ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем
(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»
(освітньо-кваліфікаційний рівень)

Розробка програмного середовища інтеграції сервісів у розподілених інформаційних системах
Development of a software environment for the integration of services in distributed information systems

Виконав: студент денної форми навчання
спеціальності 126 – Інформаційні системи та технології .
(шифр і назва напряму підготовки, спеціальності)
Освітня програма «Інформаційні системи та технології»
(назва освітньої програми)
Гальчинський Максим Вячеславович
(прізвище, ім'я, по-батькові)

Керівник Ст. викл. Максимов О. С.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Ст. викл. Трубіна Н. Ф.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№___ від «___»_____ 2023 р.
Завідувач кафедри
_____ Євгеній МАЛАХОВ
(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № ___
протокол № __ від «___» ____ 2023 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)
Голова ЕК
_____ _Володимир ВИЧУЖАНІН_
(підпис) (ім'я, прізвище)

Одеса – 2023

АНОТАЦІЯ

У дипломній роботі розглядається тема «Розробка програмного середовища інтеграції сервісів у розподілених інформаційних системах».

Метою роботи є забезпечення ефективної та безперервної взаємодії між різними компонентами системи, яка сприяє зручній та надійній інтеграції різноманітних сервісів. Розробка такого програмного середовища сприяє поліпшенню інтеграції сервісів, спрощенню розробки та підтримки цієї системи у майбутньому.

Для реалізації проєкту взято за основу вже існуючу інформаційну систему з моніторингу податків, розроблену за період навчання. Для побудови розподіленості використовується технологія Windows Communication Foundation з використанням архітектурного стилю «Брокер» та сервісно-орієнтованою архітектурою. Мову програмування обрано C#. Для взаємодії з даними використовується система керування базами даних PostgreSQL. Система для побудови клієнтський застосунків – Windows Presentation Foundation.

В результаті впровадження розподіленості в систему моніторингу податків та зборів можна легко додавати нові відділи з вузлами обробки даних, а для під’єднання цих вузлів до самого відділу необхідна мінімальна участь спеціалістів, з подальшою можливістю зміни ір-адреси хосту без їх участі. Завдяки ж реалізації сервісної структури, до застосунку можна додавати, змінювати чи зовсім видаляти будь які сервіси з урахуванням модифікації інтерфейсу.

ANNOTATION

The thesis deals with the topic "Development of a software environment for the integration of services in distributed information systems".

The purpose of the work is to ensure effective and continuous interaction between various components of the system, which contributes to convenient and reliable integration of various services. The development of such a software environment contributes to improving the integration of services, simplifying the development and support of this system in the future.

For the implementation of the project, the already existing tax monitoring information system, developed during the training period, was taken as a basis. Windows Communication Foundation technology is used to build distribution using the "Broker" architectural style and service-oriented architecture. The programming language is C#. The PostgreSQL database management system is used to interact with data. The system for building client applications is Windows Presentation Foundation.

As a result of the implementation of distribution in the tax and fee monitoring system, new departments with data processing nodes can be easily added, and to connect these nodes to the department itself, minimal participation of specialists is required, with the subsequent possibility of changing the IP address of the host without their participation. Thanks to the implementation of the service structure, any services can be added, changed or completely removed to the application, taking into account the modification of the interface.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Скорочення

AWS – Amazon Web Services

GCP – Google Cloud Platform

REST – Representational State Transfer

SOAP – Simple Object Access Protocol

WCF – Windows Communication Foundation

WPF – Windows Presentation Foundation

РІС – Розподілена Інформаційна Система

СКБД – Система Керування Базами Даних

ВСТУП

Україна є однією із лідерів країн світу, в якій розвиваються цифрові технології у всіх галузях. В останні роки стає особливо помітним цей прогрес, наприклад, в умовах територіальної розподіленості або великого навантаження, а отже звичайні інформаційні системи вже не задовольняють усіх потреб для вирішення задач.

Основною метою даного дослідження є розгляд та використання основних підходів та методів для впровадження сервісів у розподілених інформаційних системах (РІС) з подальшою можливістю масштабування системи і забезпечення стабільного зв’язку між усіма вузлами системи.

Об’єктом дослідження є процес інтеграції сервісів у розподілених інформаційних системах.

Предметом дослідження є методи та алгоритми налаштування РІС з інтеграцією сервісів.

Для досягнення цієї мети поставлені та вирішені наступні задачі:

– виконано аналіз предметної області;

– визначено групи користувачів системи та сформулювати ряд задач, які вони можуть виконувати;

– обрано архітектуру і шаблон проектування;

– обрано технології та засоби реалізації;

– створено РІС з використанням обраних технологій;

– протестувати та налагодити роботу застосунку.

1 ІНТЕГРАЦІЯ СЕРВІСІВ У РОЗПОДІЛЕНИХ ІНФОРМАЦІЙНИХ СИСТЕМАХ

В цьому розділі розглядаються основні підтеми роботи: РІС та сервіси. В кожній з них детально описано основні проблеми при побудові та способи їх вирішення.

1.1 Технологія розподілених інформаційних систем

Розподілену систему визначають як систему, в якій апаратні або програмні компоненти, розташовані на об’єднаних в мережу комп'ютерах, які спілкуються та координують свої дії завдяки передачі повідомлень[1].

На відміну від централізованих систем, де всі рішення приймаються в одному центрі, розподілена система має кілька вузлів, які можуть приймати рішення та працювати незалежно один від одного. Кожен вузол може мати свою локальну базу даних та обробляти запити від інших вузлів, що дозволяє розподіленій системі ефективніше використовувати ресурси та покращувати відмовостійкість. А за рахунок того, що вузли розподілені по різним точках світу зменшується затримка передачі даних.

Розподілені інформаційні системи забезпечують гнучкий і надійний спосіб керування та обробки великих обсягів даних вони легко піддаються масштабованості, відмовостійкості, продуктивності, рентабельності та географічного розподілу, проте разом з цим всі переваги відображаються на складності системи, її безпеці та узгодженості даних.

Взаємодія між процесами в РІС забезпечується за рахунок обміну повідомленнями. Щоб вони могли взаємодіяти без збоїв необхідне їх блокування. Воно буває синхронним і асинхронним.

Синхронна модель блокує відправника поки він не отримає відповідь на свій запит або не спрацює тайм-аут. Через те що є необхідність очікувати відповідь цей метод не є ефективним, проте легший в реалізації.

В асинхронній моделі відправник може продовжувати роботу після відправки повідомлення і очікувати на відповідь в будь-який час. За рахунок цього, цей метод більш ефективний та забезпечує масштабованість системи, але може мати певні труднощі з обробкою запитів.

При робота з повідомленнями є певні особливості: вони можуть загубитись в мережі, бути доставленими в різному порядку, дублюватися або змінювати зміст.

Для гарантії доставки повідомлення існують методи:

– Arbitrary Order: повідомлення доставляються в різному порядку, тому кожен об’єкт може отримувати їх у різній послідовності;

– FIFO Order: повідомлення від одного й того ж процесу доставляються в одному порядку для всіх інших процесів;

– Causal Order: повідомлення від одного й того ж процесу доставляється в одному порядку для всіх об’єктів стосовно цього процесу;

– Total Order: повідомлення доставляється в одному порядку для всіх об’єктів.

Для поєднання процесів використовують посередника. Завдяки цьому підвищується безпека обміну даних, система краще масштабується і процесам не потрібно знати про існування один одного. Але разом з тим, посередник має обробляти запити при великому навантаженні, через що може знижуватись продуктивність, а збій посередника призведе до повного збою обміну повідомлень.

Наступною задачею є масштабування системи. Воно може відбуватися декількома способами: або додавати нові компоненти до системи (вертикальне масштабування), або оновлювати та покращувати вже існуючі (горизонтальне масштабування). Проте завжди слід враховувати вузькі місця,

адже при існуванні таких місць будь-яке покращення системи може не призвести до необхідних очікувань.

Одним із способів масштабування також є гнучке використання існуючих ресурсів, оптимізуючи їх та використовуючи в залежності від навантаження на систему.

Основними аспектами масштабування є кількість оброблюємих запитів, об’єм даних та об’єм розрахунків на запит. Перший аспект впливає на обробку великої кількості запитів тож необхідно вирішити проблему як ефективно розподіляти запити між усіма вузлами і обробляти їх паралельно. Для впровадження наступного аспекту треба вирішити як зберігати великі масиви даних у різних місцях і паралельно з цим ефективно працювати з інформацією. І останній аспект вирішує чи здатна система обробити паралельно багато складних запитів.

Для вирішення цих проблем використовують методи реплікації, кешування або шардингу.

Реплікація – процес зберігання декількох копій однієї інформації з метою покращити відмовостійкість та оптимізувати роботу обробки інформації різним вузлам. Для її реалізації можно використовувати балансувальник навантаження (Load Balancer). Він буде розподіляти запити по різним вузлам, що забезпечить високу доступність та швидкість системи.

Так як балансувальник навантаження вирішує куди направити запит з’являється ще одна проблема, а саме як відслідковувати клієнтські сесії. Для цього слід передавати унікальний ідентифікатор від клієнта до серверу, а балансувальник буде запам’ятовувати інформацію про теперішні сеанси.

Є й інші недоліки. Через те що дані дублюється випливає необхідність в більшому об’ємі пам’яті, що в свою чергу збільшує вартість обслуговування. Ще слід зазначити, що вузли повинні між собою

обмінюватись даними, а для цього треба узгодити актуальну інформацію та справлятися з навантаженням на мережу.

Для вирішення проблеми узгодженості використовують синхронізацію вузлів, яка може бути синхронною або асинхронною.

Синхронна синхронізація забезпечує одночасне копіювання даних на всі вузли, при чому в момент розповсюдження від первинного серверу запис на нього блокується, поки не буде отримано підтвердження про запис на інші вузли. Цей метод гарантує актуальність даних на всіх вузлах, але це тягне за собою затримку в очікуванні підтвердження, особливо при великій кількості вузлів. А найголовнішою проблемою може стати неправильна робота одного з вузлів, що призводить до блокування всієї системи.

Асинхронна синхронізація спочатку записує дані на головний вузол, а потім виконує розповсюдження даних з цього вузла на інші і одночасно дає можливість маніпулювати даними на ньому. Такий підхід підвищує продуктивність і зменшує навантаження на головний вузол, але при цьому можлива втрата даних при відмові головного серверу або їх дублювання після відновлення роботи.

Для реалізації цих методів існують такі можливі архітектури:

– Реплікації без лідера;
– Реплікація з одним лідером;
– Реплікація з декількома лідерами.

Реплікація без лідера не залежить від певного вузла, який би міг координувати дії інших. Замість цього всі вузли можуть одночасно приймати операцію запису на своїй локальній системі, а потім синхронізуються з іншими вузлами. Така система гарно масштабується і рівномірно навантажує всі вузли, але тут дуже складно реалізовувати обробку конфліктів при одночасному записі.

Для покращення моніторнгу даних використовують такі механізми:

– Read repair – автоматично виправляє помилки при читанні даних і робить їх узгодженими автоматично;

– Anti-entropy – вузол може виявити, що його дані застарілі і запросити актуальну копію даних від інших різних вузлів для порівняння з власною. При виявленні невідповідностей, система оновлює свою копію.

Реплікація з одним лідером являє собою структуру, в якій є один основний вузол, який приймає дані на запис і декілька резервних, які дублюють дані з головного вузла.

Перевагами даної моделі є легке налаштування системи та подальше обслуговування, можно виконувати обслуговування чи оновлення будь якого вузла в системі без припинення роботи самої системи. Але разом з тим є незначні недоліки при відмові головного вузла до завершення передачі даних, таким чином вони можуть не зберегтися. Також проблемою може стати маленька пропускна спроможність головного вузла, що робить вузькі місця в системі. І можлива затримка при читанні, якщо дані були щойно оновлені.

Тепер виникає інша проблема: «Яким чином передавати актуальну інформацію при додаванні нового вузла?». Для цього треба створити на головному вузлі резервну копію, але не завжди є можливість зупинити систему, тому використовується підхід snapshot-based. Він робить точну копію даних на лідері на певний момент часу і потім передає знімок на інші вузли. Після чого передаються лише ті дані, які були створені після знімку. Цей спосіб простий у реалізації, но може бути не зовсім ефективним при великій кількості даних, тоді слід використовувати інші підходи.

При виході з ладу одного з другорядних вузлів головний вузол перестає надсилати йому дані, а збирає їх в лог. При відновлені роботи необхідно перевірити цілісність даних (checksum verification) і надалі відновити дані за допомогою раніше розглянутих методів.

При відмові лідера (failover), що може статися через вихід з ладу або для оновлення компонентів, слід виконати заміну на нового лідера. Зазвичай

це відбувається автоматично, при виявлені збою обирається новий лідер із доступних підлеглих. Після цього всі запити повинні бути перенаправлені на обраного лідера.

Реплікація з декількома лідерами має архітектуру, в якій декілька вузлів мають право на запис. Завдяки цьому зникає проблема систем з одним лідером при відмові одного з лідерів, але разом з цим виникає складність вирішення конфліктів одночасного запису та узгодженості даних.

Основним недоліком в реплікації з декількома лідерами є конфлікти в узгодженості даних. Для їх вирішення використовуються методи:

– Обрання операції з найбільшим ID (last write wins) – найпростіший підхід, при якому конфлікт вирішується шляхом вибору останнього вузла, що записав дані. Однак цей метод може призвести до втрати даних і не є прийнятним для всіх типів програм;

– Злиття конфліктуючих значень – цей підхід полягає у злитті значень, які не збігаються, з метою отримання єдиного значення, яке може бути записано у кожний вузол. Однак цей метод може бути досить складним у реалізації і не завжди можливим для всіх типів даних;

– Долучення користувача для вирішення конфлікту – при цьому підході програма може включати сповіщення користувача про конфлікт та запит щодо подальших дій. Однак цей метод може бути досить складний у реалізації та вимагає узгодження з кінцевими користувачами;

– Conflict-free replicated data types – найскладніший підхід, для роботи якого необхідні спеціальні типи даних, які можуть змінюватись паралельно на різних вузлах. Прикладами таких структур даних є G-Counter, LWW-Element Set, OR-Set, RGA, Treedoc.[2]

Кешування – процес збереження даних у пам’яті, яка знаходиться найближче до клієнта. Цей метод зменшує кількість запитів на сервер при повторних діях клієнту, і пришвидшує відповідь при розпоширених запитах. Цей метод має лише один недолік, він може повернути застарілу інформацію, але і це легко вирішується встановленням ліміту життя кешу.[3]

Шардинг – поділ (партиціонування) бази даних на окремі частини (шарди) так, щоб кожну з них можна було винести на окремі вузли. Кожен шард містить лише частину даних, що дозволяє більш ефективно використовувати ресурси та покращити продуктивність системи при великому обсязі даних. Також є такий процес як шардинг кешу. Він дозволяє розподіляти навантаження на кеш між кількома серверами та збільшувати масштабованість програми.

Одним із головних аспектів РІС є паралельна обробка. Для її реалізації існують такі підходи:

– Модель «мультитредингу»: процес ділиться на кілька потоків виконання, які працюють паралельно. Кожен потік обробляє свою частину завдання, що дозволяє досягти більшої швидкості виконання. Однак це може призвести до проблем із синхронізацією доступу до загальних ресурсів;

– Модель «мультипроцессинга»: у цій моделі завдання розбиваються на кілька процесів, які працюють паралельно на різних ядрах процесора. Це дозволяє розподілити навантаження на процесор та прискорити виконання завдання;

– Модель «кластерної обробки»: у цій моделі завдання виконується на кількох обчислювальних вузлах, які працюють разом у кластері. Кожен вузол обробляє свою частину завдання, а результати збираються разом. Це дозволяє збільшити масштабованість системи;

– Модель «розподіленої обробки»: у цій моделі завдання виконується на кількох вузлах у мережі, які можуть бути розташовані на різних фізичних серверах. Кожен вузол обробляє свою частину завдання та передає результати іншому вузлу для подальшої обробки. Це дозволяє збільшити продуктивність системи та покращити її відмовостійкість.

При проектуванні паралельної системи необхідно враховувати, що не всі операції можна розпаралелити, оскільки деякі частини системи можуть мати послідовний код, який не може виконуватись паралельно.

Для забезпечення узгодженості дій між вузлами необхідна координація. Для її впровадження існує кілька різних методів, наприклад:

– Централізована координація – цей метод включає використання центрального вузла, який координує всі дії в системі. Вона може бути ефективною для невеликих систем, але водночас стати вузьким місцем продуктивності та обмежувати масштабованість системи;

– Розподілена координація – в цьому методі використовується узгодження протоколів для координації дій між вузлами системи. Ці протоколи можуть використовувати методи, такі як більшість голосування, щоб прийняти рішення про дії в системі. Розподілена координація може бути ефективнішою, ніж централізована, оскільки може збільшувати масштабованість системи;

– Безкоординаційна – цей метод включає використання протоколів, які дозволяють вузлам системи узгоджувати дії без необхідності централізованого або розподіленого вузла. Цей метод може бути ефективнішим для великих систем, оскільки він забезпечує високу масштабованість та відмовостійкість.

Одним із можливих компонентів, які використовують ці методи є годинник. Він необхідний для відслідковування порядку зміни даних, їх актуальності та цілісності. Його використовують для синхронізації часу на різних вузлах. Існують такі види годинників:

– Фізичний годинник (wall-clock time) – ґрунтується на реальному часі на комп'ютері, де виконується програма. Однак, такий годинник може бути неточним і не синхронізованим між різними вузлами системи;

– Логічний годинник (logical clocks) – упорядковує операції та події в системі, використовуючи порядок подій, а не реальний час. Такий годинник може бути реалізований як лічильник, який інкрементується при кожній операції чи події;

– Векторний годинник (vector clocks) – являє собою набір логічного годинника, кожен з яких відноситься до конкретного вузла системи. Векторний годинник дозволяє враховувати порядок операцій у різних вузлах і визначати, які операції були виконані перед якими.

Для синхронізації часу між сервером та клієнтом можна використовувати такі підходи:

– Network Time Protocol (NTP) – це протокол, який використовується для синхронізації годинника між комп'ютерами в мережі. Він дозволяє визначати, який час знаходиться на сервері, і коригувати годинник клієнта відповідно до цього часу;

– Оцінка точного часу – у цьому методі використовується визначення часу на основі кількох джерел, таких як GPS або радіосигнали. Він може бути більш точним, ніж NTP, але потребує більших затрат;

– Корекція годинника клієнта – у цьому методі вважається різниця між часом на сервері і часом на клієнті, потім, відповідно до цієї різниці коригується годинник клієнта. Цей метод не такий точний, як попередні, але він швидкий і простий у використанні.

Іншим можливим варіантом узгодження даних може бути алгоритм happened-before. Він відслідковує послідовність виконання подій і потім виставляє їх у необхідному порядку. Цей алгоритм може визначати послідовність на основі двох критеріїв:

– Причинно-наслідковий зв'язок: якщо подія A є причиною події B, то A має статися раніше B;

– Відношення «запит-відповідь»: якщо подія A є запитом до системи, а подія B – відповіддю на цей запит, то A має відбутися раніше B.

Наступний чинник це безпека. Будь яка інформаційна система піддається нападам з боку злодіїв, тому слід визначити основні типи загроз та можливості їх запобігання.

Можливими атаками на інформаційну систему можуть бути: отримання конфіденційної інформації (eavesdropping), зміна переданих даних (data tampering), встроювання посередника між відправником та одержувачем для перехвату та зміни даних (man-in-the-middle), видання себе за іншого (spoofing), запам’ятовування переданих даних з метою повторення процедури

з’єднання (replaying), перевантаження системи (denial-of-service), встроювання сторонній код в систему.

Для того, щоб система мала достатній рівень захисту від подібних атак в неї бажано впровадити всі з перелічених вимог:

– Конфіденційність – система повинна надавати конфіденційну інформацію лише тим особам, які мають на неї права;

– Цілісність – система має слідкувати за цілісністю даних;

– Аутентифікація – повинна бути перевірка дійсності користувача;

– Авторизація – кожен користувач повинен мати змогу маніпулювати таким чином і лише з тими даними, які визначені для нього;

– Доступність – всі авторизовані користувачі повинні мати змогу отримати дані за їх правами;

– Масштабованість – система повина мати змогу мастабуватися, але при цьому всі існуючі правила безпеки не повинні бути порушені;

Для забезпечення цих вимог використовують різні механізми безпеки, такі як шифрування, системи антивірусів, моніторинг, аудит та захист мережі.

Окремо слід зазначити методи захисту від DoS-атак, адже такий тип загрози може призвести до повної зупинки програми. Тому можливими варіантами боротьби є:

– Обмеження кількості запитів з однієї IP-адреси;

– Фільтрація трафіку;

– Використання балансувальників навантаження;

– Використання хмарних сервісів;

– Використання спеціалізованих програм та обладнання;

– Налаштування системи моніторингу.

Для захисту кода можна ізолювати його середу виконання, завдяки чому запобігти небажаного впливу на інші системні компоненти. Іншим методом захисту кода є пошук потенційної небезпеки за допомогою

спеціальних програм, таких як: Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), Interactive Application Security Testing (IAST), Software Composition Analysis (SCA) та інші.

Ще однією проблемою в PIC є необхідність мати доступ до кожного компонента і можливість звернутися до них. Для цього достатньо знати точну адресу об’єкту, яка може бути визначена завдяки схемі іменування. Вони можуть бути:

– Плоска схема іменування – це найпростіший спосіб іменування, коли кожен компонент системи має унікальне ім'я. Ця схема зручна для невеликих систем, але при зростанні кількості компонентів зростає і необхідна кількість ідентифікаторів, що ускладнює процес;

– Структурована схема іменування – це складніша схема, яка використовує структуровані імена для компонентів системи. Ця схема може містити ієрархічні імена, які складаються з декількох рівнів, що робить її більш гнучкою та зручною для великих систем;

– Схема іменування з атрибутами – це схема, яка використовує додаткові атрибути для опису компонентів системи, такі як розташування, тип, функціональність тощо. Це дозволяє створювати більш деталізовані імена та забезпечує більш гнучкий та зручний спосіб іменування для великих систем.[4, 5]

1.2 Сервісний підхід в інформаційних системах

Сервіси відноситься до програмних компонентів, які надають певні функції або набір функцій іншим програмним компонентам або системам. Сервіс можна розглядати як окрему програму, яка виконує певні завдання та працює незалежно, або як частину більшої системи чи програми.

Сервіси можуть допомогти спростити складні програмні системи, розбиваючи їх на менші незалежні компоненти, які можна легко підтримувати та масштабувати. Вони також дозволяють різним системам і програмам взаємодіяти одна з одною та обмінюватися даними безпечним і ефективним способом. Їх часто реалізуються з використанням різних архітектурних стилів, таких як веб-сервіси, API, мікросервіси або інші. Проте для взаємодії зазвичай мають чітко визначений інтерфейс, який визначає, як інші програмні компоненти можуть взаємодіяти з ними та використовувати їхні функції.

Недоліком сервісного підходу є складність поєднання всіх компонентів ніж це було при монолітній програмі, так як треба визначати протоколи для взаємодії з кожним із сервісом. Іншою проблемою може бути вихід одного із сервісів з ладу. Така ситуація може призвести до повної зупинки системи або некоректність відповідей на запити. Через додаткову логіку взаємодії сервісів між собою знижується продуктивність системи. А якщо сервіси ще й розроблені на різних архітектурних рішеннях, то виникає складність поєднання все в одну систему.

Архітектура сервісного підходу в першу чергу складається з модульних частин – сервісів. Кожен сервіс повинен мати певний інтерфейс, який визначає можливі методи звертання та типи даних, з якими він працює. Для взаємодії використовують шину сервісів (ESB), яка являється посередником між сервісами та основною програмою. Така шина здатна перетворювати дані і контролювати помилки.

Щоб звернутися до сервісу необхідно слідкувати правилам бізнес-логіки. Ці правила зазвичай задокументовані і дають змогу автоматизовувати дії застосунку. Вони можуть бути реалізовані у вигляді скрипту або таблиць. За рахунок такому підходу можна швидко змінити

правила без змін в коді. Але при великій кількості бізнес-правил система стає неконтрольованою і може призвести до втрати продуктивності.

Сервіси діляться на:

– Бізнес-сервіси (Business Services) – сервіси, які надають бізнес-функціональність. Вони мають високий рівень абстракції тому абстрагуються від технічних деталей реалізації та сконцентровані на вирішенні бізнес-задачах;

– Інтеграційні сервіси (Integration Services) – сервіси, які використовують в якості посередників для обміну даними між системами. Вони можуть використовуватися для інтеграції зовнішніх систем або забезпечення взаємодії між різними програмами, що працюють у межах однієї компанії;

– Керуючі сервіси (Management Services) – надають функції управління та контролю за сервісами, такі як моніторинг, управління доступом, оновлення тощо. Ці сервіси можуть використовуватись для автоматичного масштабування сервісів або виділення ресурсів;

– Інфраструктурні послуги (Infrastructure Services) – надають технічну інфраструктуру, необхідну роботи сервісів. Ці послуги можуть включати послуги авторизації, кешування, маршрутизації і т.д.;

– Сервіси даних (Data Services) – надають доступом до даних, використовуючи стандартизовані інтерфейси, такі як SQL, XML чи Web Services. Вони можуть бути використані для отримання даних із різних джерел, таких як бази даних, файли або веб-служби;

– Веб-служби (Web Services) – є стандартизованим способом обміну даними між різними програмами та системами. Вони використовують протоколи та формати, такі як HTTP, SOAP, XML (eXtensible Markup Language) та WSDL (Web Services Description Language), і можуть бути використані для обміну інформацією між сервісами.

Для забезпечення взаємодії між компонентами у сервісно-орієнтованій архітектурі використовуються стандарти та протоколи. Найбільш поширеними з них є:

– SOAP – протокол обміну повідомленнями, зазвичай використовується взаємодії між сервісами. Він може використовувати різні транспортні протоколи, такі як HTTP, SMTP або JMS для передачі повідомлень між сервісами;

– WSDL – мова опису веб-сервісів, яка використовується для визначення інтерфейсів сервісів, доступних для інших програм та систем;

– XML – мова розмітки, яка широко використовується для передачі даних між різними програмами та системами. XML може використовуватися для опису структури повідомлень SOAP та WSDL;

– REST (Representational State Transfer) – архітектурний стиль, який описує, як побудувати розподілені системи, використовуючи стандартні протоколи HTTP та HTTPS. REST-сервіси використовуються для обміну даними між різними програмами та системами;

– JSON (JavaScript Object Notation) – формат передачі даних, який широко використовується в REST-сервісах для передачі структурованих даних між різними програмами та системами.[6]

1.3 Сервісно-орієнтовані технології в розподілених інформаційних системах

При створенні PIC з використанням сервісів утворюється система, яку можна підлаштовувати під себе. Сервіси грають роль блоків, які при поєднанні дають постійно нові можливості.

Найбільш поширеними прикладами таких систем є: Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure.

AWS – це платформа хмарних обчислень, яку надає компанія Amazon. Вона пропонує гнучку та масштабовану інфраструктуру, яка дозволяє організаціям розробляти та розгортати програми та сервіси у хмарі.

AWS побудувала свою екосистему таким чином, що кожен може долучитися до партнерства і інтегрувати власні рішення з цією платформою. Завдяки цьому система швидко розвивається і має багато різних інструментів та сервісів для автоматизації та управління інфраструктурою у хмарі. Це надає можливості моніторингу, управління безпекою, резервного копіювання, масштабування ресурсів, управління мережами та інше.

Але при цьому AWS залишається хмарним сервісом, а отже відбувається повна залежність від постачальника хмарних послуг та можливі складнощі при налаштуванні або використанні деяких сервісів. На рис. 1.1 зображено приклад налаштування системи.

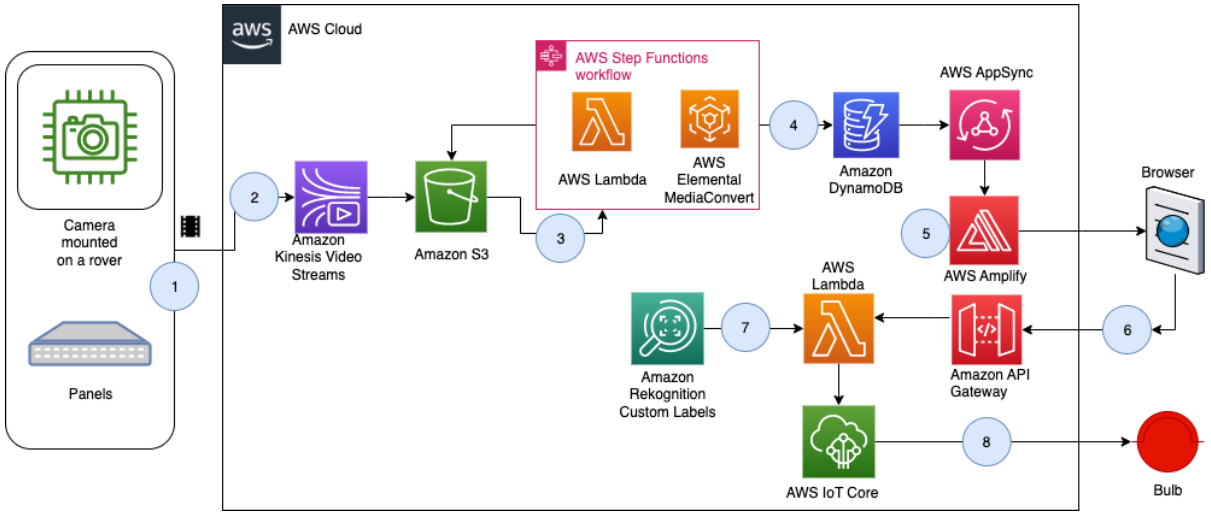


Рисунок 1.1 – Приклад налаштування AWS [7]

GCP – платформа хмарних обчислень від компанії Google. Вона пропонує обчислювальні ресурси, бази даних, аналітику, машинне навчання, мережі та інше.

GCP також надає безліч інструментів для розробки, розгортання та керування хмарними ресурсами. Наприклад, Google Kubernetes Engine (GKE)

надає можливість керування контейнеризованими програмами за допомогою оркестрації Kubernetes. Інші інструменти, такі як Google Cloud Functions та Google Cloud Pub/Sub, дозволяють розробляти та запускати серверні функції та створювати асинхронні повідомлення для обробки даних.

Але можна виділити й такі недоліки як: обмежена кількість запропонованих сервісів в деякий предметних областях на відміну від інших платформ та складність у використанні.

Microsoft Azure також є платформою хмарних технологій тільки від компанії Microsoft. Її основною перевагою є можливість інтеграції з усіма сервісами компанії, адже більшість бізнесів використовують саме продукти Microsoft у своїх системах.

Завдяки великій поширеності системи для неї реалізовано велику кількість сервісів для різних видів діяльності. Але через це, вона дуже складна в навігації і не завжди відповідає очікуванням користувача.

В таблиці 1.1 наведено порівняння навдених рішень між собою, та визначено їх основні переваги та недоліки.

Таблиця 1.1 – Порівняльна таблиця рішень

	AWS	GCP	Microsoft Azure
Функціонал	Великий	Великий	Великий
Безпека	Велика	Велика	Велика
Швидкість розгортання системи	Швидка	Швидка	Швидка
Масштабованість	Має всі можливі види	Має всі можливі види та автоматично визначає навантаження	Масштабування в залежності від потреб
Відмовостійкість	Висока	Висока	Висока

Продовження таблиці 1.1

	AWS	GCP	Microsoft Azure
Складність	Складна	Інтуїтивно зрозумілий інтерфейс	Складна
Інтеграція	Інтегрується з сервісами AWS та має можливість використовувати інші сервіси	Інтегрується з Google і має можливість використовувати інші сервіси	Інтегрується з Microsoft
Підтримка	Велика кількість документації та має велику спільноту	Має документацію та доступна підтримка зі сторони Google	Велика кількість документації і ресурси підтримки від Microsoft
Вартість	Ціна в залежності від використаних ресурсів	Ціна в залежності від використаних сервісів	Має різні моделі оплати

Таким чином розглянуто основні можливості побудови розподілених систем, визначено яким чином можна взаємодіяти з сервісами та порівняно можливу їх можливу інтеграцію з вже існуючими рішеннями.

2 ПРОЕКТУВАННЯ СИСТЕМИ

В цьому розділі описано процес вибору програмного забезпечення та представлені архітектурні рішення, компоненти системи та їх функціональні можливості.

2.1 Опис використаних технологій

За основу застосунку обрано десктопний додаток. Таке рішення забезпечує більш контрольований доступ до системи та безпечне зберігання конфіденційної інформації.

Для реалізації PIC для предметної області «Моніторинг податків» обрано наступні інструменти:

– СКБД (система керування базами даних) PostgreSQL;

– Мова програмування C#;

– середовище розробки Visual Studio Community 2022;

– модулі інтерфейсу WPF (Windows Presentation Foundation);

– технологія WCF (Windows Communication Foundation) для забезпечення комунікації;

– Фреймворк MimeKit для роботи з поштовим сервісом

– Фреймворк Entity Framework Core з бібліотекою класів Npgsql для роботи з СКБД PostgreSQL.

Створення інтерфейсу за допомогою дозволяє швидко верстати графічну оболонку застосунку, використовуючи останні розробки в сфері інтерфейсу Windows та додаткові бібліотеки, такі як Material Design, щоб зробити нативно зрозумілий адаптивний інтерфейс.

СКБД PostgreSQL було обрано з наступних причин:

– Легкий у використанні;

– Низький рівень адміністрування;

– Сумісний з різними платформами, що використовують усі основні мови та проміжне програмне забезпечення;

– Використання віконних функцій.

WCF – це технологія, призначена для проектування, побудови, супроводу та модифікації розподілених програм. WCF – повністю побудована на базі .NET Framework написана з використанням мови C# і є частиною .NET, що необхідно для роботи з модулем WPF. Також в застосунку використовується CoreWCF який повністю комбінується з WCF і використовується для роботи на платформі .NET Core, яку використовує фреймворк Entity Framework Core.

Переваги WCF:

– можливість змінювати окремі частини системи та додавати нові, не торкаючись при цьому інших її частин;

– можливість написання програм із застосуванням викликів функцій типу «Запит-Відповідь»;

– основні концепції безпеки, які забезпечують безпеку сервісів, описані специфікацією WS-Security;

– здатність системи змінювати програмну чи апаратну платформу, не торкаючись інших учасників сценарію розподілених обчислень.[8, 9]

Фреймворк MimeKit забезпечує всі необхідні компоненти для роботи з поштовим сервером. Він надає можливість зчитувати вкладені файли до повідомлень, відфільтровувати їх в залежності від: дати, відправника або за змістом заголовку.

2.2 Інтегрована сервіс орієнтована розподілена система обробки даних

Сучасний стан моніторингу місцевих податків та зборів вимагає постійного вдосконалення та усунення існуючих проблем. Однією з таких проблем є надходження не повної суми коштів в місцевий бюджет територіальної громади, через велику навантаженість податкової служби.

Оскільки територіальна громада складається з декількох населених пунктів, то й дані зберігаються окремо, проте відповідальним за бюджет всієї громади необхідно мати інформацію про всі пункти в одному місці. Звідси витікає наступна проблема – моніторинг даних для всієї громади. Необхідно також враховувати той факт, що територіальні громади постійно розширюють межі своєї території, через що з’являється проблема впровадження існуючих систем моніторингу до нового відділу.

Для даного проекту вирішено створити розподілену систему, яка б мала змогу додавати нові місцеві відділи до існуючої системи моніторингу податків територіальної громади без зупинки процесів роботи інших місцевих відділів. Кожен відділ повинен забезпечити можливість автоматизованого внесення змін в інформаційну систему отримуючи дані з різних сервісів, наприклад для отримання інформації від банку про надходження від платників податків[10].

Тому розробка РІС для моніторингу податків має велику актуальність. Вона забезпечить зручну та ефективну роботу з податковими даними, знизить ймовірність помилок та допоможе виявити шахрайства. Це все покращить якість збору та аналізу інформації, а також забезпечить більш точне та своєчасне виконання податкових обов'язків, що сприятиме розвитку територіальної громади.

Для вирішення поставлених задач визначено такі методи реалізації:

– Процеси взаємодіють між собою за допомогою мережевого протокола SOAP, завдяки чому на базі клієнта достатньо визвати задекларовану в контаркті передачі даних функцію, в результаті якої повернуться визначені дані;

– Для обміну повідомленнями використовується однонаправлений канальний зв’язок взаємодії, так як сервіс не повинен мати можливість надсилати запити, а також це дає змогу зробити паралельним процес обробки даних на базі сервісу;

– Блокування процесів відбувається в синхронному режимі, так як цей спосіб простий в реалізації;

– Процеси поєднуються між собою за рахунок посередника, головним чином для простого масштабування системи. У разі збою кожен вузол стає автономним і працює лише в своїй підмережі до моменту відновлення посередника;

– Розповсюдження змін по всій системі не потрібно так як загальні дані може бачити лише користувач головного відділу в режимі перегляду, а отже за необхідності він може в будь який час оновити стан системи;

– Для реалізації паралельної обробки використовується модель «розподіленої обробки», так як з кожного вузла необхідно отримувати власні дані, що унеможливлює використання інших методів;

– Для координації дій використовується централізована координація, так як система невелика і головний відділ лише один;

– Для забезпечення безпеки системи використовується засоби авторизації та аутентифікації і шифрування на канальному рівні;

– Для звернення до конкретного об’єкту використовується плоска схема іменування, в якій кожен компонент має унікальне ім’я.

2.3 Визначення вимог до функціональних блоків системи

Система повинна складатися з декількох вузлів обробки даних поєднаних між собою, утворюючи розподілену систему. Також вузли повинні мати змогу звертатися до сервісів і об'єднувати відповіді від них в одне ціле.

В якості вузлів виступають місцеві відділи з використанням інформаційної системи «Моніторинг податків», а також один головний відділ. Кожен з місцевих відділів має можливість обробляти повідомлення, надіслані від банку на поштову скриньку відділу, з вкладеними файлами розширення .dbf, який має задекларовану структуру згідно чинного законодавства.

Для демонстрації розподіленості є можливість звернутися до кожного з відділів, отримати від них дані та відобразити їх в аналітичному вигляді.

В якості користувачів системи визначено 3 основні групи:

– менеджер місцевого відділу – відповідає за оновлення даних;

– адміністратор місцевого відділу – відповідає за зміни у місцевому відділі та слідкує за аналітикою;

– адміністратор головного відділу – відповідає за аналітику на рівні територіальної громади.

Усі задачі користувачів перелічені в Додатку А.

Враховуючи поставлені задачі було побудовано архітектуру PIC (рис. 2.1). Вона складається з компоненту маршрутизації та відділів різного рівня, до яких можна звертатися завдяки цьому компоненту. Місцеві відділи складаються з інформаційної системи «Моніторингу податків», сервісу по обробці банківських виписок та клієнта. Головний відділ, в свою чергу, потребує лише загальне відображення інформації, яка зберігається в місцевих відділах, тому клієнт цього відділу звертається до маршрутизатору, який перенаправляє на необхідний місцевий відділ.

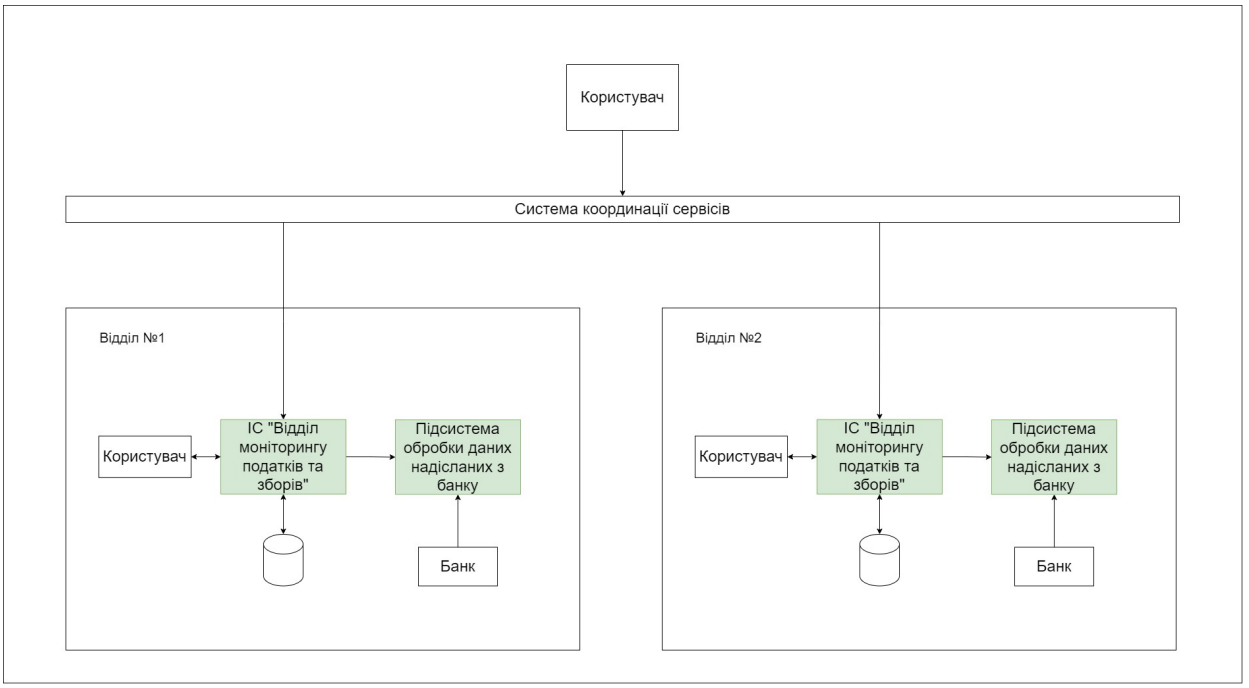


Рисунок 2.1 – архітектура розподіленої системи з предметної області «Моніторинг податків»

Така архітектура будується на основі шаблону проектування – «Брокер». Завдяки цьому система має ряд переваг, наприклад, горизонтальна масштабованість, завдяки чому можна легко додавати нові вузли, забезпечення транзакцій, що впливає на надійність доставки та цілісність даних. При цьому з’являється недолік – вихід з ладу брокеру. При такому випадку звернення до різних відділів зі сторони головного відділу стає неможливим, проте, так як всі основні процеси відбуваються всередині самих відділів, то це вплине лише на відображення даних і ніяк не відобразиться на роботі самих відділів.

Таким чином обрано основні технології для впровадження рішення, визначено постановку задачі та побудовано архітектуру системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

У даному розділі представлено програмну структуру рішення, деталі про реалізацію класів з описом використання технологій та фреймворків, а також розглянуто можливості покращення системи.

3.1 Загальна структура системи

Для реалізації PIC з використанням сервісів в предметній області «Моніторинг податків та зборів» створено структуру рішення забражену на рис. 3.1.1 Вона складається з бібліотек класів та інтерфейсів, та проектів, які використовують їх.

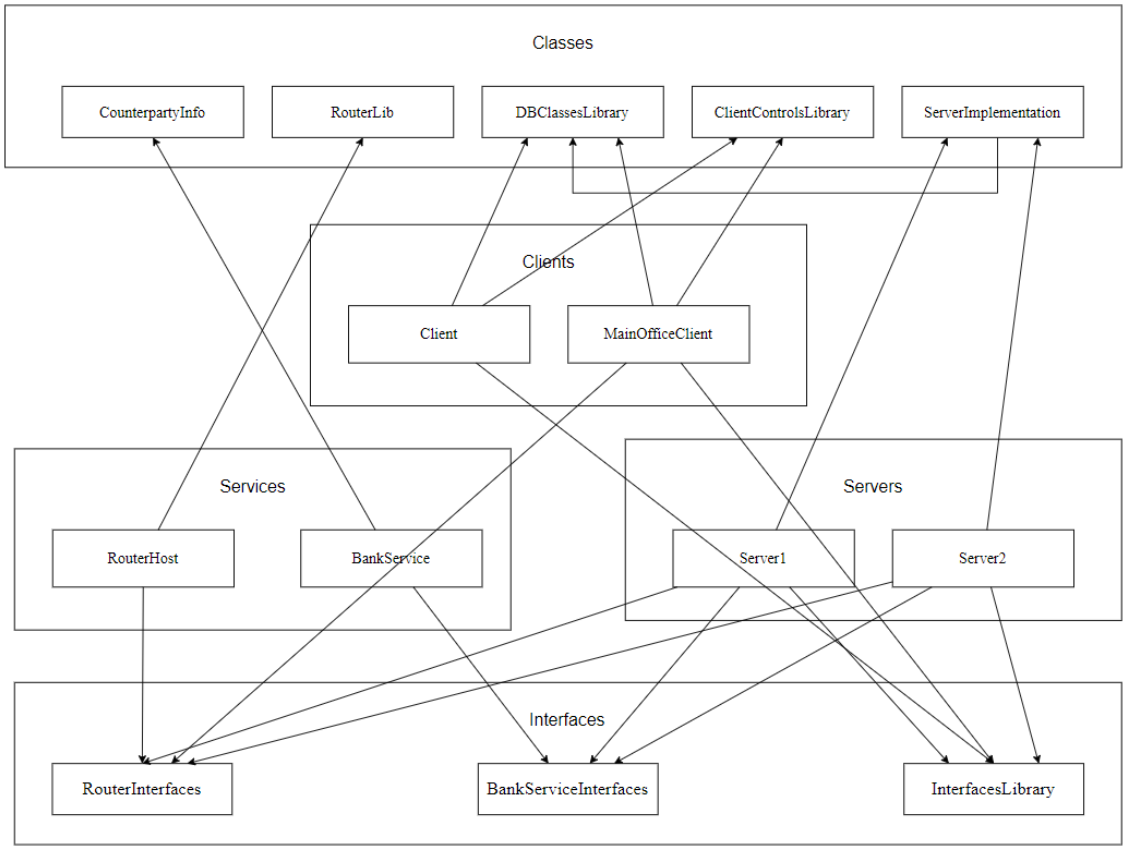


Рисунок 3.1 – структурна схема проекту

Розглянемо детальніше кожен елемент цього рішення.

У папці Classes містяться всі бібліотеки класів, які використовуються для побудови PIC. ClientControlsLibrary містить користувальницькі компонентами, які використовуються для побудови інтерфейсу користувача. В бібліотеці CounterpartyInfo зберігаються класи, які використовуються для побудови виписок з банку. DBClassesLibrary має визначення кожної сутності, яка зберігається в базі даних на мові C# для їх подальшої взаємодії. RouterLib містить класи, які потрібні для роботи сервісу маршрутизації. ServerImplementation містить всі класи, необхідні для роботи серверу. В них реалізоване звернення до БД та до сервісу з обробкою банківських виписок.

У папці інтерфейсів містяться бібліотеки, в яких прописано контракти, за якими відбувається взаємодія в PIC. Бібліотека BankServiceInterfaces потрібна для звернення до банківського сервісу, RouterInterfaces – до сервісу маршрутизації та InterfacesLibrary – до серверу.

У папці Services містяться проекти, які реалізують основні сервіси в системі. BankService відповідає банковському сервісу, в якому видаються всі банківські виписки отримані за певний період. RouterHost – сервіс маршрутизації, який в першу чергу відповідає за переадресацію кожного запиту на потрібний відділ, а як доповнення – може бути легко оновлений при масштабувані PIC.

У папці Servers проекти з різними серверами: Server1 та Server2. Вони потрібні для демонстрації розподіленості, тому функціонують окремо один від одного, використовуючи різні ip-адреси прослуховування та звертаючись до різних серверів баз даних. Кожен з них використовує всю функціональність із бібліотеки ServerImplementation.

У папці Clients розташовані проекти з різними клієнтами системи: Client – в якості клієнту місцевого відділу та MainOfficeClient – в якості клієнту головного відділу. Вони необхідні також для демонстрації розподіленості, але на відміну від серверів, які мають однакову структуру,

було вирішено спростити користувацький застосунок для головного відділу з метою відокремлення непотрібного функціоналу.

3.2 Реалізація базових класів

Головним об’єктом роботи PIC є маршрутизатор, саме він перенаправляє всі запити від одного відділу в інший. Для реалізації такого методу використовується технологія WCF. В додатку Б.1 наведено реалізацію даного сервісу в файлі конфігурації App.config. Для його написання виконано такі дії:

1) Налаштовано цей сервіс на перенаправлення запитів за допомогою впровадження в елемент serviceBehaviors елемент routing по певним фільтрам;

2) Активовано прослуховування ip-адреси для перенаправлення з типом контракту RouterInterfaces.IRouterHost, який є системним контрактом обміну даних;

3) Впроваджено фільтри за якими сервіс повинен визначати відділ призначення. Це робиться в елементі filter з використанням пошуку по шаблону в атрибуті filterData;

4) Додано всі посилання на фільтри до таблиці фільтрів filterTable для сервісу маршрутизації;

5) Впроваджено шифрування каналів WS-security за допомогою прив’язки – wsHttpBinding.

Наступна задача – масштабування системи (задачі АГ6, АГ7, АГ8). Для її реалізації створено клас RouterHostImplementation, частина реалізації якого наведена в додатку Б.2. Він активується при запуску сервісу і прослуховує окрему ip-адресу від сервісу перенаправлення запитів. Цей клас містить методи: AddRegionOffice для додавання нового відділу, EditRegionOffice – для редагування вже існуючого відділу, DeleteRegionOffice – для видалення

відділу та GetRegionOffices – для отримання списку всіх відділів зареєстрованих у системі. Всі ці методи працюють з файлом App.config динамічно змінюючи його структуру або, в випадку методу GetRegionOffices, зчитують необхідні поля.

Ще одним сервісом системи є сервіс по обробці банківських виписок, який вирішує задачі М1 та А1. Основні методи реалізовані в класі BankServiceImplementation (див. додаток Б.3), який реалізує інтерфейс IBankService в якості контракту обміну даними, який має 2 методи: SetConnection та GetDatabyPeriod. В методі SetConnection програма відкриває поштову скриньку клієнта за допомогою методу SetClientConnection. Цей клас налаштований на роботу з поштовим хостом imap.gmail.com, тому відкриватися будуть лише поштові скриньки від Gmail. Метод GetDatabyPeriod витягує всі необхідні файли за певний період часу з поштового серверу, конвертує їх із файлу .dbf у файл .xml, після чого дані серіалізуються в клас OrdersFromFile (див. додаток Б.4) і в результаті повертається список кортежів з ім'ям файлу, датою отримання та його вмістом у двійковому вигляді.

Клас OrdersFromFile містить в собі список класу Order (див. додаток Б.5). Він в свою чергу має визначення кожного стовпця із файлу .dbf, саме завдяки такій реалізації дані легко змінюються з одного формату в інший.

Розглянемо реалізацію серверів. Так як сервер має велику кількість методів, які обробляються клієнтом, то при створені кожного нового нераціонально дублювати весь код. Тому було вирішено весь функціонал перенести в бібліотеку ServerImplementation, і залишити тільки частину розгортання хосту в класі Program, для кожного із серверів (див. додаток Б.6). Для звертання клієнту відкриваються всі канали прослуховування в класі Startup (додаток Б.7). Також для більшої гнучкості додано файл appsettings.json, який містить в собі ip-адресу прослуховування, назву відділу, та строку підключення до БД (див. додаток 3.2). За рахунок цього для зміни

адреси прослуховування або при зміни розташування бази даних достатньо змінити це в файлі без внутрішнього втручання в програму.

```
{"OfficeName": "Region1",
"ConnectionStrings": {"DefaultConnection": "Host=localhost;
Port=5432;Database=TSNAP;Username={0};Password={1}"},
  "WebHostOptions": {
    "ListenIPAddress": "127.0.0.1",
    "ListenPort": 8080}}
```

Рисунок 3.2 – Приклад файлу appsettings.json

Більшість поставлених задач користувачів виконується в класі OrderImplementation, який відповідає за роботи зі звітами.

Для вирішення задач М2, А2 використовується функція GetActiveFilesOrders, яка звертається до серверу для оновлення інформації відносно останнього звернення потім зберігає всі нові виписки на сервері і повертає інформацію про всі виписки, які зберігаються на ньому.

Для вирішення задач М4, М5, А4, А5 використовується функція AddOrders, яка додає всю інформацію про сплату до БД, а при виявлені недостовірності інформації повертає помилкові дані назад.

Для вирішення задач А6, А7, АГ1, АГ2 побудовано представлення income в БД (див. додаток В.1).

Для вирішення задач А9, А10, АГ4, АГ5 побудовано представлення tax в БД (див. додаток В.2).

3.3 Користувацький інтерфейс

Для розуміння роботи з системою розглянемо її за допомогою шляху користувача. На рис. 3.3 зображено, які дії адміністратор місцевого відділу

може виконувати і який шлях для їх виклику необхідно подолати. Для менеджеру місцевого відділу шлях буде значно менше, адже більшість функцій, які виконує адміністратор йому недоступні, тому після входу в систему він буде одразу направлений до менедж-центру і може виконувати лише дії в цій гілці user flow.

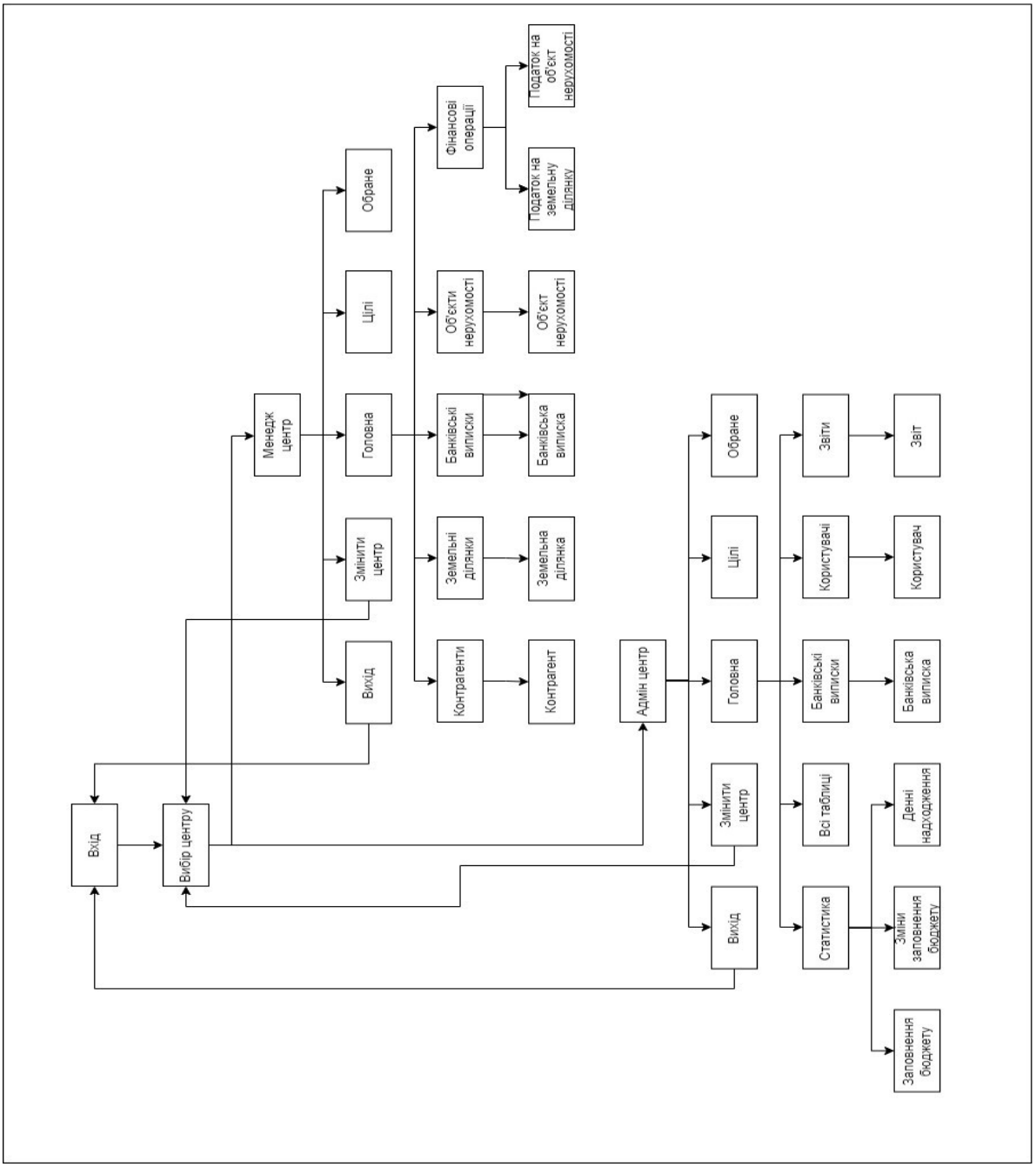


Рисунок 3.3 – User flow для адміністратору місцевого відділу

Адміністратор головного відділу не потребує такої кількості функціоналу. Тому для вирішення його задач побудовано інший шлях користувача зображений на рис. 3.4. Так як головною задачею для цього типу є аналітика, то і основні напрямки шляху завершуються на отриманні аналітичних даних, такі як графіки або звіти.

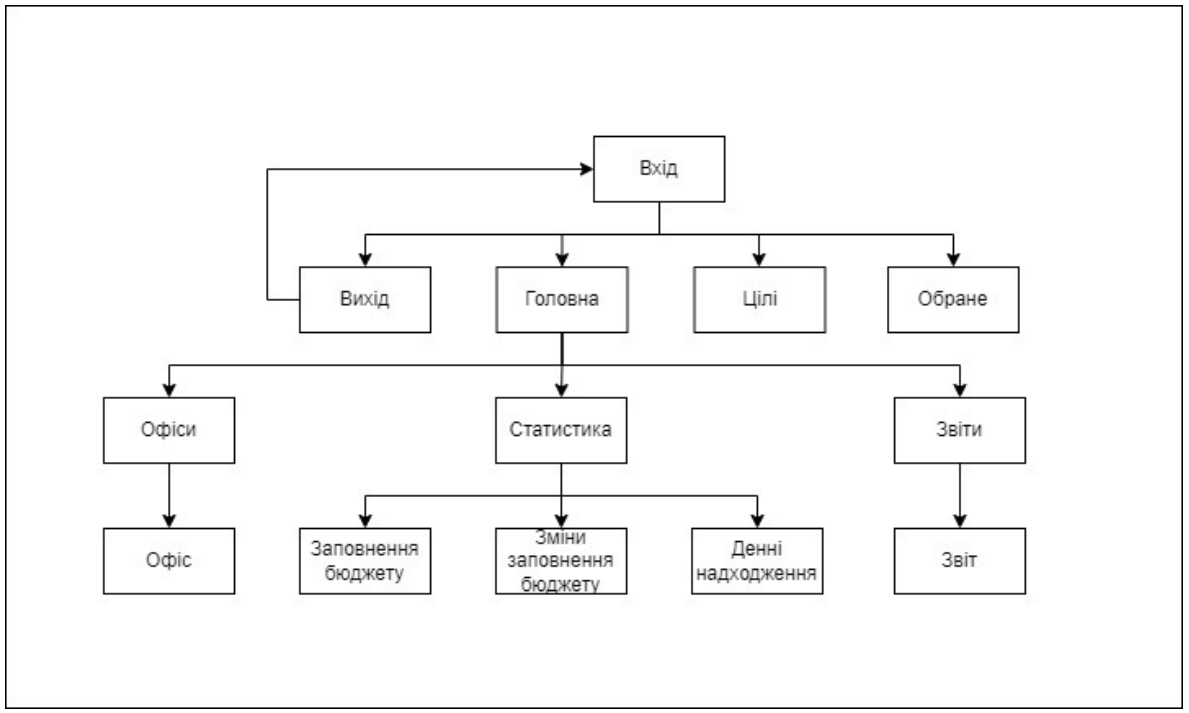
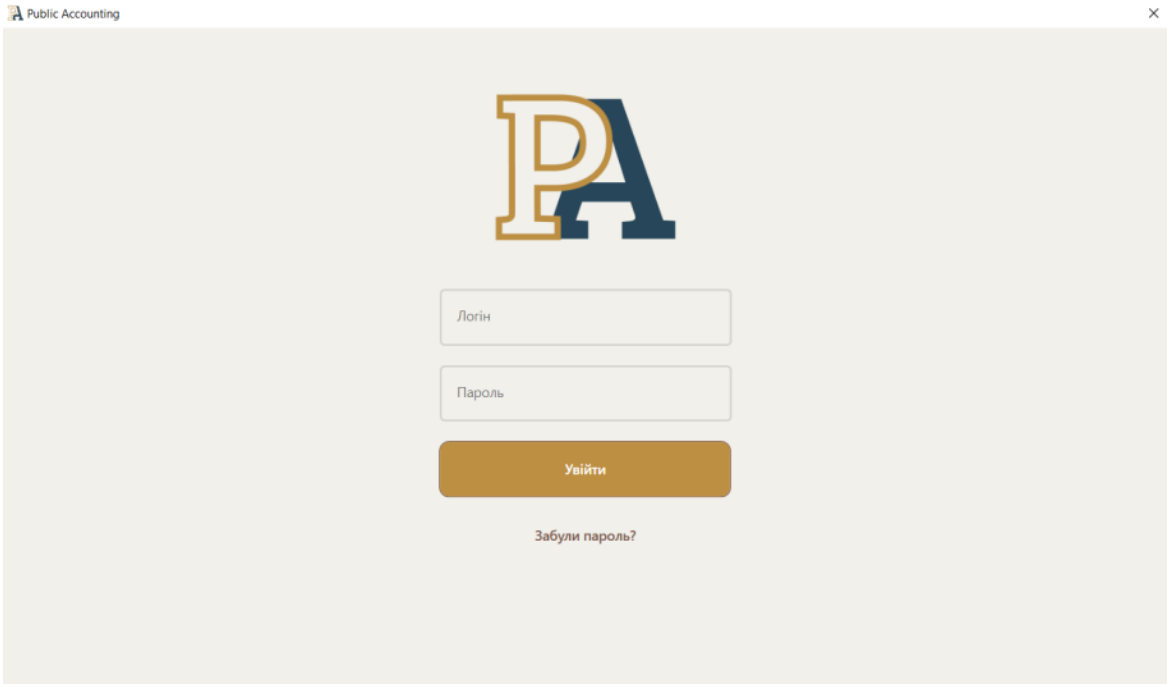


Рисунок 3.4 – User flow адміністратора головного відділу

При запуску програми відкривається вікно авторизації (рис. 3.5, а).

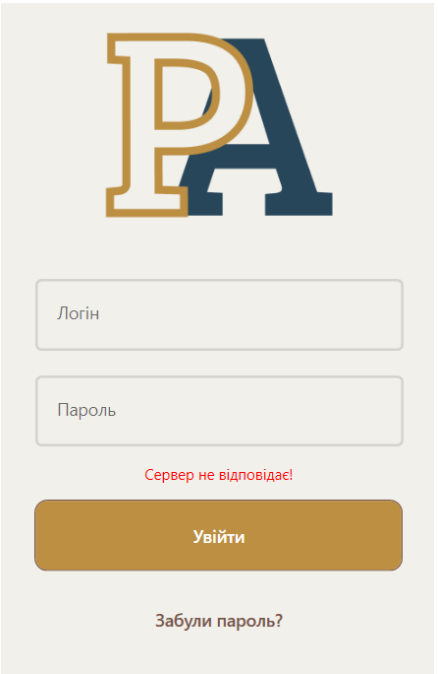
Форма авторизації містить 2 поля вводу: «Логін» та «Пароль», та 2 кнопки: «Увійти» та «Забули пароль». При натисканні на кнопку «Увійти» клієнт з’єднується із сервером та намагається отримати від нього дані. Якщо з’єднатися із сервером не вдалося на формі відображається помилка зображена на рис. 3.5,в. Якщо з’єднання налаштоване, але не існує такого користувача в системі або неправильний пароль, то відображається помилка зображена на рис. 3.5,б. Доступу до наступних вікон користувач не отримає

поки він не пройде аутентифікацію. Далі, в залежності від рівня відділу та групи користувача йому буде запропоновані наступні форми.



а)

б)



в)

Рисунок 3.5, початковий вид вікна авторизації (а), помилка вводу даних (б), помилка з’єднання (в)

Якщо це місцевий відділ і користувач виявився адміністратором – йому буде запропоноване вікно вибору (рис. 3.6). На цьому етапі адміністратор обирає що він буде робити: роботу менеджера, чи особисту.

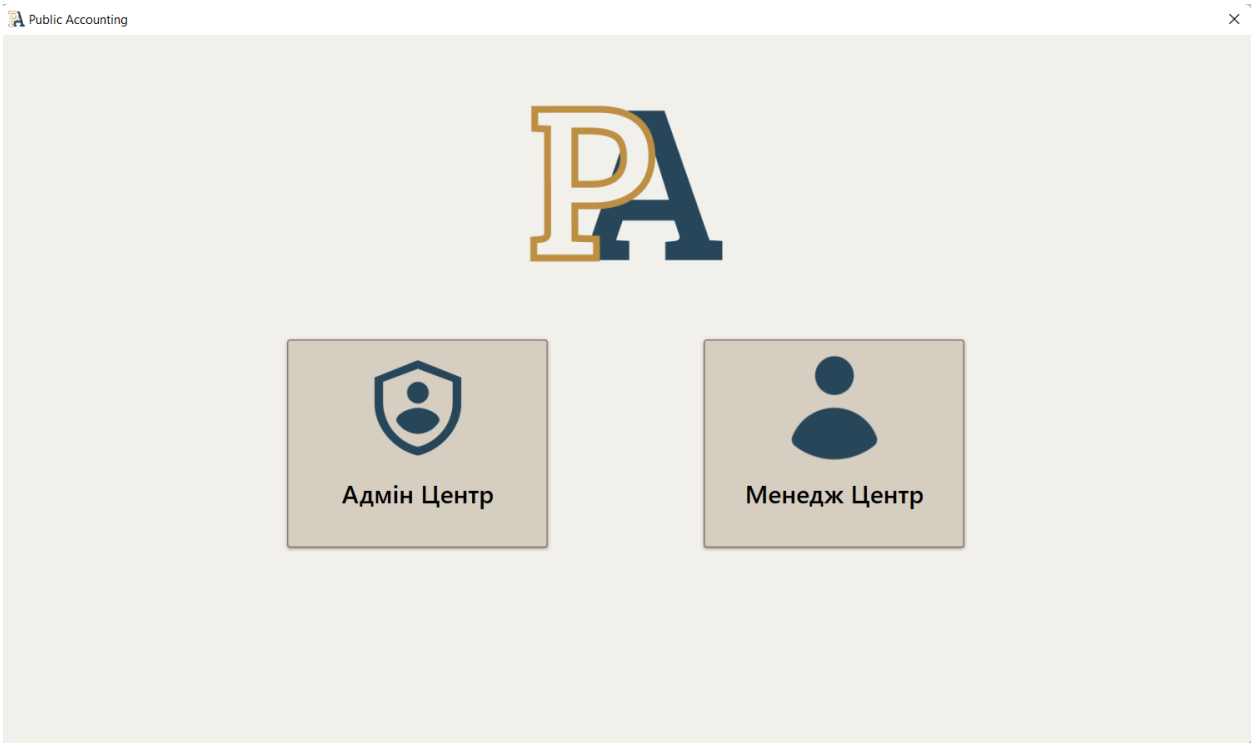


Рисунок 3.6 – Вікно вибору адміністратора

Якщо це місцевий відділ і група користувача – менеджер, або якщо адміністратор вирішив виконати роботу менеджера, то буде відкрито головну сторінку менеджера (рис. 3.7). На цій сторінці розташовані: навігаційна панель, кнопки для переходу на основні таблиці («Контрагенти», «Земельні ділянки», «Об’єкти нерухомості»), кнопка «Фінансові операції», та панель з банківськими виписками.

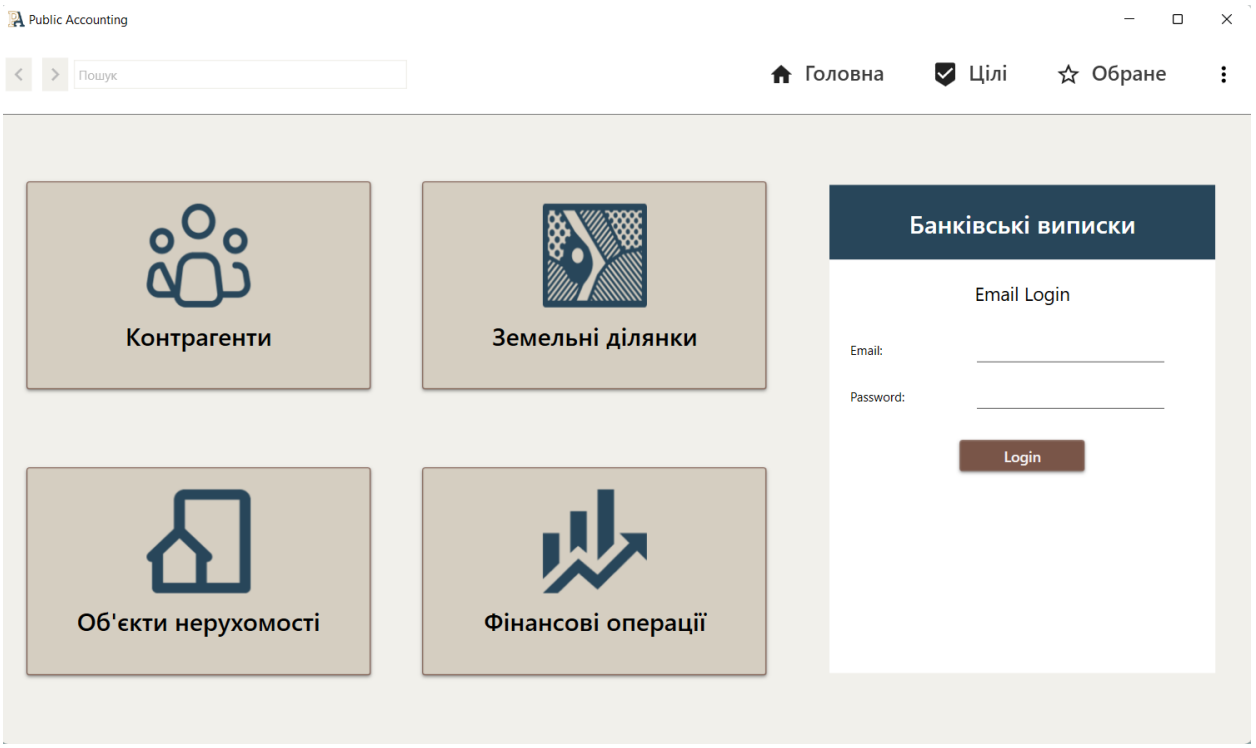


Рисунок 3.7 – Головна форма менеджера місцевого відділу

Якщо в місцевому відділі група користувача – адміністратор, то він має доступ до функцій, які недоступні менеджеру, ці функції розташовані на головній сторінці адміністратора. Вона має вигляд як і у головної сторінки менеджера, проте кнопки зовсім інші (рис. 3.8). Кнопка «Користувачі» направляє до перегляду всіх користувачів, які існують у системі. Кнопка «Всі таблиці» направляє на список усіх таблиць системи, де адміністратор має змогу редагувати дані. Натиснувши на кнопку «Звіти» є змога роздрукувати необхідні звіти. Та перейшовши за кнопкою «Статистика» можна переглянути статистичні дані системи.

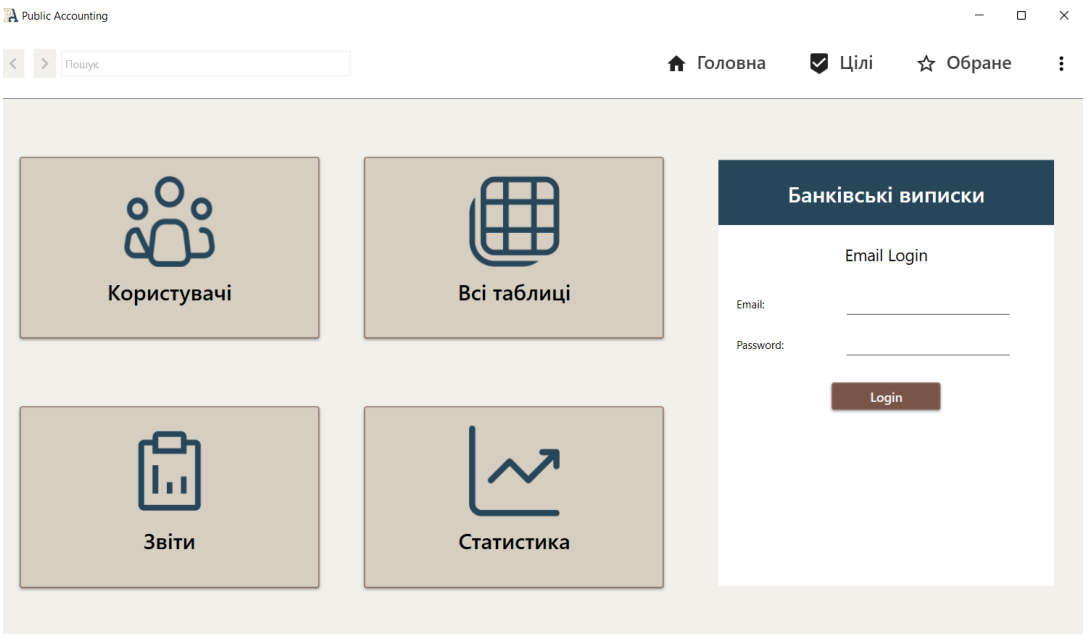


Рисунок 3.8 – Головна форма адміністратора місцевого відділу

Якщо це головний відділ, то буде відкрито сторінку адміністратора головного відділу (рис. 3.9). На цій сторінці розташовані: панель «Офіси» для перегляду офісів в системі та їх редагування, та дві інші панелі, які ідентичні функціоналу адміністратора місцевого відділу.

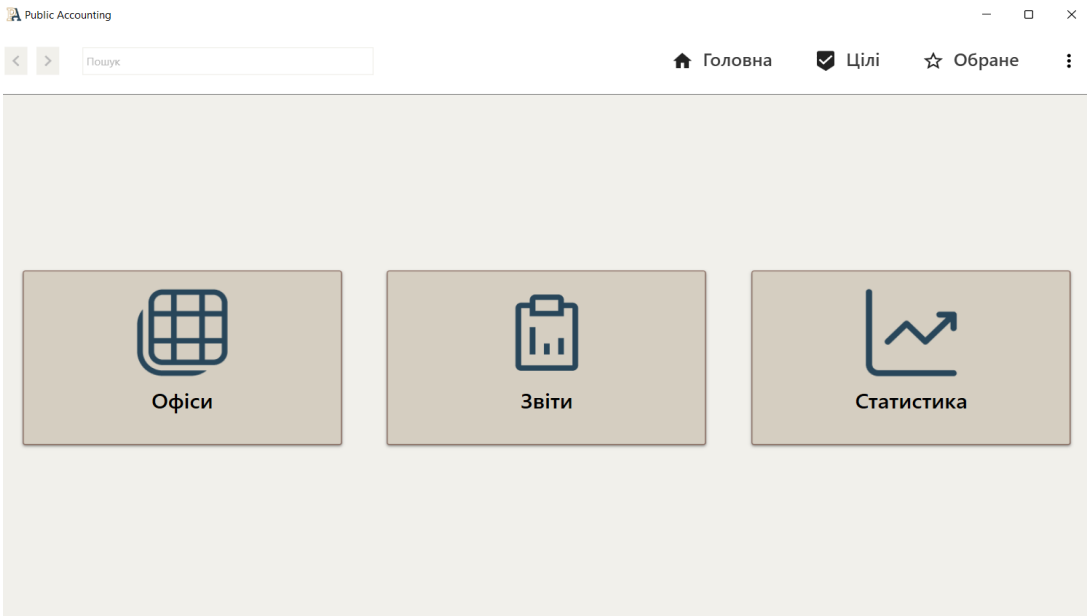


Рисунок 3.9– Головна форма адміністратора головного відділу

Навігаційна панель зображується впродовж роботи на всіх наступних формах. На ній розташована кнопка «Назад» – повернення на попередню форму, «Пошук» – пошук необхідного вікна, «Головна» – повертає на головний екран менеджера, «Цілі» – поставлені задачі менеджера, «Обране» – обрані елементи та елемент «Більше» в якому знаходяться «Змінити центр» та «Вихід».

Натиснувши на одну з цих кнопок для переходу на основні таблиці відкривається вікно перегляду даних таблиць у вигляді детального списку усіх записів (рис. 3.10). В залежності від потреб користувача їх можна сортувати за будь якими критеріями.

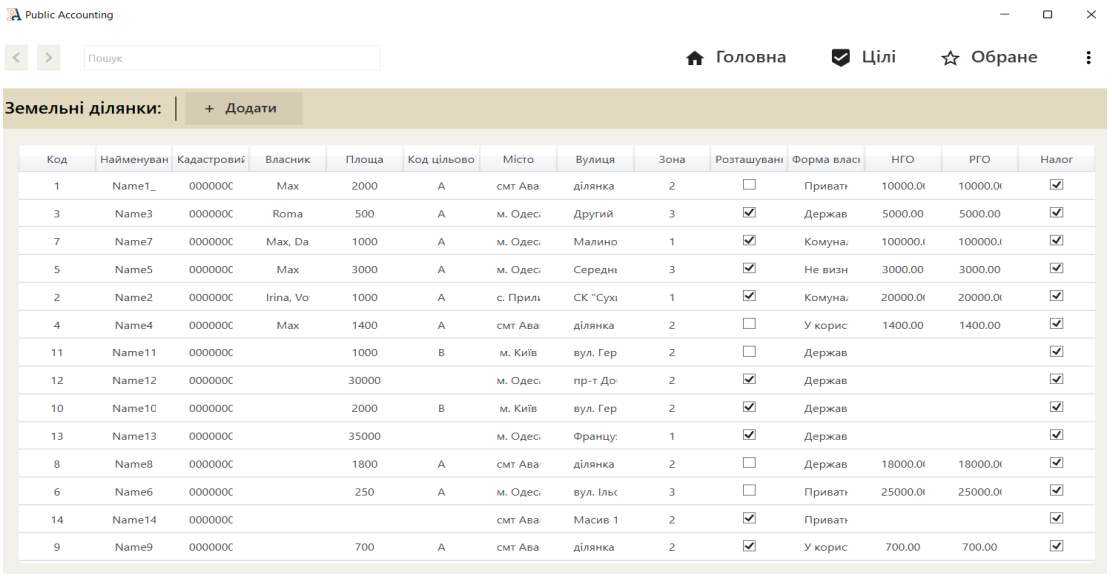


Рисунок 3.10 – Відображення списку записів «Земельних ділянок»

При двійному натисканні на потрібне поле відкривається детальна інформація про цей запис (рис. 3.11). На початку всі поля стають доступні лише для читання. Якщо натиснути кнопку «Редагувати», одразу всі дані будуть доступні для редагування, окрім поля «Код» та наявних історичних таблиць, а сама кнопка змінить свою назву на «Зберегти», яку треба натиснути для збереження даних. Кнопка «Видалити» видаляє запис із БД.

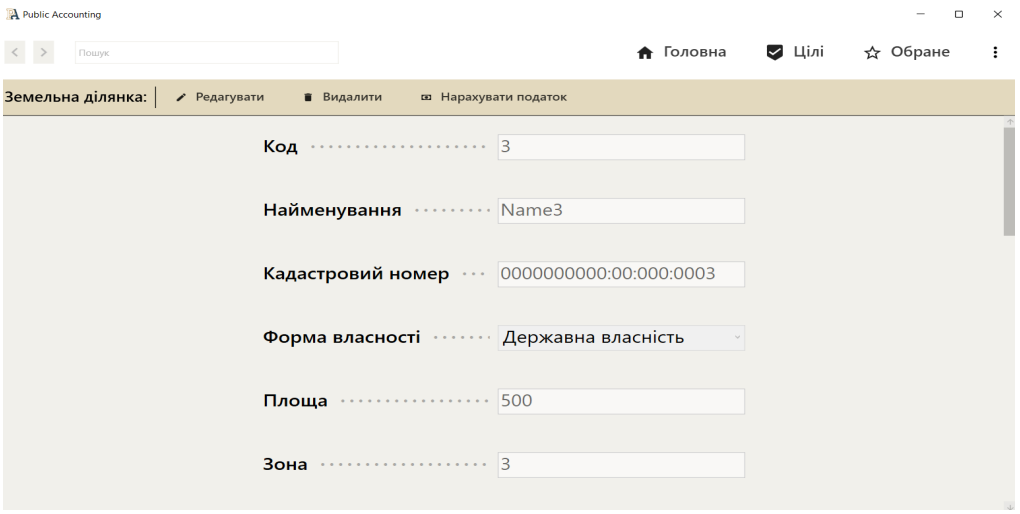


Рисунок 3.11 – Відображення картки об’єкту «Земельна ділянка»

Кнопка «Фінансові операції» відкриває вікно зі списком операцій для обчислень податку (рис. 3.12). При натисканні на одну із них відображається сповіщення, яке перевіряє, чи дійсно треба запустити операцію. Після підтвердження виконується необхідна операція.

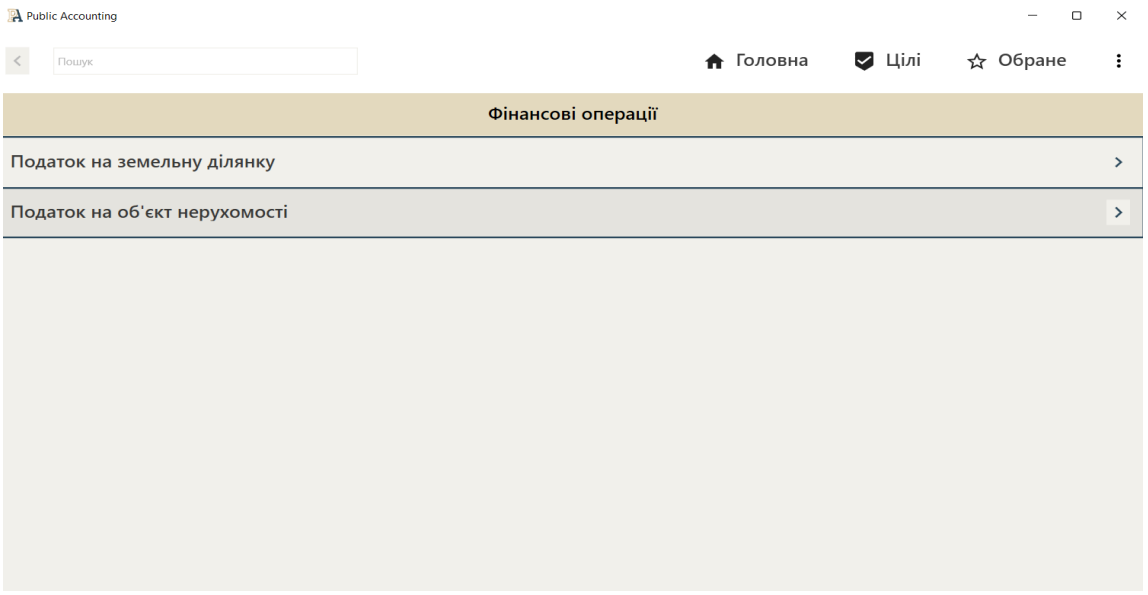


Рисунок 3.12 – Форма «Фінансові операції

Панель «Банківські виписки» дозволяє обробити документи, які прийшли з банку в файлі типу .dbf та додати зміни до БД. Після введення

даних поштового акаунту в цій панелі починає відображатись список всіх недоданих файлів до системи (рис. 3.13). Після вибору одного з файлів відкривається вікно аналогічне рис. 3.10 в якому є можливість перевірити всі платежі і при необхідності відредагувати помилкові дані аналогічно рис.3.11. Після перевірки натискається клавіша «Підтвердити» і всі виписки надсилаються на сервер. При виявленні контрагента, якого немає в системі користувачу буде повернуто недійсні виписки з метою подальшого редагування.

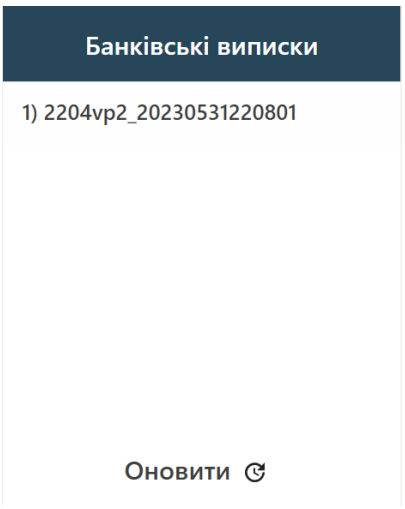


Рисунок 3.13 – відображення не внесених в систему виписок

Панель «Статистика» містить аналітичні дані різного типу, необхідні для аналізу сплати податків (див. рис 3.14). Для цього створено вкладки, при натисканні на які відображаються дані по заповненню бюджету, по щоденним надходженням та по загальним надходженням. Все це можна переглядати поквартально для кожного року, або загалом. Якщо цю вкладку відкриває адміністратор головного відділу, то в нього з’являється можливість переглянути такі дані по кожному активному відділі в системі.

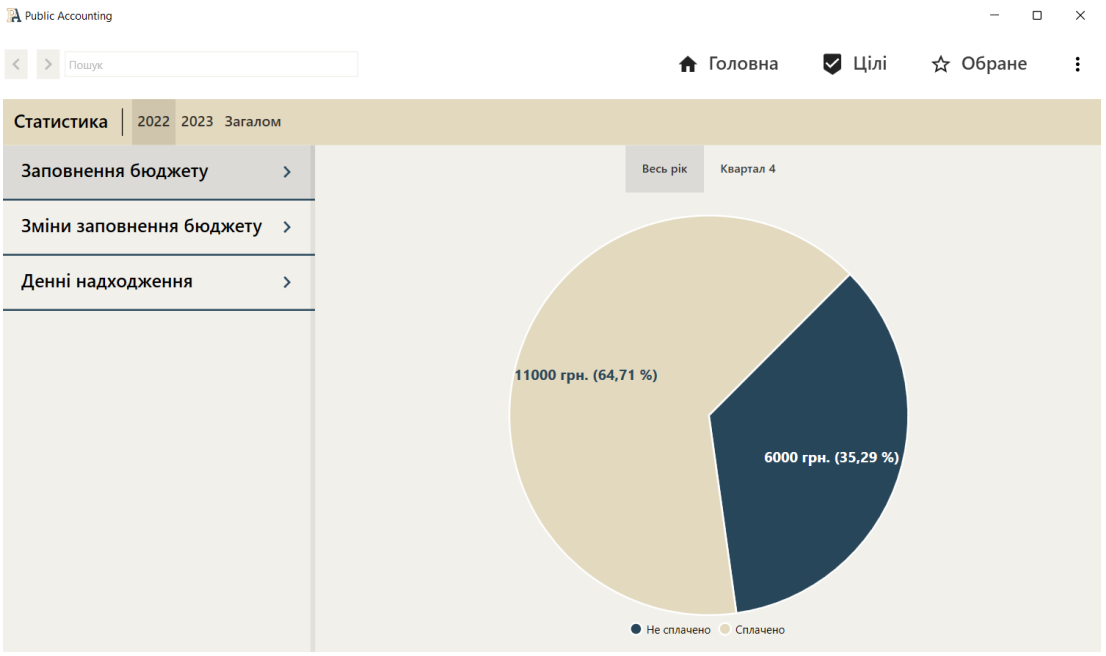


Рисунок 3.14 – відображення аналітичних даних

3.4 Перспективи і можливості подальшого розвитку системи

Головною перевагою розподілених інформаційних систем – їх масштабованість. Тому для покращення системи можна зосередитись на оптимізації поєднання вузлів та впровадження більш складного способу маршрутизації з використанням більше одного засобу переадресування.

За рахунок сервісного підходу, система може бути розширена за допомогою нових функціональних модулів або інтеграції з іншими системами, що може покращити обробку даних. До того ж можна впровадити функції вибору необхідних сервісів самим клієнтом.

Також слід звернути увагу на безпеку в системі і впровадити низку заходів для запобігання більшої групи загроз ніж є зараз.

Якщо розглядати програмне забезпечення, то можна використати нові технології та інструменти, для покращення продуктивності та безпеки.

ВИСНОВКИ

В результаті аналізу предметної області сформульована мета створення РІС і її реалізація, визначено список основних категорій користувачів (менеджер місцевого відділу, адміністратор місцевого відділу та адміністратор головного відділу) і задач, які користувачі повинні вирішувати за допомогою системи.

Для створення РІС використовується технологія WCF з використанням архітектурного стилю «Брокер» та сервісно-орієнтовану архітектуру, для забезпечення захисту обміну повідомленнями використовується шифрування на канальному рівні. Мова програмування – C#. Для взаємодії з даними використовується СКБД PostgreSQL. та технологія WPF для розробки дружнього користувальницького інтерфейсу.

Інтерфейс клієнтської програми розроблений з урахуванням вимог кожної категорії користувачів і надає необхідний функціонал для вирішення відповідних задач. Доступ до даних з боку різних категорій користувачів розмежований за допомогою механізму ролей і привілеїв. Захист від несанкціонованого доступу реалізований шляхом використання механізму аутентифікації і авторизації. Користувач якого немає в системі відкрити функціонал застосунку не може.

За рахунок розподіленої архітектури, до системи можно легко додавати нові відділи з вузлами обробки даних, а для впровадження цих вузлів до самого відділу необхідна мінімальна участь спеціалістів, з подальшою можливістю зміни ір-адреси хосту без їх участі. А завдяки реалізації сервісної структури, до застосунку можна додавати, змінювати чи зовсім видаляти будь які сервіси з урахуванням модифікації інтерфейсу.

За результатами даної роботи опубліковано тези на всеукраїнській конференції та протестовано в Авангардівській селищній раді (додаток Г).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Distributed Systems Concepts and Design / G. Coulouris, J. Dollimore, T. Kindberg, G. Blair // Addison-Wesley, 2012 – 2 с.
2. И. Грегорченко Шардинг и репликация [Електронний ресурс] / 2021 – Режим доступу: https://highload.today/sharding-i-replikatsiya/
3. Difference between Write invalidate protocol & Write Update Protocol (Bus-Snooping) [Електронний ресурс] / 2021 – Режим доступу: https://www.geeksforgeeks.org/difference-between-write-invalidate-protocol-write-update-protocol-bus-snooping/
4. Сухорослов О. В. Матеріали курсу "Розподілені системи" на факультеті комп'ютерних наук Вищої школи економіки. [Електронний ресурс] – Режим доступу: https://github.com/osukhoroslov/distsys-course-hse/tree/master/2020
5. Саша Корж Розподілені інформаційні системи: технології, проектування, забезпечення безпеки [Електронний ресурс] / С. 2-4 – Режим доступу: https://what.com.ua/rozpodileni-informaciini-siste/2/
6. What is SOA? [Електронний ресурс] – Режим доступу: https://aws.amazon.com/what-is/service-oriented-architecture/?nc1=h_ls
7. AWS Architecture Blog / Ramakant Joshi, Shekar Tippur, and Anand Iyer [Електронний ресурс] // 2023 – Режим доступу: https://aws.amazon.com/blogs/architecture/detecting-solar-panel-damage-with-amazon-rekognition-custom-labels/
8. Professional WCF 4: Windows Communication Foundation with .NET 4 / P. Cibraro, K. Claeys, F. Cozzolino, J. Grabner // Wiley, 2010
9. What Is Windows Communication Foundation [Електронний ресурс] / H. Li, N. Schonning, D. Coulter, T. Sherer та ін. // 2021 – Режим

доступу:
https://learn.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf

10. Інформатика, інформаційні системи та технології: тези доповідей двадцятої всеукраїнської конференції студентів і молодих науковців. Одеса, 28 квітня 2023 р. – Одеса, 2023. – С.136-138.

ДОДАТОК А

Задачі користувачів

Таблиця А.1 – Список задач користувачів

Номер	Задача	Вхідні дані	Вихідні дані
Менеджер місцевого відділу			
М1	Завантажити банківські виписки	Пошта, пароль	Ім’я файлу, дата
М2	Переглянути банківські виписки	Ім’я файлу, дата	Список з оплатою
М3	Редагувати помилки в банківських випсках	Код банківської виписки, ІПН, призначення платежу, сума , дата	Оновлена банківська виписка
М4	Отримувати інформацію про наявність контрагента з банківських виписок	Код банківської виписки, ІПН, призначення платежу, сума , дата	Список банківських виписок, в яких зазначені контрагенти не існують в системі
М5	Зберегти банківські виписки в системі	Код банківської виписки, ІПН, призначення платежу, сума , дата	Запис в БД о зарахуванні коштів від контрагента
Адміністратор місцевого відділу			
А1	Завантажити банківські виписки	Пошта, пароль	Ім’я файлу, дата
А2	Переглянути банківські виписки	Ім’я файлу, дата	Список з оплатою
А3	Редагувати помилки в банківських випсках	Код банківської виписки, ІПН, призначення платежу, сума , дата	Оновлена банківська виписка

Продовження таблиці А.1

Номер	**Задача**	**Вхідні дані**	**Вихідні дані**
А4	Отримувати інформацію про наявність контрагента з банківських виписок	Код банківської виписки, ІПН, призначення платежу, сума , дата	Список банківських виписок, в яких зазначені контрагенти не існують в системі
А5	Зберегти банківські виписки в системі	Код банківської виписки, ІПН, призначення платежу, сума , дата	Запис в БД о зарахуванні коштів від контрагента
А6	Отримати звітність про денну сплату податків	-	Стовпчикова діаграма із сумою за кожен день
А7	Отримати звітність про щоденне збільшення сплачених податків	-	Графік сплати податків
А8	Отримати звітність у відсотковому відношенні сплати та нарахування податку	-	Кругова діаграма
А9	Отримати звітність про денне нарахування податку	-	Стовпчикова діаграма із сумою за кожен день
А10	Отримати звітність про щоденне нарахування податку	-	Графік сплати податків
Адміністратор головного відділу			
АГ1	Отримати звітність про денну сплату податків в кожному відділі	-	Стовпчикова діаграма із сумою за кожен день
АГ2	Отримати звітність про щоденне збільшення сплачених податків в кожному відділі	-	Графік сплати податків

Продовження таблиці А.1

Номер	**Задача**	**Вхідні дані**	**Вихідні дані**
АГ3	Отримати звітність у відсотковому в кожному відділі відношенні сплати та нарахування податку в кожному відділі	-	Кругова діаграма
АГ4	Отримати звітність про денне нарахування податку в кожному відділі	-	Стовпчикова діаграма із сумою за кожен день
АГ5	Отримати звітність про щоденне нарахування податку в кожному відділі	-	Графік сплати податків
АГ6	Додати новий регіон в систему	Ім’я, ір-адреса хосту	Обробка нового регіону системою маршрутизації
АГ7	Змінити ір-адресу хоста одного з регіонів	Ім’я, ір-адреса хосту	Обробка зміненого регіону системою маршрутизації
АГ8	Видалити регіон із системи	Ім’я	Регіон видалено з системи маршрутизації

ДОДАТОК Б

Програмна реалізація

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <services>
   <service   behaviorConfiguration="RoutingServiceBehavior"
name="System.ServiceModel.Routing.RoutingService">
<endpoint        address="http://localhost:10000/Router/FromMain"
binding="wsHttpBinding"     name="FromMainRoutingServiceEndpoint"
contract="System.ServiceModel.Routing.IRequestReplyRouter" />
<endpoint      address="http://localhost:10000/Router/FromRegion"
binding="wsHttpBinding"   name="FromRegionRoutingServiceEndpoint"
contract="System.ServiceModel.Routing.IRequestReplyRouter" />
      </service>
      <service name="RouterHost.RouterHostImplementation">
<endpoint address = "http://localhost:10000/Router" binding
= "basicHttpBinding" contract="RouterInterfaces.IRouterHost" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
  <behavior name="RoutingServiceBehavior">
    <serviceDebug includeExceptionDetailInFaults="true" />
                  <routing  filterTableName="routingRules"
routeOnHeadersOnly="false" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <routing>
```

Рисунок Б.1 – Налаштування сервісу маршрутизації у файлі App.config

```
          <filters>
 <filter name="ToRegionOfficeEndpoint" filterType="EndpointName"
filterData="FromMainRoutingServiceEndpoint" />
<filter name = "Region1Pattern" filterType = "XPath" filterData
= "boolean(//*[local-name() = 'OfficeName']/text() = 'Region1')"
/>
                <filter name="Filter_Region1" filterType="And"
filter1="Region1Pattern" filter2="ToRegionOfficeEndpoint" />
          </filters>
          <filterTables>
            <filterTable name="routingRules">
     <add   filterName   =   "Filter_Region1"   endpointName   =
"Region1Endpoint" />
            </filterTable>
          </filterTables>
        </routing>
        <client>
             <endpoint binding="basicHttpBinding" contract="*"
address="http://localhost:8080/Region1/IRegionOffice"
name="Region1Endpoint" />
        </client>
      </system.serviceModel>
    </configuration>
```

Рисунок Б.1, Лист 2

```
    public class RouterHostImplementation : IRouterHost{
    private string Contract = "/IRegionOffice";
    public string AddRegionOffice(string regionName, string
address, string binding){try{
    Assembly executingAssembly = Assembly
.GetExecutingAssembly();
```

Рисунок Б.2 – Частина реалізації класу RouterHostImplementation

```
    Configuration config = ConfigurationManager
.OpenExeConfiguration(executingAssembly.Location);
    XmlDocument xmlDoc = new XmlDocument();
    xmlDoc.Load(config.FilePath);

    XmlNode endpointNode = xmlDoc
.SelectSingleNode($"/configuration/system.serviceModel/client/en
dpoint[@name='{regionName}Endpoint']");
    if (endpointNode != null) throw new Exception("Офіс з таким
ім'ям вже існує");
    XmlNode clientNode = xmlDoc
.SelectSingleNode("/configuration/system.serviceModel/client");
    XmlElement newEndpointNode = xmlDoc
.CreateElement("endpoint");
    newEndpointNode.SetAttribute("binding", binding);
    newEndpointNode.SetAttribute("contract", "*");
    newEndpointNode.SetAttribute("address", address+Contract);
    newEndpointNode.SetAttribute("name",
$"{regionName}Endpoint");
    clientNode.AppendChild(newEndpointNode);
    XmlNode filtersNode = xmlDoc
.SelectSingleNode("/configuration/system.serviceModel/routing/fi
lters");
    XmlElement newFilterNode = xmlDoc.CreateElement("filter");
    newFilterNode.SetAttribute("name", $"{regionName}Pattern");
    newFilterNode.SetAttribute("filterType", "XPath");
    newFilterNode.SetAttribute("filterData",
$"boolean(//*[local-name()= 'OfficeName']/text() =
'{regionName}')");
    filtersNode.AppendChild(newFilterNode);
    XmlElement newFilterNode1 = xmlDoc.CreateElement("filter");
    newFilterNode1.SetAttribute("name",
$"Filter_{regionName}");
```

Рисунок Б.2 , Лист 2

```
     newFilterNode1.SetAttribute("filterType", "And");
     newFilterNode1.SetAttribute("filter1",
$"{regionName}Pattern");
     newFilterNode1.SetAttribute("filter2",
"ToRegionOfficeEndpoint");
     filtersNode.AppendChild(newFilterNode1);

     XmlNode          filterTablesNode          =          xmlDoc
.SelectSingleNode("/configuration/system.serviceModel/routing/fi
lterTables");
     XmlElement        newFilterTableNode        =        xmlDoc
.CreateElement("add");
     newFilterTableNode.SetAttribute("filterName",
$"Filter_{regionName}");
     newFilterTableNode.SetAttribute("endpointName",
$"{regionName}Endpoint");
filterTablesNode.FirstChild.AppendChild(newFilterTableNode);
     xmlDoc.Save(config.FilePath);
ConfigurationManager.RefreshSection("system.serviceModel");
     return "OK";}
     catch(Exception e){
     return e.Message;}}

     public List<RegionOffice> GetRegionOffices(){
     List<RegionOffice>        regionOffices        =        new
List<RegionOffice>();
     try{Assembly               executingAssembly               =
Assembly.GetExecutingAssembly();
     Configuration                    config                    =
ConfigurationManager.OpenExeConfiguration(executingAssembly.Loca
tion);
     XmlDocument xmlDoc = new XmlDocument();
```

Рисунок Б.2 , Лист 3

```
     xmlDoc.Load(config.FilePath);
     XmlNodeList                endpointNodes                =
xmlDoc.SelectNodes("/configuration/system.serviceModel/client/en
dpoint");
     foreach (XmlNode endpointNode in endpointNodes){
     string                    endpointName                  =
endpointNode.Attributes["name"].Value;
     string      regionName      =      endpointName.Substring(0,
endpointName.Length – "Endpoint".Length);
     string                addressWithContract               =
endpointNode.Attributes["address"].Value;
     string                     address                      =
addressWithContract.Substring(0,addressWithContract.Length-Contr
act.Length);
     RegionOffice regionOffice = new RegionOffice{
     Name = regionName,URI = address,};
     regionOffices.Add(regionOffice);}}
     catch{return null;}
     return regionOffices;}}}
```

Рисунок Б.2 , Лист 4

```
     [ServiceBehavior(InstanceContextMode                      =
InstanceContextMode.Single)]
     public class BankServiceImplementation : IBankService{
     public string Username { private get; set; }
     public string Password { private get; set; }
     public string Host { private get; set; }
     private int Port { get; set; }
     private      string      DateFileName      {      get      =>
"lastExecutionTime.txt"; }
     private ImapClient Client;
```

Рисунок Б.3 – Класу BankServiceImplementation

```
    public BankServiceImplementation(){SetDefaultData();}
    ~BankServiceImplementation(){
    if (Client != null) Client.Disconnect(true);}
    public List<(string, byte[])> GetDatabyPeriod(DateTime
startDate, DateTime endDate){
    Console.WriteLine("Have Get request");
    List<OrdersFromFile> list = Read(startDate, endDate);
    if (list.Count == 0) return null;
    List<(string, byte[])> files = new List<(string,
byte[])>();
    foreach (var item in list){
    byte[] file = SerializeToXmlBytes(item);
    string fileName = item.FileName;
    files.Add((fileName, file));}
    return files;}
    public List<OrdersFromFile> Read(DateTime startDate,
DateTime endDate){
    var inbox = Client.Inbox;
    IList<UniqueId> messageIds = GetMailIdsByPeriod(Client,
startDate, endDate);
    List<OrdersFromFile> counterparties = new
List<OrdersFromFile>();
    foreach (var matchingUid in messageIds){
    var message = inbox.GetMessage(matchingUid);
    if (message.Date < startDate || message.Date > endDate)
continue;
    if (!message.Attachments.Any()) continue;
    foreach (var attachment in message.Attachments){
    string dbfFileName = attachment
.ContentDisposition?.FileName ?? "untitled";
    if (Path.GetExtension(dbfFileName).ToLower() != ".dbf")
continue;
```

Рисунок Б.3, Лист 2

```
     SaveFile(attachment, dbfFileName);
     string            xmlFileName            =            Path
.GetFileNameWithoutExtension(dbfFileName) + ".xml";
     try{ConvertFromDbfToXml(dbfFileName, xmlFileName);
     OrdersFromFile               OrdersFromFile               =
ConvertFromXMLData(xmlFileName);
     DateTimeOffset mesageDate = message.Date;
     OrdersFromFile.Date = mesageDate.DateTime;
     OrdersFromFile.FileName                                    =
Path.GetFileNameWithoutExtension(dbfFileName)+$"_{mesageDate.Dat
eTime.ToString("yyyyMMddHHmmss")}";
     OrdersFromFile.Orders=SetCredits(OrdersFromFile.Orders);
     counterparties.Add(OrdersFromFile);
     Console.WriteLine("Data send");}
     catch (Exception ex) {throw ex;}
      finally{
     File.Delete(xmlFileName);
     File.Delete(dbfFileName);}}}
     return counterparties;}
     private List<Order> SetCredits(List<Order> orders){
     foreach (var order in orders){order.S /= 100;}
     return orders;}
     private void SetDefaultData(){
                 Host = "imap.gmail.com";
                 Port = 993;}

     private    IList<UniqueId>    GetMailIdsByPeriod(ImapClient
client, DateTime startDate, DateTime endDate){
     var inbox = client.Inbox;
     inbox.Open(FolderAccess.ReadOnly);
     var    query    =    SearchQuery.DeliveredAfter(startDate)
.And(SearchQuery.DeliveredBefore(endDate.AddDays(1)));
```

Рисунок Б.3, Лист 3

```
    return inbox.Search(query); }
    private bool SaveFile(MimeEntity attachment, string
fileName){
    using (var stream = File.Create(fileName)){
    if (attachment is MimePart mimePart){
    mimePart.Content.DecodeTo(stream);
    return true; }}
    return false;}
    private OrdersFromFile ConvertFromXMLData(string fileName){
    XmlSerializer serializer = new
XmlSerializer(typeof(OrdersFromFile));
    OrdersFromFile OrdersFromFile;
    using (StreamReader stringReader = new
StreamReader(fileName)){
    OrdersFromFile =
(OrdersFromFile)serializer.Deserialize(stringReader);}
    return OrdersFromFile;}

    public void ConvertFromDbfToXml(string dbfFileName, string
xmlFileName){
    string fullPath = Path.GetFullPath(dbfFileName);
    string connectionString =
$"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source={System.IO.Path.GetDirectoryName(fullPath)};Extended
Properties=dBase IV;";
    using (OleDbConnection connection = new
OleDbConnection(connectionString)){
    connection.Open();
    string query = $"SELECT * FROM {dbfFileName} WHERE S > 0";
    using (OleDbCommand command = new OleDbCommand(query,
connection)) {
```

Рисунок Б.3, Лист 4

```
     using (OleDbDataAdapter adapter = new
OleDbDataAdapter(command)){
     DataSet dataSet = new DataSet();
     adapter.Fill(dataSet, "Table");
     dataSet.WriteXml(xmlFileName, XmlWriteMode.WriteSchema);}}
     connection.Close();}}
     private OrdersFromFile Deserialize(string responseContent){
     XmlSerializer serializer = new
XmlSerializer(typeof(OrdersFromFile));
     OrdersFromFile OrdersFromFile;
     using (StringReader stringReader = new
StringReader(responseContent)){
     OrdersFromFile =
(OrdersFromFile)serializer.Deserialize(stringReader);}
     return OrdersFromFile;}
     private void SetClientConnection(){
     if (Client != null) Client.Disconnect(true);
     Client = new ImapClient();
     try{Client.Connect(Host, Port,
SecureSocketOptions.SslOnConnect);}
     catch{
     throw new Exception("Банковський сервіс немає з'єднання з
інтернетом");}
     try{Client.Authenticate(Username, Password);}
     catch{throw new Exception("Введені дані не дійсні");}}
     private byte[] SerializeToXmlBytes<T>(T obj){
     using (MemoryStream memoryStream = new MemoryStream()){
     XmlSerializer serializer = new XmlSerializer(typeof(T));
     serializer.Serialize(memoryStream, obj);
     return memoryStream.ToArray();}}
     private DateTime LoadLastExecutionTime(){
     string filePath = DateFileName;
```

Рисунок Б.3, Лист 5

```
     if (File.Exists(filePath)){
     string dateString = File.ReadAllText(filePath);
     if      (DateTime.TryParse(dateString,      out      DateTime
lastExecutionTime)){
     return lastExecutionTime;}}
     return DateTime.Now.AddDays(-1).Date;}
     private          void          SaveLastExecutionTime(DateTime
lastExecutionTime){
     string filePath = DateFileName;
     File.WriteAllText(filePath, lastExecutionTime.ToString());}
     public string SetConnection(string login, string password){
     try{Username     =     login;     Password     =     password;
SetClientConnection();}
     catch (Exception ex){ return ex.Message;}
     return "OK";}}
```

Рисунок Б.3, Лист 6

```
     [XmlRoot("NewDataSet")]
     public class OrdersFromFile {
     [XmlElement(IsNullable = true)]
     public string FileName { get; set; }
     [XmlElement(IsNullable = true)]
     public DateTime? Date { get; set; }
     [XmlElement("Table")]
     public List<Order> Orders { get; set; }}
```

Рисунок Б.4 – Клас OrdersFromFile

```
      public class Order{
             [XmlElement("FDAT")]
             public DateTime FDAT { get; set; }
             [XmlElement("NMK")]
```

Рисунок Б.5 – Клас Order

```
public string NMK { get; set; }
      [XmlElement("IBAN")]
      public string IBAN { get; set; }
      [XmlElement("NMS")]
      public string NMS { get; set; }
      [XmlElement("OSTF")]
      public double OSTF { get; set; }
      [XmlElement("S")]
      public double S { get; set; }
      [XmlElement("ND")]
      public string ND { get; set; }
      [XmlElement("DK")]
      public double DK { get; set; }
      [XmlElement("REF")]
      public double REF { get; set; }
      [XmlElement("TT")]
      public string TT { get; set; }
      [XmlElement("NAZN")]
      public string NAZN { get; set; }
      [XmlElement("NAZN1")]
      public string NAZN1 { get; set; }
      [XmlElement("NAZN2")]
      public string NAZN2 { get; set; }
      [XmlElement("dapp")]
      public DateTime Dapp { get; set; }
      [XmlElement("DATP")]
      public DateTime DATP { get; set; }
      [XmlElement("IBANK")]
      public string IBANK { get; set; }
      [XmlElement("NB")]
      public string NB { get; set; }
      [XmlElement("NAMK")]
      public string NAMK { get; set; }
```

Рисунок Б.5, Лист 2

```
[XmlElement("OKPO")]
public string OKPO { get; set; }
[XmlElement("UNAM_A")]
public string UNAM_A { get; set; }
[XmlElement("UID_A")]
public string UID_A { get; set; }
[XmlElement("UNAM_B")]
public string UNAM_B { get; set; }
[XmlElement("UID_B")]
public string UID_B { get; set; }
[XmlElement("ADMZNE")]
public string ADMZNE { get; set; }
[XmlElement("REFNB")]
public string REFNB { get; set; }
[XmlElement("TP")]
public string TP { get; set; }
[XmlElement("CTGY")]
public string CTGY { get; set; }
[XmlElement("CTGYDTLS")]
public string CTGYDTLS { get; set; }
[XmlElement("CERTID")]
public string CERTID { get; set; }
[XmlElement("TAXAMT")]
public double TAXAMT { get; set; }
[XmlElement("ADDTLINF")]
public string ADDTLINF { get; set; }}
```

Рисунок Б.5, Лист 2

```
internal class Program{
static void Main(string[] args){ Connection();}
static void Connection(){
IConfiguration configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("appsettings.json")
        .Build();
var webHostOptions = configuration
.GetSection("WebHostOptions").Get<WebHostOptions>();
var host = WebHost .CreateDefaultBuilder(
Array.Empty<string>()).UseKestrel(options =>{
options.AllowSynchronousIO = true;
options.Listen(IPAddress.Parse(webHostOptions.ListenIPAddre
ss), webHostOptions.ListenPort);
}).UseStartup<Startup>()
.Build();
host.Run();} }
public class WebHostOptions{
public string ListenIPAddress { get; set; }
public int ListenPort { get; set; } }
```

Рисунок Б.6 – Запуск хоста на сервері

```
public class Startup{
public void ConfigureServices(IServiceCollection services){
services.AddServiceModelServices();}
public void Configure(IApplicationBuilder app){
IConfiguration configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("appsettings.json")
        .Build();
```

Рисунок Б.7 – Клас Startup

```
     string OfficeName = configuration["OfficeName"];
     app.UseServiceModel(builder =>{
     builder.AddService<AddressFunc>();
     builder.AddServiceEndpoint<AddressFunc,    IAddressFunc>(new
BasicHttpBinding(), $"/{OfficeName}/IAddressFunc");
     builder.AddService<CounterpartyFunc>();
     builder.AddServiceEndpoint<CounterpartyFunc,
ICounterpartyFunc>(new                          BasicHttpBinding(),
$"/{OfficeName}/ICounterpartyFunc");
     builder.AddService<CounterpartyProperties>();
     builder.AddServiceEndpoint<CounterpartyProperties,
ICounterpartyProperties>(new                    BasicHttpBinding(),
$"/{OfficeName}/ICounterpartyProperties");
     builder.AddService<FullData>();
     builder.AddServiceEndpoint<FullData,          IFullData>(new
BasicHttpBinding(), $"/{OfficeName}/IFullData");
     builder.AddService<LandFunc>();
     builder.AddServiceEndpoint<LandFunc,          ILandFunc>(new
BasicHttpBinding(), $"/{OfficeName}/ILandFunc");
     builder.AddService<LandProperties>();
     builder.AddServiceEndpoint<LandProperties,
ILandProperties>(new                            BasicHttpBinding(),
$"/{OfficeName}/ILandProperties");
     builder.AddService<Nace>();
     builder.AddServiceEndpoint<Nace,                 INace>(new
BasicHttpBinding(), $"/{OfficeName}/INace");
     builder.AddService<PropertyFunc>();
     builder.AddServiceEndpoint<PropertyFunc,  IPropertyFunc>(new
BasicHttpBinding(), $"/{OfficeName}/IPropertyFunc");
     builder.AddService<RealpropertyProperties>();
     builder.AddServiceEndpoint<RealpropertyProperties,
IRealpropertyProperties>(new BasicHttpBinding(),
```

Рисунок Б.7, Лист 2

```
      $"/{OfficeName}/IRealpropertyProperties");
     builder.AddService<Specialpurpose>();
     builder.AddServiceEndpoint<Specialpurpose,
ISpecialpurpose>(new                          BasicHttpBinding(),
$"/{OfficeName}/ISpecialpurpose");
      builder.AddService<Settings>();
     builder.AddServiceEndpoint<Settings,          ISettings>(new
BasicHttpBinding(), $"/{OfficeName}/ISettings");
     builder.AddService<OrderImplementation>();
     builder.AddServiceEndpoint<OrderImplementation,  IOrder>(new
BasicHttpBinding(), $"/{OfficeName}/IOrder");
     builder.AddService<BankToServerImplementation>();
     builder.AddServiceEndpoint<BankToServerImplementation,
IBankToServer>(new                            BasicHttpBinding(),
$"/{OfficeName}/IBankToServer");
     builder.AddService<GraphicImplementation>();
     builder.AddServiceEndpoint<GraphicImplementation,
IGraphics>(new BasicHttpBinding(), $"/{OfficeName}/IGraphics");
     builder.AddService<RegionOfficeImplementation>();
     builder.AddServiceEndpoint<RegionOfficeImplementation,
IRegionOffice>(new                            BasicHttpBinding(),
$"/{OfficeName}/IRegionOffice");});};}
```

Рисунок Б.7, Лист 3

ДОДАТОК В

Запити на створення представлень для вирішення задач користувачів

```
CREATE OR REPLACE VIEW public.income
AS
SELECT d.date::date AS date,
COALESCE(sum(counterpartyorder.amount), 0::numeric) AS
incomeperday,
sum(sum(counterpartyorder.amount)) OVER (ORDER BY d.date
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS totalincome
FROM generate_series((( SELECT min(t.date) AS min
FROM ( SELECT counterpartyorder_1.date
FROM counterpartyorder counterpartyorder_1
UNION
SELECT counterpartytax.date
FROM counterpartytax) t))::timestamp with time zone,
CURRENT_DATE::timestamp with time zone, '1 day'::interval)
d(date)
LEFT JOIN counterpartyorder USING (date)
GROUP BY d.date
ORDER BY (d.date::date);
```

Рисунок В.1 – Представлення income

```
CREATE OR REPLACE VIEW public.tax
AS
SELECT d.date::date AS date,
COALESCE(sum(counterpartytax.amount), 0::numeric) AS taxday,
COALESCE(sum(sum(counterpartytax.amount)) OVER (ORDER BY d.date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW), 0::numeric) AS totaltax
FROM generate_series(( SELECT min(t.date) AS min
FROM ( SELECT counterpartyorder.date
FROM counterpartyorder
UNION
SELECT counterpartytax_1.date
FROM counterpartytax counterpartytax_1) t))::timestamp with time zone, CURRENT_DATE::timestamp with time zone, '1 day'::interval) d(date)
LEFT JOIN counterpartytax USING (date)
GROUP BY d.date
ORDER BY (d.date::date);
```

Рисунок В.2 – Представлення tax

ДОДАТОК Г

Довідка про впровадження

АВАНГАРДІВСЬКА СЕЛИЩНА РАДА

вул. Добрянського, 26, смт. Авангард, Одеський район, Одеська область, 67806
тел. (048) 797-25-04, e-mail: avangardtg@odessa.gov.ua, ЄДРПОУ 23211248

16.05 2023 року № 01-27/3

Максиму ГАЛЬЧИНСЬКОМУ
67806, Одеська область, Одеський район
смт Авангард, вул. Урожайна, 15а
тел. +380975709692

ДОВІДКА

про впровадження інформаційної системи

Цим підтверджується, що на підставі Договору на безоплатне тестування програмного середовища від 12.05.2023 р., укладеного між Гальчинським Максимом Вячеславовичем та Авангардівською селищною радою Одеського району Одеської області, 15.05.2023 р. у приміщенні Авангардівської селищної ради та на електронному (комп'ютерному) обладнанні останнього, було проведено тестування спеціалізованої платформи сервісних динамічних конфігурацій програмного середовища інтеграції сервісів у розподіленій інформаційній системі «Моніторингу місцевих податків та зборів», яка була розроблена студентом Одеського національного університету імені І.І. Мечникова **Гальчинським М.В.** під час виконання ним дипломної роботи бакалавра на кафедрі математичного забезпечення комп'ютерних систем.

Данна система в цілому відповідає вимогам, як чинного законодавства України у сфері адміністрування місцевих податків і зборів так і вимогам Авангардівської селищної ради Одеського району Одеської області до такого процесу.

Селищний голова **Сергій ХРУСТОВСЬКИЙ**

Код ЄДРПОУ 23211248

Андрій Кіртока
(048) 797-24-82

Рисунок Г.1 - Довідка про впровадження

Подписано цифровой подписью: Олександр Максимов
Дата: 2023.06.19 10:28:39 +03'00'