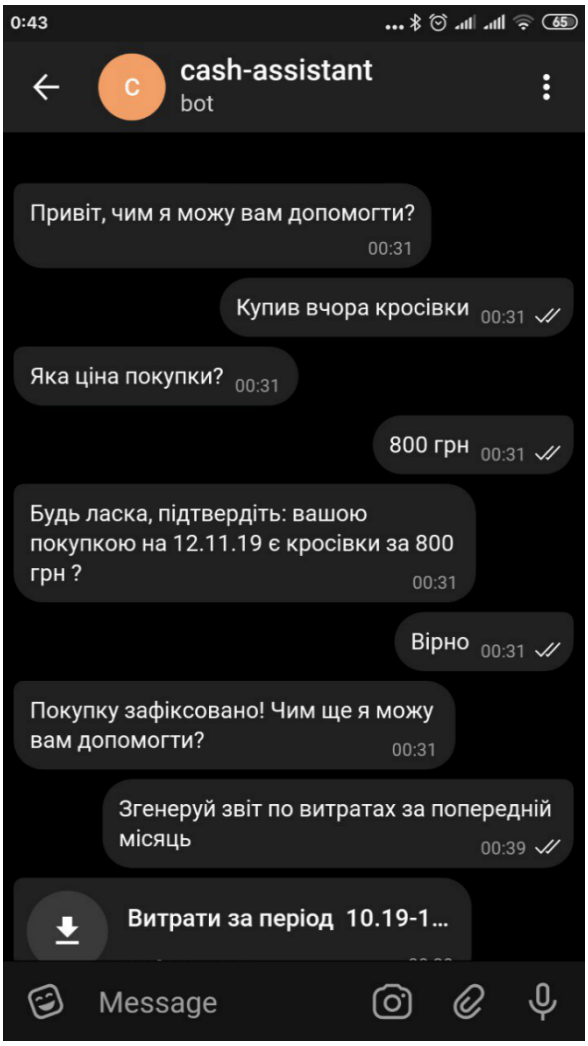

Рисунок 1.33 – Робота з асистентом у мобільному додатку

1.4.3 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 1.34).

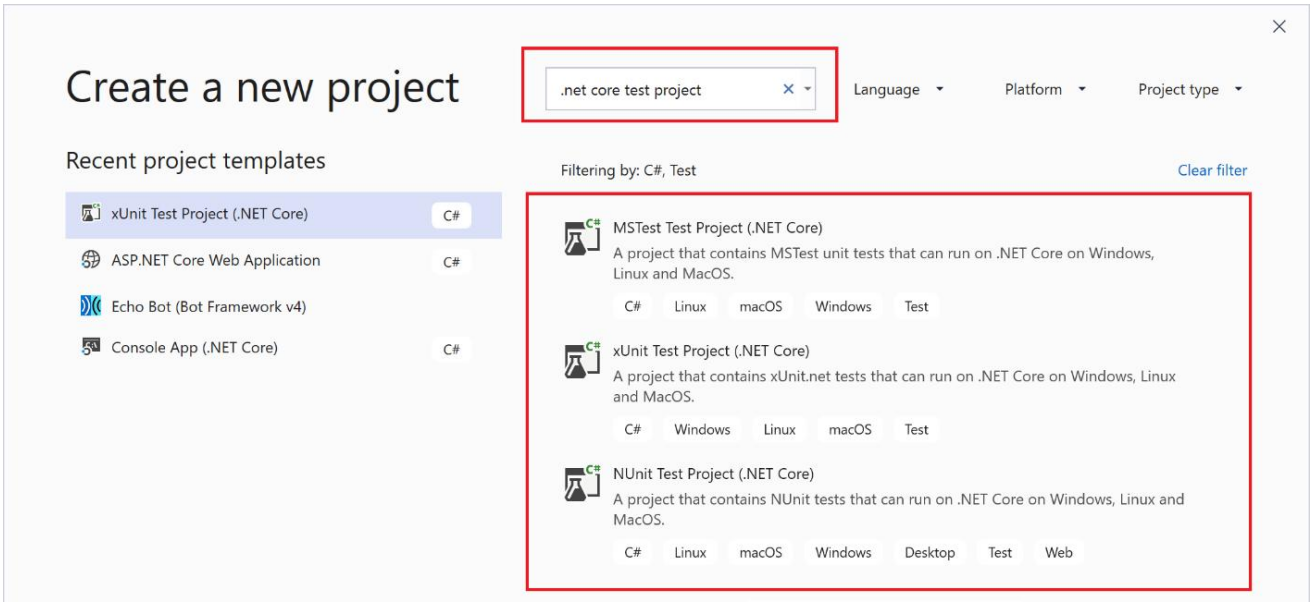

Рисунок 1.34 – Процес створення проекту для тестування

Після запуску тестів, можна спостерігати за їх прогресом виконання, та дізнатись, які не завершились успішно (Рисунок 1.35).

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

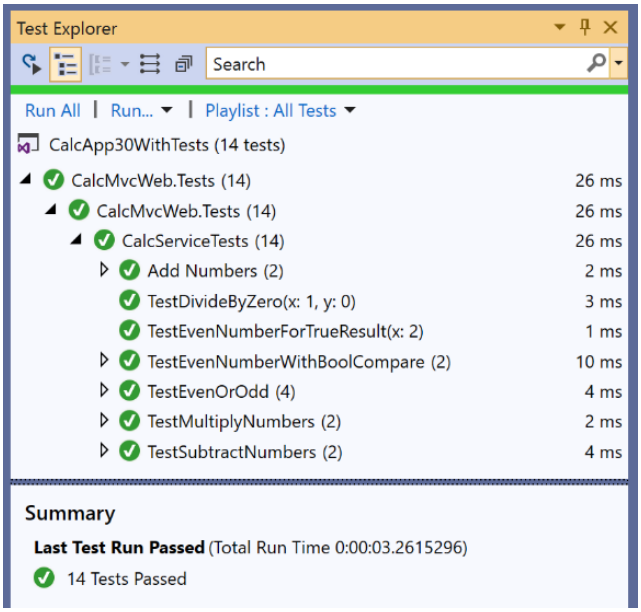

Рисунок 1.35 – Процес проходження тестів

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

2 СПЕЦІАЛЬНА ЧАСТИНА

2.1 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 (Рисунок 2.1).

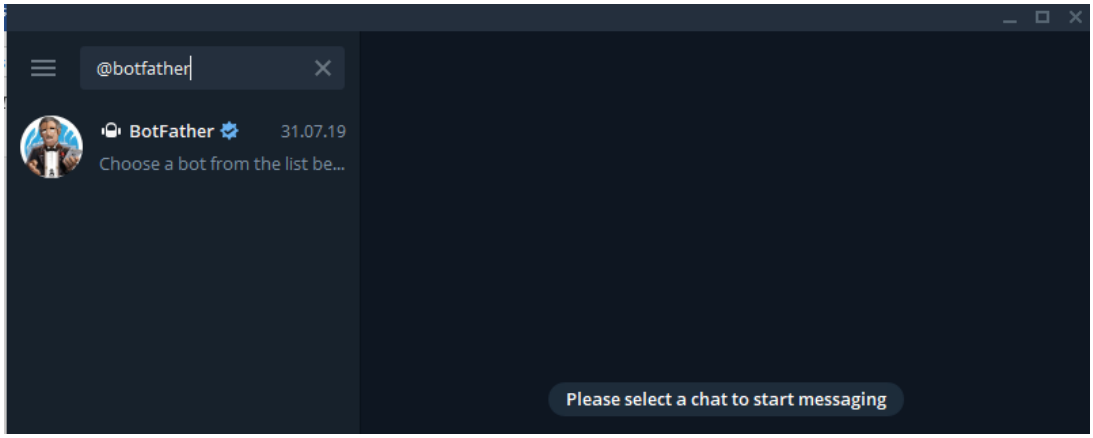

Рисунок 2.1 – Пошук каналу керування ботами

Далі потрібно вказати команду /newbot, ввести назву бота, та вказати його ідентифікатор. Весь процес створення відображено на рисунку 2.2.

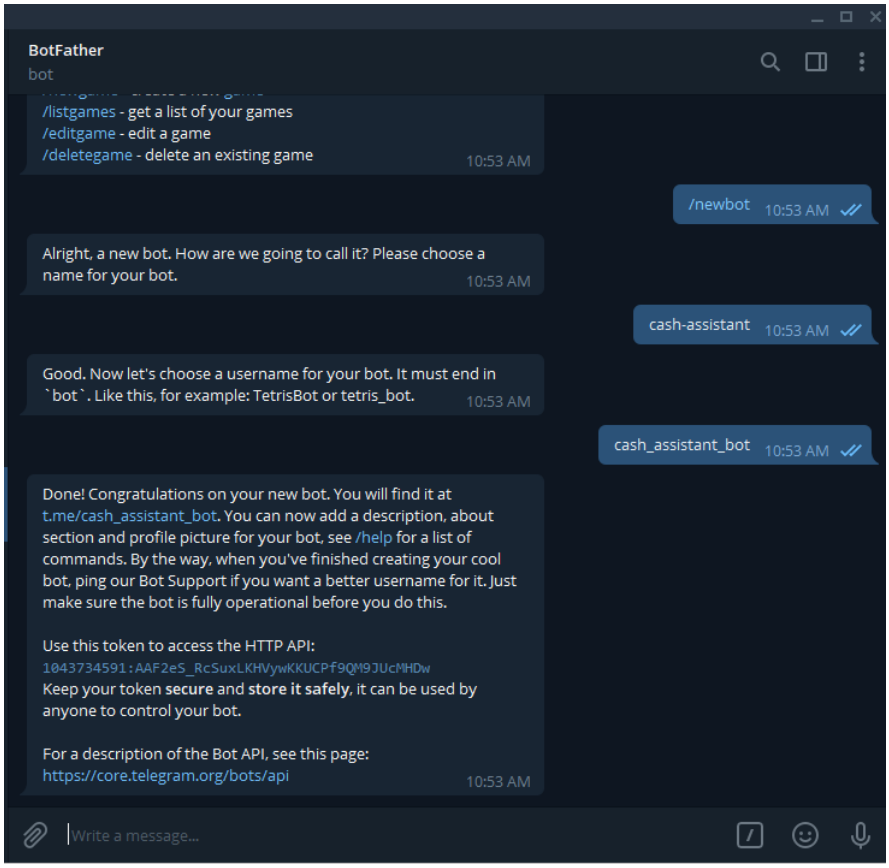

Рисунок 2.2 – Процес реєстрації бота

2.2 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 2.3. 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
        }
    ]
}
```


Рисунок 2.3 – 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 2.4.

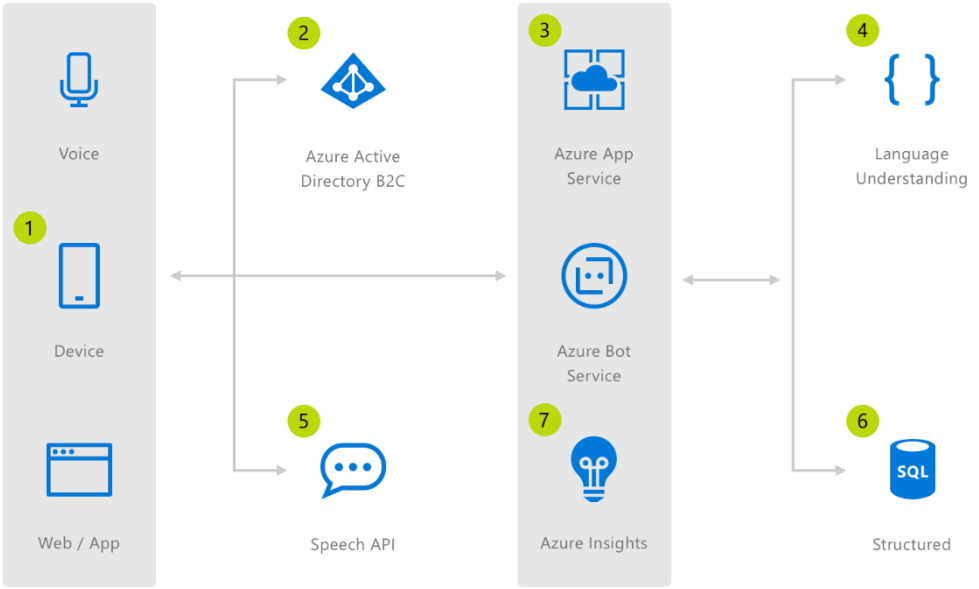

Рисунок 2.4 – Схематичне зображення можливостей LUIS та платформи Microsoft Azure

Для роботи з сервісами LUIS потрібно пройти реєстрацію на платформі Microsoft Azure Portal. Наступним кроком є, є пошук потрібних сервісів, що показано на рисунку 2.5. Після цього потрібно створити підписку та додати новий ресурс. Створення нового ресурсу зображено на рисунку 2.6.

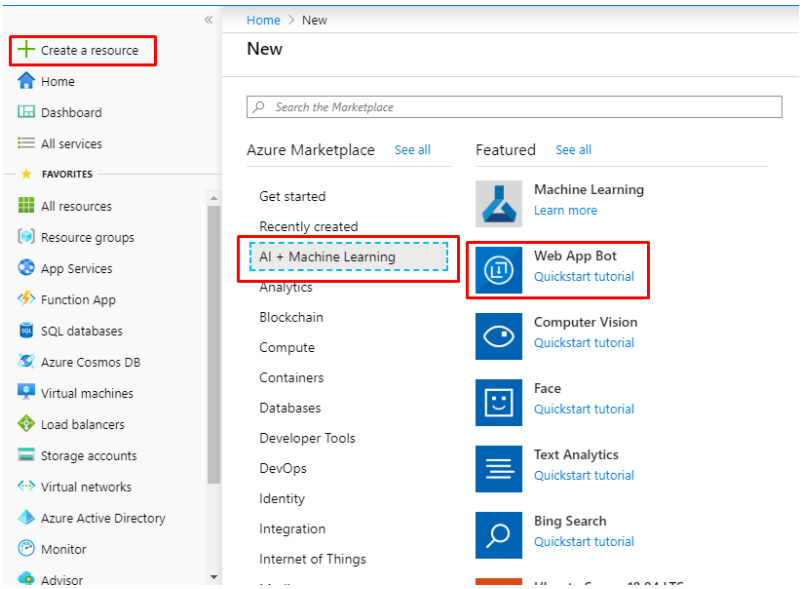

Рисунок 2.5 – Кроки для пошуку сервісів LUIS

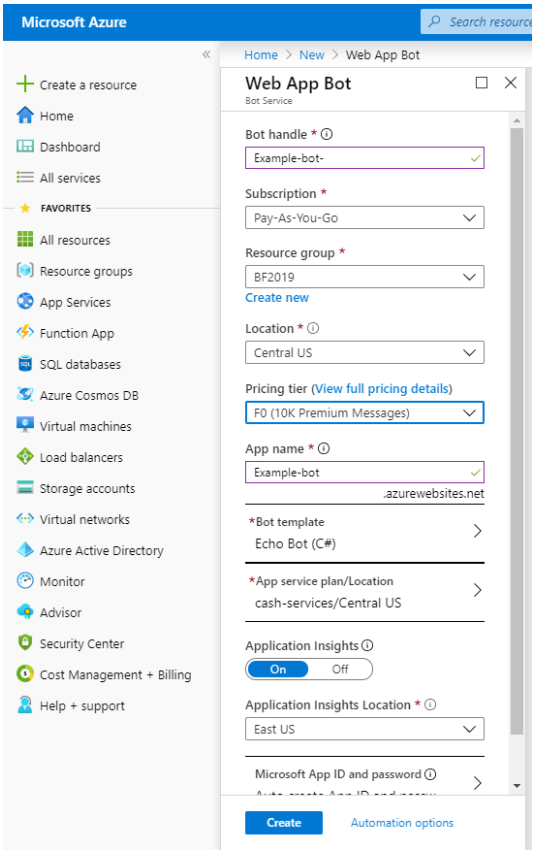

Рисунок 2.6 – Процес реєстрації нового ресурсу та додатку

Для того, щоб отримати, до якої активності відноситься вхідний текст, потрібно створити шаблони, на основі яких буде проводитись аналіз тексту. (Рисунок 2.7).

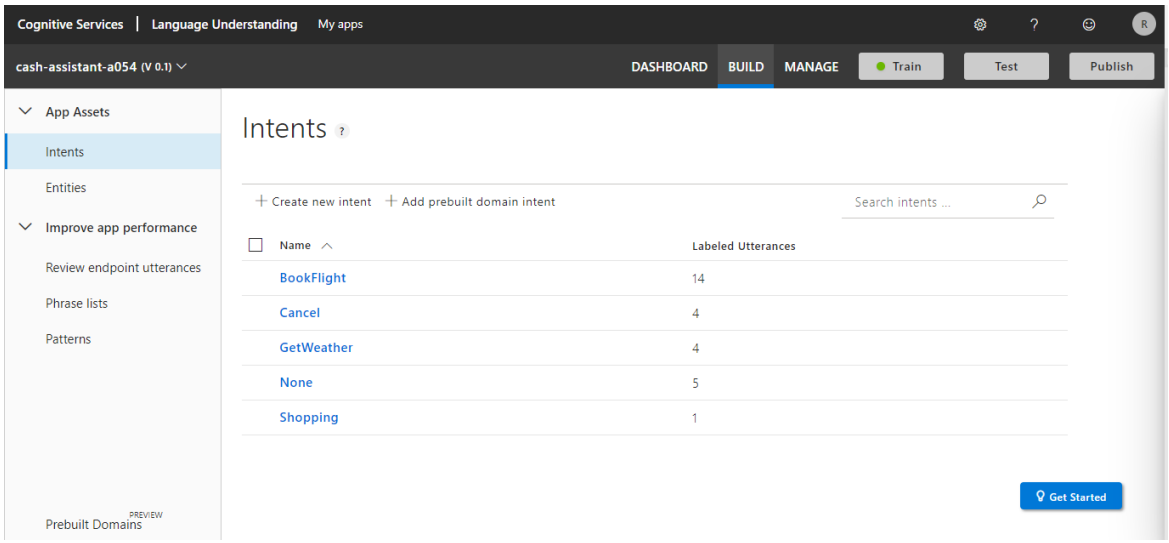

Рисунок 2.7 – Список налаштування активностей

На наступному етапі потрібно оголосити категорії, які будуть визначатись з контексту (Рисунок 2.8). У даному випадку це будуть категорії для можливих витрат, та доходів.

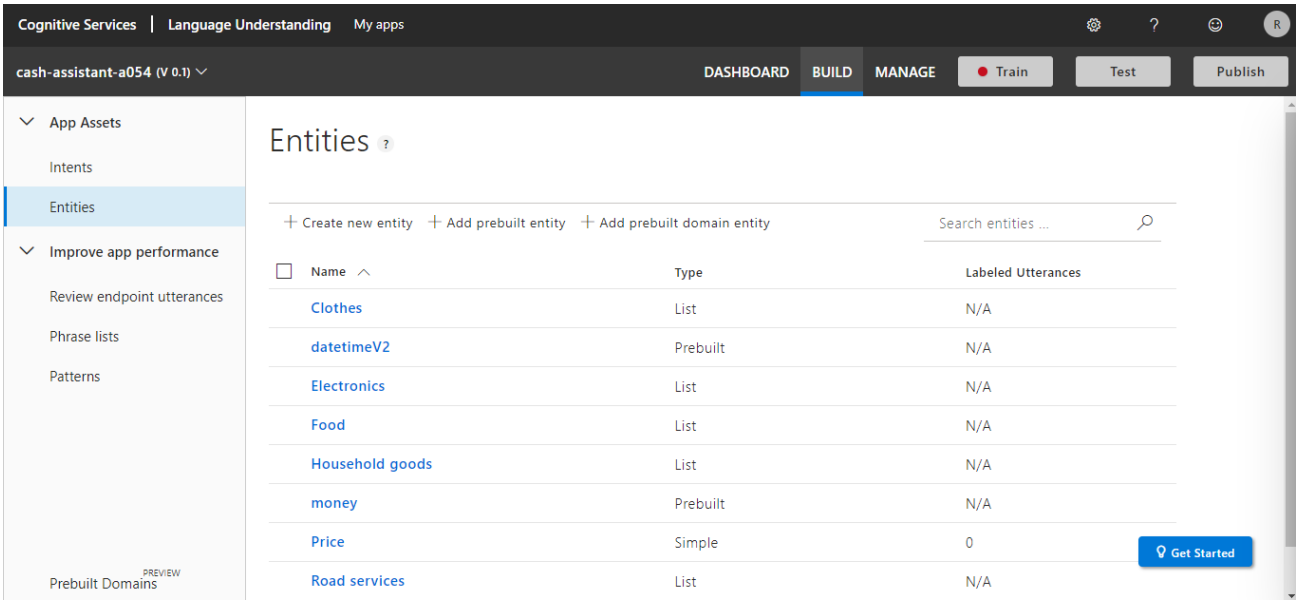

Рисунок 2.8 – Оголошення сутностей

Також кожну категорію потрібно заповнити сутностями з реального світу, для подальшого розпізнання. Їх перелік можна спостерігати на прикладі категорії «Електроніка» (Рисунок 2.9). Також, для спрощення налаштування, на основі вже введених сутностей система запропонує додати нові, що відповідають тематиці даній категорії.

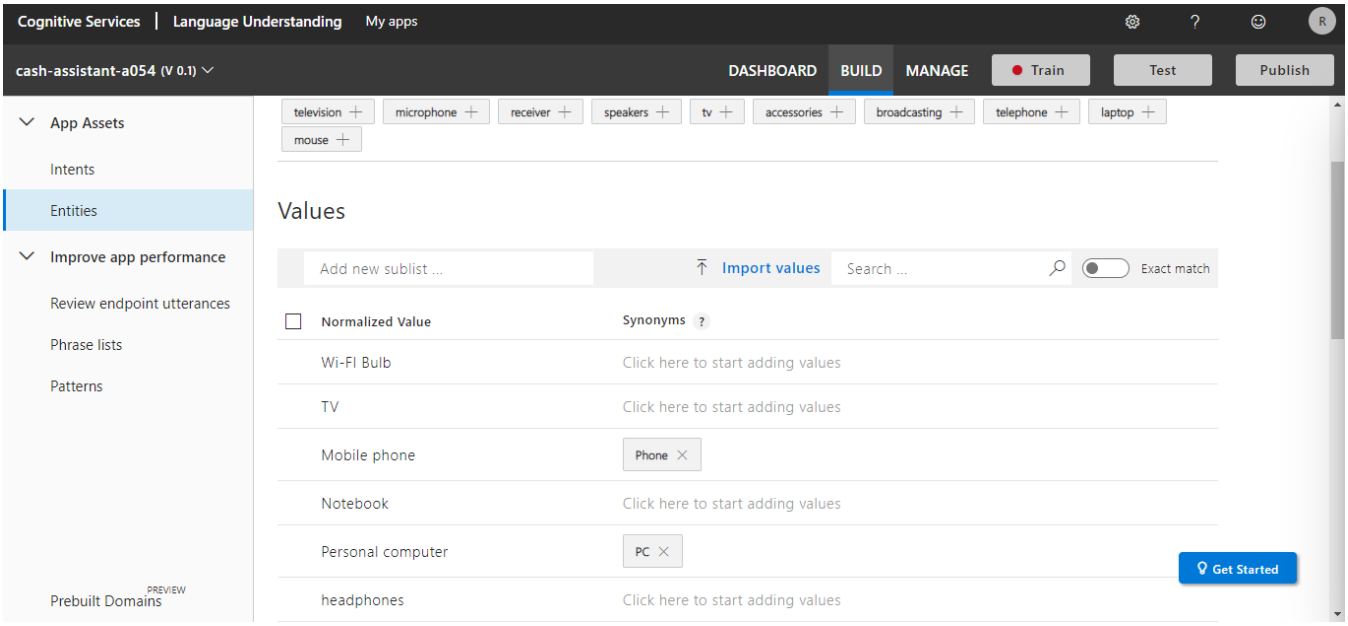

Рисунок 2.9 – Перелік об’єктів з шаблону «Електроніка»

Вказавши всі дані, потрібно описати приклади вхідних повідомлень, з використанням вказаних вище сутностей (див. рисунок 2.10). Когнітивні сервіси розпізнання тексту самі визначать, до якої з сутностей належить кожне слово.

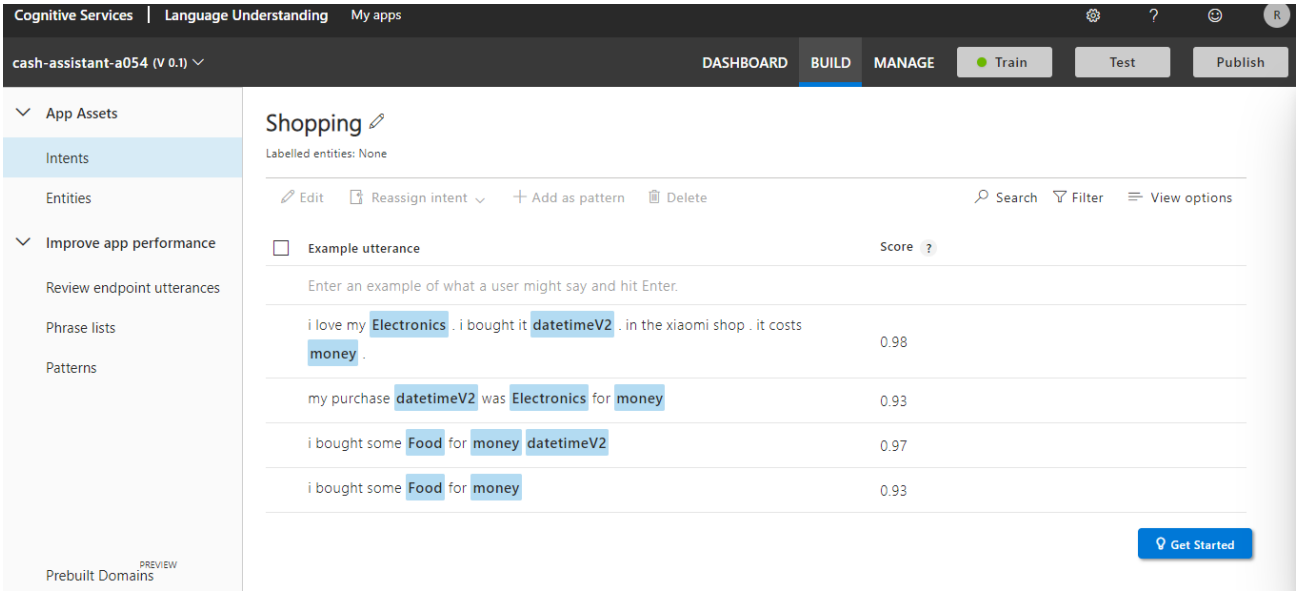

Рисунок 2.10 – Приклади вхідних даних у налаштуваннях активності

Після цього потрібно запустити процес навчання когнітивного сервісу (див. рисунок 2.11), який триватиме певний проміжок часу, та після завершення дозволить використовувати введені дані для визначення сутностей з вхідного тексту.

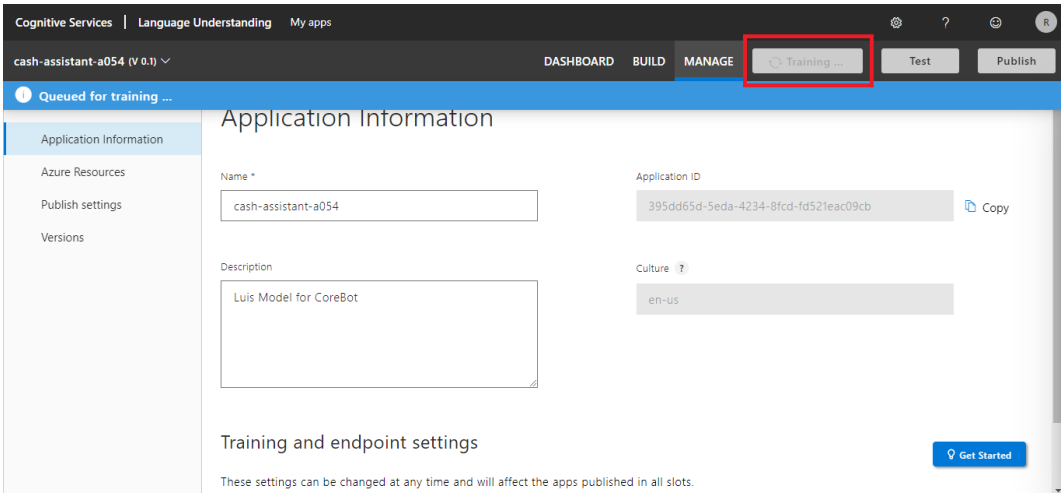

Рисунок 2.11 – Запуск процесу навчання когнітивних сервісів

Для перевірки результатів навчання на основі введених сутностей у платформі передбачено тестовий режим, що дозволяє аналізувати введений текст (див. рисунок 2.12). Даний режим визначить, до якого шаблону відноситься текст, та відобразить визначені сутності. Такий формат даних і буде повертатись у відповіді з серверу.

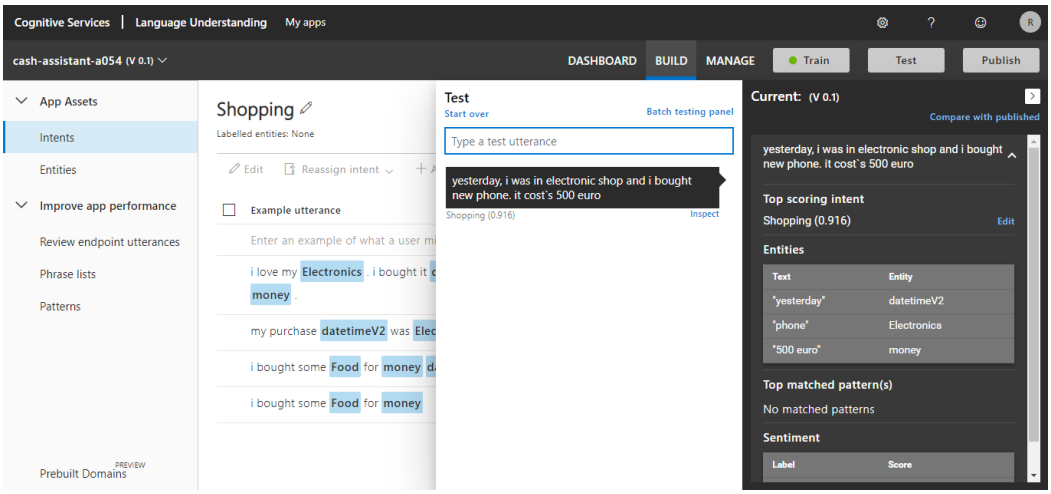

Рисунок 2.12 – Аналіз введених даних

На рисунку 2.13 графічно відображено кількість запитів до LUIS API.

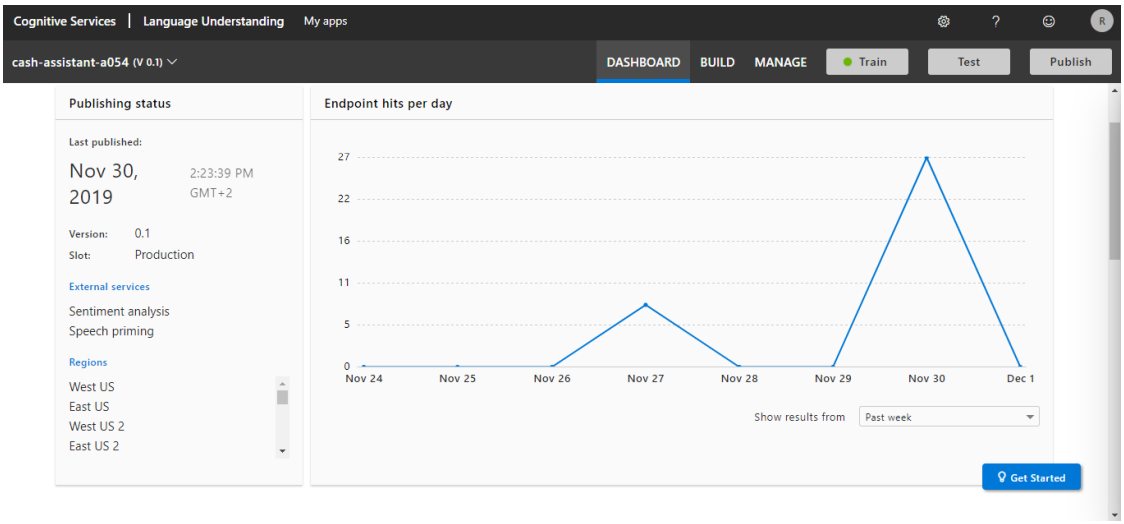

Рисунок 2.13 – Сторінка аналізу звертань до системи

3 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНА ЧАСТИНА

3.1 Загальний підхід до визначення економічної ефективності розробки

Кожна програмна система протягом свого існування проходить з певною послідовністю фази або стадії від задуму до його втілення в програми, експлуатацію та вилучення. Така послідовність фаз називається життєвим циклом розробки. На кожній фазі відбувається певна сукупність процесів, кожен з яких породжує певний продукт, використовуючи певні ресурси.

Усі продукти всіх процесів програмної інженерії являють собою певні описи — тексти вимог до розробки, погодження домовленостей, документацію, тексти програм, інструкції з експлуатації тощо.

Головними ресурсами програмної інженерії, які визначають ефективність її розробок, є час і вартість цих розробок.

Економічна ефективність - це досягнення найвищих результатів за оптимальних витрат праці. Однією з частин розробки програмного продукту є фінансові витрати на усіх етапах розробки. Тому необхідним є розробити оцінку фінансових витрат. Також виконати аналіз ефективності проекту [31].

Ефективність визначається відношенням результату (ефекту) до витрат, що забезпечили його отримання. Ефективність розкриває характер причинно-наслідкових зв'язків виробництва. Вона показує не сам результат, а те, якою ціною він був досягнутий. Тому ефективність найчастіше характеризується відносними показниками, що розраховуються на основі двох груп характеристик (параметрів) - результату і витрат.

Оцінка вартості дослідницьких розробок базується на витратному підході: використанні первісної вартості об'єктів, виходячи з фактичних витрат на розробку та доведення до комерційного використання з урахуванням

амортизації. Так, як результати роботи у вигляді математичних моделей та реалізованого на їх основі ПЗ не буде використовуватися в комерційних цілях та не підлягатиме продажу, а становить наукову та інтелектуальну цінність, то доходу від продажі ПЗ та розробки як такого не передбачається.

Згідно Статті 8 Закону № 3792-12 передбачено, що твори наукового характеру та комп'ютерні програми є об'єктами авторського права [32]. Для отримання відмінних результатів експериментів та доцільності розробки такого спеціалізованого ПЗ потрібні відповідні затрати на дослідження та розробку.

Процеси розробки програмного забезпечення в сукупності мають забезпечити шлях від усвідомлення потреб замовника до передавання йому готового продукту. На цьому шляху виділяють низку характерних робіт. Збір та аналіз вимог замовника виконавцем і подання їх у нотації, яка є зрозумілою як для замовника, так і для виконавця. Проектуванням становить перетворення вимог до розробки в послідовність проектних рішень щодо способів реалізації вимог: формування загальної архітектури програмної системи та принципів її прив'язки до конкретного середовища функціонування; визначення детального складу модулів кожної з архітектурних компонент. Тестуванням є перевірка кожного з модулів та способів їхньої інтеграції.

Це і становитиме основу витрат, які будуть здійснені протягом підготовки та виконання реалізації даного рішення. Уявно модель витрат можна поділити на дві основні частини: витрати, пов'язанані на дослідження предметної області, побудову математичних моделей, отримання попередніх результатів експериментів, та частину реалізації програмної системи, архітектури та тестування.

Враховуючи залежність якості кінцевого продукту від кваліфікації програмістів, потрібно сконцентрувати увагу на якості та результативності розробки. Для цього потрібно провести ґрунтовний аналіз предметної області, залучити найновіші та інноваційні технології, провести ґрунтовне тестування та

оцінку результатів розробки. Для забезпечення хорошої результативності при розробці доводиться йти на додаткові заходи заохочення та стимулювання у вигляді преміювання працівників, підтримання наукового дослідження досвідом іноземних науковців, дорогоцінних лабораторних дослідів.

3.2 Розрахунок вартості процесу розробки та оцінка економічної ефективності проекту

Для оцінки нематеріальних активів використовують міжнародні стандарти оцінки розрахунку вартості об’єктів інтелектуальної власності, розроблені TIAVIS (The International Assets Vabulation Standarts Commitie) [33].

Виконання розробки програмного забезпечення з огляду економічної моделі можна виконувати двома способами: процедурним та об’єктно-орієнтованим. Обидва підходи потребують залучення ресурсів у вигляді програмісті-розробників, тестувальників, керівника проекту, бізнес-аналітика. Різниця виникає в самій схемі розробки, тривалості періоду розробки та відповідній вартості. Процедурний підхід для розробки ПЗ в основі якого лежать процедури і функції передбачає розробку ПЗ як монолітного композиту, що в подальшому, як правило, вимагає великих витрат на супровід та модернізацію. Об’єктно-орієнтований підхід, що ґрунтується на основі об’єктів певних класів, що описують певну область, описують певну поведінку (методи) та володіють властивостями (атрибутами), орієнтовані на варіанти використання та покроковий процес розробки.

Для початку робіт необхідно скласти технічне завдання на розробку, яке є основним документом, що регламентує подальшу роботу, та містить докладний опис необхідних функцій програми, інтерфейс, технології, інше. Вартість складання технічного завдання переважно складає до 10% від планованої вартості розробки. Роботу зі складання технічного завдання веде керівник проекту разом із програмістами та консультуючись із замовником.

Усі програмісти, що працюють у штаті підприємства-розробника мають встановлено певний посадовий оклад. Місячний оклад, денна заробітна плата, трудомісткість (днів) і основна заробітна плата кожного учасника техпроцесу представлено у таблиці 3.1. Всі суми наведені в національній валюті – в гривні.

Таблиця 3.1 – Розрахункова вартість технологічного процесу розробки

Посада	Місячний оклад, грн.	Денна зар. плата, грн.	Об'єктно-орієнтований підхід		Процедурний підхід	
			Днів	Сума, грн.	Днів	Сума, грн.
Менеджер	13300,00	604,54	15	9068,10	17	10277,18
Програміст	15340,00	697,27	21	14642,67	25	17431,75
Тестувальник	9750,00	443,18	7	3102,26	8	3545,44
Бізнес-аналітик	10450,00	475,00	13	6175,00	13	6175,00
Додаткова зар. плата 20%			56	6597,60	63	7485,87
Фонд оплати праці 36,77%				14555,63		16515,33
Всього витрат на зар. плату				54141.26		61430,57
Військовий збір 1,5%				812,11		921.45
Єдиний соціальний внесок 3.6%				1949,08		2211,50
ПДВ, 15%				8121,18		9214,58
Всього				**65023,63**		**73778,10**

Згідно вимог та прорахованої кількості необхідних ресурсів на виконання, розробку, тестування та дослідницьку роботу було отримано основні часові рамки роботи над проектом. Так для об'єктно-орієнтованого підходу загальна тривалість роботи над ПЗ становить 56 робочих днів (під робочим днем розуміється 8-ми годинний робочий день), що включає роботу програміста, роботу тестувальника та бізнес-аналітика. Сума витрат на заробітню плату

становить 65023,63 гривень включаючи всі види додаткових оплат. Для процедурного підходу до розробки суми дещо більші, адже затрачається більше часу на розробку. Так, при використанні процедурного підходу сумарна тривалість часу розробки становить 63 робочих днів та витрати у вигляді виплат заробітної плати становлять 73778,10 гривень.

Витрати на науково-дослідницьку роботу та здійснення розробки програмних продуктів і об'єктно-орієнтованим, і процедурним способом включають:

Основна заробітна плата:

$ЗП_{осн\ 1}$ = 32988,03 грн;

$ЗП_{осн\ 2}$ = 37429,37 грн.

Додаткова заробітна плата обчислюється як $ЗП_{дод} = 0,2 \cdot ЗП_{осн}$.

$ЗП_{дод\ 1} = 0,2 \cdot 32988,03 = 6597,60$ грн;

$ЗП_{дод\ 2} = 0,2 \cdot 37429,37 = 7485,87$ грн.

Нарахування на фонд оплати праці (ФОП):

$ФОП_{ЄСВ} = 0,3677 \cdot ФЗП$

$ФОП_{ЄСВ1} = 0,3677 \cdot 39585,63 = 14555,63$ грн;

$ФОП_{ЄСВ2} = 0,3677 \cdot 44915,24 = 16515,33$ грн.

Всього витрат:

$В_{ЗП1} = ЗП_1 + ФОП_{ЄСВ1} + ЗП_{дод1} = 54141,26$ грн;

$В_{ЗП2} = ЗП_2 + ФОП_{ЄСВ2} + ЗП_{дод2} = 61430,57$ грн.

З цієї суми утримуються обов'язкові відрахування на заробітну плату: єдиний соціальний внесок, який складає 3,6% від суми нарахованої заробітної плати та податок на доходи фізичних осіб, який складає 15% від суми нарахованої заробітної плати, зменшеної на суму єдиного внеску на загальнообов'язкове соціальне страхування та податкової соціальної пільги, військовий збір у розмірі 1,5%, від суми нарахувань.

До окремих витрат також відносяться витрати на куповані вироби (матеріальне забезпечення) та спец обладнання для підтримки експерименту, накладні витрати. Витрати, що будуть супроводжувати проект розробки, порівнюватимемо в двох можливих підходах розробки.

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни (формула 3.1).

$$M_{Bi} = q_i \cdot p_i \,, \tag{3.1}$$

де q_i – кількість витраченого матеріалу i-го виду; p_i – ціна матеріалу i-го виду.

Матеріальні витрати в рамках проекту наведені в таблиці 3.2. Загальна сума матеріальних витрат становить 1105,00 гривень.

Таблиця 3.2 – Матеріальні витрати

Найменування ресурсу	Кількість, шт.	Ціна одиниці, грн	Загальна сума, грн
Папір для друку А4	1000	0,2	200,00
Тонер для принтера	1	75,00	75,00
Дошка для записів	1	750,00	750,00
Маркер для записів	4	20,00	80,00
Всього			1105,00

Розрахунок витрат на електроенергію одиниці обладнання визначаються за формулою (формула 3.2):

$$З_в = W * T * S, \tag{3.2}$$

де W – необхідна потужність, кВт; T – кількість годин роботи обладнання; S – вартість кіловат-години електроенергії, S = 2,50 грн/кВт·год.

$З_{в1}$ = 0.7 * 448 * 2.50 = 784 грн;

$З_{в2}$ = 0.7 * 504 * 2.50 = 882 грн;

Розрахунок суми амортизаційних відрахувань. Комп'ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 %, вартість яких перевищує 1000 грн. і визначається (формула 3.3):

$$A = \frac{C_Б \cdot N_A \cdot T_{ФАК}}{T_{ГОД}} \qquad (3.3)$$

де $C_Б$ – балансова вартість обладнання, грн; N_A – норма амортизаційних відрахувань в рік, %; $T_{ФАК}$ – фактичний час роботи обладнання по написанню програми, год; $T_{ГОД}$ – річний робочий фонд часу, год.

У даній формулі норма відрахувань на амортизацію рівна N_A = 0,6. Балансова вартість обладнання вказана в таблиці 3.3 і рівна $C_Б$ = 30015,00 гривень. Річний робочий фонд часу приймемо за $T_{ГОД}$ = 2320 годин. З них реальний фактичний робочий час становить $T_{ФАК}$ = 448 години згідно об'єктно-орієнтованого підходу та $T_{ФАК}$ = 504 години згідно процедурного підходу.

Згідно вищезгаданої формули витрати на амортизацію становлять 3447,60 гривень та 3912,30 гривень для кожного підходу відповідно.

Таблиця 3.3 – Перелік необхідного обладнання

Найменування	Кількість, шт	Ціна, грн	Сума, грн

Комп’ютер	2	12300,00	24600,00	
Принтер	1	5415,00	5415,00	
Середовища розробки	2	безкоштовно	безкоштовно	
Операційна система (Linux)	2	безкоштовно	безкоштовно	
Всього більше 1000 грн.			30015,00	
Всього витрат на амортизацію			3447,60	3912,30
Всього			33462,60	33927,30

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління та створення необхідних умов праці та закупівлю ресурсів та обладнання для розробки наведені в таблиці 3.3.

Залежно від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників. Нехай вона буде дорівнювати 25%, що становить 9896,40 грн для об’єктно-орієнтованого і 11228,81 грн для процедурного підходу розробки.

Проведемо розрахунок вартості створюваного програмного продукту. Вартість продукції включає у собі собівартість і планований прибуток. Найважливішим моментом для розробника, з економічної точки зору, є процес встановлення ціни.

Можна отримати загальні значення витрат на розробку та реалізацію проекту, враховуючи всі вище описані затрати та нарахування. Цей вид витрат складається з сум витрат на оплату праці (всього витрати на оплату праці), матеріальні затрати, затрати на електроенергію, накладні витрати, витрати на обладнання, враховуючи амортизації обладнання на час виконання проекту.

Собівартість продукції – це сума грошових витрат підприємства (фірми) на виробництво і збут одиниці продукції, виконання робіт та надання послуг.[34]

Повна собівартість програмного продукту дорівнює сумі усіх витрат на його виробництво: 87613,05 грн використовуючи об’єктно-орієнтований підхід, 92170,85 грн при процедурному підході розробки.

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Економічна ефективність (Е) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E = \frac{П}{Св} \tag{3.4}$$

де $П$ – прибуток, $П = В - Св$; $Св$ – собівартість.

У випадку даної розробки, маючи некомерційний проект без економічно корисного результату, можна прогнозувати, що економічна ефективність прямує до 0 у обох випадках. Однак це не є причиною для негативного економічного висновку щодо даного проекту, адже такого плану розробки приносять користь у вигляді інтелектуальних ресурсів, і, переважно, фінансуються або виконуються на замовлення організацій, зацікавлених в отриманні результатів досліджень та розробок. Фінансування не можна вважати отриманим доходом від реалізації. Однак за надходження коштів на реалізацію ззовні можна вважати ефективність проекту рівною 1 (Е = 1), що означає перекриття витрат на розробку у повній мірі, тобто фінансування.

Якщо ринкова вартість програмного продукту рівна прийнятій, то економічна ефективність визначається встановленим рівнем прибутку. Поряд із

економічною ефективністю розраховують термін окупності капітальних вкладень ($T_{ок}$):

$$T_{ок} = \frac{1}{E} \tag{3.5}$$

У нашому випадку прямого прибутку не існує. Прибуток можна прогнозувати на підприємствах, організаціях чи відомствах, що зацікавлені в дослідженні. Окупність же для розробки даного ПЗ за ефективності рівній одиниці можна вважати теж рівній 1 згідно формули 3.4.

Виходячи із експертних оцінок і складності програми, приймемо величину витрат на супровід і модернізацію програмного забезпечення, створеного за об'єктно-орієнтованим методом 20% (22054,32 грн) від початкових витрат, а за процедурним – 25% (30230,30 грн).

Однак варто зауважити, що розробка спрямована на короткотривалу підтримку та не прогнозує модернізації. У разі ж необхідності розробки такого ж або суміжного ПЗ можна вважати за доцільно розпочинати розробку з початкових етапів, що потягне за собою нові витрати у повній мірі. Тобто у разі короткотермінової підтримки все ж за доцільніше обирати об'єктно-орієнтовану модель розробки, адже вартість короткотермінової підтримки згідно цієї моделі не вплине значно на сукупну вартість розробки.

Здана в експлуатацію система не завжди цілком завершена, її треба змінювати протягом терміну експлуатації. Внаслідок змін система стає більш складною і погано керованою. Об'єктно-орієнтоване представлення програми дозволяє навіть середньому програмісту швидко і ефективно супроводжувати і модернізувати програми, що значно скорочує подальші витрати на супровід і модернізацію [35].

Сумарні дані економічного розрахунку розробки даного проекту наведені в таблиці 3.4.

Таблиця 3.4 – Загальні витрати

Вид витрат	Об’єктно-орієнтований підхід, грн	Процедурний підхід, грн
Зарплата основна	32988,03	37429,37
Зарплата додаткова	6597,60	7485,87
Фонд заробітної плати	14555,63	16515,33
Відрахування на ФОП	54141,26	61430,57
Разом на оплату праці	65023,63	73778,10
Матеріальні витрати	1105,00	1105,00
Електроенергія	784,00	882,00
Амортизація	3447,60	3912,30
Накладні витрати	9896,40	11228,81
Обладнання	30015,00	30015,00
Разом на ін. витрати	45248,00	47143,11
Собівартість	110271,63	120921,21
Прибуток	відсутній	відсутній
Вартість розробленого ПЗ	**110271,63**	**120921,21**
Модернізація і супровід	22054,32	30230,30
Загальні витрати на розробку	132325,95	151151.51
Економія ($ЗВ_1$-$ЗВ_2$)		**18825.56**

Загальна вартість пропонованих робіт становить 151151.51 гривень для процедурного і 132325,95 гривень для об’єктно-орієнтованого підходів розробки. В даному випадку реалізації проекту варто вибрати об’єктно орієнтований підхід для розробки даного ПЗ, адже фінансово це більш вигідно.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

В межах даної роботи ведеться розробка персонального фінансового асистента з використанням когнітивних технологій. Згідно до поставлених задач, користувачі даної системи будуть використовувати месенджер, для фіксації своїх витрат. Одними з пристроїв, на які можна встановити месенджер є персональний комп'ютер та смартфон. Отже, користувачі даної програмної системи повинні дотримуватися правил та норм поведінки з комп'ютерною технікою та портативними пристроями.

Вимогами безпеки робочих місць працівників з екранними пристроями вимоги безпеки під час роботи з екранними пристроями є:

1. Щодня перед початком роботи необхідно очищати екранні пристрої від пилу та інших забруднень.

2. Після закінчення роботи екранні пристрої слід відключати від електричної мережі.

3. У разі виникнення аварійної ситуації необхідно негайно відключити екранний пристрій від електричної мережі.

4. Не допускається:

- виконувати технічне обслуговування, ремонт і налагодження екранних пристроїв безпосередньо на робочому місці працівника під час роботи з екранними пристроями;
- відключати захисні пристрої, самочинно проводити зміни у конструкції та складі екранних пристроїв або їх технічне налагодження;

- працювати з екранними пристроями, у яких під час роботи виникають нехарактерні сигнали, нестабільне зображення на екрані та інші несправності.

Також необхідно постійно слідкувати за станом екрану монітора: він має бути чистим, без плям та пилу. Крім того, обов'язково потрібно слідкувати за чистотою окулярів – комп'ютерних чи звичайних. У робочому приміщенні, де встановлені комп'ютери, щодня потрібно виконувати вологе прибирання потрібно щогодини. У процесі роботи рекомендується періодично (приблизно раз на 20-30 хвилин) переводити погляд з екрану на найбільш віддалений предмет у кімнаті, а ще краще – на віддалений об'єкт за вікном. [36]

Вимоги безпеки до екранних пристроїв є наступними:

1. Екранні пристрої не мають бути джерелом ризику для працівників.

2. Усе випромінювання, за винятком видимої частини електромагнітного спектра, має бути зведене до незначного рівня з погляду безпеки і охорони здоров'я працівників.

3. Символи на екранних пристроях мають бути чіткими, відповідного розміру. Між символами і рядками символів має бути належна відстань.

4. Зображення на екрані має бути стабільним, без миготінь або інших видів нестабільності.

5. Яскравість та/або контрастність символів має легко регулюватися працівником під час роботи з екранними пристроями, а також швидко адаптуватися до навколишніх умов.

6. Вибираючи екрани, слід надавати перевагу таким екранам, які легко та вільно повертаються і нахиляються відповідно до потреби працівника.

7. За необхідності може використовуватись окрема підставка або регульований стіл для розміщення екрана.

8. Екран не має відблискувати або відбивати світло, щоб не викликати дискомфорту у працівника під час роботи з екранними пристроями.

9. Вибираючи клавіатуру, слід надавати перевагу такій клавіатурі, яка відкидається і є автономною (відокремленою від екрана), щоб працівник міг вибрати зручну робочу позу й уникнути втоми рук (кисті і верхньої частини руки).

10. Поверхня клавіатури має бути матовою, щоб уникнути віддзеркалювання. Розташування клавіш і самі клавіші мають полегшувати роботу із клавіатурою. Позначення клавіш повинно бути достатньо контрастним і розбірливим. Клавіатуру слід розташовувати на поверхні столу на відстані 100-300 мм від краю, зверненого до користувача, або на спеціальній, регульованою по висоті робочої поверхні, відокремленої від основної стільниці.

11. Устаткування, яке входить до робочої станції, не має виділяти надлишкового тепла, що може спричинити незручності працівникам під час роботи з екранними пристроями.

12. Під час розробки, вибору, замовлення та модифікації програмного забезпечення, а також під час розробки завдань, що передбачають використання устаткування з екранними пристроями, роботодавець має керуватися таким програмним забезпеченням, яке відповідає розв’язуваним завданням і є простим у використанні, а де необхідно - адаптованим до рівня знань і досвіду працівника.

У разі виникнення ситуацій, які суперечать нормам охорони праці та можуть бути причиною негативних наслідків чи завдати шкоди, потрібно припинити роботу з системою та повідомити керівника роботи чи особу, що відповідає за охорону праці в лабораторії, про порушення та проблеми. Під час розробки, тестування та впровадження автоматизованої системи були дотримані всі вимоги, норми та державні стандарти з охорони праці.

4.2 Підвищення стійкості роботи об'єктів господарської діяльності в воєнний час

Ефективність економіки держави залежить від того, наскільки окремі галузі господарства здатні стійко працювати не тільки у звичайних умовах, а й в умовах НС мирного та воєнного часу. Значні руйнування, пожежі та втрати серед населення, викликані наслідками НС, або воєнними діями можуть стати причиною різкого скорочення випуску промислової та сільськогосподарської продукції, а отже і зниження економічного потенціалу держави. Виникає потреба завчасного вживання заходів щодо забезпечення стійкої роботи промислових об’єктів на випадок виникнення НС або воєнних дій.

Під стійкістю роботи об'єкта розуміється його спроможність в умовах надзвичайної ситуації випускати продукцію в запланованому обсязі та номенклатурі, а при отриманні середніх руйнувань або порушенні зв'язків з кооперації та поставок відновлювати виробництво у мінімальні терміни [37].

Під стійкістю роботи об'єктів, які безпосередньо не виробляють матеріальні цінності, розуміється їх спроможність виконувати свої функції в умовах надзвичайних ситуацій. На стійкість роботи об'єкта в умовах надзвичайних ситуацій впливають наступні фактори:

- надійність захисту робітників та службовців;
- стійкість інженерно-технічного комплексу (будівель, споруд, систем енерго-газо- та водопостачання і каналізації, технологічного обладнання) до дій сил стихійних явищ природи, аварій та катастроф, а у воєнний час - вражаючих факторів зброї масового ураження;
- захищеність об'єкта від вторинних уражуючих факторів (пожеж, вибухів, зараження ОР та СДОР);
- надійність системи забезпечення об'єкта всім необхідним для виробництва (сировиною/ паливом, комплектуючими вузлами і деталями, електроенергією, водою, газом та іншим);

- стійкість та безперервність управління виробництвом та заходами цивільної оборони;
- підготовленість об'єкта до ведення рятувальних та інших невідкладних робіт до поновлення порушеного виробництва. Перелічені фактори є основними і загальними для усіх об'єктів.

Здійснення організаційних заходів передбачає завчасну підготовку всіх структур цивільного захисту, служб і формувань до надзвичайних ситуацій та воєнних дій.

Вжиттям технологічних заходів підвищується стійкість роботи об'єктів шляхом змінювання технологічних процесів, режимів, можливих в умовах надзвичайних ситуацій. Інженерно-технічні заходи мають забезпечити підвищену стійкість виробничих споруд, технологічних ліній, устаткування, комунікацій об'єкта до впливу уражаючих факторів під час надзвичайних ситуацій та воєнних дій.

При проведенні цих заходів необхідно враховувати конкретні умови об'єкта господарської діяльності. Проте є загальні організаційні інженерно-технічні заходи, які мають проводитись на всіх об'єктах.

Забезпечення захисту людей та їх життєдіяльності передбачає за собою наступні речі:

- створення на об'єкті надійної системи оповіщення про загрозу нападу противника, радіоактивне забруднення, хімічне і біологічне зараження, загрозу стихійного лиха і виробничої аварії;
- організація розвідки і спостереження за радіоактивним забрудненням, хімічним і біологічним зараженням; гідрометеорологічне спостереження за рівнем води, напрямком і швидкістю вітру, рухом і поширенням хмари радіоактивного забруднення, СДЯР і ОР;
- створення фонду захисних споруд ЦО, запасів засобів індивідуального захисту і забезпечення своєчасної видачі їх населенню;

- завчасна підготовка до масової санітарної обробки населення і знезаражування одягу, організація взаємодії з установами охорони здоров'я для медичного обслуговування населення у надзвичайних ситуаціях;
- підготовка до евакуації населення, розміщеного в зонах можливих руйнувань і катастрофічного затоплення. Завчасна підготовка місць евакуації, організація прийому евакуйованого населення на територію населених пунктів;
- постачання населення продуктами харчування, питною водою, предметами першої необхідності; комунальне побутове обслуговування населення з урахуванням проведення евакуаційних заходів, забезпечення захисту продовольчих запасів;
- навчання населення способам захисту, надання першої допомоги, практичним діям в умовах надзвичайних ситуацій, морально-психологічна підготовка населення для виживання;

Для забезпечення чіткої інформації про обстановку та правила дій і поведінки населення в надзвичайних ситуаціях мирного і воєнного потрібно дотримуватись наступних вимог.

Захистити цінне і унікальне устаткування можна завдяки проведенню інженерно-технічних заходів, щоб зменшити небезпеку пошкодження і руйнування цінного й унікального устаткування, станків з програмним керуванням, шліфувальних, токарних, розточних, зубофрезерних, пресових станків, автоматичних конвеєрних ліній та іншого устаткування. Варіантами такого захисту є розміщення зазначеного устаткування в заглиблених приміщеннях а також використання спеціальних захисних пристосувань, закріплення станків на фундаментах, застосування контрфорсів для підвищення стійкості проти перекидання обладнання.

Для стійкості роботи автотранспортної та іншої техніки, технологічного обладнання і механізмів потрібно:

- Реалізувати своєчасне оповіщення гаража, технологічного парку, їх керівників, водіїв, механізаторів про загрозу надзвичайної ситуації.
- Підготовку автотранспортної техніки до проведення робіт в умовах радіоактивного забруднення, хімічного біологічного зараження і світломаскування.
- Пристосувати усі види транспортних засобів для евакуації населення і перевезення потерпілих.
- Розробити заходи з метою пристосування автотранспортної, іншої техніки для виконання завдань ЦЗ.

Розробка пристосувань і технологічних процесів для відбору потужностей тракторів і автомобілів з метою приведення в дію електрогенераторів і технологічного обладнання, насосів для подачі води до місця споживання зі свердловин, відкритих водойм і шахтних колодязів.

Для забезпечення виробництва продукції необхідні електроенергія, паливо, мастила, засоби захисту рослин, міндобрива, профілактичні й лікувальні препарати ветеринарної медицини, запасні частини, сировина та інші матеріально-технічні засоби. Забезпечення об'єктів цими ресурсами дасть можливість випускати необхідну продукцію в надзвичайних умовах мирного і воєнного часу. Тому повинні проводитись такі заходи, які б забезпечили стійкість постачання і сприяли підвищенню захисту мережі електро-, водо-, газопостачання, транспортних комунікацій і джерел постачання всім необхідним для забезпечення функціонування галузей сільського господарства в надзвичайних умовах.

З метою попередження аварій на електричних мережах необхідно встановити автоматичну систему відключення перенапруги. Повітряні лінії електропостачання слід замінити на підземно-кабельні.

Газ використовується як паливо і на хімічних підприємствах у технологічному процесі. Для безперебійного забезпечення газом, газові мережі необхідно підводити до об'єкта з двох напрямків, які мають бути з'єднані в єдине кільце з обладнанням для можливого дистанційного автоматичного управління й у разі необхідності відключення пошкоджених ДІЛЯНОК.

На великих підприємствах необхідно мати підземні ємності із закачаним резервним газом. На підприємствах, де використовується пара, необхідно захистити джерела його постачання, заглибити в ґрунт комунікації паропостачання і встановити запірні пристосування.

Запас резервних матеріалів необхідно розраховувати на такі строки роботи підприємства, за які можливе відновлення регулярного постачання.

Передбачити, на випадок перебоїв в постачанні підприємствами-суміжниками, створення місцевих матеріалів, сировини для виготовлення комплектуючих виробів і інструментів силами свого підприємства.

Підготовка всієї техніки для проведення рятувальних та інших невідкладних робіт у надзвичайних умовах мирного і воєнного часу.

З перерахованих факторів впливають наступні шляхи і способи підвищення стійкості роботи галузей господарської діяльності та територій:

1) Обмеження подальшого росту великих промислових міст і виробничих сил на території країни;
2) нагромадження фонду захисних споруд та засобів індивідуаль-ного захисту (ЗІЗ);
3) будівництво найважливіших ОГД за межами зон можливих руйнувань
4) будівництво підприємств-дублерів;
5) розширення шляхів сполучення і розвиток всіх видів транспорту;
6) підсилення та дублювання енергетичних потужностей;
7) розширення міжгалузевих зв'язків;
8) створення матеріально-технічних резервів;

9) підтримування сил забезпечення життєдіяльності у надзвичай-них ситуаціях у постійній готовності;

10) навчання населення правилам дій по сигналах оповіщення, використанню засобів захисту, наданню самозахисту і взаємодопомоги.

Таким чином, підвищення стійкості роботи об’єкту народного господарської діяльності у надзвичайних ситуаціях досягається завчасним проведенням комплексу організаційних заходів, інженерно-технічних та технологічних, скерованих на максимальне зниження впливу сил стихійних явищ природи, аварій та катастроф, а у воєнний час - вражаючих факторів зброї масового ураження.

ВИСНОВОК

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.

ПЕРЕЛІК 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] - https://web.archive.org/web/20100106115112/http://www.utexas.edu/its/archive/windows/database/datamodeling/rm/rm7.html
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. Методичні вказівки для виконання розділу дипломної роботи щодо техніко-економічного обґрунтування вибору проектного рішення розробки та оцінки якості програмного забезпечення / Упор. Петрик М.Р., Кінах Я.І., Головатий А.І., Рогатинська Л.Р. – Тернопіль: Вид-во ТНТУ ім. І. Пулюя. – 2013. – 34 с.
32. Про авторське право і суміжні права [Електронний ресурс] - https://zakon.rada.gov.ua/laws/show/3792-12
33. International Valuation Standards [Електронний ресурс] - https://www.ivsc.org/standards/international-valuation-standards
34. 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.
35. Armstrong, Deborah J. (February 2006). The Quarks of Object-Oriented Development. Communications of the ACM 49 (2): 2006-08-08.
36. 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
37. Стійкість функціонування об'єктів економіки в НС [Електронний ресурс] - https://studme.com.ua/1098120514395/bzhd/ustoychivost_funktsionirovaniya_obektov_ekonomiki.htm

ДОДАТКИ

Додаток А

Міністерство освіти і науки України

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії

Кафедра програмної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедрою

програмної інженерії

"___ "_____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської дипломної роботи

на тему: «Розробка персонального фінансового асистента з використанням когнітивних технологій»

Олещука Романа Станіславовича

Керівник роботи:

к.т.н., доцент Михалик Д. М.

"___" _____ 2019р.

_____

Виконавець:

студент групи СПм-62

Олещук Роман Станіславович

"___" _____ 2019р.

_____

м. Тернопіль – 2019

ЗМІСТ

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану на 2019 рік, та згідно наказу на виконання дипломної роботи студента-магістра.

Тема проекту: «Розробка персонального фінансового асистента з використанням когнітивних технологій».

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Дипломна робота 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.

7. Мета роботи спростити ведення особистого бюджету за допомогою програмної системи, яка завдяки інтеграції у месенджери, має можливість запускатись на будь-якій сучасній платформі.
8. 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.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Функціональні характеристики

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

3.2 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

3.3 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.

Приймання проводиться спеціально створеною екзаменаційною комісією в термін до:

“26” грудня 2019р.

ДОДАТОК В

Текст тез

УДК 004.422.83, 004.81

Р. Олещук, Д. Михалик, канд. техн. наук, доц.
Тернопільський національний технічний університет імені Івана Пулюя, Україна

РОЗРОБКА ПЕРСОНАЛЬНОГО ФІНАНСОВОГО АСИСТЕНТА З ВИКОРИСТАННЯМ КОГНІТИВНИХ ТЕХНОЛОГІЙ.

R. Oleshchuk, D. Myhalyk, Ph.D, Assoc. Prof.

DEVELOPMENT OF A PERSONAL FINANCIAL ASSISTANT USING COGNITIVE TECHNOLOGIES.

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

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.

Згідно з результатими досліджень агенства мобільного маркетингу LEAD9, в 2018 році близько 45% усіх дорослих мешканців України користуються смартфонами з сенсорним екраном. Близько 91% власників смартфонів, користуються месенджерами [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.

Для реалізації асистента, використання лише одного месенджера, не є достатнім. Для того, щоб отримати потрібну інформацію про витрати з команд, які користувач відправляє у повідомленні, потрібно визначити з тексту головні сутності, та співставити їх з елементами у програмі, після цього внести їх у базу даних, для подальшого опрацювання. Саме для того, щоб розпізнати текст, використовуються когнітивні служби LUIS, на платформі microsoft azure. LUIS - це хмарний API сервіс, який застосовує спеціальну технологію машинного навчання для розмовної мови, що дозволяє передбачити загальний зміст тексту, та отримати детальну інформацію з нього [2].

Підсумовуючи весь матеріал можна дійти висновку, що розробка фінансового асистента, що інтегрується в меседжер є актуальною та необхідною для сучасного розвиненого суспільства. Адже це допоможе та спростить керування особистими фінансами, насамперед, для молоді.

Література

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

Лістинг кому для перекладу тексту

```
     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);
        }
```

ДОДАТОК D

диск