Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розробка бізнес-логіки розсилки інформації про факт оплати на базі платіжної системи Приват24

Виконав: студент 6 курсу, групи СПм-61
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

_____ Зашко Б.О.
(підпис) (прізвище та ініціали)

Керівник _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

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

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Приймак М.В.
(підпис) (прізвище та ініціали)

Тернопіль
2020

АНОТАЦІЯ

Кваліфікаційна робота магістра містить: с. – 71, рис. – 26, презентація.

Приват24 API, КРОС-ПЛАТФОРМНА РОЗРОБКА, СИСТЕМА, СИНХРОНІЗАЦІЯ, ASP.NET CORE, REST, XML, EF CORE.

Метою роботи є аналіз технологій для розробки веб-серверів та розробка бізнес-логіки розсилки інформації про факт оплати на базі платіжної системи приват 24.

Наукові методи, використані в роботі, базуються на аналізі статистичних та прогнозованих даних.

Практичні методи полягають у використанні технологій ASP.NET Core, .NET Core, мови програмування C#, середовища програмування Microsoft Visual Studio 2019 з метою проектування, розробки та тестування системи.

В результаті роботи було здобуто знання в галузі платіжних систем, набуто досвід у тсворенні подібних архітектурних рішень, зроблено порівняльний аналіз технологій та побудовано систему згідно отриманих вимог.

ABSTRACT

The qualifying work of the master contains: p. – 71, fig. – 26, presenation.

Privat24 API, CROSS-PLATFORM DEVELOPMENT, SYSTEM, SYNCHRONIZATION, ASP.NET CORE, REST, XML, EF CORE.

The aim of the work is to analyze the technologies for the development of web servers and to develop the business logic of sending information about the fact of payment on the basis of the payment system privat 24.

The scientific methods used in the work are based on the analysis of statistical and forecast data.

Practical methods are to use ASP.NET Core, .NET Core, C # programming languages, Microsoft Visual Studio 2019 programming environment to design, develop and test the system.

As a result, knowledge in the field of payment systems was gained, experience in creating similar architectural solutions was gained, a comparative analysis of technologies was made and the system was built according to the received requirements.

ЗМІСТ

ВСТУП

Темою даної кваліфікаційної роботи є розробка бізнес-логіки розсилки інформації про факт оплати на базі платіжної системи Приват 24.

Дана тема є актуальною для українського ринку загалом та для університету зокрема. Система дасть змогу покращити фінансовий облік всередині університету в таких галузях, як: оплата за навчання, оплата за гуртожиток, оплата за інтернет та ін., через спрощення перевірки платежу (відмова від паперових квитанцій).

Для виконання роботи необхідно дослідити Приват24 API, можливість комунікації з ним, розробити архітектурну модель, та розробити крос-платформне, зручне та легко налаштовне програмне забезпечення з можливостями розширення, що дасть змогу створювати розсилки та зберігати в базі даних інформацію про транзакції.

Об’єктом дослідження є Приват24 API.

Предметом дослідження є спосіб комунікації з Приват24 API для отримання транзакцій по рахунку, їх синхронізація та розсилка інформації про факт оплати.

Методами дослідження в даній роботі є:

− перегляд документації Приват24 API;

− огляд технологій, необхідних для розробки;

− оцінка архітектурних підходів до серверної розробки;

− огляд технічної літератури;

− дослідження домену і виділення моделей предметної області;

− пошук варіантів використання;

− огляд систем автентифікації/авторизації.

Наукове значення даної роботи полягає у дослідженні системи Privat24 API, проектування роботи з такими системами, аргументованості певних архітектурних і технологічних аспектів розробки подібних систем.

Практичне значення роботи полягає в створенні крос-платформної, оптимізованої, швидкодійної системи, що легко піддається конфігурації.

1 АНАЛІТИЧНИЙ ОГЛЯД ТЕХНОЛОГІЙ, НЕОБХІДНИХ ДЛЯ ВИКОНАННЯ РОБОТИ

1.1 Платіжна система.

З розвитком комп’ютерних технологій та банківської сфери, а також обсягів покупок та продаж з’явились перші інтернет магазини, що мали на меті швидко замовити товар без необхідності відвідувати магазин. Такі системи потребували створення нового способу розрахунку, який можна було б здійснювати віддалено та безготівково. У цьому випадку, покупець здійснює переказ коштів на рахунок інтернет-магазину, а магазин отримує інформацію про платіж і готує замовлення. Проблема в тому, що для кожного окремо взятого магазину проблематично і дорого створювати свій унікальний спосіб онлайн розрахунку. Так з’явились перші платіжні системи, які було легко інтегрувати на веб-сайт продавця і, в свою чергу, легко використовувати покупцям.

Наступною проблемою стало те, що для зарахування коштів на банківський рахунок, навіть в сучасних умовах, деколи необхідно декілька днів. При цьому неможливо підтвердити чи спростувати оплату, що б значно сповільнило процес і він б не набув значного поширення. Для вирішення даної проблеми сучасні платіжні системи обробляють інформацію про платіж миттєво. Це означає, що як тільки була проведена оплата, продавець має змогу побачити інформацію про платіж (інфойс), сформувати та відправити замовлення. Інформація про платіж є фактично підтвердженням того, що кошти незабаром надійдуть на рахунок. Насправді ж для зарахування коштів необхідно від одного до трьох днів.

Найвідомішими всесвітніми платіжними системами стали PayPal та Webmoney, що були створені в 1998 році.

Якщо розглядати різні платіжні системи за територіальними ознаками, можна виділити такі, як:

− Внутрішньодержавні платіжні системи – такі, в яких кошти переказуються лише всередині держави та національною валютою держави, або ж організація, що реєструє платіжну систему є резидентом держави.

− Міжнародні платіжні системи – такі, що підтримують багато валют, можуть автоматично проводити валютні конвертації та доступні для використання в багатьох країнах. Для визнання системи міжнародною необхідно, щоб її робота здійснювалась на території двох чи більше країн.

− Внутрішньобанківські платіжні системи – такі, що створюються та підтримуються банком і здійснюють переказ коштів, базуючись на внутрішніх документах банку. Такі системи переважно дають змогу перераховувати кошти всередині банку без комісій [1].

1.2 Електронна платіжна система Privat24

В Україні на сьогоднішній день існує багато різних платіжних систем, які є як внутрішньодержавними, так і внутрішньобанківськими. Серед найпопулярніших можна виділити такі, як: «FLASHPAY», «ГЛОБУС», «Wellsend», «TELEGRAF», «IBOX», «FORPOST», «iPay», «LiqPay», «EasyPay» та інші. Проте однією з найбільш популярних та зручних є платіжна система Privat24, внутрішньобанківська платіжна система банку «Приватбанк». Своєму широкому поширенню вона завдячує банку «Приватбанк», оскільки саме цей банк мав значний вільний капітал, найбільший рейтинг довіри і можливість для розробки платіжної системи.

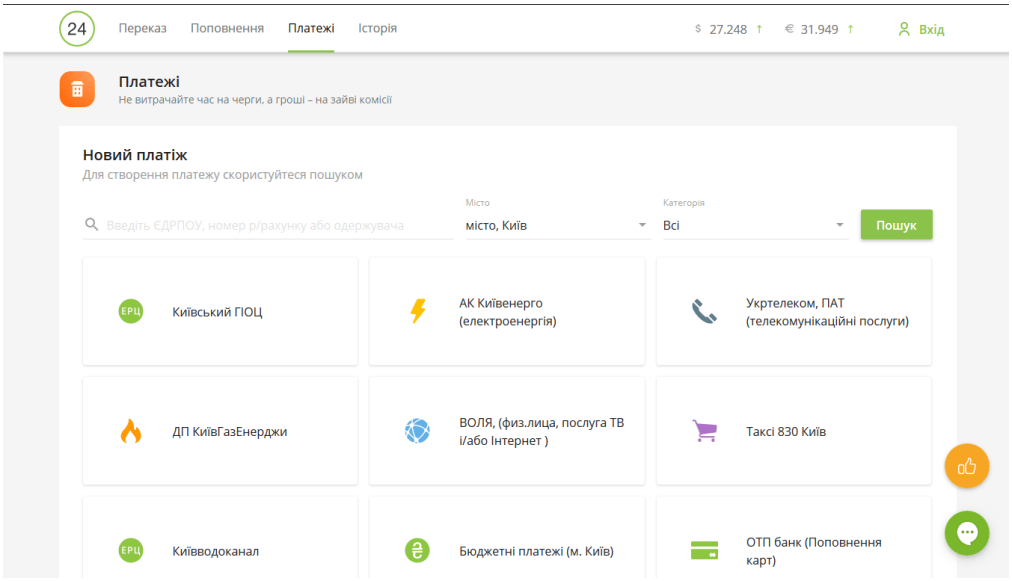

Рис. 1.1 – Платіжна система Privat24

Privat24 – одна з найперших платіжних систем в Україні, що була запущена в 2001 році (рис. 1.1). Тоді авторизація здійснювалась за допомогою коду з SMS повідомлення, а веб-сайт дозволяв перевірити стан рахунку та отримати виписки, також здійснити переказ за реквізитами. Згодом, з розвитком технологій та банківських стандартів було додано перекази на картку, а також можливість здійснювати перекази у іноземній валюті з віртуального рахунку.

У 2010 році великою подією стала можливість використовувати Privat24 не лише за допомогою веб-сайту, а й з мобільного додатку, який був випущений під iOS та Android. Це стимулювало ріст клієнтів системи та її розширенню, оскільки аналогічного конкурента за функціоналом тоді не існувало.

На сьогодні Privat24 – одна з найпоширеніших платіжних систем, що дозволяє з легкістю сплатити комунальні платежі, штрафи, поповнити мобільний рахунок, здійснити переказ на інший рахунок, на картку іншого банку, а також однією з найпоширеніших функцій є можливість зробити платіж щомісячним. Тобто можна створити регулярний платіж на фіксовану суму, який буде проводитись з заданим періодом, наприклад, щомісячне поповнення мобільного рахунку або щомісячна сплата комунальних платежів [2].

1.3 Веб-технологія RESTful API.

На сьогоднішній день комунікація з веб-сервером є основним способом взаємодії пристроїв та сервісів в мережі інтернет. Зазвичай для комунікації використовуються такі протоколи, як:

– HTTP – в основному для передачі текстових даних. Початково був створений та призначений для отримання веб-сторінок. Сьогодні використовується для передавання будь-якого вигляду текстових даних. Окрім HTML веб-сторінок, HTTP може працювати з форматами XML, JSON, YAML, та іншими поширеними форматами серіалізації даних.

– WebSockets – протокол, призначений для двостороннього потокового передавання даних. На основі даного протоколу працюють нотифікації в реальному часі та всі ігрові програми, що потребують безперервної передачі великих об'ємів даних. З'єднання встановлюється за допомогою HTTP-запитів та рукостискання (handshake), після чого клієнт та сервер домовляються про створення websocket каналу передавання даних, і вже з його використанням далі продовжують комунікацію.

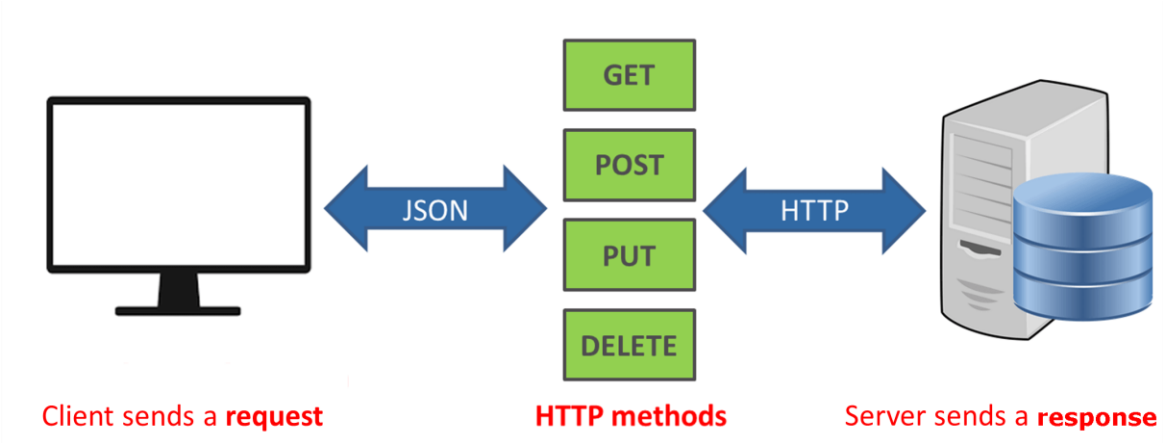

Рис. 1.2 – RESTful Web API

RESTful API – це не протокол, а швидше спосіб архітектурного проектування веб-сервера, що базується на протоколі HTTP, проте має на меті отримання не веб-сторінок, а серіалізованих об’єктів в текстовому форматі (рис. 1.2). При цьому використовуються базові методи HTTP протоколу, такі як GET – отримати об’єкт, PUT, POST – створити або оновити об’єкт, DELETE – видалити об’єкт.

Є два основні підходи для створення веб-сервісів:

− Stateful web-service – веб-сервіс, що зберігає стан даних для конкретного клієнта в пам’яті і може його використовувати для подальших запитів. Прикладом таких сервісів є Java Enterprise Edition (Java Beans). У випадку stateful сервісів, створених за допомогою J2EE сервер зберігає дані у вигляді об’єктів «beans» (з англійської - боби), та при наступному зверненні використовує дані, збережені в цих об’єктах. До прикладу, інтернет-магазин може використовувати таку технологію щоб запам’ятовувати корзину клієнта при переході між різними товарами.

− Stateless web-service – веб-сервіс, що не зберігає жодних даних між запитами одного окремо взятого клієнта. В цьому випадку клієнт сам повинен щоразу надсилати контекст – до прикладу, тримати корзину локально, або щоразу зберігати її в базі даних. Такий підхід є більш поширений і такі сервіси легше масштабувати, оскільки в кінцевому результаті клієнт може звертатися до різних екземплярів одного й того ж сервіса і отримувати аналогічні дані. При цьому клієнт обов’язково повинен передавати дані авторизації, до прикладу, JWT-токен.

RESTful API, як принцип, є stateless веб-сервісом, що дає широкі можливості масштабування та інтеграції. Саме тому такий підхід є найбільш поширеним і більшість платіжних систем надають можливість інтеграції та розробки додатків з використанням RESTful API [3].

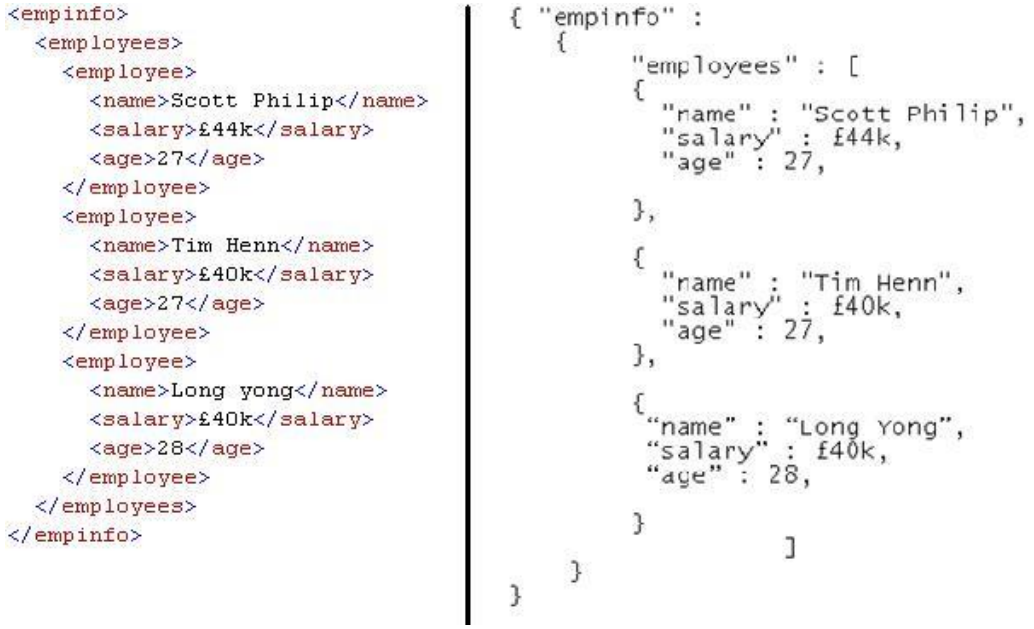

Рис. 1.3 – Порівняння JSON та XML форматів серіалізацій

Основним форматом серіалізації в RESTful API є JSON. Є й інші варіанти, як от XML, але так як логіка веб-сайтів написана мовою JavaScript, JSON є простим та зручним для використання, оскільки просто серіалізується та десеріалізується в об’єкти на клієнтській стороні. Також XML містить багато надлишкових даних, таких як початковий та кінцевий тег кожної окремо взятої властивості чи об’єкта, що робить розмір даних набагато більшим і, відповідно, передачу самих даних – набагато повільнішою (рис. 1.3).

1.4 Платформа ASP.NET Core

Для створення веб-серверів існує багато вже готових рішень та технологій, при чому для розробки можна використовувати різні мови програмування.

У даній роботі було вибрано платформу ASP.NET Core та мову програмування C#.

C# – об’єктно-орієнтована мова програмування, створена компанією Microsoft. Синтаксис мови схожий на мову програмування C та C++ з деякими

відмінностями. Проте на відміну від C та C++, програми, написані мовою C# не компілюються в бінарний виконуваний файл, а лише в певний проміжний байт-код. Дана концепція була запозичена в мови програмування Java та віртуальної машини JVM, що працює схожим чином. Такий байт-код називається MSIL – Microsoft Intermediate Language. При виконанні запускається віртуальне середовище – своєрідний інтерпретатор байт-коду, який і використовується для його інтерпретації та можливості запуску на кожній окремо взятій платформі та апаратному забезпеченні. Такий підхід дає змогу не прив'язуватись до особливостей процесора і виконувати однаковий код на різних архітектурах та апаратних пристроях (рис. 1.4).

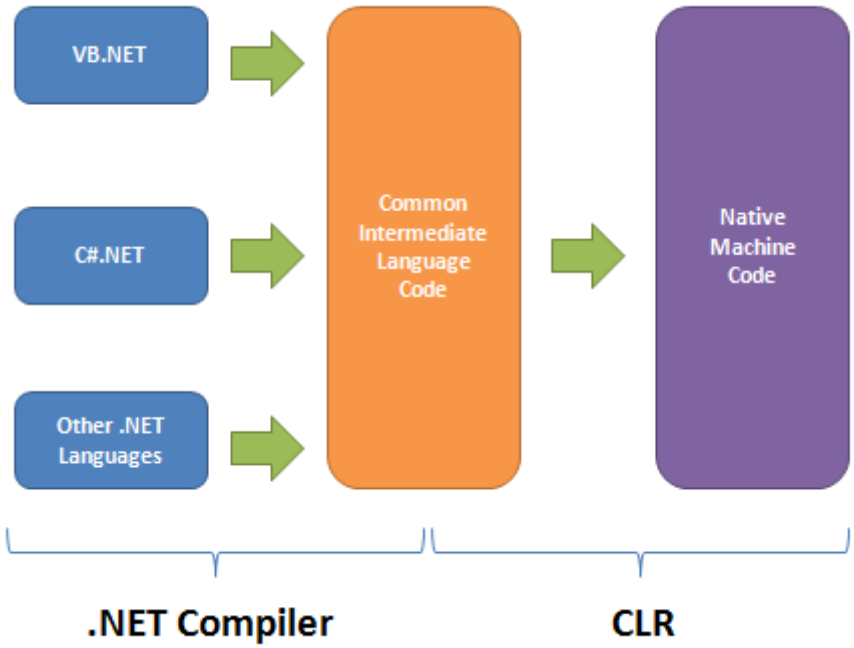

Рис. 1.4 – Принцип роботи віртуальної машини .NET CLR

Технологія, що була описана вище, називається .NET Framework, і включає в себе компілятор C#, інтерпретатор та загальні системні бібліотеки, що містять всі необхідні системні класи та операції. Окрім мови програмування C#, програмувати під .NET Framework можна також мовами Managed C++, Visual Basic, Python (IronPython), проте .NET Framework найчастіше асоціюється з мовою

програмування C#, оскільки вона є зручною і простою в вивченні. Managed C++ використовується переважно тоді, коли необхідно використати бібліотеку, написану мовою C/C++, в .NET програмі. Для цього створюється Managed C++ обгортка над бібліотекою, що дублює її виклики, а вже ця обгортка включається як окремий модуль в .NET програму. Проте є суттєвий недолік платформи .NET – початково вона розроблялась лише для ОС Windows і бібліотеки .NET Framework є тісно інтегровані з Windows API, що внеможливлює запуск додатку на платформах Linux та MacOS.

.NET Core – наступне покоління .NET Framework, що має аналогічну концепцію компіляції та запуску, проте, на відміну від попередньої технології дає змогу запускати програму не лише на Windows, а й на Linux, MacOS, смартфонах, і, в перспективі, на інших видах пристроїв.

ASP.NET Core – веб-технологія, що дозволяє створювати веб-сервіси, зокрема ASP.NET Core WEB API – stateless сервіс для отримання і збереження даних.

При написанні коду основними структурними компонентами є контролери – об’єкти, що є вхідними точками обробки запитів. Окрім цього, можна налаштувати систему фільтрів, наприклад, для авторизації користувача. При цьому запит спершу потрапляє на фільтр і за умови проходження фільтра – перенаправляється на контролер. Якщо ж запит не задовільняє умови фільтра (користувач не автентифікований або не має доступу до певного ресурсу чи не володіє необхідною роллю), то є можливість одразу повернути помилку, а в контролері вже можна бути впевненим, що користувач має доступ до даного функціоналу.

ASP.NET Core легко масштабується, що дозволяє створювати системи високого навантаження, а також легко встановлюється на сервер. З підтримкою ОС Linux ASP.NET Core потребує значно менше фізичних ресурсів, оскільки серверні версії Linux врізані і займають значно менше дискового простору. Більше того, ASP.NET Core можна встановлювати всередині контейнерів, що робить процес розгортання швидким і зручним [4].

1.5 ORM

Для роботи з базою даних необхідно передбачити запити вставки, отримання, видалення та оновлення. Використання текстових запитів напряму є найпростішим варіантом, проте є багато недоліків такого підходу:

− Потрібно добре володіти мовою запитів SQL;

− При зміні структури даних потрібно редагувати всі наявні запити до певної таблиці;

− Такі запити є вразливими, оскільки при прямій конкатенації параметрів та тіла запиту можливо зробити певні підстановки, наслідком яких може бути частковий або повний несанкціонований доступ до бази даних.

Замість стандартного підходу написання запитів до БД можна використати ORM.

ORM – об’єктно-реляційне відображення, це технологія, яка передбачає відображення структури бази даних та операцій в об’єктах та методах. При використанні даного підходу необов’язково володіти SQL, а також знати особливості зв’язків між таблицями та типи запитів. Окрім того, переважно всі ORM-рішення є захищеними від SQL-ін’єкцій, оскільки всі запити формуються під капотом і підставляють надані параметри, екрануючи при цьому заборонені символи. Таке рішення є безпечним для використання та суттєво спрощує і пришвидшує розробку.

Процес роботи з ORM виглядає наступним чином: запит формується в вигляді звичайного коду роботи з колекцією даних (у випадку C# - використовується LINQ to Entities), при цьому фільтри та умови задаються динамічними виразами. Після запуску запиту, бібліотека ORM визначає сутність, над якою проводиться операція і знаходить необхідну таблицю. Далі, на основі динамічних виразів будуються умови мовою SQL (як от WHERE %field% = %value%). Даний запит відправляється на обробку до сервера баз даних, а після отримання результату відбувається зворотня конвертація – отримана таблиця з

даними співставляється з класом даних, створюються відповідні об’єкти і заповнюються відповідні поля, причому переважна більшість ORM рішень не потребує явного зазначення назви колонок та може автоматично визначати їх з назви властивості об’єкту [5]. В кінцевому результаті, робота з базою даних за допомогою ORM зводиться до створення об’єктів та визначення необхідних операцій, а рутинна генерація та виконання запитів забезпечується самою бібліотекою.

1.6 Entity Framework Core

Entity Framework Core є ORM-рішенням від компанії Microsoft, створеним на базі Entity Framework для платформи .NET Core, що є найпопулярнішим та найзручнішим для використання, особливо в .NET програмах, написаних мовою C#.

В якості бази даних для Entity Framework можуть бути використані різні варіанти, наприклад Microsoft SQL Server, SQLite, MySQL, Azure Cosmos DB, PostgreSQL, Oracle, та інші. Така гнучкість забезпечується тим, що доступ до бази даних в Entity Framework є абстракцією, яка може бути спеціалізована за допомогою різних адаптерів. Так, для SQL Server можна використати пакет Microsoft.EntityFrameworkCore.SqlServer, в свою чергу для Oracle - Oracle.EntityFrameworkCore.

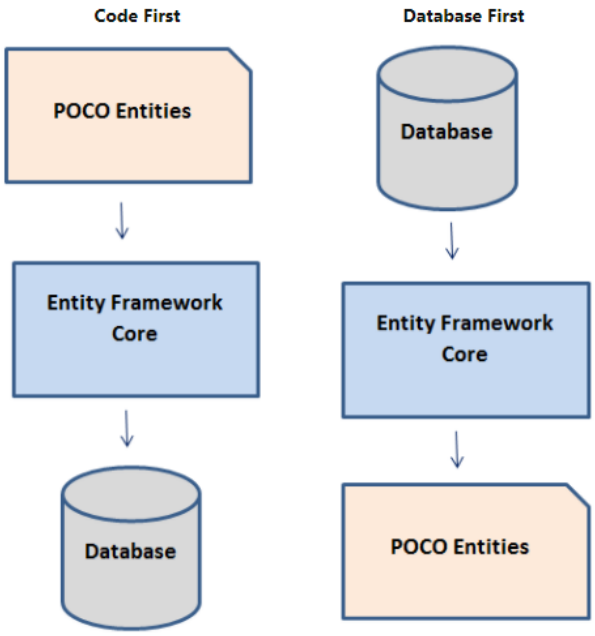

Рис. 1.5 – Різні підходи використання Entity Framework Core

Для початку роботи з Entity Framework потрібно визначити підхід, який буде використовуватись для створення та оновлення структури даних. В залежності від конкретного завдання пропонується кілька на вибір (рис. 1.5):

– Code-First – спочатку створюються класи-моделі даних (об’єкти, що містять лише властивості, які будуть збережені в базі даних). Кожен такий клас представляє окрему таблицю в базі даних. Далі створюється контекст бази – клас, в якому зазначаються колекції, кожна з яких представляє аналог набору даних в певній таблиці. Останнім задається стрічка підключення до БД, що містить інформацію про сервер, користувача та пароль, або, у випадку локальної тестової бази – файл бази даних. Після запуску такого рішення на сервері або у файлі буде автоматично створено необхідну структуру даних з таблицями. Зв’язки даних також можна задати за допомогою атрибутів або засобу FluentAPI – способу, що дає розширені можливості валідації даних та створення зв’язків між таблицями.

– Database-First – підхід, що передбачає наявність вже готової бази даних. Такий підхід буде актуальним, коли вже існувало рішення і база даних заповнена, наприклад, при переході від SQL-запитів до ORM. При цьому на основі структури БД генеруються класи даних і зв’язки між ними.

− Model-First – підхід, що бере за основу графічно створену модель даних. Відмінність даного підходу від Code-First полягає в тому, що початково не пишеться код, а створюється візуальне представлення структури бази даних, а вже на основі нього генеруються моделі та таблиці в БД. Такий підхід є зручним при необхідності створити складну структуру, яку важко буде передбачити заздалегідь та описати кодом чи скриптами створення таблиць.

Для всіх підходів зберігається можливість внесення змін та оновлення. Наприклад, при зміні класу моделі в підході Code-First генеруються спеціальні міграції, які призначені для модифікації бази даних та перенесення/заповнення/видалення даних [6].

1.7 OAuth

Кожна окрема інформаційна система передбачає наявність автентифікації та авторизації. При цьому є суттєва різниця між даними поняттями.

Автентифікація – процес входу в систему. Передбачає використання персонального логіну та паролю, за необхідності може бути ускладнена фізичним носієм. Також останнім часом набирає популярності двохфакторна автентифікація – така, що передбачає підтвердження на двох рівнях, до прикладу, паролю та згенерованого коду за допомогою SMS чи мобільного додатку. Такий підхід збільшує захищеність персональних даних та системи в цілому і ускладнює несанкціонований доступ до даних.

Авторизація – схожий процес, проте він полягає не в підтвердженні користувача, а в підтвердженні доступу даного користувача до певного ресурсу. До прикладу, система може передбачати наявність кількох ролей, як от адміністратора, бухгалтера, менеджера, звичайного користувача системи. Роль авторизації полягає в підтвердженні того, що даний користувач має доступ до ресурсу перед тим, як запит буде оброблено.

Проте при інтеграції різних інформаційних систем виникає запитання єдиного стандарту авторизації, що забезпечить доступ з одного ресурсу до даних, розміщених на іншому (до прикладу, вхід з використанням акаунту Google чи Facebook).

Популярним стандартом відкритої авторизації є OAuth. Його суть полягає в тому, що сервіс, котрий надає доступ з використанням OAuth, позначає певні ресурси відкритими для доступу з третіх ресурсів. Такий підхід дозволяє авторизуватись на сервісі постачальника даних і отримати дані з нього, не використовуючи напряму секретних даних, як-от паролів чи електронних адрес.

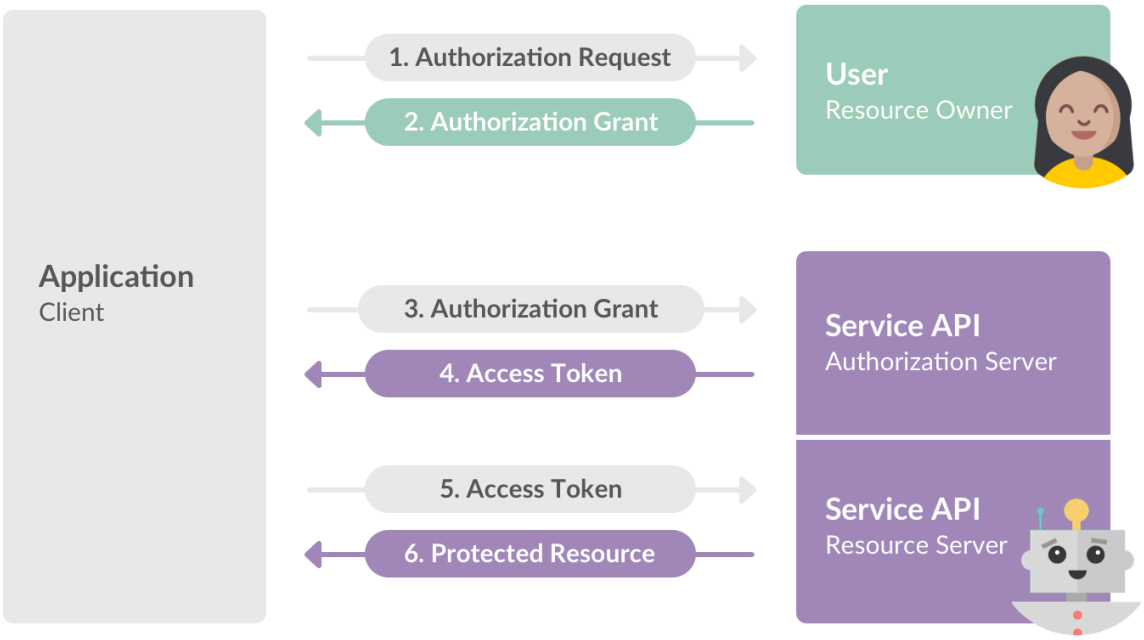

Рис. 1.6 – Стандартна схема використання OAuth

Процес роботи з OAuth полягає в наступному: для початку сервіс, що потребує доступу до даних постачальника, перенаправляє користувача на сторінку для підтвердження доступу на сервісі постачальника. При успішній автентифікації на сервісі постачальника користувач перенаправляється назад на сторінку стороннього сервісу, який отримує access token. З використанням даного токена сторонній сервіс може звертатися до постачальника і отримувати звідти дані, до яких підтверджений доступ (рис. 1.6). При чому час дії токена є обмеженим, тому разом з токеном передається також refresh token – ключ, який можна використати

для отримання наступного access токена. Такий підхід дозволяє мати постійний доступ до сервісу постачальника без необхідності повторної авторизації, а також ускладнити перехоплення і несанкціоноване використання попередньо згенерованого токена [7].

2 ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ПРОГРАМНОГО ПРОДУКТУ, МЕТОДОЛОГІЯ

2.1 Вибір методології розробки

В якості методології розробки ПЗ було обрано Скрам.

Scrum - це гнучкий процес, який найчастіше використовується для розробки продуктів, особливо для розробки програмного забезпечення. Scrum - це система управління проектами, яка застосовується до будь-якого проекту з агресивними термінами, складними вимогами та ступенем унікальності. У Scrum проекти рухаються вперед за допомогою низки ітерацій, які називаються спринтами. Кожен спринт зазвичай триває два-чотири тижні.

Команда Scrum: Типова команда Scrum налічує від п’яти до дев’яти людей, але проекти Scrum можуть легко масштабуватися до сотень. Однак Scrum може бути легко використаний командами від однієї особи, і часто він є. Ця команда не включає жодної з традиційних ролей програмної інженерії, таких як програміст, дизайнер, тестер або архітектор. Кожен, хто бере участь у проекті, працює разом, щоб виконати ту роботу, яку вони колективно взяли на себе в рамках спринту. Команди Scrum розвивають глибоку форму товариськості та відчуття, що "ми всі в цьому разом".

Власник продукту: Власник продукту є ключовою зацікавленою стороною проекту та представляє користувачів, клієнтів та інших учасників процесу. Власником товару часто є хтось із керівництва продуктом або маркетингу, ключова зацікавлена сторона або ключовий користувач.

Scrum Master: Scrum Master відповідає за те, щоб команда була максимально продуктивною. Майстер Scrum робить це, допомагаючи команді використовувати процес Scrum, усуваючи перешкоди для прогресу, захищаючи команду ззовні тощо.

Відставання товару: відставання продукту - це список пріоритетних функцій, що містить усі бажані функції або зміни продукту. Примітка. Термін "відставання"

може заплутати, оскільки він використовується для двох різних речей. Для уточнення, відставання товару - це перелік бажаних функцій продукту. Відставання спринту - це перелік завдань, які потрібно виконати в спринті.

Зустріч із планування спринту: на початку кожного спринту проводиться зустріч із планування спринту, під час якої власник продукту представляє команді найкращі елементи у відставанні товару. Команда Scrum обирає роботу, яку зможе виконати під час наступного спринту. Потім ця робота переноситься з відставання товару до відставання спринту, що є переліком завдань, необхідних для заповнення елементів відставання товару, які команда зобов'язала виконати у спринті.

Щоденна сутичка: кожного дня під час спринту проводиться коротка зустріч, яка називається щоденною сутичкою. Ця зустріч допомагає встановити контекст для кожного дня роботи та допомагає команді залишатися на шляху. Усі члени команди повинні відвідувати щоденні сутички.

Зустріч з огляду спринту: У кінці кожного спринту команда демонструє завершену функціональність на оглядовому засіданні спринту, під час якого команда показує, чого досягла під час спринту. Як правило, це приймає форму демонстрації нових можливостей, але в неформальній формі; наприклад, слайди PowerPoint заборонені. Зустріч не повинна стати самою собою завданням, ані відволіканням від процесу.

Ретроспектива спринту: Також в кінці кожного спринту команда проводить ретроспективу спринту, це зустріч, під час якої команда (включаючи свого ScrumMaster та власника продукту) розмірковує про те, наскільки добре працює Scrum для них та які зміни вони можуть побажати зробити так, щоб це працювало ще краще.

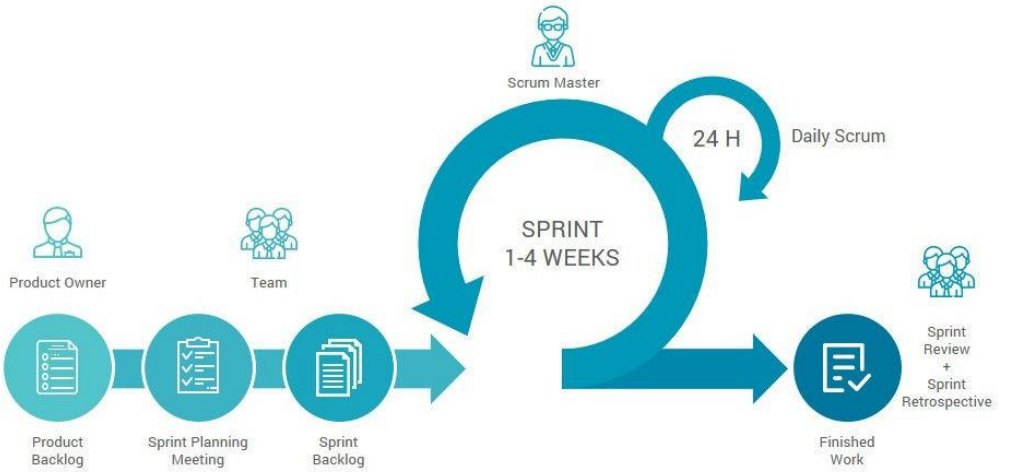

Рис. 2.1 – Схематичне зображення процесу розробки за методологією Scrum

Схематично процес зображено на рисунку 2.1. Ліворуч ми бачимо відставання товару, яке було визначене пріоритетом власника продукту та містить усе бажане в продукті, що було відомо на той час. Спринти на два-чотири тижні показані більшим зеленим колом.

На початку кожного спринту команда вибирає певний обсяг роботи із відставання продукту і зобов’язується завершити цю роботу під час спринту. Частина з’ясування того, на скільки вони можуть взяти на себе зобов’язання, - це створення відставання у спринті, тобто перелік завдань (та оцінка того, скільки часу триватиме кожне з них), необхідних для доставки вибраного набору елементів відставання товару, які потрібно виконати у спринті.

В кінці кожного спринту команда створює потенційно можливий приріст продукту, тобто робоче високоякісне програмне забезпечення. Щодня під час спринту члени команди збираються, щоб обговорити свій прогрес та будь-які перешкоди для завершення роботи для цього спринту. Це відомо як щоденна сутичка, і показано як менший зелений круг вгорі [8].

2.2 Вибір середовища розробки (Visual Studio 2019)

Visual Studio - це інтегроване середовище розробки (IDE), розроблене корпорацією Майкрософт для розробки графічного інтерфейсу користувача (графічного інтерфейсу користувача), консолі, веб-додатків, веб-програм, мобільних додатків, хмарних та веб-служб тощо. За допомогою цієї IDE ви можете створити керований код, а також власний код. Він використовує різні платформи програмного забезпечення для розробки програмного забезпечення Microsoft, такі як Windows store, Microsoft Silverlight та Windows API тощо. Це не специфічна для мови IDE, оскільки ви можете використовувати її для написання коду на C #, C++, VB (Visual Basic), Python, JavaScript та багато інших мов. Вона забезпечує підтримку 36 різних мов програмування. Visual Studio доступна як для Windows, так і для macOS (рис. 2.2).

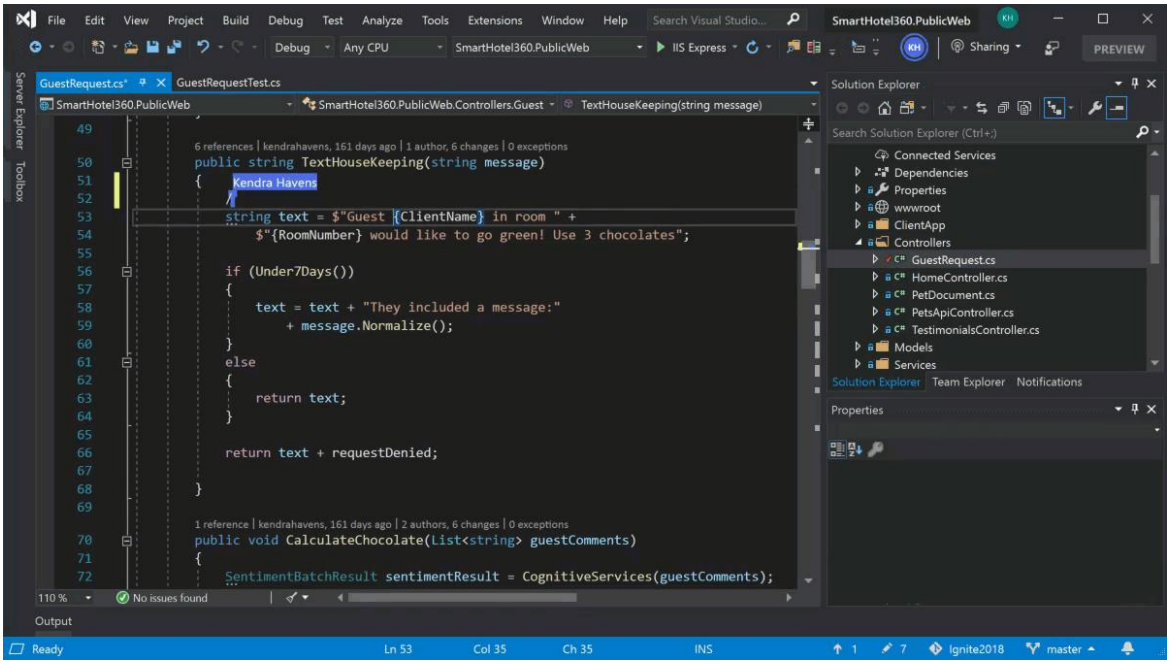

Рисунок 2.2 – Середовище розробки Visual Studio

Еволюція Visual Studio: Перша версія VS (Visual Studio) була випущена в 1997 році під назвою Visual Studio 97, що має номер версії 5.0. Остання версія Visual

Studio - це 15.0, яка вийшла 7 березня 2017 року. Вона також називається Visual Studio 2017. Підтримувані версії .Net Framework в останній Visual Studio складають від 3,5 до 4,7. Java підтримувалася у старих версіях Visual Studio, але в останній версії не підтримується мова Java.

Існує 3 версії Microsoft Visual Studio:

− Community: це безкоштовна версія, яка анонсована в 2014 році. Усі інші видання платні. Він містить функції, схожі на професійну версію. За допомогою цього видання будь-який окремий розробник може розробляти власні безкоштовні або платні додатки, такі як програми .Net, веб-програми та багато іншого. У корпоративній організації це видання має деякі обмеження. Наприклад, якщо у вашій організації більше 250 ПК і річний дохід перевищує 1 мільйон доларів (доларів США), вам не дозволяється використовувати це видання. У непідприємницькій організації це видання може використовувати до п’яти користувачів. Його основна мета - забезпечити підтримку екосистеми (доступ до тисяч розширень) та мов (Ви можете кодувати на C #, VB, F #, C ++, HTML, JavaScript, Python тощо).

− Professional: це комерційне видання Visual Studio. Воно поставляється у Visual Studio 2010 та пізніших версіях, забезпечує підтримку редагування XML та XSLT і включає такий інструмент, як провідник серверів та інтеграцію з Microsoft SQL Server. Microsoft надає безкоштовну пробну версію цього видання, і після пробного періоду користувач повинен заплатити, щоб продовжувати користуватися ним. Його основна мета - забезпечити гнучкість (професійні інструменти розробника для створення будь-якого типу додатків), продуктивність (потужні функції, такі як CodeLens, покращують продуктивність вашої команди), співпрацю (гнучкі інструменти планування проектів, діаграми тощо) та переваги передплатників, такі як програмне забезпечення Microsoft плюс Azure, Pluralsight тощо.

− Enterprise: це комплексне рішення для команд будь-якого розміру з вимогливими потребами у якості та масштабі. Microsoft надає 90-денну безкоштовну пробну версію цього видання, і після пробного періоду користувач

повинен заплатити, щоб продовжувати користуватися ним. Головною перевагою цього видання є те, що воно дуже масштабоване та забезпечує високоякісне програмне забезпечення.

Для початку роботи потрібно завантажити та встановити Visual Studio. Для цього ви можете звернутися до Завантаження та встановлення Visual Studio 2017. Не забудьте вибрати основне робоче навантаження .NET під час встановлення VS 2017. Якщо ви забудете, вам доведеться змінити інсталяцію.

Коли ви відкриєте Visual Studio і почнете писати свою першу програму, ви побачите ряд вікон інструментів наступним чином:

− Редактор коду: де користувач буде писати код.

− Вікно виводу: тут Visual Studio відображає результати, попередження компілятора, повідомлення про помилки та інформацію про налагодження.

− Провідник рішень: вікно перегляду файлів, над якими користувач працює в даний момент.

− Властивості: додаткова інформація та контекст про вибрані частини поточного проекту.

Користувач також може додавати вікна відповідно до вимог, вибираючи їх у меню Перегляд. У Visual Studio вікна інструментів можна налаштувати, оскільки користувач може додати більше вікон, видалити наявне відкрите або переміщати вікна, якнайкраще.

Різні меню у Visual Studio: користувач може знайти багато меню на верхньому екрані Visual Studio, як показано нижче

− Команди створення, відкриття та збереження проектів містяться в меню Файл.

− Команди пошуку, модифікації, рефакторингу містяться в меню Редагувати.

− Меню Перегляд використовується для відкриття додаткових вікон інструментів у Visual Studio.

− Меню проекту використовується для додавання деяких файлів та залежностей у проект.

Щоб змінити налаштування, додайте функціональність Visual Studio за допомогою розширень та отримайте доступ до різних інструментів Visual Studio за допомогою меню Інструменти [9].

2.3 Підготовка для підключення застосунку зі сторони Приват24

Основа функціонування системи – коректна синхронізація даних про транзакції із системою Приват24 API.

Приват24 надає розробникам повнофункціональний інструментарій для отримання даних про платежі, баланс на рахунку та різноманітні дані що стосуються курсів валют, розміщення банкоматів, відділень і т.ін. Також система надає доступ до керування платежами.

В даній системі нас цікавить саме точка доступу для отримання виписок по конкретній картці (рис. 2.3).

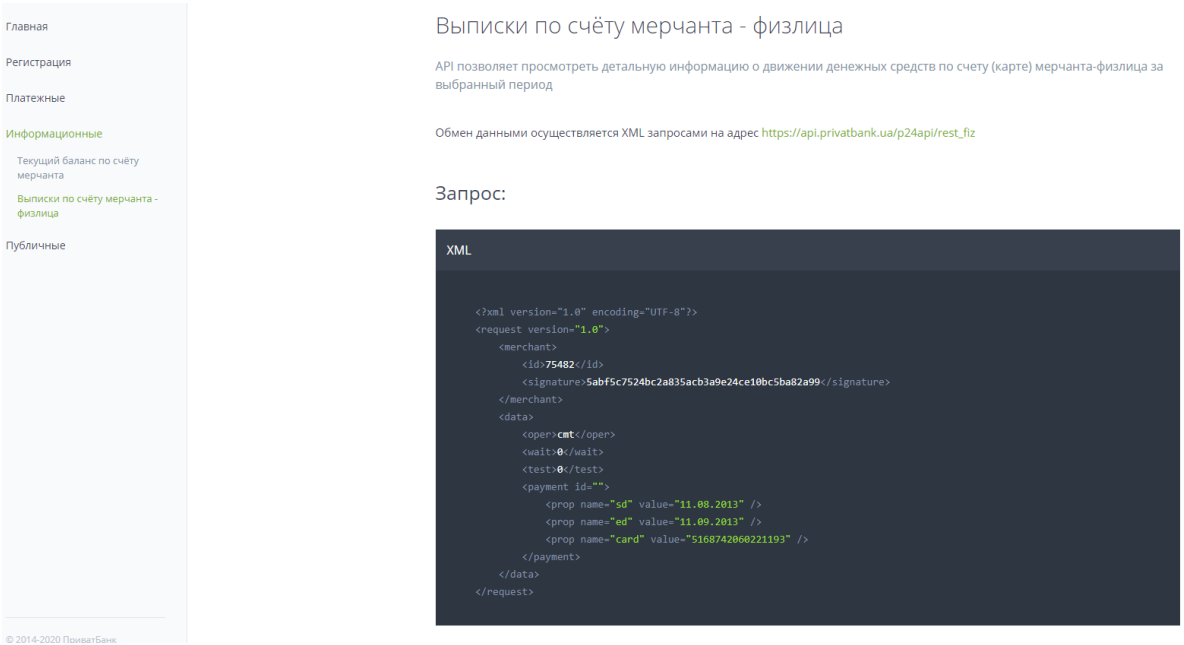

Рис. 2.3 – Запит для отримання виписки в системі Приват24

Одні з параметрів запиту це ідентифікатор мерчанта та цифровий підпис запиту.

Мерчант це аплікація, що реєструється в системі Privat24. Вона прив’язується до конкретної картки (рахунку) і їй надаються різні дозволи для блокування/розблокування доступу до даних по цій картці (рис. 2.4).

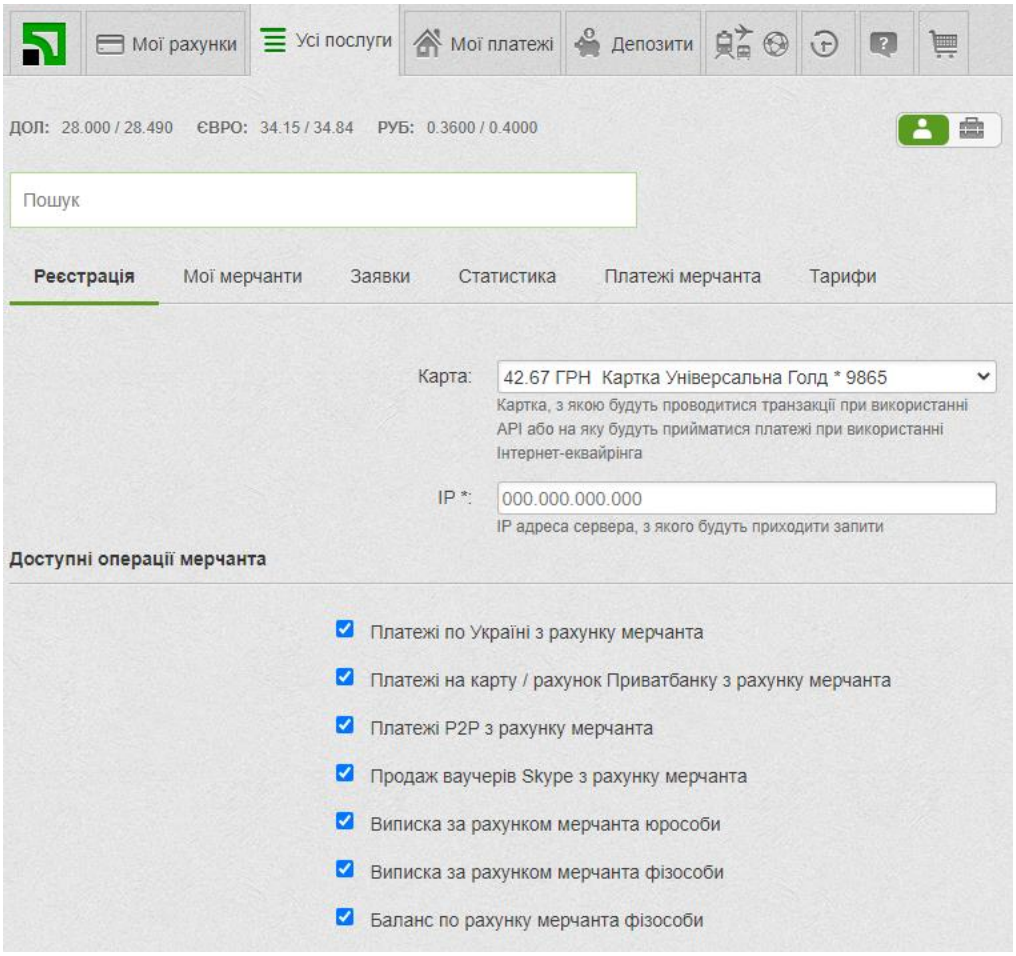

Рис. 2.4 – Налаштування мерчанта на стороні Приват24

Також в налаштуваннях необхідно вказати IP-адресу з якої будуть відправлятись запити на Приват24API.

У разі успішної реєстрації ми отримаємо пароль для цифрового підпису усіх майбутніх запитів до цієї аплікації [10].

2.4 Побудова моделей предметної області

Дослідивши предметну область та вивчивши дані, які нам необхідні із стороннього постачальника (Приват24 API), робимо висновок, що основною моделлю даного домену є транзакція (платіж) у системі Приват24.

Ці дані ми отримуємо із існуючої системи, тому нам необхідно зробити більш детальний її аналіз.

Основні поля:

− “card” – номер картки з якої/на яку була здійснена транзакція,

− “appcode” – унікальний номер транзакції в межах цього рахунку,

− “trandate” – дата здійснення транзакції в текстовому представленні,

− “trantime” – час здійснення транзакції в текстовому представленні,

− “amount” – сума транзакції із вказанням валюти,

− “cardamount” – зміна балансу рахунку в ході виконання транзакції (якщо дебетна операція, то значення буде додатнім, якщо кредитна – від’ємним) (із вказанням валюти),

− “rest” – залишок на рахунку після здійснення транзакції (із вказанням валюти),

− “terminal” – платіжний шлюз, який опрацьовував транзакції,

− “description” – текстовий опис (коментар) до платежу.

Таким чином, основну модель даних для конкретного домену можна представити в такому графічному вигляді (рис. 2.5).

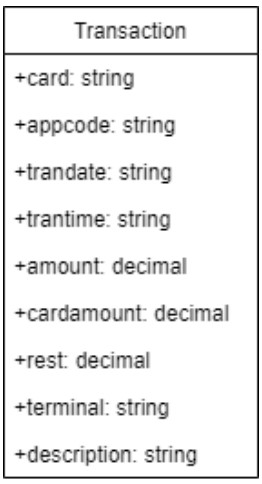

Рис. 2.5 – Діаграма моделі даних

Також, для коректного функціонування модуля синхронізації необхідно ввести в поточний домен нову службову модель даних – Інформація про час останньої синхронізації для конкретного рахунку (картки). Під синхронізацією тут розуміємо періодичне отримання даних з системи Приват24

Опишемо дану модель:

− “CardNumber” – картка (рахунок), для якого здійснювалась синхронізація

− “LastCheckDateTime” – дата і час останньої синхронізації даних про транзакції для конкретної картки (рахунку).

2.5 Розробка варіантів використання

Програмний продукт має виконувати наступні функції: отримання даних про транзакції з Приват24 API; запис інформації про платежі в базу; періодична або мануальна синхронізація даних про транзакції; отримання даних через зовнішню точку доступу системи (API); конфігурування системи для налаштування під конкретні рахунки, мерчанти, базу даних та ін.; розсилка інформації по настроюваних каналах зв’язку з користувачем.

Серед основних авторів предметної області можна виділити наступні: користувач системи, адміністратор системи, система, система Приват24 та база даних.

Користувач системи може отримувати дані про транзакції із опцією синхронізації з БД та запускати мануальну синхронізації даних про транзакції з Приват24 в базу даних (рис. 2.7).

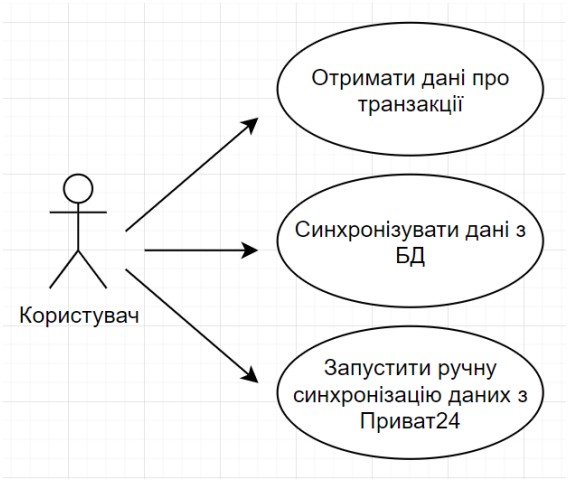

Рис. 2.6 – Варіанти використання користувача системи

Варіанти використання адміністратора системи доповнюють варіанти користувача із додаванням можливості конфігурування системи (рис.2.7).

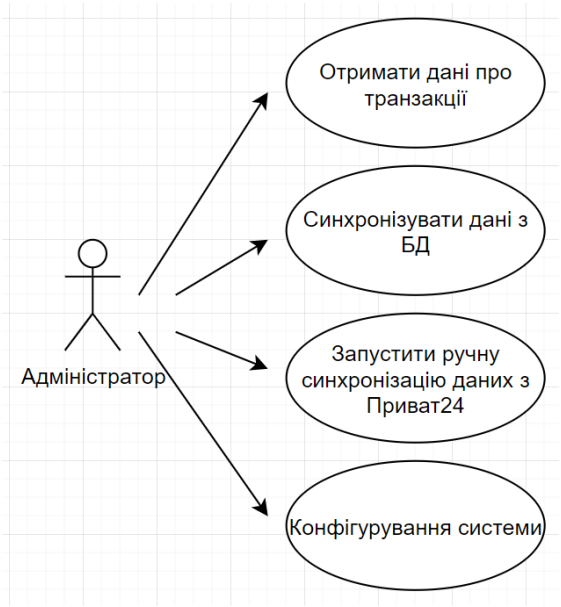

Рис. 2.7 – Варіанти використання адміністратора системи

Система здійснює синхронізації, отримує дані з Приват24, містить в собі бізнес-логіку розсилки інформації, відповідає за шифрування і підпис запитів до Приват24 а також надає користувачам доступ до інформації в БД або напряму з Приват24 API.

База даних відповідає за коректне збереження інформації про транзакції та інформації про останню синхронізацію для конкретного рахунку (картки).

2.6 Проектування архітектури системи

Дана система розроблюється як крос-платформний веб-сервер.

Оскільки система має працювати з базою даних, має мати бізнес-логіку синхронізації з БД та розсилки інформації, а також має надавати доступ до даних ззовні, очевидним вибором була серверна, тришарова архітектура.

На нижньому рівні розташувався модуль доступу до даних (бази даних) - DataAccessLayer. Рівень передбачає надання повнофункціонального, зручного і асинхронного інтерфейсу для об’єктної роботи з базою даних.

На середньому рівні розташований модуль бізнес-логіки - BusinessLogicLayer, що включає в себе основну частину сценаріїв роботи програми. Тут знаходиться бізнес-логіка обробки даних з Приват24 API, а також створення і цифрове підписування відповідних запитів.

Також тут знаходиться логіка обробки запитів клієнта ззовні для отримання транзакції.

І окрема варто зазначити опис процедури синхронізації з БД, що теж розташований на цьому рівні. Для забезпечення обміну даними між різними модулями системи, передбачена конвертація між різними моделями, що представляють суміжні сутності в різних контекстах.

На верхньому рівні розташований модуль для забезпечення зв’язку користувача із системою – WebAPI. Це дозволяє як використовувати відкриту точку доступу для отримання безпосередніх даних, так і забезпечує можливість повторного використання системи, як джерела даних для іншого програмного забезпечення [11].

Окремо знаходиться модуль зв’язку із Приват24 – Privat24Connector. Він використовується для отримання безпосередніх даних із Приват24 API та конвертації транзакції, що надходять у форматі XML у традиційний для .NET Core, об’єктний формат.

Також передбачена можливість додавання сторонніх сервісів, для розсилки інформації іншими каналами зв’язку (електронна пошта, сервіси обміну повідомленнями: Telegram, Viber або ж СМС-повідомлення і т. ін.).

2.7 База даних

2.7.1 Нормалізація

Нормалізація бази даних - це техніка проектування схеми бази даних, за допомогою якої існуюча схема модифікується для мінімізації надмірності та залежності даних.

Нормалізація розділяє велику таблицю на менші таблиці та визначає взаємозв'язки між ними, щоб підвищити чіткість в організації даних.

Деякі факти про нормалізацію бази даних:

Слова нормалізації та нормальної форми стосуються структури бази даних.

Нормалізацію розробив дослідник IBM Е.Ф. Кодд У 1970-х роках.

Нормалізація підвищує чіткість організації даних у базах даних.

Нормалізація бази даних досягається шляхом дотримання набору правил під назвою «форми» при створенні бази даних:

− Перша нормальна форма (1NF) – кожен стовпець унікальний у 1NF.

− Друга нормальна форма (2NF) – сутність повинна розглядатися вже в 1NF, і всі атрибути всередині сутності повинні залежати виключно від унікального ідентифікатора сутності.

− Третя нормальна форма (3NF) – сутність повинна розглядатися вже у 2NF, і жоден запис у стовпці не повинен залежати від будь-якого іншого запису (значення), крім ключа для таблиці. Якщо така сутність існує, перемістіть її за межі нової таблиці. 3NF досягнуто, розглядається як нормалізована база даних.

− Звичайна форма Бойса-Кодда (BCNF) – 3NF і всі таблиці в базі даних повинні бути лише одним первинним ключем.

− Четверта нормальна форма (4NF) – таблиці не можуть мати багатозначних залежностей від первинного ключа.

− П'ята нормальна форма (5NF) – складений ключ не повинен мати жодних циклічних залежностей [12].

2.7.2 Розробка бази даних

При виборі СУБД для бази даних системи було обрано MSSQL Server. Дане СУБД є розробкою компанії Microsoft, поширюється на безкоштовній основі і крім самого сервера бази даних також надає зручне середовище управління базами даних SQL Server Management Studio (рис. 2.8).

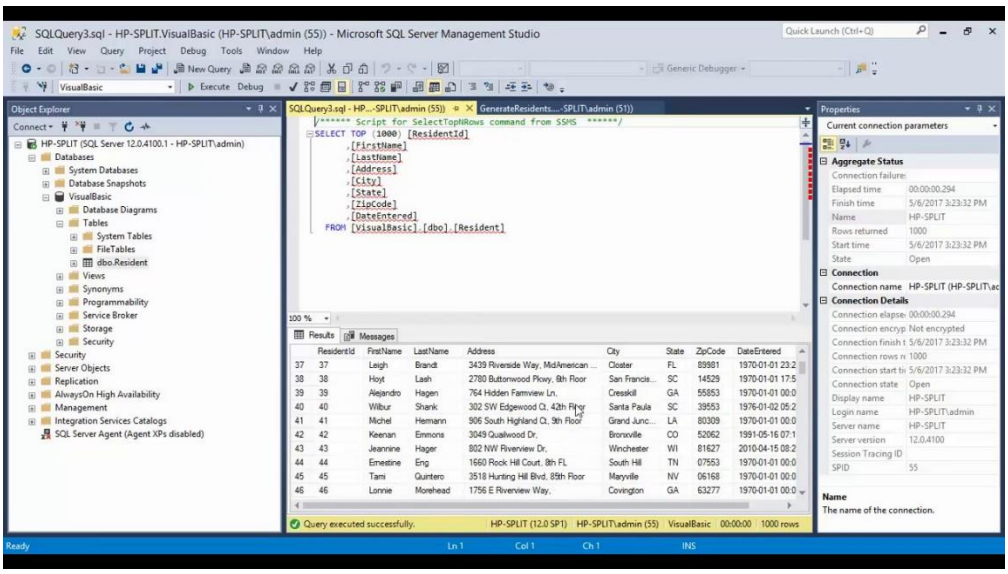

Рис. 2.8 – SQL Server Management Studio

В даному СУБД можна зручно переглядати існуючі підключення, додавати нові, запускати автоматичні запити, як от 1000 перших рядків. Також зручною функцією є можливість на основі готової бази даних згенерувати скрипти. Дана функція є особливо актуальною, коли потрібно перенести базу даних на інший сервер або створити резервну копію частини інформації. Також за допомогою генерації можна згенерувати скрипт створення бази даних, що дозволить спочатку розробити БД, а потім швидко її розгорнути на іншому середовищі [13].

Оскільки для написання системи із використанням EntityFramework Core було обрано підхід DataBase First, то необхідно написати скрипт, для створення бази даних, створення таблиць під існуючі моделі даних в поточному домені, а також налаштувати індекси для прискорення пошуку і вибірки даних по ключових полях (рис. 2.9).

```
CREATE DATABASE PaymentNotifications

USE PaymentNotifications

CREATE TABLE [LastTransactionsCheckInfos] (
    [Id] INT IDENTITY (1, 1) NOT NULL,
    [CardNumber] NVARCHAR (16) NOT NULL,
    [LastCheckDateTime] SMALLDATETIME NULL,
    CONSTRAINT [PK_LastTransactionsCheckInfos] PRIMARY KEY CLUSTERED ([Id] ASC),
    CONSTRAINT [IX_LastTransactionsCheckInfos_CardNumber] UNIQUE ([CardNumber])
);

CREATE TABLE [Transactions] (
    [Id] INT IDENTITY (1, 1) NOT NULL,
    [CardNumber] NVARCHAR (16) NOT NULL,
    [AppCode] NVARCHAR (10) NOT NULL,
    [TransactionDateTime] DATETIME2 NULL,
    [TransactionAmount] DECIMAL (38, 2) NOT NULL,
    [CardAmount] DECIMAL (38, 2) NOT NULL,
    [Currency] NVARCHAR (5) NOT NULL,
    [Rest] DECIMAL (38, 2) NOT NULL,
    [Terminal] NVARCHAR (100) NULL,
    [Description] NVARCHAR (500) NULL,
    CONSTRAINT [PK_Transactions] PRIMARY KEY CLUSTERED ([Id] ASC),
    INDEX [IX_Transactions_CardNumber] NONCLUSTERED ([CardNumber]),
    INDEX [IX_Transactions_AppCode] NONCLUSTERED ([AppCode]),
    INDEX [IX_Transactions_TransactionDateTime] NONCLUSTERED ([TransactionDateTime])
);
```

Рис. 2.9 – Скрипт створення бази даних

В даному скрипті спочатку створюється база даних з іменем PaymentNotifications.

Далі відбувається перехід в контекст новоствореної бази .

Після цього ми створюємо таблиці, які імітують собою сутності домену системи, із усіма необхідними полями. Для деяких полів встановлено прапорець NULL, що означає їх необов’язковість (деякі дані можуть не прийти з Приват24 API, але ми маємо забезпечити цілісність бази даних).

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Розробка модуля для зв'язку з Приват24 API

Даний модуль призначений для безпосередньої комунікації з Приват24 API. Тому для початку було створено моделі запиту для серіалізації в XML (рис. 3.1).

```
[XmlRoot(elementName: "request")]
1 reference
public class Privat24GetTransactionsRequest
{
    [XmlAttribute("version")]
    1 reference
    public string Version { get; set; }

    [XmlElement("merchant")]
    2 references
    public Privat24Merchant Merchant { get; set; }

    [XmlElement("data")]
    1 reference
    public Privat24RequestData Data { get; set; }
}
```

Рис. 3.1 – Модель отримання транзакцій

Цей клас є базовим елементом запиту для отримання транзакцій. В ньому міститься інформація про мерчант разом і цифровим підписом. Також далі по ієрархії іде тіло запиту де містяться фільтри транзакцій (рис. 3.2).

```
1 reference
public class Privat24RequestDataPayment
{
    [XmlAttribute("id")]
    1 reference
    public string Id { get; set; }

    [XmlElement("prop")]
    1 reference
    public List<Privat24RequestDataPaymentProperty> PaymentProperties { get; set; }
}

2 references
public class Privat24RequestDataPaymentProperty
{
    [XmlAttribute("name")]
    3 references
    public string Name { get; set; }

    [XmlAttribute("value")]
    3 references
    public string Value { get; set; }
}
```

Рис. 3.2 – Детальний опис тіла запиту

Серед фільтрів представлені номер картки, початкова та кінцева дата транзакцій. Слід зазначити, що Приват24 API не враховує час в якості фільтра, тому для забезпечення коректної фільтрації, її остаточний етап відбувається вже в самій системі.

Відповідь сервера має практично однакову структуру, за винятком того, що тіло її складається не з фільтрів а з об'єктів, що описують саму транзакції. Там присутні всі поля, для коректної конвертації в об'єкт платежу в .Net (Рис. 3.3)

```
public class Privat24ResponseDataStatement
{
    [XmlAttribute("card")]
    3 references
    public string Card { get; set; }

    [XmlAttribute("appcode")]
    3 references
    public string AppCode { get; set; }

    [XmlAttribute("trandate")]
    2 references
    public string TransactionDate { get; set; }

    [XmlAttribute("trantime")]
    2 references
    public string TransactionTime { get; set; }

    [XmlAttribute("amount")]
    3 references
    public string Amount { get; set; }

    [XmlAttribute("cardamount")]
    3 references
    public string CardAmount { get; set; }

    [XmlAttribute("rest")]
    3 references
    public string Rest { get; set; }

    [XmlAttribute("terminal")]
    2 references
    public string Terminal { get; set; }

    [XmlAttribute("description")]
    2 references
    public string Description { get; set; }

    [XmlIgnore]
    5 references
    public DateTime? TransactionDateTime { get; set; }
}
```

Рис. 3.3 – Модель транзакції у відповіді Приват24

В класі транзакції (Statement в термінології Приват24 API), є додаткове поле TransactionDateTime, яке не має відповідника в XML файлі, але встановлюється вже на стороні системи, що дозволяє робити внутрішню фільтрацію даних більш оптимізовано.

Наступним етапом було створення класу Privat24HttpClient (Приват24 працює на базі Http протоколу), для інкапсуляції роботи з Http запитами і відповідями. Основний метод – HttpGet (рис. 3.4).

```
public async Task<T> HttpGet<T>(string url, object content = null) where T : class
{
    var client = new HttpClient();

    var httpRequest = new HttpRequestMessage
    {
        Method = HttpMethod.Get,
        RequestUri = new Uri(url)
    };

    if (content != null)
    {
        var contentXml :string = Privat24XmlSerializer.Serialize(content);
        httpRequest.Content = new StringContent(contentXml, Encoding.UTF8, ApplicationXmlContentType);
    }

    var response = await client.SendAsync(httpRequest);

    if (response.StatusCode != HttpStatusCode.OK)
    {
        return null;
    }

    return await GetResult<T>(response);
}
```

Рис. 3.4 – Запит HttpGet

Цей алгоритм створює HTTP-запит, додає в нього тіло запиту у вигляді XML файлу, обробляє відповідь сервера, та десеріалізує контент відповіді в очікуваний тип.

Наступним етап – написання класу, для підписування запиту алгоритмами шифрування, використовуючи індивідуальний пароль до мерчанта (рис. 3.5).

```
public static string GetSignature(string dataForSign, string password)
{
    var sha1 = SHA1.Create();
    var md5 = MD5.Create();

    var md5Result :byte[] = md5.ComputeHash( buffer: Encoding.UTF8.GetBytes( s: dataForSign + password));
    var md5ResultString = GetByteArrayAsHexadecimalString(md5Result);

    var sha1Result :byte[] = sha1.ComputeHash( buffer: Encoding.UTF8.GetBytes(md5ResultString));
    return GetByteArrayAsHexadecimalString(sha1Result);
}
```

Рис. 3.5 – Алгоритми підпису запиту

Тут тіло запиту разом із паролем хешується спочатку алгоритмом MDA а потім результат ще раз алгоритмом SHA1. На виході ми маємо стрічку Signature, яка необхідна для підписання запиту в Приват24 API.

Останнім етапом було написання ядра модуля Privat24Connector – класу Privat24Connector, який інкапсулює в собі всі компоненти модуля, а також займається кінцевою внутрішньою фільтрацією транзакцій (з урахуванням часу меж фільтрації):

```
public async Task<ICollection<Privat24ResponseDataStatement>>
GetTransactionStatementsAsync(DateTime startDate, DateTime
endDate)
{
    var request = CreateTransactionsRequest(startDate,
endDate);
    var xmlRequest = Privat24XmlSerializer.Serialize(request);
    var requestData = GetRequestDataForSign(xmlRequest,
GetTransactionsSignStart, GetTransactionsSignEnd);
    var signature =
Privat24RequestSigner.GetSignature(requestData,
MerchantPassword);
    request.Merchant.Signature = signature;
    var transactionsResponse = await
_privat24HttpClient.HttpGet<Privat24GetTransactionsResponse>(Ur
ls.GetTransactionsUrl, request);
    var transactions =
transactionsResponse?.Data.DataInformation.Statements.Statement
s;

    return FilterTransactions(transactions, startDate,
endDate);
}
```

Даний метод звертається до всіх допоміжних сервісів модуля а також фільтрує транзакції (тіло алгоритму фільтрації наведено нижче).

```
private static List<Privat24ResponseDataStatement>
FilterTransactions(ICollection<Privat24ResponseDataStatement>
transactions, DateTime startDate, DateTime endDate)
{
    {
    foreach (var transaction in transactions)
        var transactionDateTimeString =
transaction.TransactionDate + " " +
transaction.TransactionTime;
        if (DateTime.TryParseExact(
            transactionDateTimeString,
            Privat24DateTimeFormat,
            null,
            DateTimeStyles.None,
```

```
            out var transactionDateTime))
                transaction.TransactionDateTime =
transactionDateTime;
        }

    return transactions
        .Where(transaction =>
           transaction.TransactionDateTime.HasValue &&
           transaction.TransactionDateTime >= startDate &&
           transaction.TransactionDateTime <= endDate)
            .ToList();
}
```

Тут ми встановлюємо значення властивості TransactionDateTime (про яку було згадано вище) та здійснюємо фільтрацію транзакцій по часу враховуючи не тільки дату транзакцій а і їхній час виконання.

3.2 Розробка модуля для роботи з базою даних

Даний модуль створено для забезпечення об’єктного способку взаємодії з базою даних, що по суті є обгорткою над ORM EntityFramework Core.

Для початку створюються моделі даних, що відповідають сутностям, попередньо спроектованим для бази даних (рис. 3.6)

```
public class Transaction : BaseEntity
{
    #region Properties

    7 references
    public string CardNumber { get; set; }

    7 references
    public string AppCode { get; set; }

    6 references
    public DateTime? TransactionDateTime { get; set; }

    5 references
    public decimal TransactionAmount { get; set; }

    3 references
    public decimal CardAmount { get; set; }

    4 references
    public string Currency { get; set; }

    3 references
    public decimal Rest { get; set; }

    5 references
    public string Terminal { get; set; }

    4 references
    public string Description { get; set; }

    #endregion
}
```

Рис. 3.6 – Модель транзакції

Далі, за допомогою FluentAPI ми налаштовуємо властивості цієї моделі для повного співпадіння з базою даних. Тут можна побачити, що FluentAPI володіє великим спектром можливостей, аналогів яких часто немає в декларативному підході (використання атрибутів) (рис. 3.7).

```
public void Configure(EntityTypeBuilder<LastTransactionsCheckInfo> builder)
{
    builder.HasKey(e :LastTransactionsCheckInfo => e.Id);

    builder.ToTable("LastTransactionsCheckInfos");

    builder.Property(e :LastTransactionsCheckInfo => e.CardNumber)
        .IsRequired()
        .HasMaxLength(16)
        .HasColumnName("CardNumber");

    builder.Property(e :LastTransactionsCheckInfo => e.LastCheckDateTime)
        .IsRequired(false)
        .HasColumnType("smalldatetime")
        .HasColumnName("LastCheckDateTime");

    builder.HasIndex(b :LastTransactionsCheckInfo => b.CardNumber);
}
```

Рис. 3.7 – Налаштування об’єктів за допомогою FluentAPI

Ми маємо можливість задавати типи даних в БД, їхні обмеження, констрейнти і т.д.

І одним з головних класів даного модуля є класс GenericRepository<>, що надає методи для об’єктної роботи з базою даних. При цьому ми можемо використовувати цей клас, для роботи з різними типами моделей, а, значить, різними таблицями в БД (рис. 3.8).

```
5 references
public virtual IQueryable<T> Get(Expression<Func<T, bool>> filter = null)...

1 reference
public virtual async Task<T> GetByIdAsync(int id)...

3 references
public async Task AddAsync(T entity)...

2 references
public async Task UpdateAsync(T entity)...

3 references
public async Task SaveAsync()...
```

Рис. 3.8 – Репозиторій для доступу до бази даних

Тут присутні так звані CRUD-операції (Create, Update, Delete) а також методи, що дозволяють отримати дані з бази по запиту написаному на LinqToEntities а також отримати інформації з БД по ідентифікатору (Id).

3.3 Розробка бізнес-логіки системи

Даний модуль призначений для регулювання всіх сценаріїв роботи системи, а також управління розсилкою інформації про факт оплати на сторонні сервіси.

Спочатку необхідно створити моделі даних, для спілкування між модулями системи (так звані DTO – data transfer object) (рис 3.9).

```
public class GetTransactionsRequestDTO
{
    #region Properties

    2 references
    public string CardNumber { get; set; }

    2 references
    public string AppCode { get; set; }

    2 references
    public DateTime? StartDate { get; set; }

    2 references
    public DateTime? EndDate { get; set; }

    2 references
    public decimal? TransactionAmountFrom { get; set; }

    2 references
    public decimal? TransactionAmountTo { get; set; }

    2 references
    public string Currency { get; set; }

    2 references
    public string Terminal { get; set; }

    3 references
    public string SearchValue { get; set; }

    #endregion
}
```

Рис. 3.9 – Модель запиту для отримання транзакцій

Даний клас призначений для передачі параметрів, для вибірки транзакцій з бази. В ньому присутні такі властивості, як TransactionAmountFrom та TransactionAmountTo для забезпечення можливості задання меж певної властивості, яка використовується в фільтрації.

Наступний клас називається TransactionDTO.

Він практично ідентичний типу даних Transaction але відображає деякі поля у більш зрозумілому форматі (рис. 3.10).

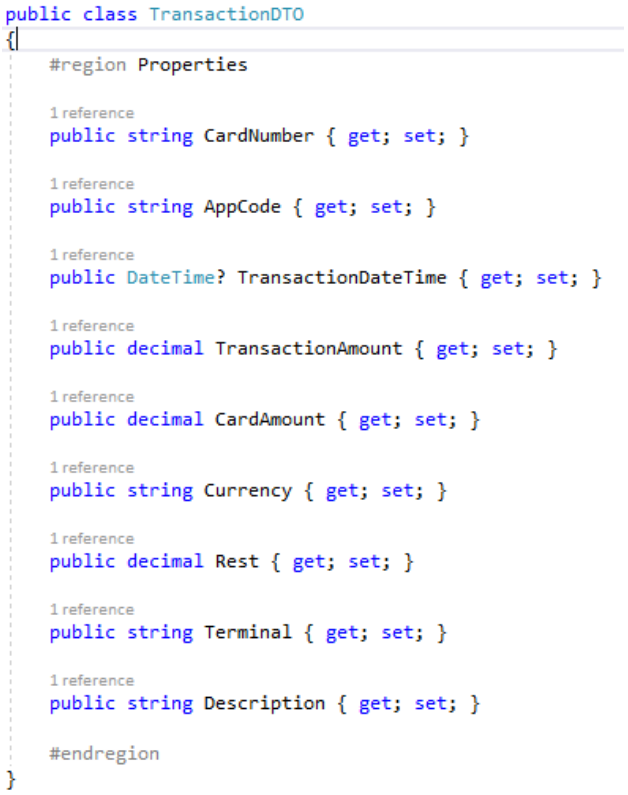

Рис. 3.10 – Модель транзакції

Ще одна причина створення таких типів є неприйнятність використання моделей даних, призначеної для одного контексту, в іншому.

Для забезпечення конвертації одних моделей в інші, створено так звані конвертери типів. Наведемо один із них:

```
internal static class TransactionConverter
{
    #region Public

    // ReSharper disable once IdentifierTypo
    public static ICollection<TransactionDTO>
ToDtos(ICollection<Transaction> transactions) =>
        transactions
            .Select(ToDto)
            .ToList();

    #endregion

    #region Private

    private static TransactionDTO ToDto(Transaction
transaction) =>
        new TransactionDTO
        {
            CardNumber = transaction.CardNumber,
```

```
            AppCode = transaction.AppCode,
            TransactionDateTime =
transaction.TransactionDateTime,
            TransactionAmount = transaction.TransactionAmount,
            CardAmount = transaction.CardAmount,
            Currency = transaction.Currency,
            Rest = transaction.Rest,
            Terminal = transaction.Terminal,
            Description = transaction.Description
        };

    #endregion
}
```

Тепер черга основною частиною бізнес-логіки. Для управління всіма процесами в системі створено тип TransactionsManager. Він має кілька основних методів.

```
public async Task<ICollection<TransactionDTO>>
GetDbTransactionsAsync(GetTransactionsRequestDTO request)
{
    var dbTransactions = await _transactionRepository
        .Get(SearchTransactions(request))
        .AsNoTracking()
        .ToListAsync()
        .ConfigureAwait(false);

    var transactions =
TransactionConverter.ToDtos(dbTransactions);

    return transactions;
}
```

Це метод призначений для отримання транзакцій, які збережені в базі даних системи. Для фільтрації використовується DTO, що містить всі необхідні критерії пошуку (описаний вище).

Наступний метод називається GetPrivat24TransactionsAsync:

```
public async Task<ICollection<Privat24TransactionDTO>>
GetPrivat24TransactionsAsync(DateTime? startDate, DateTime?
endDate, bool performSync)
{
    endDate ??= DateTime.Now;
    startDate ??= endDate.Value.AddMonths(-1);

    if (startDate.Value > endDate.Value)
        return null;

    var privat24Transactions = await _connector
        .GetTransactionStatementsAsync(startDate.Value,
endDate.Value)
        .ConfigureAwait(false);
```

```
    if (privat24Transactions is null)
        return null;

    if (performSync)
    {
        privat24Transactions = await
PerformSynchronizationAsync(privat24Transactions)
            .ConfigureAwait(false);

        await UpdateLastTransactionsCheckInfoAsync(CardNumber,
endDate.Value)
            .ConfigureAwait(false);
    }

    return
Privat24TransactionConverter.ToDtos(privat24Transactions);
}
```

Він призначений для отримання транзакцій напряму із Приват24 API. В якості параметрів вказуються часові рамки транзакцій. В разі необхідності можна задати параметр performSync, що забезпечує запис у базу системи нових транзакцій, отриманих в результаті виконання даного методу.

Останній метод призначений для автоматичної синхронізації транзакцій (без повернення їх користувачу). Якщо часові рамки не будуть задані, то використовується проміжок від останнього вдалого запуску синхронізації до поточного моменту часу.

Окремої уваги заслуговує метод, що забезпечує запис в базу системи унікальних транзакцій:

```
private async Task<ICollection<Privat24ResponseDataStatement>>
PerformSynchronizationAsync(ICollection<Privat24ResponseDataSta
tement> privat24Transactions)
{
    var transactions = Privat24TransactionConverter
        .ToEntities(privat24Transactions)
        .ToDictionary(CreateKey, transaction => transaction);

    var transactionIds = transactions
        .Select(x => x.Key)
        .ToList();

    var existingTransactionIds = await _transactionRepository
        .Get(transaction =>
            transactionIds.Contains(transaction.CardNumber +
transaction.AppCode))
        .AsNoTracking()
```

```
            .Select(transaction => CreateKey(transaction))
            .ToListAsync()
            .ConfigureAwait(false);

        var newTransactionIds = transactionIds
            .Except(existingTransactionIds)
            .ToList();

        var newTransactions = transactions
            .Where(transaction =>
    newTransactionIds.Contains(transaction.Key))
            .Select(transaction => transaction.Value)
            .ToList();

        foreach (var transaction in newTransactions)
            await _transactionRepository.AddAsync(transaction);

        await _transactionRepository.SaveAsync();

        return GetNewStatements(privat24Transactions,
    newTransactionIds);
    }
```

Оскільки поле appCode транзакції може повторюватись в різних картках (рахунках), то в якості унікального ідентифікатора обрано комбінацію полів cardNumber + appCode.

3.4 Написання відкритої точки доступу системи (API) для зв'язку ззовні

Модуль створено дня забезпечення отримання даних збережених в базі системи, отримання транзакцій напряму з Приват24 API, мануального запуску процесу синхронізації, також, можливості використання системи як постачальника даних для сторонній програмних продуктів.

Для забезпечення виконання цих цілей, було створено клас TransactionController (рис. 3.11).

```
[HttpGet(template: "dbTransactions")]
0 references
public async Task<IActionResult> GetDbTransactionsAsync([FromQuery] GetTransactionsRequestDTO request)...

// ReSharper disable once IdentifierTypo
[HttpGet(template: "privat24Transactions")]
0 references
public async Task<IActionResult> GetPrivat24TransactionsAsync(
    [FromQuery] DateTime? startDate,
    [FromQuery] DateTime? endDate,
    [FromQuery] bool performSync = false)...

[HttpGet(template: "synchronize")]
0 references
public async Task<IActionResult> SynchronizeAsync(
    [FromQuery] DateTime? startDate,
    [FromQuery] DateTime? endDate)...
```

Рис. 3.11 – Методи контролера для зовнішнього API

Даний клас задає назви точкам доступу, налаштовує шлях до них, вказує їхній тип, параметри і їхні типи.

3.5 Тестування програмного продукту

Тестування програмного забезпечення - це спосіб перевірити, чи відповідає фактичний програмний продукт очікуваним вимогам, і забезпечити відсутність дефектів програмного продукту. Він передбачає виконання програмних / системних компонентів за допомогою ручних або автоматизованих інструментів для оцінки одного або декількох цікавих властивостей. Метою тестування програмного забезпечення є виявлення помилок, прогалин або відсутніх вимог на відміну від фактичних вимог.

Тестування програмного забезпечення має важливе значення, оскільки, якщо в програмному забезпеченні є помилки або помилки, його можна виявити завчасно і вирішити перед доставкою програмного продукту. Правильно протестований програмний продукт забезпечує надійність, безпеку та високу продуктивність, що в подальшому призводить до економії часу, економічності та задоволеності споживачів.

Тестування важливо, оскільки помилки програмного забезпечення можуть бути дорогими або навіть небезпечними. Помилки програмного забезпечення потенційно можуть спричинити грошові та людські втрати.

Ось переваги використання тестування програмного забезпечення:

− Економічна: це одна з важливих переваг тестування програмного забезпечення. Своєчасне тестування будь-якого IT-проекту допоможе заощадити ваші гроші на тривалий термін. У разі виявлення помилок на попередньому етапі тестування програмного забезпечення, виправлення коштує менше.

− Безпека: це найбільш уразлива і делікатна перевага тестування програмного забезпечення. Люди шукають надійні товари. Це допомагає усунути ризики та проблеми раніше.

− Якість продукту: це основна вимога будь-якого програмного продукту. Тестування гарантує, що якісний продукт доставляється споживачам.

− Задоволеність клієнтів: основна мета будь-якого товару - задоволення своїх клієнтів. Тестування UI / UX забезпечує найкращу взаємодію з користувачем.

Відповідно до ANSI / IEEE 1059, тестування в програмній інженерії - це процес оцінки програмного продукту, щоб визначити, чи відповідає поточний програмний продукт необхідним умовам чи ні. Процес тестування включає оцінку особливостей програмного продукту щодо вимог з точки зору відсутніх вимог, помилок або помилок, безпеки, надійності та продуктивності [14].

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Метою даної кваліфікаційної роботи є розробка бізнес-логіки розсилки інформації про факт оплати на базі платіжної системи Приват24. Відповідно до постановки задачі, більшість користувачів будуть використовувати розроблену інформаційну систему за допомогою персонального комп’ютера. Отже, користувачі даної програмної системи повинні дотримуватися вимог безпеки під час роботи з екранними пристроями.

Вимоги безпеки до робочих місць працівників з екранними пристроями:

1. Робочі місця працівників з екранними пристроями мають бути спроектовані так і мати такі розміри, щоб працівники мали простір для зміни робочого положення та рухів.

2. Для забезпечення безпеки та захисту здоров’я працівників усе випромінювання від екранних пристроїв має бути зведене до гранично допустимого рівня (вплив на людину факторів довкілля - шуму, вібрації, забруднювачів, температури тощо, який не спричиняє соматичних або психічних розладів, а також змін стану здоров’я, працездатності, поведінки, що виходять за межі пристосувальних реакцій) з погляду безпеки та охорони здоров’я працівників [15].

3. Організація робочого місця працівника з екранними пристроями має забезпечувати відповідність усіх елементів робочого місця та їх розташування ергономічним, антропологічним, психофізіологічним вимогам, а також характеру виконуваних робіт.

4. Освітлення робочого місця працівника з екранними пристроями має створювати відповідний контраст між екраном і навколишнім середовищем (з урахуванням виду роботи) та відповідати вимогам ДСанПІН 3.3.2.007-98 [15].

5. Мікроклімат виробничих приміщень з робочими місцями працівників з екранними пристроями має підтримуватись на постійному рівні та відповідати

вимогам Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042- 99, затверджених постановою Головного державного санітарного лікаря України від 01 грудня 1999 року № 42 (далі - ДСН 3.3.6.042-99) [15].

6. Робочий стіл або робоча поверхня повинні бути достатнього розміру та мати поверхню з низькою відбивною здатністю, допускати гнучкість під час розміщення екрана, клавіатури, документів і відповідного устаткування.

7. Робоче крісло має бути стійким і дозволяти працівнику з екранними пристроями легко рухатися та займати зручне положення.

Мінімальні вимоги безпеки під час роботи з екранними пристроями:

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

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

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

4. Не допускається:

– виконувати технічне обслуговування, ремонт і налагодження екранних пристроїв безпосередньо на робочому місці працівника під час роботи з екранними пристроями;

– відключати захисні пристрої, самочинно проводити зміни у конструкції та складі екранних пристроїв або їх технічне налагодження;

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

5. Під час виконання робіт операторського типу, пов’язаних з нервово-емоційним напруженням, у приміщеннях під час роботи з екранними пристроями, на пультах і постах керування технологічними процесами та в інших приміщеннях мають дотримуватися оптимальні умови мікроклімату відповідно до вимог ДСН 3.3.6.042-99 [15].

Мінімальні вимоги безпеки до екранних пристроїв:

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

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

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

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

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

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

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

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

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

10. Поверхня клавіатури має бути матовою, щоб уникнути віддзеркалювання. Розташування клавіш і самі клавіші мають полегшувати роботу із клавіатурою. Позначення клавіш повинно бути достатньо контрастним і розбірливим.

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

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

Під час розробки, тестування та впровадження інформаційної системи були дотримані всі вимоги, норми та державні стандарти з охорони праці.

4.2 Безпека в надзвичайних ситуаціях

4.2.1 Аналіз потенційних небезпек

Основними потенційними небезпеками при проведенні робіт в офісі є такі:

- небезпека ураження електричним струмом, внаслідок недотримання правил електробезпеки або виходу з ладу електроприладів;

- порушення роботи кістково-м’язового апарату внаслідок тривалих статичних навантажень при роботі з ПК.

- нервово-психічні перевантаження внаслідок постійного контакту з клієнтами, колегами по роботі, керівництвом при вирішенні робочих питань, які можуть носити конфліктний характер і призвести до емоційного дискомфорту, внутрішнього роздратування, емоційної нестабільності та захворювань нервової системи;

- незадовільні ергономічні характеристики робочого місця внаслідок нераціонального планування робочого місця, що може призвести до механічних травм, уражень електричним струмом та порушень кістково-м’язового апарату;

- негативний вплив недостатнього освітлення робочої зони на зір та продуктивність роботи працюючого, внаслідок несправності освітлювальних приладів або неправильного проектування освітлювальної системи;

- негативний вплив незадовільних параметрів повітряного середовища робочої зони на здоров’я працюючого, внаслідок неправильного проектування системи вентиляції або несправності її несправності;

- негативний вплив підвищеного рівня шуму на психоемоційний стан працюючого, який пов’язаний з використанням застарілої периферійної техніки, кондиціонерів, копіювальної техніки, освітлювальних приладів;

- небезпека загоряння у зв'язку із несправністю електричного обладнання, недотримання, або порушення правил протипожежної безпеки обслуговуючим персоналом, що може призвести до пожежі.

- неправильні дії персоналу у надзвичайних ситуаціях.

4.2.2 Заходи щодо забезпечення безпеки

Приміщення офісу, в яких перебувають співробітники галузі управління персоналом, відносяться до приміщень без підвищеної небезпеки ураження електричним струмом.

Обладнання, що використовується в цих приміщеннях є споживачем електроенергії, що живиться від змінного струму 220 В від мережі з заземленою нейтраллю, та відноситься до електроустановок до 1000В закритого виконання. За способом захисту людини від ураження електричним струмом відповідає згідно з ГОСТ 12.2.007.0-75* (2001) «ССБТ. Изделия электротехнические. Общие требования безопасности» I (стаціонарні комп’ютери,) та II (освітлювальні прилади, кондиціонери, опалювальні пристрої, ноутбуки, сканери) класу захисту.

Згідно «Правилам улаштування електроустановок» (далі «ПУЕ») виконані такі групи заходів з електробезпеки:

Конструктивні заходи забезпечують захист від випадкового дотику до струмопровідних частин за допомогою їх ізоляції та захисних оболонок.

Згідно з ГОСТ 12.1.009-76 (1999) «ССБТ. Электробезопасность. Термины и определения» у приладах II класу захисту використовується подвійна ізоляція - електрична ізоляція, що складається з робочої і додаткової ізоляції.

Так як згідно з НПАОП 40.1-1.32-01 «Правила устройства электроустановок. Электрооборудование специальных установок» офісні приміщення у більшості своїй відносяться до класу пожежонебезпечної зони П-IIа (приміщення, в яких містяться тверді горючі речовини), тому передбачений ступінь захисту ізоляції обладнання IP44.

Схемно-конструктивні заходи.

Призначені для забезпечення захисту від ураження електричним струмом при дотику до металевих оболонок, які можуть опинитися під напругою в результаті аварій.

Згідно з ГОСТ 12.1.030-81 (2001) «ССБТ. Электробезопасность. Защитное заземление, зануление» у приміщеннях галузі управління персоналом влаштовується занулення.

Організаційні заходи.

Експлуатація електроустановок і електроустаткування проводиться відповідно до НПАОП 40.1-1.01-97 «Правила безпечної експлуатації електроустановок» (далі «ПБЕЕ») та НПАОП 40.1-1.21-98 «Правила безпечної експлуатації електроустановок споживачів» (далі «ПБЕЕС»).

Для запобігання статистичного навантаження при користуванні ПК рекомендовано використовувати перерви в роботі 10 хв. через кожні дві години. Синдром зап’ястного каналу, або тунельний синдром зап’ястя, який може бути наслідком хронічної травми, трапляється у людей внаслідок тривалої роботи з мишею: постійні напруга і здавлювання приводить до мікротравм, здавлювання нерву прилеглими оточуючими тканинами, через що виникає набряк.

Щоб тунельний синдром вас не турбував, потрібно дотримуватися кількох правил організації робочого місця:

- оптимальна висота клавіатури від підлоги – 65-75 см;

- наявність ергономічних і зручних особисто для вас миші і клавіатури;

- можливість регулювання висоти і нахилу клавіатури (відстань від поверхні стола до середини клавіатури – не більше 30 мм, кут підйому клавіатури – від 2° до 15°);

- наявність у клавіатури підставки для рук; - наявність килимка для миші з захистом від тунельного синдрому (спеціальний виступ забезпечує правильне положення кисті);

- наявність стільця або крісла з підлокітниками.

При роботі з мишкою і клавіатурою також слід дотримуватися певних правила. Коли ви набираєте текст, рука повинна бути зігнута в лікті під прямим кутом (90°), а при роботі з мишкою стежте, щоб кисть була прямою і лежала на столі якнайдалі від краю. До речі, час роботи з комп'ютером слід обмежити до дійсно необхідного.

Щоб попередити тунельний синдром потрібно робити спеціальні вправи для кистей – чим частіше, тим краще. Ці вправи допоможуть поліпшити кровообігу в м'язах і розтягнути їх. Комплекс вправ потрібно повторювати приблизно кожні 45 хвилин, тривалість однієї вправи – 1-2 хв.

Нервове напруження впливає на серцево-судинну систему, збільшуючи артеріальний тиск і частоту пульсу, а також на терморегуляцію організму та емоційні стани працівника. Особливу роль у запобіганні втомі працівників відіграють професійний відбір, організація робочого місця, правильне робоче положення, ритм роботи, раціоналізація трудового процесу, використання емоційних стимулів, впровадження раціональних режимів праці і відпочинку тощо. Боротьба зі втомою, в першу чергу, зводиться до покращення санітарно-гігієнічних умов виробничого середовища (ліквідація забруднення повітря, шуму, вібрації, нормалізація мікроклімату, раціональне освітлення тощо).

Крім того, для профілактики втоми працівників застосовуються специфічні методи, до яких можна віднести засоби відновлення функціонального стану зорового та опорно-рухового апарату, зменшення гіподинамії, підсилення мозкового кровообігу, оптимізацію розумової діяльності.

Загальні ергономічні вимоги встановлено ДСТУ ISO 9241-1:2003 «Ергономічні вимоги до роботи з відео-терміналами в офісі. Частина 1. Загальні положення». Організація робочого місця передбачає: правильне розміщення робочого місця у виробничому приміщенні; вибір ергономічно-обґрунтованого

робочого положення, виробничих меблів з урахуванням антропометричних характеристик людини; раціональне компонування обладнання на робочих місцях; врахування характеру та особливостей трудової діяльності.

ВИСНОВКИ

Під час виконання роботи було здійснено аналіз існуючих платіжних систем, обрано актуальну та найпоширенішу платіжну систему Приват24, що має масове використання та використовується великою кількістю користувачів.

Було оглянуто технічну літературу на предмет створення веб-сервісів, що дають змогу комунікувати з постачальниками даних та авторизувати їх у системі постачальника.

Для розробки було вибрано ASP.NET Core Web API, проаналізовано переваги та недоліки даного рішення.

Здійснено огляд актуальних способів авторизації на стороні постачальника, обрано OAuth.

Для інтеграції з Приват24 API здійснено короткий огляд існуючих функцій, способів автентифікації та авторизації.

Спроектовано серверну архітектуру, що є оптимальною для даного типу рішень, забезпечує масштабованість, здатність до розширення функціоналу та підтримки в майбутньому.

Здійснено короткий огляд середовища розробки Visual Studio та середовища управління базами даних SQL Server Management Studio, а також платформи .NET Core та мови програмування C#, особливостей реалізації та виконання програм, написаних з використанням даної технології.

Розглянуто способи роботи з базою даних та здійснено короткий огляд ORM-рішень, що дозволяють здійснювати роботу з БД без написання SQL запитів. Описано можливі підходи при роботі з ORM Entity Framework Core, обрано Database First як основний для виконання роботи.

Як результат виконання роботи спроектовано та розроблено систему, що відповідає технічним, функціональним та нефункціональним вимогам, здійснює комунікацію з Приват24 API і здійснює синхронізацію платежів (транзакцій).

СПИСОК ПОСИЛАНЬ

1. What is a payment system? [Електронний ресурс] // StreetDirectory. – 2020. – Режим доступу до ресурсу: https://www.streetdirectory.com/travel_guide/162147/money_management/what_is_a_payment_system.html.

2. Payments and transfers [Електронний ресурс] // PrivatBank – Режим доступу до ресурсу: https://en.privatbank.ua/payments-and-transfers.

3. What is REST [Електронний ресурс] // ReftfulApi. – 2018. – Режим доступу до ресурсу: https://restfulapi.net/.

4. Introduction to ASP.NET Core [Електронний ресурс] // Microsoft. – 2017. – Режим доступу до ресурсу: https://aspnetcore.readthedocs.io/en/stable/intro.html.

5. What is an ORM and Why You Should Use it [Електронний ресурс] // Medium. – 2018. – Режим доступу до ресурсу: https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a.

6. Entity Framework Core [Електронний ресурс] // Entity Framework Tutorial. – 2020. – Режим доступу до ресурсу: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx.

7. What is OAuth? How the open authorization framework works [Електронний ресурс] // CSOOnline. – 2019. – Режим доступу до ресурсу: https://www.csoonline.com/article/3216404/what-is-oauth-how-the-open-authorization-framework-works.html.

8. Scrum Overview for Agile Software Development [Електронний ресурс] // MountainGoat. – 2020. – Режим доступу до ресурсу: https://www.mountaingoatsoftware.com/agile/scrum/resources/overview.

9. Introduction to Visual Studio [Електронний ресурс] // GeeksforGeeks. – 2019. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/introduction-to-visual-studio/.

10. Регистрация мерчанта Приват24 [Електронний ресурс] // PrivatBank. – 2020. – Режим доступу до ресурсу: https://api.privatbank.ua/#p24/registration.

11. 3-Tier Architecture: A Complete Overview [Електронний ресурс] // LogiReport. – 2020. – Режим доступу до ресурсу: https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/.

12. Database Normalization [Електронний ресурс] // W3Schools. – 2020. – Режим доступу до ресурсу: https://www.w3schools.in/dbms/database-normalization/.

13. What is MS SQL? [Електронний ресурс] // HostShopper. – 2020. – Режим доступу до ресурсу: https://www.host-shopper.com/what-is-ms-sql.html.

14. What is Software Testing? Definition, Basics & Types [Електронний ресурс] // Guru99. – 2020. – Режим доступу до ресурсу: https://www.guru99.com/software-testing-introduction-importance.html.

15. ДСанПІН 3.3.2.007-98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин – 1998.

16. МЕТОДИЧНІ ВКАЗІВКИ до виконання атестаційної роботи магістра за спеціальністю 121 – ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (Освітньо-професійна програма - «ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМ», Освітньо-наукова програма - «ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ») для с / М. Р.Петрик, Д. М. Михалик, О. Ю. Петрик, Г. Б. Цуприк. – 2020. – С. 52.

ДОДАТКИ

ДОДАТОК А

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ
КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку кваліфікаційної роботи
«Розробка бізнес-логіки розсилки інформації про факт оплати на базі платіжної системи Приват24»

Розробники:
виконавець ст. гр. СПм-61
Зашко Богдан Олегович

_____
(підпис)

керівник роботи
Петрик Михайло Романович

_____
(підпис)

Тернопіль 2020

ЗМІСТ

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану підготовки магістрів за спеціальністю 121 «Інженерія програмного забезпечення».

Тема проекту: «Розробка бізнес-логіки розсилки інформації про факт оплати на базі платіжної системи Приват24».

Термін виконання: до «___» _____ 2020р.

2 ПРИЗНАЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Інформаційна система призначена перегляду транзакцій з Приват24, синхронізації їх в базу даних, та вибірка з бази даних.

Інформаційна система буде корисною для працівників ТНТУ та має адніміструативне призначення.

Інформаційна система дозволить відмовитись від використання паперових квитанцій про оплату на користь електронного відображення платежів.

3 ВИМОГИ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1Функціональні вимоги

Система повинна передбачати дві ролі:

− користувач

− адміністратор

Для користувачів система надає доступ до наступних функцій:

- отримувати дані про транзакції із опцією синхронізації з БД
- запускати мануальну синхронізації даних про транзакції з Приват24 в базу даних

Для адміністратора цей список поповнюється можливістю конфігурації системи

Набір даних функцій дозволяє користувачу краще контролювати фінансовий обіг на рахунку в університетських цілях.

3.2 Технічні вимоги

Вимоги до серверної частини: .NET Core, не старіше 3.1 версії, можливість конфігурації системи, стабільність, висока швидкість роботи.

3.3 Програмні вимоги

Використання СУБД: MS SQL.

Розробка клієнтської частини: ASP.NET Core.

Додаткові вимоги: можливість отримання транзакцій напряму з Приват24.

4. ЕТАПИ РОЗРОБКИ

Розробка інформаційної системи проводиться в наступному порядку:

- аналіз існуючого API;
- вибір засобів розробки та архітектури системи;
- розробка програмного забезпечення системи;
- тестування інформаційної системи на реальних даних;
- оформлення супровідної документації;
- здача роботи.

Результати виконання кожного етапу проекту погоджуються з керівником роботи.

5. СУПРОВІДНА ДОКУМЕНТАЦІЯ

Для інформаційної системи повинні бути розроблені наступні документи:

- пояснювальна записка до роботи;
- презентація роботи;
- рецензія на роботу;
- диск з кодом інформаційної системи.

Пояснювальна записка до кваліфікаційної роботи оформляється згідно діючих вимог до нормоконтролю проектів.

6. ПОРЯДОК ЗДАЧІ РОБОТИ

Розроблена інформаційна системи повинна відповідати вимогами, що складаються з перерахованих у п.3.1 цього документу характеристик.

Для здачі проекту необхідно підготувати весь перелік документів зазначений у п.5 цього документу.

Приймання проекту проводиться спеціально створеною комісією в термін зазначені в п.1 цього документу.

7. ВІДМІТКИ ПРО ВИКОНАННЯ ЕТАПІВ ТА ЗМІНИ В ПРОЕКТІ

Назва етапу	Відмітка*
Аналіз існуючого API	
Вибір засобів розробки та архітектури системи	
Розробка програмного забезпечення системи	
Тестування інформаційної системи на реальних даних	
Оформлення супровідної документації	

* відмітки про виконання етапу ставляться керівником проекту

ДОДАТОК Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

VIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

9–10 грудня 2020 року

ТЕРНОПІЛЬ
2020

СЕКЦІЯ 4. **ПРОГРАМНА ІНЖЕНЕРІЯ ТА МОДЕЛЮВАННЯ СКЛАДНИХ РОЗПОДІЛЕНИХ СИСТЕМ**

ДОДАТОК В