МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) Автоматизації і комп'ютерних систем
Кафедра Інформаційних технологій, штучного інтелекту і кібербезпеки

«До захисту в ЕК»		**«До захисту допущено»**	
Директор інституту(декан факультету)		Завідувач кафедри	
_____	___Андрій ФОРСЮК__	_____	_Сергій ГРИБКОВ__
(підпис)	(ім'я та прізвище)	(підпис)	(ім'я та прізвище)
«02» __червня_____ 2025р.		«02» ____червня_____ 2025р.	

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

зі спеціальності 122 «Комп'ютерні науки»
(код та назва спеціальності)

освітньо-професійної програми __Комп'ютерні науки

на тему: Розроблення інформаційної системи для ТОВ "САППОРТ СЕРВИС СОЛУШНС"

Виконав: здобувач _4_ курсу, групи __КН-4-4ск___
__Радзіловський Євгеній Олексійович _____ _____
(прізвище, ім'я, по батькові повністю) (підпис)

Керівник _Литвинов Валерій Андроникович _____ _____
(прізвище , ім'я та по батькові повністю) (підпис)

Консультанти_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

Рецензент _____ _____
(ім'я та прізвище) (підпис)

Я як здобувач(ка) Національного університету харчових технологій розумію і підтримую політику університету з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволеної допомоги під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач_____
(підпис)

Київ – 2025р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) Автоматизації і комп'ютерних систем
Кафедра Інформаційних технологій, штучного інтелекту і кібербезпеки
Освітній ступінь бакалавр
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інформаційних технологій, штучного інтелекту і кібербезпеки
Сергій ГРИБКОВ

" 28 " квітня 2025 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Радзіловського Євгена Олексійовича
(прізвище, ім'я, по батькові)

1. Тема роботи **Розроблення інформаційної системи для ТОВ "САППОРТ СЕРВИС СОЛУШНС"**
керівник роботи Литвинов Валерій Андроникович, професор, доктор технічних наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від 28 квітня 2025 року № 254-кс
2. Строк подання здобувачем роботи 30.05.2025 р.
3. Вихідні дані до роботи

Структура підприємства, інформація про відділи, доступ до внутрішньої документації

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

1 Загальна характеристика підприємства ТОВ "САППОРТ СЕРВИС СОЛУШНС", 2 Технічне завдання, 3 Розробка сервісу автоматизації, Висновки, Список використаних джерел, Додатки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3	Литвинов В.А., проф.	28.04.2025	26.05.2025

7. Дата видачі завдання_____28 квітня 2025 року_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Дослідження предметної області	29.04.2025	Виконано
2	Розробка функціональної моделі	30.04.2025	Виконано
3	Розробка концептуальної моделі	01.05.2025	Виконано
4	Розробка технічного завдання	05.05.2025	Виконано
5	Визначення вимог до функцій системи	10.05.2025	Виконано
6	Реалізація задач автоматизації системи	15.05.2025	Виконано
7	Оформлення пояснювальної записки	20.05.2025	Виконано
8	Доопрацювання роботи з урахуванням зауважень керівника	26.05.2025	Виконано
9	Створення презентації	30.05.25	Виконано

Здобувач _____ _Радзіловський Є.О._____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _Литвинов В.А._____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

В даній роботі проведено аналіз діяльності компанії ТОВ “САППОРТ СЕРВИС СОЛУШНС”, яка також має іншу рекламну назву «Tranzzo». Розглянуто діяльність та структуру компанії та її відділів. Розроблено систему автоматизації процесу, який дозволяє виконувати актуалізацію статусів для транзакцій, що в свою чергу оптимізує процес обробки запитів від клієнтів компанії для відділів які за це відповідальні.

В свою чергу робота містить опис основних етапів проєктування та розробки такої системи. Головними інструметами та засобами для розробки було обрано: мову програмування Python, фреймворк для веб-серверу Django, базу даних PosgreSQL, клієнтську мову програмування JavaScript та HTML.

Кваліфікаційна робота складається з 56 сторінок, 22 рисунків, 25 використаних джерел.

Ключові слова: WEB, ІНТЕРНЕТ, PYTHON, DJANGO, POSTGRES, SQL.

ANNOTATION

This work analyzes the activities of the company "SAPPORT SERVICE SOLUTIONS" LLC, which also has another advertising name "Tranzzo". The activities and structure of the company and its departments are considered. A process automation system has been developed that allows updating statuses for transactions, which in turn optimizes the process of processing requests from the company's clients for the departments responsible for this.

In its conclusion, the work contains a description of the main stages of designing and developing such a system. The main tools and means for development were chosen: the Python programming language, the Django web server framework, the PosgreSQL database, the JavaScript and HTML client programming languages.

The qualification work consists of 56 pages, 22 figures, 25 sources used.

Keywords: WEB, INTERNET, PYTHON, DJANGO, POSTGRES, SQL.

ЗМІСТ

ВСТУП

У сучасному динамічному середовищі електронної комерції ефективна обробка платежів стала критичним елементом успіху будь-якого бізнесу. Зі стрімким розвитком цифрової економіки та електронних фінансових сервісів зростає потреба в надійних, швидких та захищених системах обробки транзакцій, які здатні забезпечити безперебійну роботу з великим обсягом платіжних операцій. Однією з ключових проблем у діяльності «Tranzzo» є процес актуалізації та фіналізації статусів транзакцій. Своєчасне та коректне оновлення статусів платежів є критичним для ефективного функціонування всієї платіжної системи, оскільки це безпосередньо впливає на фінансову звітність, аналітику та взаємодію з клієнтами. Наразі цей процес вимагає значних ресурсів та часу, що може призводити до затримок у обробці інформації та прийнятті рішень.

Розробка запропонованої інформаційної системи базується на сучасних технологіях та методологіях розробки програмного забезпечення в галузі комп'ютерних наук. Програмний продукт матиме модульну структуру, що забезпечить гнучкість та масштабованість рішення відповідно до зростаючих потреб компанії.

Таким чином, дана кваліфікаційна робота спрямована на вирішення конкретної практичної проблеми в діяльності «Tranzzo» шляхом розробки спеціалізованої інформаційної системи, яка оптимізує один з ключових бізнес-процесів компанії – актуалізацію та фіналізацію статусів транзакцій, що в кінцевому підсумку сприятиме підвищенню ефективності роботи компанії та зміцненню її позицій на ринку платіжних систем.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальна характеристика ТОВ «САППОРТ СЕРВИС СОЛУШНС»

ТОВ «САППОРТ СЕРВИС СОЛУШНС» – сучасна українська компанія, яка була заснована у 2015 році в Україні. Має маркетингову назву «Транзо», яку використовує як лице для впізнаності на ринку послуг, а також при отримані різних нагород серед інший сервісів. Також надалі ми будемо використовувати її в описі як скорочення.

Послугами компанії користуються більше 3000 клієнтів не тільки з України, а і інших частин світу, які мають дуже велику базу користувачів, від інтернет-магазину «Rozetka» до національного застосунку «Дія» разом з «UNITED24».

Основна і пріоритетна спеціалізація якої є розробка, впровадження та довготривала підтримка платіжних рішень для бізнесу. Основна діяльність компанії зосереджена на створенні технологічної інфраструктури, яка обробляє електроні платежі, забезпечуючи повний цикл операцій від ініціювання до повного завершення транзакцій. Компанія також надає різний вид комплексних рішень для інтеграції платіжних шлюзів для клієнтів. Такими рішення є сторінки для проведення оплати у веб-застосунках, мобільних додатках клієнтів, а також можливість надавати реалізацію обробки платежів клієнту з можливістю змінювати існуючу інтеграцію під свої вподобання та нюанси реалізації системи, віддаючи компанії тільки можливість займатись обробкою транзакцій до банківської установи або іншого провайдера послуг.

Оскільки компанія обслуговує великі кількість клієнтів, технічний сервіс обробляє в середньому 10 мільйонів платежів щомісяця. Це дає можливість також мати більше 180 платіжних методів для оплати, більше 100 різних валют для 190 країн по всьому світу.

1.2. Організаційна структура, роль і взаємодія підрозділів

Нижче наведена схема організаційної структури ТОВ «САППОРТ СЕРВИС СОЛУШНС» що зображена на (Рис. 1.1)

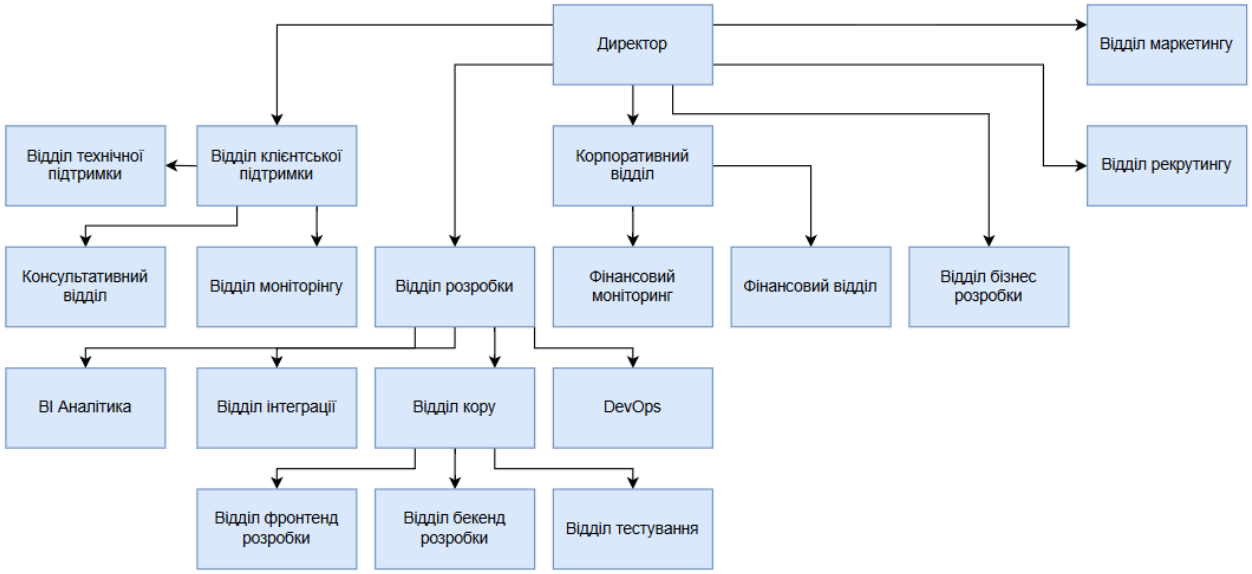

Рисунок 1.1 – Схема організаційної структури компанії

Відділів що підпорядковуються директору багато, оскільки директор відповідає, та слідкує за взаємодію між всіма відділами і їх продуктивністю для покращення кожного моменту роботи.

Вирішальне значення для майбутньої системи є відділ розробки, для якого треба буде спроєктувати сервіс, що оптимізує процес, який має вагоме значення для обробки різних типів платежів. Задачі що виконуються впливають на кожний етап, починаючи з можливості швидко обробляти запити від клієнтів до розробку нового функціоналу, який дає змогу покращити та збільшити швидкість обробки платежів.

Основні функції цього відділу:

- Розробка нового функціоналу
- Покращення існуючого функціоналу
- Тестування існуючого функціоналу
- Вирішення технічних проблем, які виникають під час користування поточними функціями сервісу

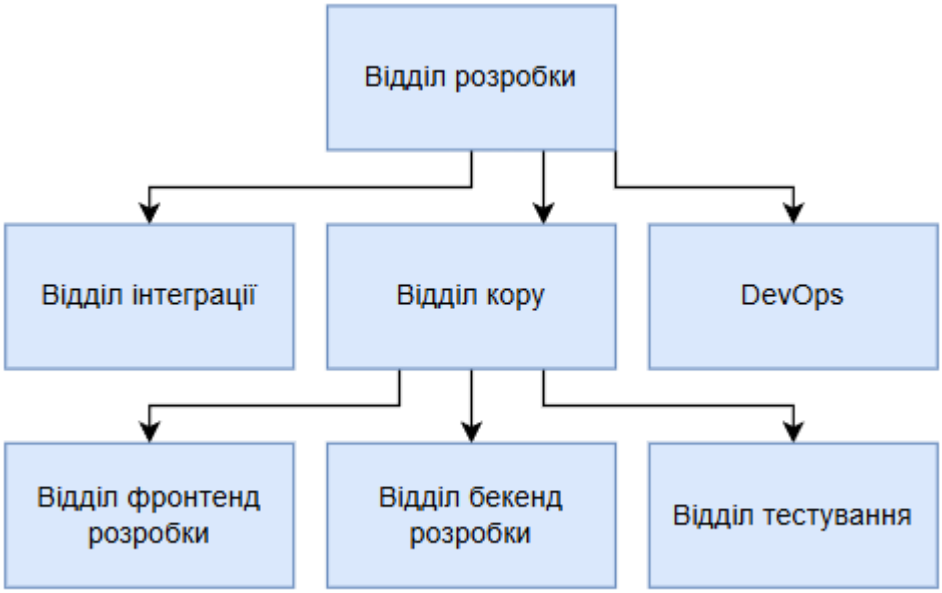

Рисунок 1.2 – Схема організації

Даний відділ взаємодіє з відділом клієнтської підтримки (Рис. 1.2), оскільки більшість задач, які потребують швидкого реагування надходить саме звідси. Ці задачі мають пріоритет, з тих причин що саме клієнтський відділ це перша лінія що взаємодіє з клієнтом і може швидко реагувати у випадку якщо під час роботи виникають технічні помилки, або для пріоритетних клієнтів потребується доробка нового функціоналу, або його виправлення.

1.2.1. Завдання та функції відділу розробки

Основним завданням відділу розробки є впровадження нового функціоналу, що покращує користувацький досвід під час користуванням продуктом. Це відповідальна функція, оскільки для такого виду розробки також необхідно розробляти додаткові внутрішні інструменти, що дають змогу реалізувати ефективний супровід та обробку додаткової інформації згідного нового функціоналу для інших відділів.

Велику та відповідальну роль в цьому відділі має підрозділ тестування, це важливий момент, оскільки всі задачі з інших відділів спочатку переходять до тестувальників. Які поміж обов’язку перевірки коректної роботи існуючого і нового функціоналу, також перевіряють всі задачі перед тим як вони потраплять

до розробників для виправлення. Більшість задач не потребують доробки, але мають помилки під час виконання роботи із-за складності розуміння як працює система і як результат попереднього невірного налаштування іншими співробітниками. Перелік функцій відділу розробки наведено у таблиці 1.1.

Таблиця 1.1 Завдання і функції відділу розробки

№	Задачі	Функції
1	Розробка нових програмних рішень для обробки транзакцій	Аналіз вимог бізнесу, проєктування архітектури, реалізація функціоналу
2	Підтримка та розвиток існуючої платформи	Виправлення помилок, оптимізація продуктивності, оновлення компонентів системи
3	Інтеграція з банками та платіжними провайдерами	Реалізація API-з'єднань, тестування взаємодії, супровід інтеграцій
4	Забезпечення безпеки платіжних процесів	Впровадження стандартів безпеки (наприклад, PCI DSS), контроль доступу, шифрування даних
5	Взаємодія з іншими відділами компанії	Отримання технічних завдань, консультації, підтримка менеджерів та клієнтської підтримки
6	Участь у DevOps-процесах	Налаштування CI/CD, автоматизація деплойменту, моніторинг стабільності сервісів
7	Документування програмних рішень	Створення технічної документації, API-описів, інструкцій для користувачів і колег
8	Впровадження нових технологій та інструментів	Дослідження інновацій, оцінка доцільності впровадження, навчання команди

1.2.2 Структура відділу розробки

Зробимо таблицю (Табл. 1.2) зі скороченим і загальним прикладом того, як саме відділ взаємодіє з іншими, та яку загалом інформацію отримує.

Таблиця 1.2 Взаємодія відділу виробництва з іншими відділами компанії

№	Відділ	Отримує	Надає
1	Бізнес-аналітики / Продакт-менеджери	Технічні завдання, бізнес-вимоги, пріоритети розвитку	Оцінку трудозатрат, прототипи, реалізовані функції, технічні зауваження
2	Відділ підтримки клієнтів	Звіти про помилки користувачів, побажання клієнтів	Статус виправлення помилок, технічні пояснення, рекомендації
3	Відділ тестування (QA)	Технічна реалізація, оновлений код, доступ до середовищ	Звіти про баги, результати тестування, підтвердження стабільності
4	Відділ DevOps	Налаштоване програмне забезпечення, вимоги до оточення	CI/CD скрипти, оновлення сервісів, логування, моніторинг
5	Відділ безпеки	Рекомендації щодо захисту даних, аудит безпеки, звіти про інциденти	Впровадження стандартів безпеки, реалізація шифрування, контроль доступу
6	Керівництво компанії	Стратегічні цілі, пріоритети проєктів	Технічні звіти, статуси проєктів, прогнози готовності
7	Відділ маркетингу та продажу	Інформація про потреби клієнтів, запити нових функцій	Демонстрації нових можливостей, технічні обмеження або потенціал

1.3 Аналіз нинішнього стану комп’ютеризації «Транзо»

Компанія притримується дещо нестандартного підходу до культури «працювати в офісі». Не дивлячись на те що офіс є не тільки в Україні, політика компанії полягає в тому, що у співробітників компанії немає персонального

робочого місця. Тобто кожен хто знаходиться в офісі, має можливість обрати будь яке місце для роботи, оскільки весь офіс притримується дизайну інтер'єра відкритого простору. Де у кожного є можливість мати робоче місце, з розетками та додатковим монітором. Кожний співробітник не має персонального робочого комп'ютера, всім без вийнятку надається ноутбук, будь то з системою Windows або MacOS.

Такий крок на перший погляд може здаватись дивним, але це підвищує мобільність та дає вибір кожному співробітнику обира будь який формат праці, віддаленої роботи з дома або працювати в офісі. Також це дає змогу контролювати ситуацію під час надзвичайних випадків, коли необхідно швидко залишити місце роботи, при тому не шкодувати за робочим приладдям та даними що там зберігаються.

Всі необхідні програми для роботи вже заздалегідь встановлені на ноутбуках працівників. Додатковим і дуже важливим кроком є встановлення VPN, програмного забезпечення, що надає додатковий рівень захисту під час обміну інформації в мережі інтернет, яким би не був захист мережі з якої підключається працівник.

З таких міркувань компанія користується багатьма готовими рішеннями інших компаній. К таким рішенням відноситься наприклад «Jira». Це програмний продукт, який дозволяє всім відділам в компанії комунікувати між собою під час розробки або впровадження нового функціоналу. Цей програмний продукт не дивлячись на користь, все ж таки має недолік при виконані певних процесів у вирішенні проблем з обробкою транзакцій, що веде за собою незручність у ефективності та втраті часу на виконання і задіяння інших відділів і підрозділів компанії.

1.4 Функціональне моделювання та аналіз існуючих бізнес-процесів

1.4.1 Побудова функціональної моделі

Для розуміння та створення власної системи, яка дозволить покращити існуючий функціонал, що вирішує проблему відділу, необхідно провести аналіз

існуючої системи. Для цього випадку ми побудуємо функціональну модель існуючого процесу використовуючи програмний продукт AllFusion Process Modeler.

Побудова такої моделі, це обов’язковий крок, оскільки така модель дозволяє обстежити наявні процеси конкретної системи. Така модель дозволяє зафіксувати об’єкти, які використовуються при виконані функцій на різних рівнях деталізації процесу. А також що є ще одним з важливих якостей аналізу, дозволяє виділити та систематизувати процеси у системі яку ми розглядаємо при її функціонуванні.

На основі моделі, досягається консенсус між різними етапами процесу по тому, «хто і що зробив» і кожен етап додається в процес. Функціональна модель є основної точкою для аналізу потреб підприємства, виявлення проблем і розробки проекту вдосконалення процесів. Модель дозволяє з'ясувати, «що і як відбувається зараз» перед тим, як визначити те, «що і як буде відбуватись» у майбутньому. Аналіз функціональної моделі дозволяє зрозуміти, де знаходиться проблемна ситуація, в чому полягатимуть переваги нових процесів і яким змінам піддасться існуюча структура організації процесу. Дослідження необхідності реструктуризації (виявлення і ліквідація недоліків) в існуючих процесах досягається за рахунок застосування декомпозиції (аналізу), що виробляється навіть там, де функціональність на перший погляд є очевидною (Рис. 1.3).

Опис системи за допомогою IDEF0 називається функціональною моделлю. Функціональна модель призначена для опису існуючих бізнес-процесів. Для передачі інформації про конкретну систему джерелом графічного мови є сама методологія IDEF0.

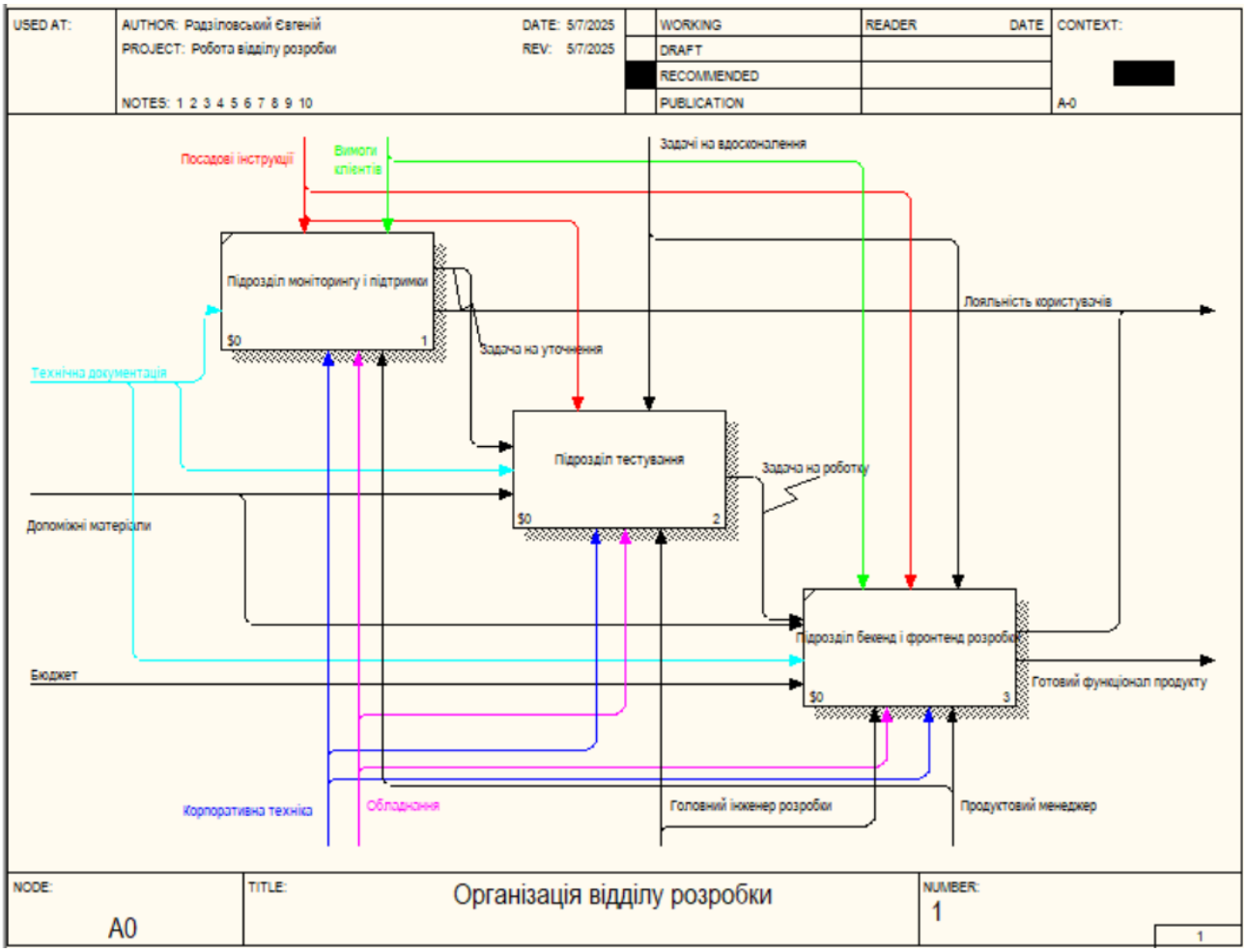

Рисунок 1.3 – Модель «AS IS». Загальна схема організації роботи відділу розробки.

Контекстна діаграма функціональної моделі що зображена. Для даної діаграми треба зробити додаткові пояснення для розуміння повного контексту роботи процесів, які складаються з:

- Вхідні дані: технічна документація, допоміжні матеріали, бюджет.
- Контролюючі дані: посадові інструкції, вимоги клієнтів, задачі на вдосконалення.
- Механізми: корпоративна техніка, додаткове обладнання, головний інженер розробки, продуктовий менеджер
- Вихідні дані: готовий функціонал продукту, лояльність користувачів.

Для першого рівня декомпозиції містить наступні загальні етапи роботи відділу:

- Збір запитів від користувачів щодо додаткових покращень, або виправлень помилок в існуючому фунціоналі;
- Первинний контроль якості програмного забезпечення;
- Створення документації описуючий новий функціонал;
- Формування звітів по змінам;
- Процес розробки нового функціоналу;

З наданих пунктів була проведена основна декомпозиція яка формує собою наступний основний етап роботи:

Процес розробки та тестування нового і доступного функціоналу системи, що обробляє транзакції та відображено на (Рис. 1.4)

- Прийом заявок від інших відділів;
- Перевірка критичності виявленої помилки;
- Консультація щодо покращення користування функціоналом для внутрішніх співробітників;
- Розробка нового функціоналу згідно новим вимогам;
- Тестування перед поточним релізом для мінімізації помилок системи;
- Допомога у відновленні синхронізації інформації в базах даних для різних сервісах;
- Виявлення нових проблемних «місць» у системі;

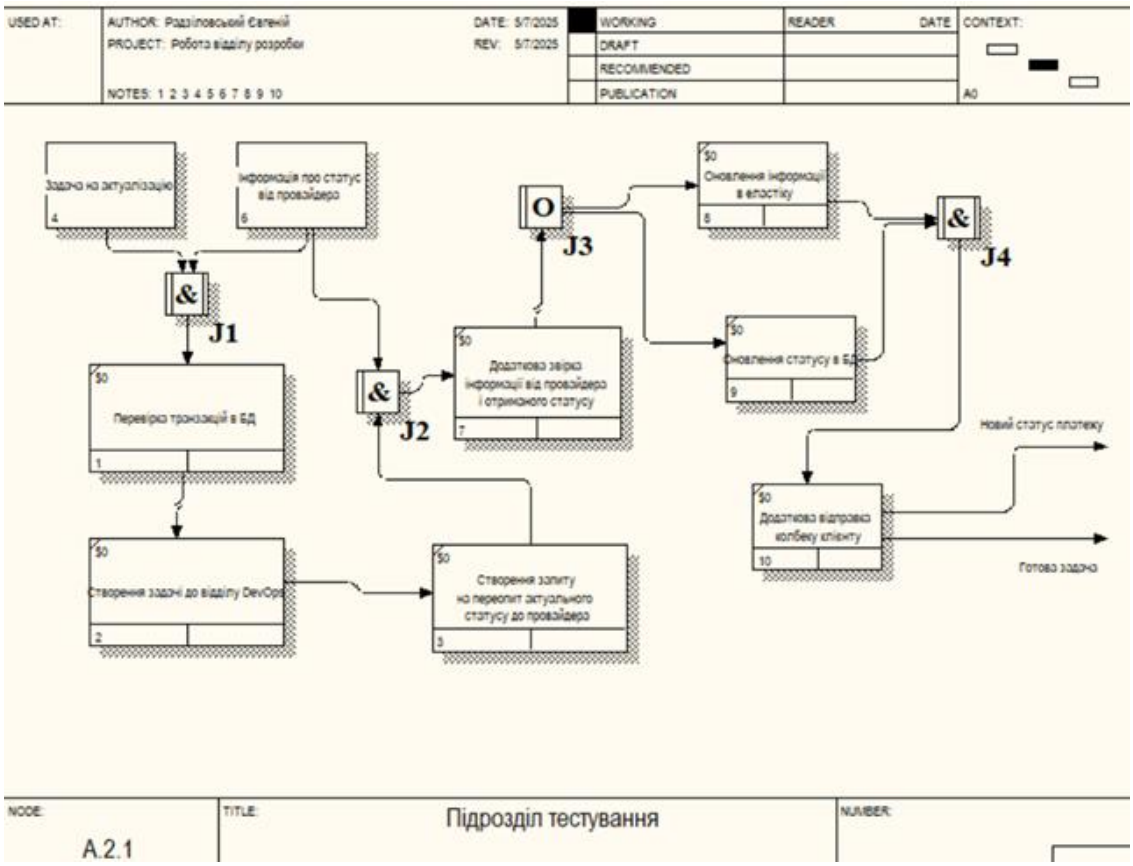

Рисунок 1.4 – Модель «AS IS». Процесс актуалізації транзакції.

Процес синхронізації інформації в базі даних щодо достовірної інформації для всіх видів транзакцій згідно оновлення від провайдера платіжних системи або банків та поточної інформації від клієнтів (Додаток А, Рис. А.3):

- Звірка інформації в базах даних щодо статусу платежів;
- За необхідності створення задач до інших відділів що мають можливість робити зміни в базах даних;
- Додатковий переопит статусів від провайдерів послуг напряму для додаткової звірки;
- Оновлення інформації в еластіку для можливості її відображення для інших відділів;
- Перевідправка інформації до клієнта з актуальним статусом платежу;

1.4.2 Виявлення проблеми

При детальному дослідженні поточної системи актуалізації платежів для компанії, було виявлено деякі основні недоліки і обмеження поточного процесу. Підхід який базується на ручній обробці даних з залученням допоміжних завдань і інших відділах, демонструє значну проблему з точки зору ефективності та точності.

Наразі поточний процес актуалізації статусів по платежам та іншим видам транзакцій відбувається наступним чином:

1. Створюється перша задача на актуалізацію транзакції від фінансового відділу до підрозділу моніторингової системи;
2. Підрозділ моніторингу проводить первинний аналіз перевірки відповідності статусів у еластику і в системі;
3. Якщо є розходження між даними та задачею від фінансового відділу, робиться запит до платіжного провайдеру або банку для додаткового уточнення статусу.
4. Далі задача передається до підрозділу тестування, який робить перевірку інформації в базі даних, оскільки має більший доступ;
5. При виявленні розбіжності в таблицях додатково створює підзадачу до відділу DevOps на виконання певних запитів для зміни інформації;
6. Після зміни зі сторони DevOps, підрозділ тестування за допомогою скриптів робить оновлення даних в еластику і додатково робить перевірку;
7. Якщо все гаразд з актуалізацію інформації всередині системи, робить відправку додаткового зворотного зв’язку системи до акаунта клієнта;

Одним із найсуттєвіших недоліків цього процесу є значні часові витрати на обробку даних. Кожне оновлення статусу від фінансового відділу до провайдера послуг потребує створення відповідного завдання, його розподілу між співробітниками, подальшого виконання та контролю. Експериментальні вимірювання середнього часу на обробку однієї транзакції показали, що від моменту надходження інформації від провайдера до фактичного оновлення в базі

даних минає від 30 хвилин до 1-2 днів, що є неприпустимим для сучасної фінансової компанії, особливо в контексті обробки термінових платежів. Також людський факто який впливає на неточності, може призводити до додаткових витрат на коригування та може спричиняти серйозні фінансові та репутаційні втрати для компанії. Ручний процес актуалізації транзакцій створює значні труднощі для забезпечення належного аудиту та відстеження змін. Відсутність автоматизованого журналювання всіх дій призводить до неможливості повноцінного контролю за внесеними змінами та ускладнює процедури аудиту, що є критичним для фінансової установи з точки зору регуляторних вимог та внутрішньої безпеки.

1.4.3 Перспективи використання розробки

Розробка нової системи вкрай необхідна, оскільки процес який було розглянуто є ключовим для компанії, яка спеціалізується на обробці платежів. Це також важлива річ для майбутніх, та поточних клієнтів, оскільки їм необхідно отримувати якомога швидше актуальну інформацію для надання послуг своїм клієнтам. Схему оптимізації такого процесу ми реалізуємо через функціональну модель «TO BE» (Рисунок 1.5), яка дозволить бачити спрощений процес що реалізується за допомогою створеного сервісу.

На схемі можна побачити що необхідність створювати нові задачі для DevOps підрозділу вже не має, оскільки сервіс самостійно робить оновлення інформації до бази даних. Це також біль безпечніше в порівнянні між старим процесом, оскільки зменшується небажаний вплив людського фактору на зміну інформації в базі даних по відношенню до інших параметрів та значень.

Як ми бачимо (Рис. 1.5), в порівнянні з існуючою схемою роботи (Рис. 1.4), новий підхід не тільки стане безпечніше, а й вбирає в себе автоматичну обробку наступних запитів, таких як додатковий перопит актуального статусу від провайдера та оновлення таблиці для еластику разом з додатковою відправкою системного зворотного виклику по конкретним платежам.

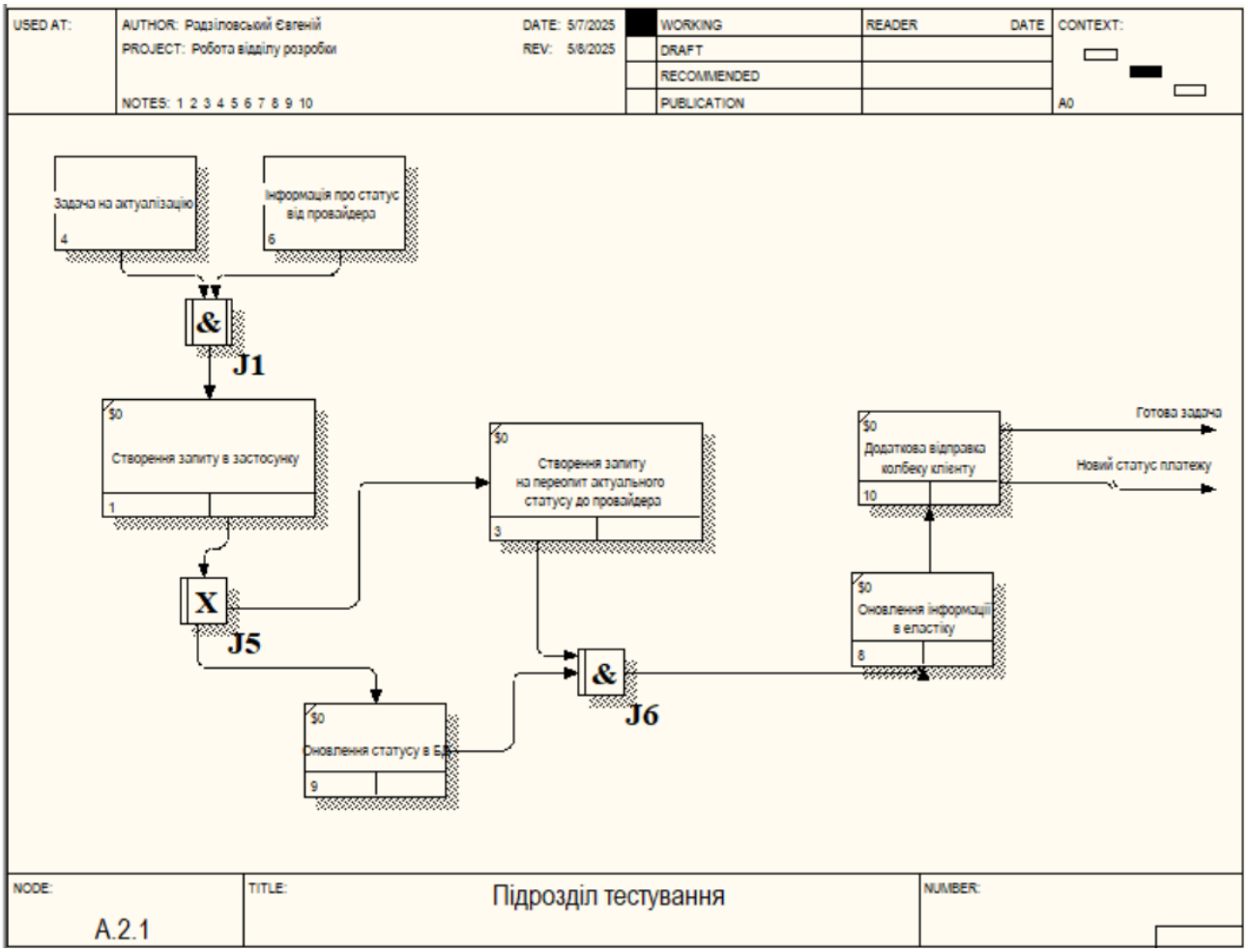

Рисунок 1.5 – Модель «TO BE». Процесс актуалізації транзакції.

Ґрунтуючись на комплексному аналізі процесів, наразі виявлено критичні системні недоліки моделі, що проявляються у надмірних часових витратах, високій залежності від людського фактора та відсутності системи аудиту. Нове рішення не лише відміняє виявлені обмеження, але й створює передумови для підвищення ефективності та дотримання регуляторних вимог, що є критичним для поточної системи.

1.5 Огляд існуючих рішень для розв’язання проблем

Під час дослідження було проведено аналіз наявних на поточний момент інформаційні системи, які пропонують певну функціональність для можливості автоматизувати процес обробки запитів на актуалізацію платежів для фінансових компаній з різним обсягом обробки і наявності клієнтів та кількості транзакцій.

1.5.1 Система «Way4»

«Way4» це платформа для обробки платіжних транзакцій, розроблена компанією «OpenWay Group», яка широко застосовується у банківському та фінтех-секторі для автоматизації процесів включаючи актуалізацію платежів. Також підтримує інтеграцію з різними провайдерами платіжних послуг, що дозволяє автоматично отримувати та обробляти статуси транзакцій.

Основні характеристики:

- Функціонал: забезпечує автоматизацію обробки транзакцій, включаючи моніторинг статусів у реальному часі та генерацію звітів;
- Інтеграція: платформа сумісна з міжнародними платіжними системами, такими як «Visa» та «Mastercard», а також локальними провайдерами в Україні;
- Інтерфейс: доступний англійською мовою, однак можлива локалізація українською за додаткову плату;
- Швидкодія: здатна обробляти до 3000 транзакцій за секунду, що робить її придатною для компаній із великим обсягом операцій;
- Вартість: являється дорогою через необхідність ліцензування та налаштування. Орієнтовна вартість стартує від 100 000 доларів, яка залежить від масштабу проєкту;

Переваги та недоліки:

Система «Way4» дозволяє значно скоротити ручну працю завдяки автоматизації, однак висока вартість і складність впровадження можуть бути бар’єрами для невеликих фінансових компаній. Крім того, система потребує додаткового кваліфікованого персоналу для підтримки та налаштування.

1.5.2 Система «Creatio»

«Creatio» це платформа для автоматизації бізнес-процесів, розроблена українською компанією «Terrasoft». Хоча система більше відома у сфері CRM, її модуль для управління процесами (BPM) може бути адаптований для автоматизації актуалізації транзакцій.

Характеристики:

- Функціональність: дозволяє створювати автоматизовані процеси для обробки транзакційних даних, інтегруватися з платіжними провайдерами через API та налаштовувати правила для оновлення статусів у базі даних;
- Швидкодія: обробка до 10 000 транзакцій за секунду і є достатнім для компаній середнього розміру;
- Сумісність: сумісна з SQL-базовими системами (Microsoft SQL Server, Oracle) та хмарними сервісами, такими як AWS.
- Вартість: ліцензія для корпоративного використання починається від 30 000 доларів на рік, що включає базовий пакет автоматизації процесів;

Переваги:

- Інтуїтивно зрозумілий інтерфейс для створення процесів;
- Можливість інтеграції з CRM-модулями для комплексного управління;
- Високий рівень підтримки від локального розробника;

Недоліки:

- Висока вартість ліцензії;
- Обмежена масштабованість для компаній із великими обсягами транзакцій;

1.5.3 Система «Corezoid»

«Corezoid» як хмарна платформа для автоматизації бізнес-процесів, розроблена українською компанією «Middleware». Вона використовується для управління транзакціями, інтеграції з платіжними провайдерами та автоматизації обробки даних. Система орієнтована на створення процесів за допомогою візуального конструктора, що зменшує потребу в глибоких знаннях програмування.

Характеристики:

- Функціональність: підтримує інтеграцію через API з платіжними провайдерами, має автоматичне оновлення статусів транзакцій у реальному часі, а також обробку великих обсягів даних;
- Швидкодія: система здатна обробляти до 100 000 транзакцій за секунду;
- Сумісність: інтегрується з більшістю сучасних баз даних (MySQL, PostgreSQL, MongoDB) та платіжних систем через REST API;
- Вартість: працює за моделлю підписки, яка залежить від обсягу транзакцій та кількості активних процесів;

Переваги:

- Висока гнучкість у налаштуванні процесів;
- Хмарна архітектура, що зменшує витрати на інфраструктуру;
- Підтримка локальних платіжних провайдерів, таких як «PrivatBank» чи «MonoBank»;

Недоліки:

- Висока початкова вартість для невеликих компаній;
- Потребує часу на навчання персоналу для ефективного використання конструктора процесів;

На основі вищезазначеного ми можемо створити орієнтовну таблицю переваг та недоліків (Табл. 1.3). Це дозволить простіше провести аналіз існуючих систем і в подальшому визначити основні моменти під час створення системи для компанії.

Таблиця 1.3 Порівняння існуючих рішень.

Критерій	**Corezoid**	**Creatio**	**Way4**
Вартість	Від $500/міс	Від $30 000/рік	Від $10 000/рік
Швидкодія (транз./сек)	100 000	10 000	50 000
Автоматизація транзакцій	Повна (API, реальний час)	Часткова (налаштування через BPM)	Повна (включає авторизацію)
Сумісність	MySQL, PostgreSQL, MongoDB	MS SQL, Oracle, AWS	Oracle, PostgreSQL, гібрид
Мова інтерфейсу	Укр, англ, рос	Укр, англ	Англ, рос (укр за доп. плату)
Складність впровадження	Середня (навчання персоналу)	Низька (інтуїтивний інтерфейс)	Висока (потрібні спеціалісти)
Підтримка локальних провайдерів	Висока (PrivatBank, MonoBank)	Середня (обмежена API)	Висока (локальні та міжнародні)
Масштабованість	Висока (хмарна архітектура)	Середня (обмеження для великих обсягів)	Висока (гібридна модель)
Рівень безпеки	Високий (ISO 27001)	Високий (GDPR, ISO)	Дуже високий (PCI DSS)
Технічна підтримка	24/7, локальна та міжнародна	Локальна, 24/7 за додаткову плату	24/7, але переважно міжнародна

Розглянуті системи не можуть відповідають потребам компанії через такі обмеження: висока вартість впровадження та підтримки, складність інтеграції з наявною інфраструктурою, недостатня гнучкість для специфічних вимог малого бізнесу, а також потреба в додаткових ресурсах для навчання персоналу чи залучення спеціалістів. Ці фактори роблять їх менш придатними для оптимізації актуалізації транзакцій у поточних умовах.

1.6 Розрахунок техніко-економічного обґрунтування впровадження створюваного програмного забезпечення

Процес компанії щодо актуалізації налаштований таким чином, що впровадження існуючих рішень не виправдовує грошові витрати на інтеграцію та навчання інших співробітників опануванню функціоналу, який значною мірою відрізняється від функціоналу внутрішнього продукту. Це також зумовлено побоюванням на можливість витоку внутрішньої інформації для конкурентів на ринку. В такому випадку більш доцільно є самостійна розробка внутрішнього невеликого сервісу, або оптимізації процесу, для якого потрібно буде мінімально застосовувати ресурси фірми на навчання інших співробітників. Оскільки персонал вже знайомий з процесом актуалізації і такий сервіс навпаки інтуїтивно можливо було зрозуміти на самому початку роботи з ним.

Також на основі представлених рішень у підрозділі 1.5.1 – 1.5.3 та таблиці, ми побачили що такі пропозиції є занадто дорогими для їх впровадження у систему тільки для однієї оптимізації з поміж інших, які також можуть мати, або вже мають певні можливості для покращення та оптимізації. Готовий функціонал таких сервісів також має перенавантажений вибір обробки, який є не актуальним на поточний момент. Тому на основі вищезгаданого, самостійна розробка та проєктування такої системи є більш обґрунтованою.

Розглянемо вартість розробки програмного забезпечення для реалізації потрібного функціоналу:

- Заробітна плата програмістів (2 особи × 1 місяць × 40 000 грн) = 80 000грн.

- Заробітна плата тестувальника (1 особа × 0,5 місяця × 35 000 грн) = 17 500грн
- Заробітна плата бізнес-аналітика (1 особа × 0,5 місяця × 35 000 грн) = 17 500грн

Всього на розробку потрібно 115 000 грн. Далі необхідно розрахувати витрати на забезпечення інфраструктури під час розгортання:

- Використання існуючого серверного обладнання: 0 грн
- Ліцензії на програмне забезпечення з відкритим кодом: 0 грн
- Додаткове обладнання для тестування: 15 000 грн

Всього витратити на інфраструктуру необхідно 15 000 грн. Надалі робимо розрахунок щодо навчання впровадження нового функціоналу системи для співробітників:

- Налаштування системи: 20 000грн
- Інтеграція з існуючими системами: 40 000грн
- Навчання персоналу: 0грн
- Документація: 10 000грн

Всього на впровадження необхідно 70 000грн. Разом інвестиційні витрати які необхідно зробити за весь період: 115 000 + 15 000 + 70 000 = 200 000грн.

1.7 Обґрунтування доцільності проєктування й розроблення

Головна ідея проєкту, який буде розроблятись полягає в оптимізації не тільки поліпшення процесу, але й можливості оптимізації у використанні апаратними потужностями. Застосунок буде являти собою веб-сервіс, який можливо відкрити у будь якому браузері, що дасть змогу мати доступ до сервісу з будь якої корпоративної техніки що видається співробітникам з доступом до корпоративного ВПН, без необхідності встановлювати будь якого програмного забезпечення.

Такий підхід також дозволить віддалено контролювати доступ і за необхідності обмежувати його. Сервіс буде створено з мінімальним внутрішнім функціоналом, який мінімізує необхідність в проведенні навчання

співробітників компанії і швидкому та зручному його використані. Якщо буде необхідно такий сервіс можливо також в майбутньому доповнювати необхідним функціоналом, а також якщо того потребує політика компанії – розробити його «з нуля» з мінімальним використанням технологій, котролюючи повністю програмний код для додаткової впевненості що дані надійно зберігаються в середині компанії і витоку інформації не відбудеться.

На основі дослідження поточної системи актуалізації платежів виявлено суттєві недоліки, які негативно впливають на ефективність роботи компанії. Багатоетапний процес з залученням різних відділів (фінансового, моніторингу, тестування та DevOps) призводить до значних часових затримок, які можуть тривати від 30 хвилин до 2 днів на обробку однієї транзакції. Людський фактор під час ручної обробки даних збільшує ризик помилок, що може спричинити фінансові та репутаційні втрати. Відсутність автоматизованого журналювання ускладнює аудит та відстеження змін, що є критичним для фінансової установи з точки зору безпеки. Готові рішення на ринку мають обмеження через високу вартість, складність інтеграції та недостатню гнучкість для специфічних потреб компанії.

Впровадження власної автоматизованої системи є доцільним рішенням, яке дозволить підвищити ефективність процесу, зменшити часові витрати та мінімізувати ризики помилок.

РОЗДІЛ 2. ТЕХНІЧНЕ ЗАВДАННЯ НА ПРОЄКТУВАННЯ

2.1 Загальні положення

Найменування системи: «Сервіс актуалізації транзакцій».

2.2 Призначення і цілі створення системи

2.2.1 Призначення системи

Сервіс актуалізації транзакцій призначений для автоматизації і спрощення процесу обробки та актуалізації статусів фінансових транзакцій в реальному часі за допомогою мінімізації звернень між відділами для досягнення поставленої мети.

2.2.2 Цілі створення системи

1. Підвищення ефективності обробки транзакцій - зменшення часу обробки транзакцій за рахунок автоматизації процесів.
2. Забезпечення високої надійності – досягнення високого показника доступності системи та зменшення кількості помилок обробки.
3. Підвищення прозорості фінансових операцій - забезпечення повної простежуваності транзакцій від ініціації до завершення.
4. Покращення безпеки фінансових операцій - впровадження системи захисту даних та моніторингу підозрілих операцій.
5. Оптимізація операційних витрат - зниження витрат на обробку транзакцій за рахунок автоматизації та оптимізації процесів.
6. Забезпечення масштабованості - створення системи, здатної обробляти зростаючі обсяги транзакцій без втрати продуктивності.

2.3 Характеристика об’єкта автоматизації

Об’єктом автоматизації є процес обробки фінансових транзакцій у компанії. Наразі процес обробки транзакцій характеризуються наступними особливостями:

- Різноманітність типів транзакцій: платежі, виплати, двостадійні платежі;
- Багатоетапність обробки: від ініціації до підтвердження завершення з проміжними статусами.
- Взаємодія з зовнішніми системами: банки-партнери, платіжні системи;.
- Регуляторні вимоги: необхідність відповідати вимогам міжнародних платіжних систем та законодавства про фінансовий моніторинг.
- Вимоги до швидкості обробки: очікування клієнтів щодо миттєвої обробки та сповіщення про статус транзакцій.

Поточні проблеми, що потребують вирішення:

- Затримки при оновленні статусів транзакцій
- Складність відстеження проблемних транзакцій
- Неоптимальні процеси обробки великих обсягів даних
- Обмежені можливості для аналітики та звітності

2.4 Вимоги до системи

2.4.1 Вимоги до системи в цілому

Система повинна мати архітектуру з наступними основними компонентами:

1. Модуль введення та валідації транзакцій:
    - Інтерфейси прийому транзакцій
    - Підсистема первинної валідації даних
    - Компонент класифікації транзакцій за типами
2. Модуль обробки транзакцій:
    - Компонент маршрутизації транзакцій
    - Підсистема бізнес-логіки обробки за типами
3. Модуль актуалізації статусів:
    - Компонент моніторингу статусів

- Підсистема автоматичного оновлення
- Компонент сповіщень про зміну статусів

4. Модуль управління та моніторингу:
    - Адміністративна панель
    - Підсистема налаштування правил обробки
5. Модуль звітності та аналітики:
    - Підсистема формування звітів
    - Підсистема експорту даних
6. Модуль безпеки та аудиту:
    - Компонент авторизації
    - Компонент захисту даних

2.4.2 Вимоги до структури і функціонування системи

Система повинна мати клієнт-серверну архітектуру, що використовує та обслуговує єдину базу даних (надалі - БД). Бізнес-логіка обробки і функціонування системи повинно бути реалізовано єдиним модулем верхнього рівня, що дозволить робити міграцію кодової бази системи до іншого джерела розгортання для серверної архітектури виконання. Графічне відображення інтерфейсу системи у вигляді кодової бази повинно мати можливість зберігатись на іншому сервері для запобігання підробки файлів і збереженню цілісності функціонування.

2.4.3 Вимоги до чисельності персоналу

Система повинна обслуговуватись наступним типом персоналу:

- Системний адміністратор (1 особа);
- Фінансовий аналітик (1-2 особи);
- Бекенд розробник (1 особа);
- Фронтенд розробник (1 особа);
- Оператор системи (2-3 особи);

Персонал, що має необхідність користуватись системою на постійній основі, повинен дотримуватись таких вимог:

- Дотримуватись вимог до зберігання інформація і збереження резервних копій БД;
- Мати досвід у розумінні логіки обробки інформації системою в межах компаній від 3 тижнів;
- Мати права доступу до ресурсів компанії для можливості створення та перегляду задач, що необхідні для виконання актуалізаці в рамках системи;

Користувачем системи може виступати аналітик даних, розробник, тестувальник, персонал підрозділу системного моніторингу та фінансовий відділ.

2.4.4 Вимоги до надійності

Система призначена для використання протягом всього дня. Вхід у систему повинен здійснюватися за допомогою окремої сторінки авторизації, де буде зазначений логін та пароль, що були заздалегідь додані до БД з можливістю проведення перевірки користувача для надання доступу до ресурсу. Облікові дані додаються окремо системним адміністратором як додаткова перевірка безпеки системи. Користувач, що не є працівником фірми, не буде доданий до БД і як результат це унеможливлює отримати доступ до ресурсу, оскільки відсутня сторінка реєстрації.

Система повинна мати комплекс технічних засобів що передбачають наступні можливості:

- Можливість запуску з корпоративних пристроїв персоналу (обмеження ВПН);
- Контроль вхідної інформації з обмеженням інформації що не відноситься до робочого процесу;
- Повідомлення користувачу про помилки при внесення змін що не відносяться до цільового функціоналу застосунку;

- Засобів від збоїв, помилкових дій тощо;

2.4.5 Вимоги з технічної естетики

Інтерфес застосунку не повинен навантажувати користувача функціоналом, що не відноситься до цільового призначення застосунку з можливості концентруватись на цільових функцій виконання, таких як: актуалізація, фіналізація та перегляд останніх змін. Візуальне відображення функціоналу повинно бути однаковим в різних браузерах та на різних засобах відображення інформації.

2.4.6 Вимоги до функцій

Функції мають забезпечити організацію роботи користувачів на основі таких технологій: заповнення БД, формування звіту, зміна даних в БД, створення переопиту статусу до провайдера. Також не менш пріоритетним є зручність використання користувачем інформації за рахунок легкого меню інтерфейсу та підказок. Перелік функцій зі зазначенням вхідної та вихідної інформації наведено у таблиці 2.1.

Таблиця 2.1. Перелік функцій вхідної, вихідної інформації

№	Найменування функції	Вхідна інформація	Вихідна інформація
1	Автентифікація користувачів	Логін, пароль	Статус автентифікації, інформація про користувача
2	Реєстрація запиту актуалізації	Дані транзакції (ідентифікатор, валюта), опис, валюта, посилання на задачу	Статус запиту
3	Оптимізація обробки транзакцій	Параметри транзакцій, час обробки,	Оптимізований час обробки
4	Формування загального звіту запитів на обробку	Запит на оновлення, дані транзакції, дані користувачів	Список задач зі статусом обробки
5	Генерація звітів	Параметри звіту, часовий період	Структурований звіт
6	Створення задачі на оновлення даних для еластіку	Посилання на задачу, список транзакцій	Статус обробки задачі, прощений звіт

2.4.7 Вимоги до системи

Загальносистемне програмне забеспечення повинно забеспечувати якісне виконання функціональних завдань системи. Будь-який пристрій як наприклад ноутбук з мінімальним комплектуванням буде більш достатньо для виконання всіх необхідних функцій системи зі сторони користувача. Додаткові мінімальні умови технічного забезпечення:

Вимоги до апаратного забезпечення:

- Процесор з тактовою частотою: 1Ггц або вище;
- Оперативна пам’ять: 2Гбайт або більше;
- Жорсткий диск: 1Гбайт або більше;
- Монітор: 16 дюймів;
- Миша;
- Клавіатура;

Вимоги до програмного забезпечення:

- Операційна система: будь-яка;
- Браузер: будь-який, що підтримує код JavaScript;

Таким чином ми бачимо, що мінімального технічного і програмного забезпечення будь якого сучасного ноутбука, або комп’ютера буде більш достатньо для можливості працювати з програмою. Єдиним винятком залишається стабільне інтернет з’єднання і встановлений корпоративний ВПН.

2.4 Склад і зміст робіт по створенню системи

Надалі створимо таблицю, в якій опишемо послідовні стадії виконання робіт для реалізації функціоналу (Табл. 2.3).

Таблиця 2.3 Найменування робіт при створенні системи

№	Найменування робіт	Сроки виконання
1	Дослідження роботи відділу та основних процесів.	29.04.2025
2	Дослідження процесу актуалізації та виявлення потреб в її оптимізації.	01.05.2025
3	Формування бачення поточної системи.	05.05.2025
4	Формування бачення оптимізації майбутньої системи та перевірка ідеї на актуальність.	08.05.2025
5	Технічне завдання.	13.05.2025
6	Створення чорнового варіанту для перевірки роботи системи.	15.05.2025
7	Технічний проєкт.	19.05.2025
8	Оформлення документації.	21.05.2025

Для систематизації проєкту по часу також розробимо діаграму Ганта (Рис. 2.1), що дозволяє візуально бачити час, який необхідно витратити на створення повноцінної системи та її додаткової перевірки перед первиною експлуатацією.

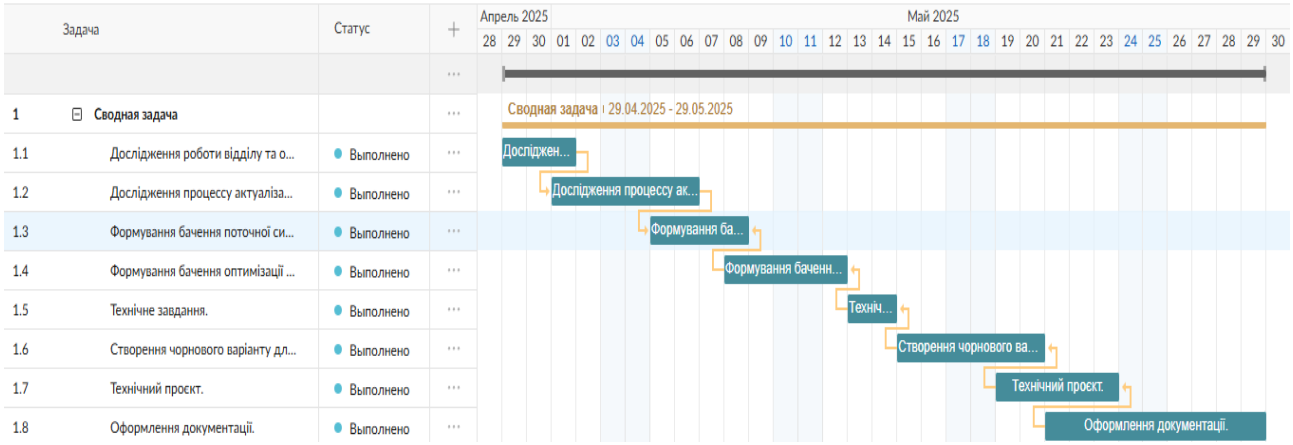

Рисунок 2.1 – Діаграма Ганта створення системи в часі.

РОЗДІЛ 3. РОЗРОБЛЕННЯ СИСТЕМИ АКТУАЛІЗАЦІЇ ТРАНЗАКЦІЙ

Система що розробляється, має основну мету для кінцевого користувача, оптимізувати процес актуалізації та фіналізації статусів по необхідним транзакціям, що за різних причин не синхронізували між собою інформацію взаємодії. Головним чинником такого сервісу є максимальна простота в користувацькому інтерфейсі та мінімально необхідний функціонал, що не перенавантажений зайвими функціями. Це все дає змогу користувачу набагато скоріше опанувати інструментом, та не губити концентрацію під час роботи для мінімізації помилок. Оскільки це критичним фактором при роботі з фінансовими звітами.

3.1 Обґрунтування вибору програмно-технічних засобів розроблення програмного продукту

Головним критерієм вибору засобів розробки є швидкість реалізації прототипу сервісу, та його впровадження до існуючої системи компанії. Також можливість за необхідності реалізувати той же функціонал при виборі іншох програмних засобів або застосувати ті ж засоби для можливості розширення існуючого функціоналу.

Засоби, які були використані для розробки сервісу:

- Мова програмування: Python
- Фреймворк (для бекенд частини): Django
- Система управління бази даних (далі СУБД): PostgreSQL
- Веб-інтерфейс: вреймворк Bootstrap

Основною мовою програмування було обрано Python. Вона має простий і зрозумілий синтаксис, який сприяє швидкості розробки та спрощує підтримку коду різного за об'ємом роботи. Також Python являється основною і єдиною мовою для розробки бекенд частини сайту при використанні фреймворку Django. Неможливо також не зазначити можливість швидко розробляти та тестувати

функціональність, що дозволяє скоротити час виведення готового базового продукту на ринок, або в продове середовище для більш скорішої роботи.

Фреймворк Django є одним із найпопулярніших вреймворків, що використовується у зв’язці з мовою програмування Python. Це високорівневий веб-фреймворк, який має архітектурний шаблон типу Model-View-Template (далі MVT), який дає змогу одразу формувати більш коректну архітектуру системи будь якого розміру вміщуючи в собі інтегровану та більш легку залежність у зв’язці між базою даною, що використовується та веб-інтерфейсом, використовуючи шаблони маршрутизації. Фреймворк має дуже чітку та потужну систему «ORM», яка робить можливим швидку та ефективну взаємодію з базою даних без написання самих SQL-запитів власноруч, додаючи можливість ці запити формувати безпосередньо використовуючи синтаксис Python.

Інші можливості які не можна ігнорувати при виборі цього інструмента є внутрішня система аутентифікації та авторизації, яка вбудована в саму архітектуру фреймворка. Це дозволяє гнучко налаштовувати права авторизовані користувачів, додаючи обмеження які також можна накладати на групу користувачів. Додатковим захистом є вбудовані механізми від поширених атак, такі як CSRF, XSS, SQL-ін’єкції, що є критичним для майже усіх сучасних систем. Також реалізація масштабованості, завдяки архітектурі «MVT» дозволяє з самого початку і до розробки дуже великої системи ефективно обробляти великі обсяги даних, витримувати велику навантаженість під час обробки запитів великого кількості користувачів одночасно.

Для розробки такої системи була обрана СУБД «PostgreSQL», ця система є однією з найпопулярніших серед інших СУБД на сьогодні. Вона повністю підтримує ACID-сумісність, що також гарантує цілісність даних навіть у випадку збоїв системи. Також багатоверсійна система контролю одночасності що також реалізована в цій СУБД дозволяє забезпечити високу продуктивність при одночасному доступі до багатьох користувачів, що є наразі критичним для розробляємої системи. Додатковим важливим чинником вибору є популярність її використання в компаніях, які мають постійну роботу з фінансовими

операціями такі як банкові установи. Оскільки швидкість обробки багатьох запитів до бази даних на запис та зміну дуже добре реалізовано саме для цій СУБД.

Для відображення інформації у веб-інтерфейсі для користувача був обраний фреймворк «Bootstrap», який використовується для створення сайтів і веб-додатків, що містять HTML і CSS шаблони оформлення для форм, кнопок та інших компонентів інтерфейсу, і також має додаткові розширення з використанням мови JavaScript, що забезпечує реалізацію різної анімації цих елементів. Це все дозволяє забезпечити адаптивність веб-інтерфейсу до розмірів екрану. Головною перевагою використання цього інструменту є забезпечення єдиного стилю оформлення всіх елементів інтерфейсу, що підвищує зручність використання системою.

3.2 Проєктування та створення бази даних

Під час проєктування системи було використано існуючу схему обробки і збереження транзацій до бази даних компанії. Оскільки сервіс спроєктовано для оптимізації роботи з цими даними, було прийнято рішення не змінювати структуру, але доповнити її таблицями даних що необхідні для коректної роботи створеного сервісу. Оскільки ми не маємо доступу до даних компанії, для проєктування прототипу ми створимо власну схему тієї частини, яка відповідає напряму вимогам нашої системи для збереження її загальної логіки обробки даних. Ці таблиці є копією самої бази даних, оскільки в самої компанії для кожного сервісу є своя БД, які між собою не пов’язані будь якими зв’язками. І зміни та синхронізація в таблицях цих БД виконується за допомогою інших сервісів що реалізовані на різних мовах програмування.

Таблиці які будуть використовуватись для запису інформації, та фіксації змін будуть створені за допомогою фреймворку Django та його системи «ORM». Це дозволить більш якісно контролювати відношення між таблицями, та в разі необхідності змінювати параметри полів як під час розробки, так і під час використання, для можливості доналаштовувати БД.

В даному випадку основні сутності системи будуть наступні:

- Користувач (auth_user):
  Особа, яка авторизується в системі та виконує функції до яких має доступ.
- Транзакція (service_transaction)
  Сутність що зберігає основну інформацію по фінансовим операціям.
- Інтеграція (service_integration)
  Зберігає інформацію по операції щодо інтеграції, яка використовується для відправки цієї інформації до провайдера для подальшої обробки.
- Еластік (service_transactionelastic)
  Сутність, що є копією таблиці Транзакція, але відокремлена оскільки це інший сервіс в компанії, який має запис своїх даних для співробітників, які не мають прямого доступу до бази даних.
- Оновлення (service_update)
  Сутність, що зберігає дані задачі на оновлення інформації між таблицями Транзакція та Еластік.
- Актуалізація (service_actualization)
  Сутність, що зберігає дані задачі на оновлення інформації, яка стосується запиту на переопит статусу для іншого сервісу.
- Фіналізація (service_finalization)
  Сутність, що зберігає дані задачі на оновлення інформації, яка стосується запиту на зміну статусу транзакції.

Далі наведено фізичну модель (Рис. 3.1) цих сутностей у СУБД, яка імітує поведінку і відношення між цими таблицями в компанії. Таблиці «Транзакція», «Інтеграція», «Еластік», не мають між собою відношень, оскільки імтують реальну взаємодію між собою в продовому середовищі компанії, яка використовує свої внутрішні сервіси для зміни інформації для цих таблиць.

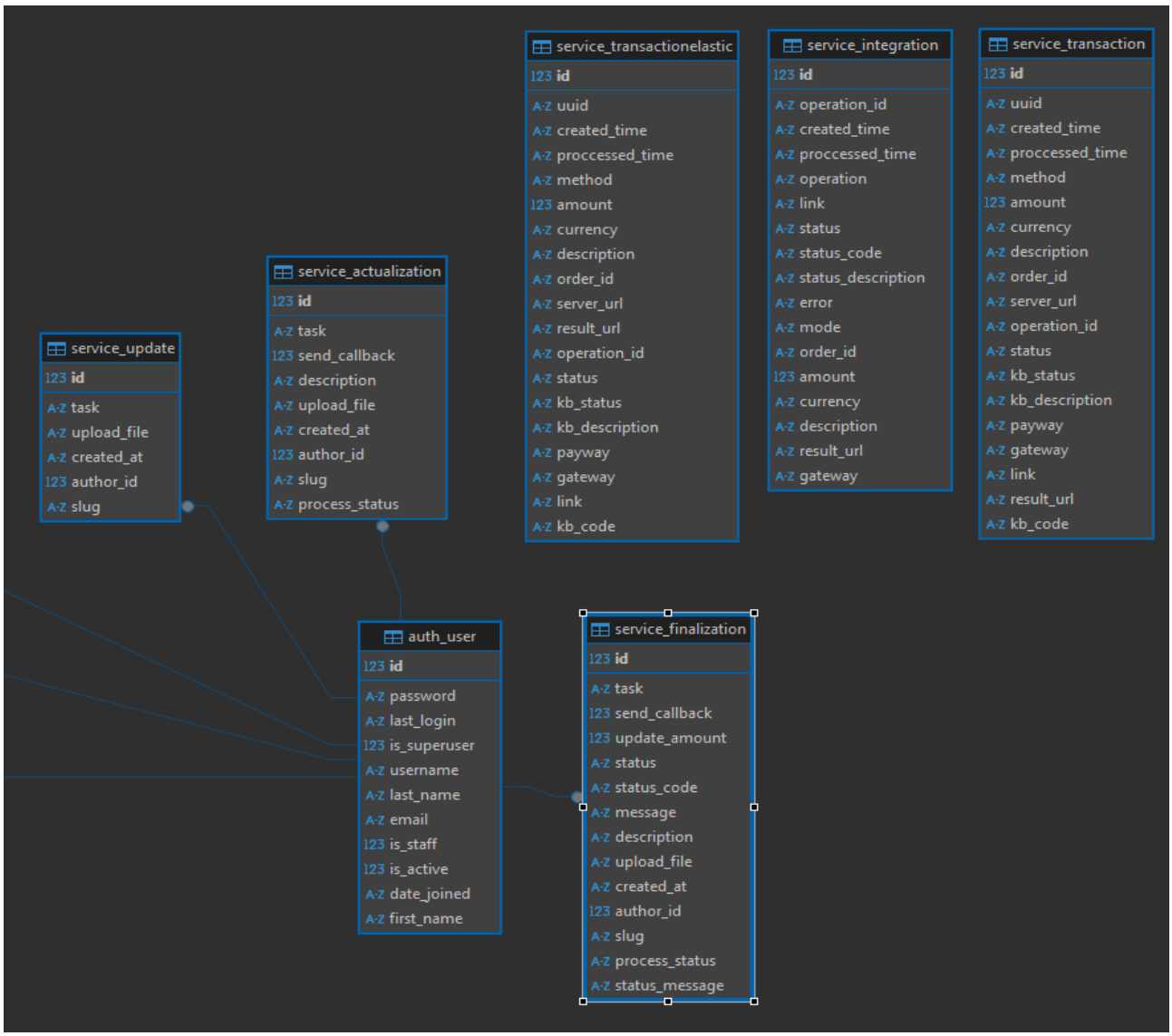

Рисунок 3.1 – Фізична модель основних таблиць сервісу для бази даних

Таблиці «Оновлення», «Актуалізація», «Фіналізація» мають відношення один-до-багатьох з таблицею «Користувач» оскільки користувач сервісу, має можливість створювати багато заявок на зміну даних по операціям для зміни опису статусу транзакцій в відповідних таблицях.

Загальна база даних має додаткові сутності, які не мають прямого відношення до системи, але автоматично створені системою «ORM», яка додатково забезпечує сервіс можливістю формувати групи користувачів, логувати активні сесії авторизованих користувачів. Кінцевий результат створення всіх таблиць наведено далі (Рис. 3.2).

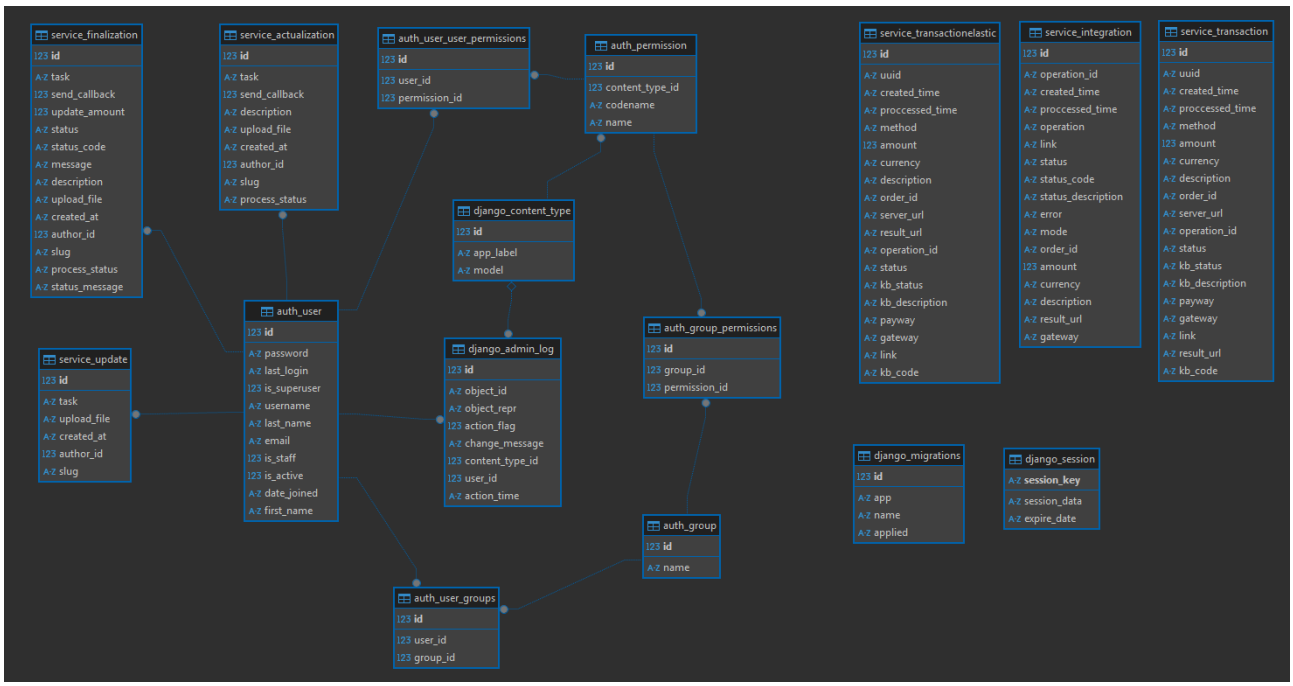

Рисунок 3.2 – Загальна модель сутностей бази даних для сервісу

Перед реалізацією функцій, які буде виконувати системи, необхідно створити за допомогою «ORM» структуру для таблиць бази даних, що будуть використовуватись для запису дій користувача відповідно до задач, які він буде виконувати користуючись інструментом.

```
class Actualization(models.Model):
    PROCESS_CHOICES = (···

    author = models.ForeignKey(User, on_delete=models.CASCADE)
    task = models.URLField(blank=False)
    send_callback = models.BooleanField(blank=True)
    description = models.TextField(blank=True)
    upload_file = models.FileField(upload_to='uploads/', blank=False)
    created_at = models.DateTimeField(auto_now_add=True)
    slug = models.SlugField(max_length=255, unique=True, blank=True)
    process_status = models.CharField(max_length=15, blank=True, choices=PROCESS_CHOICES)
```

Рисунок 3.3 – Створення структури таблиці «Актуалізація»

Як видно на (Рис. 3.3) за допомогою «ORM» інструменту стає можливим створювати таблиці з різною структурою, полями, та правилами відношення до інших таблиць за допомогою мови програмування Python. Це пришвидчує час розробки та надає можливість під час вирішення змінити логіку, також змінити код запису інформації до полів БД. Ця функція реалізується за допомогою

«міграцій» фреймворка – функціонал що досить поверхнево, але дуже ефективно копіює логіку контролю версій тільки по відношенню до змін конфігурації БД. Додатково серед правил можливо також дописувати іншу логіку по відношенню до таблиці про яку ми говорили раніше.

```
class Meta:
    ordering = ('-created_at',)

def save(self, *args, **kwargs):
    if not self.slug:
        self.slug = generate_unique_slug()
        while Actualization.objects.filter(slug=self.slug).exists():
            self.slug = generate_unique_slug()
    super().save(*args, **kwargs)

def get_absolute_url(self):
    return reverse('service:actualization_detail', kwargs={'slug': self.slug})

def __str__(self):
    return self.slug
```

Рисунок 3.4 – Додатковий функціонал обробки сутності таблиці «Актуалізація»

На (Рис. 3.4) ми бачимо можливість описувати функціонал до кожної сутності що зберігається та відноситься до цієї таблиці. Так, наприклад, на початку ми описуємо порядок автоматичного сортування всіх даних за замовчуванням, що будуть записані та знайдені піч час виклику команди зчитування. Далі прописуємо логіку зберігання, що виконується під час зберігання нової сутності прямо перед цією подією. Ця логіка відповідає за створення унікального посилання, що ми зможемо відобразити або в веб-інтерфейсі, або під час звичайного пошуку як ім’я цього об’єкта. Функція «get_absolute_url» реалізує можливість робити пошук, та напряму посилатись на знайдений об’єкт, що записаний в базі даних. Та остання функція дає можливість робити пряме посилання до об’єкту на зчитування не по його унікальному ідентифікатору, а по унікальній назві, що формується зазначеною раніше функцією «save()», що виконується під час збереження новостворених даних до таблиці.

Подібним чином заздалегідь ми створили таку саму структуру для інших таблиць, які потрібні нам для простої фіксації дій щодо внесення змін до інших таблиць користувачами системи. До цих таблиць належать: «Актуалізація», «Фіналізація», «Оновлення», повний код яких надано в додатку Г.1 - Г.3 відповідно. Дані що там збережені не відносяться до таблиць де зберігається інформація про транзакції, тут фіксується тільки зміна, яка була виконана тим чи іншим користувачем, що має доступ до системи.

Після того як будуть створені таблиці що фіксуються зміни зроблені користувачем, нам також додатково потрібно створити таблиці, що імітують роботу продової бази даних компанії. Треба зазначити, що оскільки зі сторони компанії ці таблиці мають таку саму логіку і структуру, наша спроба створити копію в межах таблиць не зіпсує логіку обробки транзакцій. Як зі сторони компанії, так і з нашої сторони таблиці «Транзакція», «Еластік», «Інтеграція» не мають між собою ніяких зв’язків, як описувалось раніше і в такому випадку імітують реальну взаємодію процесу між собою. Оскільки зміну даних для таблиць виконують інші внутрішні сервіси компанії.

Для прототипу цієї системи достатньо було обійтись без цих таблиць. Але ми все таки їх створимо щоб мати можливість наглядної демонстрації зміни, які виконує наш сервіс.

3.3 Реалізація функцій системи

Перед тим як створювати повноцінний фунціонал сервісу, необхідно створити заготовку у вигляді архітектури проєкту. Для цього ми завантажуємо фреймворк Django через пакет насталювання «pip», що встановлюється разом з мовою програмування Python. Після завантаження фреймворку у вигляді бібліотеки, за допомогою команди «django-admin startaproject» через термінал, в папці проєкту будуть додані основна папка з файлами проєкту та файл, що в майбутньому дозволить створювати інші додаткові сервіси та використовувати команди для налаштування різних видів конфігурації проєкту.

Розробка кодової бази будемо робити через легкий та зручний редактор коду з підствіткою синтаксиса, та можливості автодоповнення коду під час його написання за допомогою програми «Windows Visual Code». Структура проєкту зображено на (Рис. 3.5).

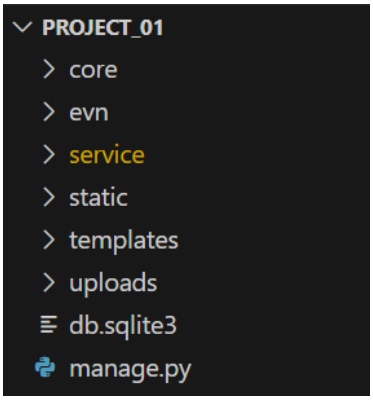

Рисунок 3.5 – Загальна структура папок проєкту для розміщення коду.

Після створення проєкту, одразу додатково додамо до загальної структури папку з файлами, де будемо додавати нову логіку. Вся логіка обробки буде прописуватись у файлі «views.py», повний лістинг коду з файлу наведений в додатку Г. Головна функція цього файлу полягає в обробці URL-запиту що відправляє користувач з певними даними через форму та подальшої обробки інформації.

Оскільки сервіс створено для внутрішнього користування, ми не будемо реалізовувати реєстрацію нових користувачів, під час роботи для нового співробітника адміністратор ресурсу на запит керівника відділу буде самостійно додавати нового юзера до системи, після його створення, логін та пароль будуть надсилатись співробітнику, це додатковий захист системи. Таку можливість можна зробити через внутрішню адмін-панель, яка вже реалізована у фреймворку Django. Вона створюється одразу під час ініціалізації проєкту. Це також прискорює розробку та мінімізує необхідність писати логіку панелі адміністратора, оскільки існуюча реалізація (Рис. 3.6) повністю відповідає необхідними вимогам до адміністрування групами та окремими записами користувачів.

При редіректі користувача на ресурс реалізуємо логіку, за допомогою якої неможливо буде потрапити на головну сторінку, поки авторизація не буде успішною, для цього потрібно реалізувати сторінку для входу в сервіс (Рис. 3.7). Для цього створимо HTML-форму, яка буде перевіряти і обробляти вхідні дані, такі як логін і пароль, що були створені адміністратором з самого початку.

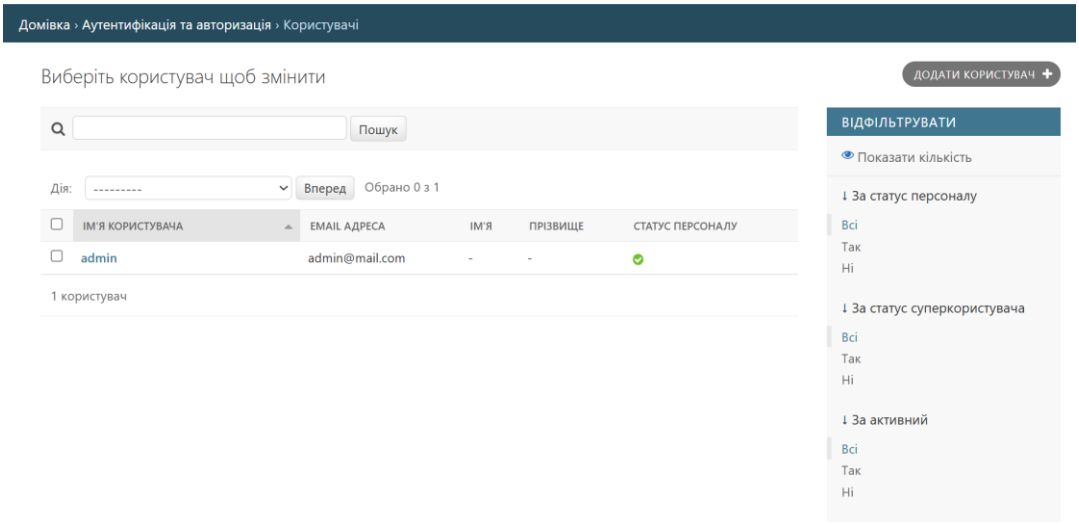

Рисунок 3.6 – Панель адміністратора сервісу для редагування нових та існуючих користувачів.

Перевірку робить проміжний рівень фреймворку через змінні «шаблонізатору». Цей інструмент додатково дає змогу вводити зміні коду в HTML-шаблон, та відправляти дані POST-запиту форми до «представлення», що описується логікою у файлі views.py.

Всі змінні шаблонізатору у формах позначаються як «{% назва_зміної %}». Це один з основних елементів для роботи з формою і передачі інформації для обробки якими ми будемо користуватись і надалі.

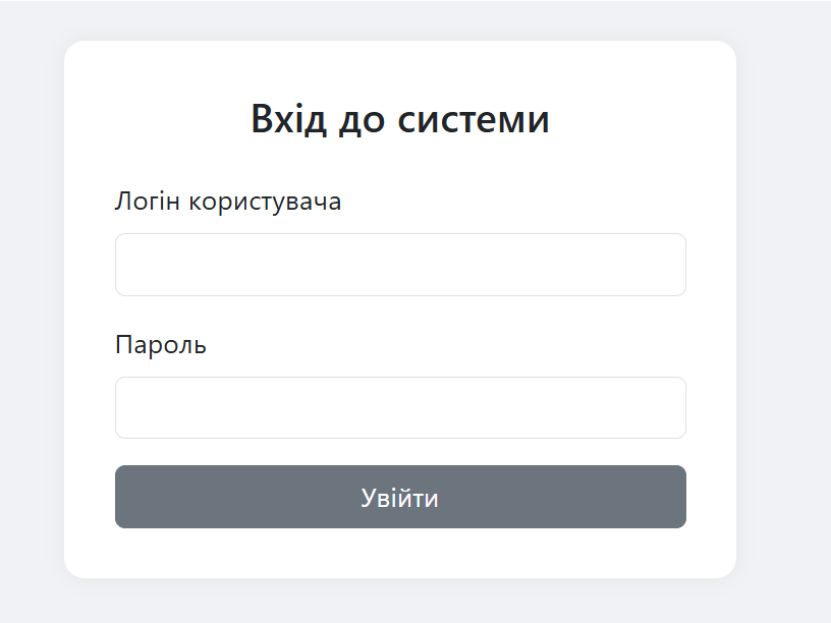

Рисунок 3.7 – Форма для авторизація користувача для доступу на сервіс

Для створення головної сторінки (Рис. 3.8) також використовуємо HTML-шаблон з підключенням додаткової бібліотеки «Bootstrap». Як було зазначено раніше (розділ 3.1), вона дозволяє використовувати існуючі стилі та javaScript-код для можливості реалізовувати однакий для всіх шаблонів стиль і базовий набір анімації, такий як відображення при певних діях користувача додаткові параметри з певими діями.

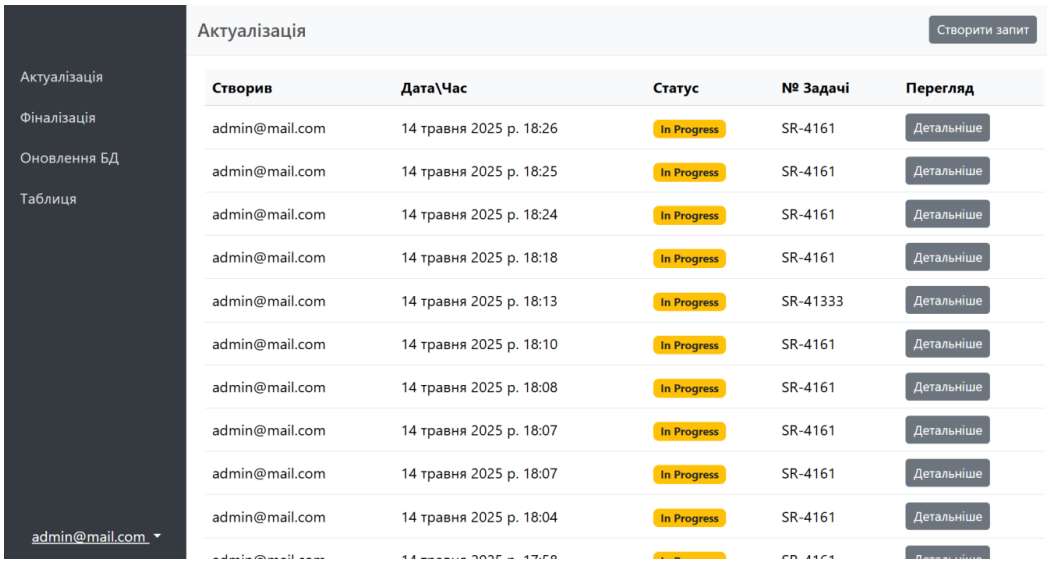

Рисунок 3.8 – Головна сторінка сервісу після авторизації користувача

Оскільки головним чинником є простота інтерфейсу, та його базова функціональність щодо вимогам технічного завдання, ми не будемо створювати надлишкові елементи інтерфейсу. Це дозволить більш ефективно користуватись інструментом не відволікаючись на функціонал, що не має відношення до процесу роботи. На головній сторінці «Актуалізація» зробимо відображення всіх задач, які були створені різними користувачами для короткого перегляду та в подальшому, можливості ввести звітність щодо того, хто мав права на втручання в систему для зміни інформації. Для сторінок «Актуалізація» та «Фіналізація» шаблон є аналогічним до дизайну, обидві сторінки мають звіт який відображає наступні дані:

- Хто робив задачу
- Коли задача була створена
- Який статус виконання задачі
- Номер запиту в сервісі Jira, для якої було необхідно створити задачу
- Можливість переглянути

Іншим відображенням подібного шаблону була зроблена сторінка «Оновлення», яка не містить статус обробки задачі на оновлення платіжних операцій, та додаткового перегляду, оскільки оновлення статусу можна переглянути додатково або окремо в «Кібані», що є зовнішнім продуктом, яким частіше користуються співробітники компанії, або напряму в таблиці в розділі «Таблиця», що також являє собою копію цих даних.

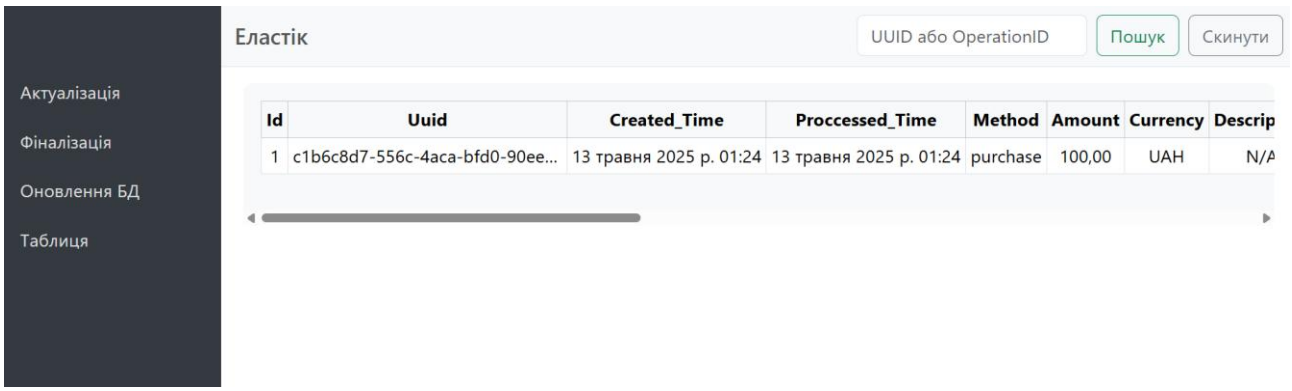

Рисунок 3.9 – Сторінка відображення таблиці з пошуком

Додатково реалізовано пошук інформації по uuid або operation_id, полям таблиці «Еластік» (Рис. 3.9). Повний лістинг коду, що реалізує пошук по заданим параметрам для таблиці наведено в додатку Д.

Основним функціоналом є реалізація функції що дозволяє створити задачу на зміну статусів транзакцій і валідація форми для коректного завантаження файлу операцій зі сторінки по створенню задачі що додатково зображено на (Рис. 3.10).

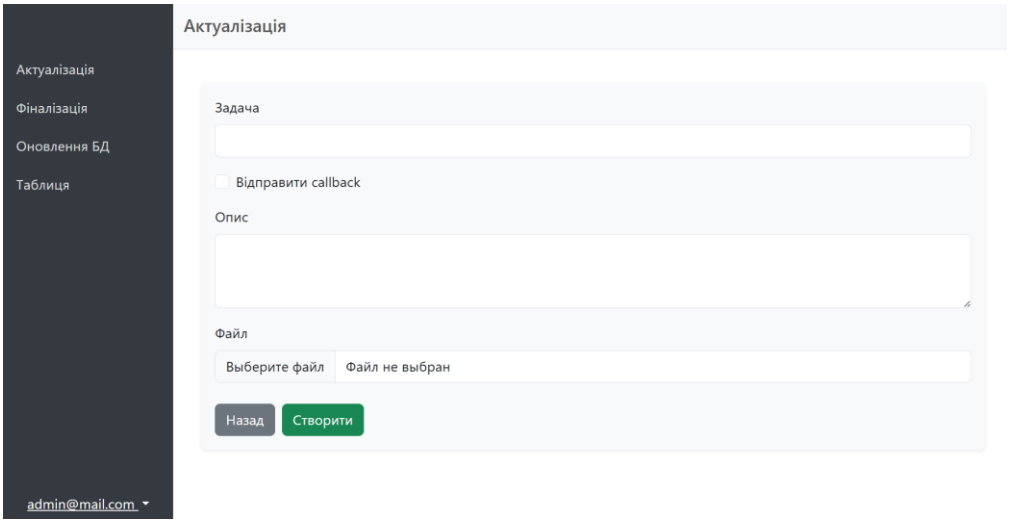

Рисунок 3.10 – Сторінка створення задачі на актуалізацію.

Валідація полів перевіряється окремим функціоналом, що прописано в файлі forms.py, де ми можемо попередньо описувати як саме буде відображатись поле, та як саме буде відбуватись валідація записаних даних перед тим як користувач ініціює POST-запит. Нижще наведено код, що робить перевірку та показ полів необхідних для заповнення.

```
class ActualizationForm(forms.ModelForm):
class Meta:
model = Actualization
fields = ['task', 'send_callback', 'description', 'upload_file']
widgets = {
'task': forms.TextInput(attrs={'class': 'form-control', 'placeholder':
'https://tranzzo.atlassian.net/browse/'}),
'send_callback': forms.CheckboxInput(attrs={'class': 'form-check-input'}),
'description': forms.Textarea(attrs={'class': 'form-control'}),
'upload_file': forms.ClearableFileInput(attrs={'class': 'form-control-file'}),
}

def clean_upload_file(self):
```

```
file = self.cleaned_data['upload_file']
if not file.name.endswith('.txt'):
raise forms.ValidationError("Only .txt files are allowed.")
if file.content_type != 'text/plain':
raise forms.ValidationError("Uploaded file must be a plain text file.")
return file
```

Подібна валідація також реалізована для сторінки створення задачі і для форми фіналізації та оновлення таблиці.

Одним із головних реалізацій функціоналу щодо задачі є також перевірки файлу з ідентифікаторами що завантажується користувачем. Такий підхід було обрано оскільки для обробки у файлі, цих записів записів може бути дуже багато. У файлі ідентифікатори платежів повинні бути встановлені з нового рядка та без коми, якщо кома є – необхідно додати суму, яку також необхідно буде змінити та оновити замінюючи неактуальну суму. Код що перевіряє коректність файлу:

```
def processing_file(file):
    uuid_amount_map = {}
    with open(file, 'r', encoding='utf-8') as f:
        for line in f:
            line = line.strip()
            if not line:
                continue
            parts = line.split(',')
            uuid = parts[0].strip()
            if len(parts) == 2:
                try:
                    amount = int(parts[1].strip())
                except ValueError:
                    amount = None
            else:
                amount = None
            uuid_amount_map[uuid] = amount
```

Надалі після перевірки кожна функція, яка відноситься до розділу, перевіряє самостійно по ідентифікатору наявність транзакції та робить оновлення статусу, для можливості активувати сервіс компанії для переопиту статусу зі сторони провайдера під час продового виконання цього застосунку. Якщо, казати щодо розробки та тестування, нам необхідно тільки змінити статус в таблиці «Інтерграція».

Для актуалізації та фіналізації після зчитування з файлу даних по ідентифікатору транзакції виконуєтьс пошук в таблиці «Транзакція», якщо збіг інформації дає позитивний результат, ми беремо параметр operation_id за яким також робимо пошук транзакції в таблиці «Інтеграція». За знайденими ідентифікаторами потрібно зробити додаткову перевірку відповідності параметрів: status, gateway, link, operation_id. Це необхідно оскільки, таблиці імітують реальні бази даних компанії в яких також система робить перевірку саме за цими параметрами на відповідність. У випадку якщо ці параметри не співпадають у будь-якому випадку неохідно робити додаткову задачу до відділу тестувальників для перевірки, чому система не записала саме коректні дані. Якщо інформація збігається – наступний крок це зміна статусу з її описом у таблицях. Код що реалізує перевірку та валідацію для польшої зміни статусу.

```
def actualization_transaction(transaction):
    for uuid, _ in transaction.items():
        transaction = Transaction.objects.filter(uuid=uuid).values('gateway', 'link',
'operation_id')
        tr_fields = transaction.first()
        if transaction.exists():
            operation =
Integration.objects.filter(operation_id=tr_fields['operation_id']).values('gatewa
y', 'link', 'operation_id')
            op_fields = operation.first()
            if op_fields['gateway'] == tr_fields['gateway'] and op_fields['link'] ==
tr_fields['link'] and str(op_fields['operation_id']) == tr_fields['operation_id']:
                operation.update(status='Pending', status_code='22001',
status_description='Transaction is pending')
                transaction.update(status='Pending')
```

В такому випадку статус змінюється на проміжний «Pending» що позначає операцію як не фінальну. Інший сервіс в такому випадку має тригер, який повідомляє сервіс в необхідності зробити додатковий переопит зі сторони провайдера платіжних послуг додаткову перевірку.

Пі час виконання актуалізації або фіналізації, нам не потрібно робити зміну статусу на проміжний в інтеграції для створення задачі переопиту іншого сервісу. В нашому випадку необхідно просто змінювати статус в таблицях «Інтеграція» та «Транзакція» і окрім статусу також інші параметри, які слугують

перевіркою під час їх оновлення для інших операцій. В такому випадку нам необхідно зробити перевірку на співпадіння та у разі знайдення співпадіння в різних таблицях дозволити робити зміну на необхідні параметри. Код що реалізує цю логіку.

```
def finalization_transaction(transaction, finalization):
    task = Finalization.objects.filter(slug=finalization).first()
    print(transaction)
    for uuid, amount in transaction.items():
        print(uuid, amount)
        transaction = Transaction.objects.filter(uuid=uuid).values('gateway', 'link', 'operation_id')
        tr_fields = transaction.first()
        if transaction.exists():
            operation = Integration.objects.filter(operation_id=tr_fields['operation_id']).values('gateway', 'link', 'operation_id')
            op_fields = operation.first()
            if op_fields['gateway'] == tr_fields['gateway'] and op_fields['link'] == tr_fields['link'] and str(op_fields['operation_id']) == tr_fields['operation_id']:
                operation.update(status=task.status, status_code=task.status_code, status_description=task.status_message, error=task.message, amount=amount)
                transaction.update(status='Processed', kb_status=task.status, kb_code=task.status_code, kb_description=task.status_message, amount=amount)

Finalization.objects.filter(slug=finalization).update(process_status=Finalization.PROCESS_CHOICES[0][0])
```

3.4 Інструкція користувача

Під час першого користування, користувачу необхідно пройти авторизацію на сервісі за допомогою форми для вводу логіну та пароля (додаток Д Рис. Д.1). Після входу відбувається редірект на головну сторінку, де відображається перший пункт, який відноситься до актуалізації (додаток Д Рис. Д.3).

Надалі користувачу надається можливість обрати один з трьох варіантів роботи з транзакціями, згідно задачі:

- Актуалізація транзакцій (додаток Д Рис. Д.3)
- Фіналізація транзакцій (додаток Д Рис. Д.4)

- Оновлення БД (додаток Д Рис. Д.5)

Кожна сторінка цього функціоналу схожа між собою можливістю створити задачу, де необхідно вказати необхідну інформацію та надати список транзакцій, для яких необхідно робити зміни (додаток Д, Рис. Д.7). Після створення задачі, користувача автоматично редіректить на головну сторінку процесу, для якого створювалась задача, таким чином можливо одразу побачити статус виконання задачі.

У разі необхідності також є можливість перевірити додаткові дані обробки, а також побачити скільки зі списку транзакцій було оброблено (додаток Д, Рис. Д.9).

Система має внутрішні реалізації перевірки коректності вводу даних, та формат файлу в якому було завантажено набір транзакцій. У випадку якщо буде спроба завантажити файл формату «pdf», форма не дозволить цього зробити, оскільки доступний тільки формат «txt».

Після роботи в сервісі користувачу необхідно натиснути на свою пошту, яка знаходиться внизу зліва, після цього буде показана кнопка виходу з сервісу.

ВИСНОВКИ

В цій дипломній роботі було реалізовано прототип системи, що дає змогу оптимізувати процес актуалізації та фіналізації платіжних операцій, для яких необхідно перевіряти співвідношення статусів та інших критерії за якими операція повинна виконуватись в системі для коректного відображення кінцевому користувачу. Цей процес полегшує та спрощує систему оновлення ручної роботи коректних транзакцій у разі не співпадіння їх суми та статусу в різних таблицях баз даних.

Впровадження цієї системи дасть змогу полегшити та пришвидшити процес обробки без необхідності створювати додаткові задачі до підрозділу тестування, знімаючи велику частину додаткової роботи з інших працівників. Підключення інших відділів до поточного процесу тепер буде відбуватись у разі необхідності перевірки некоректної роботи процесу і сервісу. В свою чергу це дає можливість сконцентруватись на покращенні обробки запитів працівників для інших задач.

Сервіс наразі є легким за об’ємом та має мінімалістичний дизайн, який не буде відволікати непотрібним функціоналом співробітників від людського фактору в можливості робити помилку. Також більшість . Це реалізується завдяки мінімально необхідним функціоналом, та тільки необхідною інформацію що відображається при створені задачі, що також полегшує додаткову перевірку та контроль по змінам у разі необхідності додаткової перевірки останніх дій.

Сервіс за необхідності можливо покращити та вдосконалити іншими метриками в майбутньому при необхідності, це дозволить мінімізувати деякі витрати у використанні інших сервісів для відслідковування дій. Або також можливо позбавити графічного інтерфейсу імплементуючи цей функціонал в інший сервіс де ці зміни будуть автоматично відбуватись на події дій користувача і інших сервісах завдяки своєму невеликому об’єму кодової бази та швидкості реалізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Основна інформація про компанію Tranzzo. URL: https://tranzzo.com/uk-ua/about (дата звернення: 12.05.2025).
2. Tranzzo. API Документація Tranzzo. URL: https://docs.tranzzo.com/uk/docs/integration/options/ (дата звернення: 12.05.2025).
3. AllFusion Process Modeler. Моделювання бізнес-процесів засобом AllFusion Process Modeler. URL: https://fmab.khadi.kharkov.ua/fileadmin/F-FUB/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%96%D0%BD%D0%BD%D1%8F_%D1%82%D0%B0_%D0%B0%D0%B4%D0%BC%D1%96%D0%BD%D1%96%D1%81%D1%82%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F/5_Reinginir_L1.pdf (дата звернення: 13.05.2025)
4. Openwaygroup. Сайт компанії Openwaygroup. URL: https://openwaygroup.com/ (дата звернення: 14.05.2025).
5. Corezoid. Офіційний сайт компані «Corezoid». URL: https://corezoid.com/ / (дата звернення: 14.05.2025)
6. Creatio. Офіційний сайт компані «Creatio» URL: https://www.creatio.com/ (дата звернення: 14.05.2025)
7. Ковальчук Т.Т., Лук'янов В.С. "Сучасні платіжні системи". Київ: Знання, 2021. 334 с.
8. Ресурс створення діаграми Ганта URL: https://app.ganttpro.com/#/project/1746826168593/gantt (дата звернення: 15.05.2025)
9. Django. Документація для роботи з фреймворком Django. URL: https://docs.djangoproject.com/ (дата звернення: 17.05.2025)
10. Django. Опис «ORM» взаємодії створення бази даних. URL: https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping (дата звернення: 17.05.2025)

11. Морозов А.Ю. "Проектування інформаційних систем фінансових установ". Харків: Фоліо, 2022. (дата звернення: 16.05.2025)
12. Писаренко Н.В. "Автоматизація фінансових процесів". Львів: Новий Світ, 2023. 256 с. (дата звернення: 16.05.2025)
13. Дейнека М.М. "Оптимізація бізнес-процесів у фінансовому секторі". Київ: Ліра-К, 2023. 296 с. (дата звернення: 18.05.2025)
14. Bootstrap. Документація користування фреймворком «Bootstrap». URL: https://getbootstrap.com/docs/5.3 (дата звернення: 18.05.2025)
15. Django. Інтеграція та налаштування зовнішніх бібліотек для фреймворку «Django» URL: https://www.w3schools.com/django (дата звернення: 17.05.2025)
16. King B. "Bank 4.0: Banking Everywhere, Never at a Bank". Wiley, 2019. 352 p. (дата звернення: 13.05.2025)
17. Скіннер К. "Цифровий банкінг: революція в обслуговуванні та платежах". Пер. з англ. Київ: Самміт-Книга, 2021. 320 с. (дата звернення: 18.05.2025)
18. W3school. Налаштування шаблонів фрейморку Django. URL: https://www.w3schools.com/django/django_templates.php (дата звернення 18.05.2025)
19. Повний посібник з розробки фінтех-програмного забезпечення 2024. URL: https://youteam.io/blog/complete-guide-to-fintech-software-development/ (дата звернення: 12.05.2025)
20. DashDevs. Роботизована автоматизація процесів у фінтех та банківському секторі. URL: https://dashdevs.com/blog/robotic-process-automation-is-to-become-a-growth-strategy-for-the-financial-industry/ (дата звернення: 15.05.2025)
21. Workato. Правильний підхід до автоматизації процесів для фінтех. URL: https://www.workato.com/the-connector/process-automation-for-fintech/ (дата звернення: 18.05.2025)

22. Kleppmann, M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017. 616 p. (дата звернення: 19.05.2025)
23. Chishti, S., Barberis, J. The FINTECH Book: The Financial Technology Handbook for Investors, Entrepreneurs and Visionaries. Wiley, 2016. 304 p. (дата звернення: 17.05.2025)
24. Національний банк України. Розділ "Платіжні системи та інноваційний розвиток". URL: https://bank.gov.ua/ua/promo/payments-innovations (дата звернення: 14.05.2025)
25. ISO 20022. Official ISO 20022 Website. URL: https://www.iso20022.org (дата звернення: 20.05.2025)

ДОДАТКИ

Додаток А. Функціональна схема

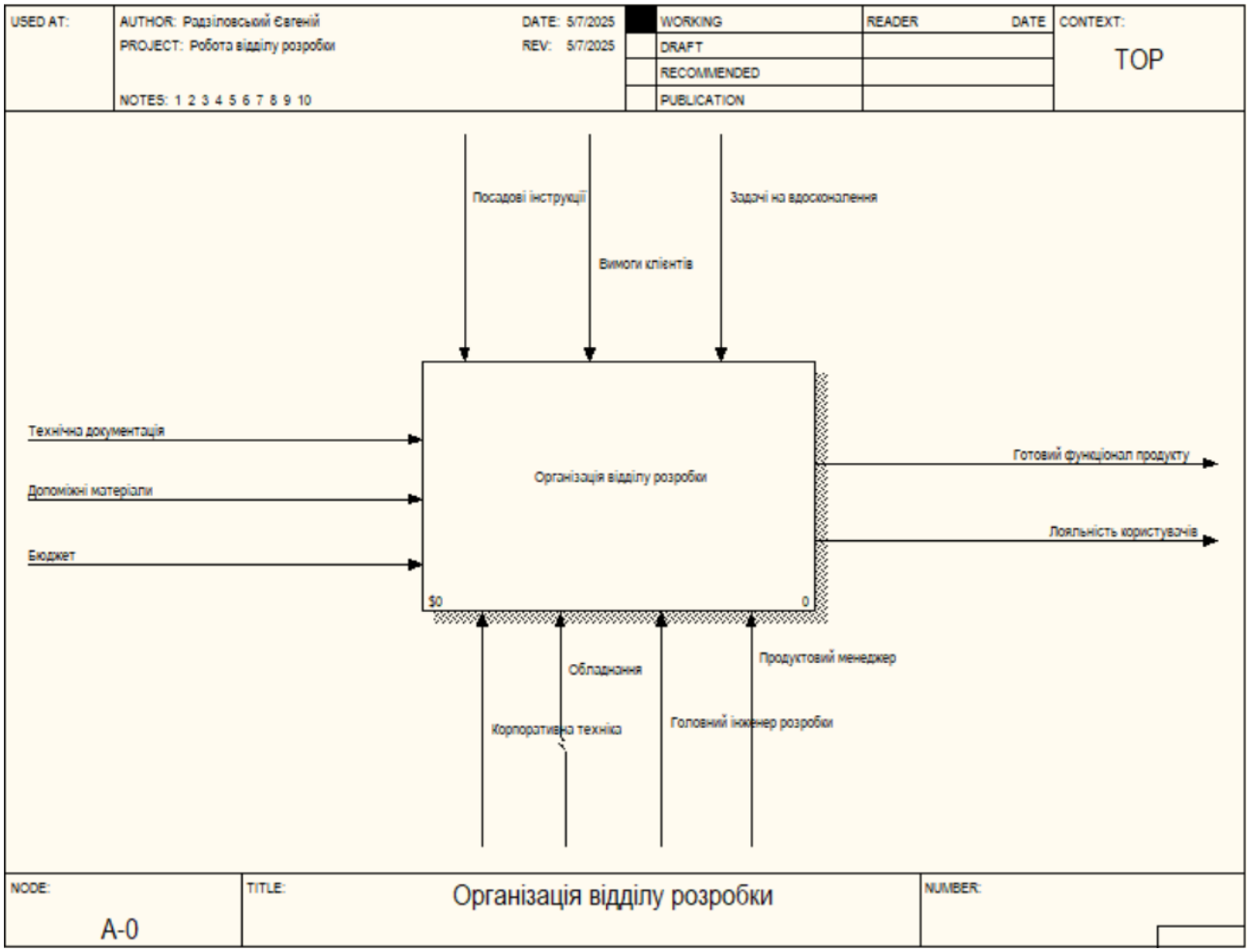

Рисунок А.1 – Модель «AS IS». Контестна діаграма верхнього рівня

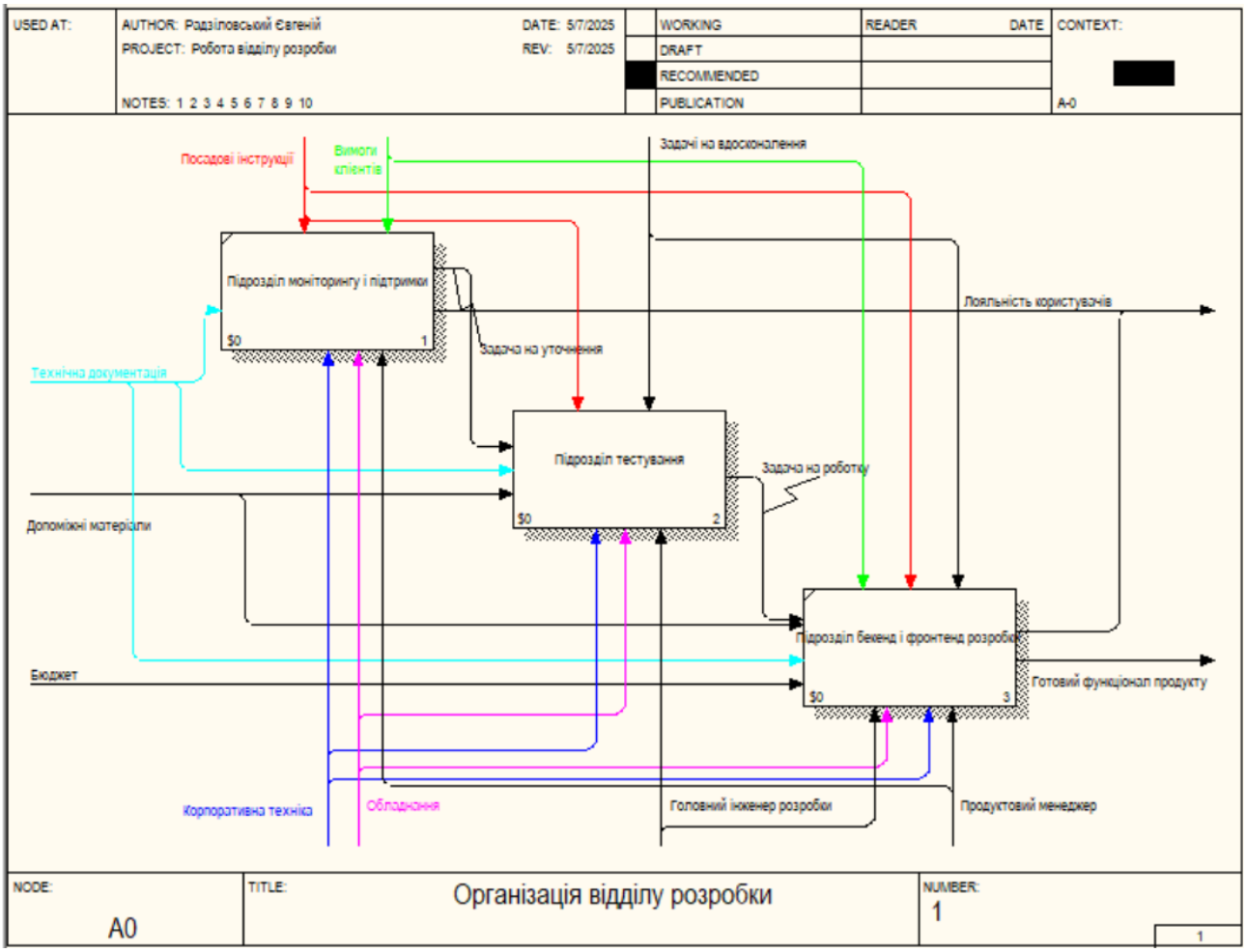

Рисунок А.2 – Модель «AS IS». Загальна схема організації роботи відділу розробки.

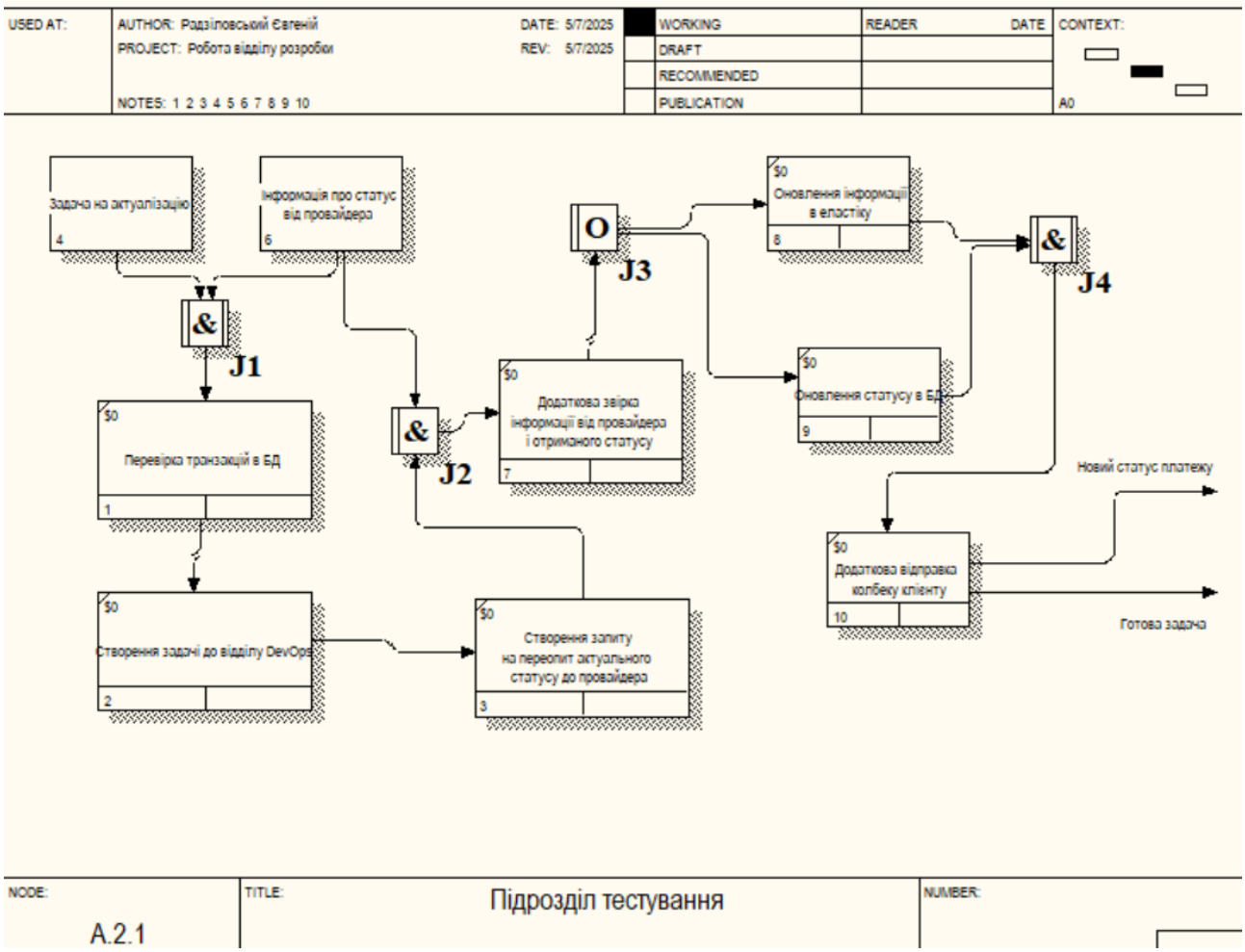

Рисунок А.3 – Модель «AS IS». Процесс актуалізації транзакції.

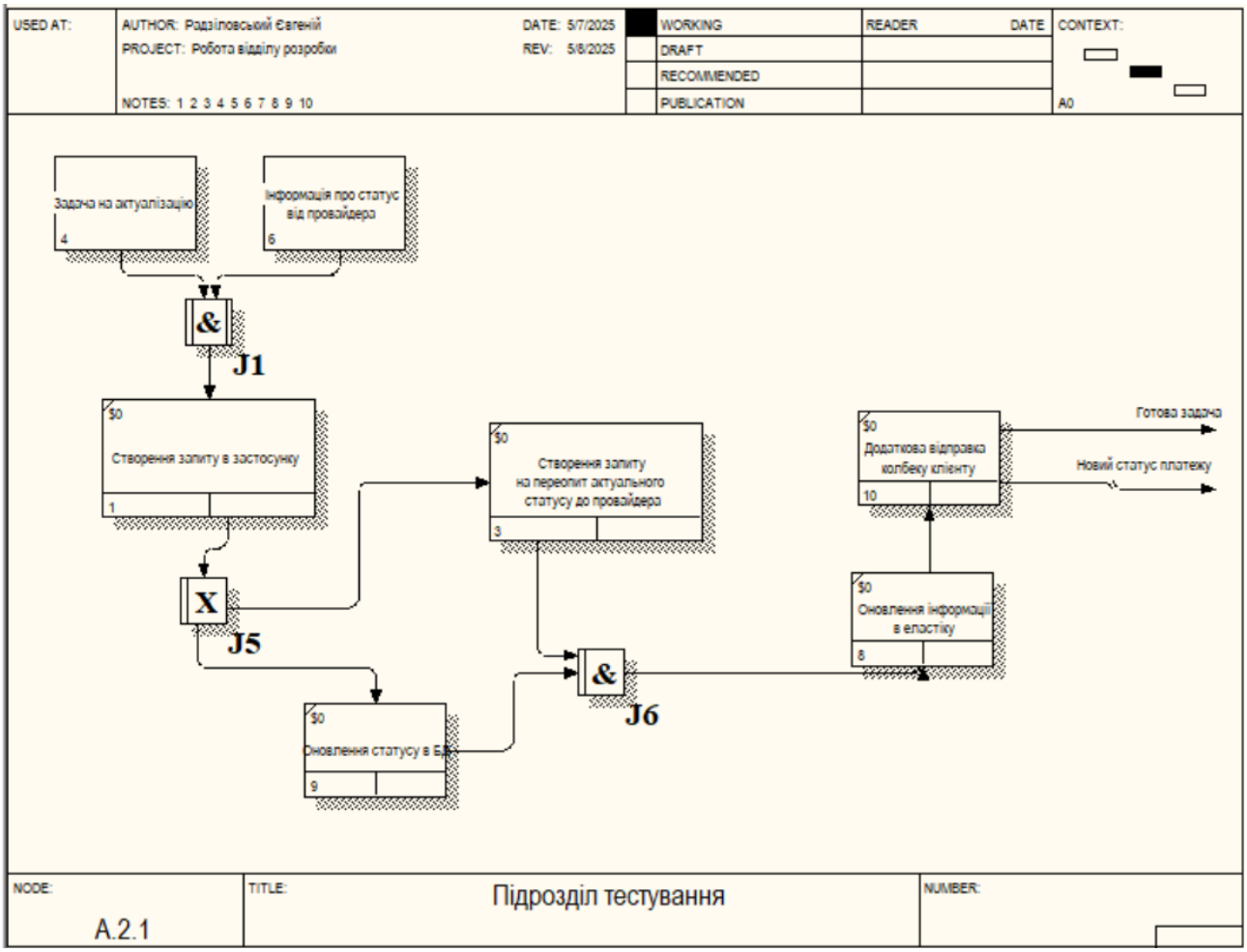

Рисунок А.4 – Модель «TO BE». Процесс актуалізації транзакції.

Додаток Б. Діаграма розподілу часу

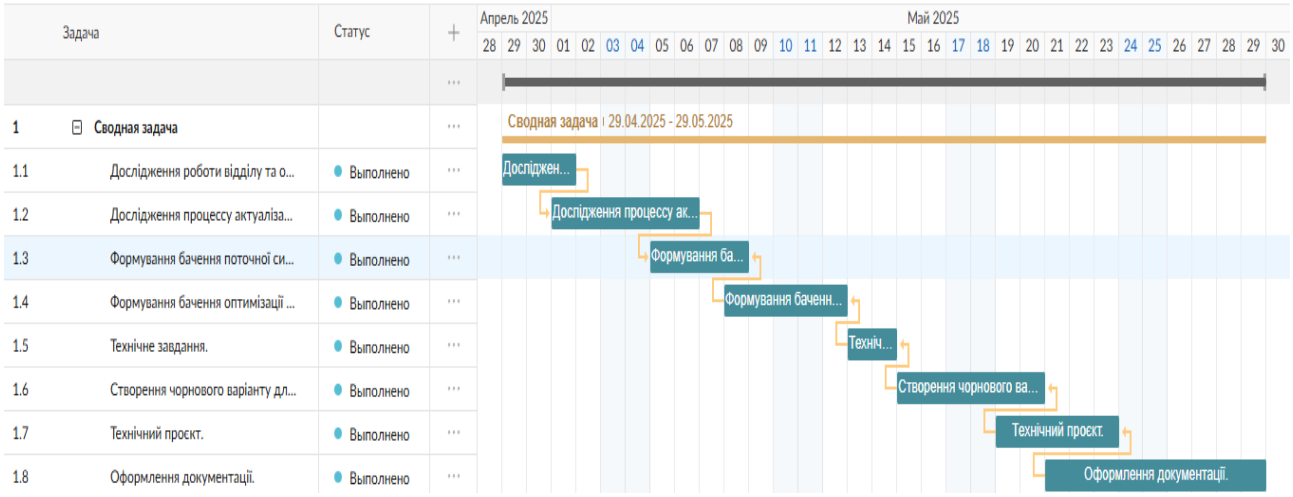

Рисунок Б.1 – Діаграма Ганта. Розподіл часу на створення системи.

Додаток В. Схема бази даних

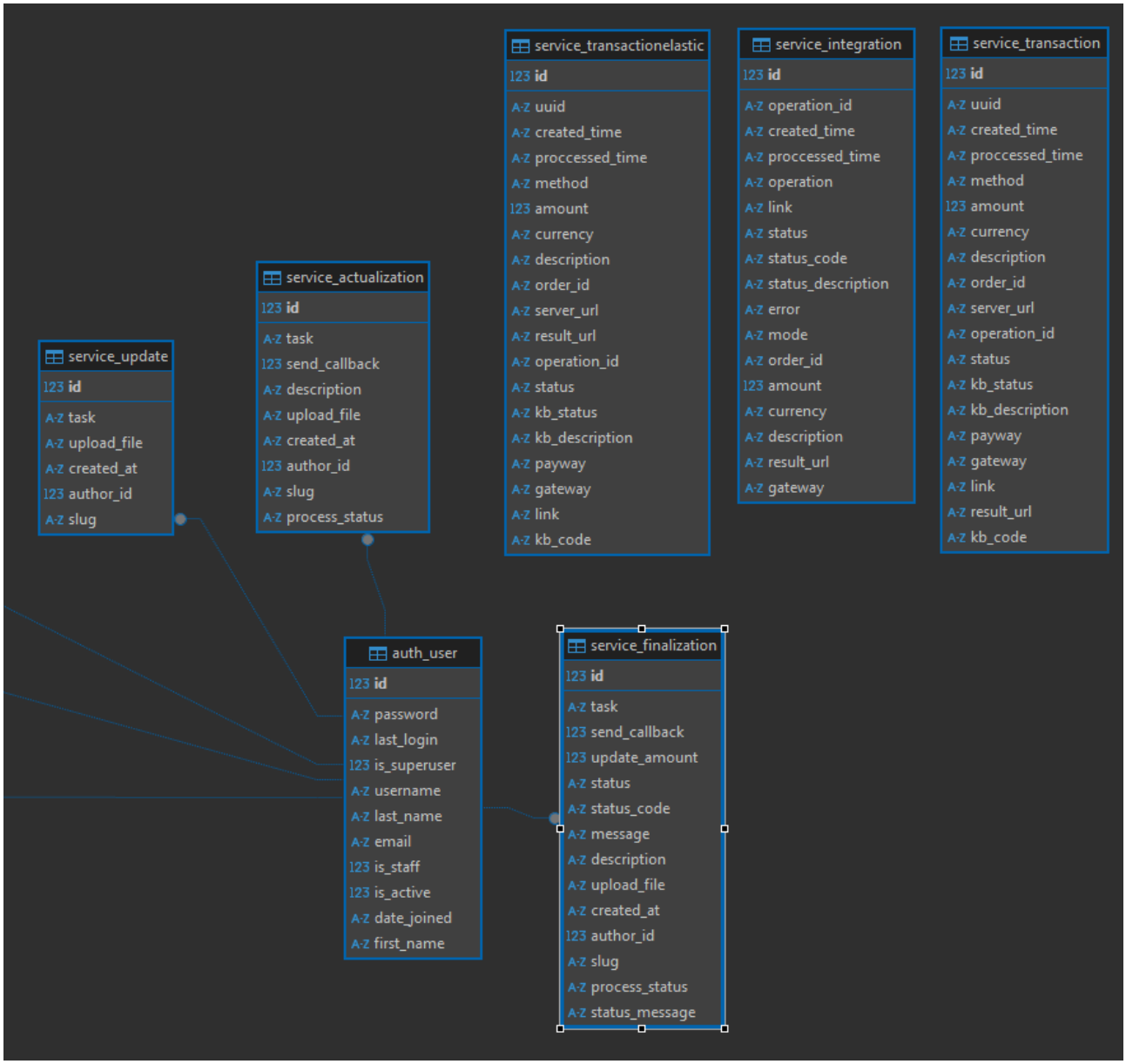

Рисунок В.1 – Фізична модель основних таблиць сервісу для бази даних

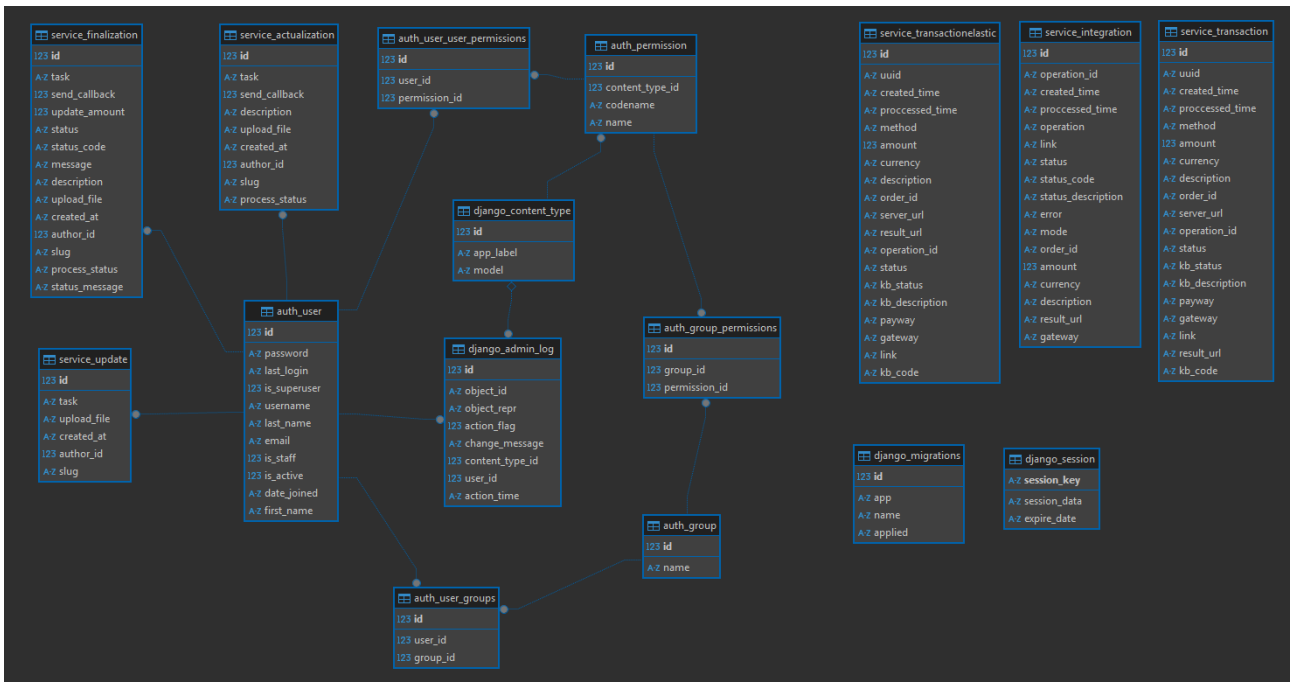

Рисунок В.2 – Схема фізичної бази даних сервісу

Додаток Г. Код розробки

```python
class Actualization(models.Model):
    PROCESS_CHOICES = (
        ('success', 'Success'),
        ('in_progress', 'In Progress'),
        ('failed', 'Failed'),
    )

    author = models.ForeignKey(User, on_delete=models.CASCADE)
    task = models.URLField(blank=False)
    send_callback = models.BooleanField(blank=True)
    description = models.TextField(blank=True)
    upload_file = models.FileField(upload_to='uploads/', blank=False)
    created_at = models.DateTimeField(auto_now_add=True)
    slug = models.SlugField(max_length=255, unique=True, blank=True)
    process_status = models.CharField(max_length=15, blank=True, choices=PROCESS_CHOICES)

    class Meta:
        ordering = ('-created_at',)

    def save(self, *args, **kwargs):
        if not self.slug:
            self.slug = generate_unique_slug()
            while Actualization.objects.filter(slug=self.slug).exists():
                self.slug = generate_unique_slug()
        super().save(*args, **kwargs)

    def get_absolute_url(self):
        return reverse('service:actualization_detail', kwargs={'slug': self.slug})

    def __str__(self):
        return self.slug
```

Рисунок Г.1 – Опис таблиці «Актуалізація»

```python
class Finalization(models.Model):
    STATUS_CHOICES = (…
    STATUS_CODE = (…
    PROCESS_CHOICES = (…

    author = models.ForeignKey(User, on_delete=models.CASCADE)
    task = models.URLField(blank=False)
    send_callback = models.BooleanField(blank=True)
    update_amount = models.BooleanField(blank=True)
    status = models.CharField(choices=STATUS_CHOICES, max_length=10, blank=False)
    status_code = models.CharField(choices=STATUS_CODE, max_length=4, blank=False)
    status_message = models.CharField(max_length=255, blank=False)
    message = models.CharField(blank=False)
    description = models.TextField(blank=True)
    upload_file = models.FileField(upload_to='uploads/', blank=False)
    created_at = models.DateTimeField(auto_now_add=True)
    slug = models.SlugField(max_length=255, unique=True, blank=True)
    process_status = models.CharField(max_length=15, blank=True, choices=PROCESS_CHOICES)

    class Meta:
        ordering = ('-created_at',)

    def save(self, *args, **kwargs):
        if not self.slug:
            self.slug = generate_unique_slug()
            while Finalization.objects.filter(slug=self.slug).exists():
                self.slug = generate_unique_slug()
        super().save(*args, **kwargs)

    def get_absolute_url(self):
        return reverse('service:finalization_detail', kwargs={'slug': self.slug})

    def __str__(self):
        return self.slug
```

Рисунок Г.2 – Опис таблиці «Фіналізація»

```python
class Update(models.Model):
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    task = models.URLField(blank=False)
    upload_file = models.FileField(upload_to='uploads/', blank=False)
    created_at = models.DateTimeField(auto_now_add=True)
    slug = models.SlugField(max_length=255, unique=True, blank=True)

    class Meta:
        ordering = ('-created_at',)

    def save(self, *args, **kwargs):
        if not self.slug:
            self.slug = generate_unique_slug()
            while Update.objects.filter(slug=self.slug).exists():
                self.slug = generate_unique_slug()
        super().save(*args, **kwargs)

    def get_absolute_url(self):
        return reverse('service:update_detail', kwargs={'slug': self.slug})

    def __str__(self):
        return self.slug
```

Рисунок Г.3 – Опис таблиці «Оновлення»

```python
class Transaction(models.Model):
    uuid = models.UUIDField(default=uuid.uuid4, unique=True)
    created_time = models.DateTimeField(auto_now_add=True)
    proccessed_time = models.DateTimeField(auto_now=True)
    method = models.CharField(max_length=255, blank=True)
    amount = models.DecimalField(max_digits=10, decimal_places=2)
    currency = models.CharField(max_length=3, blank=True)
    description = models.TextField(blank=True)
    order_id = models.CharField(max_length=255, blank=True)
    server_url = models.URLField(blank=True)
    result_url = models.URLField(blank=True)
    operation_id = models.CharField(max_length=255, blank=True)
    status = models.CharField(max_length=255, blank=True)
    kb_status = models.CharField(max_length=255, blank=True)
    kb_code = models.CharField(max_length=25, blank=True)
    kb_description = models.TextField(blank=True)
    payway = models.CharField(max_length=255, blank=True)
    gateway = models.CharField(max_length=255, blank=True)
    link = models.TextField(blank=True)

    def __str__(self):
        return str(self.uuid)
```

Рисунок Г.4 – Схема таблиці «Транзакція»

```python
class TransactionElastic(models.Model):
    uuid = models.UUIDField(default=uuid.uuid4, unique=True)
    created_time = models.DateTimeField(auto_now_add=True)
    proccessed_time = models.DateTimeField(auto_now=True)
    method = models.CharField(max_length=255, blank=True)
    amount = models.DecimalField(max_digits=10, decimal_places=2)
    currency = models.CharField(max_length=3, blank=True)
    description = models.TextField(blank=True)
    order_id = models.CharField(max_length=255, blank=True)
    server_url = models.URLField(blank=True)
    result_url = models.URLField(blank=True)
    operation_id = models.CharField(max_length=255, blank=True)
    status = models.CharField(max_length=255, blank=True)
    kb_status = models.CharField(max_length=255, blank=True)
    kb_code = models.CharField(max_length=25, blank=True)
    kb_description = models.TextField(blank=True)
    payway = models.CharField(max_length=255, blank=True)
    gateway = models.CharField(max_length=255, blank=True)
    link = models.TextField(blank=True)

    def __str__(self):
        return str(self.uuid)
```

Рисунок Г.5 – Схема таблиці «Еластік»

```python
class Integration(models.Model):
    operation_id = models.UUIDField(default=uuid.uuid4, unique=True)
    created_time = models.DateTimeField(auto_now_add=True)
    proccessed_time = models.DateTimeField(auto_now=True)
    operation = models.CharField(max_length=255, blank=True)
    link = models.TextField(blank=True)
    status = models.CharField(max_length=255, blank=True)
    status_code = models.CharField(max_length=255, blank=True)
    status_description = models.TextField(blank=True)
    error = models.CharField(max_length=255, blank=True)
    mode = models.CharField(max_length=255, blank=True)
    order_id = models.CharField(max_length=255, blank=True)
    amount = models.DecimalField(max_digits=10, decimal_places=2, blank=True)
    currency = models.CharField(max_length=3, blank=True)
    description = models.TextField(blank=True)
    result_url = models.URLField(blank=True)
    gateway = models.CharField(max_length=255, blank=True)

    def __str__(self):
        return str(self.operation_id)
```

Рисунок Г.6 – Схема таблиці «Інтеграція»

Фрагмент коду програми

Лістинг коду з файлу views.py (Логіка обробки сервісу)

```
from django.shortcuts import render, redirect
from .models import Actualization, Finalization, Update, Transaction,
TransactionElastic, Integration
from .forms import ActualizationForm, FinalizationForm, UpdateForm
from django.contrib.auth import get_user_model
from django.contrib.auth.decorators import login_required

User = get_user_model()

@login_required
def actualization_list(request):
    actualizations = Actualization.objects.all()
    return render(request, 'service/actualization/actualization_list.html',
{'actualizations': actualizations})

@login_required
def actualization_detail(request, slug):
    actualization = Actualization.objects.get(slug=slug)
    return render(request, 'service/actualization/actualization_detail.html',
{'actualization': actualization})

@login_required
def actualization_create(request):
    if request.method == 'POST':
        form = ActualizationForm(request.POST, request.FILES)
```

```
        if form.is_valid():
            actualization = form.save(commit=False)
            actualization.author = request.user
            actualization.process_status = 'in_progress'
            actualization.save()
            transaction = processing_file(actualization.upload_file.path)
            actualization_transaction(transaction)
            return redirect('service:actualization_list')
    else:
        form = ActualizationForm()
    return render(request, 'service/actualization/actualization_create.html', {'form':
form})

@login_required
def finalization_list(request):
    finalizations = Finalization.objects.all()
    return render(request, 'service/finalization/finalization_list.html', {'finalizations':
finalizations})

@login_required
def finalization_detail(request, slug):
    finalization = Finalization.objects.get(slug=slug)
    return render(request, 'service/finalization/finalization_detail.html', {'finalization':
finalization})

@login_required
def finalization_create(request):
    if request.method == 'POST':
        form = FinalizationForm(request.POST, request.FILES)
        if form.is_valid():
            finalization = form.save(commit=False)
            finalization.author = request.user
            finalization.process_status = 'in_progress'
            finalization.save()
            tranzaction = processing_file(finalization.upload_file.path)
            finalization_transaction(tranzaction, finalization)
            return redirect('service:finalization_list')
    else:
        form = FinalizationForm()
    return render(request, 'service/finalization/finalization_create.html', {'form': form})

@login_required
def update_list(request):
    updates = Update.objects.all()
    return render(request, 'service/update/update_list.html', {'updates': updates})
```

```
@login_required
def update_create(request):
    if request.method == 'POST':
        form = UpdateForm(request.POST, request.FILES)
        if form.is_valid():
            update = form.save(commit=False)
            update.author = request.user
            update.save()
            transaction = processing_file(update.upload_file.path)
            update_elastic_db(transaction)
            return redirect('service:update_list')
    else:
        form = UpdateForm()
    return render(request, 'service/update/update_create.html', {'form': form})

def processing_file(file):
    uuid_amount_map = {}
    with open(file, 'r', encoding='utf-8') as f:
        for line in f:
            line = line.strip()
            if not line:
                continue
            parts = line.split(',')
            uuid = parts[0].strip()
            if len(parts) == 2:
                try:
                    amount = int(parts[1].strip())
                except ValueError:
                    amount = None
            else:
                amount = None
            uuid_amount_map[uuid] = amount
    return uuid_amount_map

def update_elastic_db(transaction):
    for i in transaction:
        old_transaction = Transaction.objects.filter(uuid=i).first()
        new_transaction = TransactionElastic.objects.filter(uuid=i)
        if old_transaction is not None and new_transaction.exists() is False:
            source_data = Transaction.objects.get(uuid=i)
            TransactionElastic.objects.create(
                uuid=source_data.uuid,
                created_time=source_data.created_time,
                proccessed_time=source_data.proccessed_time,
```

```
                method=source_data.method,
                amount=source_data.amount,
                currency=source_data.currency,
                description=source_data.description,
                order_id=source_data.order_id,
                server_url=source_data.server_url,
                result_url=source_data.result_url,
                operation_id=source_data.operation_id,
                status=source_data.status,
                kb_status=source_data.kb_status,
                kb_code=source_data.kb_code,
                kb_description=source_data.kb_description,
                payway=source_data.payway,
                gateway=source_data.gateway,
                link=source_data.link
            )
        elif old_transaction is not None and new_transaction.exists():
            source_data = Transaction.objects.get(uuid=i)
            TransactionElastic.objects.update(
                uuid=source_data.uuid,
                created_time=source_data.created_time,
                proccessed_time=source_data.proccessed_time,
                method=source_data.method,
                amount=source_data.amount,
                currency=source_data.currency,
                description=source_data.description,
                order_id=source_data.order_id,
                server_url=source_data.server_url,
                result_url=source_data.result_url,
                operation_id=source_data.operation_id,
                status=source_data.status,
                kb_status=source_data.kb_status,
                kb_code=source_data.kb_code,
                kb_description=source_data.kb_description,
                payway=source_data.payway,
                gateway=source_data.gateway,
                link=source_data.link
            )

def show_table_transaction(request):
    transactions = TransactionElastic.objects.all()
    fields = [field.name for field in TransactionElastic._meta.fields]

    context = {
        'transactions': transactions,
```

```
        'fields': fields
    }
    return render(request, 'service/update/table_transaction.html', context)

def actualization_transaction(transaction):
    for uuid, _ in transaction.items():
        transaction = Transaction.objects.filter(uuid=uuid).values('gateway', 'link',
'operation_id')
        tr_fields = transaction.first()
        if transaction.exists():
            operation =
Integration.objects.filter(operation_id=tr_fields['operation_id']).values('gateway',
'link', 'operation_id')
            op_fields = operation.first()
            if op_fields['gateway'] == tr_fields['gateway'] and op_fields['link'] ==
tr_fields['link'] and str(op_fields['operation_id']) == tr_fields['operation_id']:
                operation.update(status='Pending', status_code='22001',
status_description='Transaction is pending')
                transaction.update(status='Pending')


def finalization_transaction(transaction, finalization):
    task = Finalization.objects.filter(slug=finalization).first()
    print(transaction)
    for uuid, amount in transaction.items():
        print(uuid, amount)
        transaction = Transaction.objects.filter(uuid=uuid).values('gateway', 'link',
'operation_id')
        tr_fields = transaction.first()
        if transaction.exists():
            operation =
Integration.objects.filter(operation_id=tr_fields['operation_id']).values('gateway',
'link', 'operation_id')
            op_fields = operation.first()
            if op_fields['gateway'] == tr_fields['gateway'] and op_fields['link'] ==
tr_fields['link'] and str(op_fields['operation_id']) == tr_fields['operation_id']:
                operation.update(status=task.status, status_code=task.status_code,
status_description=task.status_message, error=task.message, amount=amount)
                transaction.update(status='Processed', kb_status=task.status,
kb_code=task.status_code, kb_description=task.status_message, amount=amount)
    Finalization.objects.filter(slug=finalization).update(process_status=Finalization.PR
OCESS_CHOICES[0][0])
```

Лістинг коду з файлу table_transaction.html (Логіка реалізації пошуку на сторінці)

```
document.addEventListener('DOMContentLoaded', function () {
   var tooltipTriggerList = [].slice.call(document.querySelectorAll('[data-bs-
toggle="tooltip"]'))
   tooltipTriggerList.forEach(function (tooltipTriggerEl) {
      new bootstrap.Tooltip(tooltipTriggerEl)
   })

   const searchForm = document.getElementById('searchForm');
   const input = document.getElementById('searchInput');
   const clearBtn = document.getElementById('clearBtn');
   const table = document.getElementById('dataTable');
   const tbody = table.querySelector('tbody');
   const rows = tbody.getElementsByTagName('tr');
   const uuidIndex = 1;
   const operationIdIndex = 11;

   function filterTable() {
      const filter = input.value.trim().toLowerCase();
      if (filter === '') {
         for (let row of rows) {
            row.style.display = '';
         }
         return;
      }
      for (let row of rows) {
         const cells = row.getElementsByTagName('td');
         const uuidText = cells[uuidIndex]?.textContent.trim().toLowerCase() || '';
         const opIdText = cells[operationIdIndex]?.textContent.trim().toLowerCase() ||
'';
         if (uuidText === filter || opIdText === filter) {
            row.style.display = '';
         } else {
            row.style.display = 'none';
         }
      }
   }
   searchForm.addEventListener('submit', function (e) {
      e.preventDefault();
      filterTable();
   });
   clearBtn.addEventListener('click', function () {
      input.value = '';
      filterTable();
      input.focus();
   });
```

Додаток Д. Інтерфейс сервісу

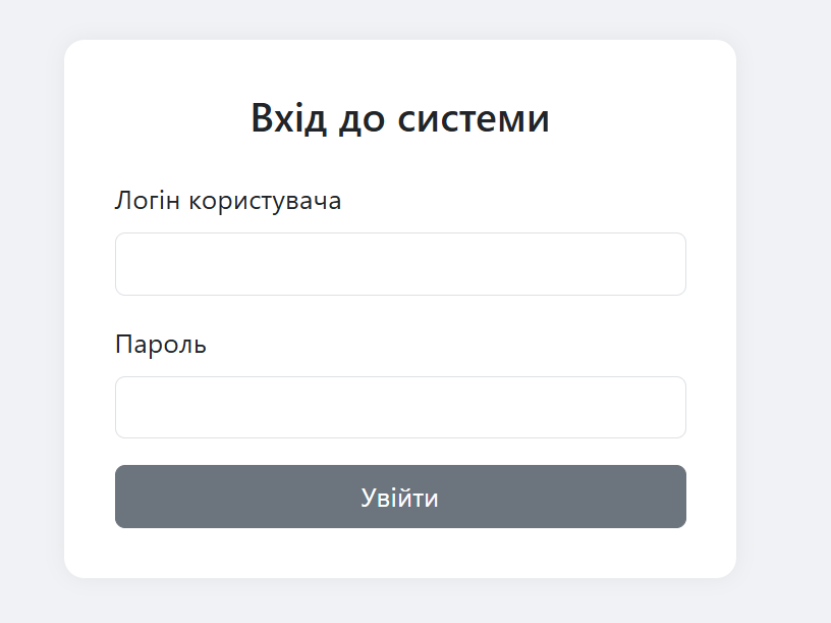

Рисунок Д.1 – Форма введення даних для авторизації на сервіс.

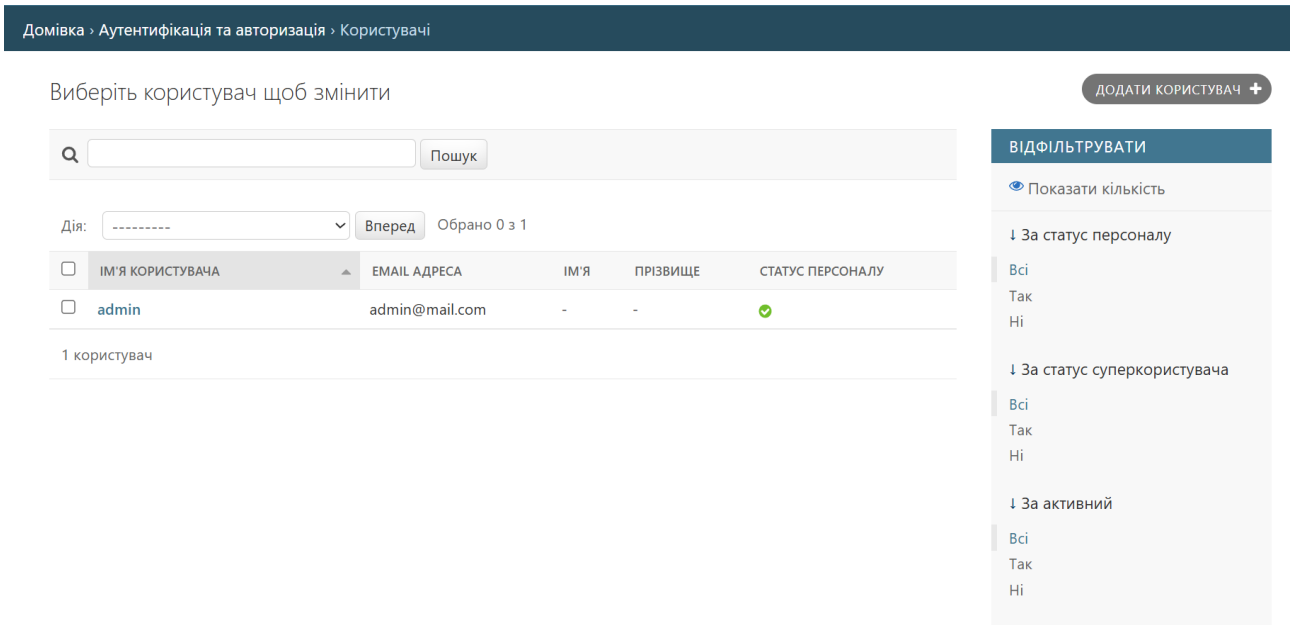

Рисунок Д.2 – Меню адміністратора для додавання або зміни даних користувача

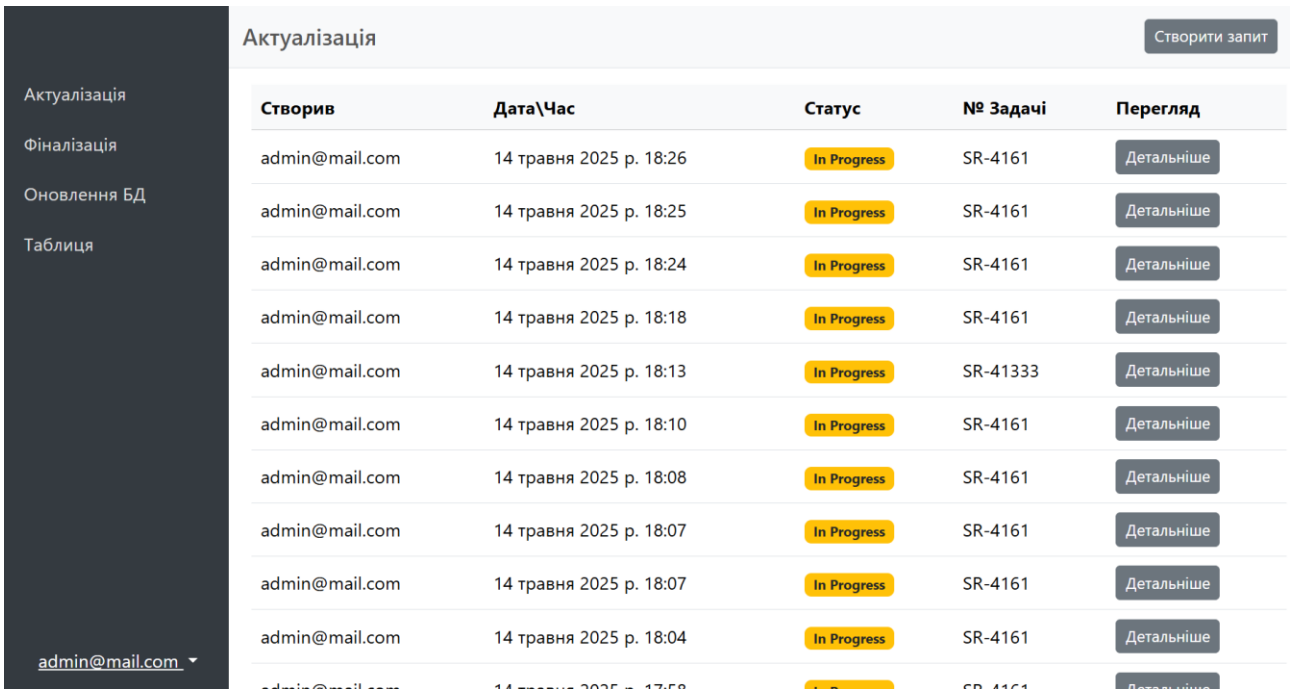

Рисунок Д.3 – Головне меню з актуалізації, що відображається після успішної авторизації.

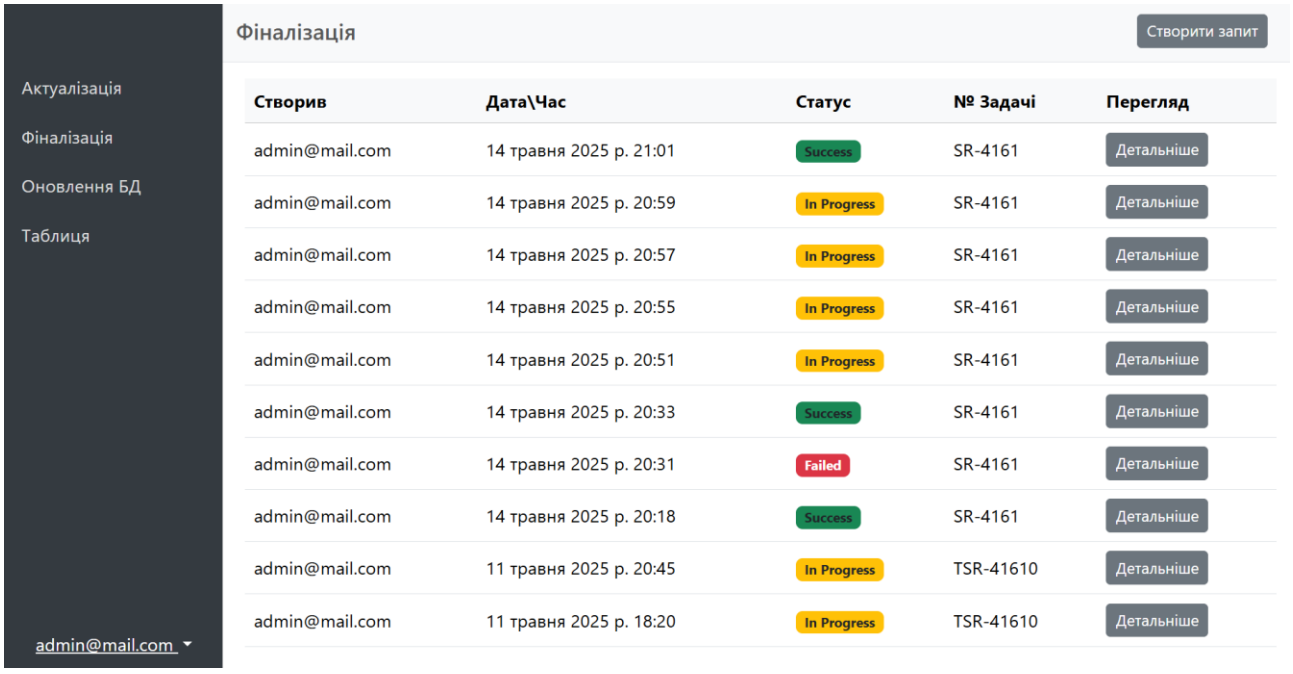

Рисунок Д.4 – Меню задач фіналізації

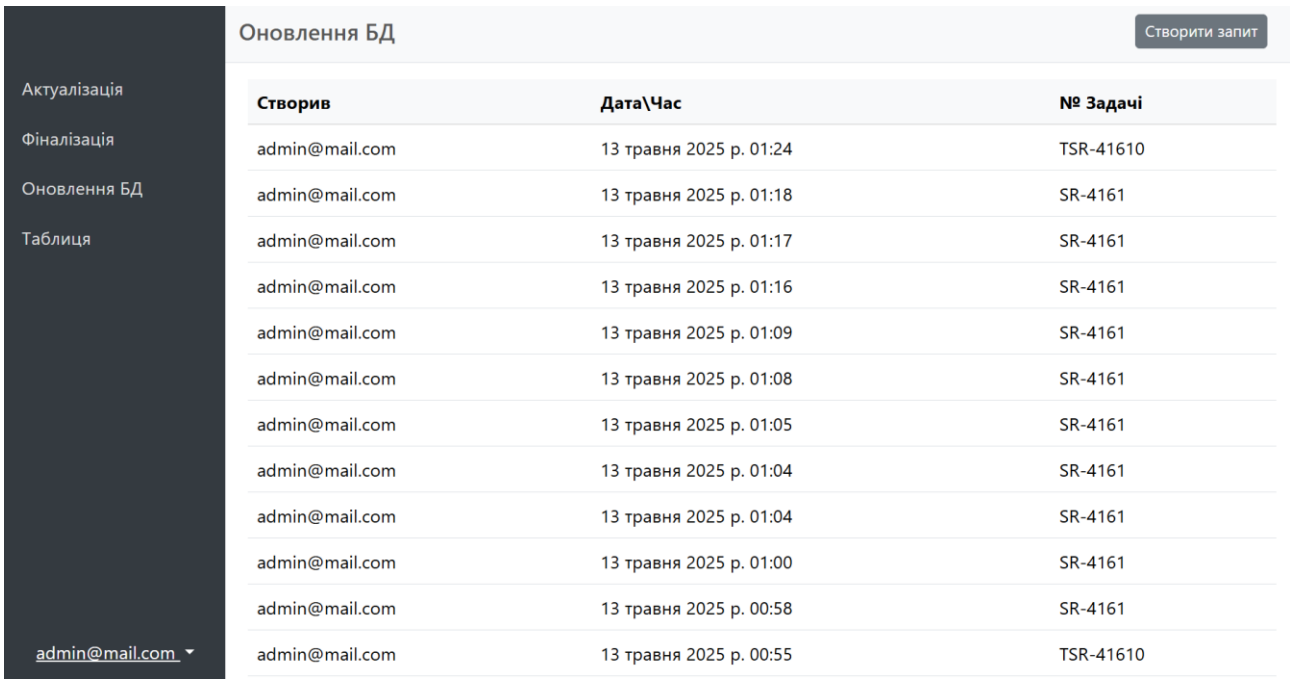

Рисунок Д.5 – Меню відображення задач на створення оновлення в Еластіку

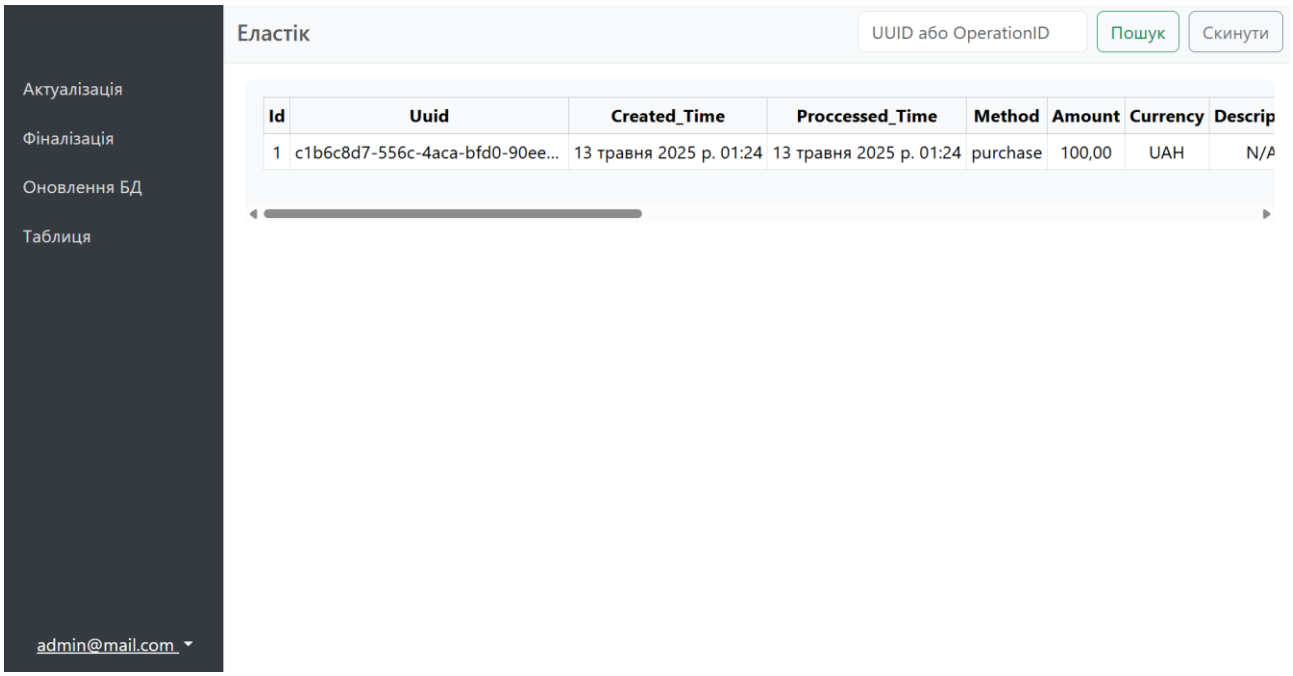

Рисунок Д.6 – Сторінка на якій відображається копія БД «Транзакція»

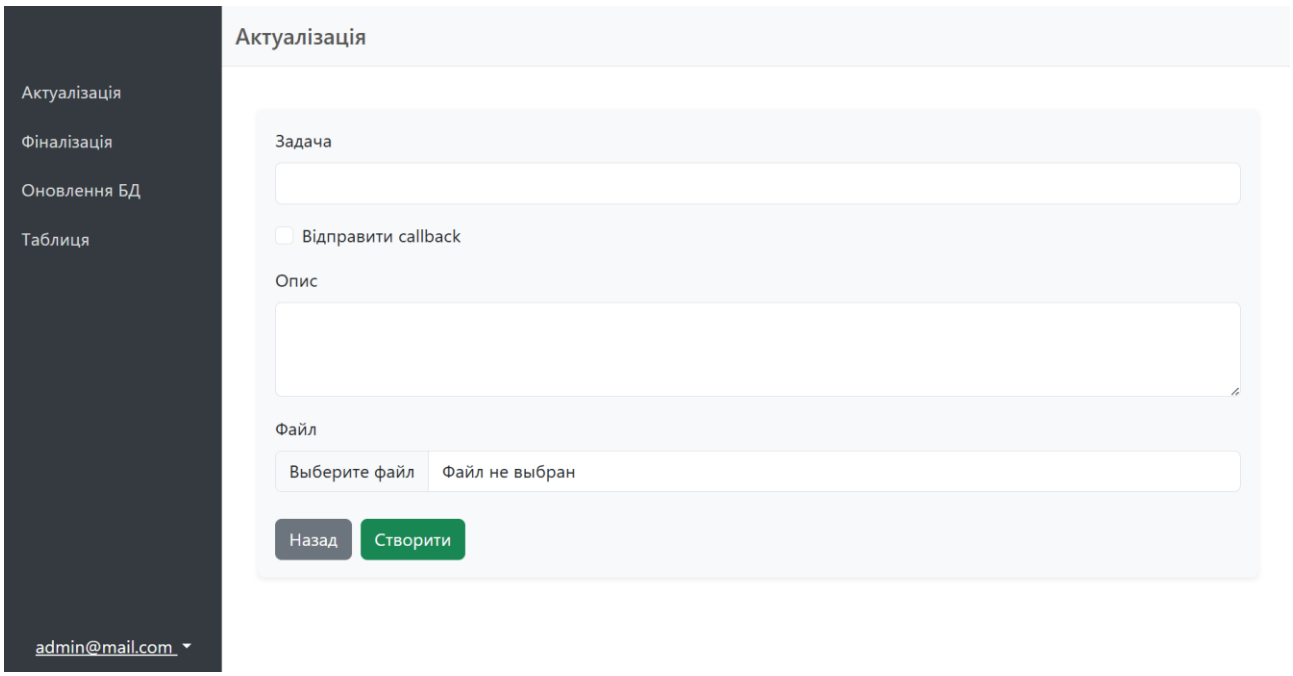

Рисунок Д.7 – Форма для створення автоматичної задачі на актуалізацію даних

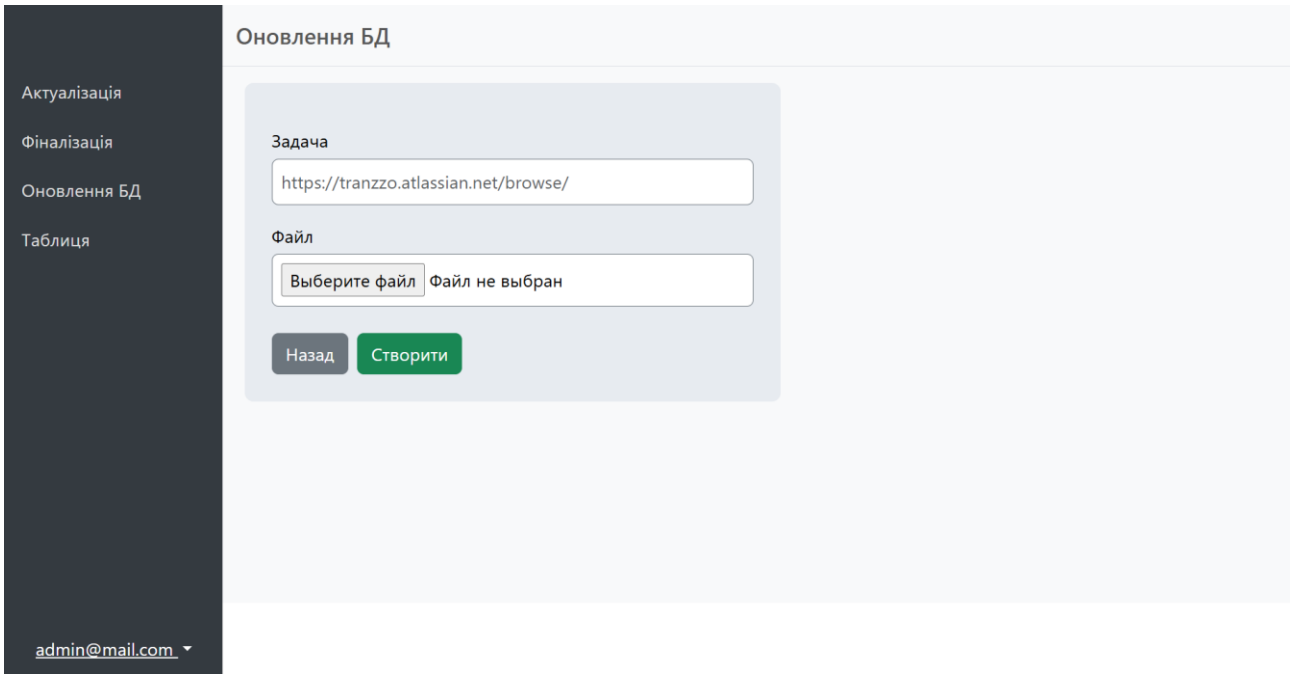

Рисунок Д.8 – Форма для створення автоматичної задачі на оновлення даних в таблиці «Еластік»

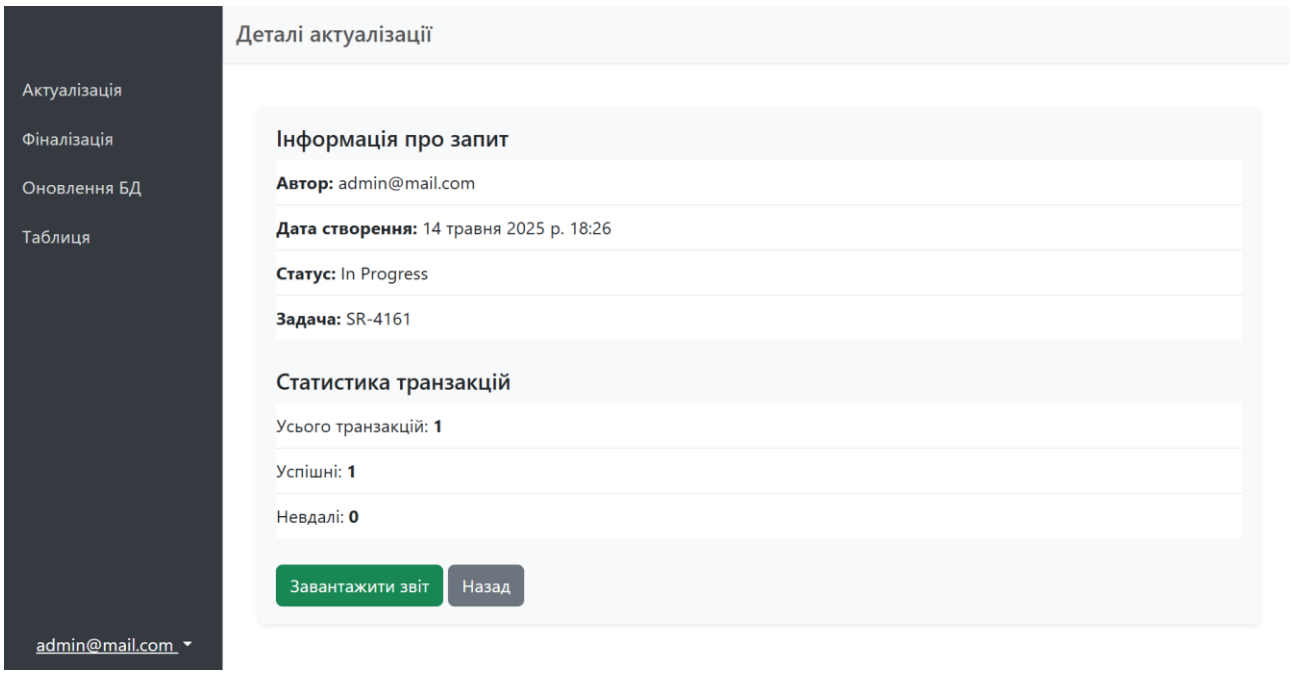

Рисунок Д.9 – Форма результату обробки задачі на автоматичну актуалізацію транзакцій