КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

ВПРОВАДЖЕННЯ ТА ІНТЕГРАЦІЯ ПЛАТІЖНИХ ІНТЕРФЕЙСІВ

Виконав студент 4-го курсу
ГОЛОВАЧ Олександр В'ячеславович _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
ПАНЧЕНКО Тарас Володимирович _____
(підпис)

Засвідчую, що в цій роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри теорії та технології
програмування
«____» _____ 2023 р.,
протокол № ___
Завідувач кафедри
Микола НІКІТЧЕНКО _____
(підпис)

Київ — 2023

РЕФЕРАТ

Обсяг роботи 55 сторінок, 9 ілюстрацій, 8 джерел посилань.

ВЕБ-ЗАСТОСУНОК, СИСТЕМА ЕЛЕКТРОННИХ ПЛАТЕЖІВ, ЕЛЕКТРОННИЙ ПІДПИС, C#, .NET FRAMEWORK, TYPESCRIPT, ANGULAR, DOCKER

Об’єктом роботи є створення на основі технології .NET Framework системи, інтеграції, для автоматичної обробки міжбанківських платіжних операцій.

Предметом роботи є програмний застосунок, що може бути підключений до автоматизованої банківської системи, а реалізований функціонал може бути використаний у цій системі.

При створенні системи було використано мову програмування C#, мову програмування TypeScript [1], фреймворк .NET5 [2], систему управління базами даних Oracle.

В результаті виконання роботи було виконано огляд засобів та методів розробки, проаналізовано переваги розробки інтеграції для автоматизованих банківських систем, розроблено програмний продукт.

Розроблений в дипломній роботі веб-сервіс було інтегровано та впроваджено для використання в промисловому середовищі Державного Ощадного Банку України .

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ОС – операційна система.

API (прикладний програмний інтерфейс) – це комплекс вже готових класів, функцій, структур і тд., які надаються застосунком (бібліотекою, сервісом) чи операційною системою для використання у зовнішніх програмних продуктах.

CLI (Command Line Interface) – інтерфейс командного рядка, консоль.

HTML – це мова розмітки веб-сторінок у мережі Інтернет.

HTTP – протокол передачі даних, що використовується у інфокомунікаційних мережах.

IDE (Integrated Development Environment) – інтегроване середовище розробки.

JavaScript (JS) – мова програмування високого рівня.

ЕЦП – електронний цифровий підпис.

АЦСК – акредитований центр сертифікації ключів.

ЦЗО – центральний засвідчу вальний орган.

НБУ – Національний банк України

СЕП – Система електронних платежів Національного банку України

ЦОСЕП – Центр оброблення СЕП

САБ – Система автоматизації банку

ТКР - Технічний кореспондентський рахунок

Банківський день - період робочого часу банку, протягом якого проводяться розрахункові операції з клієнтами банку і позначаються (датуються) цим числом.

Календарний день – проміжок часу від 00:00 до 24:00 з відповідною календарною датою.

Платіжне повідомлення – повідомлення ISO 20022 [8], яке містить електронні розрахункові документи на виконання переказу коштів через СЕП, тобто в термінах ISO 20022 – платіжну інструкцію.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Згідно до закону “Про платіжні послуги в Україні”, прийнятого 27 липня 2022 року парламентом, кожна фінансова установа має реалізувати своє рішення для інтеграції Системи автоматизації банку з центром обробки системи електронних платежів розміщеним у центральній розрахунковій палаті Національного банку України.

На поточний час в Україні існує 6 основних систем автоматизації банку, які широко використовуються різними фінансовими установами. Інтеграція систем стає все більш актуальною задачею, оскільки вона сприяє оптимізації робочих процесів, забезпечує швидкий доступ до необхідної інформації та підвищує ефективність функціонування банку в цілому. У цьому контексті велике значення набуває інтеграція Системи автоматизованого банківського обслуговування (САБ) з Центральною обробною системою електронних платежів (ЦОСЕП), що дозволяє покращити якість обслуговування клієнтів та забезпечити надійність та безпеку операцій.

Об’єкт, методи й засоби розроблення. Об’єктом роботи є платіжні інтерфейси систем автоматизації банку для інтеграції з центром обробки СЕП Національного Банку України. Предметом роботи є вебзастосунок для автоматизації процесу обробки платіжних документів.

Під час розробки вебзастосунку використовувались такі технології, як:

- Фреймворк Entity Framework Core (EF Core);
- Фреймворк ASP.NET WEB API;
- Система управління базами даних Oracle;
- Фреймворк для розробки інтерфейсу візуалізації Angular 11;
- Мова програмування C#;

Мета й завдання роботи. Метою даної дипломної роботи є розробка універсального інтеграційного рішення, яке може використовуватися з різними САБ системами українських банків в рамках реалізації Системи електронних платежів 4-го покоління (СЕП-4). Проект має на меті вирішити проблему розділеності інформаційних систем, яка часто стає перепоною для

ефективного взаємодії банківських систем та реалізації інноваційних рішень у сфері банківських послуг. Для досягнення поставленої мети були визначені такі завдання:

- провести аналіз предметної області;
- створити новий проєкт та підключити існуючу САБ систему;
- налаштувати, створити контролери та представлення;
- стилізувати візуальну частину проєкту;
- розробити автоматизоване рішення обробки платіжних пакетів;
- налаштувати фільтри пошуку;
- реалізувати формування та збереження архівів пакетів оброблених за банківський день;
- реалізувати авторизацію користувача;

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз системи електронних платежів

СЕП-4 є наступним поколінням державної системи електронних міжбанківських розрахунків (СЕП). Її відмінними рисами порівняно з СЕП-3.х є:

- стандартизація структур обміну інформацією з учасниками СЕП та депозитаріями за стандартом ISO 20022;
- цілодобове (24/7) функціонування;
- застосування нової системи захисту інформації, у тому числі криптографічного;

Розроблення нового покоління СЕП (СЕП-4) передбачено проектом “Впровадження ISO 20022 в платіжній інфраструктурі України”, який, у свою чергу, базується на “Хартії приєднання до ISO 20022” (ISO 20022 Harmonisation Charter), підписаної Національним банком України 23.05.2016.

Технологія та правила оброблення інформації в СЕП є однаковими протягом всього часу її функціонування, зокрема, умови приймання платежів до СЕП не залежать від часу доби. Передбачається можливість технологічних перерв для зміни програмного забезпечення та інших технологічних робіт. НБУ має попереджати учасників СЕП про технологічні перерви заздалегідь. Проте технічна реалізації СЕП має якомога зменшити потребу в таких технологічних перервах.

СЕП виконує переказ коштів виключно в національній валюті (гривня). Проте у структурах повідомлень СЕП передбачені реквізити, що в майбутньому нададуть можливість роботи СЕП також в інших валютах.

У СЕП-4.0 реалізовані такі інструменти ISO 20022:

- Кредитовий переказ (Credit Transfer), у складі якого передбачено реалізацію кредитового переказу коштів агента на рівні агентів (Financial Institution Credit Transfer) та кредитового переказу коштів клієнта на рівні агентів (FIToFI Customer Credit Transfer);

- Повернення платежу, ініційоване клієнтом-отримувачем, фінансовою установою-отримувачем або Агентом отримувача (FIToFI Customer Credit Transfer або Financial Institution Credit Transfer – Payment Return) ;
- Запит отримувача на здійснення платежу платником (Creditor Payment Activation Request);
- Запит на відкликання (скасування) (FIToFI Payment Cancellation Request)
- Відповідні статусні та підсумкові повідомлення;
- Міжбанківський прямий дебет без права опротестування як інструмент, що його використовуватиме виключно НБУ для безспірного / примусового списання (стягнення) коштів, договірного списання за заборгованістю банку перед НБУ, відображення від'ємної нетто-позиції за клірингом НПС "ПРОСТІР";
- Exceptions and Investigations щонайменше в обсязі, який відповідає нормативній базі з питань уточнення реквізитів платежу станом на 2 квартал 2020 року та відповідному функціоналу чинної СЕП-3;

СЕП не постачає інших інструментів зміни змісту вже проведеної трансакції.

Управління рахунками, у складі якого передбачено реалізацію функцій:

- інформування учасника про стан його технічного рахунку та установлені для нього ліміти
- керування з боку головного банку роботою його філій-безпосередніх учасників СЕП шляхом установлення лімітів для філій та отримання інформації про технічні рахунки філій.

СЕП здійснює розрахунки в коштах центробанку (за міжнародною термінологією). Це означає, що:

- в Головній книзі НБУ для учасників СЕП відкрито кореспондентські рахунки банків та аналогічні їм рахунки "Інших установ" (далі – коррахунки)

– ЦОСЕП веде технічні рахунки учасників (далі – ТКР), на яких у режимі реального часу відображає розрахунки, здійснені учасниками через СЕП
– ЦОСЕП з певною періодичністю передає обороти по ТКР, виконані через СЕП, до Операційного департаменту НБУ для відображення на коррахунках у Головній книзі НБУ. Заборонено виконання будь-яких інших операцій з коррахунками у Головній книзі НБУ, крім відображення розрахунків через СЕП. Таким чином, ТКР учасника в СЕП представляє собою відображення в реальному часі стану коррахунку цього учасника. 2. НБУ забороняє овердрафт (від`ємний залишок) на коррахунках учасників.

Крім зазначених, можуть бути розроблені додаткові умови приймання трансакцій та/або технологічні контролі, які унеможливлять виникнення від`ємного залишку на технічному рахунку учасника для випадку здійснення Національним банком України безспірного/примусового/договірного списання. Мають бути розроблені заходи для мінімізації впливу надзвичайних ситуацій та людського фактору, які гарантуватимуть дотримання Національним банком регламенту відшкодування від`ємного залишку на ТКР учасників СЕП після відображення від’ємної нетто-позиції за клірингом НПС «ПРОСТІР» до кінця календарного дня.

1.2. Автоматизована банківська система

Система автоматизації банку (САБ) є важливим програмним забезпеченням, яке забезпечує ефективну та надійну обробку поточної внутрішньобанківської діяльності, включаючи бухгалтерський облік, обслуговування рахунків клієнтів та інші важливі функції. Ця система допомагає банкам оптимізувати свою роботу, підвищувати продуктивність і забезпечувати високу якість обслуговування клієнтів.

Однією з головних функцій САБ є бухгалтерський облік. Вона дозволяє автоматизувати процеси обліку фінансових операцій, управління активами та

пасивами, звітності та аналізу фінансової діяльності банку. Це допомагає зменшити ризики помилок, спростити процеси звітності та забезпечити точність фінансових даних.

Крім того, САБ забезпечує обслуговування рахунків клієнтів. Це охоплює всі аспекти взаємодії з клієнтами, такі як відкриття рахунків, внесення та зняття коштів, перекази, операції з кредитування та багато іншого. Завдяки автоматизації цих процесів, банки можуть прискорити обробку операцій, зменшити час очікування клієнтів та забезпечити швидкий доступ до рахункової інформації.

САБ також надає можливості для управління ризиками. Вона дозволяє банкам встановлювати та контролювати внутрішні правила та обмеження, що стосуються фінансових операцій. Це допомагає виявляти та запобігати шахрайству, викривати неправомірні дії та мінімізувати фінансові ризики для банку.

Загалом, Система автоматизації банку є необхідним інструментом для банківських установ у сучасному світі. Вона допомагає підвищити ефективність, знизити витрати, поліпшити обслуговування клієнтів і забезпечити високий рівень безпеки та точності даних. Без САБ, банкам було б важко впоратися зі складними завданнями сучасного банкінгу, тому ця система є невід'ємною частиною їхньої успішної діяльності.

1.3. Базові характеристики СЕП як платіжної системи

Система електронних міжбанківських розрахунків (СЕП) є платформою, в якій беруть участь банки, їх філії, Національний банк та інші установи, які мають право працювати в рамках СЕП згідно з українським законодавством. ("Інші" установи). Технологічний процес роботи "Інших" установ в СЕП, якщо вони не є банками, в цілому аналогічний роботі банків, за винятком окремих відмінностей, якщо такі є. Клієнти банків не мають прямого доступу до СЕП.

СЕП є системою високого рівня (RTGS) та не використовує механізми клірингу. Коли електронний розрахунковий документ надходить до Центральної оброблювальної системи СЕП (ЦОСЕП), рішення про його виконання моментально відображається на технічному рахунку відправника і отримувача в ЦОСЕП.

СЕП обробляє повідомлення в момент їх фізичного надходження та, в разі успішної перевірки, негайно виконує дії, передбачені для цього типу повідомлення. Наприклад, у випадку платіжного повідомлення він негайно здійснює проведення за технічними рахунками і передає платіжне повідомлення отримувачу. Не передбачається можливість "запланувати платіж наперед, щоб СЕП виконала його в певний час".

Структури обміну інформацією між ЦОСЕП та учасниками СЕП базуються на міжнародному стандарті ISO 20022 з урахуванням відповідних нормативно-правових актів, що регулюють імплементацію стандарту в Україні.

Повідомлення, що надійшли до СЕП, є остаточними та безвідкличними. Коли платіжне, супутнє (інформаційне) або технологічне повідомлення прийнято для виконання в ЦОСЕП, відправник не може його відкликати, а отримувач не може відмовитися від його прийняття. Повернення або відкликання платіжної інструкції є окремим інструментом взаємодії між фінансовими установами та їх клієнтами і реалізується в СЕП за допомогою окремого набору повідомлень ISO 20022 "повернення" / "відкликання (скасування)".

СЕП не веде чергу платежів, які не виконуються через недостатність коштів на технічному рахунку учасника. У разі недостатньої кількості коштів на технічному рахунку СЕП повертає учаснику повідомлення-квитанцію (в термінології ISO 20022 - статус оброблення повідомлення або звіт про статус платежу) про невиконання цього платежу. Далі учасник самостійно вирішує питання черги невиконаних платежів.

Повідомлення в СЕП обробляються в порядку їх фізичного надходження до ЦОСЕП. Немає пріоритетів обробки повідомлень на основі будь-яких критеріїв, окрім черговості надходження до ЦОСЕП.

Загальне обмеження щодо суми одного платежу або загальної суми платежів в повідомленні визначається у форматі реквізиту "Сума" стандарту ISO 20022. Додаткові обмеження щодо суми платежу (наприклад, обмеження на основі залишку на технічному рахунку) встановлюються окремо для кожного інструменту СЕП.

СЕП як платіжна система не забезпечує автоматизовану підтримку ліквідності учасників. Рішення про необхідність підтримки ліквідності банків та відповідні заходи приймаються відповідними функціональними підрозділами Національного банку відповідно до нормативно-правових актів.

1.4. Платіжні повідомлення pacs.008 та pacs.009

Повідомлення про платіж pacs.008 "Кредитовий переказ коштів клієнта на рівні агентів" використовується для передачі коштів клієнта на рівні агентів (безпосередніх учасників СЕП). Повідомлення про платіж pacs.009 "Кредитовий переказ коштів агента на рівні агентів" використовується для передачі коштів агента на рівні агентів (безпосередніх учасників СЕП). Визначення використовуваного повідомлення для кожного конкретного випадку залежить від юридичного статусу платника та отримувача:

1. У повідомленні pacs.008 платник (дебітор) та отримувач (кредитор) ідентифікуються лише як фізичні/юридичні особи. Тому, якщо фінансова установа - банк або ASPSP виступає як відправник/отримувач, то їх слід описувати як юридичну особу, а не фінансову установу. Відповідно, перевірка, чи є ця юридична особа такою самою фінансовою установою, не проводиться.
2. У повідомленні pacs.009 платник (дебітор) та отримувач (кредитор) ідентифікуються як фінансові установи. Тому для них виконуються

перевірки відповідності платника (дебітора)/отримувача (кредитора) та ролей агентів у повідомленні. Повідомлення про платіж pacs.008/pacs.009 може містити одну трансакцію (Рис.2 "Алгоритм взаємодії в процесі здійснення переказу коштів на рівні агент - ЦОСЕП - агент (повідомлення містить одну трансакцію)") або більше однієї трансакції (Рис.3 "Алгоритм взаємодії в процесі здійснення переказу коштів на рівні агент - ЦОСЕП - агент (повідомлення містить більше однієї трансакції)"). Якщо повідомлення про платіж pacs.008/pacs.009 містить кілька трансакцій, то агент отримувача повинен бути одним і тим самим для всіх цих трансакцій. Іншими словами, одне повідомлення про платіж не розбивається на кілька інших під час проходження через СЕП до агента отримувача.

Проведення успішних трансакцій, що містяться в платіжному повідомленні, в ЦОСЕП відбувається в порядку їх розміщення у повідомленні без сортування. ЦОСЕП виконує процедури прийняття до виконання, наступної обробки та пересилання коректного платіжного повідомлення pacs.008/pacs.009 на рівні агентів. Крім того, ЦОСЕП надсилає такий набір повідомлень залежно від результатів обробки платіжного повідомлення pacs.008/pacs.009:

1. Повідомлення pacs.002 "Звіт про статус на рівні агентів" відображає статуси обробки pacs.008/pacs.009 в ЦОСЕП. Він може мати наступні статуси:
    - Відхилено (RJCT): платіжне повідомлення відхилено в цілому, не було прийнято жодної трансакції. Вказується причина відхилення повідомлення.
    - Частково прийнято (PART): у разі, якщо платіжне повідомлення містить кілька трансакцій, частина з них може бути прийнята, а частина відхилена. Інформація про успішно оброблені трансакції та список відхилених трансакцій надсилається у повідомленні pacs.002. Для кожної відхиленої трансакції вказується причина відхилення.

2. Повідомлення camt.054 "Повідомлення про зарахування/списання коштів з рахунку" використовується в ЦОСЕП для надсилання підтвердження про списання коштів Інструктуючому агенту та підтвердження про зарахування коштів Проінструктованому агенту. Воно містить інформацію про всі успішно оброблені трансакції з pacs.008/pacs.009.

Повідомлення pacs.002 "Звіт про статус на рівні агентів" носить інформаційний характер стосовно статусу обробки платіжного повідомлення і не є підтвердженням оплати. Воно надсилається лише для одного платіжного повідомлення (pacs.008/pacs.009).

Якщо всі трансакції з платіжного повідомлення успішно оброблені, то ЦОСЕП не формує повідомлення pacs.002 та не надсилає його Інструктуючому агенту.

Повідомлення camt.054 "Повідомлення про зарахування/списання коштів з рахунку" використовується для надсилання підтвердження про списання та зарахування коштів за всіма успішно обробленими трансакціями з pacs.008/pacs.009. Це повідомлення надсилається як Інструктуючому агенту, так і Проінструктованому агенту з відповідною інформацією для кожної трансакції.

1.5. Електронний підпис XML повідомлень

WS-Security є стандартом безпеки для веб-сервісів, який забезпечує підпис XML повідомлень. Цей протокол є важливим інструментом для забезпечення конфіденційності, цілісності та автентичності даних, що передаються через веб-сервіси.

WS-Security дозволяє захищати повідомлення на рівні XML-документу. Це означає, що дані вміщуються в елементи XML і підписуються за допомогою цифрового підпису. Цей підпис дозволяє перевірити автентичність повідомлення та його цілісність при його отриманні.

Один з основних компонентів WS-Security - це підпис XML повідомлень. Це забезпечує гарантію, що повідомлення не було змінено під час передачі. Цифровий підпис генерується за допомогою приватного ключа, який відомий лише відправнику, і перевіряється за допомогою публічного ключа, який відомий всім сторонам, які отримують повідомлення. Це забезпечує надійний механізм перевірки цілісності повідомлення.

XML DSignature є стандартом для електронного підписування XML-документів. Ця технічна документація надає опис використання чотирьох основних компонентів: CanonicalizationMethod, BinarySecurityToken, DigestMethod та Transforms. Використання цих компонентів забезпечує цілісність та конфіденційність XML-даних, а також довіреність підпису.

CanonicalizationMethod є частиною XML-DSig специфікації і використовується для забезпечення однозначності XML-документу. Він дозволяє видаляти зайві пробіли, коментарі та інші незначні деталі, що не впливають на значення документу. CanonicalizationMethod забезпечує стабільний вигляд документа незалежно від додаткових елементів, які можуть бути додані або вилучені.

BinarySecurityToken використовується для представлення підпису у вигляді бінарних даних. Це може бути зашифроване значення підпису, яке додатково забезпечує безпеку даних. BinarySecurityToken може містити ключі, сертифікати або інші ідентифікаційні дані, які допомагають перевірити автентичність підпису.

DigestMethod використовується для обчислення хеш-функції документа. Це дозволяє згенерувати унікальний ідентифікатор для документа, який використовується для порівняння з підписом. DigestMethod гарантує, що навіть незначні зміни в документі призведуть до різних значень хеш-функції, що забезпечує цілісність даних.

Transforms використовується для зміни XML-документа перед обчисленням підпису. Це може включати видалення або додавання елементів, перетворення значень атрибутів або заміну частин документа. Transforms забезпечує гнучкість управління даними, які включаються в підпис, і дозволяє забезпечити конфіденційність інформації.

1.6. Фреймворк .NET5

.NET представляє собою вільну крос-платформену систему з відкритим кодом для розробки інноваційних, масштабованих та ефективних настільних, веб-, хмарних та мобільних програм. Нинішня версія .NET - це .NET 5.0, яка прийшла на зміну .NET Core 3.1 та .NET Framework 4.6. До появи .NET 5.0 існували дві варіанти - .NET Framework і .NET Core, але вони об'єдналися в одну версію в .NET 5.0. .NET тепер виступає як уніфікована платформа для розробки додатків для ПК, веб, хмарних сервісів, мобільних пристроїв, ігор, IoT і AI (рис. 1.1). Екосистема .NET включає єдину стандартну бібліотеку, оточення виконання, компілятори мов і інструменти.

Рис. 2.3 Платформи, що підтримуються .NET

У минулому .NET Framework працював тільки на пристроях Windows. Проекти Xamarin і Mono займалися тим, щоб розширити .NET до мобільних пристроїв, macOS і Linux. .NET Core надає стандартну базову бібліотеку, яку тепер можна використовувати на Windows, Linux, macOS та мобільних пристроях (через Xamarin).

Архітектура .NET має чотири ключові елементи:

– Загальна специфікація мови (CLS) визначає реалізацію об'єктів для універсальності в .NET. CLS є підмножиною загальної системи типів (Common Type System - CTS), яка створює універсальний метод опису всіх типів.

– Бібліотека класів Framework (FCL) - це основна бібліотека, яка містить багатократно використовувані класи, інтерфейси і типи значень.
– Загальномовне середовище виконання (CLR) - це віртуальна машина, що запускає і керує виконанням програм .NET.
– Інструменти, як-то Visual Studio, для розробки різних програм, інтерактивних веб-сайтів, веб-програм та веб-служб. Розробники використовують .NET framework для створення настільних програм Windows і серверних програм. Вони також використовують ASP.NET для веб-програм. .NET Core застосовується для створення серверних програм, що працюють на Windows, Linux і Mac. Він наразі не підтримує створення настільних програм з користувальницьким інтерфейсом. Розробники можуть писати програми і бібліотеки у VB.NET, C # і F # в обох робочих середовищах. Тому, як випливає з вищенаведеного, для розробки кросплатформного ПЗ вибір слід зробити на користь .NET Core, оскільки .NET Framework підтримується тільки на ОС Windows.

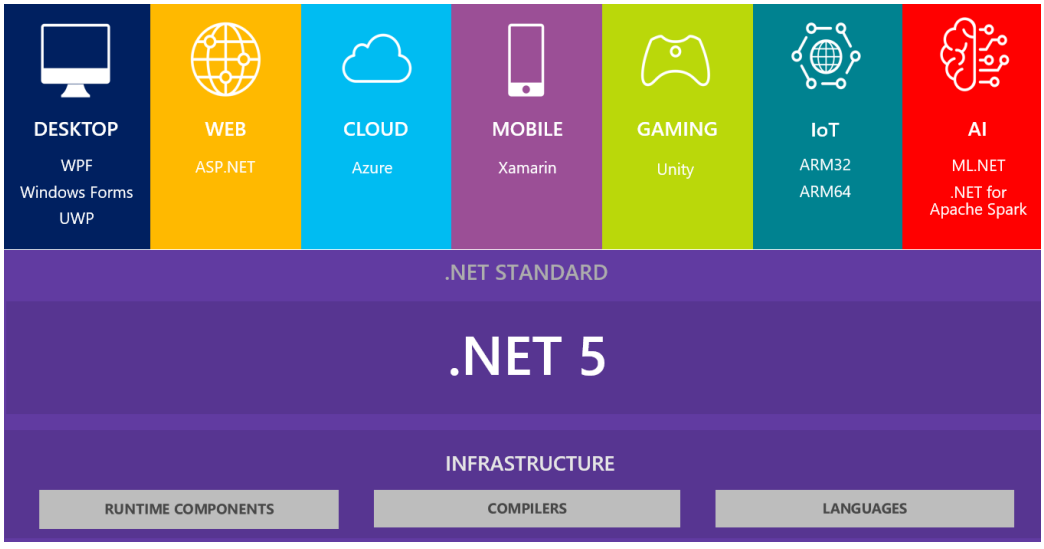

Рисунок 1.1 — Платформи які підтримує .NET

1.7. Application Programming Interface

API - це сукупність відомих дефініцій та протоколів, що використовуються для створення та вбудови програмних додатків. Це абревіатура від "інтерфейс програмування додатків". API визначає способи комунікації між різними компонентами програми, дозволяючи одному додатку використовувати функціональні можливості іншого.

API може бути використаний для реалізації різних сценаріїв взаємодії між програмами. Наприклад, веб-додаток може використовувати API соціальної мережі для авторизації користувачів або отримання інформації з їхніх профілів. API також використовується для інтеграції різних сервісів, наприклад, платіжних шлюзів або геолокаційних сервісів.

API може бути представлений у різних форматах, таких як REST [4], SOAP або GraphQL. REST (Representational State Transfer) [4] є одним з найпоширеніших стандартів для реалізації API. У REST API ресурси, такі як дані або функції, ідентифікуються за допомогою унікальних URL-адрес, і можна використовувати HTTP-методи (GET, POST, PUT, DELETE) для взаємодії з ними.

API розширює можливості програмування, дозволяючи розробникам використовувати функціонал інших додатків без необхідності реалізації його заново. Це пришвидшує процес розробки програмного забезпечення, забезпечує зручну взаємодію між різними системами та підвищує переносимість програм.

У сучасному світі API відіграють ключову роль в розробці програмного забезпечення. Вони дозволяють підключати сторонні сервіси, спілкуватися з іншими системами та забезпечують інтероперабельність між різними програмними компонентами. Зростання використання хмарних технологій та мікросервісної архітектури також збільшує значення API як засобу комунікації між окремими компонентами системи.

Усього використовуючи API, розробники отримують широкі можливості для побудови взаємодіючих та розширюваних програмних додатків. Завдяки API сучасні програми стають більш функціональними та інтегрованими, що сприяє покращенню якості та ефективності розробки програмного забезпечення.

1.8. Протокол обміну повідомленнями SOAP

SOAP (Simple Object Access Protocol) - це протокол обміну повідомленнями, який використовується для комунікації між компонентами програмного забезпечення через мережу. SOAP був розроблений з метою забезпечення стандартизованого способу обміну даними між різними платформами та мовами програмування.

Протокол SOAP базується на XML (Extensible Markup Language) і використовується для кодування повідомлень. SOAP-повідомлення складаються зі заголовків та тіла, де заголовки містять метадані, такі як ідентифікатори, параметри безпеки та інші додаткові відомості, а тіло містить самі дані. Використовуючи SOAP, клієнтська програма може зробити запит до сервера, викликати віддалені процедури та отримувати відповіді.

Одна з ключових переваг SOAP полягає в його незалежності від платформи та мови програмування. SOAP може використовуватись на різних платформах, таких як Windows, Linux або Mac, та забезпечує інтероперабельність між різними системами. Він підтримує використання різних протоколів передачі даних, таких як HTTP, SMTP або JMS, що дозволяє йому працювати в різних мережевих середовищах.

XSD (XML Schema Definition) - це мова для визначення структури та типів даних в XML. XSD використовується для визначення схеми даних, яка описує, які елементи та атрибути можуть міститися в XML-документі, як вони впорядковані та які значення можуть мати.

XSD надає можливість задавати обмеження на дані, такі як типи даних (наприклад, цілі числа, рядки або дати), обов'язковість елементів, унікальність значень та інші правила валідації. Використання XSD дозволяє створювати XML-документи, які відповідають певним правилам і можуть бути перевірені на валідність.

XSD є стандартом, що використовується в SOAP для опису структури повідомлень. Використовуючи XSD, можна визначити, які елементи повинні міститися в SOAP-повідомленнях, які типи даних мають вони мати та які обмеження повинні бути задоволені.

У підсумку, SOAP є протоколом обміну повідомленнями, який дозволяє стандартизовано обмінюватися даними між компонентами програмного забезпечення на різних платформах. Використання XSD дозволяє визначати структуру та типи даних в XML, що використовуються в SOAP-повідомленнях, забезпечуючи їх валідацію та стандартизацію.

РОЗДІЛ 2
ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ ДЛЯ ІНТЕГРАЦІЇ АБС З ЦОСЕП

Реалізований веб-сервіс дозволяє фінансовим установам інтегрувати автоматизовану банківську систему з центром оброблення системи електронних платежів розміщеним у центральній розрахунковій палаті. Форматом обміну даними з сервісом є XML. Веб-сервіс відповідає за автоматичне формування XML повідомлень, обробки платіжних та інформаційних інструкцій, а також транспортування повідомлень. Для розробки використовувався програмний продукт Visual Studio Code, .NET5 для написання сервісу для запуску джобів, TypeScript [5] та Angular [7] для побудови користувацького інтерфейсу.

2.1. Опис структури програмного забезпечення

Враховуючі всі вище описані вимоги, було розроблено архітектуру проекту (рис. 2.1).

Було обрано найбільш оптимальні засоби розробки, а саме:

- Visual Studio Code – середовище розробки програмних застосунків;
- PostMan – для тестування HTTP запитів;
- Angular – фрейморк для розробки веб-додатків користувацького інтерфейсу.
- HTML5 – мова розмітки;
- .NET5– для роботи автоматизованих джобів.

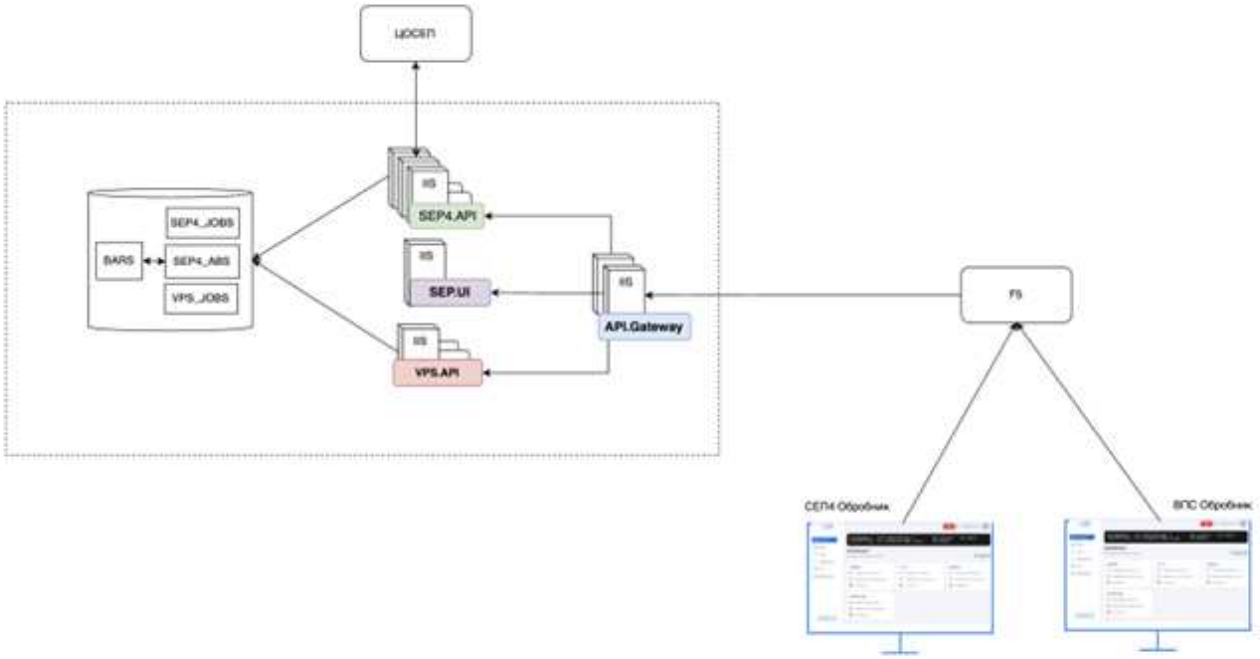

Рисунок 2.1 — Архітектура розроблених веб-сервісів

2.2. Структура автоматизованого веб-сервісу

Головний клас, який буде точкою входу в проект, і самі ці методи будуть визиватися при запуску сервісу називається Startup (Додаток А).

В цьому класі знаходяться головні методи для конфігурації сервісу, такі як:

1. Конфігурація середовища: Клас Startup дозволяє налаштувати середовище для додатка. Це може включати в себе встановлення режиму розробки або режиму виробництва, налаштування доступу до бази даних, зовнішніх сервісів і інше.
2. Налаштування сервісів: Клас Startup дозволяє реєструвати сервіси, які будуть використовуватися в додатку. Це можуть бути сервіси баз даних, логування, автентифікації, авторизації, кешування та інші.
3. Конфігурація маршрутизації: Клас Startup дозволяє визначати маршрути для обробки HTTP-запитів у додатку. Ви можете налаштувати

маршрутизацію до різних контролерів або дій (actions), які обробляють запити.

4. Конфігурація інших налаштувань: Клас Startup також використовується для налаштування інших аспектів додатка, таких як конфігураційні параметри, настройки безпеки, обробка помилок та інше.

Метод для реєстрації автоматизованих джобів (рисунок 2.2).

```
public static partial class StartupExtensions
{
    private static void AddJob<T>(string cron) where T : IProcess
    {
        ProcessManager.AddRecurrentJobProcess<T>(cron);
    }

    public static void RegisterProcessServices(this IServiceCollection services)
    {
        services.AddScoped<SepParticipantsSyncProcess>();
        services.AddScoped<InterbankCreditTransfersShowCaseProcess>();
        services.AddScoped<AccountMessagesProcess>();
        services.AddScoped<CreditorPaymentActivationPacketsProcess>();
        services.AddScoped<CreditorPaymentActivationReportsProcess>();
        services.AddScoped<FiToFiPaymentCancellationsProcess>();
        services.AddScoped<InterbankCreditTransferPacketsSyncProcess>();
        services.AddScoped<InvestigationPacketsProcess>();
        services.AddScoped<LimitsRequestsProcess>();
        services.AddScoped<SecuritiesPacketsProcess>();
        services.AddScoped<SepQueueProcess>();
        services.AddScoped<SepMessagesProcess>();
        services.AddScoped<FillTransfersQueueProcess>();
        services.AddScoped<ResendSepRequestsProcess>();
        services.AddScoped<StaticDataRequestsProcess>();
        services.AddScoped<UnableToApplyPacketsProcess>();
        services.AddScoped<RequestToModifyPaymentsProcess>();
        services.AddScoped<ClaimNonReceiptProcess>();
        services.AddScoped<ArchivePacketsToStorageProcess>();
    }
}
```

Рисунок 2.2 — Клас Startup Extensions

2.3. Інструкція з використання програмного забезпечення

Головний початковий екран інтерфейсу користувача СЕП Обробника зображено на рисунку 1. Він складається з трьох частин: «Головне меню», «Заголовок. Загальна інформація», «Головне інформаційне вікно». (рис. 2.3).

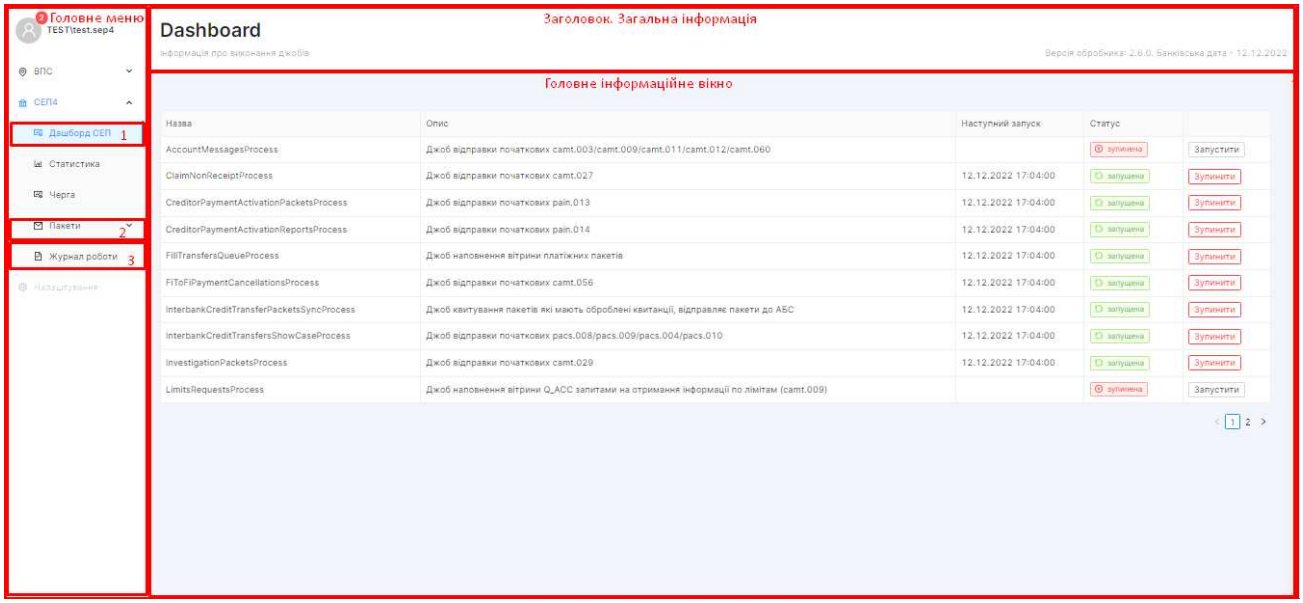

Рисунок 2.3 — Початкова сторінка користувацького інтерфейсу

Вкладка “Dashboard” необхідна для зупинки та запуску джобів, для управління обробкою пакетів у СЕП обробнику. За замовчуванням, всі джоби обробника вимкнені під час запуску серверу. Їх статус зберігається у БД. Для запуску обробника лише у режимі прийому вхідних пакетів потрібно запустити відповідні джоби. Для кожної джоби є опис її роботи в таблиці інтерфейсу. (рис. 2.4).

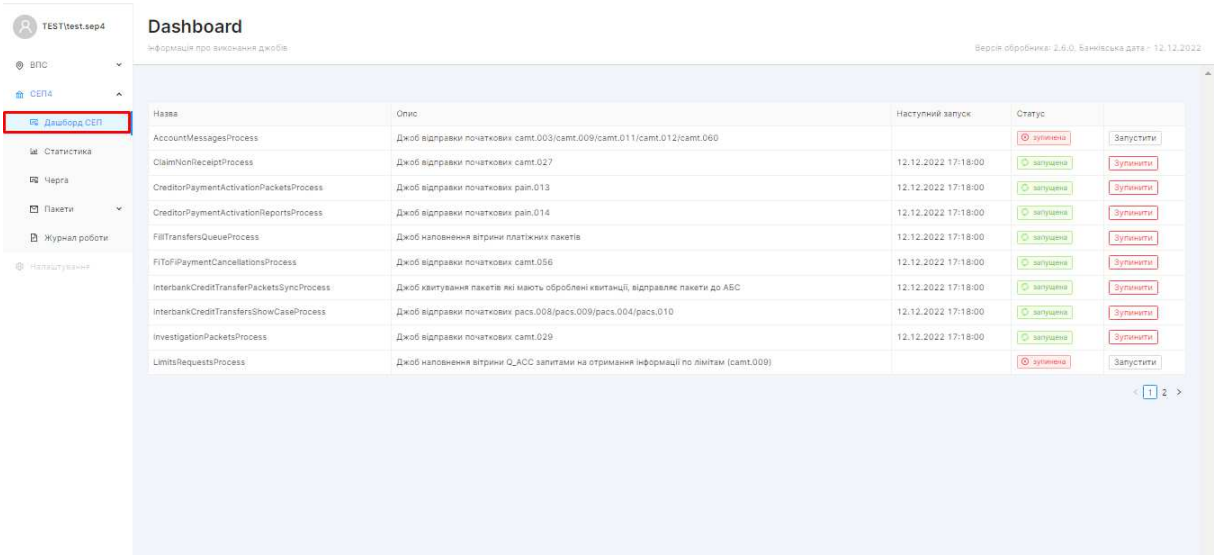

Рисунок 2.4 — Зображення вкладки Dashboard

Після того як користувач запустить процесі, система розпочне автоматичне формування, відправку, отримання та обробку повідомлень. Статус обробки пакетів можна переглянути на відповідних вкладках інтерфейсу користувача (рис. 2.5).

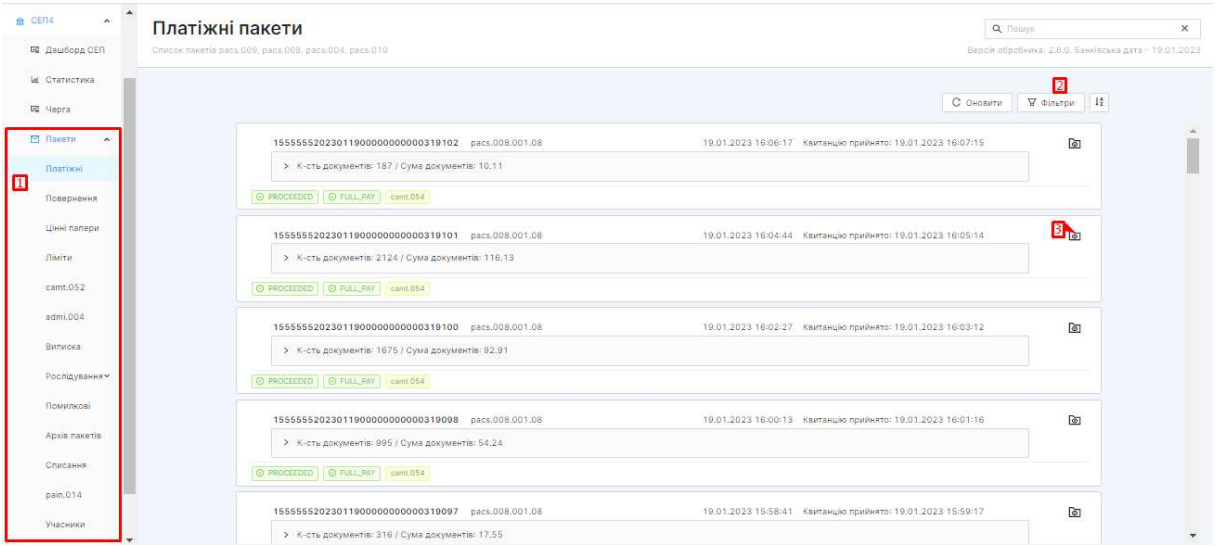

Рисунок 2.5 — Пункти меню

Частини екрану, склад інформації та доступні функції її обробки відповідають функціям, виконання яких вибрав користувач. Блоки в Dashboard показують Тип пакетів та інформацію про пакети цього типу.

Є кнопка для перегляду інформації про пакет, при натисканні на неї, з'явиться інформація про пакет з можливістю завантаження (рис 2.6).

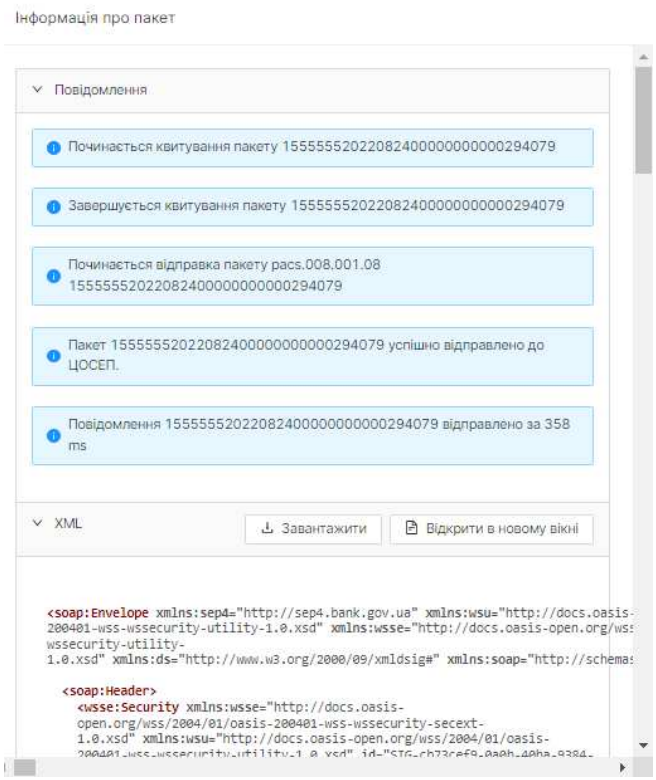

Рисунок 2.6 — Перегляд інформації про пакет

2.4. Загальний опис процесу формування та квитування початкових:

Документ отримує останню візу виконавця в ЦА або надійшов з іншого РУ та очікує відправки до СЕП. При цьому у відповідній таблиці ARC_RRP виставляється ознака (поле PRTY), яка вказує на тип пакету, що буде відправлено на ЦОСЕП.

Відповідний процес для ЦА Обробника ВПС виконуючи процедуру БД, сканує документи, які потребують відправки до СЕП. Групує такі документи у пакети (аналогічно до файлів) по типу пакета, по МФО відправника та МФО отримувача. Такі пакети можна переглянути у функції ‘СЕП. Інформація про файли’ на рівні ЦА. Для сумісності з СЕП 3.2 таким пакетам буде надане ім’я afqke. Якщо документ не пройшов процес перевірки реквізитів/ецп був відбракований або підпадає під умови встановлених бізнес правила - такий документ блокується та може бути переглянутий у функції ‘СЕП. Блоковані документи’

Відповідний джоб Обробника СЕП-4 сканує документи в ММФО, що вже об’єднані в пакети та виконує заповнення відповідних вітрин (в залежності від типу пакету - виконується наповнення відповідних вітрин). При цьому пакет в ММФО вважається відправленим, створюється документ на суму пакету операцією R01 та виконується наступна проводка: списання з загального транзиту та зарахування на транзит насквитованих сум: Дб(3739/T00) - Кр (3739/TNB). В обробнику можна переглянути такий пакет в Платіжних із статусом ‘INSERTED’.Відповідний джоб обробника бере на обробку такий пакет для подальшої відправки. В обробнику такий пакет буде мати статус ‘GOT2SEP’

Надалі обробник формує XML пакет, підписує його накладає на нього КЕП та направляє до ЦОСЕП. В обробнику такий пакет буде мати статус ‘POST2SEP’.

Документ отримує останню візу виконавця в ЦА або надійшов з іншого РУ та очікує відправки до СЕП. При цьому у відповідній таблиці ARC_RRP виставляється признак (поле PRTY), який вказує на тип пакету, що буде відправлено на ЦОСЕП.

Відповідний джоб для ЦА Обробника ВПС виконуючи процедуру БД, сканує документи, які потребують відправки до СЕП. Групує такі документи у пакети (аналогічно до файлів) по типу пакета, по МФО відправника та МФО отримувача. Такі пакети можна переглянути у функції ‘СЕП. Інформація про файли’ на рівні ЦА. Для сумісності з СЕП 3.2 таким пакетам буде надане ім’я afqke. Якщо документ не пройшов бізнес правила - такий документ блокується та може бути переглянутий у функції ‘СЕП. Блоковані документи’.

Відповідний процес Обробника СЕП-4 сканує документи в ММФО, що вже об’єднані в пакети та виконує заповнення відповідних вітрин (в залежності від типу пакету - виконується наповнення відповідних вітрин). При цьому пакет в ММФО вважається відправленим, створюється документ на суму пакету операцією R01 та виконується наступна проводка: списання з загального транзиту та зарахування на транзит насквитованих сум:

Дб(3739/T00) - Кр (3739/TNB). В обробнику можна переглянути такий пакет в Платіжних із статусом ‘INSERTED’.

Відповідний процес обробника бере на обробку такий пакет для подальшої відправки. В обробнику такий пакет буде мати статус ‘GOT2SEP’

Надалі обробник формує XML пакет, підписує його та направляє до ЦОСЕП. В обробнику такий пакет буде мати статус ‘POST2SEP’

Незалежно від відправки початкових, інший джоб обробника сканує вхідну чергу від ЦОСЕП.

Якщо з ЦОСЕП надійшла квитанція camt.054, яка:

- містить повний перелік всіх документів, тоді початкови пакет квитується. Для цього Обробник визиває відповідні процедуру БД та для відповідного документу, що було створено при надсиланні первинного пакету додається проводка з типом операції RT0: списання з транзиту незаквитованих сум та зарахування на коррахунок банку: Дб(3739/TNB)- Кр(1200/N00). Після виконання квитовки пакет набуває статусу ‘PROCEDED’ з типом оплати ‘FULL_PAY’. Сквитований початковий пакет можна переглянути у функції ‘СЕП. Інформація про файли’
- містить частину платежів первинного пакету, обробник буде очікувати надходження camt.002, який надасть перелік відбракованих документів з кодами помилок. Коли отримано від ЦОСЕП пакет camt.002 - відбувається часткова квитовка пакету, а саме:
    - змінюється сума та склад первинного пакету (це видно у функції ‘СЕП Інформація про файли’). Сума та склад документів пакету тепер дорівнює сумі та складу квитанції camt.054.
    - для відповідного документу, що було створено при надсиланні первинного пакету додається проводка з типом операції RT0 на суму тільки підтверджених документів. списання з транзиту незаквитованих сум та зарахування на коррахунок банку: Дб(3739/TNB)- Кр(1200/N00). На суму

не підтверджених документів формується документ з проводкою: списання з транзиту незаквитованих на загальний транзит: Дб(3739/TNB)- Кр(3739/T00). Всі забраковані докумнети попадають у заблоковані із відповідним з НБУ кодом помилки. Такі документи можна переглянути в функції ‘СЕП. Заблоковані документи’

  - Після виконання квитовки пакет набуває статусу ‘PROCEDED’ з типом оплати ‘PART_PAY’. Сквитований початковий пакет можна переглянути у функції ‘СЕП. Інформація про файли’. Важливо. що такий пакет буде мати тільки документи, що підтверджені НБУ

- квитанція camt.054 не прийшла, а прийшла тільки відбійна квитанція camt.002 з кодом помилки на весь пакет. В такому випадку в ММФО пакет сторнується та до основного документу, що було створено при відправці пакету додається проводка: списання з транзиту незаквитованих та зарахування на загальний транзит на суму пакету Дб(3739/TNB) - Кр(3739/T00). Всі документи пакету будуть забраковані з відповідним з НБУ кодом. Пакет набуває статусу ‘PROCEDED’ з типом оплати ‘FULL_STORNO’.

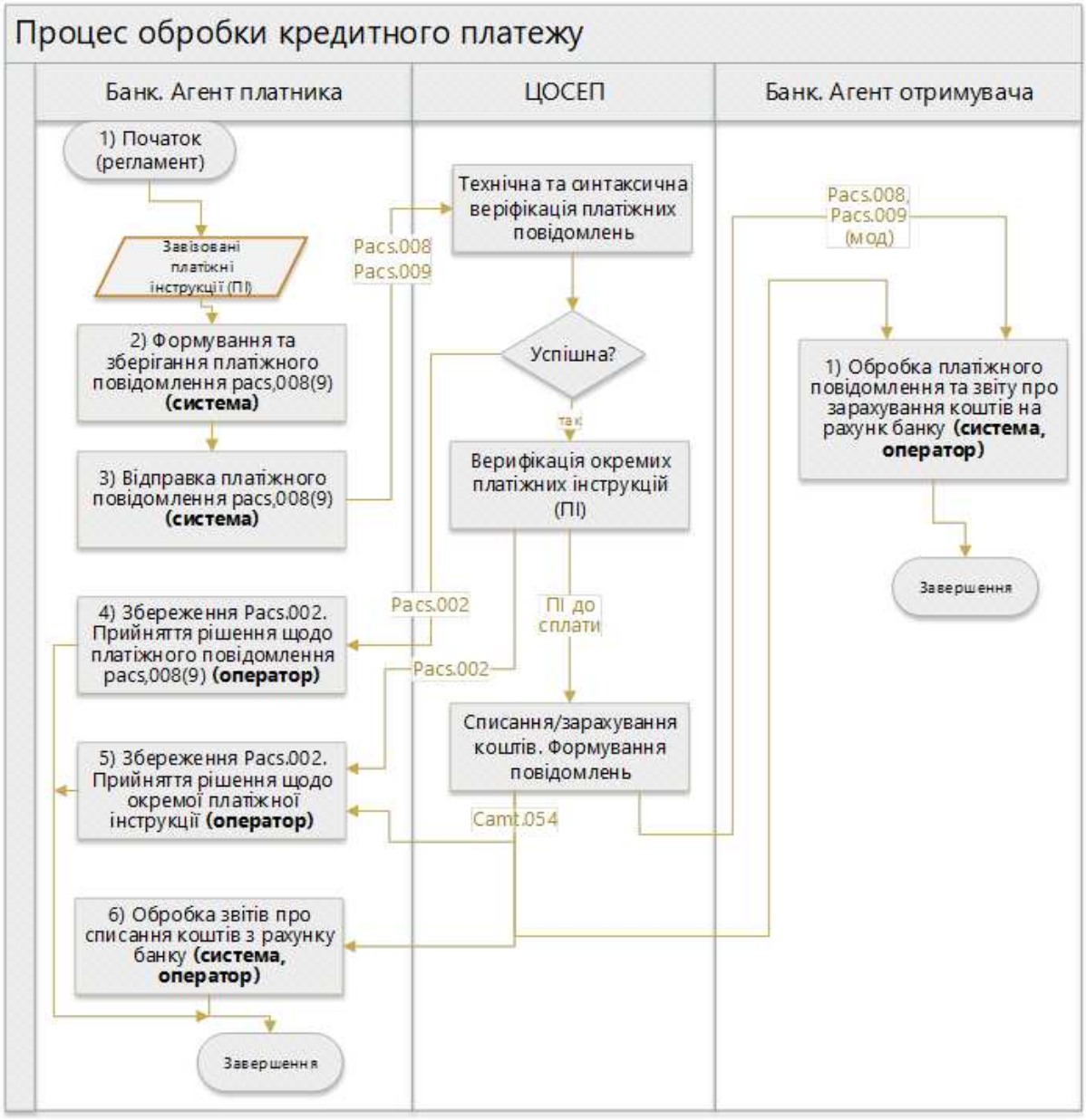

Рисунок 2.7 — Процес обробки кредитного платежу

Процес “Розслідування” використовується для виявлення причин помилок, що виникають при сплаті коштів:

- у разі неможливості виконання платежу - використовується повідомлення camt.026 “Запит у зв’язку з неможливістю виконання “;
- у разі неотримання платежу - використовується повідомлення camt.027 “Запит про неотримання коштів“;

Даний процес відбувається за умов переказу ЦОСЕП коштів між рахунками Агентів Платника та Отримувача.

У випадку проведення розслідування при неможливості виконання платежу Агент отримувача ініціює обмін відповідними повідомленнями, загальна схема якого зображена на рисунках нижче

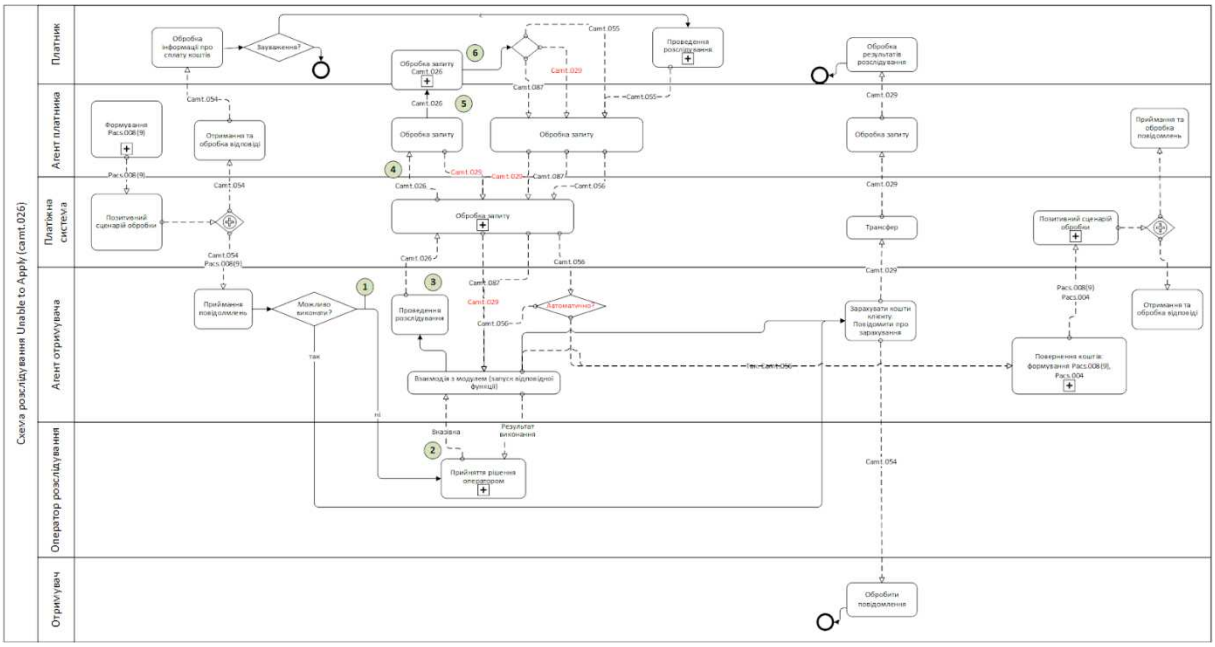

Рисунок 2.8 — Процес обробки пакетів розслідувань

Camt.026 відправляється Агентом Отримувача Агенту Платника вразі «неможливість виконання платіжного доручення». Термін відправки не може перевищувати час зберігання первинного платіжного повідомлення.

Визначено, що виконання окремої транзакції неможливо, за умови що всі попередні перевірки самого повідомлення дали позитивний результат. Рішення про проведення розслідування приймається відповідальним оператором.

Рішення про проведення розслідування приймається відповідальним оператором. Він вручну вказує на формування запиту «неможливо виконати платіжне доручення» та зазначає причину або причини проведення розслідування. Для однієї транзакції може бути вказано декілька причин, що обґрунтовують запит.

Комплекс для кожної транзакції формує окремий camt.026 та направляє до ЦОСЕП. Ланцюг передачі camt.026 має бути таким самим, за яким прийшло первинне платіжне повідомлення, транзакцію якого направлено на розслідування.

ЦОСЕП/Посередник у разі успішної перевірки повідомлення camt.026 переправляє його далі по вказаному ланцюгу без змін та обробки.

При цьому ЦОСЕП/Посередник здійснює пошук у своїй БД:

- первинного повідомлення, якого стосується даний camt.026, за вказаним в camt.026 реквізитом Ідентифікатор оригінального повідомлення (Original Message Identification) <OrgnlMsgId>
- трансакції з первинного повідомлення за вказаним в camt.026 реквізитом UETR оригінальної трансакції (Original UETR) <OrgnlUETR>
- та перевіряє приналежність ідентифікаторів Наскрізний ідентифікатор оригінальної трансакції (Original End To End Identification) <OrgnlEndToEndId> та UETR оригінальної трансакції (Original UETR) <OrgnlUETR> до однієї і тієї ж трансакції з первинного повідомлення, щодо якої сформований camt.026
- та перевіряє відповідність значення реквізиту “Сума міжбанківського переказу” (оригінальної платіжної інструкції) (Original Interbank Settlement Amount) <OrgnlIntrBkSttlmAmt> в camt.026 до значення реквізиту Сума міжбанківського переказу (Interbank Settlement Amount) <IntrBkSttlmAmt> платіжного повідомлення.

Агент-платника перевіряє camt.026 за правилами ЦОСЕП/Посередник. У разі негативного результату перевірки формує повідомлення про відхилення camt.029.

Інакше формує та направляє одне з повідомлень:

- camt.056 на відкликання транзакції
- camt.087 на уточнення реквізитів
- не відповідає (наприклад, у разі відкликання платіжки раніше).

У випадку проведення розслідування при неотриманні коштів, Агент платника ініціює обмін відповідними повідомленнями, загальна схема якого зображена на рисунках нижче. Camt.027 передається за тим самим маршрутом,

що і первинне платіжне повідомлення - від Інструктуючого Агента до Проінструктованого.

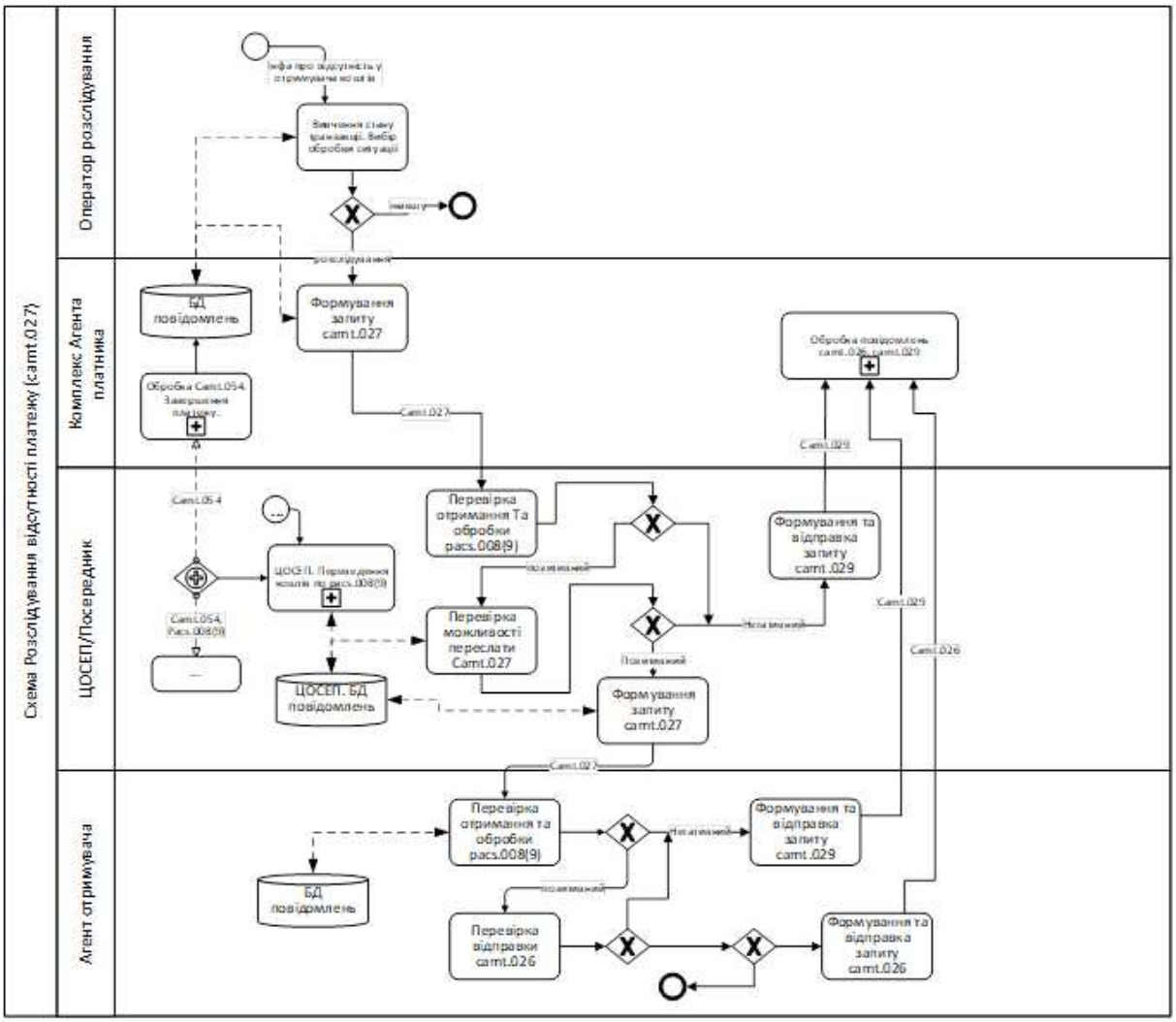

Рисунок 2.9 — Схема розслідування відсутності платежу

Рішення про проведення розслідування приймається відповідальним оператором. Він, після вивчення ситуації вибирає «розпочати розслідування про неотримання коштів»;

Комплекс формує camt.027 окремо для вибраної транзакції та направляє до ЦОСЕП/Посередника. Дотримується правила передачі – направити до наступного проінструктованого агента pacs.008(9), що містив неотриману транзакцію;

ЦОСЕП/Посередник здійснює у своїй БД пошук:

- первинного повідомлення, якого стосується даний camt.027, за вказаним в camt.027 реквізитом Ідентифікатор оригінального повідомлення (Original Message Identification) <OrgnlMsgId>
- трансакції з первинного повідомлення за вказаним в camt.027 реквізитом UETR оригінальної трансакції (Original UETR) <OrgnlUETR>
- перевіряє приналежність ідентифікаторів “Наскрізний ідентифікатор оригінальної трансакції (Original End To End Identification) <OrgnlEndToEndId> та UETR оригінальної трансакції (Original UETR) <OrgnlUETR> до однієї і тієї ж трансакції з первинного повідомлення, щодо якої сформований запит
- та перевіряє відповідність значення реквізиту Сума міжбанківського переказу (оригінальної платіжної інструкції) (Original Interbank Settlement Amount) <OrgnlIntrBkSttlmAmt> в camt.027 до значення реквізиту Сума міжбанківського переказу (Interbank Settlement Amount) <IntrBkSttlmAmt> платіжного повідомлення.
- перевіряє можливість відправки camt.027 проінструктованому агентові з відповідного первинного платіжного повідомлення

У разі успішної перевірки формує camt.027 та пересилає його проінструктованому агенту.

У разі не проходження перевірки, неможливості передати далі по ланцюгу – повідомлення camt.027 відхиляється з направленням його відправнику повідомлення camt.029. За такого випадку, Агент посередник/ЦОСЕП, який формує camt.029, зобов’язаний вказати свою ідентифікацію в реквізиті Учасник, який встановив статус щодо запиту (Originator) повідомлення camt.029 з зазначенням причини відхилення запиту (вказується із довідника № 77 ISO 20022 [8] ExternalPaymentCancellation Rejection1Code у відповідному реквізиті camt.029) (детальний опис в Специфікації camt.029). Доручення – ідентифікує повідомлення та сторону відправника і отримувача camt.029:

- Вирішена справа (Resolved Case) - копіюється значення відповідних реквізитів, що були зазначені в повідомленні camt.027, у блоці Справа (Case)

– Статус (Status) – в реквізиті нижчого рівня Підтвердження (Confirmation) зазначається код RJCR
– Детальні дані щодо запиту (Cancellation Details) – зазначаються ідентифікатори платіжного повідомлення та оригінальної трансакції, про яку було отримано запит camt.027, ідентифікатори запиту та зазначається причина відхилення запиту camt.027.
– Агент отримувача проводить перевірку camt.027 за схемою ЦОСЕП/Посередник, виключаючи перевірку можливості відправки camt.027 проінструктованому агенту.

У разі неуспіху готує та відправляє camt.029.

У разі успішної перевірки формує оператору повідомлення, для вибору можливих варіантів дій:

– підготувати camt.029
– підготувати camt.026
– провести транзакцію
– не реагувати на запит, якщо відповідний camt.026 був вже відправлений.

Повідомлення camt.029 формується Агентом платника/Агентом отримувача/Посередником та надсилається через СЕП та посередників (за наявності їх в ланцюгу обміну повідомленням) у відповідь на запит camt.026/camt.027/camt.056/camt.087 (далі-запит) з метою надання інформації, щодо статусу запиту. Camt.029 формується тільки у разі відхилення зазначених запитів.

Не допускається формування camt.029 з учасниками, відмінними від задіяних у ланцюгу обміну первинним платіжним повідомленням pacs.008/pacs.009, щодо якого сформовано запит (тобто, задіяні мають бути учасники, вказані відправником у первинному повідомленні). За відсутності можливості дотримання даної умови – використання camt.029 неприпустиме. Якщо немає можливості передати далі camt.029, проінструктований агент повертає camt.025 із зазначенням причин. Дана вимога не висувається лише у

випадку, коли camt.029 сформовано у відповідь на camt.056, щодо відкликання повідомлення pain.013. Повідомлення camt.029 (у відповідь на camt.056, щодо повідомлення pain.013) направляється від Агента платника /ЦОСЕП/Посередника до Агента Стягувача/Отримувача.

ВИСНОВКИ

У рамках даної дипломної роботи було проведено дослідження та розроблено універсальне інтеграційне рішення, спрямоване на побудову інтеграції Системи автоматизованого банківського обслуговування (САБ) з Центральною обробною системою електронних платежів (ЦОСЕП) в рамках реалізації Системи електронних платежів 4-го покоління (СЕП-4).

Під час аналізу сучасного стану інтеграції було виявлено проблеми, пов'язані з розділеністю інформаційних систем, що часто перешкоджають ефективній взаємодії банківських систем та імплементації інноваційних рішень. З метою подолання цих проблем, було розроблено універсальне інтеграційне рішення, яке може використовуватися з різними САБ системами українських банків.

Результати дослідження показали, що розроблене рішення дозволяє покращити швидкість та якість обробки платежів, знижує час реакції на зміни в системах банку і сприяє поліпшенню загального клієнтського досвіду. Розроблений веб-сервіс було інтегровано та впроваджено для використання в промисловому середовищі Державного Ощадного Банку України .

Завдяки розробленому рішенню банки зможуть підвищити свою ефективність та конкурентоспроможність на ринку фінансових послуг, забезпечуючи швидку та безперервну обробку платежів, зменшення ризиків і підвищення рівня надійності своїх послуг. Крім того, розроблене рішення може бути використаною українськими банками з різними САБ системами, що робить його універсальним та масштабованим у контексті банківської індустрії.

Розроблений в дипломній роботі веб-сервіс було інтегровано та впроваджено для використання в промисловому середовищі Державного Ощадного Банку України .

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Powers S. Learning Node / S. Powers. – S.: O`Reilly Media, 2012 – 374 с.

2. CLR via C#. Richter Jeffrey, 2012 – 896 с.

4. REST [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/REST

5. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript

6. User Interface style sheet language [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/User_interface_style_sheet_language

7. Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/

8. ISO 20022 [Електронний ресурс] – Режим доступу до ресурсу: https://bank.gov.ua/ua/payments/project-iso20022

ДОДАТКИ

ДОДАТОК А

Реалізація класу Startup для запуску веб-додатку

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        var oracleSettings = OracleDatabaseExtension.
             BuildOracleDatabaseSettings(Configuration);
        services.RegisterOracleDbContext < ShowCaseDbContext > (oracleSettings, "11");
        services.RegisterOracleDbContext < XmlStorageDbContext > (oracleSettings, "11");
        services.RegisterOracleDbContext < SepPacketsDbContext > (oracleSettings, "11");
        services.RegisterOracleDbContext < SepJobsDbContext > (oracleSettings, "11");
        services.RegisterConfigurations();
        services.RegisterLogging(Configuration);
        services.RegisterAutoMapper();
        services.RegisterMediatR();
        services.AddMemoryCache();
        services.RegisterProcessServices();
        services.RegisterPacketConsumerServices();
        services.RegisterServices();
        var appSettings = new AppSettings();
        Configuration.GetSection("AppSettings").Bind(appSettings);
        var jobsSettings = new JobsSettings();
        Configuration.GetSection("JobsSettings").Bind(jobsSettings);
        services.AddHangfire(x => x.UseStorage(new OracleStorage(
            () => new OracleConnection(appSettings.JobsDbConnectionString), new
OracleStorageOptions
            {
                SchemaName = "SEP4_JOBS",
                    JobExpirationCheckInterval = TimeSpan.FromDays(1)
            })));
        services.AddHangfireServer(opt =>
        {
            opt.WorkerCount = jobsSettings?.WorkersCount ?? 10;
            if (jobsSettings.EnabledQueues == null || jobsSettings.EnabledQueues.Count == 0)
            {
                opt.Queues = new []
                {
                    "receipts",
                    "send_transfer_packets",
                    "income_packets",
                    "fetch_income_packets_from_sep",
                    "default"
                };
            }
            else
            {
                opt.Queues = jobsSettings.EnabledQueues.Select(v => v).ToArray();
```

```
            }
        });


        services.AddOData();
        services.AddControllers().AddNewtonsoftJson(options =>
        {
            options.SerializerSettings.ContractResolver = new DefaultContractResolver();
        });
    }
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        app.UseRouting();
        app.UseMiddleware < ExceptionHandlingMiddleware > ();
        app.UseMiddleware < HttpContextDataMiddleware > ();
        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
            endpoints.EnableDependencyInjection();
            endpoints.Select().Filter().Expand().Count().OrderBy().MaxTop(100);
        });
        var options = new DashboardOptions
        {
            AppPath = null,
                Authorization = Array.Empty < IDashboardAuthorizationFilter > ()
        };
        app.UseHangfireDashboard("/sep/hangfire/dashboard", options);
        GlobalJobFilters.Filters.Add(new AutomaticRetryAttribute
        {
            Attempts = 0, OnAttemptsExceeded = AttemptsExceededAction.Delete
        });
        JobStorage.Current.JobExpirationTimeout = TimeSpan.FromDays(1);
        var message = $ "Starting api server v{ApiVersion.Version}";
        BackgroundJob.Enqueue < IServiceLogger < SEP4.Startup >> (s =>
s.LogInformation(message, ""));
    }
}
```

ДОДАТОК Б

Приклад реалізації процесу для наповнення черги відправки початкових платіжних повідомлень

```
public sealed class FillTransfersQueueProcess : SepProcessBase
{
    public override async Task ExecuteAsync()
    {
        try
        {
            await EnsureAbsIsAvailable();
            await mediator.Send(new FillCreditTransfersQueueCommand());
        }
        catch (AbsUnavailableException e)
        {
            await logger.LogInformation("Наповнення вітрини неможливе через ексклюзивний
режим.", "");
        }
    }

    public FillTransfersQueueProcess(SepJobsDbContext context, AppSettings settings, IMediator
mediator,
        ShowCaseDbContext showCaseDbContext, IServiceLogger<SepProcessBase> logger) :
base(context, settings,
        mediator, showCaseDbContext, logger)
    {
    }
}
```

ДОДАТОК В

Приклад реалізації процесу для відправки початкових платіжних повідомлень

```
/// <summary>
/// - Fetches the pacs.008, pacs.009, pacs.010 packets from queue and processes them
/// </summary>
[Queue("send_transfer_packets")]
public sealed class InterbankCreditTransfersShowCaseProcess : SepProcessBase
{
    private readonly JobsSettings jobsSettings;

    public InterbankCreditTransfersShowCaseProcess(SepJobsDbContext context, AppSettings
settings,
        IMediator mediator, ShowCaseDbContext showCaseDbContext,
IServiceLogger<SepProcessBase> logger,
        JobsSettings jobsSettings) : base(
        context, settings, mediator, showCaseDbContext, logger)
    {
        this.jobsSettings = jobsSettings;
    }

    public override async Task ExecuteAsync()
    {
        var queryable = await mediator.Send(new GetCreditTransfersPacketsQuery());
        var creditTransferPacketIds = await queryable
            .Where(p =>
                p.Status == PacketStatus.INSERTED && p.PacketDirection ==
PacketDirection.Outcome)
            .Select(p => new
            {
                p.MessageType,
                p.MessageId
            })
            .Take(jobsSettings.TransferPacketsIterationLimit ?? 1000)
            .ToListAsync();

        foreach (var packet in creditTransferPacketIds)
        {
            await ProcessPacket<IInterbankCreditTransferService, CreditTransfersPacket>(s =>
                s.SendMessageToSepAsync(packet.MessageId, packet.MessageType),
packet.MessageId);
        }
    }
}
```

ДОДАТОК Г

Приклад реалізації класу для обробки платіжних повідомлень та їх інструкцій

```
public class InterbankCreditTransferPacketsService :
    SepServiceBase<InterbankCreditTransferPacketsService, CreditTransfersPacket>,
    IInterbankCreditTransferService
{
    public InterbankCreditTransferPacketsService(IMapper mapper,
        ISignService signService, ISoapService soapService,
        ShowCaseDbContext context, IServiceLogger<InterbankCreditTransferPacketsService>
logger,
        IMediator mediator) : base(mapper,
        signService, soapService, logger, mediator, context)
    {
    }

    private IRequest SendProcessedCreditTransfersPacketToAbsAsync(string messageId,
PacketDirection direction)
    {
        switch (direction)
        {
            case PacketDirection.Income:
                return new SendIncomeCreditTransfersPacketToAbsCommand(messageId);
            case PacketDirection.Outcome:
                return new SendCreditTransfersPacketToAbsCommand(messageId);
            default:
                throw new ArgumentOutOfRangeException(nameof(direction), direction, null);
        }
    }

    /// <summary>
    /// Marks the credit transfer packet as FAILED by updating the fields, setting the error
messages, and calling ABS procedure.
    /// Note: ErrorCode is set to Rejected for all transactions which are valid but are in the
packet where at least one transaction is broken.
    /// </summary>
    async Task SetPacketValidationFailedStatus(CreditTransfersPacket packet,
        ValidateCreditTransfersPacketResult result)
    {
        foreach (var transfer in packet.CreditTransfers)
        {
            var validationResult = result.TransactionsResults.First(r =>
                r.UniqueEndToEndTransactionReference ==
transfer.UniqueEndToEndTransactionReference);
            switch (validationResult.ProcessingErrorCode)
            {
                case TransactionProcessingErrorCode.InvalidSignature:
                    transfer.TransferStatusCode = CreditTransferStatus.INVALID_SIGNATURE;
                    transfer.ErrorCode = "5511";
                    transfer.ErrorInformation = validationResult.ErrorMessage;
```

```
                    break;
                default:
                    transfer.TransferStatusCode = CreditTransferStatus.REJECTED;
                    transfer.ErrorInformation = validationResult.ErrorMessage;
                    break;
            }
        }

        packet.Status = PacketStatus.FAILED;
        packet.ReceiptStatus = ReceiptStatus.FULL_STORNO;
        packet.ErrorCode = "5511";
        packet.ErrorMessage = "Невірний підпис";

        await mediator.Send(new UpdatePacketCommand<CreditTransfersPacket>(packet));
        await mediator.Send(
            SendProcessedCreditTransfersPacketToAbsAsync(packet.MessageId,
PacketDirection.Outcome));
    }

    /// <summary>
    /// Processes the Failed Report message (pacs002). There are 3 cases of response:
    /// 1. All documents were rejected, setting the status for them and packet if FULL_STORNO
    /// 2. Part pay, failed documents are set to rejected status, no packet final status
    /// 3. All documents are successfully payed, setting Payed Status and packet is FULL_PAY
    /// </summary>
    /// <param name="receiptDocument">Soap model of Failed Transfers packet report</param>
    /// <param name="documentXml">Original xml message that was received in SOAP
message</param>
    public async Task SubmitPacket(CreditTransfersPacketFailedStatusReport receiptDocument,
string documentXml)
    {
        if (receiptDocument?.FiToFiPmtStsRpt?.OrgnlGrpInfAndSts?.StsRsnInf?.Rsn?.Cd == "DU01")
        {
            return;
        }

        var messageId = receiptDocument.FiToFiPmtStsRpt.OrgnlGrpInfAndSts.OrgnlMsgId;
        var groupStatus = receiptDocument.FiToFiPmtStsRpt.OrgnlGrpInfAndSts;
        var queryable = await mediator.Send(new GetCreditTransfersPacketsQuery());
        var transfersPacket = await queryable.FirstOrDefaultAsync(p => p.MessageId ==
messageId);

        if (transfersPacket.Status == PacketStatus.PROCEEDED)
        {
            return;
        }

        var statusesInfo = new List<TransferStatusInfo>();
        var submitReceiptCommand = new SubmitCreditTransferPacketReceiptCommand(messageId);
        var commands = new List<IRequest> { submitReceiptCommand };


        submitReceiptCommand.ReceiptMessageId = receiptDocument.FiToFiPmtStsRpt.GrpHdr.MsgId;
```

```
        var transferReports = receiptDocument.FiToFiPmtStsRpt.TxInfAndSts;

        transferReports.ForEach(r =>
        {
            var errCode = r.StsRsnInf.Rsn.Cd;
            var errMessage = r.StsRsnInf.AddtlInf;
            var id = string.IsNullOrEmpty(r.OrgnlUETR)
                ? r.OrgnlEndToEndId
                : r.OrgnlUETR;
            statusesInfo.Add(new TransferStatusInfo(id, errMessage, errCode));
        });

        switch (groupStatus.GrpSts)
        {
            case "RJCT":
                submitReceiptCommand.TotalFailedTransfersAmount =
groupStatus.OrgnlCtrlSum.Value;
                submitReceiptCommand.FailedTransfersCount = groupStatus.OrgnlNbOfTxs;
                submitReceiptCommand.ErrorMessage = receiptDocument.FiToFiPmtStsRpt?
                    .OrgnlGrpInfAndSts.StsRsnInf?.AddtlInf;
                submitReceiptCommand.ErrorCode = receiptDocument.FiToFiPmtStsRpt?
                    .OrgnlGrpInfAndSts?.StsRsnInf?.Rsn?.Cd;
                submitReceiptCommand.ReceiptStatus = ReceiptStatus.FULL_STORNO;

                commands.Add(new SetCreditTransfersStatusCommand(statusesInfo, messageId,
                    transfersPacket.MessageType,
                    CreditTransferStatus.REJECTED, true));
                commands.Add(new SetAllCreditTransfersStatusCommand(messageId,
                    transfersPacket.MessageType, CreditTransferStatus.REJECTED));
                break;
            case "ACSC":
                break;
            case "PART":
                var rejectedTransfersInfo = receiptDocument.FiToFiPmtStsRpt.OrgnlGrpInfAndSts
                    .NbOfTxsPerSts.First(r => r.DtldSts == "RJCT");
                submitReceiptCommand.TotalFailedTransfersAmount =
rejectedTransfersInfo.DtldCtrlSum.Value;
                submitReceiptCommand.FailedTransfersCount = rejectedTransfersInfo.DtldNbOfTxs;

                if (rejectedTransfersInfo.DtldNbOfTxs == transfersPacket.TransactionsCount)
                {
                    submitReceiptCommand.ReceiptStatus = ReceiptStatus.FULL_STORNO;
                    submitReceiptCommand.ErrorMessage = receiptDocument?.FiToFiPmtStsRpt
                        .OrgnlGrpInfAndSts?.StsRsnInf?.AddtlInf;
                    submitReceiptCommand.ErrorCode = receiptDocument.FiToFiPmtStsRpt?
                        .OrgnlGrpInfAndSts?.StsRsnInf?.Rsn?.Cd;
                }

                commands.Add(new SetCreditTransfersStatusCommand(statusesInfo, messageId,
                    transfersPacket.MessageType,
                    CreditTransferStatus.REJECTED, true));

                break;
```

```
        }

        await ExecuteCommandsInTransaction(commands.ToArray());
    }


    /// <summary>
    /// Processes the camt054 message.
    /// Sets the Payed status to transactions that are listed in message, sets the payed
amount and count.
    /// Sends the transactions packet to ABS only if packet is fully payed.
    /// </summary>
    /// <param name="receiptDocument">Soap model of Transfers Packet Receipt</param>
    /// <param name="documentXml">Original xml message that was received in SOAP
message</param>
    public async Task SubmitPacket(CreditTransfersPacketReceipt receiptDocument, string
documentXml)
    {
        var originalMessageId =
receiptDocument.BkToCstmrDbtCdtNtfctn.Ntfctn.Ntry.NtryDtls.Btch.MsgId;
        var queryable = await mediator.Send(new GetCreditTransfersPacketsQuery());
        var transfersPacket = await queryable.FirstOrDefaultAsync(p => p.MessageId ==
originalMessageId);

        if (transfersPacket.Status == PacketStatus.PROCEEDED)
        {
            return;
        }

        var transferReceipts =
receiptDocument.BkToCstmrDbtCdtNtfctn.Ntfctn.Ntry.NtryDtls.TxDtls;

        var submitReceiptCommand = new
SubmitCreditTransferPacketReceiptCommand(originalMessageId);
        var commands = new List<IRequest> { submitReceiptCommand };

        if (transfersPacket.TransactionsCount == transferReceipts.Count)
        {
            submitReceiptCommand.ReceiptStatus = ReceiptStatus.FULL_PAY;
            commands.Add(new SetAllCreditTransfersStatusCommand(originalMessageId,
transfersPacket.MessageType,
                CreditTransferStatus.PAYED));
        }
        else
        {
            var statusesInfo = transferReceipts.Select(r => new
TransferStatusInfo(r.Refs.UETR)).ToList();
            commands.Add(new SetCreditTransfersStatusCommand(statusesInfo, originalMessageId,
                transfersPacket.MessageType,
                CreditTransferStatus.PAYED));
        }

        if (transfersPacket.PacketDirection == PacketDirection.Income)
```

```
        {
            submitReceiptCommand.Status = PacketStatus.RECFSEP;
        }

        submitReceiptCommand.ReceiptMessageId =
receiptDocument.BkToCstmrDbtCdtNtfctn.GrpHdr.MsgId;
        submitReceiptCommand.IsCamt054Receipt = true;
        submitReceiptCommand.TotalPayedTransfersAmount = transferReceipts.Sum(r =>
r.Amt.Value);
        submitReceiptCommand.PayedTransfersCount = transferReceipts.Count;
        submitReceiptCommand.BOOKGDT =
receiptDocument.BkToCstmrDbtCdtNtfctn.Ntfctn.Ntry.BookgDt.DtTm;
        submitReceiptCommand.NotificationId = receiptDocument.BkToCstmrDbtCdtNtfctn.Ntfctn.Id;

        await ExecuteCommandsInTransaction(commands.ToArray());
    }

    /// <summary>
    /// Inserts the packet with documents into QUEUE table, calls the procedure to process
packet.
    /// </summary>
    /// <param name="packet">Mapped credit transfers packet</param>
    /// <param name="messageType">Message Type</param>
    public async Task ProcessIncomeCreditTransfersPacketAsync(CreditTransfersPacket packet,
string messageType)
    {
        packet.MessageType = messageType;
        packet.CreatedAt = packet.ProcessedAt;
        var transfersList = GetTransfersListFromPacket(packet);
        foreach (var transfer in transfersList)
        {
            transfer.TransferStatusCode = CreditTransferStatus.INSERTED;
            transfer.PacketDirection = PacketDirection.Income;
        }

        if (packet.FiToFiCustomerCreditTransfers != null)
        {
            foreach (var detail in packet.FiToFiCustomerCreditTransfers)
            {
                if (detail.StructuredRemittanceInformation != null)
                {
                    var dateFromEndToEndId = detail.EndToEndId.GetDateFromEndToEndId();
                    detail.StructuredRemittanceInformation.DATE_PAY =
                        dateFromEndToEndId?.Date ?? packet.ProcessedAt?.Date;
                }
            }
        }

        if (packet.InterbankDirectDebitDetails != null)
        {
            packet.InterbankDirectDebitDetails.MessageId = packet.MessageId;
            packet.InterbankDirectDebitDetails.TransferStatusCode =
CreditTransferStatus.INSERTED;
```

```
            packet.InterbankDirectDebitDetails.PacketDirection = PacketDirection.Income;
        }

        var submitPacketCommand =
            new SubmitIncomePacketCommand<CreditTransfersPacket>(packet);
        await ExecuteCommandsInTransaction(submitPacketCommand);
    }

    private IEnumerable<CreditTransferBase> GetTransfersListFromPacket(CreditTransfersPacket
packet)
    {
        var transfersList = new List<CreditTransferBase>()
            .Concat(packet.CreditTransfers?.Select(p => (CreditTransferBase)p) ?? new
List<CreditTransferBase>())
            .Concat(packet.FiToFiCustomerCreditTransfers?.Select(p => (CreditTransferBase)p)
??
                    new List<CreditTransferBase>())
            .Concat(packet.PaymentReturns?.Select(p => (CreditTransferBase)p) ?? new
List<CreditTransferBase>());
        return transfersList;
    }

    public async Task SubmitPacket(InterbankCreditTransfersPacketSoapModel soapModel, string
documentXml)
    {
        var packet = mapper.Map<CreditTransfersPacket>(soapModel);
        await ProcessIncomeCreditTransfersPacketAsync(packet,
XmlNamespace.Pacs009.GetSepMessageType());
    }

    public async Task SubmitPacket(FiToFiCustomerCreditTransferSoapModel soapModel, string
documentXml)
    {
        var packet = mapper.Map<CreditTransfersPacket>(soapModel);
        await ProcessIncomeCreditTransfersPacketAsync(packet,
XmlNamespace.Pacs008.GetSepMessageType());
    }

    public async Task SubmitPacket(PaymentReturnSoapModel soapModel, string documentXml)
    {
        var packet = mapper.Map<CreditTransfersPacket>(soapModel);
        await ProcessIncomeCreditTransfersPacketAsync(packet,
XmlNamespace.Pacs004.GetSepMessageType());
    }

    public async Task SubmitPacket(InterbankDirectDebitSoapModel soapModel, string
documentXml)
    {
        var packet = mapper.Map<CreditTransfersPacket>(soapModel);
        await ProcessIncomeCreditTransfersPacketAsync(packet,
XmlNamespace.Pacs010.GetSepMessageType());
    }
```

```
    public async Task RequireTransferPacketReceiptAsync(string messageId) =>
        await mediator.Send(new RequireTransferPacketReceiptCommand(messageId));

    public async Task SendMessageToSepAsync(string messageId, string messageType)
    {
        var transfersPacket = await mediator.Send(new
GetCreditTransfersPacketByIdQuery(messageId));
        transfersPacket.INTRBKSTTLMDT = DateTime.Now;
        await logger.LogInformation($"Починається відправка пакету {messageType} {messageId}",
messageId);

        switch (messageType)
        {
            case "pacs.008.001.08":
                await
SendPacketToSepAsync<FiToFiCustomerCreditTransferSoapModel>(transfersPacket);
                break;
            case "pacs.009.001.09":
                await
SendPacketToSepAsync<InterbankCreditTransfersPacketSoapModel>(transfersPacket);
                break;
            case "pacs.004.001.09":
                await SendPacketToSepAsync<PaymentReturnSoapModel>(transfersPacket);
                break;
            case "pacs.010.001.04":
                await SendPacketToSepAsync<InterbankDirectDebitSoapModel>(transfersPacket);
                break;
        }

        await mediator.Send(new SetTransferPacketTodaySettlementDateCommand(messageId));
    }

    public async Task SendPacketToAbsAsync(string messageId, string messageType,
PacketDirection direction,
        ReceiptStatus? status = default, CreditTransferStatus? transfersStatus = default)
    {
        await using var transaction = await context.Database.BeginTransactionAsync();
        try
        {
            await logger.LogInformation(Sep4Resources.SendPacketToAbsStarted(messageId),
messageId);

            if (status.HasValue)
            {
                await mediator.Send(new SetCreditTransferPacketReceiptStatusCommand(messageId,
status));
            }

            if (transfersStatus.HasValue)
            {
                await mediator.Send(
                    new SetAllCreditTransfersStatusCommand(messageId, messageType,
transfersStatus.Value));
```

```
            }


            await mediator.Send(SendProcessedCreditTransfersPacketToAbsAsync(messageId,
direction));
            await logger.LogInformation(Sep4Resources.SendPacketToAbsCompleted(messageId),
messageId);
            await transaction.CommitAsync();
        }
        catch (ProcedureCallException ex)
        {
            await transaction.RollbackAsync();
            await logger.LogError(ex,
                Sep4Resources.PacketProcessingFailed(messageId, messageType), messageId);
        }
        catch (WaitForDatabaseUnlockException ex)
        {
            await transaction.RollbackAsync();
            await logger.LogError(ex,
                Sep4Resources.PacketProcessingFailedWithDatabaseLocked(messageId,
messageType), messageId);
            await mediator.Send(
                new SetCreditTransferPacketStatusCommand(messageId, PacketStatus.WAITUNLOCK));
        }
        catch (Exception ex)
        {
            await transaction.RollbackAsync();
            await logger.LogError(ex,
                Sep4Resources.PacketProcessingFailed(messageId, messageType), messageId);
        }
    }
}
```

ДОДАТОК Г

Приклад реалізації класу для обробки відповідних повідомлень

```
public class SepMessageProcessingService : ISepMessageProcessingService
{
    private readonly IServiceProvider serviceProvider;
    private readonly IServiceLogger<SepMessageProcessingService> logger;
    private readonly ISoapPacketsQueueService service;
    private readonly IMediator mediator;
    private readonly IUnableToProcessIncomePacketService failedReportService;
    private readonly ISignService signService;

    public SepMessageProcessingService(IServiceProvider serviceProvider,
        IServiceLogger<SepMessageProcessingService> logger, ISoapPacketsQueueService service,
IMediator mediator,
        IUnableToProcessIncomePacketService failedReportService, ISignService signService)
    {
        this.serviceProvider = serviceProvider;
        this.logger = logger;
        this.service = service;
        this.mediator = mediator;
        this.failedReportService = failedReportService;
        this.signService = signService;
    }

    public async Task ProcessMessagesBatch(List<string> messageIds)
    {
        foreach (var messageId in messageIds)
        {
            try
            {
                await ProcessMessage(messageId);
            }
            catch (Exception e)
            {
                await logger.LogError(e, "Error while processing message", messageId);
            }
        }
    }

    [Queue("income_packets")]
    public async Task<PacketProcessingResult> ProcessMessage(string messageId)
    {
        try
        {
            await mediator.Send(new LoginDbUserCommand());
            await mediator.Send(new EnsureAbsIsAvailable());
            var packet = await service.GetSepPacketByIdAsync(messageId);
```

```
            await mediator.Send(new EnsureOriginalPacketProcessed(packet.OriginalMessageId));
            await mediator.Send(new
SetPacketProcessedAtTimeCommand<FailedSoapPacket>(packet));
            var xml = await mediator.Send(new GetFailedPacketXmlQuery(packet.MessageId));
            signService.VerifySignature(xml, packet.CreatedAt.Value);
            xml = xml.GetDocumentXmlFromEnvelope();

            await mediator.Send(new EnsurePacketBankingDate(xml));

            await ProcessPacket(xml, packet.MessageId);

            await mediator.Send(new UpdateSepPacketStatusCommand(packet.MessageId,
SepPacketStatusCode.Processed,
                string.Empty, string.Empty));

            await logger.LogInformation(
                $"Пакет успішно оброблено {packet.MessageType} {packet.MessageId}",
packet.MessageId);
        }
        catch (InvalidSoapSignatureException ex)
        {
            if (ex.ErrorCode == 27)
            {
                await logger.LogInformation("Cannot verify root certificate of NBU",
messageId);
                await mediator.Send(new UpdateSepPacketStatusCommand(messageId,
                    SepPacketStatusCode.InitialPacket, string.Empty, string.Empty));
                return new PacketProcessingResult(messageId, true);
            }
            await logger.LogError(ex, ex.Message, messageId);
            var errorMessage =
ex.ToString().TruncateToMaxLength(StringMaxLength.FailedPacketErrorMessageMaxLength);
            await mediator.Send(new UpdateSepPacketStatusCommand(messageId,
                SepPacketStatusCode.ProcessingFailed, errorMessage));
            return new PacketProcessingResult(messageId, false, ex.ToString());
        }
        catch (AbsUnavailableException ex)
        {
            await logger.LogInformation("ABS is unavailable, will retry later", messageId);
            await mediator.Send(new UpdateSepPacketStatusCommand(messageId,
                SepPacketStatusCode.InitialPacket, string.Empty, string.Empty));
        }
        catch (OriginalPacketNotProcessedException ex)
        {
            await logger.LogInformation("Original packet is not processed yet, will retry
later", messageId);
            await mediator.Send(new UpdateSepPacketStatusCommand(messageId,
                SepPacketStatusCode.InitialPacket, string.Empty, string.Empty));
        }
        catch (CannotProcessFuturePacketDateException ex)
        {
            await logger.LogInformation("Packet date exceeds current Banking date, will retry
later " + messageId, messageId);
```

```
            await mediator.Send(new UpdateSepPacketStatusCommand(messageId,
                SepPacketStatusCode.InitialPacket, string.Empty, string.Empty));
        }
        catch (Exception ex)
        {
            if (ex.InnerException is OracleException &&
ex.InnerException.Message.Contains("ORA-00001"))
            {
                await logger.LogInformation("Пакет вже був оброблений " + messageId,
messageId);
                await mediator.Send(new UpdateSepPacketStatusCommand(messageId,
                    SepPacketStatusCode.Processed, string.Empty, string.Empty));
                return new PacketProcessingResult(messageId, true);
            }

            if (ex is ProcedureCallException && ((ProcedureCallException)ex).ErrorCode == "-
20987")
            {
                await logger.LogError(ex,
                    $"Procedure call failed, will retry later {messageId}", messageId);
                await mediator.Send(new UpdateSepPacketStatusCommand(messageId,
                    SepPacketStatusCode.InitialPacket, string.Empty, string.Empty));
                return new PacketProcessingResult(messageId, false, ex.ToString());
            }

            await logger.LogError(ex, ex.Message, messageId);
            var errorMessage =
ex.ToString().TruncateToMaxLength(StringMaxLength.FailedPacketErrorMessageMaxLength);
            await mediator.Send(new UpdateSepPacketStatusCommand(messageId,
                SepPacketStatusCode.ProcessingFailed, errorMessage));
            return new PacketProcessingResult(messageId, false, ex.ToString());
        }
        finally
        {
            await mediator.Send(new LogoutDbUserCommand());
        }

        return new PacketProcessingResult(messageId, true);
    }

    private async Task ProcessPacket<T>(string xml, string messageId, string messageType)
    {
        try
        {
            var service = serviceProvider.GetRequiredService<ISepMessageConsumerService<T>>();
            var packet = XmlConverter.Deserialize<T>(xml);
            await service.SubmitPacket(packet, xml);
        }
        catch (Exception ex)
        {
            await logger.LogError(ex, Sep4Resources.PacketProcessingFailed(messageId,
messageType), messageId);
            throw;
```

```
        }
    }

    private async Task ProcessPacket(string documentXml, string messageId)
    {
        var messageType = documentXml.GetMessageTypeFromDocumentXml().Split(':').Last();

        switch (messageType)
        {
            case "camt.054.001.08":
                await ProcessPacket<CreditTransfersPacketReceipt>(documentXml, messageId,
messageType);
                break;
            case "pacs.002.001.10":
                await ProcessPacket<CreditTransfersPacketFailedStatusReport>(documentXml,
messageId,
                    messageType);
                break;
            case "pacs.009.001.08":

                await ProcessPacket<InterbankCreditTransfersPacketSoapModel>(documentXml,
messageId,
                    messageType);
                break;
            case "camt.010.001.08":
                await ProcessPacket<AccountLimitsResponseSoapModel>(documentXml, messageId,
messageType);
                break;
            case "camt.004.001.08":
                await ProcessPacket<AccountBalanceResponseSoapModel>(documentXml, messageId,
messageType);
                break;
            case "camt.052.001.08":
                await ProcessPacket<BankToCustomerAccountReportSoapModel>(documentXml,
messageId, messageType);
                break;
            case "camt.053.001.08":
                await ProcessPacket<AccountStatementResponseSoapModel>(documentXml, messageId,
messageType);
                break;
            case "pacs.008.001.08":

                await ProcessPacket<FiToFiCustomerCreditTransferSoapModel>(documentXml,
messageId, messageType);
                break;
            case "camt.091.001.01":
                await ProcessPacket<SecuritiesPacketSoapModel>(documentXml, messageId,
messageType);
                break;
            case "camt.025.001.05":
                await ProcessPacket<SoapMessageErrorPacketResponse>(documentXml, messageId,
messageType);
                break;
```

```
            case "camt.026.001.08":
                await ProcessPacket<UnableToApplyPacketSoapModel>(documentXml, messageId,
messageType);
                break;
            case "pacs.004.001.08":
            case "pacs.004.001.09":

                await ProcessPacket<PaymentReturnSoapModel>(documentXml, messageId,
messageType);
                break;
            case "camt.056.001.08":
                await ProcessPacket<FiToFiPaymentCancellationRequestSoapModel>(documentXml,
messageId,
                    messageType);
                break;
            case "camt.029.001.09":
                await ProcessPacket<ResolutionOfInvestigationSoapModel>(documentXml,
messageId, messageType);
                break;
            case "pain.013.001.07":
                await ProcessPacket<CreditorPaymentActivationRequestSoapModel>(documentXml,
messageId,
                    messageType);
                break;
            case "pacs.010.001.03":

                await ProcessPacket<InterbankDirectDebitSoapModel>(documentXml, messageId,
                    messageType);
                break;
            case "admi.007.001.01":
                await ProcessPacket<ReceiptAcknowledgementSoapModel>(documentXml, messageId,
                    messageType);
                break;
            case "pain.014.001.07":
                await
ProcessPacket<CreditorPaymentActivationRequestStatusReportSoapModel>(documentXml, messageId,
                    messageType);
                break;
            case "pain.087.001.07":
                await ProcessPacket<RequestToModifyPaymentSoapModel>(documentXml, messageId,
                    messageType);
                break;
            case "admi.010.001.02":
                await ProcessPacket<StaticDataReportSoapModel>(documentXml, messageId,
messageType);
                break;
            case "admi.004.001.02":
                await mediator.Send(new SubmitSepNotificationEventMessageCommand(messageId,
documentXml));
                break;
            case "camt.087.001.07":
                await ProcessPacket<RequestToModifyPaymentSoapModel>(documentXml, messageId,
messageType);
```

```
                break;
            case "camt.027.001.08":
                await ProcessPacket<ClaimNonReceiptSoapModel>(documentXml, messageId,
messageType);
                break;
            case "admi.998.001.02":
                var packet =
XmlConverter.Deserialize<SepParticipantsListSoapModel>(documentXml);
                if (packet.AdmstnPrtryMsg.PrtryData.Tp == "SUch" ||
packet.AdmstnPrtryMsg.PrtryData.Tp == "SUchTom")
                {
                    await ProcessPacket<SepParticipantsListSoapModel>(documentXml, messageId,
messageType);
                }
                else
                {
                    await ProcessPacket<SepAspSpOrganizationsResponseSoapModel>(documentXml,
messageId,
                        messageType);
                }

                break;
            default:
                throw new Exception(Sep4Resources.UnsupportedPacketFormat);
        }
    }
}
```