Київський національний університет імені Тараса Шевченка
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

СТВОРЕННЯ ПРОГРАМНОГО КОМПЛЕКСУ ДЛЯ ГЕНЕРАЦІЇ, ВІДПРАВКИ ТА ОТРИМАННЯ ЗВІТІВ

Дипломна робота магістра
студента 2 року навчання
спеціальність: 123 «Комп'ютерна інженерія»
Андрія ЛУК'ЯНЕНКА

Науковий керівник
канд. фіз.-мат. наук,
доцент Сергій ЗАГОРОДНЮК

Рецензент
канд. фіз.-мат. наук Всеволод ДЕМИДОВ
доцент кафедри геоінформатики
ННІ «Інститут геології»

До захисту допускаю: Завідувач кафедри

Юрій БОЙКО /

Ухвалено на засіданні кафедри "____" _____ 2022 р., протокол № ___

Київ - 2022

РЕФЕРАТ

Випускна кваліфікаційна робота магістра: 51 сторінка, 20 рисунків, 3 таблиці, 7 джерел.

В ході виконання дипломної роботи модернізована і розширена архітектура існуючої системи для генерації, передачі та аналізу звітів. Впроваджені необхідні зміни, що усунули недоліки системи та підвищили зручність і продуктивність її експлуатації.

Змінено принцип роботи з записами для генерації звітів. Для цього розширено структуру роботи з записами, змінено механізми визначення надлишкових записів та механізм визначення цінових параметрів записів.

Розроблено функціонал для генерації та відправлення звітів, а також механізм оновлення таблиць з даними для генерації і перевірки звітів.

Ключові слова: звіт, контракт, чат-бот, таблиця, запит, telegram, python, pandas, aiogram.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ 4

ВСТУП 5

МЕТА РОБОТИ 6

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 7

1.1 Поняття чат-боту 7

1.2 Програма миттєвих повідомлень Telegram та telegram-боти 9

1.3 Функціонал та особливості telegram-ботів 11

2 ЗАСОБИ РЕАЛІЗАЦІЇ ЧАТ-БОТУ 12

2.1 Мова програмування 12

2.2 Фреймворк для створення telegram-ботів 13

2.3 Бібліотека для роботи з великими масивами даних 15

2.4 Бібліотека для роботи з базами даних 16

3 ПРАКТИЧНА ЧАСТИНА 18

3.1 Опис робочого процесу бота 18

3.2 Впроваджені зміни у процес роботи з записами для звітів 19

3.3 Процес генерації звіту 25

3.4 Процес відправки листів зі звітами 30

3.5 Перевірка даних для звітів 36

3.6 Оновлення даних у таблицях необхідних для генерації звітів 39

3.7 Обробка інформації для використання нової ліцензії 45

ВИСНОВКИ 51

ПЕРЕЛІК ПОСИЛАНЬ 52

СПИСОК СКОРОЧЕНЬ

SQL	**S**tructured **Q**uery **L**anguage	Мова структурованих запитів
API	**A**pplication **P**rogramming **I**nterface	Програмний інтерфейс додатку
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol	Протокол передачі гіпертексту
ЄДРПОУ		Код **є**диного **д**ержавного **р**еєстру **п**ідприємств та **о**рганізацій **У**країни

ВСТУП

У сучасному світі швидкість і якість виконуваних послуг є основною властивістю підприємства. Кожна компанія намагається прискорити процес роботи і при цьому не втрачаючи в якості. Зменшення людського фактору теж є важливою складовою цього питання.

Тому не дивно, що на цей час з кожним днем популярність засобів автоматизації і роботизації різного рівня невпинно зростає. Наприклад, один бот може оперативно давати відповідь тисячам користувачів одночасно, бути простим інтерфейсом для складного алгоритму обчислень чи обробки даних.

Чат-боти є важливою підмножиною ботів і також підвищують свою популярність. Все більше підприємств і компаній використовує чат-боти в своїх програмних комплексах через їх гнучкість, простоту в оновлені та доступність.

Все вище перераховане, безперечно, підтверджує, що створення та підтримка чат-ботів є важливою і актуальною в наш час технічною задачею і відповідає загальносвітовим тенденціям.

МЕТА РОБОТИ

Обрати необхідні програмні засоби розробки для ефективної і довготривалої побудови механізму генерації і перевірки звітів.

Ввести систему додаткових багаторівневих фінансових маркерів, що уточнюють формування собівартості і ціноутворення, а також етап розрахунку із замовниками. Скоротити час оновлення даних перед початком генерації звіту та забезпечити коректну обробку динамічної зміни кількості записів під час формування звіту.

Створена підпрограма для швидкого оновлення таблиць, необхідних при генерації звітів, яка показує, що введення і попередня обробка таблиць коригувань, таблиць ліцензій та контрактів дозволяє оптимізувати обчислювальні ресурси.

Вдосконалити процес генерації звіту, виключивши необхідність ручної корекції і виправлення згенерованих звітів, як це регулярно вимагає попередня наявна система генерації звітів.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття чат-боту

Чат-бот - програмний сценарій, що являє собою імітацію реальної розмови з користувачем. Такі програми дозволяють вести діалог використовуючи текстові або звукові повідомлення у програмах миттєвих повідомлень, різноманітних інтернет сайтах або мобільних додатках.

Чат-боти зазвичай використовуються як сервіси ретранслятори повідомлень, сервіси електронної комерції, автоматизатори бізнес процесів та контакт центри. Використання чат-ботів обмежено їх функціоналом, який зазвичай є дуже вузькоспрямованим, що не дозволяє використовувати чат-боти для великого спектру спілкування з людиною. З точки зору реалізації можна виділити два види класифікації сучасних чат-ботів:

- Бізнес-класифікація чат-ботів (таблиця 1);
- Технічна класифікація чат-ботів (таблиця 2).

Бізнес-класифікація чат-ботів:

- Розмовні чат-боти – не мають конкретної мети, створені для імітації реальної розмови між людьми.
- Чат-боти помічники – мають конкретну заздалегідь визначену мету. Відповіді користувача зазвичай використовуються для отримання додаткової інформації і її подальшого використанні у роботі чат-боту. Часто використовуються у заповненні web-форм чи анкет для допомоги користувачам.
- Q&A (questions and answers) чат-боти - створені щоб відповідати на питання користувача у форматі: одне питання – одна відповідь.

Таблиця 1. Бізнес-класифікація

Бізнес-класифікація		
Розмовні чат-боти	Чат-боти помічники	чат-боти Q&A

Технічна класифікація чат-ботів:

- Засновані на бізнес-правилах – чат-боти діалог з якими побудований за певними правилами, має дерево-подібну структуру розмови. Такі чат-боти зазвичай не підтримують відповідь користувачу у довільній формі, а надають декілька підготованих варіантів відповідей, тому діалог з користувачем йде за попередньо визначеним розробником сценарієм діалогу.
- Основані на штучному інтелекті – чат-боти у роботі яких використовуються технології штучного інтелекту та машинного навчання (Neuro-Linguistic Programming – нейролінгвістичне кодування; Natural-Language Understanding – розуміння природної мови або інтерпретація природної мови; Neural Network – нейронні мережі). Боти такого типу не мають заздалегідь визначеного сценарію розмови, а їх тема розмови та відповіді засновані на тренувальних даних, що були використані на етапі навчання моделі машинного навчання. Таке навчання також є основною проблемою чат-ботів цього типу, тому що потребує великого масиву навчальних даних для того щоб бот міг підтримувати розмову з користувачем у подібній до реальної розмови формі.
- Гібридні – чат-боти які представляють собою комбінацію ботів заснованих на бізнес-правилах та основаних на штучному інтелекті. Діалог з користувачем також ведеться за певним сценарієм, а штучний інтелект допомагає отримувати необхідні для подальшої роботи дані з відповідей користувача.

Таблиця 2. Технічна класифікація

Технічна класифікація		
Основані на бізнес-правилах	Основані на штучному інтелекті	Гібридні

1.2 Програма миттєвих повідомлень Telegram та telegram-боти

Під час розвитку популярності мобільних соціальних мереж у 2014-2016 роках найбільш стрімко розвиваємим, безпечним та популярним став Telegram.

На початку свого існування Telegram створювався як звичайна мобільна платформа для миттєвого листування, прив’язана до номеру телефону. Головними перевагами Telegram стали захищеність і конфіденційність листування користувачів та можливість створення групових чатів, платформо-незалежність – Telegram можна використовувати на будь якому пристрої який передбачає використання такого функціоналу. Простий та зрозумілий інтерфейс також зіграв свою роль, він дозволяє швидко пристосуватися до нового додатку та повноцінно використовувати Telegram вже з перших хвилин роботи з ним.

Велику частину аудиторії Telegram привернули так звані телеграм-канали у яких користувачі могли ділитися інформацією з іншими на кшталт інформаційних агенцій. Різного роду та розміру компанії і ентузіасти почали створювати власні канали для розповсюдження різного роду новин, торгівлі, реклами та інших видів діяльності. Але для ефективного та швидкого використання цього функціоналу підприємцям треба було шукати способи автоматизації робочих процесів.

Таким способом автоматизації стали telegram-боти. Telegram-бот – додаток, розміщений на сервері, що використовує API Telegram для підключення до користувачів Telegram. Такі боти використовують звичайні повідомлення у чаті Telegram та кнопки для взаємодії з користувачем. Також

користувачі мають можливість завантажувати файли різноманітного розширення, наприклад фото, відео, текстові файли з чат-бота чи навпаки відправляти йому файли для подальшої обробки.

Telegram-боти стали настільки популярними, що розробники Telegram створили окреме середовище, що керує ботами – Botfather. Можливості telegram-ботів постійно збільшуються: пошук різного роду інформації, торгівля різноманітними товарами, переклад, інтеграція з іншими сервісами та інші види послуг. Також розробники почали використовувати telegram-боти як простий та знайомий інтерфейс для власних програм.

Telegram використовують мільйони користувачів та компаній кожен день. Функціонал цієї програми миттєвих повідомлень постійно збільшується, а старі функції покращуються.

Таким чином, можна визначити цілий ряд переваг, який утворився за час існування програми миттєвих повідомлень Telegram, а саме:

- Telegram є безкоштовним додатком і не має інтегрованої реклами;
- Додаток Telegram є «open source» проектом з відкритим вихідним кодом;
- Сервери Telegram розташовані по всьому світу, що покращує швидкість та стабільність роботи;
- Telegram використовує хмарні технології, тому з одним обліковим записом можна працювати одночасно на декількох пристроях незалежно від їх типу;
- Через Telegram можливо відправляти файли до двох гігабайтів;
- Усе листування зашифроване, також можна налаштувати знищення повідомлень через потрібний користувачу час;
- Функціонал Telegram постійно розширюється завдяки telegram-ботам і Telegram вже складно назвати простою програмою миттєвих повідомлень.

1.3 Функціонал та особливості telegram-ботів

Telegram-боти можуть виконувати різноманітні задачі, наприклад працювати як ретранслятори повідомлень з інших джерел інформації, отримувати платежі від користувачів через Apple Pay та Google Pay, інтегрувати роботи різних служб (Gmail – gmailbot, Trello – trello_bot та інші), служити інтерфейсом для різноманітних програм для груп користувачів. Вагомою перевагою telegram-ботів також є те, що їх не потрібно встановлювати на свій пристрій, достатньо просто написати боту. Робота з декількома користувачами одночасно теж є сильною стороною telegram-ботів, але якщо бот проводить якісь тривалі обчислення чи звертається до баз даних розробнику потрібно налаштовувати процес паралелізації обчислень самостійно.

Також telegram-боти мають декілька відмінностей від звичайних користувачів:

- Ім’я усіх ботів закінчується на «bot»;
- У ботів немає статусу який відображає статус присутності, замість нього відображається мітка «Бот»;
- При додаванні бота до групи, за замовчанням, він не буде отримувати всі повідомлення. Зазвичай боти реагують на попередньо налаштовані розробником команди, які починаються з символу «/»;
- Боти не можуть розпочинати чати зі звичайними користувачами. Користувач повинен першим писати боту, або додати його до групового чату.

2 ЗАСОБИ РЕАЛІЗАЦІЇ ЧАТ-БОТУ

2.1 Мова програмування

Python є універсальною мовою програмування, яка часто виступає у ролях, пов'язаних із написанням сценаріїв. Python зазвичай визначається як об'єктно орієнтована мова – визначення, яке комбінує підтримку ООП із загальною орієнтацією на сценарні ролі, є мовою високого рівня з інтегрованою динамічною структурою для підтримки та розробки додатків.

Великою перевагою Python є його розширюваність. Відбувається це за допомогою різноманітних бібліотек з додатковим функціоналом, що дозволяє додавати потрібний розробнику функціонал до свого проекту.

Python може використовуватись на різних системах та платформах, що дозволяє використовувати один і той самий код у різних системах, що полегшить взаємодію розробників та підвищить швидкість роботи над проектами.

Створюючи telegram-ботів можна вибирати з великої кількості різних мов програмування, наприклад: C, C++, Java, C#, NodeJS, PHP, Python. Але Python частіше використовується для створення telegram-ботів, код написаний на Python легко сприймається та модернізується, що підвищує швидкість внесення змін у проект, коли функціонал потрібен «вже вчора». Також Python має потужні бібліотеки для роботи з великими масивами різноманітних даних.

Для наочності можна порівняти Python і C#, теж популярну мову програмування для створення telegram-ботів.

Таблиця 3. Порівняльна характеристика Python та C#

Python	**C#**
Python сумісний та переноситься в системи Windows, macOS та Unix/Linux	Не такий гнучкий, як інші мови програмування, оскільки залежить від платформи .NET
Динамічна типізація	Статична типізація

Може компілюватись з .NET, JavaScript, C та Java	Потребує .NET SDK
Мова, що інтерпретується	Мова, що компілюється
Має великий набір попередньо запакованих бібліотек	Підтримка бібліотек заснована на .NET framework

І Python, і C# є об'єктно орієнтованими мовами загального призначення. Python буде найкращим варіантом, якщо проект пов'язаний з дослідженням та обробкою даних, оскільки він має велику стандартну бібліотеку. Вибір C# буде корисним для розробки адаптивних веб-сайтів, веб-сервісів та настільних програм.

Організована структура C# гарантує відсутність невідповідностей у синтаксисі та правилах форматування. З іншого боку, можна писати та тестувати Python код швидше, ніж C#.

2.2 Фреймворк для створення telegram-ботів

Фреймворк – це програмне середовище спеціального призначення, своєрідний каркас, використовуваний для того щоб істотно полегшити процес об'єднання певних компонентів під час створення програм. Це основа, яка дає змогу додавати компоненти залежно від потреб проекту. База, де можна сформувати програму будь-якого призначення досить швидко і без особливих труднощів.

Класифікація фреймворків:

- Фреймворки додатків – «каркас» для додатку, що дозволяє спростити процес створення користувальницького інтерфейсу;
- Фреймворки програмних моделей – «каркас» для програмної системи, що може включати у себе різноманітні бібліотеки, допоміжні програми та

інше програмне забезпечення для полегшення розробки програмного проекту (наприклад Django);

- Фреймворки концептуальні моделі – використовуються у дослідженнях для визначення можливих способів вирішення проблеми чи представлення ідеї, абстрактна структура.

Для створення telegram-ботів існує велика кількість різних фреймворків. Найпотужнішим з них є Aiogram.

Aiogram – асинхронна структура для Telegram Bot API (інтерфейс на основі HTTP, створений для розробників) написана на Python 3.7 з використанням asyncio та aiohttp.

Asyncio – дизайн паралельного програмування який отримав спеціальну підтримку в Python, для визначення співпрограм (coroutines) використовуються ключові слова async/await.

Aiohttp – є HTTP-клієнтом/сервером для asyncio. В цілому ця бібліотека дозволяє писати асинхронні клієнти та сервери.

Переваги Aiogram над іншими структурами:

- Aiogram є асинхронним, що робить його швидшим в частині задач;
- Підтримка Python 3.7+ і вище;
- Постійні оновлення та підтримка, відкритий чат, де можна поспілкуватись з розробниками та постійними користувачами;
- Якісно реалізований «polling», завдяки чому бот постійно не вимикається і непотрібно відслідковувати помилку з’єднання.

Чат-боти повинні отримувати сповіщення від серверу моментально. Вони не можуть перевіряти оновлення кожну секунду, це неефективно. Більшість відповідей від сервера будуть неінформативні, на кшталт «У вас поки що немає нових повідомлень». Такий підхід, коли раз у n секунд опитується сторонній сервіс, називається «polling».

2.3 Бібліотека для роботи з великими масивами даних

Pandas – це бібліотека Python для обробки та аналізу структурованих даних. Pandas надає значні структури даних і функції, призначені для швидкої роботи зі структурованими даними. Сама назва pandas походить від «panel data», економетричного терміну для багатовимірних структурованих наборів даних, і самого аналізу даних Python.

Pandas є одним з важливих компонентів, що дозволяє Python стати потужним і продуктивним середовищем аналізу даних. Основним об’єктом у pandas є DataFrame, двовимірна таблична, орієнтована на стовпці структура даних з мітками як рядків, так і стовпців:

	edrpou	report_on	has_lab	month_m	year_m
0	00185028	2020-11-01	1	10	2020
1	01983163	2020-11-01	1	10	2020
2	01985423	2020-11-01	1	10	2020
3	01990654	2020-11-01	1	10	2020
4	01992831	2020-11-01	1	10	2020
...	...	...	...	...	...
546	42751893	2021-12-01	1	11	2021
547	01280527	2021-12-01	1	11	2021
548	01993730	2021-12-01	1	11	2021
549	01993865	2021-12-01	1	11	2021
550	43221703	2021-12-01	1	11	2021

551 rows × 5 columns

Рисунок 1. Структура даних DataFrame

Pandas поєднує в собі високопродуктивні можливості NumPy для обчислення масивів з гнучкими можливостями маніпулювання даними електронних таблиць і реляційних баз даних (наприклад, SQL). Він надає

складну функціональність індексування щоб спростити зміну форми, нарізки та виконувати агрегації і вибирати підмножини даних.

Основними перевагами Pandas є:

- Швидкий та ефективний об'єкт DataFrame для маніпулювання даними з інтегрованим індексуванням;
- Гнучка зміна форми та поворот наборів даних;
- Інтелектуальне вирівнювання даних та інтегрована обробка відсутніх даних;
- Високопродуктивне merge та join наборів даних.

2.4 Бібліотека для роботи з базами даних

PostgreSQL — це об'єктно-реляційна система управління базами даних, заснована на POSTGRES, версія 4.2 — програма, розроблена на факультеті комп'ютерних наук Каліфорнійського університету в Берклі. Вона підтримує велику частину стандарту SQL і пропонує безліч сучасних функцій:

- складні запити;
- зовнішні ключі;
- багатоверсійність;
- змінювані представлення;
- тригери;
- транзакційна цілісність.

Крім того, користувачі можуть постійно розширювати можливості PostgreSQL, наприклад створювати свої:

- типи даних;
- агрегатні функції;
- методи індексування;
- функції;

- оператори;
- процедурні мови.

Для взаємодії з базами даних була використана бібліотека SQLAlchemy.

SQLAlchemy — це набір інструментів Python SQL та Object Relational Mapper, який надає розробникам додатків повну потужність та гнучкість SQL.

SQLAlchemy надає повний набір добре відомих шаблонів збереження на робочому рівні, розроблених для ефективного та високопродуктивного доступу до бази даних, адаптованих до простої мови домену Pythonic.

SQLAlchemy розглядає базу даних як механізм реляційної алгебри, а не просто набір таблиць. Рядки можна вибирати не тільки з таблиць, але й з об’єднань та інших операторів вибору; будь-який з цих підрозділів може бути складений у більшу структуру.

SQLAlchemy найбільш відомий своїм «object-relational mapper» (ORM), додатковим компонентом, який надає шаблон відображення даних, де класи можуть бути зіставлені з базою даних різними способами, що дозволяє об’єктній моделі та схемі бази даних розвиватися по чітко поставленому маршруту із самого початку.

3 ПРАКТИЧНА ЧАСТИНА

3.1 Опис робочого процесу бота

Розроблюваний бот представляє собою реалізацію бізнес процесу звітності клієнтів по виконаній роботі, а саме частину від агрегації записів наданих виконавцями послуг до передачі згенерованих ботом та підписаних усіма суб’єктами цього процесу звітів у бухгалтерію для оплати.

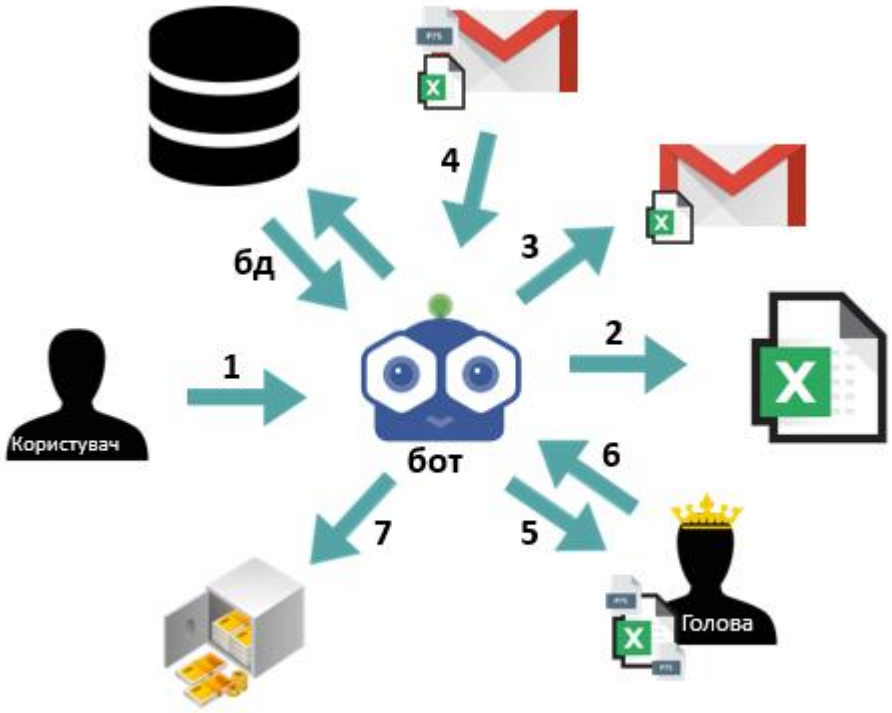

Рисунок 2. Схема роботи бота

Сам процес генерації звіту складається з семи етапів:

1. Користувач перевіряє дані для генерації звітів і за потреби оновлює потрібні джерела даних використовуючи наданий функціонал бота;
2. Користувач запускає процес генерації ботом файлу звіту розширення .xlsx;
3. Користувач відправляє згенеровані файли-звіти клієнтам на перевірку;
4. Якщо клієнт згоден з представленою у звіті інформацією, він підписую звіт своїм електронно цифровим підписом та відправляє підписаний файл звіту назад;

5. Отримані підписані звіти перевіряються на збіг сум до відправки файлу та після і збіг ім’я підписанта у звіті та у електронно цифровому підписі. Після перевірки звіти відправляються на голові структури.
6. Голова підписує звіти своїм електронно цифровим підписом та повертає їх назад.
7. З підписаних файлів звітів формуються реєстри для їх оплати, які містять необхідну для оплати інформацію: ЄДПОУ, номер звіту, iban, суму оплати і ще декілька полів з додатковою інформацією. Створені реєстри відправляються на оплату до бухгалтерії.

Весь необхідний функціонал складається з восьми кнопок представлених на рисунку 3.

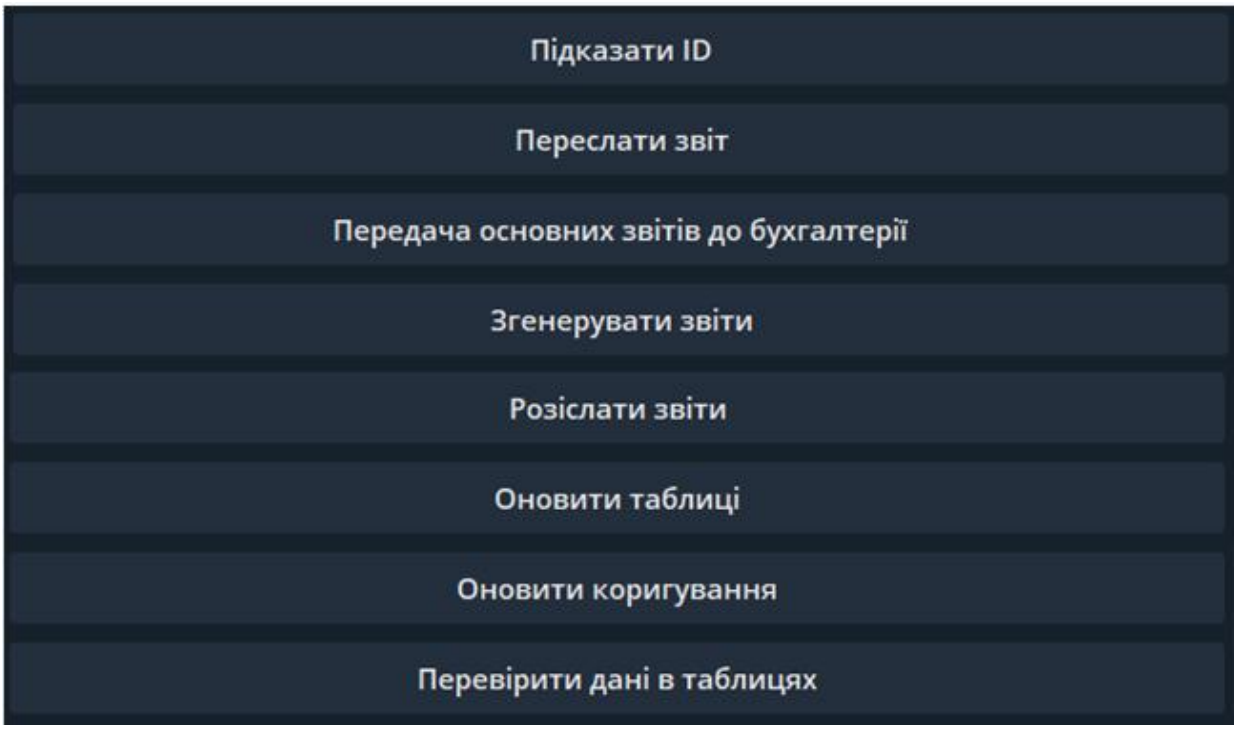

Рисунок 3. Функціонал бота

3.2 Впроваджені зміни у процес роботи з записами для звітів

У створюваних звітах висвітлюється інформація про те скільки послуг було надано клієнтом за минулий місяць, їх сумарну ціну та іншу додаткову

інформацію, наприклад залишок суми договору, коригування чи скільки вже було сплачено коштів.

Таблиця 3. Приклад представлення даних по наданих послугах

Назва медичної послуги	Код медичної послуги	Фактична кількість пролікованих випадків (Cases), шт.	Ваговий коефіцієнт діагностично-споріднених груп
Операції на сітківці	C03	2	0,513
Операції на носі	D10	2	0,846
Тонзилектомія та аденоїдектомія	D11	1	0,625
Інші операції на вусі, носі, роті і горлі	D12	1	1,133
Ампутація верхніх кінцівок і пальців ніг при порушеннях кровообігу	F13	3	1,956
Лігування та екстирпація вен	F20	3	0,800
Інші загальні втручання на органах кровообігу	F21	1	2,906
Апендектомія	G07	1	1,419
Операції з вправлення грижі	G10	11	1,535
Операції на задньому проході та стомі	G11	3	1,778
Гастроскопія	G47	1	0,118
Колоноскопія	G48	1	0,161
Лапароскопічна холецистектомія	H08	5	3,318
Інші операції на кульшовому суглобі і стегні	I08	3	3,863
Процедури на м'яких тканинах	I27	1	1,250
Процедури при пілонідальній кісті та в періанальній ділянці	J09	1	0,552
Інші операції на шкірі, підшкірній клітковині і молочних залозах	J11	3	0,829
Операції на чоловічому статевому органі	M03	1	1,712
Діагностичне вишкрібання та діагностична гістероскопія	N10	2	0,304
Аборт з загальними втручаннями	O05	1	0,375
Спленектомія	Q01	1	3,255
Всього		48	X
Базова ставка за пролікований випадок (BR), грн		X	7 506,000
Фактичний індекс структури випадків (CMI)		X	1,625
Коефіцієнт частки застосування ставки на пролікований випадок (PPD)		X	0,200
Коефіцієнт збалансованості бюджету		X	1,000
Гірський коефіцієнт		X	1,200
Коефіцієнт готовності надавати медичну допомогу дітям або дорослим		X	X
Всього за пролікувані випадки (drgi), грн		X	140 512,320

Для генерації таких таблиць для звітів створювалася таблиця з агрегованими даними для наповнення звіту інформацією про кількість наданих послуг. Дані отримувались напряму з таблиці з усіма записами які пройшли перевірку та були промарковані для визначення необхідності їх подальшої оплати. Структура роботи з записами виглядала так:

Повна таблиця записів ⇒ Таблиця сум для звітів ⇒ Звіт

Така структура мала три важливих недоліки, а саме:

- Неможливість відслідковувати які саме записи були вже використані та оплачені у звіті;

- Постійна зміна кількості та вмісту записів, що призводило до того, що сума виплати у звіті з оновленням таблиці сум могла змінюватись при кожному оновленні цієї таблиці;
- У зв’язку з постійним збільшенням розміру повної таблиці з записами, час оновлення даних у таблиці сум для звітів суттєво збільшувався, що призводило до затримки інших робочих процесів.

Описані вище недоліки призводили до того, що доводилося постійно звіряти суми у вже згенерованих та відправлених звітах, їх переробки та ускладненому контролю за тим які надані клієнтами послуги вже були оплачені, а які – ні.

Для усунення цих недоліків було зміно підхід до роботи з записами. Нова структура виглядає так:

Повна таблиця записів ⇒ Помісячний зріз записів ⇒ Таблиця сум для звітів ⇒ Звіт

Така структура дозволила:

- Зафіксувати з якими саме записами ведеться робота;
- Перенести обрахунок ціни записів з таблиці сум для звітів до нової таблиці;
- Перенести виділення надлишкових записів з повної таблиці записів до нової таблиці;
- При генерації маркувати які саме записи були відображені у звіті і коли;
- Додати функціонал для підрахунку коригувань сум звітів у наступних звітних періодах.

Створена таблиця складається з тринадцяти стовпців, а саме:

- edrpou – ЄДРПОУ клієнта;
- edrpou_reorg – ЄДРПОУ клієнта на який було переоформлено договір;
- packet – номер пакету по якому надаються послуги;
- service – номер послуги;

- price – фінальна ціна послуги з усіма використовуваними коефіцієнтами;
- is_pay – поле типу bool. Використовується для виділення записів які неможливо оплатити через перевищення сплаченої суми за укладеним договором;
- report – номер звіту у якому був згенерований цей запис;
- gen_time – час генерації звіту;
- date – дата надання послуги;
- inserted_at – дата внесення послуги до системи;
- r_month – місяць у звіті якого потрібно виплати цей запис;
- r_year – рік у звітах якого потрібно виплати цей запис;
- r_comment – поле для коментарів.

Для наповнення таблиці використовується SQL запит з використанням оператору INSERT INTO, який складається с декількох підзапитів.

```
select distinct on (client_id, packet) edrpou, event_id, packet, service,
"date", inserted_at, r_month, r_year from events_2022
where r_month = extract(month from now())::numeric - 1
and r_year = extract(year from now())::numeric
and packet in ('1', '5', '26', '38', '50')
and exclusion_reason isnull
order by client_id, packet, "date" DESC, inserted_at
```

Наприклад, на рисунку 4 представлений один з підзапитів для отримання потрібних записів, у якому для пакетів 1, 5, 26, 38, 50 використовуються унікальні ідентифікатори отримувачів послуг для визначення оплачуваних записів, тому що записи у цих пакетах оплачуються по унікальним отримувачам послуг, а не кількості наданих послуг.

Як видно з запиту, вісім стовпців edrpou, event_id, packet, service, date, inserted_at, r_month, r_year отримуються з вже підготованих даних. Інші поля заповнюються при обробці з цих записів.

Поле «price» також заповнюється з використанням SQL запиту, який звертається до таблиці-довідника з цінами кожної послуги та допоміжних

таблиць з додатковими коефіцієнтами, запит для отримання яких представлений нижче.

```
select epm.edrpou, epm.packet,
coalesce(max(s.val) filter (where s.packet::text = epm.packet and s.packet in (1,2,21,37) and s.coef_id = '19'), 1) as m_coef,
coalesce(max(s.val) filter (where s.code_medical::text = epm.packet and sm.code_medical in (1,2,21) and s.coef_id = '10'), 1) as rd_coef,
coalesce(max(sm2.val) filter (where sm2.packet = epm.packet and sm2.packet in ('12','13') and sm2.coef_id in ('12', '13')), 1) as in_coef,
coalesce(max(sm2.val) filter (where sm2.packet = epm.packet and sm2.packet in ('23') and sm2.coef_id = '14'), 1) as eit_coef
from events_per_month ep
left join (select distinct on (packet, m_month, contract_number) *
          from contract_month_koef where m_year = extract(year from now())::numeric
          and m_month = extract(month from now())::numeric - 1 and coef_id in ('19', '10')
          order by packet, m_month, contract_number, t_time desc, doc desc
         ) s
on ep.edrpou = s.edrpou and ep.r_year = s.m_year and ep.r_month = s.m_month
and ep.packet = s.packet::text and s.packet in (1,2,21,37)
left join (select distinct on (packet, contract_number) *
           from (
               select k.edrpou, k.tender, s.contract_number, s.packet::text as packet, k.coef_id, k.coef_val, k.doc, s.m_year
               from koef k join (select * from contract_month where m_year = extract(year from now())::numeric) s
               on k.edrpou = s.edrpou and k.tender::text = s.tender and k.doc = s.doc
           ) j where coef_id in ('12', '13', '14')
           order by packet, contract_number, doc desc
         ) s2
on ep.edrpou = s2.edrpou and ep.r_year = s2.m_year and ep.packet = s2.packet
group by ep.edrpou, ep.packet
```

Також присутні пакети які потребують унікального підходу до обрахунку ціни їх послуги. Таким, наприклад, є дев’ятий пакет ціна послуги якого залежить від кількості послуг наданих клієнтом отримувачу послуг та кількості команд які ці послуги надають.

```
update events_per_month ep2
set price = round(case when ep.ev_count >= 50 then price * ep.team_count
when ep.ev_count BETWEEN 35 AND 49 then price * 0.75 * ep.team_count
when ep.ev_count BETWEEN 20 AND 34 then price * 0.5 * ep.team_count
when ep.ev_count BETWEEN 1 AND 19 then price * 0.25 * ep.team_count
else 0 end, 0) / ev_count
from (
    select ep.edrpou, ep.packet, count(event_id) as ev_count, max(s.team_count) as team_count
    from events_per_month ep
    join (
        select distinct on (packet, m_month, contract_number) *
        from contract_month
        where m_year = extract(year from now())::numeric and m_month = 0 and packet = 9
        order by packet, m_month, contract_number, t_time desc, doc desc
    ) s
    on ep.edrpou = s.edrpou and ep.r_year = s.m_year and 0 = s.m_month
    where packet = '9' group by ep.edrpou, ep.packet
) epm
where ep2.edrpou = ep.edrpou and ep2.packet = ep.packet and ep2.packet = '9';
```

Тепер треба визначити чи є можливість оплатити всі надані клієнтом послуги, що відображається у стовпці «is_pay». Іноді виникає така ситуація при

який клієнт перевищив свої попередні показники по кількості отримувачів послуг та кількості самих послуг і йому не вистачає виділених по цьому договору коштів.

```
def get_overlimit_events(events, limits, payments):

    overs = limits.subtract(payments, fill_value=0).astype(float).round(2)
    overs.index = overs.index.map(lambda x: tuple(x.split('_')))
    overs.index.names = ['edrpou', 'packet']
    overs.name = 'budget'

    cumulation = (events[events['packet'].notnull()]
                  .set_index(['edrpou', 'packet'])
                  .join(overs)
                  .reset_index())
```

Для такої перевірки використовується загальна сума договору, сума усіх попередніх виплат та нові записи. Спочатку виконуємо порівняння порядкового номеру запису з залишковою кількістю записів за договором.

```
cumulation['over_limit'] = cumulation['c_sum'] > cumulation['budget']
limit_mapper = cumulation.set_index('e_id')['over_limit']
f_overs = (events_cumulation
           .query('~over_limit')
           .eval('left = budget - c_sum')
           .groupby(['edrpou', 'packet'])['left']
           .min())

temp = overs
temp.update(f_overs)
f_overs = temp
exc = cumulation.query('over_limit')

for row in exc.itertuples():
    if row.payment <= f_overs.loc[row.edrpou, row.packet]:
        exc.loc[row.Index, 'over_limit'] = False
        f_overs.loc[row.edrpou, row.packet] -= row.payment

mapper = exc.set_index('e_id')['over_limit']
limit_mapper.update(mapper)

return limit_mapper
```

Якщо порядок запису перевищує залишок і пакет належить до переліку пріоритетних, то цей запис позначається у полі «is_pay» як такий, що перевищує суму договору.

Після усіх виконаних дій записи готові до використання у генерації звітів. Так як записи вже мають ціну та промарковані на можливість оплати, то їх можна використовувати у «таблиці сум звітів». Такий підхід до роботи з записами дозволив скоротити оновлення «таблиці сум звітів» з 15-20 хвилин до 2-3 хвилин.

Також присутній функціонал збереження інформації про генерацію звіту, а саме номеру звіту та часу генерації звіту у стовпчиках «report» і «gen_time» відповідно.

```
gen_info = self.report_lines[['edrpou', 'packet', 'report']]
gen_info = gen_info.assign(gen_time=pd.Timestamp.now())
con = db.bd()
gen_info.to_sql('temp_update_events_gen_info', con=con, if_exists='replace',
                index=False)
con.execute(
    "update events_per_month e set e.report = inf.report, e.gen_time = inf.gen_time "
    "from temp_update_events_gen_info inf where e.edrpou = inf.edrpou and "
    "e.packet = inf.packet and not e.is_over"
)
con.close()
```

У pandas неможливо використовувати повний об’єкт DataFrame в оновленні таблиць, тому потрібно спочатку зберегти цей об’єкт у тимчасовій таблиці, а вже потім використати SQL запит з оператором UPDATE для оновлення даних у вже заповненій таблиці.

3.3 Процес генерації звіту

Процес генерації звіту ботом є основним його призначенням. Спочатку користувач проходить сценарій у якому йому потрібно ввести, або вибрати з наданих варіантів відповідей (рисунок 4), додаткові параметри генерації які

будуть виступати параметрами при створенні об’єкту класу PaymentGenerator. Далі цей об’єкт буде використаний у генерації файлів звітів.

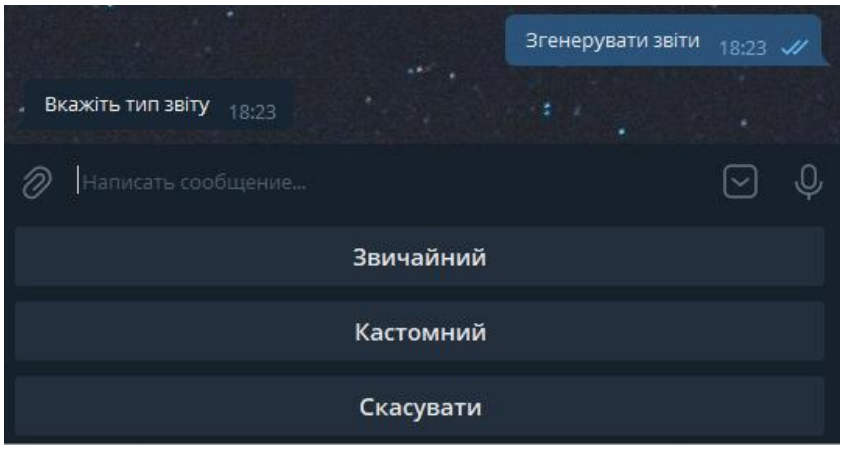

Рисунок 4. Приклад вибору у сценарії генерації звіту

Розроблено сценарій налаштування генерації звіту який складається з декількох етапів у яких користувачу потрібно обрати потрібні йому варіанти налаштування процесу генерації. Створений сценарій можна умовно робити на дві частини. У першій користувач налаштовує процес генерації (рисунок 5).

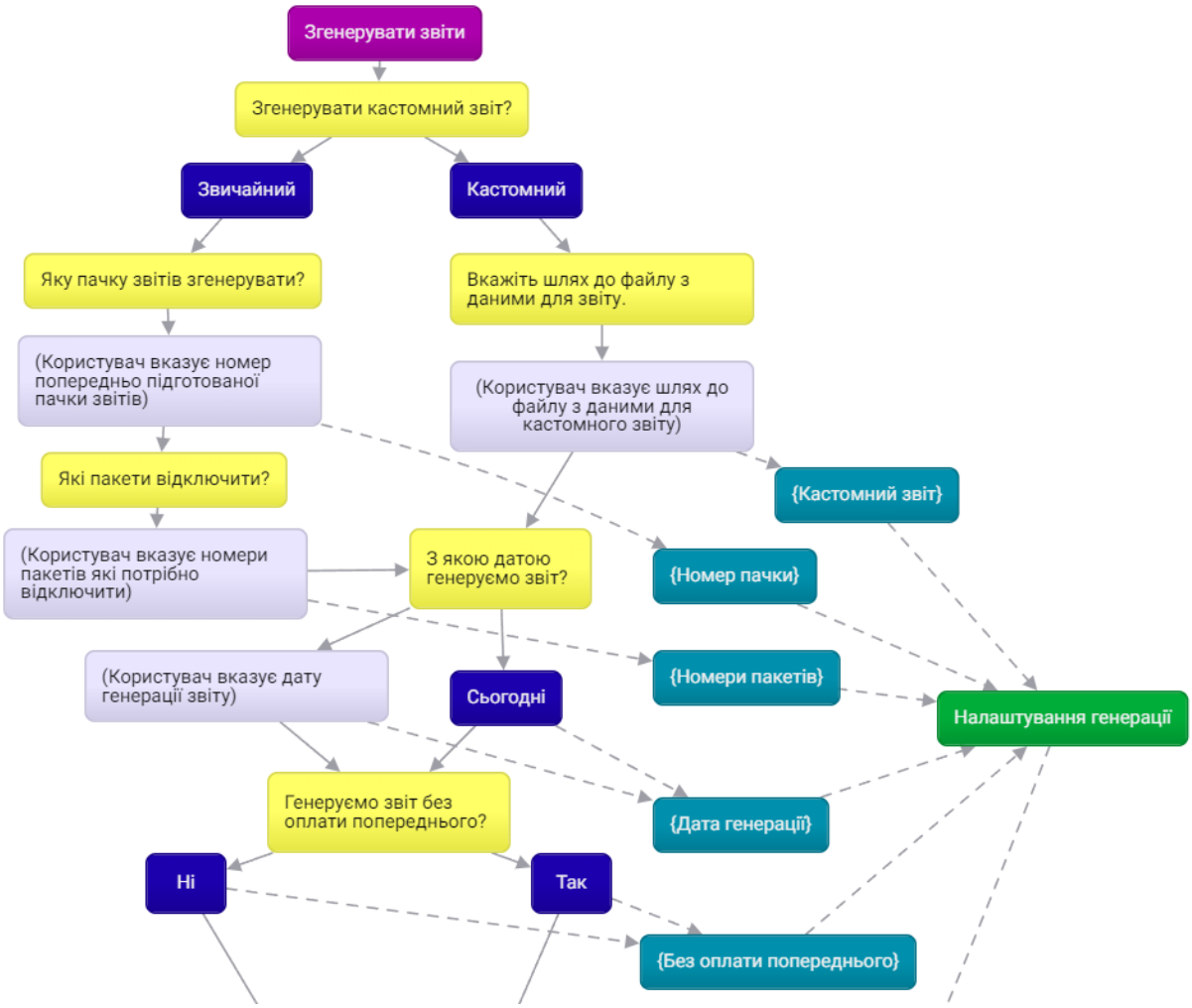

Рисунок 5. Схема роботи сценарію для генерації звіту, частина 1

Для створення сценарію опитування користувача використовувались обробники команд з фреймворку Aiogram, які використовують ключові слова async/await для реалізації взаємодії з користувачем. Приклад використання цього функціоналу можна побачити на фрагменті коду нижче.

```
@dp.message_handler(state=PaymentGenerationPackStates.start)
async def _(message: types.Message, state: FSMContext):
    await state.update_data(custom=message.text)
    if message.text == 'Звичайний':
        await PaymentGenerationPackStates.pack.set()
        await message.answer('Починаємо генерацію звичайних звітів ' + u'\U0001F649')
        await message.answer('Яку пачку згенерувати?', reply_markup=contracts_verification_handlers.cancel_markup())
    elif message.text == 'Кастомний':
        await PaymentGenerationPackStates.custom.set()
        await message.answer('Починаємо генерацію кастомних звітів ' + u'\U0001F648')
        await message.answer(
            'Вкажіть назву файлу у {0} з кастомними звітами\n\u2757Пропустіть крок якщо ви вже його '
            'завантажували\u2757'.format(file_dir), reply_markup=yes_no(['Пропустити крок', 'Скасувати']))
    else:
        if message.text != 'Скасувати':
            await message.answer('Щось пішно не так, спробуйте ще раз ' + u'\U0001F621',
                                 reply_markup=contracts_verification_handlers.cancel_markup())
        else:
            await state.finish()
```

Після проходження всіх попередніх кроків бот створює об’єкт PaymentGenerator:

```
except BaseException:
    await message.answer(f'Вказано "{date}", генерую сьогоднішньою датою.')
    pg = PaymentGenerator(pack=pack, off_packet=off_packet, isCustom=isCustom, prod=True,
                          without_payment=without_payment)
else:
    def custom_func(report_values: ReportValues):
        report_values.df.report_date = date

    pg = PaymentGenerator(pack=pack, off_packet=off_packet, prod=True, isCustom=isCustom,
                          custom_values_function=custom_func, without_payment=without_payment)
```

Далі бот проводить декілька перевірок потрібної для генерації звіту інформації. Ці перевірки представляють собою порівняння результатів декількох SQL запитів чи перевірки інформації отриманої з зовнішніх джерел, наприклад перевірка статусу закладу у єдиному державному реєстрі, інформація з якого постійно оновлюється і зберігається у форматі таблиці в базі даних:

```
def check_status(self):
    edrpou = self.sm_lines.edrpou
    data_query = f""" select edrpou, state, state_text from edr_data
        where state not in ('1','6') and edrpou in %(numbers)s;"""
    data = pd.read_sql_query(sql=data_query, con=con, params={'numbers': tuple(edrpou)})

    dict = defaultdict(list)
    for i, row in data.iterrows():
        dict[row['edrpou']].append(row['state_text'])
    return dict
```

Результати перевірок виводяться користувачу після чого починається сам процес генерації звіту.

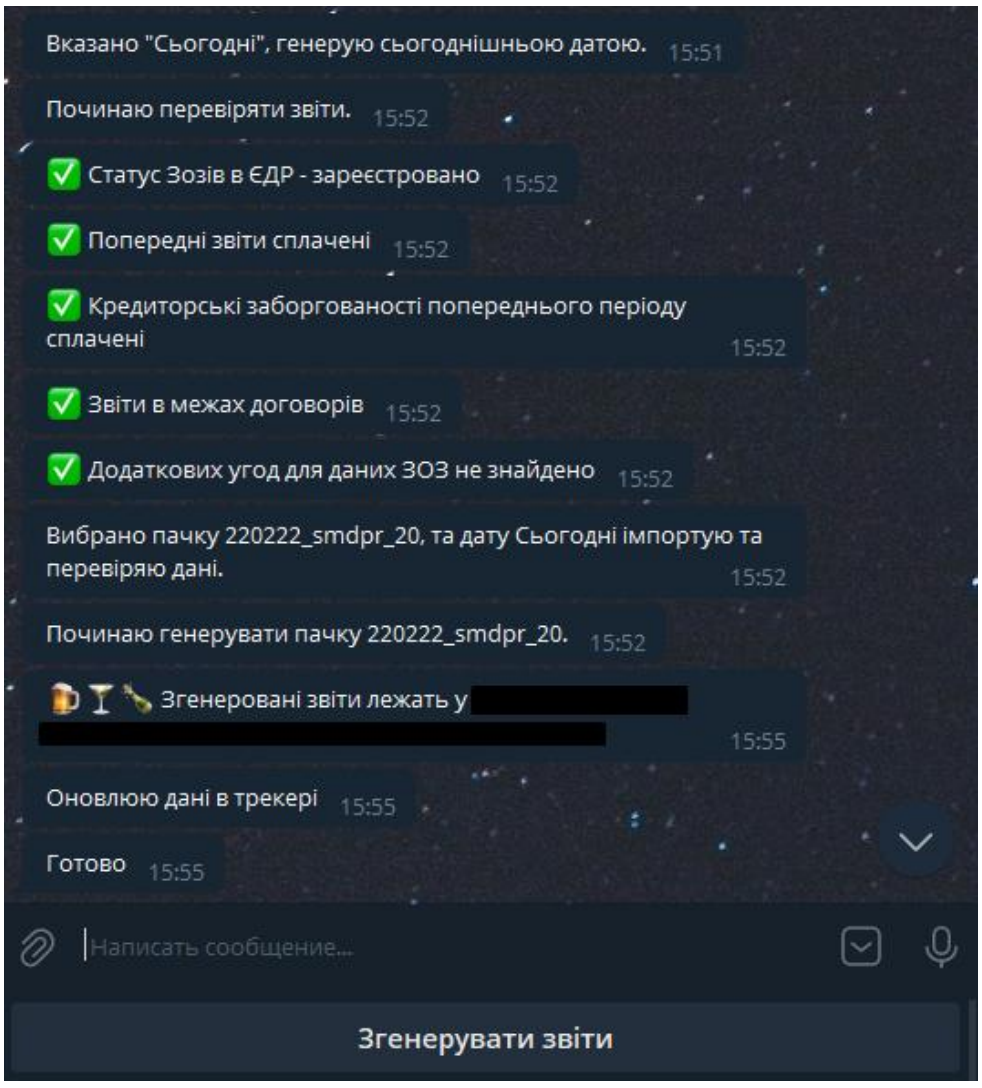

Рисунок 6. Результат роботи сценарію

Процес генерації являє собою отримання і агрегацію даних отриманих за допомогою SQL запитів у класах report_values та report_lines і подальшого їх

використання у класі xlsx_generator по назві якого можна зрозуміти, що цей клас створює файл звіту у форматі .xlsx. Схематичне представлення другої частини сценарію генерації звіту можна побачити на рисунку 7.

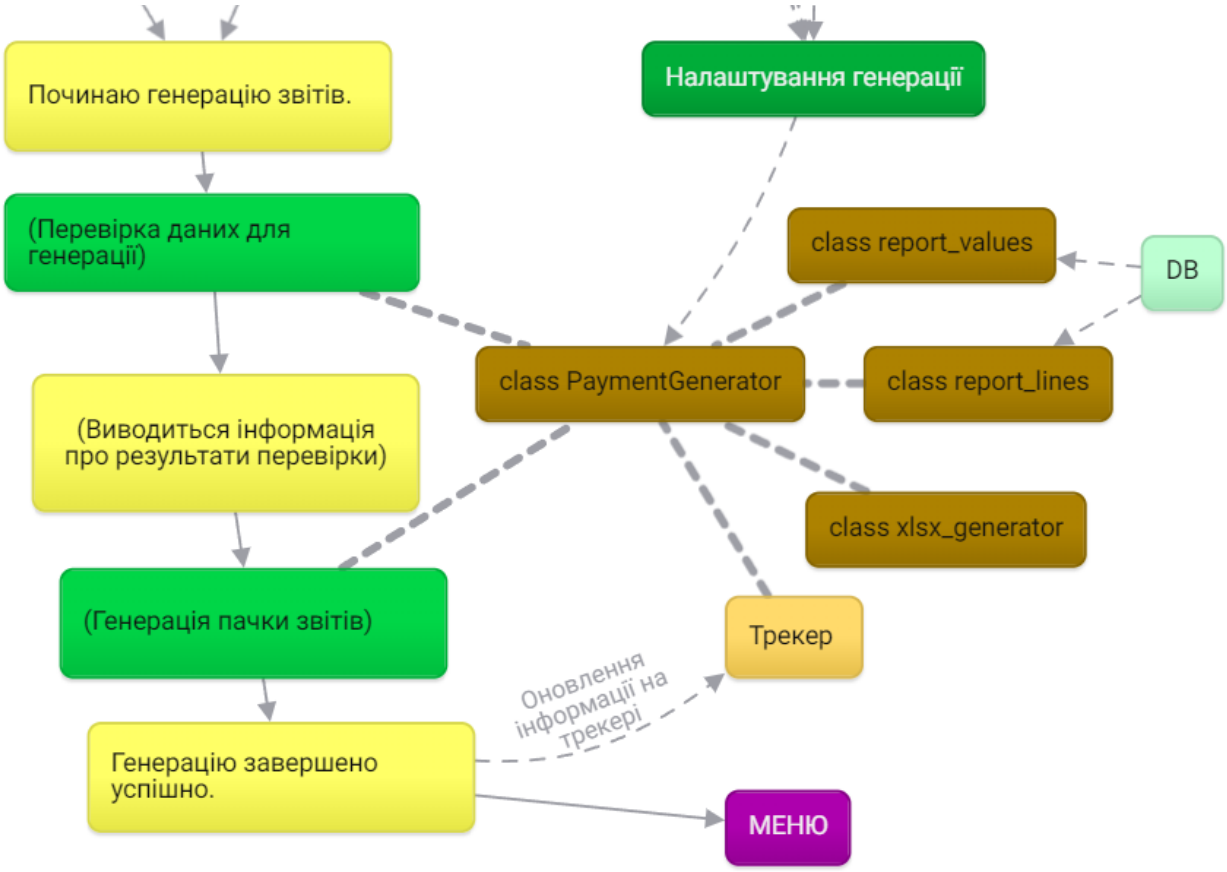

Рисунок 7. Схема роботи сценарію для генерації звіту, частина 2

3.4 Процес відправки листів зі звітами

Після генерації звітів їх потрібно відправити клієнтам для цього у боті створений функціонал відправки звітів. Спочатку користувач вводить назву пачки звітів, потім бот перевіряє наявність пачки з вказаною користувачем назвою.

```
@dp.message_handler(state=PackStates.pack)
async def _(message: types.Message, state: FSMContext):
    pack = message.text
    if packCheck:
        await message.answer(f'Пачка {pack} відсутня, перевірте назву пачки.')
    else:
        await message.answer(f'Вибрано пачку {pack}, починаю росилати.')

        print('payment sending is started')
        lists = MainSender(pack=pack)
        lists.send_all()
        print('payment sending is ended')

        await message.answer('Звіти поставлено в чергу')
    await state.finish()
```

Якщо пачка з такою назвою існує, то створюється об’єкт класу MainSender. Спочатку бот отримує інформацію з зовнішнього API, а саме Google Sheets, у документі якого визначені пачки звітів для роботи з ними

```
def get_pack(self):
    df = import_workbook()
    df = df[df['Generation pack'] == self.pack]
    self.pack_list = df
```

Після цього бот отримує список ЄДРПОУ і відповідним їм email-лам для подальшого визначення адресатів для розсилки.

```
def get_emails(self):
    df = pd.read_sql_query('select edrpou, email from contracts_add_inf', con)
    df = df.groupby('edrpou').apply(lambda x: set(x.email)).rename('emails').reset_index()
    df['emails'] = df['emails'].apply(lambda x: ', '.join([y for y in x if y]))
    self.emails = df.set_index('edrpou')['emails']
```

Далі бот використовує отриману у попередніх двох етапах інформацію. За допомогою класу reportConverter з номеру звіту можна визначити рік, місяць, ЄДРПОУ та тип звіту, що буде використовуватись у створенні листа для клієнта.

```
def prepare_messages(self):
    subject = pd.read_csv('N:/sender/subject.txt', delimiter='$').columns[0]
    body = Path('N:/sender/body.txt').read_text(encoding='utf-8')

    def get_message(row):
        rn = reportConverter(number=row.report)
        path = Path('N:') / f'reports/{row.report}.xlsx'
        assert path.exists(), path

        return {
            'sender': '***',
            'receiver': self.emails[rn.edrpou],
            'title': subject.replace('month', rn.month_name.lower()).replace('year', str(rn.year)) + ' ' + rn.edrpou,
            'message': body.replace('month', rn.month_name.lower()).replace('year', str(rn.year)),
            'links': [path],
            'status': 'START',
            'campaign': rn.code,
        }

    messages = [get_message(row)for index, row in self.pack_list.iterrows()]
    self.messages = messages
```

Після наповнення листа інформацією та визначення email-ла адресату викликається метод send_all класу MainSender для встановлення листів у чергу на відправку. Відправку виконує інший сценарій на сервері, на цьому етапі створюється запис у базі даних зі статусом «START» який потім оброблюється і конвертується у email та відправляється клієнту.

```
def send_all(self):
    con = db_engines.con()
    Session = sessionmaker(bind=con)
    session = Session(autoflush=True)
    for message in self.messages:
        session.add(SenderMessage(**message))

    session.commit()
    session.close()
```

Сценарій роботи описаного у цьому пункті функціоналу можна побачити на рисунку 8, а результат роботи на рисунку 9.

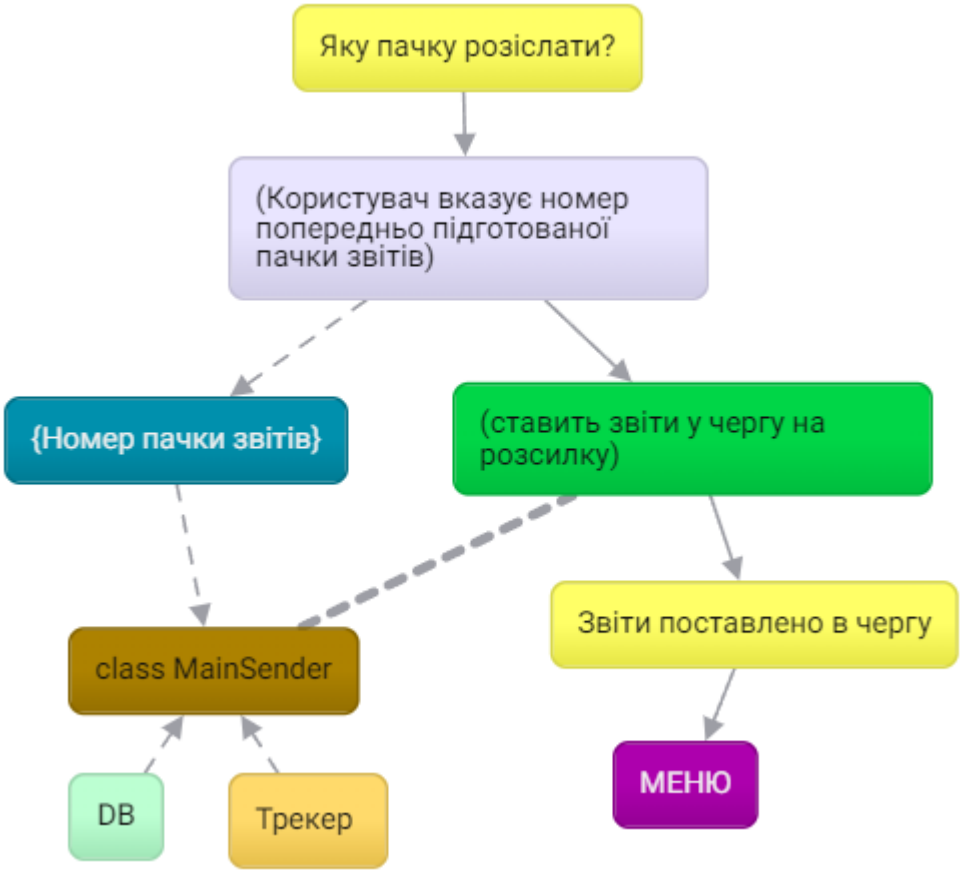

Рисунок 8. Сценарій функціоналу відправки згенерованих звітів клієнтам

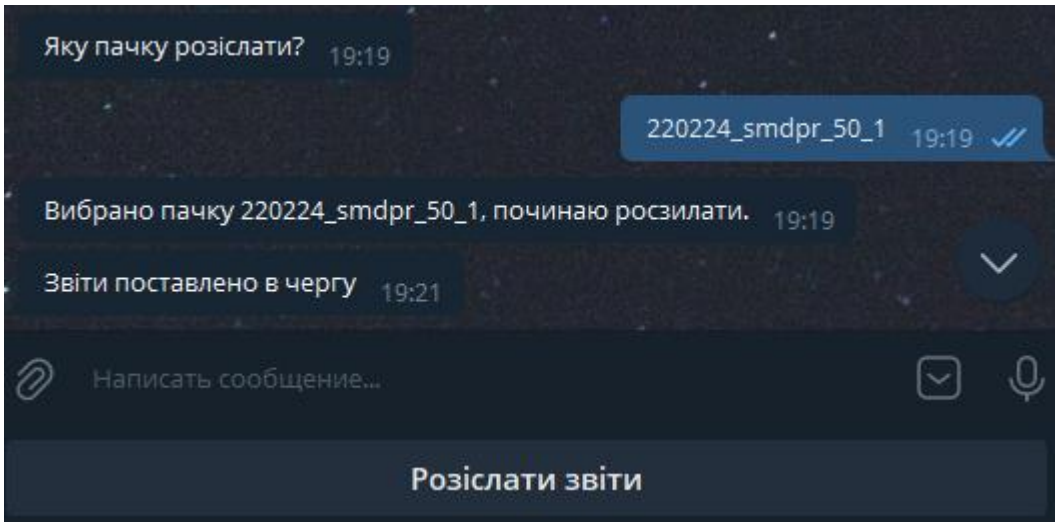

Рисунок 9. Приклад роботи сценарію відправки звітів клієнтам

Іноді виникає необхідність повторно надіслати звіт клієнту, наприклад клієнт випадково видалив лист зі звітом чи втратив доступ до своєї поштової скриньки. Для цього був реалізований функціонал для повторної відправки звітів.

Процес дуже схожий на процес описаний для звичайного надсилання листів, але має свої додаткові елементи. Спочатку користувач вводить потрібний йому номер звіту.

```python
@classmethod
def find_report(cls, report: str):
    rn = reportConverter(number=report)
    path = Path('N:') / f'reports/{rn.number}.xlsx'
    return path_str if path.exists() else None
```

Якщо такий звіт існує користувачу надається інформація про збережені в базі даних назви електронних поштових скриньок.

```python
emails = MainSender.find_emails(edrpou)
await state.update_data(emails=emails)
await message.answer(f'Знайдені емейли {emails}')

await message.answer(
    'Якщо email підходить, то натисніть "Підходить", якщо ні, введіть необхідний.',
    reply_markup=contracts_verification_handlers.cancel_markup([accept_default_emails_text]))

await PackStates.emails.set()
```

Далі користувачу надається можливість обрати надані йому стандартні варіанти чи ввести необхідну йому поштову скриньку. Подальший процес використовує функціонал описаний раніше на сторінках 28-30 дипломної роботи.

Сценарій роботи описаного у цьому пункті функціоналу можна побачити на рисунку 10, а результат роботи на рисунку 11.

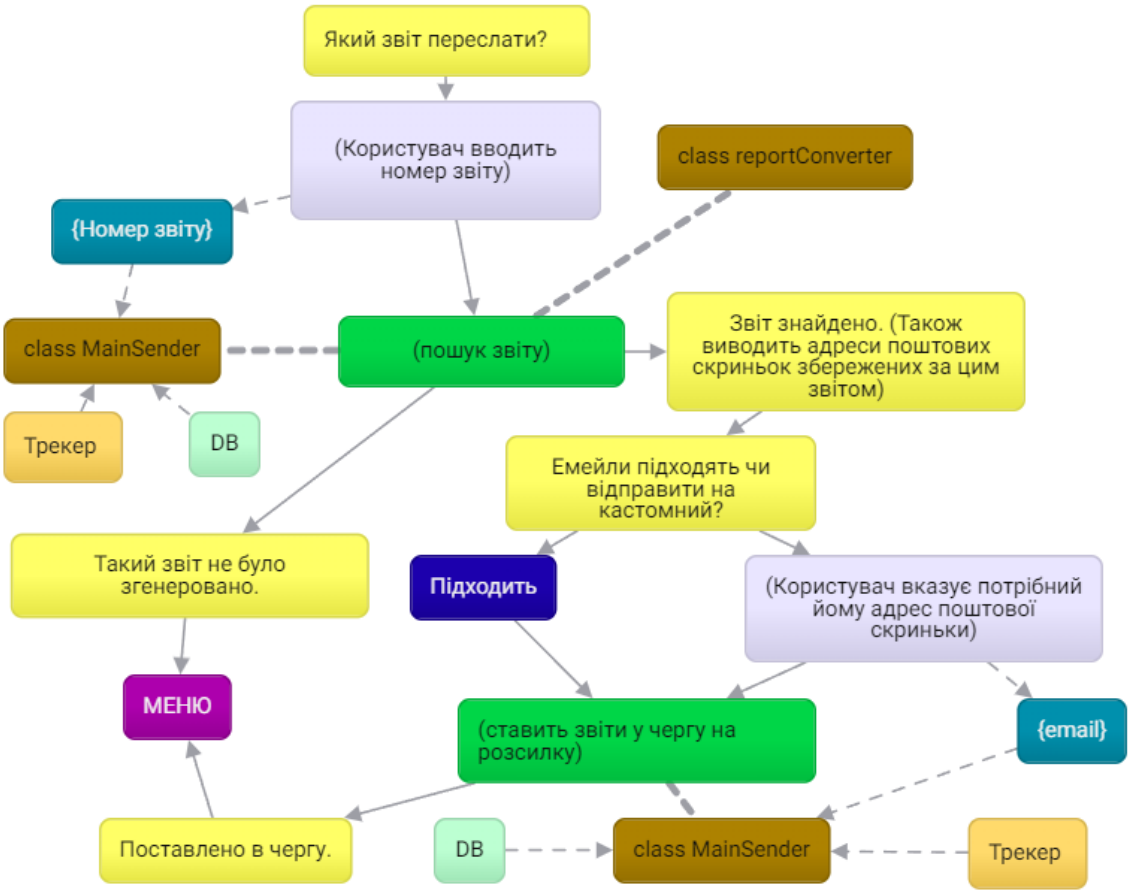

Рисунок 10. Сценарій повторної відправки згенерованих звітів клієнтам

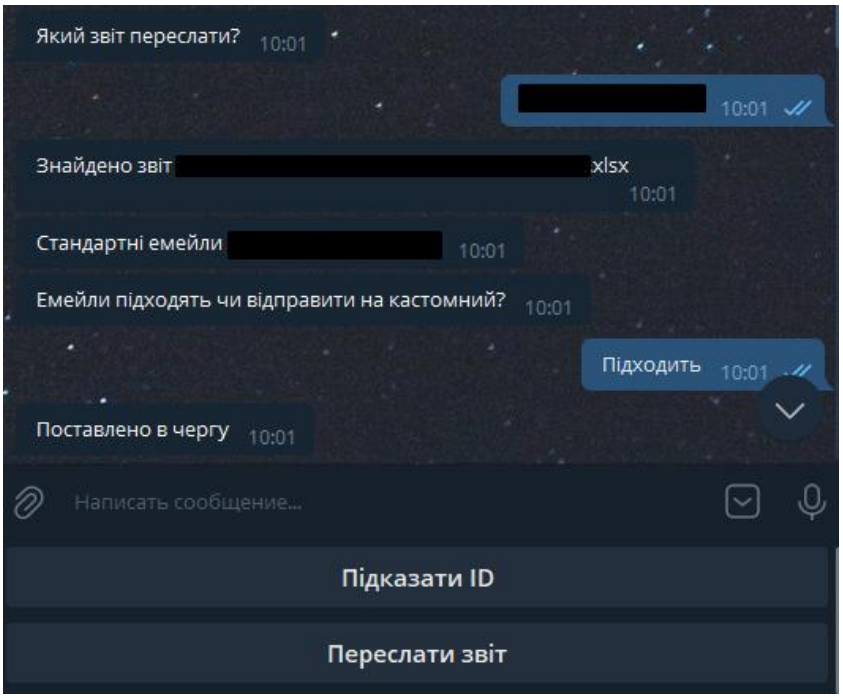

Рисунок 11. Приклад роботи сценарію повторної відправки звітів клієнтам

3.5 Перевірка даних для звітів

Постійно виникають ситуації коли в одній з таблиць дані не встигли оновитися чи хтось випадково завантажив некоректні дані. Тому було прийнято рішення створити інструмент для перевірки даних для звітів перед безпосередньою генерацією звіту.

Користувач вказує номер місяця, звіти у якому потрібно перевірити інформацію.

```
@dp.message_handler(state=ReportValuesCheckStates.start)
async def _(message: types.Message, state: FSMContext):
    try:
        month = int(message.text)
        if month > 0:
            check_values(month)
            await message.answer(
                'Закінчив перевірку, результати збережені тут: ***')
            await message.answer('Починаю')
        else:
            await message.answer('Приймаю місяць тільки за його порядковим номером')
    except:
        await message.answer('Приймаю місяць тільки за його порядковим номером')

    await state.finish()
```

У методі check_values бот збирає та агрегує дані з різних таблиць і проводить тринадцять перевірок значень на ознаки помилок, а саме:

- Перевірка глобальної ставки за звітний період;

  ```
  (sm_gl_b_m - global_budget) end as check_1,
  ```

- Перевірка суми з урахуванням дії ліцензії;

  ```
  (global_budget - global_budget_license) as check_2,
  ```

- Перевірка коригувань за звітний період;

  ```
  (cor - correction_previous) as check_3,
  ```

- Перевірка коригувань за попередні роки;

  ```
  (corr_prev - correction_previous_yrs) as check_4,
  ```

- Перевірка загальної вартості наданих клієнтами послуг, що підлягає оплаті;

```
(l - payment_total) as check_5,
```

- Перевірка граничної суми оплати у звітному періоді згідно з договором;

```
case when packet in ('packets') and SUBSTRING(report, 11, 1) = '2'
    then (sm_pay_m - contract_max - sm_gl_b_m)
    when packet in ('packets') and SUBSTRING(report, 11, 1) = '1'
    then (sm_pay_m - contract_max - sm_budget_m)
    else (sm_pay_m - contract_max) end as check_6,
```

- Перевірка різниці між граничними сумами сплат у звітному періоді;

```
case when packet in ('packets') and SUBSTRING(report, 11, 1) = '2'
then (sm_pay_m - l - payment_diff - sm_gl_b_m)
    when packet in ('packets') and SUBSTRING(report, 11, 1) = '1'
    then (sm_pay_m - l - payment_diff - sm_budget_m)
    else (sm_pay_m - l - payment_diff) end as check_7,
```

- Перевірка загальної вартості медичних послуг за пакетом, що підлягає оплаті за звітний період;

```
case when l > sm_pay_m then sm_pay_m - packet_display else l - packet_display end as check_8,
```

- Перевірка різниці між граничними сумами сплат у попередніх звітних періодах;

```
(smm_sum_az - sp_sum_ba - payment_diff_prev) as check_9,
```

- Перевірка сплаченої суми за звітний період;

```
(d_uab_sum_bal_av - period_prepayment) as check_10,
```

- Перевірка суми, що підлягає оплаті за результатами звітного періоду;

```
(l - d_uab_sum_bal_av - payment_sum) as check_11,
```

- Перевірка суми до сплати у межах граничної суми звітного періоду;

```
(case when (sm_pay_m - l) > 0
    then (case when (sm_pay_m - l + smm_sum_az - sp_sum_ba) < 0
        then ( case when l - d_uab_sum_bal_av + sm_pay_m - l + smm_sum_az - sp_sum_ba < 0
            then 0 else (l - d_uab_sum_bal_av + sm_pay_m - l + smm_sum_az - sp_sum_ba) end)
        else (l - d_uab_sum_bal_av) end
    )
    when (sm_pay_m - l) <= 0
    then (case when (sm_pay_m - l + smm_sum_az - sp_sum_ba) < 0
        then (case when sm_pay_m - d_uab_sum_bal_av + smm_sum_az - sp_sum_ba < 0
            then 0 else sm_pay_m - d_uab_sum_bal_av + smm_sum_az - sp_sum_ba end)
        else (l - d_uab_sum_bal_av) end
    )
    else null end) - payment_sum_max as check_12,
```

- Перевірка залишку граничної суми оплати у звітному періоді (нарастаючим підсумком).

```
sm_pay_m - l + smm_sum_az - sp_sum_ba - payment_diff_sa) as check_13 from checker
```

l – загальна вартість наданих клієнтами послуг, що підлягають оплаті

```
round((case when sml.packet in ('packets')
then sml.dsg_total_sum + coalesce(cor.corr, 0) + coalesce(cor_prev.corr_prev, 0)
when sml.packet in ('packets')
then coalesce(cor.corr, 0) + coalesce(cor_prev.corr_prev, 0) + sml.global_budget_license
when sml.packet in ('packets') and SUBSTRING(sml.report_number, 15, 1) = '2'
then sml.dsg_total_sum + coalesce(cor.corr, 0)
+ coalesce(cor_prev.corr_prev, 0) + sml.global_budget_license
when sml.packet in ('packets') and SUBSTRING(sml.report_number, 15, 1) = '1'
then coalesce(cor.corr, 0) + coalesce(cor_prev.corr_prev, 0) + sml.global_budget_license
else null end)::numeric, 2) as l
```

У кінці користувач отримує шлях до файлу з результатами перевірок по кожному звіту у місяці. Сценарій роботи описаного у цьому пункті функціоналу можна побачити на рисунку 12, а результат роботи на рисунку 13.

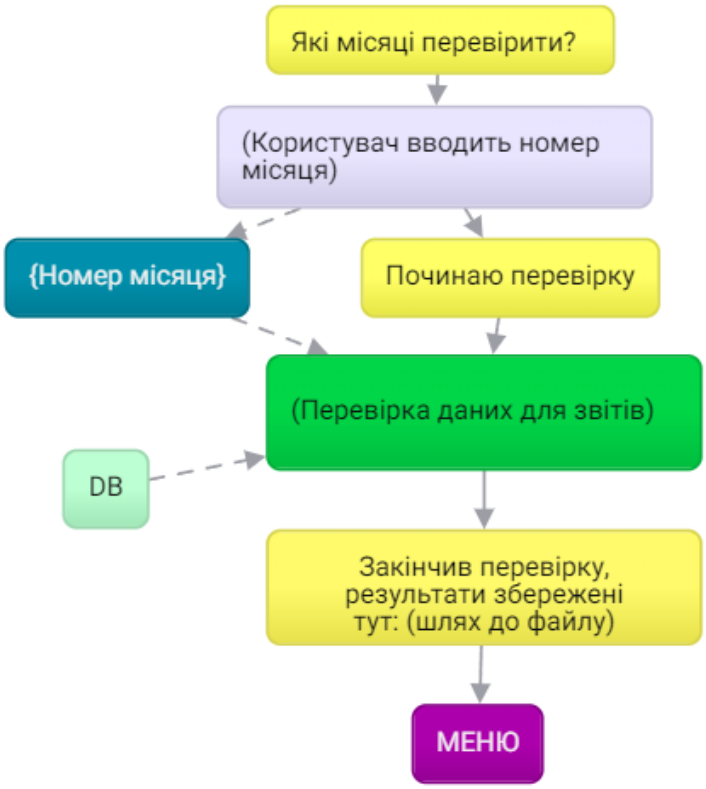

Рисунок 12. Сценарій роботи функціоналу для перевірки даних для звітів

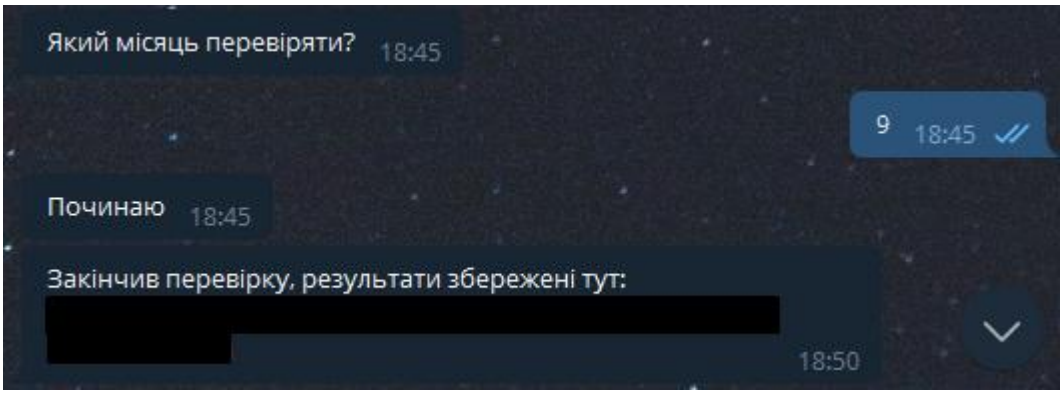

Рисунок 13. Приклад роботи функціоналу для перевірки даних для звітів

3.6 Оновлення даних у таблицях необхідних для генерації звітів

Для пришвидшення робочих процесів було розроблено функціонал для швидкого оновлення необхідних для генерації таблиць у базі даних. У процесі генерації звітів використовується декілька таблиць, а саме:

- Таблиця контрактів – у ній зберігається інформація про наявні контракти на пакети. На базі цієї таблиці створюється таблиця сум звітів;

- Таблиці ліцензій – у цих таблицях зберігається інформація на підставі якої обраховується кількість днів підчас яких клієнт міг надавати послуги згідно його ліцензій, обов'язкова наявність яких визначена у договорі;
- Таблиця підписання договорів – існує декілька часових проміжків у продовж року за які клієнт може підписати договір з установою, в цій таблиці зберігається інформація до якого проміжку відноситься цей договір;
- Таблиця коригувань – таблиця для коригувань сум виплат клієнтам, коригування можуть бути як на доплату, так і на зняття коштів;
- Таблиця сум звітів – основна таблиця у якій на базі усіх інших використовуваних таблиць формуються суми виплат і саме на її базі формуються кінцеві файли звітів.

Для оновлення таблиць коригувань був створений сценарій у якому користувачу спочатку потрібно занести коригування до файлу які при використанні функціоналу «Оновити коригування» будуть зчитані.

```
@dp.message_handler(state=ReportsStates.updatecorr)
async def _(message: types.Message, state: FSMContext):
    if message.text == 'Так':
        con = db_engines.con()
        df = pd.read_excel('V:/Коригування.xlsx', sheet_name='2022',
                           dtype={'edrpou': str, 'packet': str, 'payment_type': str})
        df.columns = ['edrpou', 'contract', 'packet', 'month', 'correction',
                      'comment', 'year', 'owner', 'type', 'inserted_at']

        df.edrpou = df.edrpou.str.zfill(8)
```

Далі інформація зберігається у таблицю коригувань і використовується для оновлення таблиці контрактів.

```
con.execute('delete from correction_sum')
df.to_sql('correction_sum', con, if_exists='append', index=False)
cor_q = """
update contract_lines con
set corr_prev = 0;

update contract_lines con
set cor_prev = cor.correction::numeric
from correction_sum cor
where con.edrpou = cor.edrpou and con.m_month = cor.month
and con.m_year = cor.year and con.packet = cor.packet
and con.ptype = cor.ptype and con.contract_name = cor.contract
"""
con.execute(cor_q)
con.close()
```

Сценарій роботи описаного у цьому пункті функціоналу можна побачити на рисунку 14, а результат роботи на рисунку 15.

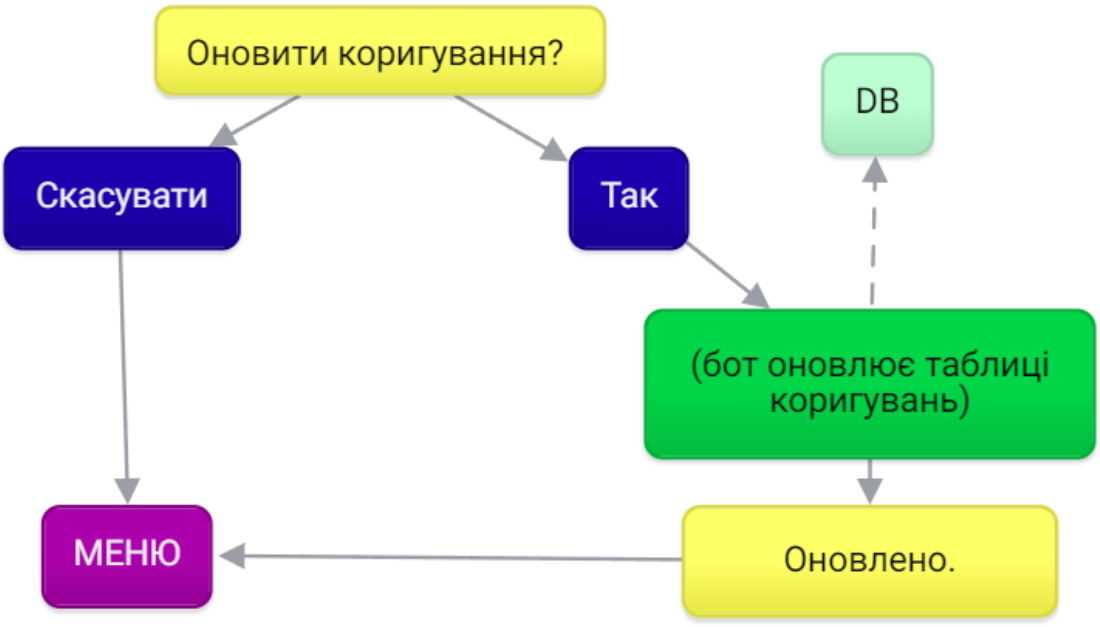

Рисунок 14. Сценарій роботи функціоналу для оновлення таблиці коригувань.

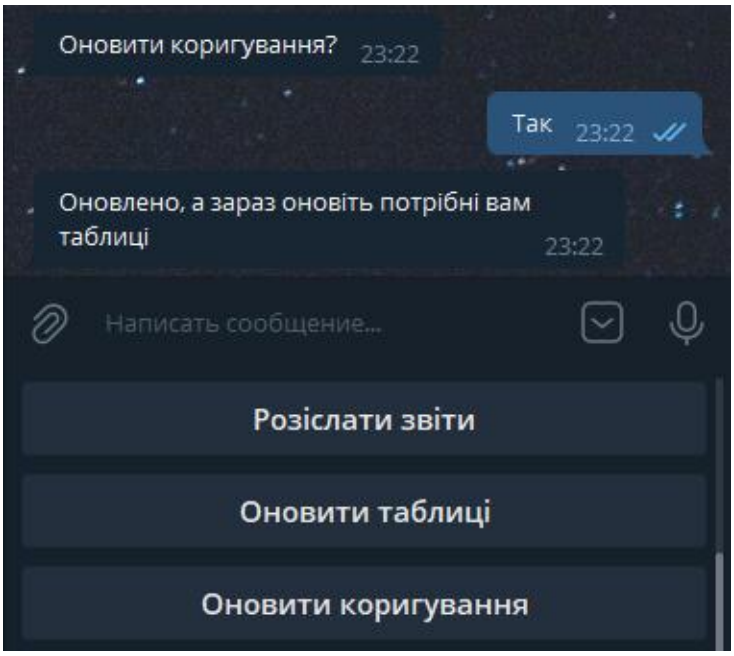

Рисунок 15. Приклад роботи функціоналу для оновлення таблиці коригувань

Для оновлення таблиць контрактів, ліцензій та таблиць підписання договорів був створений сценарій «Оновити таблиці». Сценарій роботи описаного у цьому пункті функціоналу можна побачити на рисунку 16, а результат роботи на рисунку 17.

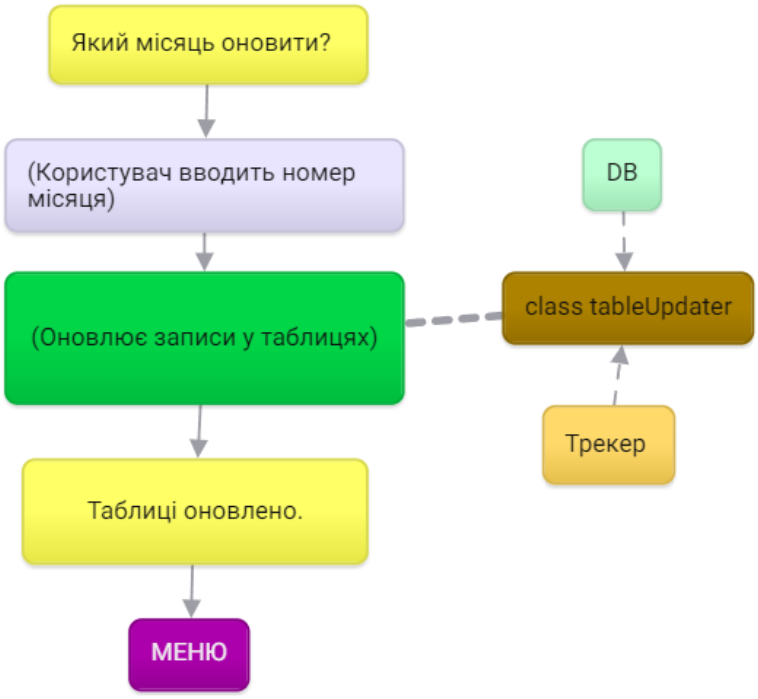

Рисунок 16. Сценарій для оновлення допоміжних таблиць

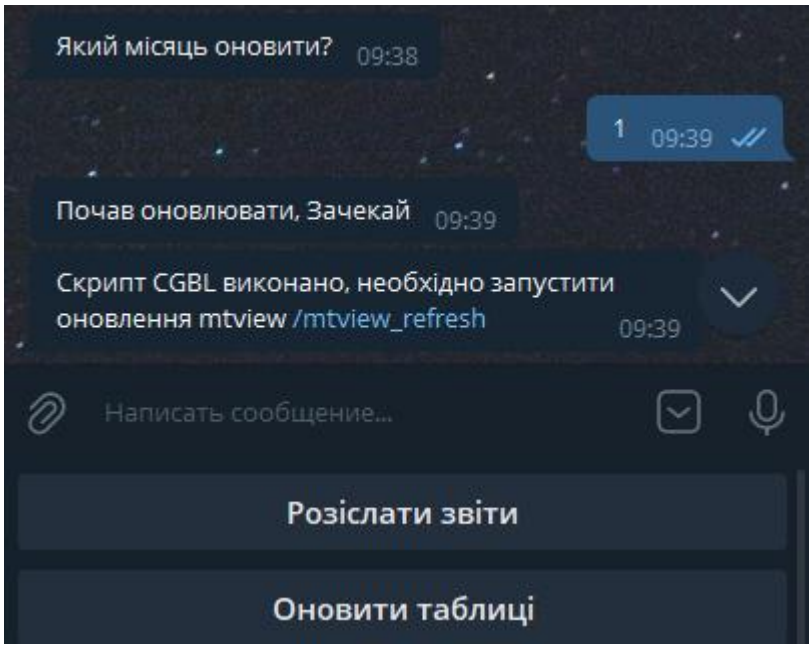

Рисунок 17. Приклад роботи функціоналу для оновлення допоміжних таблиць

Для оновлення таблиць у сценарії «Оновити таблиці» використовується клас tableUpdater та його метод updateTables який використовує вказаний користувачем місяць і поточний рік для отримання необхідної інформації, наприклад цей метод використовується для отримання інформації про частину ліцензій клієнтів.

```
def get_licenses_data(month) -> pd.DataFrame:
    lics_q = f"""SELECT * FROM licenses where date_t('month', date) = '{year}-{month}-01'"""
    lics = pd.read_sql_query(lics_q, con)
    lics['month'] = lics['date'].dt.month

    return lics
```

Подібні для цього методи використовуються для отримання іншої інформації, а саме інформацію по контрактам, відкладеним вимогам, коригуванням та іншої частини необхідних ліцензій.

Після отримання інформації спочатку коригуємо статус дії договору, враховуючи стан виконання відкладених вимог. Потім розраховуємо частки днів з діючими ліцензіями.

```
lics['lpractice'] &= lics['payment_true']
lics['npractice'] &= lics['payment_true']

dfractions = dlics['dc_days'] / days_in_month
lfractions = lics.groupby(['contract', 'edrpou', 'packet'])['lpractice'].mean()
nfractions = lics.groupby(['contract', 'edrpou', 'packet'])['npractice'].mean()

lfractions.update(dfractions)
nfractions.update(dfractions)
```

Далі перевіряємо вимогу пріоритетної ліцензії і якщо вона вимагається використовуємо тривалість її дії, якщо ні, то використовуємо тривалість звичайної ліцензії.

```
contracts = contracts.merge(ndict, 'left',
                            left_on=['contract', 'edrpou', 'packet'],
                            right_on=['contract', 'edrpou', 'code'])

contracts['nrequirement_cond'].fillna(False, inplace=True)

contracts['lic'] = contracts.contract_name_edrpou_packet.map(lfractions)
contracts['narc'] = contracts.contract_name_edrpou_packet.map(nfractions)

contracts['global_budget_license'] = contracts['global_budget_license'] * contracts['lic']
contracts.loc[contracts['nrequirement_cond'],
              'global_budget_license'] = contracts['global_budget_license'] * contracts['narc']
```

Також потрібно визначити відсутні контракти в таблиці підписання договорів, якщо вони там відсутні, то визначити часовий проміжок у якому вони були підписані та додати їх індекс до таблиці.

```
contracts_no_index = append_index(contracts, month, year)
if contracts_no_index is not None:
    contracts_no_index.to_sql('contract_number', con,
                              if_exists='append', index=False, method='multi')
```

Для цього використовується метод append_index у якому визначається список відсутніх у таблиці підписання договорів

```
ex_contracts = pd.read_sql_query(f'select * from contract_number where m_year = {year}', con)

contracts_no_index_list = contracts.loc[
    (~contracts['contract_name'].isin(ex_contracts['contract_name']))
    & (contracts['packet'].astype(float).isin(packets)), 'contract_name']
```

Далі визначаємо чергу підписання цих договорів та конвертуємо її у відповідну їй букву для подальшого використання її у формуванні номеру звіту.

```
def count_index(row: pd.Series):
    edrpou = row['edrpou']
    index = row['number']
    cn = ex_contracts
    while len(cn[(cn['edrpou'] == edrpou) & (cn['number'] == index)]):
        index += 1
    row['number'] = index
    return row

contracts_no_index = contracts_no_index.apply(count_index, axis=1)
contracts_no_index['number_letter'] = (contracts_no_index['number'].astype(int) + 64).apply(chr)

return contracts_no_index[ex_contracts.columns]
```

Залишається тільки завантажити оновлення до відповідної таблиці.

```
contracts.to_sql('contract_lines', con, if_exists='append', index=False, method='multi')
```

3.7 Обробка інформації для використання нової ліцензії

У 2022 році для більш надійного контролю за виконанням клієнтом необхідних вимог було прийнято рішення перевіряти ще одну ліцензію. Самі ліцензії видаються іншими органами контролю. Створений функціонал використовується тільки для перевірки їх наявності та тривалості їх дії. Через бюрократичні складнощі з отриманням даних по цій ліцензії інформацію доводиться отримувати напряму від клієнтів, тому дані для перевірки отримуються з .xlsx файлів.

На початку потрібно стандартизувати та обробити отримані дані. Для цього виконаємо декілька перетворень і відсортуємо дані.

```
conv.edrpou = conv.edrpou.str.zfill(8)
conv['start_date'] = conv['start_date'].apply(lambda x: x.replace('.', ''))
conv['end_date'] = conv['end_date'].apply(lambda x: x.replace('.', ''))
conv['inserted_at'] = conv['inserted_at'].apply(lambda x: x.split(',')[0].replace('.', ''))

conv['start_date'] = pd.to_datetime(conv['start_date'], format='%d%m%Y')
conv['end_date'] = pd.to_datetime(conv['end_date'], format='%d%m%Y')
conv['inserted_at'] = pd.to_datetime(conv['inserted_at'], format='%d%m%Y')

conv['inserted_at'] = pd.to_datetime(conv['inserted_at'], format='%d%m%Y')
conv = conv.sort_values(['edrpou', 'end_date']).drop_duplicates(subset=['edrpou'], keep='last')
conv = conv.reset_index(drop=True)
```

Після цього отримуємо такий DataFrame дат для роботи з ними.

	edrpou	start_date	end_date	inserted_at
0	00184945	2009-05-20	2024-05-20	2020-11-20
1	00185011	2009-07-08	2024-06-27	2021-06-15
2	00185028	2009-04-01	2024-04-01	2021-06-15
3	00440221	2013-04-23	2025-04-23	2020-11-20
4	01107935	2018-12-05	2021-12-05	2020-11-20
...	...	...	...	...
1294	43877579	2021-09-29	2026-09-29	2021-10-04
1295	44018588	2021-09-15	2026-09-15	2021-09-21
1296	44075316	2021-08-12	2026-08-12	2021-08-14
1297	44294992	2021-12-08	2026-12-08	2021-12-14
1298	44497955	2017-05-24	2022-05-24	2022-01-21

1299 rows × 4 columns

Рисунок 18. DataFrame з обробленими даними клієнтів

У цього виду ліцензії є одна особливість – якщо клієнт не має необхідного обладнання, то він може скористатися послугами зовнішнього підрядника і замість перевірки ліцензії клієнта потрібно перевіряти ліцензію підрядника.

Ліцензії підрядників отримуються у схожому форматі, тому використовується схожий на попередньо розглянути код для обробки файлів з даними клієнтів і код для впорядкування даних.

```
edr.edrpou = edr.edrpou.str.zfill(8)
edr.edrpou_clinic = edr.edrpou_clinic.str.zfill(8)

p['podr_inserted_at'] = pd.to_datetime(p['podr_inserted_at'], format='%d%m%Y')
p = p.sort_values(['edrpou', 'podr_end_date']).drop_duplicates(subset=['edrpou'], keep='last')
p = p.reset_index(drop=True)
p = p[['edrpou', 'podr_start_date', 'podr_end_date']]
p = p.rename(columns={'edrpou': 'edrpou_clinic'})

pod = pd.merge(edr, p, on=['edrpou_clinic'], how='left')
pod = pod.sort_values(['edrpou', 'packet_number', 'podr_end_date']).drop_duplicates(
    subset=['edrpou', 'packet_number'], keep='first')
pod = pod.reset_index(drop=True)
```

Після цього отримуємо такий DataFrame з даними підрядників для роботи з ними.

	packet_number	edrpou	edrpou_clinic	podr_start_date	podr_end_date
0	4	00185028	36484040	2014-04-18	2022-04-18
1	4	01110937	34639424	2010-06-04	2023-06-04
2	3	01111032	38756199	NaT	NaT
3	4	01111032	38756199	NaT	NaT
4	4	01111598	30742592	2014-06-05	2019-06-05
...	...	...	...	...	...
592	3	44591929	44496574	NaT	NaT
593	4	44591929	44496574	NaT	NaT
594	3	44625774	38557859	2014-10-31	2026-10-31
595	4	44625774	38557859	2014-10-31	2026-10-31
596	38	44625774	01996792	2011-01-26	2024-01-26

597 rows × 5 columns

Рисунок 19. DataFrame з обробленими даними підрядників

Тепер об’єднуємо створені раніше DataFrame-ми і додаємо новий індикатор use_podr для відображення який клієнт може користуватись послугами підрядника, а який ні.

```
comp['use_podr'] = comp['use_podr'].map(d)
comp = pd.merge(comp, conv, on=['edrpou'], how='left')
comp = pd.merge(comp, pod, on=['edrpou', 'packet'], how='left')
comp['start_date'] = comp['start_date'].apply(pd.to_datetime)
comp['end_date'] = comp['end_date'].apply(pd.to_datetime)
comp['podr_start_date'] = comp['podr_start_date'].apply(pd.to_datetime)
comp['podr_end_date'] = comp['podr_end_date'].apply(pd.to_datetime)
```

Далі розбиваємо часові проміжки на кількість днів дії ліцензії у кожному місяці протягом поточного року.

```
split = pd.DataFrame(comp.apply(lambda x: get_durations(x['start_date'].date(), x['end_date'].date(), x['use_podr'],
                                                        x['podr_start_date'].date(), x['podr_end_date'].date()),
                                axis=1).to_list(),
                     columns=['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12'])
comp = pd.concat([comp, split], axis=1)
```

Для цього використовуємо два методи get_durations і get_months_and_durations.

```
def get_durations(zoz_start_date, zoz_end_date, use_podr, podr_start_date, podr_end_date):
    if ((pd.isnull(zoz_start_date) or pd.isnull(zoz_end_date)) and not use_podr) or (
            ((pd.isnull(zoz_start_date) or pd.isnull(zoz_end_date)) and use_podr) and (
            pd.isnull(podr_start_date) or pd.isnull(podr_end_date))):
        return z
    elif (not pd.isnull(zoz_start_date)) and (not pd.isnull(zoz_end_date)) and not use_podr:
        start_date = zoz_start_date if zoz_start_date >= date(year, 1, 1) else date(year, 1, 1)
        end_date = zoz_end_date if zoz_end_date <= date(year, 12, 31) else date(year, 12, 31)
        return get_months_and_durations(start_date, end_date)
    elif (pd.isnull(zoz_start_date) or pd.isnull(zoz_end_date)) and use_podr and (
            (not pd.isnull(podr_start_date)) and (not pd.isnull(podr_end_date))):
        start_date = podr_start_date if podr_start_date >= date(year, 1, 1) else date(year, 1, 1)
        end_date = podr_end_date if podr_end_date <= date(year, 12, 31) else date(year, 12, 31)
        return get_months_and_durations(start_date, end_date)
    else:
        zoz_start_date = zoz_start_date if zoz_start_date >= date(year, 1, 1) else date(year, 1, 1)
        zoz_end_date = zoz_end_date if zoz_end_date <= date(year, 12, 31) else date(year, 12, 31)
        zoz = get_months_and_durations(zoz_start_date, zoz_end_date)
        if not pd.isnull(podr_start_date) and not pd.isnull(podr_end_date):
            podr_start_date = podr_start_date if podr_start_date >= date(year, 1, 1) else date(year, 1, 1)
            podr_end_date = podr_end_date if podr_end_date <= date(year, 12, 31) else date(year, 12, 31)
            podr = get_months_and_durations(podr_start_date, podr_end_date)
        else:
            podr = z
        return list(map(lambda x, y: max(x, y), zoz, podr))
```

get_durations відповідає за визначення яку з ліцензій клієнта чи підрядника можна використовувати, наприклад якщо клієнт і його підрядник

обидва мають ліцензію, але у одного вона закінчується коли у іншого вона тільки починається, то потрібно обрати кращу кількість днів у місяці за яку компанія повинна заплатити.

```
def get_months_and_durations(start_date, end_date):
    current = start_date
    result = [current]

    current = current.replace(day=1)
    while current <= end_date:
        current += timedelta(days=32)
        current = current.replace(day=1)
        result.append(datetime(current.year, current.month, 1).date())

    durations = []
    for curr in result[:-1]:
        curr_range = calendar.monthrange(curr.year, curr.month)
        curr_duration = (curr_range[1] - curr.day) + 1

        if (curr.month == end_date.month) & (curr.year == end_date.year) & (curr.month != start_date.month):
            durations.append(end_date.day)
        elif (curr.month == end_date.month) & (curr.year == end_date.year) & (curr.month == start_date.month):
            durations.append(end_date.day - start_date.day + 1)
        else:
            durations.append(curr_duration)
    result = result[:-1]
    cond_days = []
    for i in range(1, 13):
        if len(result) == 0 or result[0].month != i:
            cond_days.append(0)
        else:
            cond_days.append(durations[0])
            del durations[0]
            del result[0]
    return cond_days
```

get_months_and_durations використовується для обчислення кількості днів у кожному місяці між датою початку та датою закінчення періоду. Повертає список з дванадцяти чисел які відповідають кількості днів у кожному місяці, що попадають у проміжок між цими двома датами.

Далі потрібно розгорнути отриману таблицю, щоб отримати рядок у якому відображається тільки кількість днів для відповідного йому місяця. Отриману таблицю можна побачити на рисунку 20.

	edrpou	contract_number	packet	month_m	custom_cond_days	year_m
0	01107935	0030-E122-P000	10	1	0	2022
1	01111227	0038-E122-P000	10	1	31	2022
2	01280527	0051-E122-P000	10	1	31	2022
3	01280970	0053-E122-P000	10	1	31	2022
4	01981224	0057-E122-P000	10	1	31	2022
...	...	...	...	...	...	...
36953	05483581	0967-E122-P000	4	12	0	2022
36954	05492255	0985-E122-P000	20	12	31	2022
36956	01994959	0385-E122-P000	4	12	31	2022
36957	01994959	0385-E122-P000	3	12	31	2022
36959	01994959	0385-E122-P000	17	12	31	2022

36132 rows × 6 columns

Рисунок 20. Кінцева таблиця для визначення кількості днів дії ліцензії

У кінці зберігаємо отриману таблицю для подальшого використання її у класі tableUpdate.

```
comp.to_sql('dlicense_cond_days', con, if_exists='replace', index=False)
```

ВИСНОВКИ

Розроблено програмний механізм генерації і перевірки звітів, який доводить ефективність використання стеку технологій, що включає в себе мову програмування Python, програмного фреймворку Aiogram, бібліотеки Pandas та набору інструментів розробки SQLAlchemy.

В роботі продемонстровано, що шляхом введення додаткових багаторівневих фінансових маркерів, що уточнюють формування собівартості і ціноутворення, а також етап розрахунку із замовниками дозволяє істотно прискорити оновлення таблиці інтегральних показників звітів з 15-20 хвилин до 2-3 хвилин, а також вирішити задачу динамічної зміни кількості записів під час формування звіту.

Створена підпрограма для швидкого оновлення таблиць, необхідних при генерації звітів, яка показує, що введення і попередня обробка таблиць коригувань, таблиць ліцензій та контрактів дозволяє оптимізувати обчислювальні ресурси і скоротити час оновлення даних перед початком генерації звіту.

Вдосконалений процес генерації звіту демонструє, що шляхом введення програмної перевірки додаткової інформації на етапі генерації звіту можна повністю виключити необхідність ручного виправлення згенерованих звітів, що вимагала попередня система.

В програмному комплексі вперше введено механізм контролю зі сторони замовника за процесом виконання замовлених ним послуг, а саме контроль за тим, що упродовж усього терміну виконання замовлення виконавець має всі необхідні ліцензії на господарську діяльність і на спеціалізовані комерційні програмні продукти, а отже у такий спосіб не порушує Господарський кодекс України та Закон про авторські та суміжні права.

ПЕРЕЛІК ПОСИЛАНЬ

1. Wes McKinney. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. First Edition. – 2013. – P. 155-218, 251-289

2. Pandas Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://pandas.pydata.org/docs/user_guide/index.html (дата звернення: 15.04.2022)

3. SQLAlchemy Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.sqlalchemy.org/en/14/ (дата звернення: 10.04.2022)

4. Aiogram Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aiogram.dev/en/latest/ (дата звернення: 10.04.22)

5. Telegram FAQ [Електронний ресурс] – Режим доступу до ресурсу: https://telegram.org/faq/ (дата звернення: 25.04.2022)

6. Telegram Bot API [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/api/ (дата звернення: 25.04.22)

7. PostgreSQL Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/ (дата звернення: 21.03.2022)