Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для проведення соціальних опитувань. Back-end, API

(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-6

Поронько К.А.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Валенда Н.А.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Пороньку Кирилу Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для проведення соціальних опитувань. Back-end, API

Затверджена наказом по університету від 20.05. 2024р. № 471 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18.06.2024

3. Вихідні дані до роботи Розробити програмну систему для проведення соціальних опитувань, а саме такий елемент системи: Back-end API частина.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	09.04.2024	*виконано*
2	Створення специфікації ПЗ	16.04.2024	*виконано*
3	Проектування ПЗ	24.04.2024	*виконано*
4	Розробка ПЗ	23.05.2024	*виконано*
5	Тестування ПЗ	01.06.2024	*виконано*
6	Оформлення пояснювальної записки	06.06.2024	*виконано*
7	Підготовка презентації та доповіді	08.06.2024	*виконано*
8	Попередній захист	13.06.2024	*виконано*
9	Нормоконтроль, рецензування	13.06.2024	*виконано*
10	Здача роботи у електронний архів	15.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	15.06.2024	*виконано*

Дата видачі завдання 08 квітня 2024р.

Студент _____ Поронько К. А.
(підпис)

Керівник роботи _____ доц. кафедри ПІ Валенда Н.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Звіт з передатестаційної практики містить 100 сторінок, 5 розділів, 55 рисунків, 11 джерел посилання.

ОПИТУВАННЯ, СОЦІУМ, ТАРГЕТИНГ, API, .NET, ЧИСТА АРХІТЕКТУРА, ORM, СТАТИСТИКА, ПІДПИСКА, PAYPAL.

Об'єктом розробки є Back-end частина веб-додатку «Pure Survey», а саме API призначеного для створення та проведення соціальних опитувань онлайн.

Метою розробки є проектування та створення системи, яка дасть змогу користувачам не лише створювати опитування з гнучкими налаштуваннями, але й вбудовувати їх на сторонні веб-сайти, отримуючи при цьому вичерпну статистику відповідей по різноманітних критеріях.

Метод рішення – ASP.NET Core Web API для створення RESTful API, Entity Framework Core ORM для спрощення роботи з базою даних та забезпечення об'єктно-орієнтованого підходу, а також MS SQL Server як систему керування базами даних.

У ході роботи було визначено вимоги, спроєктовано та розроблено API-частину програмної системи «Pure Survey» для проведення опитувань, яка побудована за принципами чистої архітектури та має потенціал стати важливим інструментом для збору даних та аналітики.

SURVEY, SOCIETY, TARGETING, API, .NET, CLEAN ARCHITECTURE, ORM, STATISTICS, SUBSCRIPTION, PAYPAL.

The object of the development of this report is the Back-end part of the Pure Survey web application, namely the API designed to create and conduct social surveys online.

The purpose of the development is to design and create a system that will allow users not only to create surveys with flexible settings, but also to embed them on third-

party websites, while obtaining comprehensive statistics of responses according to various criteria.

Solution method – ASP.NET Core Web API to create RESTful API, Entity Framework Core ORM to simplify working with the database and provide an object-oriented approach, as well as MS SQL Server as a database management system.

In the course of the work, the requirements were determined, the API part of the Pure Survey software system for creating surveys was designed and developed, which is built on the principles of pure architecture and has the potential to become an important tool for data collection and analytics.

Я, Поронько Кирил Андрійович, студент гр. ПЗПІ-20-6, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система для проведення соціальних опитувань. Back-end, API», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений із діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

В епоху цифрових технологій збір даних та проведення досліджень стали невід'ємною частиною багатьох галузей, включаючи бізнес, маркетинг, соціологію, освіту, охорону здоров'я та інші. Опитування є одним із найпоширеніших і найефективніших інструментів для отримання цінної інформації від цільових аудиторій. Проте традиційні методи проведення опитувань, такі як телефонні дзвінки, поштові розсилки чи особисті інтерв'ю, можуть бути дорогими, трудомісткими та часто не забезпечують достатньої гнучкості для задоволення мінливих потреб дослідників.

Саме тому виникла необхідність у створенні зручної та масштабованої платформи для створення, проведення та аналізу опитувань в онлайн-режимі. «Pure Survey» – це новітній веб-сервіс, розроблений для задоволення цих потреб. Він дозволяє користувачам легко створювати різноманітні опитування з використанням простого конструктора, налаштовувати їх під свої вимоги та цільову аудиторію, а також інтегрувати їх на інші веб-сайти, соціальні мережі чи мобільні додатки для збору відповідей.

Ця платформа забезпечує зручний інтерфейс для створення опитувань з різноманітними типами запитань, гнучкі інструменти налаштування зовнішнього вигляду, логіки проходження, правил валідації тощо. Вона також пропонує можливість таргетингу на певні демографічні групи, країни чи регіони, а також потужні аналітичні інструменти для перегляду та аналізу зібраних даних у вигляді діаграм та графіків.

Завдяки «Pure Survey» дослідники, маркетологи, підприємці, викладачі та інші зацікавлені сторони отримують ефективний, економічний та зручний спосіб проводити опитування, збирати цінні відгуки та приймати рішення на основі достовірних даних. Ця система прагне спростити та вдосконалити процес збору, аналізу та використання даних з опитувань, забезпечуючи користувачів потужним, але інтуїтивно зрозумілим інструментом для досягнення своїх цілей у різних сферах діяльності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Соціологічне дослідження – це будь-яке дослідження, незалежно від методу збору та аналізу даних, яке формує, розвиває, уточнює чи спростовує теорії в галузі соціології. В Україні цей термін часто ототожнюють з опитуваннями громадської думки, хоча насправді репрезентативне опитування є лише одним із багатьох методів соціального дослідження. Дослідники проводять соціологічні дослідження для розширення знань у різних сферах або перевірки гіпотез, що, в свою чергу, приносить користь суспільству та задовольняє його потреби. Інформація, зібрана безпосередньо від людей, допомагає уряду та бізнесу розробляти цілеспрямовані заходи реагування на потреби суспільства.

Існують два основні способи проведення соціальних досліджень: кількісний та якісний. Якісні дослідження передбачають збір даних у природному середовищі респондентів, що сприяє зміцненню довіри та отриманню реальних і точних даних. Елементами якісного дослідження є питання з відкритою відповіддю та інтерв'ю. Збір даних може відбуватися за допомогою фокус-груп, інтерв'ю, етнографічних досліджень, кейс-стаді тощо. Кількісні дослідження передбачають відбір репрезентативної вибірки з цільової аудиторії для збору даних за допомогою опитувань, анкетувань, інтерв'ю, експериментальних досліджень тощо. Метою є отримання числових даних, які можна легко проаналізувати за допомогою статистичних програм.

Серед методів кількісних досліджень, особливо в Україні, превалює проведення опитувань (анкетувань та інтерв'ю). Їхні удосконалення, такі як імовірнісна вибірка та стандартизовані вимірювання, дозволяють дослідникам бути впевненими, що вибірка не є упередженою, і що вони отримують порівнянну інформацію про кожну особу з відібраної сукупності. Переваги опитувань полягають у гнучкості (можливість ставити багато запитань на одну тему та легко додавати нові під час розробки чи тестування), можливості включити велику кількість змінних для статистичного контролю потенційних помилок, можливості

перевірити гіпотетичні причинно-наслідкові зв'язки, самостійності проходження, поширення різними каналами та зниженні витрат завдяки структурованим шаблонам. Недоліками є необхідність розробки узагальнених відповідей, що може призвести до втрати деталей, обмеження вимірювання лише тим, що люди готові сказати, ігнорування соціально неприйнятних ставлень, а також відсутність доказів, які підтверджують надані відповіді.

Саме онлайн опитування мають низку переваг: вони дешевші, швидші, можуть охопити ширшу аудиторію і забезпечити більшу анонімність респондентів. Однак існують труднощі зі створенням, налаштуванням і розповсюдженням таких опитувань, а також зі збором та аналізом отриманих результатів. Існуючі програмні рішення для проведення онлайн опитувань мають як переваги, так і недоліки. З одного боку, вони дозволяють швидко створювати опитування з різними типами питань, налаштовувати їх дизайн та логіку, поширювати різними каналами та збирати відповіді в режимі реального часу. Але з іншого боку, наявні системи часто мають обмежений функціонал, незручний інтерфейс, відсутність аналітичних інструментів тощо.

Досить показовою є статистика, що ілюструє сприйняття респондентами зрозумілості питань в традиційних опитуваннях та онлайн-опитуваннях (див. рисунок 1.1).

З рисунку видно, що онлайн-опитування рідше викликають плутанину та незрозумілість питань серед респондентів порівняно з особистими інтерв'ю. При створенні опитувань дослідники мають керуватися принципами, що дозволяють підвищити якість опитувальників та збільшити кількість відгуків. Це включає наявність змістовного вступу з інструкціями, інформацією про організацію та тривалість, чіткий та зрозумілий дизайн, правильне розташування запитань, використання простої термінології, уникнення двозначностей та надмірної деталізації, а також наявність залишкового варіанту відповіді.

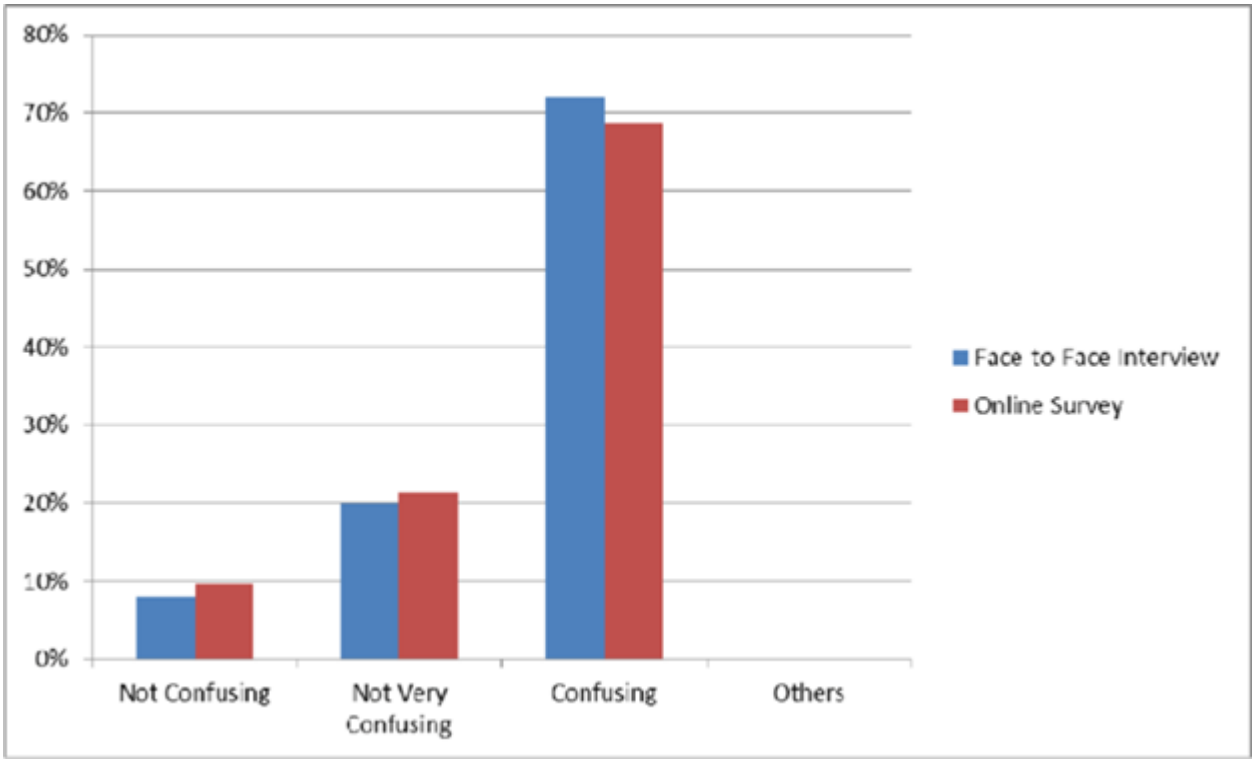


Рисунок 1.1 – Порівняння сприйняття зрозумілості питань в опитуваннях різних типів

Сучасні системи для створення соціальних онлайн опитувань мають відповідати ряду вимог, серед яких: зручний конструктор для швидкого створення різноманітних опитувань з багатьма типами питань, гнучке налаштування дизайну, логіки проходження, правил валідації, підтримка таргетингу, інтеграція з веб-сайтами, соцмережами, додатками для поширення опитувань, аналітичні інструменти для перегляду статистики, будування діаграм, модульна архітектура для розширюваності, відповідність вимогам безпеки, захисту даних та доступності. Створення якісного програмного рішення для опитувань дозволить значно спростити та вдосконалити процес збору і аналізу даних для соціологічних, маркетингових, освітніх, наукових та інших досліджень в онлайн середовищі. Це відкриває нові можливості для бізнесу, державних установ, науковців у прийнятті ефективних рішень на основі своєчасної та достовірної інформації від широких цільових аудиторій.

Загалом, галузь соціальних досліджень, зокрема онлайн опитувань, перебуває на стадії активного зростання і розвитку. Поширення інтернету,

мобільних пристроїв, хмарних технологій робить процес збору даних більш доступним, гнучким та масштабованим. Разом з тим, зростають вимоги до якості та функціональності відповідних програмних рішень. Сучасні веб-платформи та додатки мають надавати потужний та водночас інтуїтивно зрозумілий інструментарій

1.2 Виявлення проблем та актуалізація рішень

Однією з основних проблем у процесі проведення соціальних опитувань є складність створення опитувань з гнучкими налаштуваннями та можливістю вбудовування на різні вебсайти. Існуючі рішення часто є обмеженими або незручними у використанні, що ускладнює процес розробки та розповсюдження опитувань. Крім того, проблемою є низька залученість респондентів через необхідність відвідування спеціальних сайтів для проходження опитувань, що значно знижує ефективність та репрезентативність зібраних даних.

Ще однією суттєвою проблемою є складність аналізу зібраних даних та перегляду статистики відповідей. Часто статистика є обмеженою або не дає можливості фільтрувати дані за потрібними критеріями, такими як демографічні показники, геолокація чи часові проміжки. Це ускладнює процес отримання цінної аналітичної інформації та робить складнішим прийняття рішень на основі зібраних даних.

Для вирішення цих проблем пропонується створити систему «Pure Survey», яка дозволить користувачам легко створювати опитування з гнучкими налаштуваннями, публікувати їх на різних вебсайтах за допомогою вбудованих скриптів та переглядати детальну статистику відповідей з можливістю фільтрації та візуалізації. Система повинна забезпечити зручний та інтуїтивно зрозумілий інтерфейс для створення опитувань, включаючи можливість налаштування різних типів питань, перекладів, таргетингу за країнами, правилами проходження опитувань та іншими параметрами.

Проаналізувавши конкурентів на даному ринку, де найбільш популярним аналогом, звісно, є Google Forms, можна зробити певні порівняльні характеристики.

Безперечно, Google Forms є популярним і зручним рішенням для проведення простих онлайн опитувань. Його безкоштовність, інтеграція з екосистемою Google та інтуїтивно зрозумілий інтерфейс роблять його привабливим варіантом для невеликих проектів чи індивідуального використання. Проте, незважаючи на свою простоту, Google Forms має певні обмеження в налаштуваннях, можливостях аналізу даних та персоналізації, а також не дозволяє вбудовувати опитування на сторонні веб-сайти.

У порівнянні з Google Forms, система «Pure Survey» пропонує значно більше гнучкості та функціональних можливостей. Завдяки використанню сучасних технологій, таких як ASP.NET Core Web API та чистої архітектури, дана система забезпечує вищий рівень масштабованості, безпеки та можливостей для подальшого розширення. Одна з ключових переваг «Pure Survey» – це можливість налаштування різноманітних типів питань, перекладів, таргетингу за країнами, правил проходження опитувань та персоналізації зовнішнього вигляду за допомогою шаблонів. Це надає користувачам безпрецедентну гнучкість у створенні опитувань, адаптованих до їхніх конкретних потреб.

Крім того, наша система дозволяє вбудовувати опитування на різні веб-сайти за допомогою спеціальних скриптів, що значно підвищує залученість респондентів та зручність розповсюдження. Це є суттєвою перевагою порівняно з Google Forms, яка не пропонує такої можливості.

Ще однією вагомою перевагою «Pure Survey» є наявність потужної аналітики та статистики відповідей. Користувачі матимуть змогу переглядати детальну інформацію про відповіді, фільтрувати дані за різними критеріями та візуалізувати результати у зручному форматі. Це дозволить отримувати більш цінну аналітичну інформацію для прийняття обґрунтованих рішень на основі зібраних даних.

Проаналізуємо ще декілька аналогів системи.

Typeform, в свою чергу, вирізняється сучасним та привабливим дизайном своїх опитувальників, пропонуючи зручний конструктор з перетягуванням для створення опитувань з цікавою візуалізацією питань. Його інтерфейс інтуїтивно зрозумілий, що робить процес створення опитувань простим і швидким навіть для користувачів без технічних навичок. Користувачі можуть легко налаштовувати опитування, використовуючи різні типи питань, інтегровані мультимедійні елементи та логіку проходження опитувань. Однак функціональність платформи обмежена для складних багаторівневих опитувань, що може стати значним недоліком для користувачів, які потребують більш просунутих функцій. Порівняно з нашою системою, аналітика та фільтрація даних на платформі не є достатньо потужними, що ускладнює отримання глибоких інсайтів з зібраних даних. Крім того, підтримка вбудовування на сторонні сайти також є обмеженою, що знижує гнучкість у розміщенні опитувань на різних платформах та вебсайтах.

SurveyLegend позиціонується як доступне рішення для створення опитувань, пропонуючи низьку вартість або безкоштовні тарифи та базовий набір функцій. Платформа дозволяє користувачам створювати опитування з використанням простого редактора та базових налаштувань, що може бути привабливим для невеликих підприємств або індивідуальних користувачів з обмеженим бюджетом. Проте система страждає від обмежених можливостей налаштування логіки, дизайну та типів питань, що обмежує її застосування для більш складних та специфічних завдань. Відсутність підтримки багатомовності та таргетингу значно знижує її привабливість для міжнародних досліджень або кампаній, які потребують персоналізованого підходу до різних аудиторій. Низька продуктивність та надійність на безкоштовних тарифах може призводити до проблем зі стабільністю роботи сервісу під час високих навантажень, що негативно впливає на користувацький досвід. Крім того, система не пропонує достатньо гнучкої аналітики та візуалізації даних, що робить її менш привабливою для користувачів, які потребують детального аналізу результатів опитувань. У порівнянні з нашою системою, SurveyLegend значно програє у функціональності, гнучкості налаштувань та можливостях аналітики.

1.3 Постановка задачі

У ході роботи повинна бути створена система для проведення опитувань, яка дозволяє користувачам легко створювати гнучкі опитування, вбудовувати їх на сторонні веб-сайти та отримувати детальну статистику відповідей за різними критеріями.

З технічної точки зору, система повинна бути розроблена з використанням сучасних технологій для забезпечення гнучкості, масштабованості та надійності. API-частина системи має бути розроблена з використанням ASP.NET Core Web API [1] та чистої архітектури [2] для забезпечення гнучкості, можливості розширення та тестування. Вибір цих технологій обумовлений їхньою здатністю підтримувати високу продуктивність та масштабованість системи. У якості системи керування базами даних (СКБД) необхідно використовувати MS SQL Server [3] для надійного зберігання даних опитувань, відповідей та облікових даних користувачів. Слід також звернути увагу на забезпечення безпеки, конфіденційності даних, масштабованості та сумісності з різними браузерами та пристроями.

Однією з важливих вимог є можливість вбудовування опитувань на різні веб-сайти. Система повинна генерувати вбудовані скрипти або віджети, які можна легко інтегрувати на сторонніх веб-сайтах. Це дозволить користувачам більш зручно розповсюджувати опитування та підвищувати залученість респондентів. Важливо забезпечити, щоб ці скрипти та віджети були легкими у використанні, інтеграції та налаштуванні, аби користувачі могли без зайвих технічних знань вбудовувати їх на свої платформи. Можливість налаштування зовнішнього вигляду опитувань для гармонійної інтеграції з дизайном веб-сайту забезпечить естетичну привабливість та відповідність бренду. Це включає вибір кольорів, шрифтів та стилів, що відповідають корпоративному дизайну веб-сайту, де буде розміщене опитування.

Система повинна надавати детальну статистику відповідей на кожне опитування, включаючи відсоток відповідей, демографічні дані респондентів, географічне розподілення та мова проходження. Це дозволить дослідникам

отримувати повну картину про поведінку та характеристики респондентів. Для отримання більш цінної аналітичної інформації необхідно реалізувати можливість фільтрації даних за різними критеріями, такими як стать, мова, геолокація та інші характеристики. Візуалізація даних у вигляді діаграм та графіків повинна бути зручною та інформативною, що дозволить дослідникам швидко та ефективно аналізувати результати. Діаграми, графіки та інші візуальні інструменти мають бути інтерактивними, щоб користувачі могли взаємодіяти з даними та проводити детальний аналіз.

Для забезпечення фінансової стійкості сервісу необхідно розробити механізм монетизації, який включає механізм оплати для публікації опитувань в Інтернеті. Це може бути досягнуто за допомогою інтеграції безпечних платіжних шлюзів PayPal, що забезпечить захист платіжних даних користувачів. Розробка відповідної інфраструктури для проведення запитів є критично важливою для забезпечення фінансової стійкості сервісу. Система повинна підтримувати модель підписки, яка дозволить користувачам отримувати доступ до розширених функцій та аналітики на постійній основі. Підписка може включати можливість проведення необмеженої кількості опитувань, публікація опитувань на інших веб-сайтах, доступ до розширених аналітичних інструментів тощо.

Всі ці кроки є ключовими для розробки ефективної та функціональної системи «Pure Survey», яка відповідатиме сучасним вимогам і забезпечить високоякісні результати соціологічних досліджень в онлайн-форматі. Важливим аспектом є також ретельне тестування системи. Це має включати як функціональні, так і нефункціональні аспекти, щоб гарантувати її надійність та відповідність заданим вимогам. Тестування повинно охоплювати різні сценарії використання, включаючи юніт-тестування, яке перевіряє окремі компоненти системи; інтеграційне тестування, що перевіряє взаємодію між модулями; мануальне тестування, яке імітує поведінку кінцевих користувачів для оцінки юзабіліті та загальної функціональності. Таке комплексне тестування забезпечить високу якість та надійність кінцевого продукту, а також задовольнить потреби користувачів у різних умовах експлуатації.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Функціональні вимоги

Розглянемо функціональні вимоги до системи. Вона повинна надавати можливість незареєстрованим користувачам створювати облікові записи, у процесі чого вони вказуватимуть свою електронну адресу, пароль та ім’я. Після цього зареєстровані користувачі матимуть змогу входити до системи, використовуючи свої облікові дані для авторизації. Авторизовані користувачі отримають право створювати групи опитувань. Під час створення групи вони зможуть налаштовувати різноманітні правила проходження опитувань у цій групі. Наприклад, можна буде дозволити чи заборонити отримання повідомлень після завершення опитування, дозволити або не дозволити зникнення форми після її проходження, встановити обмеження на кількість проходжень опитувань з одного пристрою за день та кількість разів, коли одне опитування може бути пройдене з одного пристрою. Крім того, користувачі матимуть змогу обирати тип відображення опитувань у групі, налаштовуючи колір, шрифт, розмір, місце розташування та шаблон.

Система забезпечить можливість для користувачів переглядати список створених ними груп опитувань. У цьому списку будуть відображатися всі групи, які були створені користувачем. Користувач зможе бачити назви груп, кількість опитувань у кожній групі та інші важливі параметри. Крім того, користувачі матимуть доступ до детальної інформації про кожну групу. Це включатиме всі налаштування, які були задані під час створення групи, такі як правила проходження опитувань, налаштування повідомлень, обмеження на проходження опитувань та інше. Також користувачі зможуть переглядати список опитувань, що входять до кожної групи. Це дозволить їм отримати повне уявлення про структуру та вміст кожної групи опитувань. Користувачі матимуть право видаляти створені ними групи опитувань. Видалення групи буде включати видалення всіх опитувань, що входять до цієї групи, а також усіх налаштувань, пов'язаних з групою. Це

дозволить користувачам ефективно управляти своїми групами опитувань і підтримувати їх актуальність та відповідність їхнім потребам.

Щодо самих опитувань, система забезпечить можливість користувачам редагувати текст питань, варіанти відповідей, переклади питань та налаштування таргетингу. Це дозволить створювати і налаштовувати опитування відповідно до конкретних вимог і цілей користувача. Користувачі зможуть переглядати список всіх створених опитувань з базовою інформацією, такою як назва, кількість питань та дата створення. Цей список буде доступний у вигляді зручного інтерфейсу, де користувач зможе швидко знаходити необхідні опитування та отримувати основну інформацію про них. Також вони матимуть доступ до детальної інформації про кожне опитування. Це включатиме список питань, їх переклади, налаштування таргетингу та дані про групу опитувань, до якої воно належить. Користувачі зможуть редагувати всі ці параметри, що дозволить їм адаптувати опитування під різні аудиторії та потреби. Задля активації опитування та можливості вбудовувати опитування в інші веб-сайти, користувач повинен буде сплатити за підписку на цю функцію. Тому повинна бути реалізована функціональність з підписки платіжної системи PayPal. Це дозволить користувачам легко та безпечно здійснювати оплату і отримувати доступ до додаткових функцій системи. Після вбудовування на інші веб-сайти та проходження опитувань певними гостями тих сайтів, користувач нашої системи матиме змогу побачити статистику опитувань. Це включатиме результати вибору варіантів, детальну статистику за гендером, за країною, за мовою тощо. Таким чином, користувачі зможуть аналізувати результати опитувань, отримувати важливу інформацію про аудиторію і приймати обґрунтовані рішення на основі цих даних.

Функціонал користувача-адміністратора в більшості полягає в створенні, редагуванні та видаленні темплейтів для опитувань, які будуть надавати певний шаблон зовнішнього вигляду опитування. Звичайний користувач зможе обирати ці темплейти для створення свого опитування. Адміністратори також матимуть доступ до управління налаштуваннями системи, контролю за дотриманням правил використання і забезпеченням безпеки даних користувачів та опитувань.

2.2 Нефункціональні вимоги

Розглянемо нефункціональні вимоги до системи опитувань. Одним з ключових аспектів є продуктивність. API системи повинно забезпечувати швидкий час відгуку на запити користувачів, щоб уникнути затримок та забезпечити належний досвід користувача. Користувачі очікують швидкого і безперебійного доступу до системи, тому час відгуку API повинен бути мінімальним навіть під час пікових навантажень. Крім того, API має бути здатним обробляти великі обсяги запитів та даних без значного зниження продуктивності. Це означає, що система повинна ефективно справлятися з високою кількістю одночасних з'єднань і запитів. Для підвищення продуктивності необхідно реалізувати належне кешування та оптимізацію запитів.

Безпека є іншим важливим аспектом. API системи повинно забезпечувати захист від різноманітних загроз, таких як ін'єкції, атаки перебором, атаки міжсайтового скриптингу (XSS) та підробки запитів міжсайтової підробки (CSRF). Для забезпечення безпеки необхідно реалізувати належне управління токенами доступу та їх перевірку. Важливі дані, такі як облікові дані користувачів та статистика опитувань, повинні бути захищені відповідним чином. Генерація JWT токенів при авторизації та їх використання для доступу до захищених ендпоінтів є важливим механізмом безпеки, який гарантує, що лише авторизовані користувачі можуть отримати доступ до конфіденційної інформації та функціоналу системи.

При авторизації користувача система генерує JWT (JSON Web Token) токен, який є унікальним для кожної сесії. Під час процесу авторизації користувач вводить свої облікові дані, такі як електронну адресу та пароль, які передаються на сервер для перевірки. Якщо облікові дані правильні, сервер генерує JWT токен, що містить закодовану інформацію про користувача і строк дії токену. Цей токен відправляється назад на клієнтську сторону і зберігається там, зазвичай у локальному сховищі (local storage) або у куках (cookies). Кожен наступний запит до захищених ендпоінтів API повинен включати цей токен у заголовку авторизації (Authorization header). Сервер перевіряє наявність токену, його дійсність та

автентичність. Якщо токен валідний, сервер обробляє запит та надає доступ до запитаної інформації. У випадку, якщо токен відсутній або недійсний, доступ до захищених ресурсів буде відмовлено. Це забезпечує, що тільки авторизовані користувачі можуть отримати доступ до конфіденційної інформації та функціоналу системи, а сторонні особи без токену не мають можливості взаємодіяти із захищеними ендпоінтами.

Масштабованість також є важливою вимогою. API системи «Pure Survey» повинно бути побудоване на гнучкій архітектурі, яка легко масштабується для обробки зростаючої кількості користувачів, опитувань та даних. Ця архітектура ґрунтується на принципах модульності, відділення компонентів та використання API.

Модульність означає, що система складається з окремих модулів, які можна легко додавати, видаляти або замінювати. Це робить систему більш гнучкою та адаптивною до мінливих потреб.

Відділення компонентів гарантує, що різні компоненти системи, такі як база даних, сервери опитувань та інтерфейс користувача, працюють автономно. Це забезпечує кращу продуктивність та стійкість до збоїв.

Нарешті, API системи повинно дотримуватися стандартів та найкращих практик розробки RESTful API [4]. Це включає чітке визначення ресурсів, використання правильних HTTP методів, таких як GET, POST, PUT, DELETE, та належне оброблення помилок. Крім того, необхідно забезпечити сумісність з різними клієнтами та платформами, що будуть використовувати API. Це дозволить легко інтегрувати API з різноманітними веб-додатками, мобільними додатками та іншими системами, забезпечуючи широке використання та гнучкість системи опитувань.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕЧННЯ

3.1 UML проектування ПЗ

Система матиме два типи користувачів: звичайний користувач (User) та адміністратор (Admin). Певні функції та можливості API будуть доступні лише для адміністратора, в той час як інші будуть доступні лише для звичайних користувачів. Це розмежування необхідне для забезпечення належного рівня безпеки та контролю над системою.

Для наочного відображення функціональних можливостей кожного типу користувача були побудовані діаграми прецедентів (Use Case діаграми) за методологією UML. На рисунку 3.1 зображена діаграма прецедентів для звичайного користувача.

Як можна побачити на діаграмі, звичайний користувач матиме змогу взаємодіяти з різними ендпоінтами API для виконання широкого спектру дій, пов'язаних з опитуваннями, групами опитувань, таргетингом, шаблонами зовнішнього вигляду, підписками та статистикою.

Детальніше про кожну можливість:

- реєстрація нового облікового запису: користувач зможе створити новий обліковий запис, надавши свою електронну адресу, пароль та ім'я й прізвище;
- вхід в існуючий обліковий запис: користувач зможе увійти в свій обліковий запис, використовуючи електронну пошту та пароль;
- отримання інформації про опитування: користувач зможе переглянути детальну інформацію про конкретне опитування, надавши його ідентифікатор (ID);
- перегляд списку власних опитувань: користувач зможе отримати список усіх опитувань, які він створив;

- створення нового опитування: користувач зможе створити нове опитування, додаючи питання, варіанти відповідей та інші необхідні налаштування;
- видалення опитування: користувач зможе видалити одне з власних опитувань;
- редагування опитування: користувач зможе внести зміни до існуючого опитування, включаючи таргетинг, зміст питань, варіанти відповідей, порядок питань тощо;
- перегляд шаблонів системи: користувач зможе отримати список усіх доступних шаблонів системи (Templates);
- перегляд створених зовнішніх виглядів: користувач зможе отримати список усіх створених ним зовнішніх виглядів груп опитувань (Unit Appearance);
- перегляд деталей зовнішнього вигляду: користувач зможе переглянути детальну інформацію про конкретний створений зовнішній вигляд групи;
- створення шаблону зовнішнього вигляду: користувач зможе створити новий шаблон зовнішнього вигляду для груп опитувань;
- редагування зовнішнього вигляду: користувач зможе внести зміни до існуючого зовнішнього вигляду групи опитувань;
- видалення зовнішнього вигляду: користувач зможе видалити створений ним зовнішній вигляд групи;
- отримання інформації про групу: користувач зможе отримати детальну інформацію про конкретну групу опитувань за її ідентифікатором (ID);
- перегляд списку власних груп: користувач зможе отримати список усіх груп опитувань, які він створив;
- створення нової групи: користувач зможе створити нову групу опитувань, налаштувавши необхідні параметри та зовнішній вигляд;
- видалення групи: користувач зможе видалити одну з власних груп опитувань;

- редагування групи: користувач зможе внести зміни до існуючої групи опитувань, включаючи налаштування проходження групи, редагування зовнішнього вигляду тощо;
- створення таргетингу: користувач зможе створити цільову аудиторію (таргетинг) для опитування;
- оновлення таргетингу: користувач зможе оновити існуючий таргетинг, змінивши, наприклад, країни, на які спрямоване опитування;
- перегляд списку створених таргетингів: користувач зможе отримати список усіх створених ним таргетингів;
- перегляд деталей таргетингу: користувач зможе переглянути детальну інформацію про певний таргетинг за його ідентифікатором (ID);
- отримання списку доступних країн: користувач зможе отримати перелік країн, доступних для використання в таргетингу;
- перегляд країн в таргетингу: користувач зможе отримати список країн, включених до певного створеного ним таргетингу;
- створення підписки: користувач зможе створити нову підписку за допомогою платіжної системи PayPal;
- активація підписки: користувач зможе активувати підписку, перейшовши за посиланням на систему PayPal;
- збереження інформації про підписку: система зможе зберігати інформацію про підписку за допомогою вебхуків PayPal;
- зміна статусу підписки: система зможе змінювати статус підписки (активна, неактивна тощо) за допомогою вебхуків PayPal;
- отримання статистики: користувач зможе отримати статистичні дані для кожного питання в опитуванні.

Таким чином, система надасть звичайним користувачам повний набір функціональних можливостей для створення, редагування, видалення та керування опитуваннями, групами опитувань, таргетингом, шаблонами зовнішнього вигляду,

а також для роботи з підписками та отримання статистичних даних. Кожна з цих функцій буде доступна через відповідні ендпоінти API.

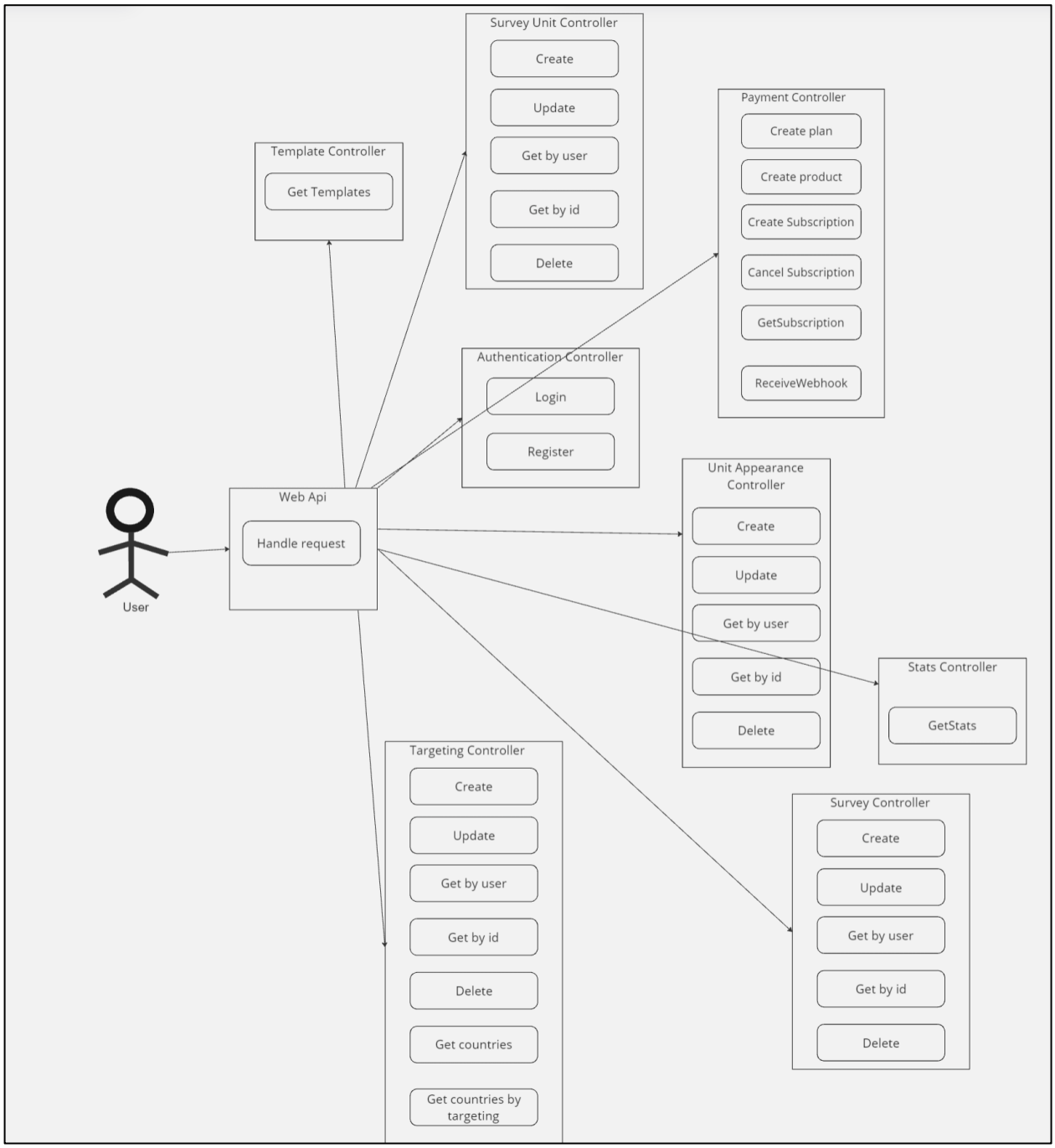


Рисунок 3.1 – Use Case діаграма для звичайного користувача

На рисунку 3.2 зображено діаграму для адміністратора системи

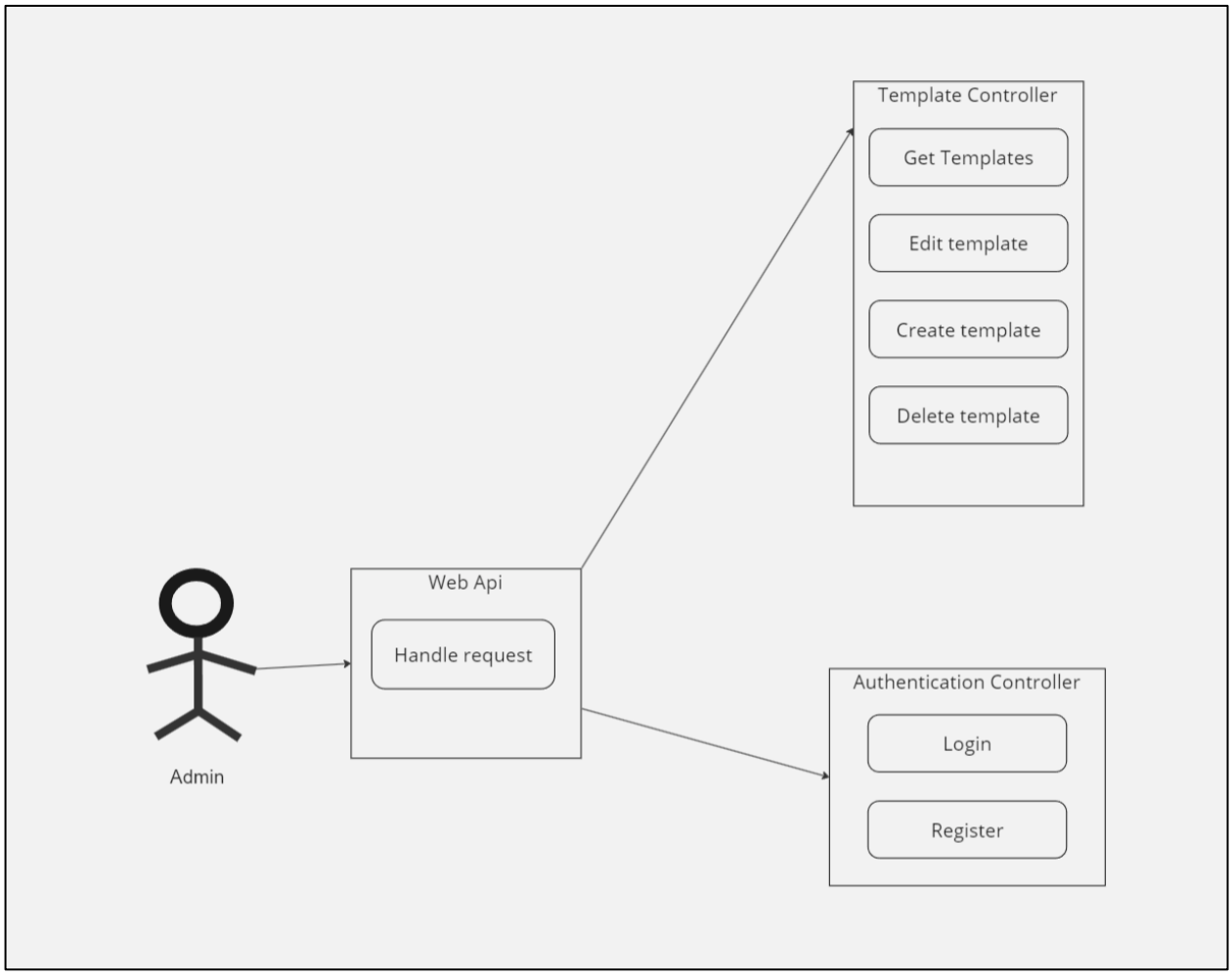


Рисунок 3.2 - Use Case діаграма для адміністратора

Діаграма показує, що основні функції, які доступні адміністратору – це логін, реєстрація та дії з шаблонами опитувань: створення, видалення, редагування та отримання темплейтів.

3.3 Проектування архітектури ПЗ

API частина системи буде реалізована за допомогою фреймворка ASP.NET Core Web API, з використанням чистої архітектури. ASP.NET Core Web API - це потужний фреймворк для розробки веб-додатків на мові програмування C# з використанням платформи .NET Core. Він забезпечує широкі можливості для створення високопродуктивних, масштабованих та безпечних веб-API, які можуть бути розгорнуті на різних платформах та використовуватися різними клієнтами. Основні переваги ASP.NET Core Web API полягають у його відкритості, підтримці

крос-платформеності, високій продуктивності, а також легкій інтеграції з іншими платформами та сервісами, такими як Entity Framework Core для роботи з базами даних.

Web API є способом побудови програми ASP.NET, який спеціально оптимізований для роботи в стилі REST (Representation State Transfer або "передача стану подання"). REST-архітектура передбачає використання стандартних HTTP-методів або типів запитів (GET, POST, PUT, DELETE тощо) для взаємодії із сервером, забезпечуючи тим самим уніфікований інтерфейс для різних клієнтів.

Для взаємодії з базою даних у цьому проекті було використано технологію Entity Framework Core [5]. Entity Framework Core (EF Core) є об'єктно-орієнтованою, легковажною та розширюваною технологією від компанії Microsoft для доступу до даних. EF Core є ORM-інструментом (object-relational mapping – відображення даних на реальні об'єкти), що дозволяє працювати з базами даних на більш високому рівні абстракції, використовуючи об'єктно-орієнтовані конструкції та концепції замість низькорівневих SQL-запитів.

Крім того, я також використовував створену базу даних Vertica для зберігання статистичних даних з опитувань. Доступ до цієї бази даних я отримував за допомогою технології ADO.NET, яка забезпечує можливість взаємодії з різними джерелами даних, включно з реляційними базами даних.

Рішення було розбито на шари відповідно до принципів чистої архітектури. Було створено шість основних проектів: API (api interface), Core (core layer), Application (business logic) та 3 проекти шару доступу до даних: Infrastructure, Infrastructure.PayPal та Infrastructure.Stats.

Проект API відповідає за конфігурацію Web API, налаштування контролерів та визначення ендпойнтів (точок входу) для взаємодії з веб-додатком. Цей проект є зовнішнім інтерфейсом для взаємодії веб-клієнтів із серверною частиною додатку.

Проект Core (Domain) є доменним рівнем у чистій архітектурі. Він містить основні сутності програми, їхні специфікації та правила бізнес-логіки. Цей рівень розташований у центрі архітектури і містить такі ключові елементи, як моделі бази

даних, що використовуються Entity Framework для створення віртуальної бази даних, базові моделі програми, визначення помилок та методи розширення.

Проект Application є серцем бізнес-логіки програмного додатку. На цьому рівні реалізована вся основна функціональність програми. Тут знаходяться сервіси, які інкапсулюють базову доменну логіку та надають інтерфейси для взаємодії з інфраструктурним рівнем, зокрема для отримання даних. Саме на рівні Application визначаються та реалізуються операції, необхідні для обробки даних, виконання бізнес-правил та забезпечення функціональності програмного продукту. Також цей рівень відповідає за виконання валідації даних та взаємодії з іншими внутрішніми інтерфейсами додатку. У проекті Application реалізовано всі основні сервіси для взаємодії з основною логікою додатку, а також сервіс для роботи з підписками та сервіс для роботи зі статистикою, які використовують відповідні репозиторії з рівня Infrastructure.

Проект Infrastructure відповідає за інфраструктуру програмного додатку, зокрема за збереження та обробку даних. Основна функція цього рівня – забезпечити доступ до бази даних. Тут знаходяться міграції бази даних для збереження структури даних відповідно до моделей програмного додатку. Крім того, на цьому рівні здійснюється конфігурація моделей бази даних і контексту бази даних для забезпечення зручного підключення та взаємодії з даними. Також тут можна знайти реалізацію інтерфейсів репозиторіїв, які дозволяють взаємодіяти з базою даних та використовуються на рівні Application.

Проект Infrastructure.PayPal містить реалізацію репозиторіїв для взаємодії з системою PayPal. У цьому проекті міститься необхідна логіка для інтеграції з платіжною системою PayPal [6], обробки платежів та підписок, що дозволяє забезпечити функціональність оплати в додатку.

Проект Infrastructure.Stats відповідає за взаємодію з базою даних, яка зберігає статистику додатку. Тут реалізовані репозиторії для зчитування статистичних даних опитувань, а також допоміжні класи, які оброблюють ці дані.

Зв'язок компонентів API можна побачити на рис. 3.3.

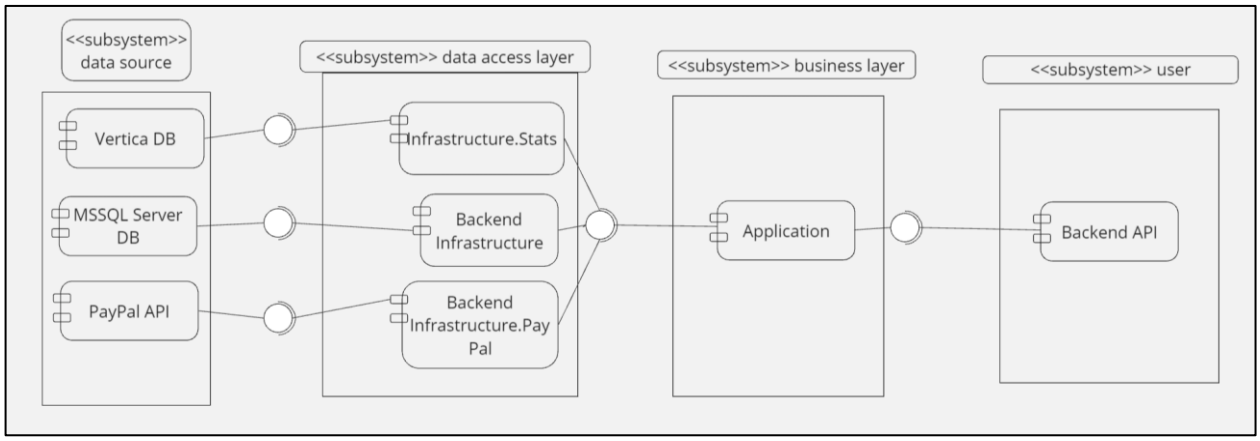


Рисунок 3.3 – Діаграма компонентів API-частини системи

Щоб зобразити взаємодію API-частини з іншими компонентами було розроблено діаграму розготрання (див. рис. 3.4).

Розглядаючи діаграму, ми бачимо, що взаємодія з веб-додатком відіграє ключову роль у цій архітектурі. Користувач звертається до веб-додатку, який через HTTP-запити взаємодіє з Web API. Це дозволяє клієнтському застосунку отримувати доступ до функціональності, реалізованої на сервері.

При обробці запитів, Web API використовує свої контролери для маршрутизації та обробки вхідних даних. Ці контролери, у свою чергу, взаємодіють із різноманітними сервісами, що представляють логіку рівня Application. Це надає гнучкість і модульність архітектури, дозволяючи легко розширювати та оновлювати функціональність системи.

Одним із важливих компонентів інфраструктури є проєкт, який взаємодіє з базою даних Microsoft SQL Server. Цей компонент відповідає за збереження та управління основними даними системи, забезпечуючи їх стабільність і доступність.

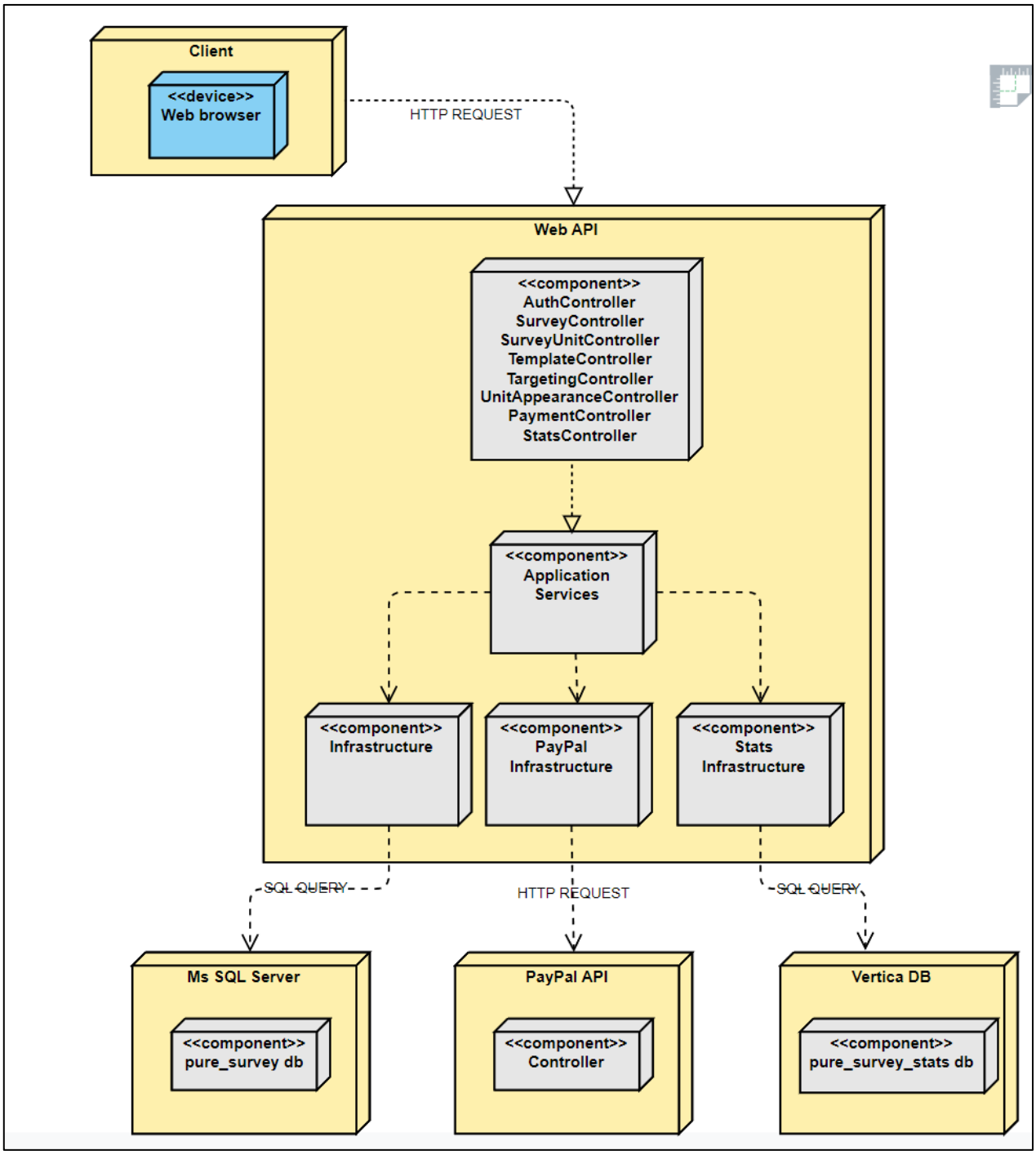


Рисунок 3.4 – Діаграма розготрання API-частини системи

Крім того, архітектура включає дві ключові інфраструктури: PayPal Infrastructure та Stats Infrastructure. PayPal Infrastructure взаємодіє з PayPal API через HTTP-запити, забезпечуючи обробку платежів та інших фінансових операцій. Stats Infrastructure, своєю чергою, взаємодіє з Vertica API через SQL-запити, дозволяючи отримувати статистичні дані про опитування та аналізувати ефективність системи.

Конфігурація системи береться з файлу appsettings.json, що є стандартним підходом у середовищі ASP.NET Core для управління налаштуваннями додатку. Використання цього файлу дозволяє зручно зберігати конфігураційні параметри у форматі JSON, що робить їх легко читабельними та редагованими. Однією з основних переваг цього підходу є можливість централізованого зберігання всіх налаштувань, що спрощує їх зміну та управління. Крім того, ASP.NET Core підтримує багато середовищ (наприклад, Development, Staging, Production), і для кожного з них можна створити свій власний конфігураційний файл (наприклад, appsettings.Development.json), що дозволяє налаштовувати додаток відповідно до специфічних потреб кожного середовища. Це забезпечує гнучкість і зручність у налаштуванні та розгортанні системи, робить код менш залежним від жорстко закріплених налаштувань і дозволяє легко змінювати параметри без необхідності перекомпіляції додатку. Додатково, appsettings.json може бути поєднаний з іншими джерелами конфігурації, такими як змінні середовища чи секрети, що підвищує безпеку та адаптивність додатку.

При розробці системи я притримався принципів SOLID, що дозволило створити гнучкий та легко підтримуваний код. Принцип єдиного обов'язку (Single Responsibility Principle) проявляється у тому, що кожен клас у системі має лише одну відповідальність. Наприклад, контролери відповідають лише за обробку HTTP-запитів, тоді як сервіси відповідають за бізнес-логіку. Принцип відкритості/закритості (Open/Closed Principle) було реалізовано шляхом створення абстракцій та інтерфейсів, що дозволяють розширювати функціональність без зміни існуючого коду. Принцип підстановки Лісков (Liskov Substitution Principle) дотримувався шляхом використання інтерфейсів, які дозволяють замінювати об'єкти їхніми підтипами без порушення роботи системи. Принцип сегрегації інтерфейсів (Interface Segregation Principle) забезпечується створенням невеликих, специфічних інтерфейсів замість великих, монолітних, що дозволяє клієнтам залежати лише від необхідних методів. Принцип інверсії залежностей (Dependency Inversion Principle) реалізований через використання впровадження залежностей

(Dependency Injection), що дозволяє абстрагувати високорівневі модулі від низькорівневих.

Крім того, було дотримано усіх принципів чистого коду, що зробило код більш зрозумілим, читабельним і легким у підтримці. Принцип зрозумілих імен (Meaningful Names) був реалізований шляхом використання чітких та описових назв для змінних, методів і класів, що відображають їх функціональність. Принцип DRY (Don't Repeat Yourself) дотримувався шляхом уникнення дублювання коду і створення багаторазових компонентів. Код був розбитий на невеликі методи, кожен з яких виконує одну чітко визначену задачу, що відповідає принципу KISS (Keep It Simple, Stupid) і допомагає уникнути складності. Я також дотримувався принципу SRP (Single Responsibility Principle) у рамках чистого коду, забезпечуючи, щоб кожен клас і метод мали лише одну відповідальність. Принципи форматування та стильового оформлення коду були реалізовані шляхом використання єдиного стилю кодування, що сприяє його легкому читанню і підтримці. Крім того, було забезпечено якісне покриття тестами, що дозволяє легко виявляти та виправляти помилки на ранніх стадіях розробки.

3.3 Проектування структури зберігання даних

У якості системи керування базами даних (СКБД) буде використовуватись Microsoft SQL Server. Ця технологія була обрана через низку вагомих переваг, які роблять її оптимальним рішенням для нашої системи. MS SQL Server підтримує мови запитів T-SQL (Transact-SQL), ANSI SQL та багато інших мов програмування, що забезпечує гнучкість і зручність у розробці та управлінні базами даних. Це дозволяє нам створювати складні запити та процедури, що оптимізують роботу з даними.

MS SQL Server працює на Windows-платформі, що забезпечує високу сумісність з існуючими системами і сервісами. Це дає можливість легко інтегрувати базу даних у наявну IT-інфраструктуру, використовуючи знайомі інструменти та технології. Додатковою перевагою є підтримка широких можливостей з безпеки, включаючи шифрування даних, аудит та контроль доступу

на основі ролей. Це забезпечує захист конфіденційних даних та відповідність стандартам безпеки.

MS SQL Server відзначається високою масштабованістю, що дозволяє системі зростати разом з потребами бізнесу. Це важливо для нашого проекту, оскільки обсяг даних і кількість користувачів можуть значно збільшуватись з часом. MS SQL Server підтримує функції кластеризації та реплікації, що забезпечує високу доступність та відмовостійкість бази даних. Це означає, що система зможе працювати безперервно навіть у випадку збою окремих компонентів.

Крім того, MS SQL Server має низку додаткових інструментів і сервісів, які значно полегшують управління базами даних і розширюють їх функціональність. SQL Server Management Studio (SSMS) надає зручний графічний інтерфейс для адміністрування баз даних, створення запитів і аналізу продуктивності. SQL Server Integration Services (SSIS) дозволяє ефективно здійснювати імпорт, експорт та трансформацію даних, що особливо корисно при інтеграції з іншими системами. SQL Server Reporting Services (SSRS) забезпечує створення та розповсюдження звітів, що дозволяє користувачам отримувати аналітичну інформацію у зручному форматі. На рисунку 3.5 зображено логічну схему БД, яка описує сутності у базі даних.

Як можна побачити з логічної моделі бази даних, система включає кілька ключових сутностей, кожна з яких має власні атрибути та зв'язки з іншими сутностями, що дозволяє організувати дані у зручний та логічний спосіб. Нижче наведено детальний опис кожної сутності та їх взаємозв'язків.

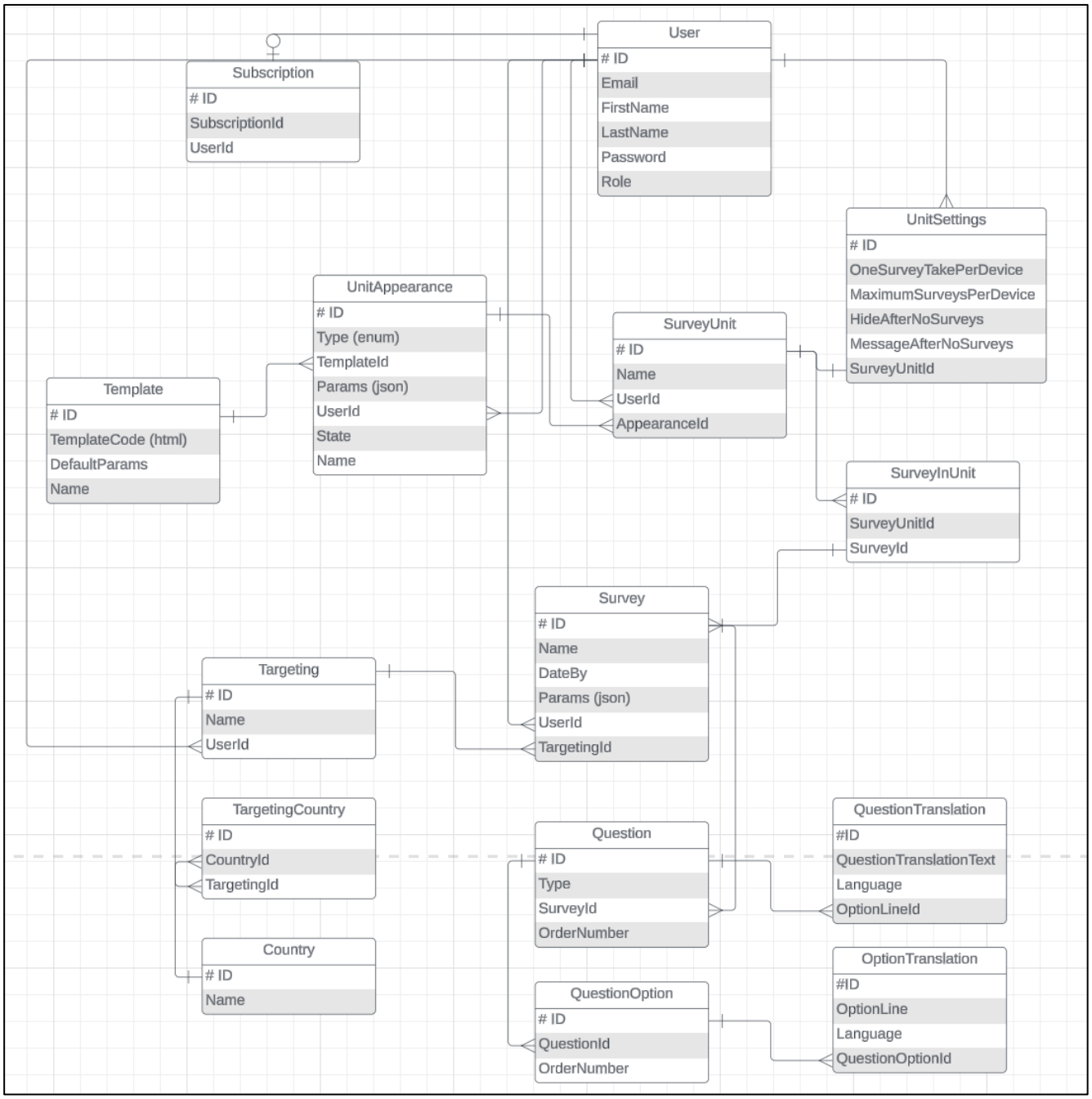


Рисунок 3.5 – Логічна модель бази даних

Сутність User (Користувач) є основною і містить інформацію про користувачів системи. Кожен користувач має унікальний ідентифікатор (ID), електронну пошту (Email), ім'я (FirstName), прізвище (LastName), пароль (Password) та роль (Role), яка визначає права доступу користувача в системі (наприклад, адміністратор або звичайний користувач).

Сутність Subscription (Підписка) зберігає інформацію про підписки користувачів. Кожна підписка має унікальний ідентифікатор (ID), назву (Name),

дату створення (DateCreate), прізвище користувача, пов'язане з підпискою (LastName), статус активності (IsActive) та ідентифікатор користувача (UserId), який володіє підпискою. Це дозволяє відстежувати, які користувачі мають активні підписки та управляти їхнім станом. Зв'язок між користувачем і підпискою є таким, що користувач може мати лише одну підписку або взагалі не мати жодної, в той час як підписка не може існувати без прив'язки до користувача.

Сутність Template (Шаблон) визначає шаблони, що використовуються в системі. Вона містить унікальний ідентифікатор (ID), код шаблону у форматі HTML (TemplateCode), параметри за замовчуванням (DefaultParams) та назву шаблону (Name). Шаблони забезпечують гнучкість у створенні різноманітних опитувань з різними форматами та параметрами.

Сутність UnitAppearance (Зовнішній вигляд блоку) відповідає за візуальне представлення одиниць опитування. Кожен запис має унікальний ідентифікатор (ID), тип (Type), ідентифікатор шаблону (TemplateId), параметри у форматі JSON (Params), ідентифікатор користувача (UserId), стан (State) та назву (Name). Це дозволяє користувачам налаштовувати зовнішній вигляд своїх опитувань відповідно до своїх потреб. Зв'язок між користувачем та зовнішнім виглядом блоку є таким, що кожен користувач може створювати кілька записів зовнішнього вигляду блоку, але кожен запис зовнішнього вигляду повинен бути прив'язаний до конкретного користувача.

Сутність SurveyUnit (Одиниця опитування) об'єднує інформацію про конкретні одиниці опитування. Вона містить унікальний ідентифікатор (ID), назву (Name), ідентифікатор користувача (UserId), який створив опитування, та ідентифікатор зовнішнього вигляду (AppearanceId). Ця сутність дозволяє створювати та організовувати різні опитування для користувачів. Зв'язок між користувачем та одиницею опитування є таким, що кожен користувач може створити кілька одиниць опитування, але кожна одиниця опитування повинна бути прив'язана до конкретного користувача.

Сутність UnitSettings (Налаштування блоку) визначає налаштування для одиниць опитування. Вона включає унікальний ідентифікатор (ID), параметри, такі

як один опитування на пристрій (OneSurveyTakePerDevice), максимальна кількість опитувань на пристрій (MaximumSurveysPerDevice), приховування після закінчення опитувань (HideAfterNoSurveys) та повідомлення після закінчення опитувань (MessageAfterNoSurveys). Ці налаштування дозволяють гнучко керувати поведінкою опитувань у системі. Кожен запис налаштувань блоку повинен бути прив'язаний до конкретної одиниці опитування, і одна одиниця опитування може мати лише один набір налаштувань.

Сутність Survey (Опитування) містить інформацію про конкретні опитування. Вона включає унікальний ідентифікатор (ID), назву (Name), дату завершення (DateBy), параметри у форматі JSON (Params), ідентифікатор користувача (UserId), який створив опитування, та ідентифікатор цільової групи (TargetingId). Це дозволяє створювати та керувати різними опитуваннями в системі. Зв'язок між користувачем та опитуванням є таким, що кожен користувач може створювати кілька опитувань, але кожне опитування повинно бути прив'язане до конкретного користувача.

Сутність Targeting (Цільова група) відповідає за визначення цільових груп для опитувань. Вона містить унікальний ідентифікатор (ID), назву (Name) та ідентифікатор користувача (UserId), який створив цільову групу. Це дозволяє направляти опитування на конкретні групи користувачів. Кожен користувач може створювати кілька цільових груп, але кожна цільова група повинна бути прив'язана до конкретного користувача.

Сутність TargetingCountry (Країна цільової групи) використовується для визначення країн, до яких належить цільова група. Вона містить унікальний ідентифікатор (ID), ідентифікатор країни (CountryId) та ідентифікатор цільової групи (TargetingId). Це дозволяє налаштовувати опитування для конкретних географічних регіонів. Кожна цільова група може бути прив'язана до кількох країн, але кожен запис у цій сутності повинен бути прив'язаний до конкретної цільової групи.

Сутність Country (Країна) містить інформацію про країни, включаючи унікальний ідентифікатор (ID) та назву (Name). Вона організовує опитування за

географічними ознаками. Кожен запис унікальний для конкретної країни та може бути прив'язаний до кількох цільових груп.

Сутність Question (Питання) зберігає інформацію про питання в опитуваннях. Вона включає унікальний ідентифікатор (ID), тип питання (Type), ідентифікатор опитування (SurveyId) та порядковий номер питання (OrderNumber). Це дозволяє створювати структуровані опитування з різними типами питань. Кожне опитування може містити кілька питань, прив'язаних до конкретного опитування.

Сутність QuestionOption (Варіант відповіді) містить інформацію про варіанти відповідей. Вона включає унікальний ідентифікатор (ID), ідентифікатор питання (QuestionId) та порядковий номер варіанту (OrderNumber). Це дозволяє створювати багатоваріантні питання. Кожне питання може мати кілька варіантів відповідей, прив'язаних до конкретного питання.

Сутність QuestionTranslation (Переклад питання) зберігає переклади питань на різні мови. Вона містить унікальний ідентифікатор (ID), текст перекладу (QuestionTranslationText), мову (Language) та ідентифікатор варіанту (OptionLineId). Це забезпечує мультимовну підтримку опитувань. Кожен переклад прив'язаний до конкретного варіанту та мови.

Сутність OptionTranslation (Переклад варіанту відповіді) зберігає переклади варіантів на різні мови. Вона включає унікальний ідентифікатор (ID), текст перекладу (OptionLine), мову (Language) та ідентифікатор варіанту (QuestionOptionId). Це дозволяє користувачам відповідати на питання рідною мовою. Кожен переклад прив'язаний до конкретного варіанту та мови.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Гнучкість та модульність системи

Предметна область, в межах якої розробляється система для проведення опитувань, не передбачає складних математичних алгоритмів та аналізу даних. Але під час створення програмної системи було прийнято багато програмних рішень щодо розділення системи на модулі відповідно до принципу єдиної відповідальності (SRP) та поліпшення гнучкості системи через використання абстракцій. На прикладі логіки оплати підписки можна побачити, як ці рішення допомагають досягти чистої архітектури.

Принцип єдиної відповідальності передбачає, що кожен клас або модуль має відповідати лише за одну функціональну область або частину бізнес-логіки. У нашому рішенні цей принцип реалізований через використання інтерфейсів та окремих класів для різних аспектів логіки платежів. Наприклад, клас PaymentService, який реалізує різноманітні методи для управління підписками та платежами, використовує два інтерфейси: IPaymentProxy для інтеграції з PayPal та IPaymentRepository для взаємодії з базою даних.

```
public class PaymentService : IPaymentService
{
    private readonly IPaymentProxy _payPalProxy;
    private readonly IPaymentRepository _paymentRepository;

    public PaymentService(IPaymentProxy paymentProxy,
    IPaymentRepository paymentRepository)
    {
        _payPalProxy = paymentProxy;
        _paymentRepository = paymentRepository;
    }

    // Методи для роботи з підписками та платежами...
}
```

Це рішення забезпечує гнучкість, оскільки дозволяє легко замінити реалізацію IPaymentProxy або IPaymentRepository без зміни логіки в PaymentService. Наприклад, якщо з’явиться потреба змінити платіжну систему, заміна IPaymentProxy на іншу реалізацію не вплине на основну логіку PaymentService.

Рівень інфраструктури також розділений на окремі компоненти. PayPalProxy реалізує інтеграцію з PayPal через інтерфейс IPaymentProxy, використовуючи IPayPalFacade для відокремлення логіки формування запитів до PayPal API.

```
public class PayPalProxy : IPaymentProxy
{
    private readonly IPayPalFacade _payPalFacade;
    private readonly PayPalConfig _configuration;

    public PayPalProxy(IConfiguration configuration, IPayPalFacade
    payPalFacade)
    {
        _configuration = new PayPalConfig(configuration);
        _payPalFacade = payPalFacade;
    }

    // Методи для роботи з PayPal API...
}
```

Це рішення додає рівень абстракції, що дозволяє відокремити логіку взаємодії з PayPal API від бізнес-логіки. Проксі використовує IPayPalFacade для формування HTTP-запитів до PayPal API, роблячи PayPalProxy більш чистим та зосередженим лише на інтеграції з PayPal.

```
public class PayPalFacade : IPayPalFacade
{
    private readonly IRestClient restClient;

    public PayPalFacade(IConfiguration provideConfiguration,
    IRestClient restClient)
    {
        this.restClient = restClient;
        var configuration = new PayPalConfig(provideConfiguration);
        restClient.Timeout = configuration.TimeOut;
        restClient.BaseUrl = new Uri(configuration.BaseUrl);
    }

    // Методи для виконання HTTP-запитів до PayPal API...
}
```

Використання інтерфейсів у нашому рішенні для проведення опитувань приносить численні переваги, які роблять систему більш гнучкою, модульною, тестованою та підтримуваною.

Чітке розділення відповідальності на рівні бізнес-логіки (PaymentService), інтеграції (PayPalProxy) та формування запитів (PayPalFacade) дозволяє легко

замінювати реалізації без значних змін у кодовій базі. Це робить систему більш гнучкою та адаптивною до змін у вимогах або технологіях.

Виділення логіки на різні рівні сприяє більшій модульності коду. Це робить його більш організованим, зрозумілим та легким для розширення. Кожен компонент відповідає за свою специфічну функцію, що полегшує додавання нових функціональних можливостей.

Чітке розділення відповідальності та модульний дизайн роблять код більш тестованим. Кожен компонент можна тестувати окремо, що полегшує виявлення та виправлення помилок.

4.2 Безпека та контроль доступу

4.2.1 Генерація та використання JWT токенів

У нашій системі реалізовано генерацію та валідацію JWT токенів, що забезпечує безпеку та контроль доступу користувачів. Використання JWT токенів дозволяє ефективно управляти автентифікацією та авторизацією, забезпечуючи високий рівень захисту даних. Генерація та валідація токенів є невід'ємною частиною системи, яка відповідає за ідентифікацію користувачів і контроль доступу до ресурсів.

Для генерації JWT токенів використовується клас UserExtensions з методом GenerateTokenFromUser. Цей метод створює токен на основі даних користувача та параметрів конфігурації.

```
public static class UserExtensions
{
    public static string GenerateTokenFromUser(this User user, string
jwtKey, string issuer, string audience)
    {
        var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtKey));
        var credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);

        var claims = new[]
        {
            new Claim(ClaimTypes.Email, user.Email),
            new Claim(ClaimTypes.GivenName, user.FirstName),
            new Claim(ClaimTypes.Surname, user.LastName),
```

```
            new Claim(ClaimTypes.Role, user.Role.ToString()),
            new Claim(ClaimTypes.Sid, user.Id.ToString())
        };

        var token = new JwtSecurityToken(
            issuer,
            audience,
            claims,
            expires: DateTime.Now.AddMinutes(30000),
            signingCredentials: credentials
        );

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}
```

Метод GenerateTokenFromUser використовує симетричний ключ для підпису токена, що забезпечує безпеку даних. У токені міститься інформація про користувача, така як електронна пошта, ім'я, прізвище, роль та ідентифікатор, що дозволяє легко ідентифікувати користувача та визначити його права доступу. Крім того, токен має термін дії, що додає додатковий рівень безпеки, запобігаючи використанню старих токенів.

Для отримання даних користувача з JWT токена використовується клас CurrentUserRetriever з методом GetCurrentUser, який витягує дані користувача з токена.

Метод GetCurrentUser перевіряє наявність заявок у токені та витягує дані користувача з них. Якщо заявки невалідні або користувач не знайдений, метод повертає відповідну помилку, що запобігає несанкціонованому доступу. Включення обробки виключень робить метод стійким до непередбачених ситуацій, що підвищує загальну надійність системи. На рис. 4.1. ми можемо побачити відповідь на запит входу до системи у випадку правильно введених даних.

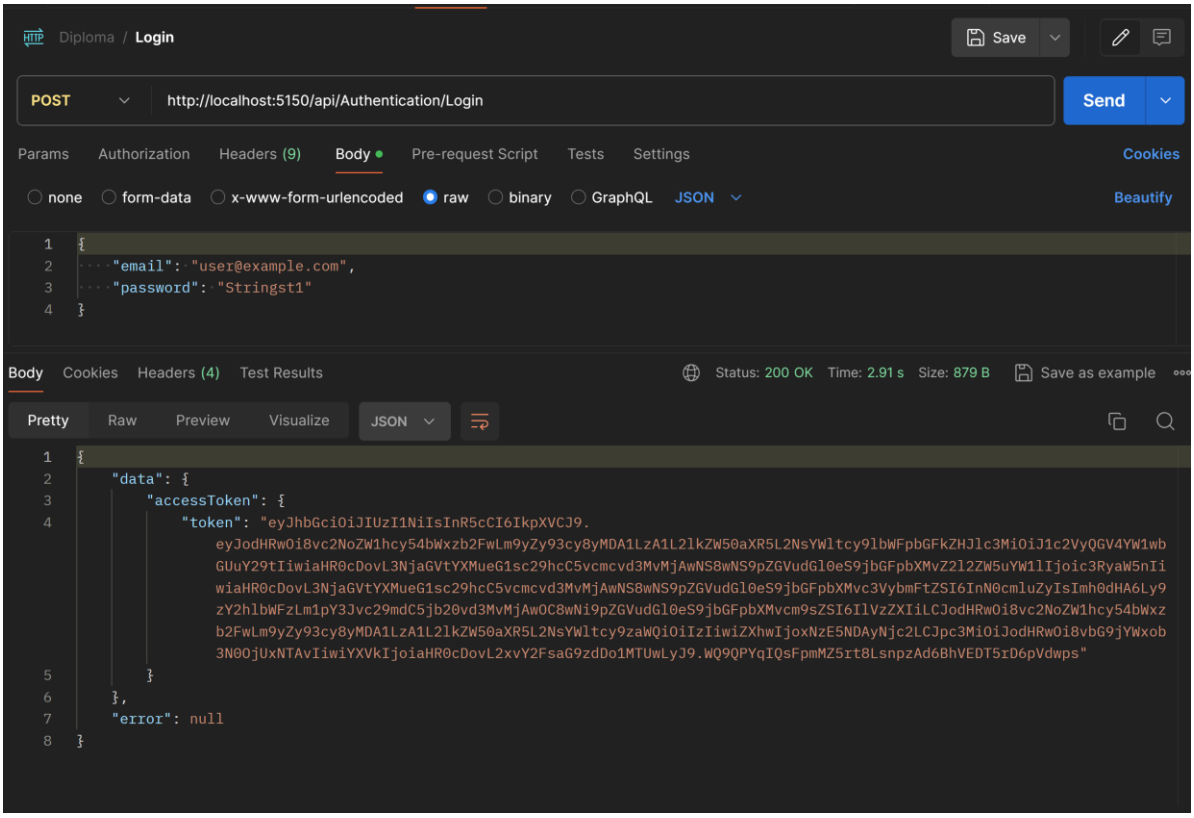


Рисунок 4.1 – Відповідь на запит входу до системи у вигляді JWT-токена

Також на рисунку 4.2 зображена відповідь у випадку неправильного або простроченого токена доступу

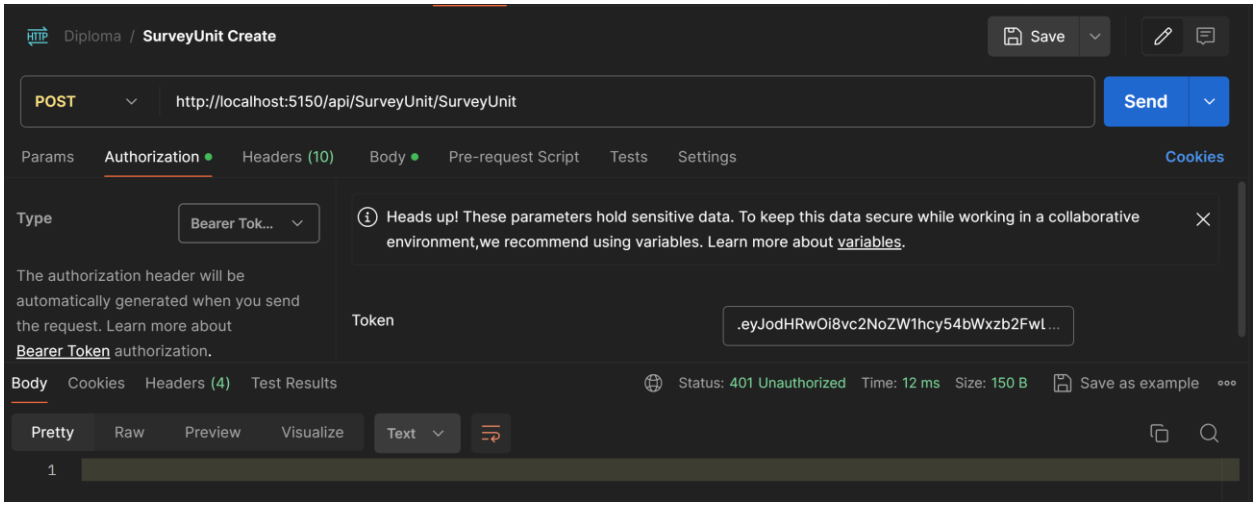


Рисунок 4.2 – Відповідь на запит до API з невалідним токеном доступу

Використання JWT токенів у нашій системі має кілька значних переваг. По-перше, це забезпечує високий рівень безпеки завдяки використанню симетричного

ключа та алгоритму HMACSHA256 для підпису токенів. По-друге, токени містять детальну інформацію про користувача, що дозволяє точно ідентифікувати користувачів та керувати їхніми правами доступу. По-третє, термін дії токенів додає додатковий рівень захисту, запобігаючи використанню старих або неавторизованих токенів. Нарешті, конфігурація токенів дозволяє легко налаштовувати та підтримувати систему, роблячи її більш гнучкою та зручною в адмініструванні.

4.2.2 Хешування паролів

У нашій системі паролі користувачів зберігаються у вигляді хешів, що значно підвищує рівень безпеки та забезпечує захист даних користувачів навіть у випадку витоку інформації. Хешування паролів є важливим кроком у забезпеченні безпеки, оскільки дозволяє зберігати паролі в незворотному вигляді. Для цього використовується алгоритм Bcrypt, реалізований у методі ConvertPasswordToHash класу StringExtensions.

```
public static string ConvertPasswordToHash(this string password)
    {
        string salt = BCrypt.Net.BCrypt.GenerateSalt();
        string hashedPassword =
BCrypt.Net.BCrypt.HashPassword(password, salt);
        return hashedPassword;
    }
```

Метод ConvertPasswordToHash перетворює пароль у хеш за допомогою алгоритму Bcrypt. Спочатку генерується соляний рядок (salt), який додається до пароля перед хешуванням. Потім створюється хеш паролю разом з сіллю за допомогою функції BCrypt.Net.BCrypt.HashPassword. Отримане зашифроване значення і є хешем пароля. Цей підхід забезпечує високий рівень безпеки завдяки декільком важливим аспектам.

По-перше, зберігання паролів у вигляді хешів захищає їх навіть у випадку витоку даних. Навіть якщо зловмисники отримають доступ до бази даних, вони не зможуть дізнатися оригінальні паролі. Це досягається завдяки тому, що хеш-функція є односторонньою, тобто неможливо легко відновити оригінальний пароль

з його хешу. Це означає, що хешування значно ускладнює завдання зловмисників, оскільки вони не можуть просто "розхешувати" значення, щоб отримати вихідний пароль.

По-друге, використання алгоритму Bcrypt забезпечує стійкість до атак методом перебору (brute-force). Bcrypt є адаптивною функцією хешування, яка вимагає значних обчислювальних ресурсів для генерації хешів. Це означає, що обчислення хешу займає дуже багато часу, що ускладнює злом паролів шляхом перебору всіх можливих варіантів. Завдяки цьому, навіть якщо зловмисники спробують зламати пароль за допомогою автоматизованих інструментів, процес займе надзвичайно багато часу і ресурсів.

При реєстрації нового користувача, його пароль хешується перед збереженням у базі даних. При спробі автентифікації (логіні) користувача перевіряється, чи відповідає введений пароль збереженому хешу пароля. Використовується алгоритм Bcrypt для хешування введеного пароля з тією ж сіллю, що була використана для збереженого хешу, і порівнює отриманий хеш зі збереженим.

```
private async Task<BaseResponse<User>> GetUser(LoginRequest
loginRequest)
{
User? user = await
_authRepository.GetUserByEmailAsync(loginRequest.Email);

if (user != null && BCrypt.Net.BCrypt.Verify(loginRequest.Password,
user.Password))
{
return BaseResponseGenerator.GenerateValidBaseResponse<User>(user);
}
```

Зберігання паролів у вигляді хешів захищає їх від компрометації навіть у випадку витоку даних. Хешування є одностороннім процесом, що унеможливлює легке відновлення оригінального пароля з його хешу. Використання алгоритму SHA-256 забезпечує стійкість до атак методом перебору, оскільки обчислення хешу займає значний час, що ускладнює злом паролів.

4.2.3 Захист даних і безпека запитів до бази даних

У процесі розробки додатка було прийнято рішення дотримуватись принципів безпеки щодо обробки запитів до бази даних. Це дозволяє забезпечити ефективну роботу додатка та захист від можливих атак, таких як SQL-ін'єкції.

У проекті було створено клас StatsRepository, який реалізує інтерфейс IStatsRepository. Цей клас відповідає за взаємодію з базою даних Vertica, а саме за отримання статистики по конкретному питанню. Наведений нижче код демонструє основну реалізацію цього класу:

```
public class StatsRepository : IStatsRepository
{
    private readonly string _connectionString;

    public StatsRepository(IConfiguration configuration)
    {
        _connectionString = BuildConnectionString(configuration);
    }

    public async Task<StatsForQuestion> GetStatsForQuestion(int
questionId)
    {
        using (var connection = new
VerticaConnection(_connectionString))
        {
            await connection.OpenAsync();

            using (var command = CreateCommand(connection,
QueryProvider.GetStatsForQuestionQuery, questionId))
            using (var reader = await command.ExecuteReaderAsync())
            {
                if (await reader.ReadAsync())
                {
                    return new StatsForQuestion
                    {
                        QuestionId =
reader.GetInt32(reader.GetOrdinal("question_id")),
                        Views =
reader.GetInt32(reader.GetOrdinal("views")),
                        Answered =
reader.GetInt32(reader.GetOrdinal("answered"))
                    };
                }
            }
        }

        return null;
    }

    private VerticaCommand CreateCommand(VerticaConnection
connection, string query, int questionId)
    {
        var command = connection.CreateCommand();
```

```
            command.CommandText = query;
            command.Parameters.Add(new VerticaParameter("@QuestionId",
questionId));
            return command;
        }
}
```

Одним з основних принципів, які були дотримані під час розробки, є безпека даних. SQL-ін'єкції є однією з найпоширеніших вразливостей, які можуть призвести до витоку даних або навіть знищення бази даних. У наведеному коді було прийнято певні заходи для запобігання SQL-ін'єкціям.

Перш за все, метод CreateCommand створює об'єкт VerticaCommand, який використовує параметризовані запити. Параметризовані запити автоматично обробляють значення параметрів таким чином, що вони не можуть бути використані для маніпуляції запитами SQL. Це значно знижує ризик SQL-ін'єкцій, забезпечуючи захищеність даних у базі.

Крім того, рядок підключення до бази даних зберігається у конфігураційному файлі та передається через інтерфейс IConfiguration. Це дозволяє централізовано керувати конфігурацією та уникати жорстко закодованих значень у коді, що також підвищує безпеку системи. Зберігання конфігураційних даних окремо від коду сприяє кращій управлінню ними та знижує ризик витоку інформації.

Важливим аспектом є інкапсуляція логіки доступу до бази даних у класі StatsRepository. Це забезпечує легке тестування та модифікацію логіки без впливу на інші частини додатка. Завдяки цьому підходу, зміни в логіці доступу до даних можуть бути внесені швидко і безпечно, з мінімальним впливом на загальну функціональність системи.

4.3 Редагування Групи Опитувань

Редагування групи опитувань (Survey Unit) є важливою функцією в системах управління опитуваннями, що дозволяє користувачам оновлювати налаштування та асоціації опитувань. У нашій системі реалізовано алгоритм, який забезпечує коректність та безпеку оновлення даних.

Спочатку перевіряється існування користувача, зовнішнього вигляду (Appearance) та самої групи опитувань. Якщо якийсь із цих елементів не знайдено, повертається відповідь з відповідною помилкою:

```
var user = await _repository.GetUserByIdAsync(userJwt.Id);
if (user == null)
    return
BaseResponseGenerator.GenerateBaseResponseByErrorMessage<SurveyUnitRe
sponse>(ErrorCodes.UserNotFound.ToString());

var appearance = await
_repository.GetUnitAppearanceByIdAsync(surveyUnitEditRequest.Appearan
ceId);
if (appearance == null)
    return
BaseResponseGenerator.GenerateBaseResponseByErrorMessage<SurveyUnitRe
sponse>(ErrorCodes.UnitAppearanceNotFound.ToString());

var surveys = await
_repository.GetSurveysByIdsAsync(surveyUnitEditRequest.SurveyIds);

var model = await
_repository.GetSurveyUnitByIdAsync(surveyUnitEditRequest.Id);
if (model == null)
    return
BaseResponseGenerator.GenerateBaseResponseByErrorMessage<SurveyUnitRe
sponse>(ErrorCodes.SurveyUnitNotFound.ToString());
```

Після цього оновлюються властивості групи опитувань відповідно до даних із запиту:

```
var response = await EditSurveyUnitModel(surveyUnitEditRequest,
model);
```

Далі видаляються старі опитування, які більше не належать до групи, і додаються нові опитування:

```
var requestSurveysIds = surveyUnitEditRequest.SurveyIds ?? new
List<int>();
var surveyUnitActualSurveysIds = surveyUnit.SurveyInUnits.Select(x =>
x.SurveyId).ToList();
var surveysIdsToDelete =
surveyUnitActualSurveysIds.Except(requestSurveysIds).ToList();
var surveysInUnitsToDelete = surveyUnit.SurveyInUnits.Where(x =>
surveysIdsToDelete.Contains(x.SurveyId)).ToList();
await _repository.DeleteSurveyInUnitsAsync(surveysInUnitsToDelete);
var surveyInUnitsIdsToAdd = requestSurveysIds.Where(x =>
!surveyUnitActualSurveysIds.Contains(x)).ToList();
```

```
var surveyInUnitsToAdd = surveyInUnitsIdsToAdd.Select(x => new
SurveyInUnit { SurveyId = x, SurveyUnitId = surveyUnit.Id
}).ToList();
await _repository.AddSurveyInUnitsAsync(surveyInUnitsToAdd);
```

Нарешті, зміни зберігаються до бази даних, а оновлена група опитувань перетворюється у відповідний формат для створення валідної відповіді:

```
var surveys = await
_repository.GetSurveysBySurveyUnitIdAsync(surveyUnit.Id);
return
BaseResponseGenerator.GenerateValidBaseResponse(SurveyUnitMapper.MapS
urveyUnitToResponse(model, surveys));
```

Цей алгоритм забезпечує високий рівень безпеки завдяки використанню параметризованих запитів та перевірок на існування користувача і групи опитувань, що захищає від SQL-ін'єкцій та інших типів атак. Логіка редагування інкапсульована в окремому методі, що робить код структурованим і легким для підтримки. Алгоритм враховує всі аспекти редагування, включаючи оновлення властивостей, видалення старих даних та додавання нових, що гарантує цілісність даних. Використання DTO (Data Transfer Objects) дозволяє легко розширювати функціонал без значних змін у базовому коді, а чітка структура та інкапсуляція методів забезпечують легкість тестування, що важливо для якості програмного забезпечення.

4.4 Універсальна модель відповіді на запити

У нашій системі всі контролери та сервіси повертають відповіді у вигляді універсального типу BaseResponse<T>. Це забезпечує стандартизований підхід до обробки та повернення даних, що значно спрощує роботу з відповідями у додатку. Статичний клас BaseResponseGenerator містить методи для генерації як успішних, так і помилкових відповідей.

```
public static class BaseResponseGenerator
{
    public static BaseResponse<T>
    GenerateBaseResponseByErrorMessage<T>(string errorMessage)
    {
```

```
            return new BaseResponse<T>
            {
                Error = new Error
                {
                    Message = errorMessage
                }
            };
        }

        public static BaseResponse<T> GenerateValidBaseResponse<T>(T
        value)
        {
            return new BaseResponse<T>
            {
                Data = value,
            };
        }
    }
```

Уніфікація та стандартизація дозволяють спростити обробку та перевірку даних та помилок, оскільки всі методи повертають результати у відомому форматі. Наприклад, метод оновлення опитування в контролері повертає відповідь у форматі BaseResponse<SurveyResponse>, що дозволяє легко обробляти результат на клієнтській стороні.

```
[HttpPatch]
[Route("Survey")]
[Authorize(Roles = "User")]
public async Task<ActionResult<BaseResponse<SurveyResponse>>>
Update([FromBody] SurveyEditRequest request)
{
    var identity = HttpContext.User.Identity as ClaimsIdentity;
    var user = CurrentUserRetriever.GetCurrentUser(identity);
    if (user.Error != null)
    {
        return Unauthorized(user);
    }

    var result = await _surveyService.EditSurvey(user.Data, request);
    if (result.Error != null)
    {
        return BadRequest(result);
    }

    return Ok(result);
}
```

Гнучкість та розширюваність системи досягається завдяки використанню дженериків BaseResponse<T>, що дозволяє легко розширювати систему, додаючи

нові типи відповідей без необхідності змінювати базову логіку. У сервісах використання BaseResponse<T> також забезпечує стандартне повернення результатів та спрощує обробку помилок.

```
public async Task<BaseResponse<SurveyResponse>> EditSurvey(User
userJwt, SurveyEditRequest surveyEditRequest)
{
    var user = await _surveyRepository.GetUserByIdAsync(userJwt.Id);
    if (user == null)
        return
BaseResponseGenerator.GenerateBaseResponseByErrorMessage<SurveyRespon
se>(ErrorCodes.UserNotFound.ToString());

    var survey = await
_surveyRepository.GetSurveyByIdWithDetailsAsync(surveyEditRequest.Id)
;
    if (survey == null)
        return
BaseResponseGenerator.GenerateBaseResponseByErrorMessage<SurveyRespon
se>(ErrorCodes.SurveyNotFound.ToString());

    // Оновлення властивостей опитування, видалення старих питань та
додавання нових

    var surveyResponse =
SurveyMapper.ConvertSurveyToSurveyResponse(survey);
    return
BaseResponseGenerator.GenerateValidBaseResponse(surveyResponse);
}
```

На рисунку 4.3 зображена відповідь на запит реєстрації нового користувача з правильними даними.

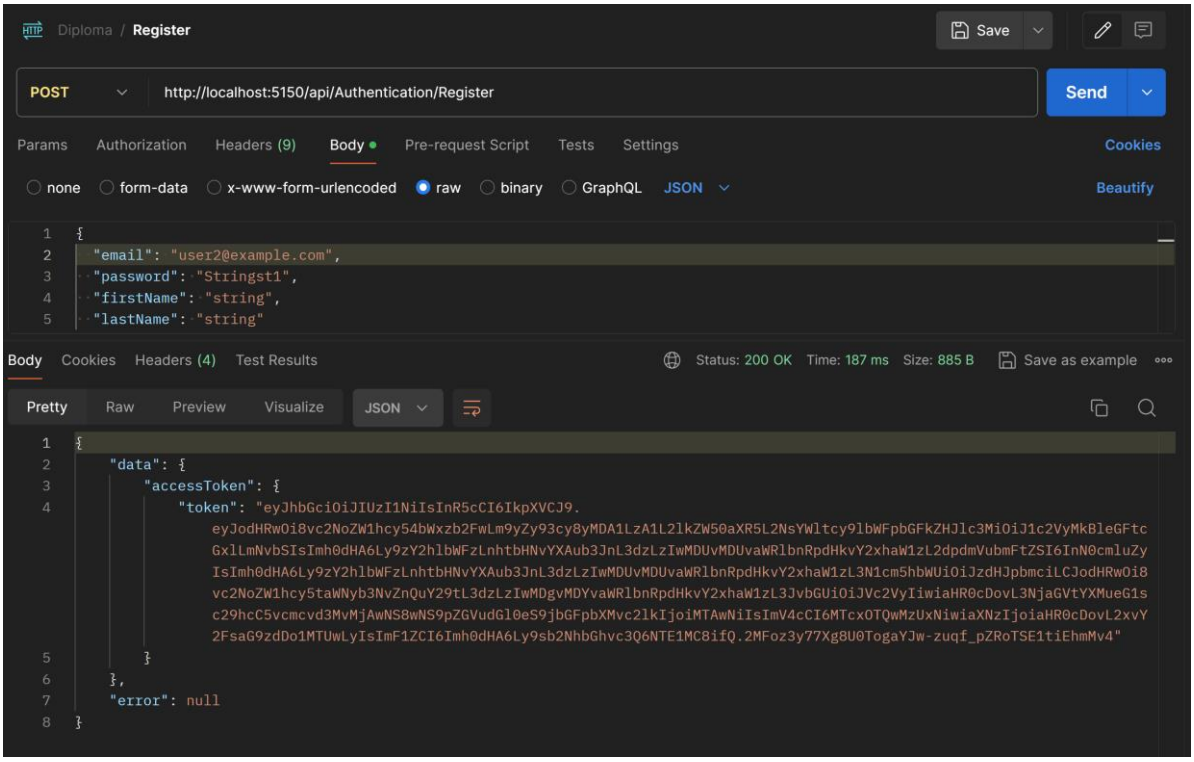


Рисунок 4.3 – Стандартизована відповідь на запит з валідними даними, що повертає дані

У свою чергу на рис. 4.4 можна побачити відповідь на запит з неправильно введеними даними, які призводять до повертання помилки у стандартизованому вигляді.

Рисунок 4.4 – Стандартизована відповідь на запит з невалідними даними, що повертає текст помилки

Зменшення дублікації коду відбувається за рахунок використання статичного генератора відповідей, що підвищує підтримуваність та чистоту коду. Підвищення якості коду забезпечується уніфікованими відповідями, що спрощують написання та підтримку тестів, оскільки всі відповіді мають однакову структуру.

4.5 Алгоритм активації підписки за допомогою PayPal

Одним із аспектів нашої системи є реалізація підписки через PayPal, яка забезпечує зручність і безпеку для користувачів. Процес оплати підписки починається з методу CreateSubscription, який приймає запит на створення підписки та перевіряє ідентифікацію користувача. Якщо користувач не автентифікований, повертається відповідний статус помилки. У випадку успішної автентифікації, метод звертається до PayPal сервісу для створення підписки.

Метод CreateSubscription перевіряє наявність активної підписки у користувача. Якщо така підписка вже існує, повертається повідомлення про помилку (рис. 4.5). Якщо підписки немає або вона не активна, система отримує токен доступу до PayPal та створює нову підписку.

```
var existingSubscription =
_paymentRepository.GetSubscriptionByUserId(jwtUser.Id);
if (existingSubscription != null && existingSubscription.IsActive)
{
return
BaseResponseGenerator.GenerateBaseResponseByErrorMessage<PayPalSubscr
iptionResponse>(ErrorCodes.SubscriptionAlreadyExists.ToString());
}

var token = await _payPalProxy.GetTokenAsync();
var subscrResponse = await CreateSubscriptionAsync(request,
token.access_token);
```

Після успішного створення підписки система зберігає її в базі даних з відповідними даними користувача та повертає дані підписки та посилання на її оплату в системі PayPal, по якому користувач переходить для завершення процесу оплати (див. рис. 4.6).

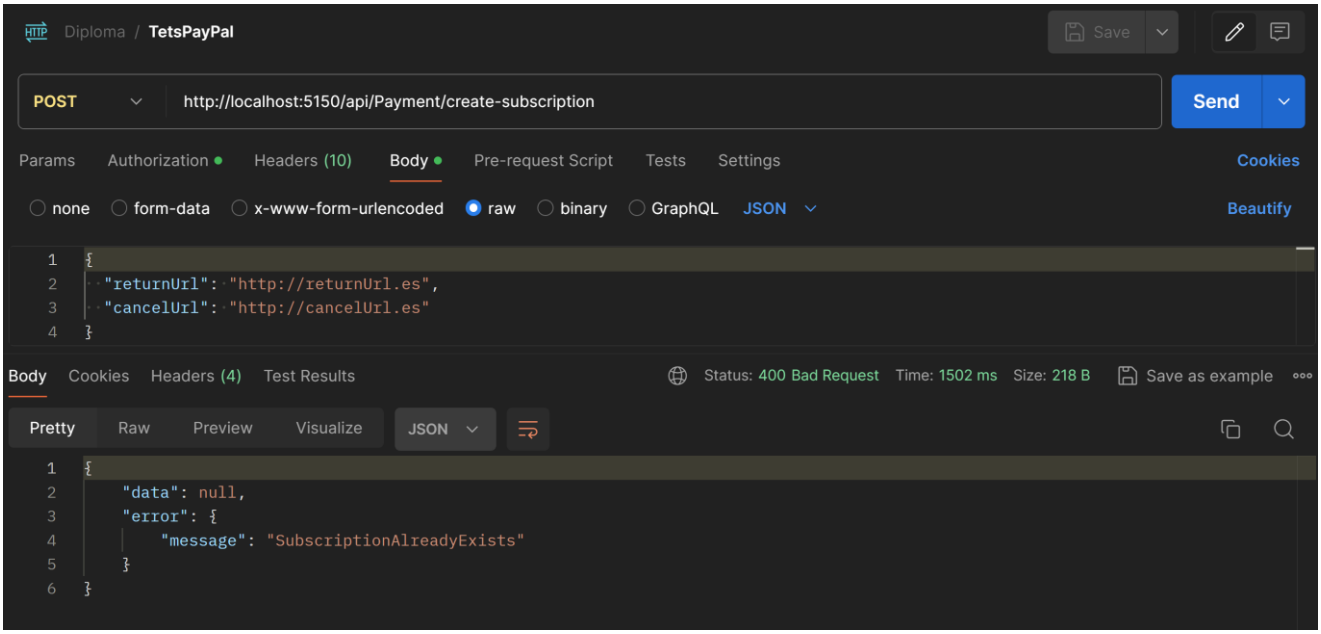


Рисунок 4.5 – Відповідь API у випадку існуючої активної підписки в користувача

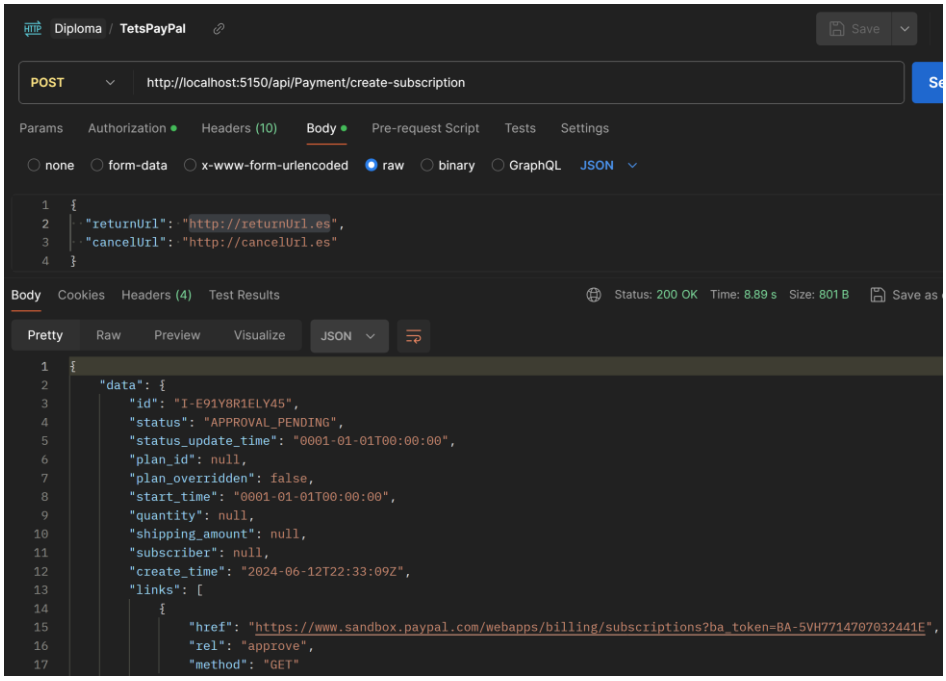


Рисунок 4.6 – Відповідь API після успішного створення підписки у системі PayPal та збереженні в базі даних системи

Після переходу користувачем за посиланням, яке відправляється на клієнт з API та логін в системі PayPal, він побачить вікно оплати, зображене на рисунку 4.7.

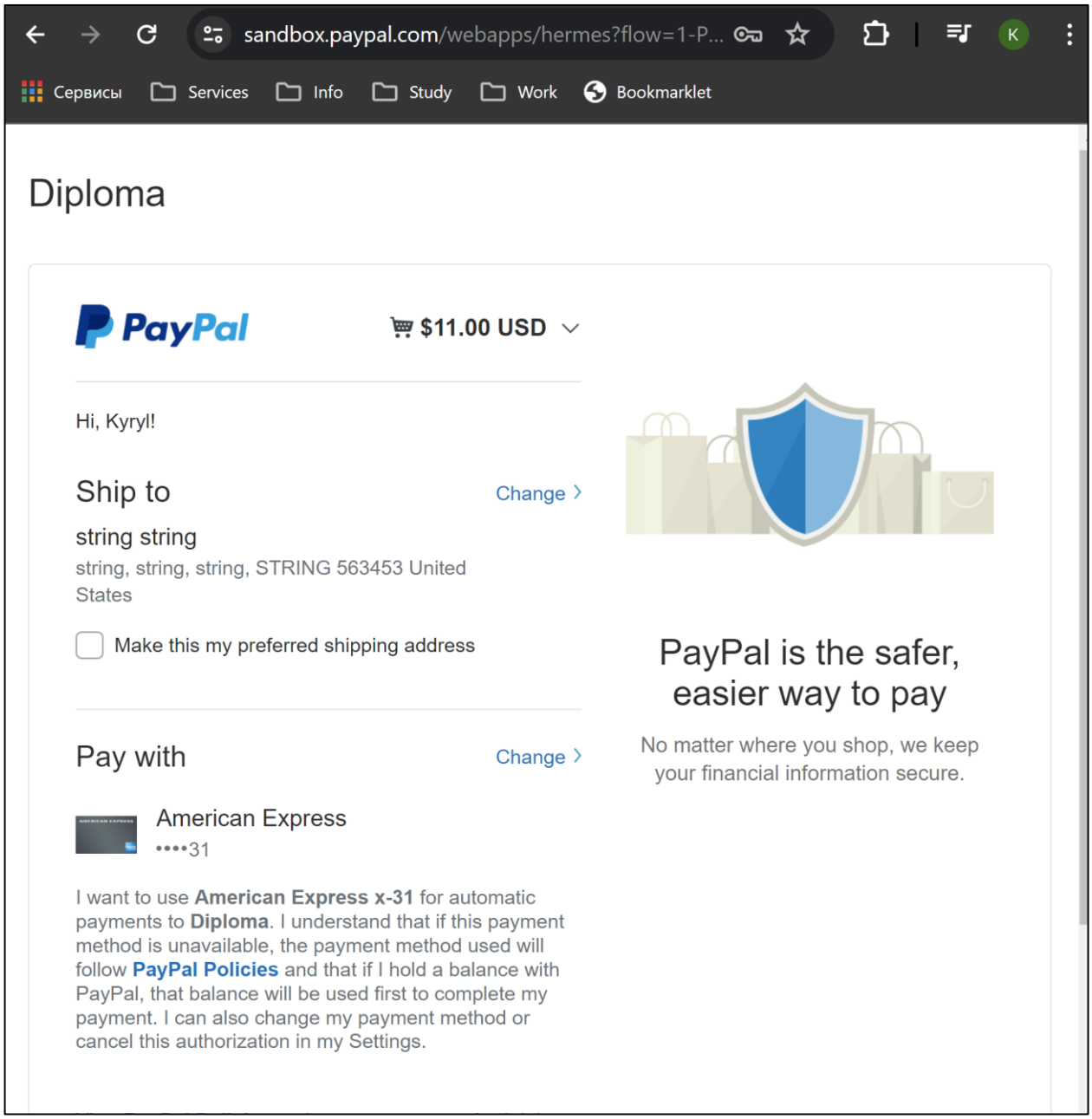


Рисунок 4.7 – Вікно оплати підписки у системі PayPal

Після зміни статусу підписки користувацький інтерфейс відображає відповідні функції, пов'язані з підпискою. Такий підхід забезпечує високу гнучкість та безпеку при роботі з підписками. Використання PayPal дозволяє легко інтегрувати оплату, отримувати актуальну інформацію про статус підписки та автоматично обробляти зміни статусу через вебхуки. Це знижує ймовірність помилок і підвищує зручність для користувачів.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Юніт-тестування

Юніт-тестування є невід'ємною частиною забезпечення якості програмного забезпечення. Це дозволяє виявити та виправити помилки на ранніх етапах розробки, підвищуючи загальну стабільність та надійність системи. В рамках проекту «Pure Survey» було приділено особливу увагу розробці юніт-тестів для кожного класу рівня бізнес-логіки та певних класів рівня інфраструктури.

Метою юніт-тестування було переконатися, що всі окремі частини коду працюють належним чином і відповідають вимогам. Для цього було створено тести, що охоплюють широкий спектр можливих сценаріїв використання, включаючи як позитивні, так і негативні випадки. Було приділено увагу не лише основним сценаріям, а й менш вірогідним випадкам, що дозволило виявити та виправити приховані помилки.

Тестування програмного забезпечення можна порівняти з випробуванням на міцність. Подібно до того, як ми перевіряємо будинок перед заселенням, тестування допомагає виявити і усунути всі можливі проблеми, забезпечуючи комфорт і безпеку для користувачів. Юніт-тестування є першим рівнем цієї перевірки, що дозволяє детально перевірити кожен окремий компонент системи.

В рамках юніт-тестування було створено значну кількість тестів, що охоплюють всі основні аспекти системи. Рисунок 5.1 демонструє кількість тестів та результати їх виконання, що дозволяє оцінити обсяг та ефективність тестування.

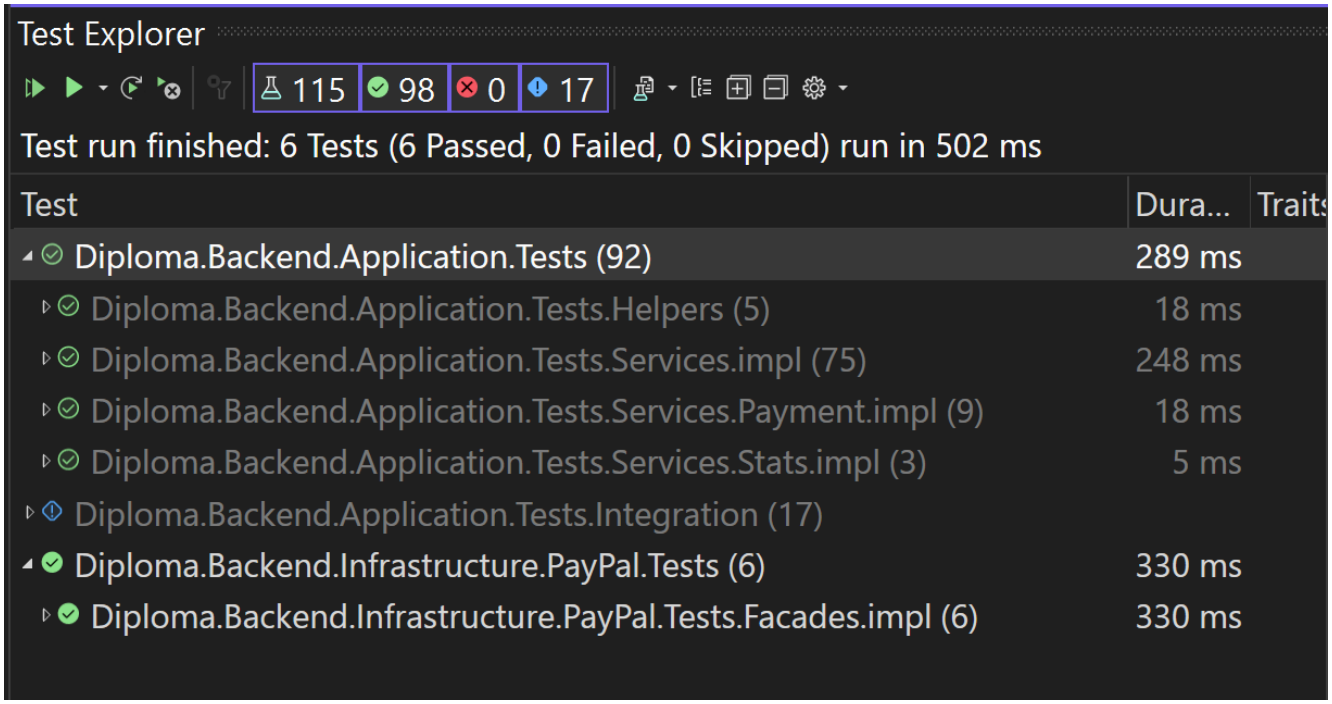
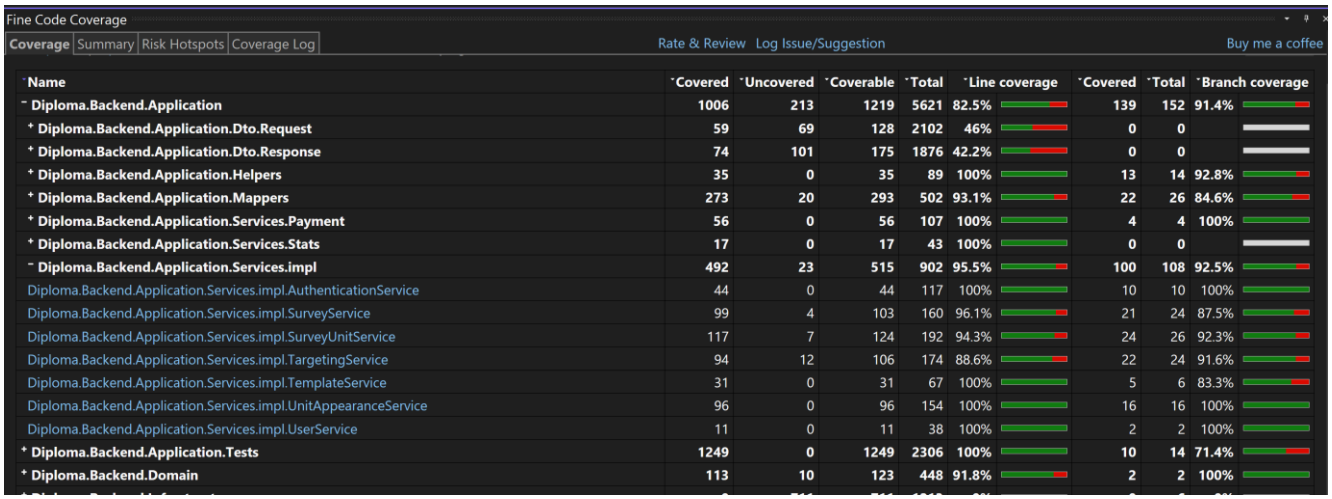


Рисунок 5.1 – Результат виконання юніт-тестів

З рисунка видно, що було створено 98 юніт-тестів, з яких 98 пройшли успішно. Це свідчить про високу якість написаних тестів та їх відповідність заявленим вимогам. Тести охоплюють різні частини системи, зокрема:

– Diploma.Backend.Application.Tests: 92 тестів, що охоплюють різні сервіси та компоненти;
– Diploma.Backend.Application.Tests.Helpers: 5 тестів для допоміжних класів;
– Diploma.Backend.Application.Tests.Services.impl: 75 тестів для основних сервісів;
– Diploma.Backend.Application.Tests.Services.Payment.impl: 9 тестів для платіжних сервісів;
– Diploma.Backend.Application.Tests.Services.Stats.impl: 3 тести для статистичних сервісів;
– Diploma.Backend.Infrastructure.PayPal.Tests: 6 тестів для взаємодії з PayPal.

Кожен клас, що містить бізнес-логіку, був ретельно протестований. Юніт-тести були розроблені для перевірки коректності роботи всіх методів, включаючи обробку даних, маппінг та інші внутрішні процеси. На рисунку 5.2 наведено рівень покриття коду (code coverage), отриманий за допомогою розширення Fine Code Coverage.



Name	Covered	Uncovered	Coverable	Total	Line coverage	Covered	Total	Branch coverage
− Diploma.Backend.Application	1006	213	1219	5621	82.5%	139	152	91.4%
+ Diploma.Backend.Application.Dto.Request	59	69	128	2102	46%	0	0	
+ Diploma.Backend.Application.Dto.Response	74	101	175	1876	42.2%	0	0	
+ Diploma.Backend.Application.Helpers	35	0	35	89	100%	13	14	92.8%
+ Diploma.Backend.Application.Mappers	273	20	293	502	93.1%	22	26	84.6%
+ Diploma.Backend.Application.Services.Payment	56	0	56	107	100%	4	4	100%
+ Diploma.Backend.Application.Services.Stats	17	0	17	43	100%	0	0	
− Diploma.Backend.Application.Services.impl	492	23	515	902	95.5%	100	108	92.5%
Diploma.Backend.Application.Services.impl.AuthenticationService	44	0	44	117	100%	10	10	100%
Diploma.Backend.Application.Services.impl.SurveyService	99	4	103	160	96.1%	21	24	87.5%
Diploma.Backend.Application.Services.impl.SurveyUnitService	117	7	124	192	94.3%	24	26	92.3%
Diploma.Backend.Application.Services.impl.TargetingService	94	12	106	174	88.6%	22	24	91.6%
Diploma.Backend.Application.Services.impl.TemplateService	31	0	31	67	100%	5	6	83.3%
Diploma.Backend.Application.Services.impl.UnitAppearanceService	96	0	96	154	100%	16	16	100%
Diploma.Backend.Application.Services.impl.UserService	11	0	11	38	100%	2	2	100%
+ Diploma.Backend.Application.Tests	1249	0	1249	2306	100%	10	14	71.4%
+ Diploma.Backend.Domain	113	10	123	448	91.8%	2	2	100%

Рисунок 5.2 – Рівень покриття коду юніт-тестами

З рисунка видно, що загальне покриття коду становить 82.5%, через те, що не всі властивості коду було використано у тестах, що не є обов'язковим, а покриття кейсів становить 91.4% що є дуже високим показником та є більш показовим критерієм, який підраховує кількість можливих варіантів проходження коду. Показники покриття коду наступні:

– Diploma.Backend.Application: 82.5% покриття, включаючи 91.4% branch coverage;
– Diploma.Backend.Application.Services.impl: 95.5% покриття, що свідчить про високу надійність сервісів;
– Diploma.Backend.Application.Services.Payment: 100% покриття, що важливо для платіжних операцій;
– Diploma.Backend.Application.Services.Stats: 100% покриття, забезпечуючи точність статистичних даних.

Крім бізнес-логіки, деякі класи інфраструктури також були піддані юніт-тестуванню. Це включало перевірку правильності взаємодії з зовнішніми системами, такими як PayPal, і забезпечення коректної обробки запитів та відповідей. Особливу увагу приділено тестуванню механізмів безпеки та валідації даних.

Загалом, юніт-тестування стало фундаментом для подальших етапів тестування, забезпечивши впевненість у стабільній роботі всіх компонентів системи. Було виявлено та виправлено численні помилки, що забезпечило високу якість та надійність системи «Pure Survey».

5.2 Інтеграційне тестування

Інтеграційне тестування є важливою складовою процесу забезпечення якості програмного забезпечення. В рамках проекту «Pure Survey» було проведено інтеграційні тести для перевірки взаємодії між різними модулями та компонентами системи. Загалом було створено 17 інтеграційних тестів для бізнес-сервісів рівня Application (див. рис. 5.3).

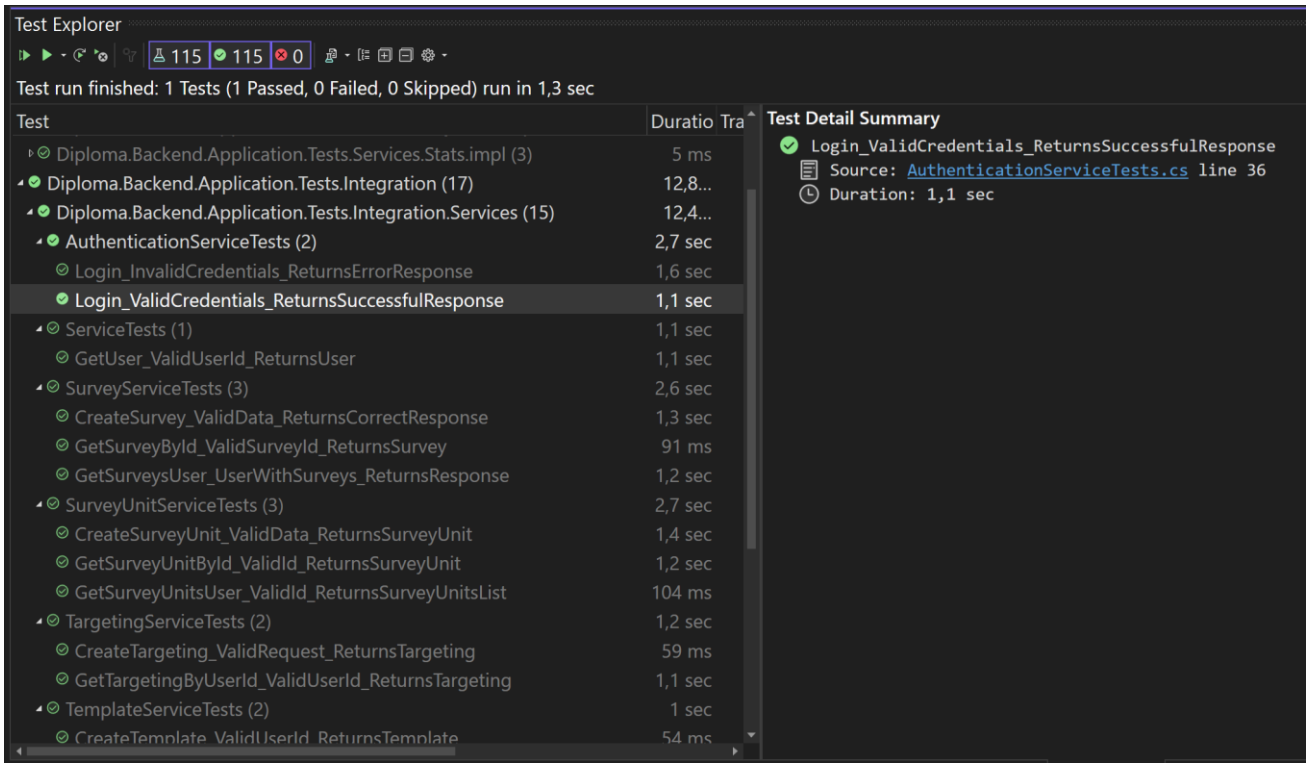


Рисунок 5.3 – Результат виконання інтеграційних тестів

Метою інтеграційного тестування є перевірка коректної взаємодії між різними компонентами системи. На відміну від юніт-тестів, які зосереджені на перевірці окремих класів та методів, інтеграційні тести охоплюють ширші сценарії використання, включаючи взаємодію між сервісами та модулями. Це дозволяє виявити можливі проблеми у взаємодії компонентів, які можуть залишитися непоміченими під час юніт-тестування.

У проекті «Pure Survey» інтеграційне тестування було спрямоване на перевірку ключових сценаріїв взаємодії між бізнес-сервісами. Було створено 17 інтеграційних тестів, які охоплюють основні випадки використання системи. Важливо зазначити, що не всі методи були протестовані інтеграційними тестами – фокус був на головних функціональних можливостях та критичних шляхах обробки даних. Це рішення було прийнято, оскільки більшість можливих кейсів вже була детально протестована на рівні юніт-тестів, які забезпечують високий рівень покриття коду та перевіряють коректність роботи окремих компонентів, а також через значні ресурси, необхідні для розробки та підтримки великої кількості інтеграційних тестів.

Проведені інтеграційні тести продемонстрували високу стабільність системи. Всі ключові бізнес-сервіси успішно пройшли перевірку на коректну взаємодію. Виявлені незначні проблеми були оперативно виправлені, що дозволило підвищити загальну надійність системи.

Приклади основних інтеграційних тестів включають перевірку коректної взаємодії сервісів з класами рівня інфраструктури та базою даних, включаючи валідацію користувацьких даних та генерацію токенів, тестування основних операцій з опитуваннями, включаючи створення, редагування та видалення опитувань, отримання статистики опитувань, а також перевірку коректної роботи сервісу управління користувачами, включаючи створення нових користувачів тощо.

Інтеграційне тестування дозволило переконатися у коректній взаємодії ключових компонентів системи. Обмежений обсяг тестів був виправданий завдяки високому рівню покриття юніт-тестами та фокусу на критичних точках взаємодії.

Це забезпечило ефективне використання ресурсів та високу якість кінцевого продукту.

5.3 Мануальне тестування

Мануальне тестування є важливою складовою забезпечення якості програмного забезпечення. Воно дозволяє перевіряти функціональність системи вручну, що особливо корисно для виявлення проблем, які автоматизовані тести можуть пропустити. У рамках проекту «Pure Survey» було вирішено проводити мануальне тестування API за допомогою інструменту Postman.

Postman є потужним інструментом, який надає можливість відправляти HTTP-запити, отримувати відповіді та аналізувати їх. Це дозволяє ефективно тестувати взаємодію між клієнтською частиною та сервером, перевіряти коректність обробки запитів та відповідей, а також виявляти можливі помилки у передачі та обробці даних.

Мануальне тестування за допомогою Postman включало перевірку всіх можливих випадків використання системи. Це забезпечило високу якість перевірки функціональності та дозволило переконатися, що всі ключові сценарії обробки даних працюють коректно. Зокрема, було протестовано реєстрацію користувачів, авторизацію, створення та редагування усіх сутностей системи, взаємодію між різними компонентами системи та багато інших функціональних можливостей.

На додаток до тестування API, також було проведено мануальне тестування через клієнтську частину системи. Виконано основні потоки програми, що дозволило перевірити інтеграцію та коректну роботу функціональності з точки зору кінцевого користувача. Це включало перевірку реєстрації, авторизації, створення і редагування опитувань та інших ключових сценаріїв використання.

ВИСНОВКИ

У ході роботи було створено API-частину системи для проведення опитувань, яка дозволяє користувачам легко створювати гнучкі опитування, вбудовувати їх на сторонні веб-сайти та отримувати детальну статистику відповідей за різними критеріями. Система розроблена з використанням сучасних технологій що забезпечує гнучкість, масштабованість та надійність.

Для забезпечення фінансової стійкості сервісу було розроблено механізм монетизації з інтеграцією безпечних платіжних шлюзів PayPal, що включає одноразові платежі за публікацію опитувань та модель підписки для доступу до розширених функцій та аналітики.

Під час виконання роботи було визначено специфікацію API-частини програмної системи для проведення соціальних опитувань. Було проведено аналіз предметної галузі соціологічних досліджень та опитувань, зокрема різних методів збору даних і особливостей онлайн опитувань.

На основі виявлених проблем була поставлена задача створення системи «Pure Survey» для їх вирішення. Було визначено основні вимоги до системи та детальні функціональні та нефункціональні вимоги.

Наступним етапом стало визначення архітектури системи та її проектування. Було створено UML діаграми прецедентів (use case діаграми) для візуалізації функціональних можливостей системи для звичайних користувачів та адміністраторів. Спроектовано загальну архітектуру з використанням ASP.NET Core Web API, підходу чистої архітектури та Entity Framework Core для роботи з базою даних. Також було створено діаграми компонентів та розгортання, та представлено логічну модель бази даних.

Крім того, було виконано комплексне тестування системи, включаючи юніт-тестування, інтеграційне тестування та мануальне тестування з великим покриттям, що забезпечило високу якість та надійність програмного забезпечення. Було надано детальну інформацію про проведене тестування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ASP.NET Core Web API documentation – URL: https://dotnet.microsoft.com/en-us/apps/aspnet/apis (дата звернення: 03.04.2024).

2. Clean Architecture in ASP .NET Core Web API – URL: https://juldhais.net/clean-architecture-in-asp-net-core-web-api-4e5ef0b96f99 (дата звернення: 03.04.2024).

3. SQL Server technical documentation – URL: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16 (дата звернення: 02.04.2024).

4. REST API tutorial. URL: https://restfulapi.net/ (дата звернення: 04.04.2024).

5. Entity Framework documentation hub – URL: https://learn.microsoft.com/en-us/ef/ (дата звернення: 05.04.2024).

6. Get started with PayPal REST APIs – URL: https://developer.paypal.com/api/rest/ (дата звернення: 01.05.2024).

7. Martin, R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design: підручник. Prentice Hall, 2017. 432 с.

8. Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software: підручник. Addison-Wesley, 2004. 560 с.

9. Richardson, L., Amundsen, M. RESTful Web APIs: підручник. O'Reilly Media, 2013. 404 с.

10. Deutchman, D. M., Fassnacht, I. C., Vaughn, C. J. Social Survey Methods: підручник. Sage Publications, 2007. 276 с.

11. Dillman, D. A., Smyth, J. D., Christian, L. M. Web Survey Methods: підручник. Wiley, 2014. 512 с.

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

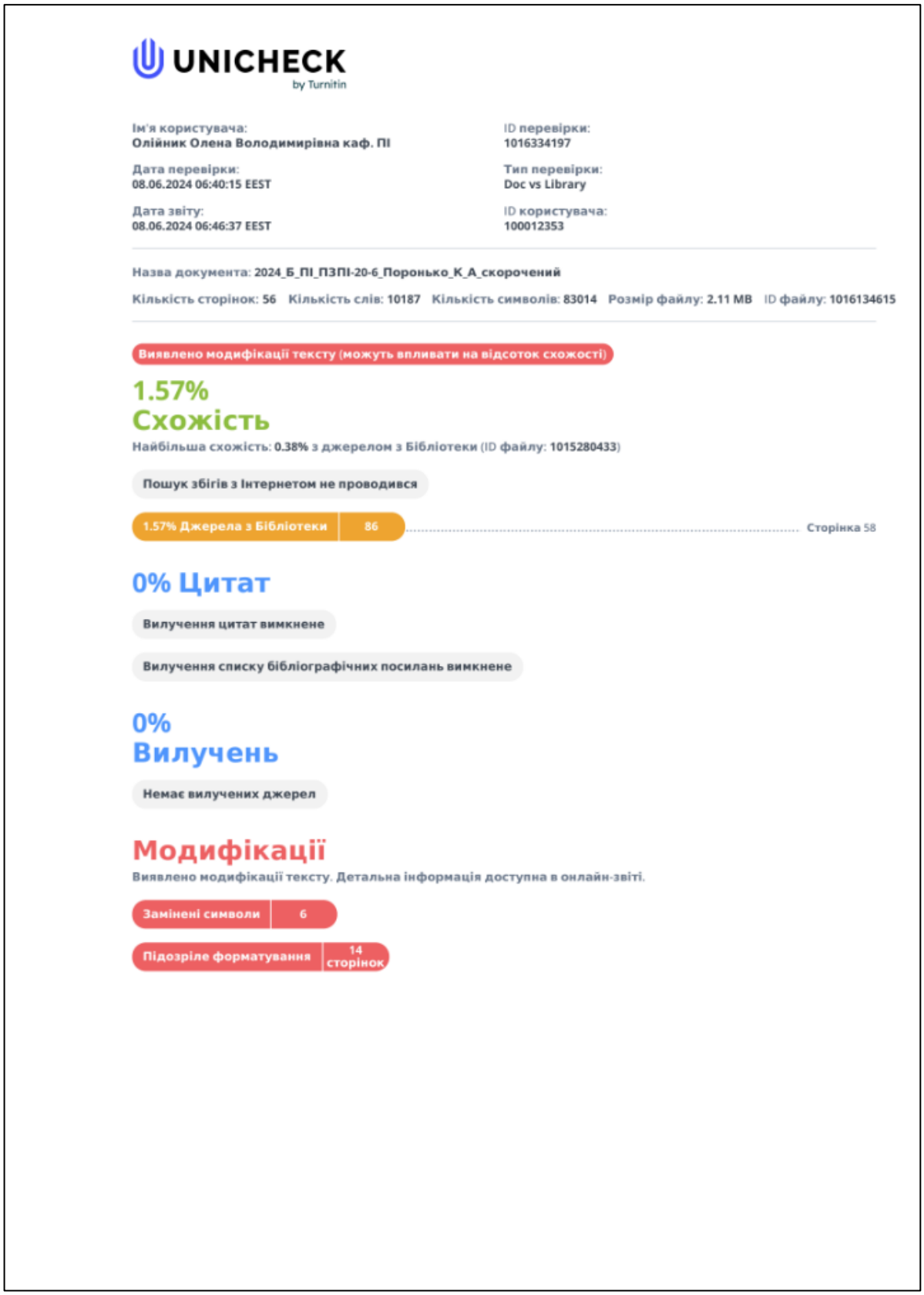


Рисунок А.1 – Результаті перевірки на унікальність тексту в базі ХНУРЕ

ДОДАТОК Б

Слайди презентації

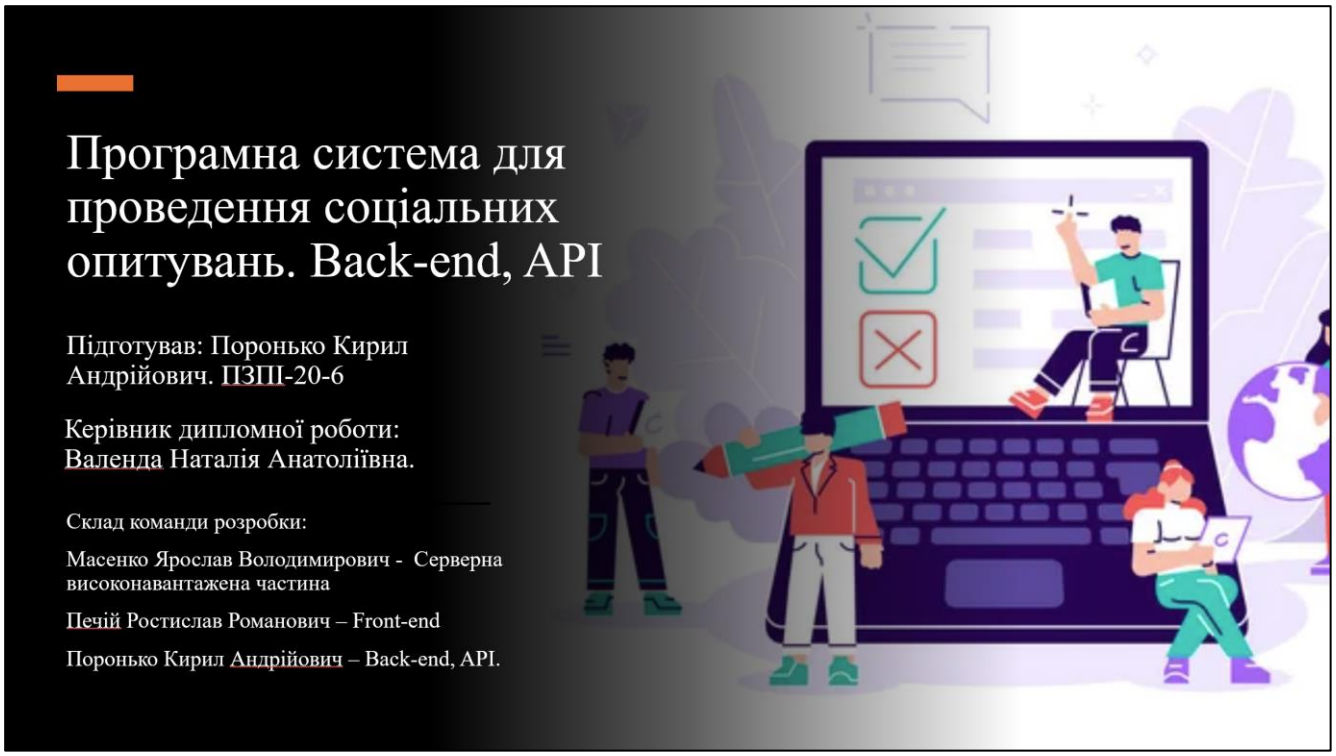


Рисунок Б.1 – Слайд 1

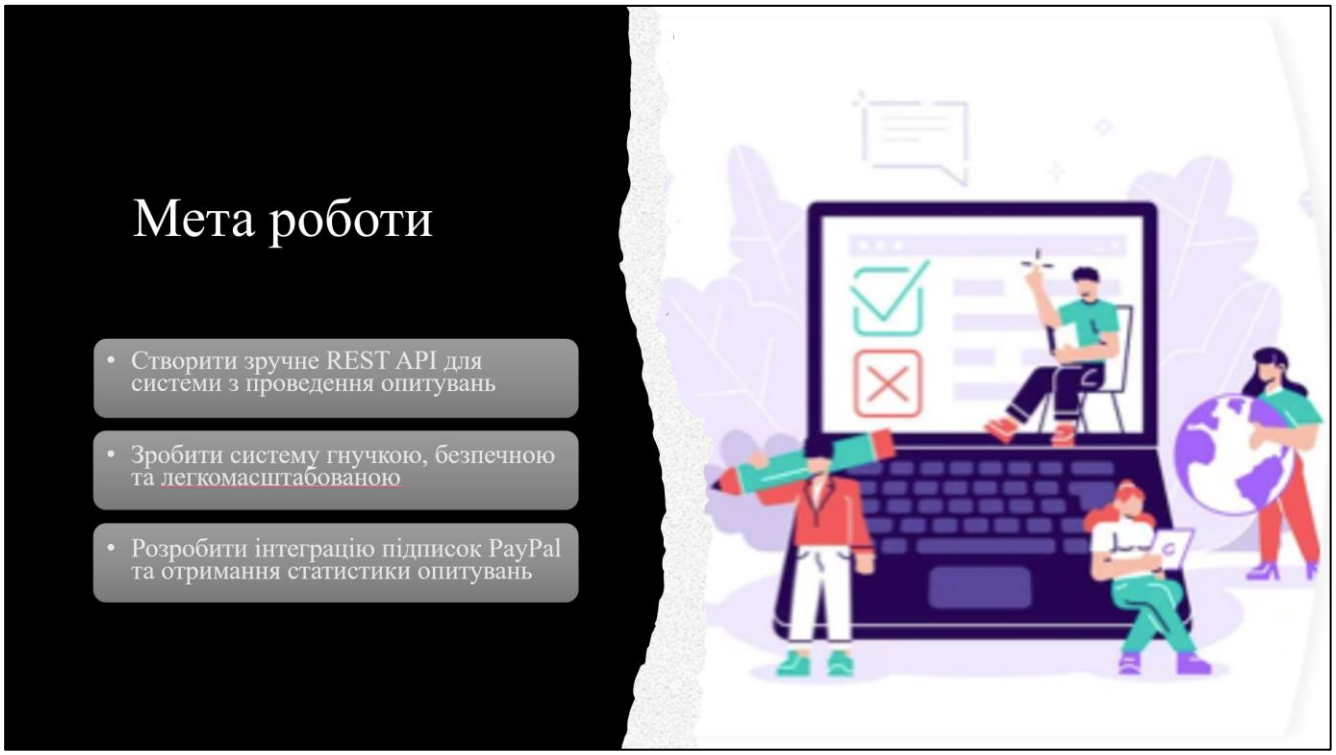


Рисунок Б.2 – Слайд 2

Рисунок Б.3 – Слайд 3

Рисунок Б.4 – Слайд 4

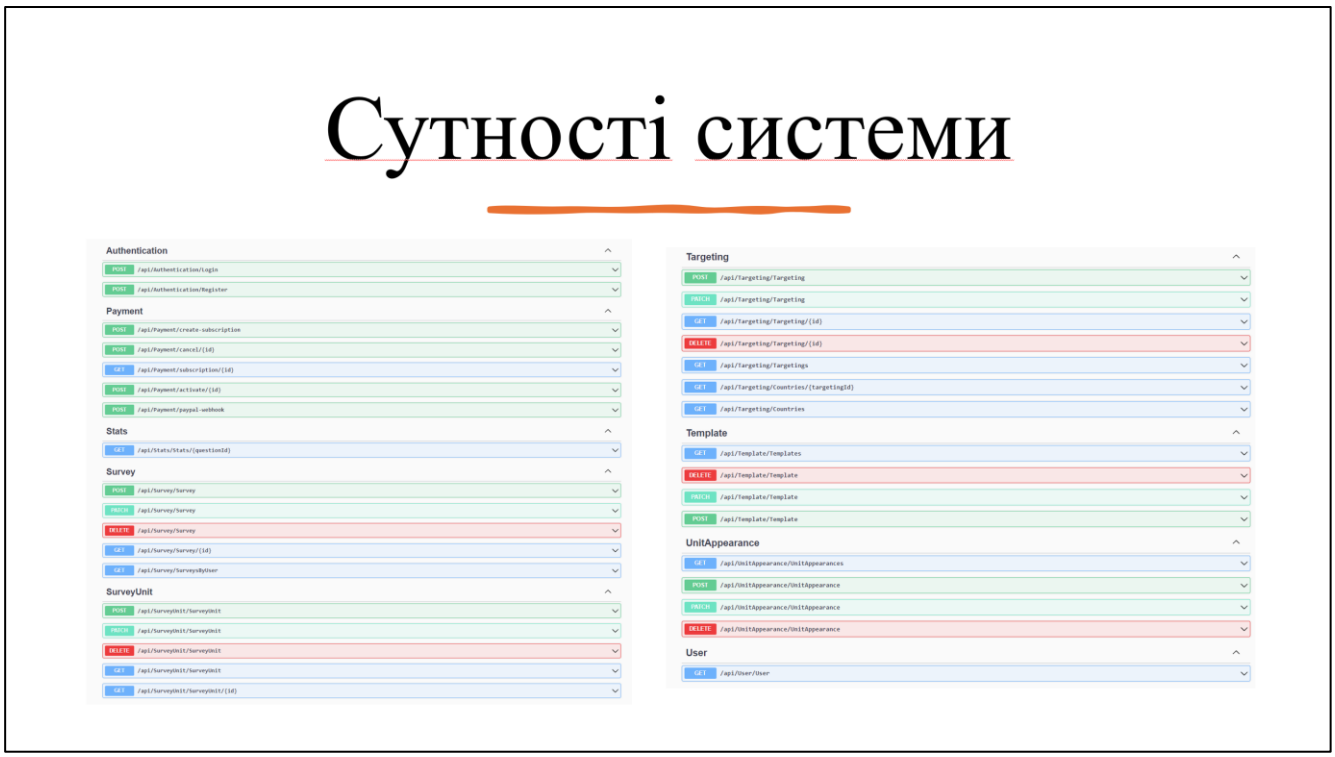


Рисунок Б.5 – Слайд 5

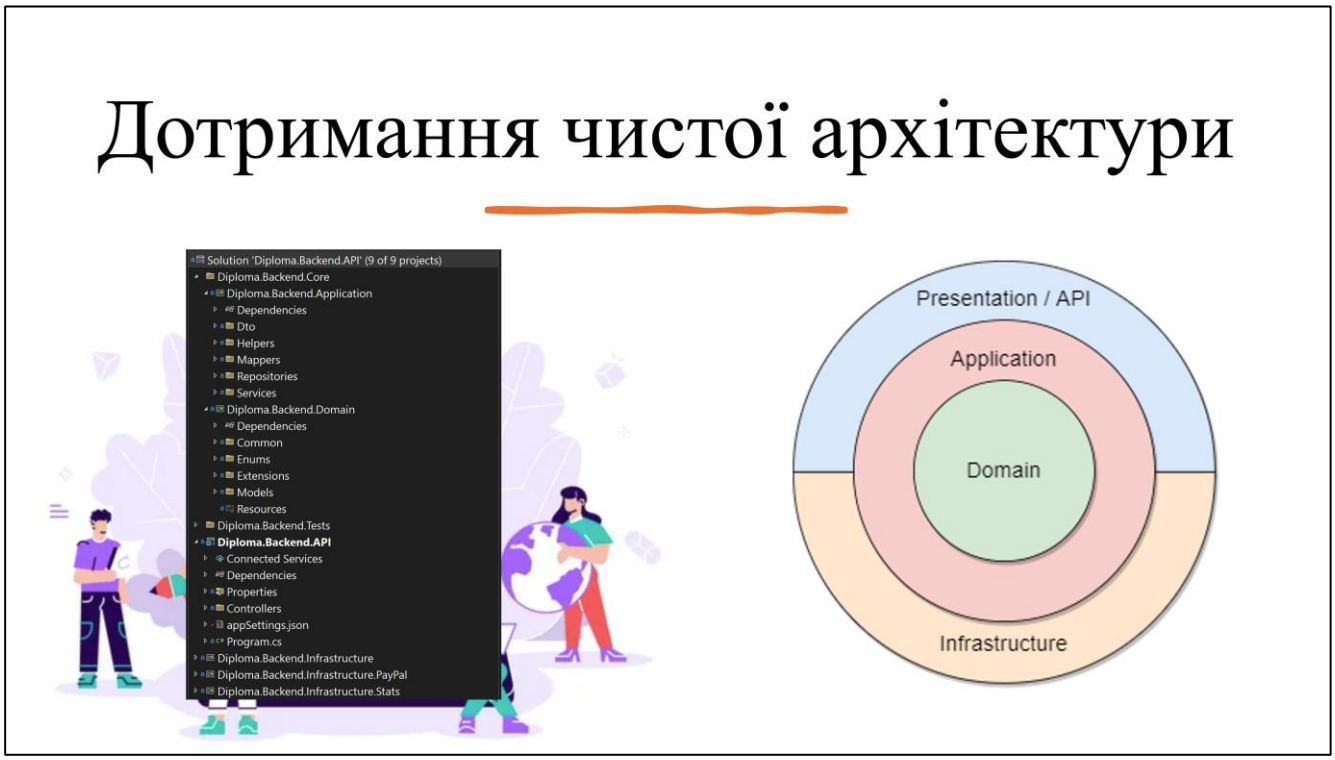


Рисунок Б.6 – Слайд 6

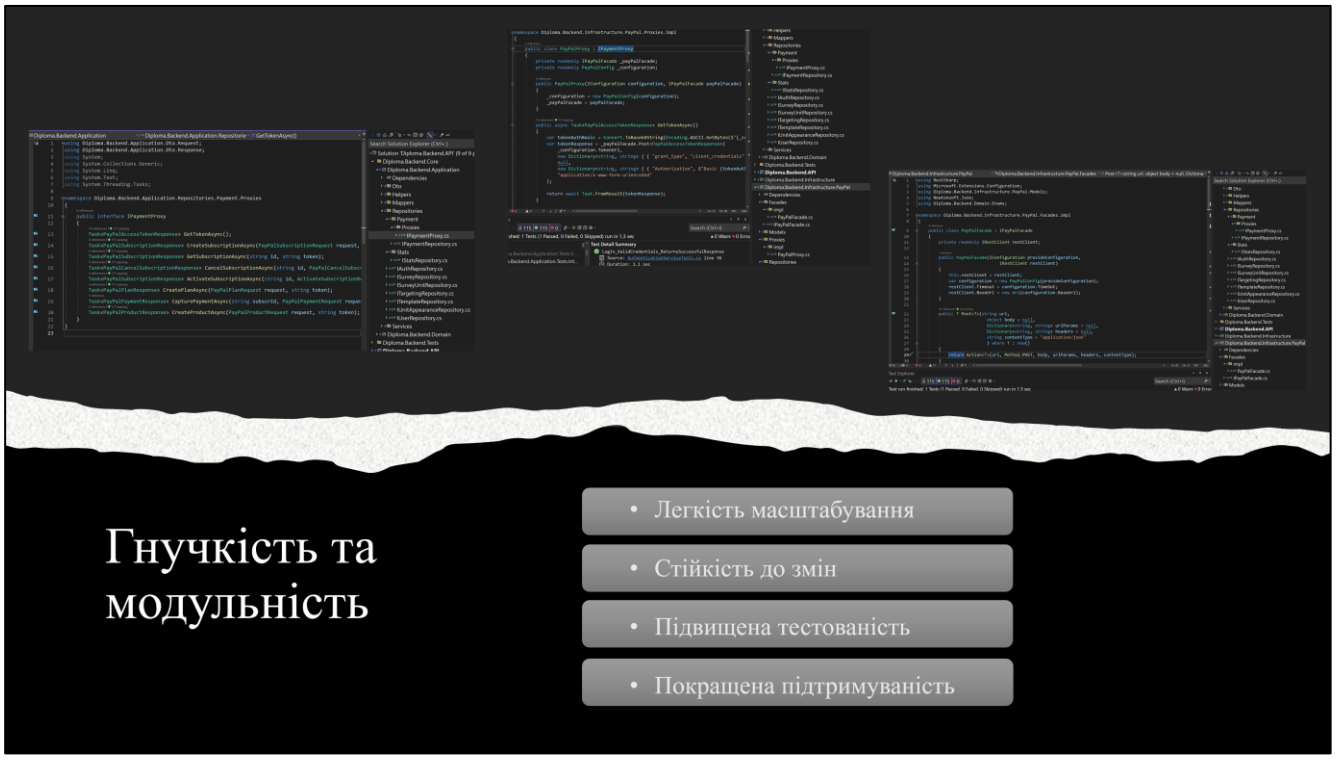


Рисунок Б.7 – Слайд 7

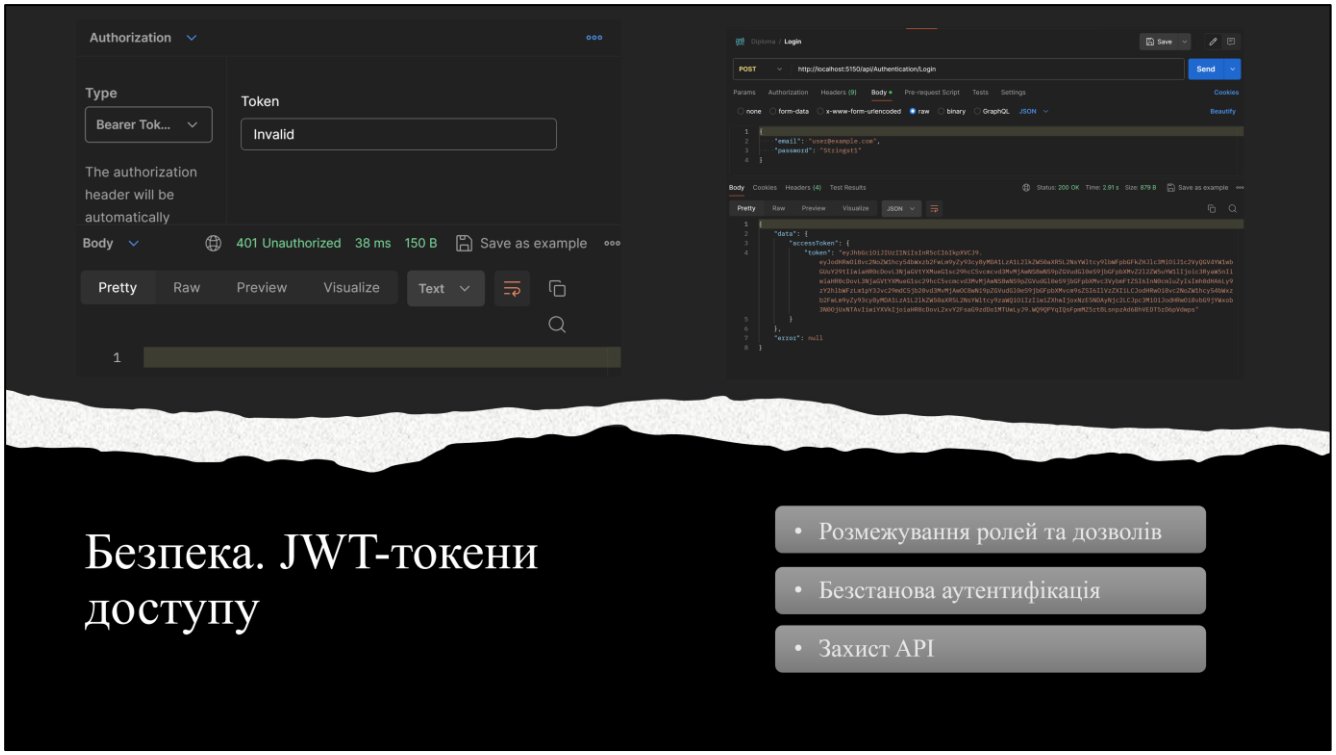


Рисунок Б.8 – Слайд 8

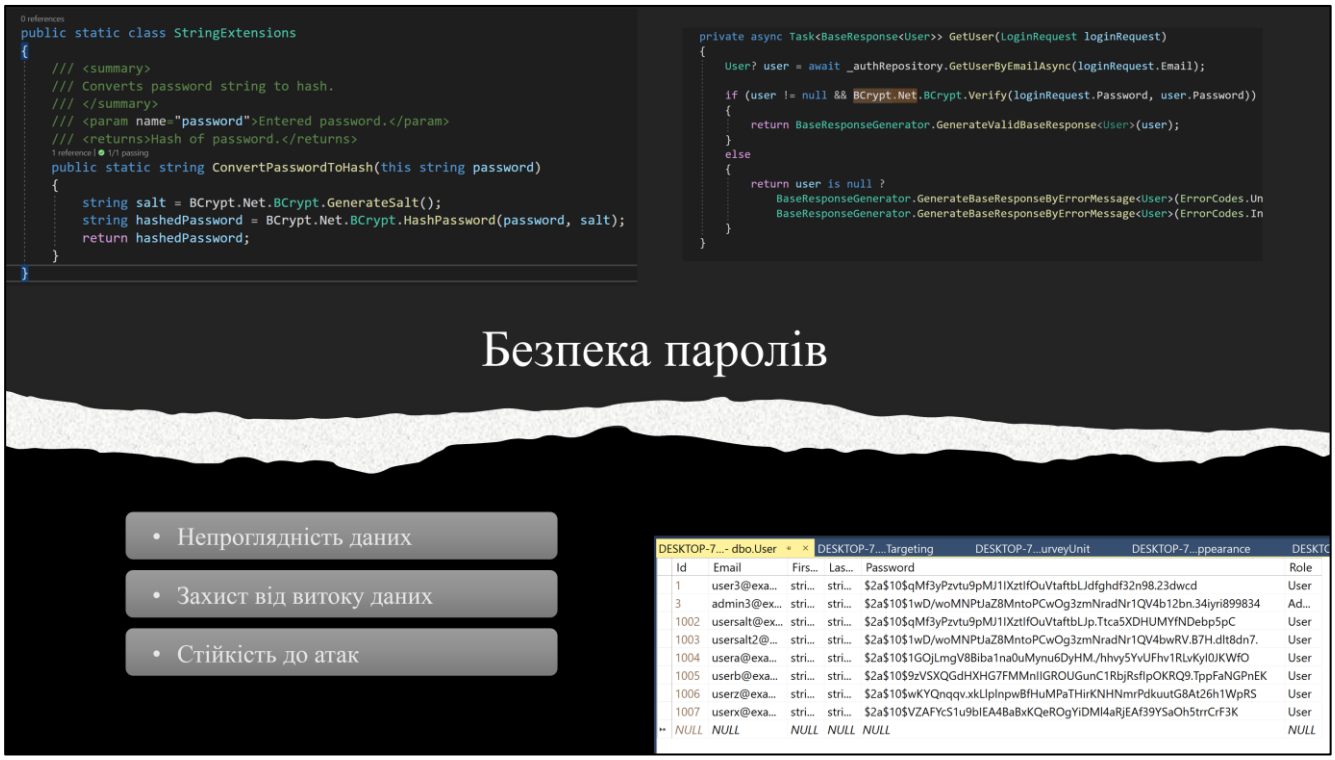


Рисунок Б.9 – Слайд 9

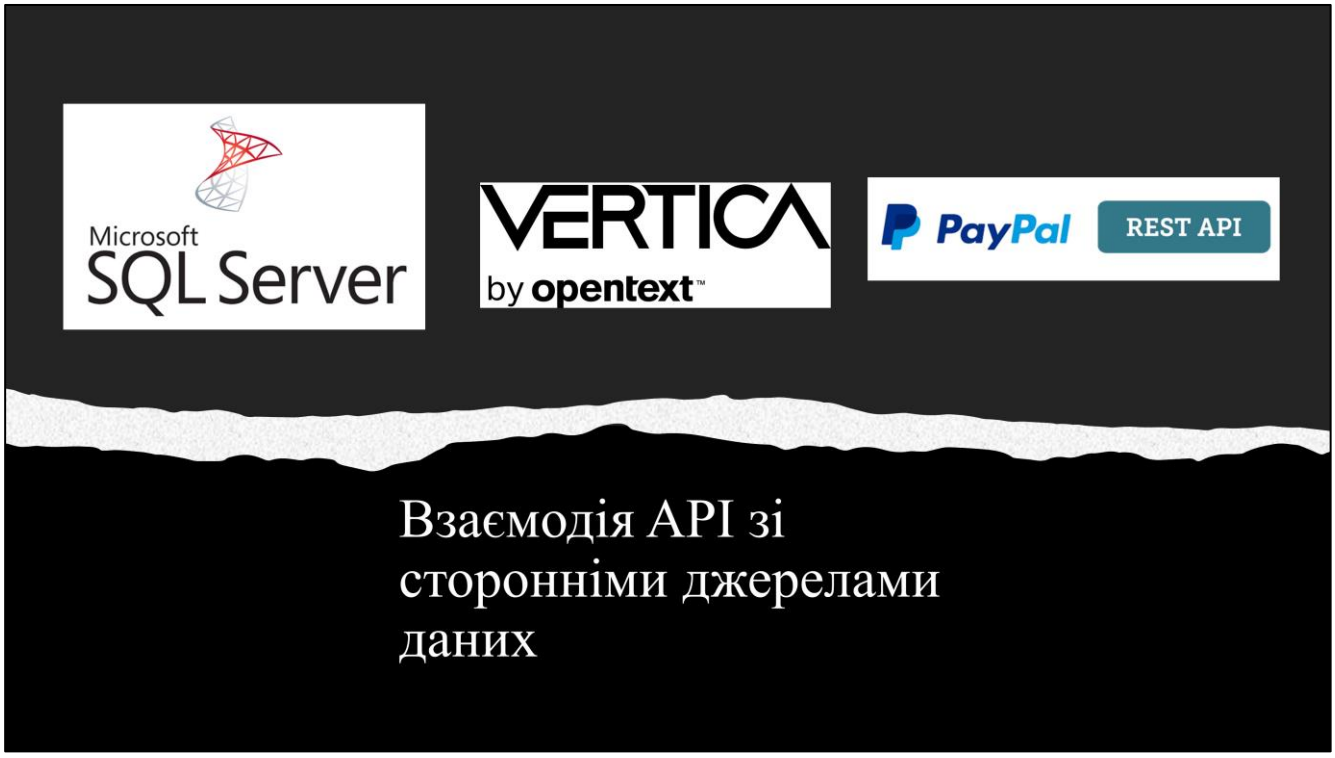


Рисунок Б.10 – Слайд 10

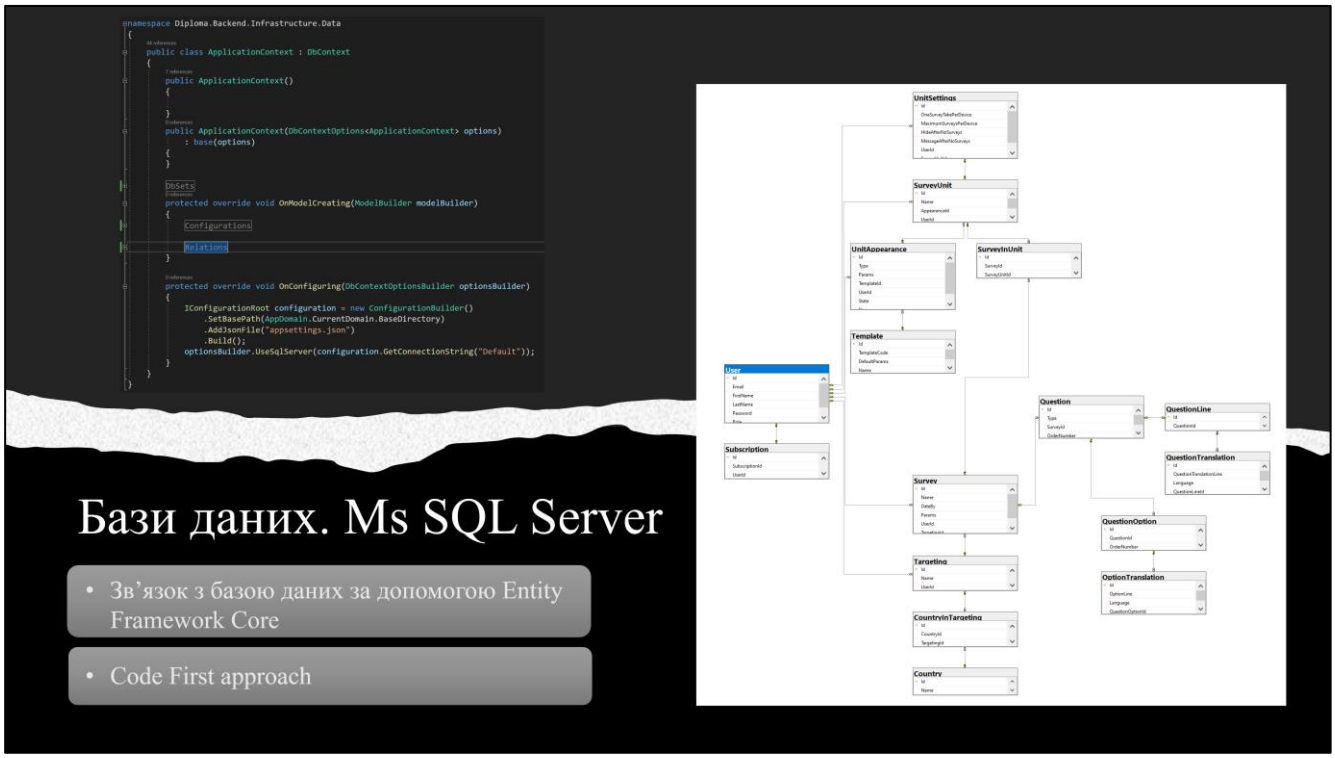


Рисунок Б.11 – Слайд 11

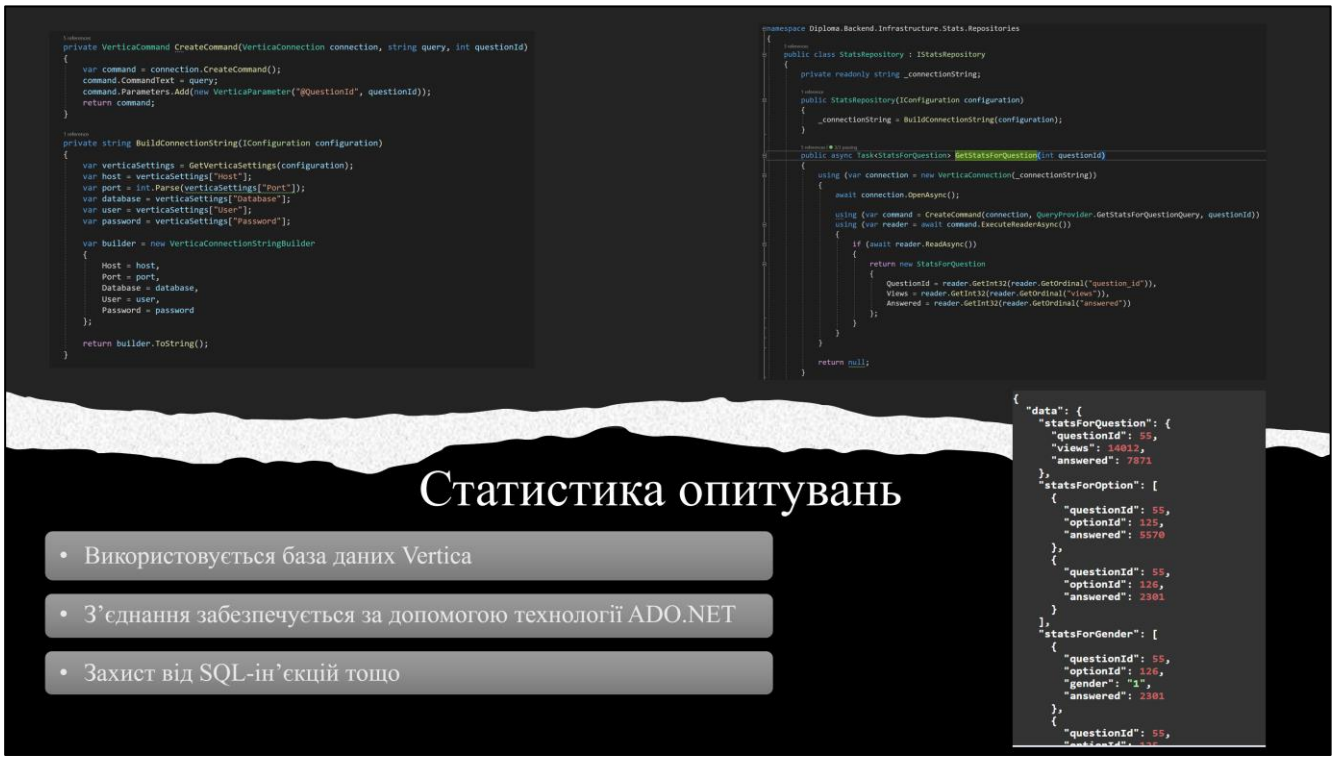


Рисунок Б.12 – Слайд 12

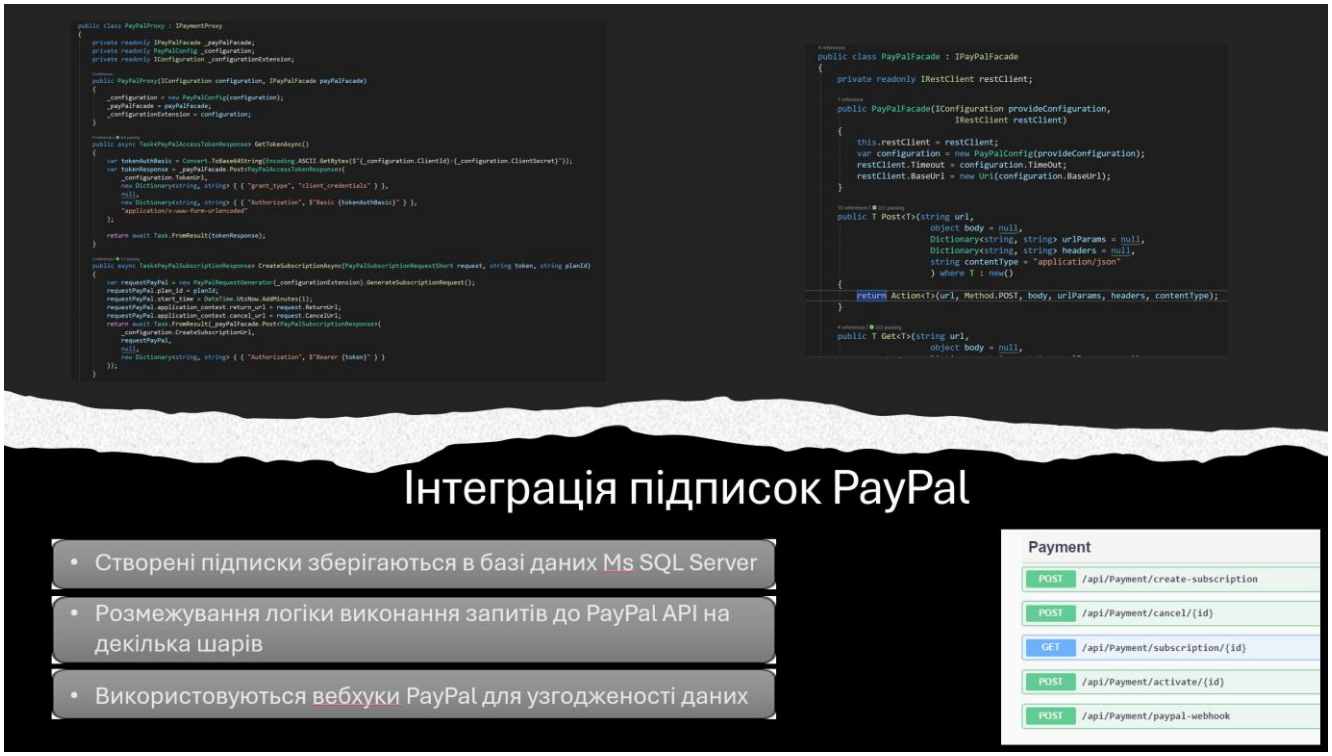


Рисунок Б.13 – Слайд 13

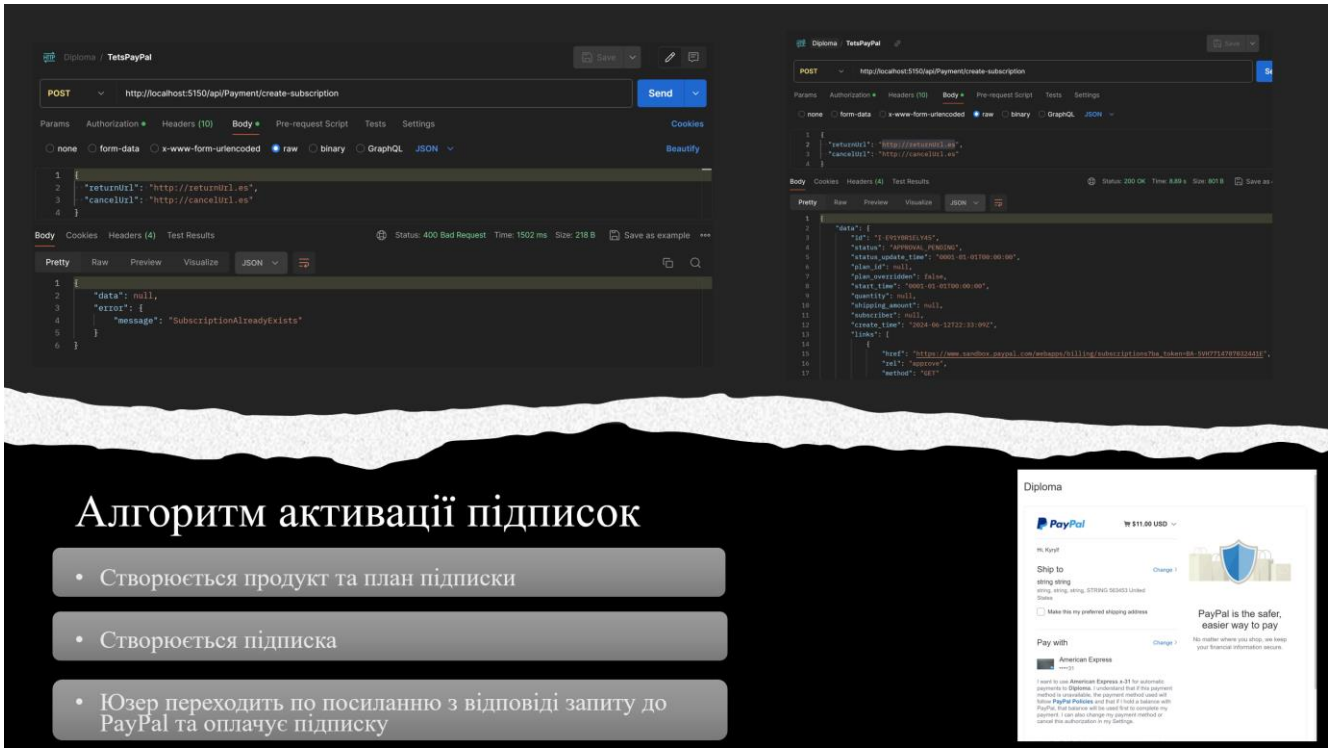


Рисунок Б.14 – Слайд 14

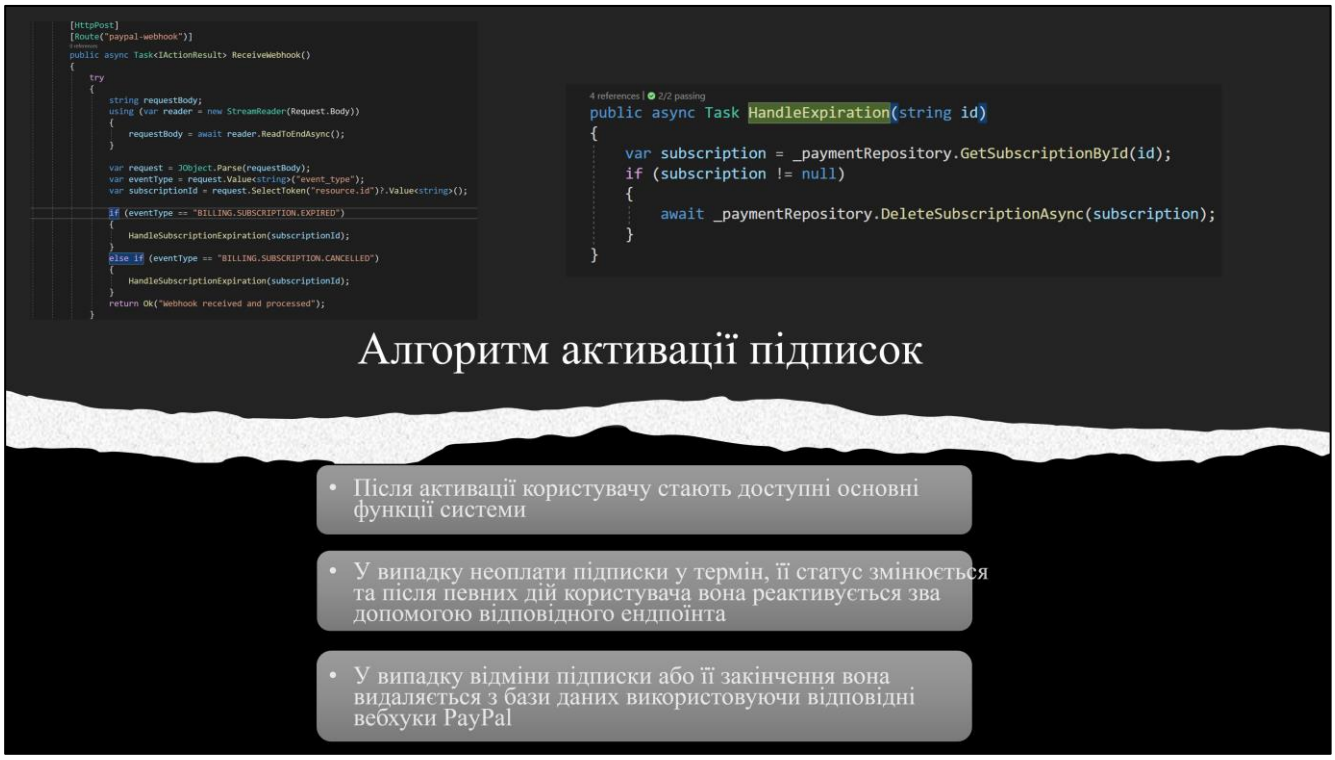


Рисунок Б.15 – Слайд 15

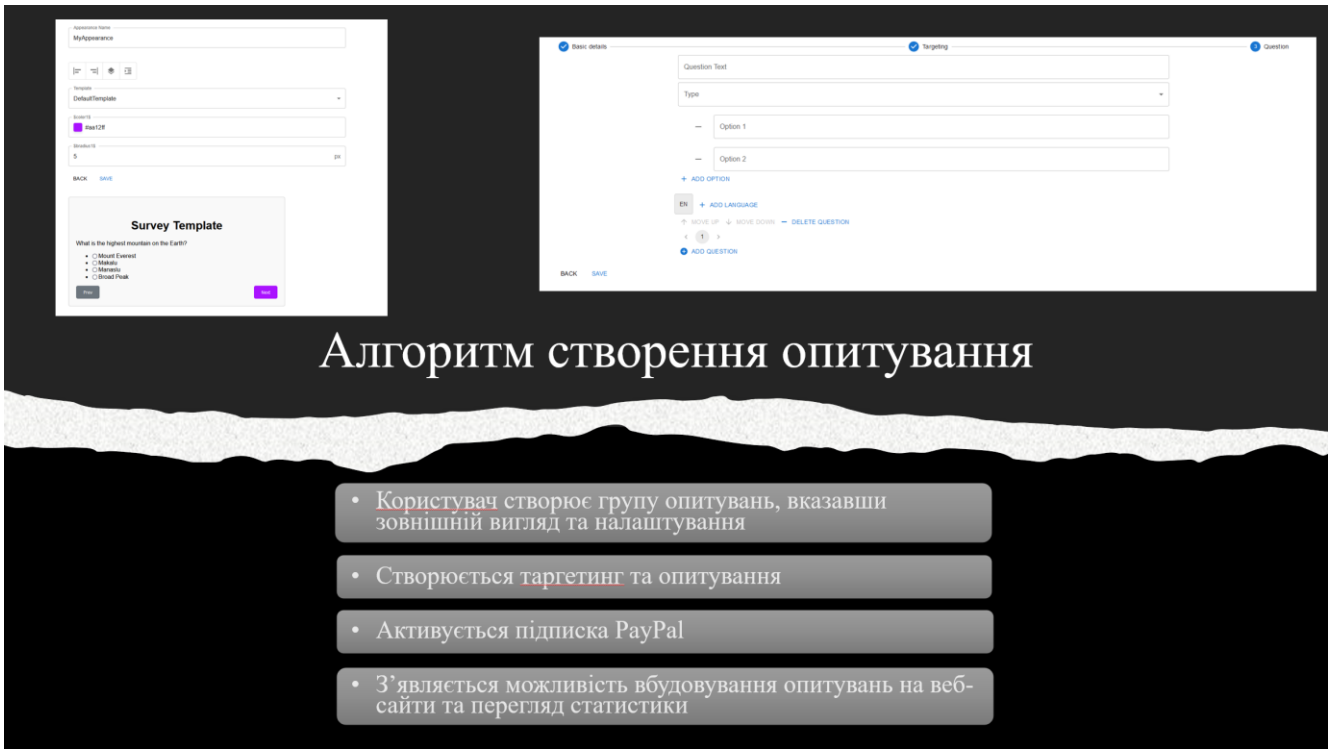


Рисунок Б.16 – Слайд 16

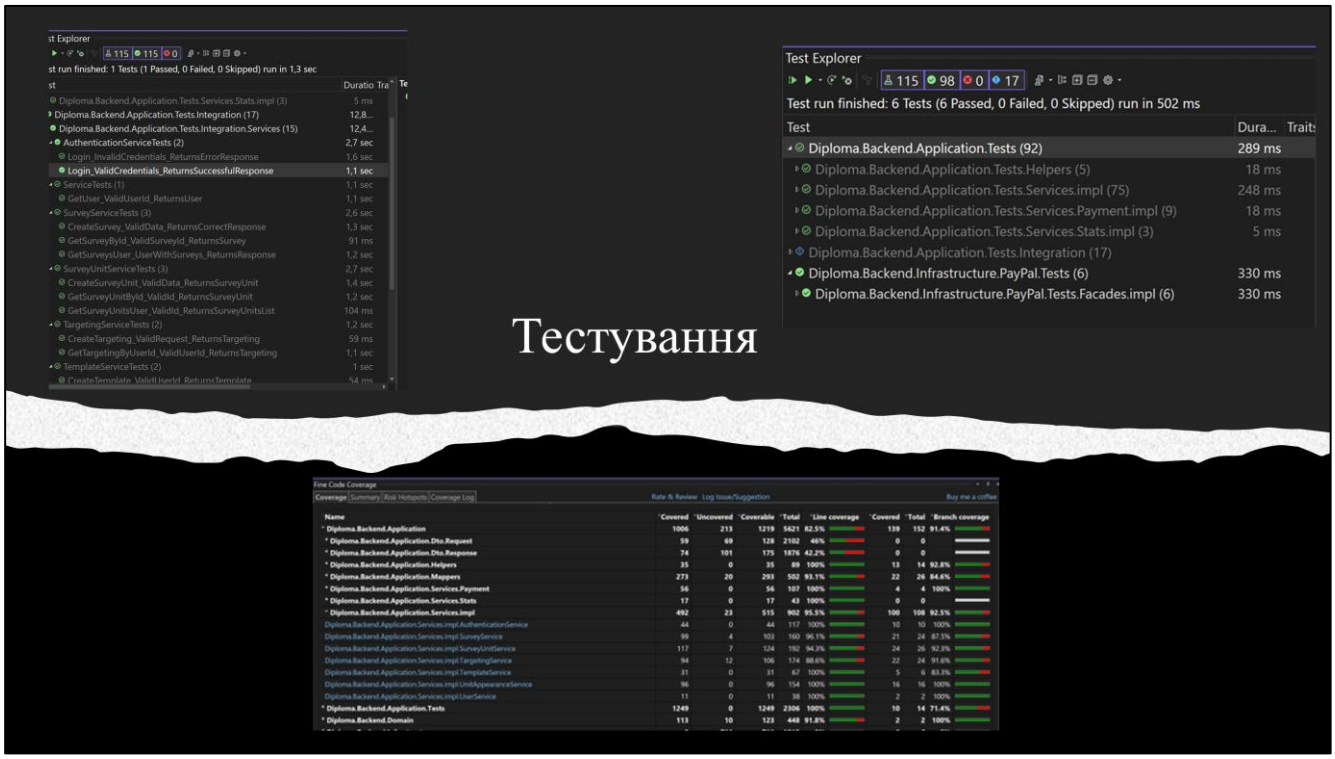


Рисунок Б.17 – Слайд 17

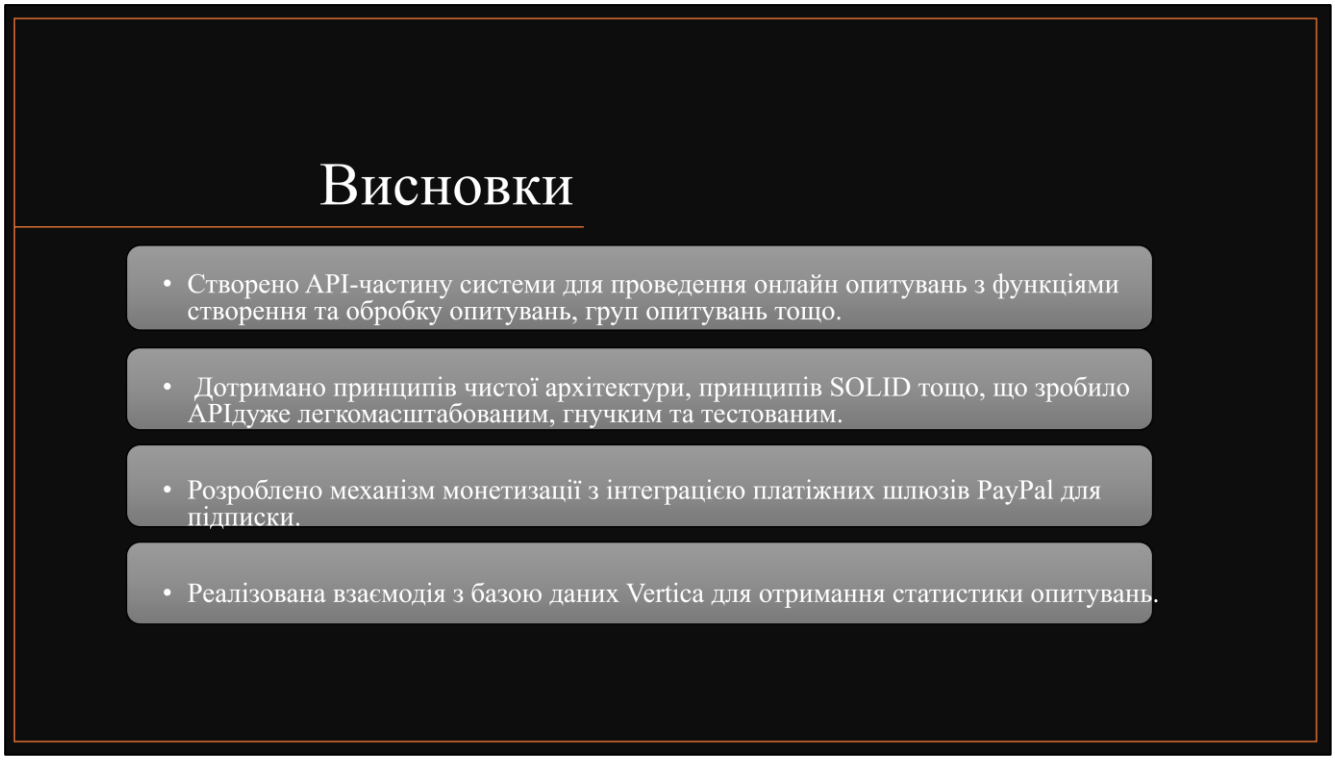


Рисунок Б.18 – Слайд 18

ДОДАТОК В

Специфікація програмного забезпечення

СПЕЦИФІКАЦІЯ ПЗ

До системи для проведення соціальних опитувань

Виконали:

Поронько Кирил Андрійович

Печій Ростислав Романович

Масенко Ярослав Володимирович

Рисунок В.1 – Сторінка 1 специфікації пз

1. Вступ

1.1 Огляд продукту

Ціль цього документа полягає у створенні програмної системи для проведення опитувань «Pure Survey», яка дозволяє користувачам створювати різноманітні опитування, обираючи певні налаштування, з можливістю вбудовувати опитування на інші вебсайти, та переглядати статистику відповідей на опитування.

1.2 Межі

«Pure Survey» це програмна система, яка дозволяє користувачам створювати свої власні соціальні опитування, використовуючи гнучкі інструменти редагування.

Зареєстровані користувачі можуть оглянути та використовувати функціонал зі створення опитувань, змінювати відповідну аудиторію, на яку націлене опитування та їх редагувати.

Користувачі можуть бачити інформацію про створені опитування, групи в яких вони містяться, а з активною підпискою – також переглядати статистику обрану користувачем з певними відібраними даними по певних опитуваннях. Також експортувати створені опитування у вигляді скрипту для їх використання на різних веб-сайтах.

Додатково користувач може додати певні кошти на свій особистий рахунок профілю для публікації опитувань в мережі «Інтернет», відповідно використовуючи кошти за певну кількість отриманих відгуків.

1.3 Означення та абревіатури

Таблиця 1 - Означення

Термін	**Означення**
Таргетинг	Цільова аудиторія для якої буде доступне опитування. Залежить від країни користувача
Зареєстрований користувач	Користувач, який створив профіль у системі та може використовувати функції створення опитувань
Опитування	Елемент опитування створений юзером, залежний від налаштувань таргетингу. Опитування можна вбудовувати в інші вебсайти

Рисунок В.2 – Сторінка 2 специфікації пз

Група опитувань	Декілька однаково налаштованих опитувань в контексті зовнішнього вигляду та налаштувань правил опитування, які відрізняються таргетингом та вмістом питань
Front-end/Клієнт	Графічний інтерфейс, який відображається користувачеві за допомогою HTML, CSS і JavaScript, щоб користувачі могли переглядати дані та взаємодіяти з ними
Серверна частина/API	Інтерфейс взаємодії між клієнтом і сервером, який також взаємодіє з базою даних та відповідає за базову частину програмної системи.
Високонавантажена серверна частина	Інтерфейс взаємодії між клієнтом і сервером, який також взаємодіє з базою даних та відповідає за опрацювання взаємодії з опитуваннями.
Темплейт	Зовнішній вигляд групи опитувань, задається у форматі html-коду.

2. Загальний опис

2.1 Перспективи продукту

Програмна система складається з таких частин: веб-додаток користувача, веб-сервер для додатку, основний сервер бази даних для додатку, додаткова високонавантажена серверна частина і база даних для додаткової системи

Основна база даних міститься в СКБД MSSQL, яка містить в собі дані користувачів та опитувань. Користувач не може напряму комунікувати з сервером бази даних, але може використовувати основне API системи.

При потребі даних клієнт формує запит і посилає його на API веб-серверу. Даний веб-сервер інтерпретує запит і повертає помилку, якщо запит не може бути виконаний, інакше він формує та надсилає запит на сервер бази даних, або на інший додатковий веб-сервер відповідно до запиту користувача. База даних повертає помилку або дані, після чого веб-сервер перерозподіляє цю інфомацію необхідним чином і надсилає клієнтові. Тільки клієнт відповідає за надання даних у графічному представленні.

Веб-сервер може додавати або змінювати дані під час обміну інформації. API має бути прихована від користувачів, але доступна для інших частин системи і за її межами.

Рисунок В.3 – Сторінка 3 специфікації пз

Додатковий веб-сервер використовує відповідно створену для нього базу даних і відповідає за внесення згенерованих подій під час проходження опитування і їх зберігання у додаткову базу даних, яка міститься СКБД Vertica. Також ця частина системи відповідальна за генерацію скриптів для вбудовування на інші веб-сайти.

2.2 **Функції продукту**

За допомогою клієнта зареєстровані користувачі зможуть створювати свої групи опитувань, налаштовуючи їхній зовнішній вигляд, правила проходження опитування, та створення одиниць опитування в групах, для яких будуть доступні налаштування контенту опитування та таргетингу. Опитування можна влаштовувати в інші вебсайти, де й будуть проходити більшість проходжень опитувань. Також буде реалізовано функціонал оплати підписки задля можливості активації функцій системи, таких як вбудовування опитувань на інші веб-сайти.

Адміністратори зможуть створювати та змінювати темплейти для груп опитувань та налаштовувати систему.

2.3 **Характеристики користувачів**

У системі існує 3 типи користувачів: зареєстровані користувачі з підпискою і без неї та адміністратори.

Зареєстрованим користувачам без активної підписки доступний такий функціонал системи як: створення опитувань, груп опитувань, налаштування опитувань та зовнішній вигляд.

Зареєстрованим користувачам з активною підпискою доступний весь функціонал системи, таких як: створення опитувань та отримання скриптів для вбудовування в інші вебсайти, перегляд їх статистики.

Незареєстрованим користувачам доступні лише функції логіна та реєстрації.

Адміністратори зможуть оперувати темплейтами які будуть доступні в системі.

2.4 **Загальні обмеження**

Веб-програма клієнта обмежена веб-браузером, встановленим на пристрої користувача. Оскільки існує кілька веб-браузерів і різні пристрої з

Рисунок В.4 – Сторінка 4 специфікації пз

різною роздільною здатністю екрана, інтерфейс, швидше за все, не буде однаковим для кожного з них.

Підключення до Інтернету також є обмеженням для програми. Програма підключається до веб-сервера через Інтернет, і немає способу отримати інформацію без зв'язку між клієнтом і сервером.

Сервер обмежений характеристиками системи баз даних і сервера баз даних. Коли є багато запитів, база даних може бути змушена поставити їх у чергу, що збільшує час, необхідний для отримання даних.

2.5 **Припущення й залежності**

Основне припущення полягає в тому, що більшість сучасних браузерів працюють однаково та підтримують більш-менш новітні технології, такі як HTML5. Якщо браузер не підтримує нові технології, інтерпретація коду та виконання інструкцій можуть призвести до помилок.

3. **Конкретні вимоги**

3.1 **Вимоги до зовнішніх інтерфейсів**

3.1.1 **Інтерфейс користувача**

Зареєстровані користувачі можуть створювати опитування та оплачувати підписку, після чого отримувати скрипти для вбудовування в інші вебсайти, переглядати їх статистику.

3.1.2 **Апаратний інтерфейс**

Для відображення інтерфейсу користувача до пристрою має бути підключений дисплей.

3.1.3 **Програмний інтерфейс**

Щоб мати доступ до веб-системи, користувач повинен використовувати програмне забезпечення для веб-перегляду. Він повинен підтримувати Javascript, HTML5, CSS3.

3.1.4 **Комунікаційний протокол**

Щоб користувач мав доступ до веб-системи, необхідно встановити підключення до Інтернету.

Рисунок В.5 – Сторінка 5 специфікації пз

3.2 **Функції продукту**

3.2.1 Реєстрація

3.2.1.1 Опис

Незареєстровані користувачі (відвідувачі) не мають дозволу на вхід до основного контенту системи, тому що це може призвести до хакерських атак. Щоб розпочати процес реєстрації, користувач повинен натиснути «створити обліковий запис». Це перемістить його до «реєстраційної форми».

3.2.1.2 Вхідні дані

Текст: Електронна пошта, пароль (8-50 символів, з перевіркою складності), ім'я(2-50 символів) прізвище (2-50 символів).

3.2.1.3 Обробка

При натисканні кнопки реєстрації клієнт надсилає запит із усіма вхідними даними на веб-сервер, який спілкується з API.

Веб-сервер перевіряє правильність логіна, пароля та електронної пошти. Якщо правильно – запитує сервер бази даних знайти користувачів із таким логіном або електронною поштою. Якщо знайдено – користувач залишається незареєстрованим і клієнт показує повідомлення з описаною причиною. В іншому випадку пароль буде хешовано, а всі дані, включаючи хеш перевірки, збережуться у базі даних.

3.2.1.4 Вихідні дані

Якщо форма була заповнена успішно, після відправки клієнт видає повідомлення про те, що акаунт було успішно створено та перейде на сторінку логіна.

Якщо форма була заповнена невдало, перед формою з'являється повідомлення з відповідним текстом.

3.2.1.5 Обробка помилок

Повідомлення з помилкою з'явиться перед формою і не буде надіслано, якщо:

- неправильна електронна адреса;
- підтвердження електронною поштою не можна надіслати на адресу електронної пошти;
- користувач із цією електронною адресою вже зареєстрований;
- немає зв'язку з базою даних;
- невалідний пароль.

3.2.1.6 Обмеження

Неавторизовані/незареєстровані користувачі.

Рисунок В.6 – Сторінка 6 специфікації пз

3.2.2 Авторизація

3.2.2.1 Вступ

Неавторизовані користувачі (відвідувачі) можуть увійти, якщо вони мають зареєстрований профіль у системі. Це принесе їм привілеї. Щоб увійти, користувач повинен натиснути «увійти» в заголовку сторінки (в меню). Це перемістить його до «форми входу».

3.2.2.2 Вхідні дані

Текст: email (до 50 символів), пароль (8-50 символів).

3.2.2.3 Обробка

Веб-сервер хешує пароль і запитує базу даних для пошуку користувача з такою електронною поштою. Якщо знайдено та підтверджено користувача, у випадку якщо хешований пароль збігається – вхід виконано успішно. Це переносить дані користувача, що зберігаються в базі даних, до сеансу на стороні клієнта.

3.2.2.4 Вихідні дані

Якщо вхід виконано успішно, система вітає авторизованого користувача.

3.2.2.5 Обробка помилок

Якщо вхід не вдається, перед формою входу з’являється повідомлення з відповідним текстом.

3.2.2.6 Обмеження

Неавторизовані/незареєстровані користувачі.

3.2.2 Створення групи опитувань

3.2.2.1 Вступ

Користувачі зможуть створювати групу опитувань, яка є базою для кожного опитування, тобто для групи вказуються певні правила проходжень опитувань в цій групі, їх зовнішній вигляд, а саме: колір, розмір та шрифт. Після створення, користувачі можуть експортувати скрипт для вбудування опитування в інші вебсайти

3.2.2.2 Вхідні дані

Користувач заповнює форму створення групи опитувань, а саме такі налаштування як:

- дозвіл на повідомлення після завершення опитування;
- дозвіл на зникнення форми після проходження;

Рисунок В.7 – Сторінка 7 специфікації пз

– кількість проходжень опитувань з одного пристрою за день;
– кількість проходження одного опитування одним пристроєм.

Також користувач обирає тип відображення, шаблон, задає певні параметри налаштування

3.2.2.3 Обробка

Після налаштування відображення опитування та правил проходження, юзер підтверджує форму. Після підтвердження форми клієнт відправляє запит на сервер зі створення групи опитувань та повертає статус створення та оновлює список груп опитувань з новоствореною групою.

3.2.2.4 Вихідні дані

Якщо якісь параметри було введено неправильно клієнт видає повідомлення про помилку. Якщо усе пройшло успішно – клієнт відображає нову групу опитувань у списку.

3.2.2.5 Обробка помилок

Якщо якісь параметри було введено неправильно клієнт видає помилку про те, що саме неправильно введено. Якщо було правильно введено усі дані, але була помилка на стороні сервера, то про це видається відповідна помилка. Якщо усе пройшло успішно – клієнт видає повідомлення про успішне створення та відображає нову групу опитувань у списку.

3.2.2.6 Обмеження

Доступно лише для зареєстровних користувачів/

3.2.3 Перегляд набору опитувань

3.2.3.1 Вступ

Для зареєстрованого користуача доступна функція перегляду усіх груп опитувань, для цього на відповідній сторінці буде відображено список груп опитувань, які було створено юзером. Також користувач зможе переглянути вміст кожної групи опитувань натиснувши на певну групу та переглянути її дані: усі опитування групи, тип відображення, правила проходження тощо.

3.2.3.2 Вхідні дані

Для перегляду усіх груп опитувань користувач повинен перейти до відповідної сторінки усіх груп, щоб перейти до інформації певної групи опитувань потрібно натиснути на потрібний елемент групи опитувань, що направить користувача на сторінку інформації про конкретну групу опитувань.

3.2.3.3 Обробка

Рисунок В.8 – Сторінка 8 специфікації пз

Після переходу на сторінку усіх груп опитувань, клієнт відправляє запит на отримання усіх груп опитувань поточного юзера, які відображає після отримання відповіді. Після натискання на конкретну групу опитувань, клієнт відправляє запит до сервера про отримання інформації про задану групу опитувань та відображає дані.

3.2.3.4 Вихідні дані

Усі дані про групи опитувань користувач бачить на екрані гаджета, а саме таку інформацію як: назва групи, тип відображення, кількість опитувань в ній.

Усі дані про конкретну групу опитувань користувач бачить на екрані гаджета, а саме таку інформацію як:

- правила проходження (дозвіл на повідомлення після завершення опитування, дозвіл на зникнення форми після проходження, кількість проходжень опитувань з одного пристрою за день, кількість проходження одного опитування одним пристроєм)
- тип відображення (колір, шрифт, розмір, місце розташування, шаблон)
- список опитувань у групі (назва та базова інформацію про опитування)

3.2.3.5 Обробка помилок

Якщо в групі немає опитувань – повідомляємо користувача про це

3.2.3.6 Обмеження

Авторизовані користувачі.

3.2.4 Видалення набору опитувань

3.2.4.1 Вступ

Користувачу доступна функція видалення групи опитувань, у випадку, коли вона стає йому непотрібна.

3.2.4.2 Вхідні дані

Для видалення групи опитувань користувачу потрібно перейти до сторінки конкретної групи опитувань та натиснути на кнопку видалення, потім виконати підтвердження, натиснувши на відповідну кнопку.

3.2.4.3 Обробка

Після натискання користувачем на кнопку підтвердження видалення, клієнт виконує запит на видалення групи опитувань на сервер, після чого отримує результат запиту.

Рисунок В.9 – Сторінка 9 специфікації пз

3.2.4.4 Вихідні дані

Кнопка видалення буде знаходитися на екрані користувача на сторінці конкретної групи опитувань. Повідомлення про успішне видалення чи помилку буде відображено модальним вікном на екрані користувача.

3.2.4.5 Обробка помилок

У випадку успішного видалення буде відображено відповідне повідомлення, у випадку помилки буде відображена причина помилки.

3.2.4.6 Обмеження

Авторизовані користувачі. Створена група опитувань для видалення

3.2.5 Редагування опитувань

3.2.5.1 Вступ

Користувачу доступна функція редагування опитування. Він може змінювати контент питання, їх переклад, варіанти відповіді та налаштування тагретингу.

3.2.5.2 Вхідні дані

Для редагування опитування, користувач має перейти на сторінку опитування та натиснути на кнопку редагування. Після цього змініти дані про таргетинг та питання і варіанти відповіді.

3.2.5.3 Обробка

Після відправки користувачем форми зі зміненими даними опитування, клієнт відправляє запит на сервер з оновлення даних про опитування, результат операції повертається на клієнт та залежно від результату відображається у користувача на екрані.

3.2.5.4 Вихідні дані

Якщо результат редагування успішний – система каже про успішну операцію користувачу

3.2.5.5 Обробка помилок

У випадку помилки при редагуванні, клієнт видасть помилку про це, відповідно до типу помилки, яку передає сервер.

3.2.5.6 Обмеження

Авторизований користувач, створене опитування для редагуваня.

3.2.6 Перегляд опитувань

3.2.6.1 Вступ

Рисунок В.10 – Сторінка 10 специфікації пз

Користувач має можливість продивитися інформацію про опитування в групі, усі створені опитування та дані про конкретне опитування. У списку він має бачити таку інформацію, як: назва, кількість питань тощо. У деталізованих даних про опитування, користувач має змогу передивитися список питань, їх переклад, таргетинг та дані про групу опитувань, в якій він знаходиться з даними про групу, таку як тип відображення, налаштування тощо.

3.2.6.2 Вхідні дані

Для перегляду списку опитувань, користувачу потрібно перейти на сторінку певної групи опитувань, щоб подивитися спсиок опитувань у групі або на сторінку усіх опитувань, щоб переголянути усі створені. Щоб переглянути дані про конкретне опитування треба натиснути на певне опитування в списку, після чого користувач перейде на сторінку опитування.

3.2.6.3 Обробка

При переході користувача на сторінку усіх опитувань або опитувань у групі, клієнт відправляє відповідний запит на сервер, та, після отримання відповіді – відображає результат на сторінці. При переході до сторінки конкретного опитування, клієнт також посилає запит на АПІ, та відображає дані після отримання результату.

3.2.6.4 Вихідні дані

Усі дані про список опитувань чи дані про конкретне опитування будуть відображені на екрані користувача одразу по завантаженню відповідної сторінки.

3.2.6.5 Обробка помилок

У випадку помилки під час запиту на АПІ, на екрані користувача буде відображено відповідне повідомлення. Якщо у користувача немає опитувань, він не зможе перейти до сторінки конкретного опитування, а на сторінці спсика опитувань він побачить повідомлення про відсутність опитувань.

3.2.6.6 Обмеження

Авторизований користувач.

3.2.7 Створення опитувань

3.2.7.1 Вступ

Користувачу доступна функція додавання опитування. Він може додати питання, їх переклад, варіанти відповіді, переклади, та налаштування тагретингу.

3.2.7.2 Вхідні дані

Рисунок В.11 – Сторінка 11 специфікації пз

Для додавання опитування, користувач має перейти на сторінку усіх опитувань групи, в яку він хоче додати опитування та натиснути на кнопку «Додати». Після цього обрати дані про таргетинг, додати питання, варіанти відповіді, їх переклад.

3.2.7.3 Обробка

Після відправки користувачем форми з даними опитування, клієнт відправляє запит на сервер з додавання нових даних про опитування, результат операції повертається на клієнт та залежно від результату відображається у користувача на екрані.

3.2.7.4 Вихідні дані

Якщо результат додавання успішний – система каже про успішну операцію користувачу та у списку одразу з’явиться нове опитування.

3.2.7.5 Обробка помилок

У випадку помилки при редагуванні, клієнт видасть помилку про це, відповідно до типу помилки, яку передає сервер.

3.2.7.6 Обмеження

Авторизований користувач, доступнийбезкоштовний слот для створення опиитування або куплена підписка.

3.2.8 Видалення опитувань

3.2.8.1 Вступ

Користувачу доступна функція видалення опитування, у випадку, коли воно стає йому непотрібне.

3.2.8.2 Вхідні дані

Для видалення опитування користувачу потрібно перейти до сторінки конкретного опитування та натиснути на кнопку видалення, потім виконати підтвердження, натиснувши на відповідну кнопку.

3.2.8.3 Обробка

Після натискання користувачем на кнопку підтвердження видалення, клієнт виконує запит на видалення опитування на сервер, після чого отримує результат запиту.

3.2.8.4 Вихідні дані

Кнопка видалення буде знаходитися на екрані користувача на сторінці конкретного опитування. Повідомлення про успішне видалення чи помилку буде відображено модальним вікном на екрані користувача.

Рисунок В.12 – Сторінка 12 специфікації пз

3.2.8.5 Обробка помилок

У випадку успішного видалення буде відображено відповідне повідомлення, у випадку помилки буде відображена причина помилки.

3.2.8.6 Обмеження

Авторизовані користувачі. Створене опитування для видалення.

3.2.9 Додавання запитань до опитування

3.2.9.1 Вступ

Користувачу доступна функція додавання питання в опитування. Він може створити нове питання обравши варіанти відповіді та переклади для певних обраних мов.

3.2.9.2 Вхідні дані

Для додавання нового питання користувачу потрібно перейти до сторінки редагування певного опитування та натиснути на кнопку додавання нового питання. Після натискання на кнопку з'являється вікно створення нового питання, де користувач вписує текст питання та обирає відповідні варіанти відповідей та їх типи, а також написати текст питання на обраних мовах опитування.

3.2.9.3 Обробка

Після натискання кнопки збереження питання, клієнт виконує запит на додавання нового питання на сервер, після чого отримує результат запиту на додавання.

3.2.9.4 Вихідні дані

Кнопка додавання питання буде відображена на сторінці опитування. Після успішного додавання, користувач побачить повідомлення про це та також буде відображено щойно створене питання.

3.2.9.5 Обробка помилок

У випадку неуспішного додавання питання користувачеві буде відображена помилка з текстом причини результату запиту.

3.2.9.6 Обмеження

Авторизований користувач, створене опитування.

3.2.10 Видалення запитань з опитування

3.2.10.1 Вступ

Рисунок В.13 – Сторінка 13 специфікації пз

Користувачу доступна функція видалення питання з опитування у разі його неактуальності.

3.2.10.2 Вхідні дані

Для видалення питання користувачу потрібно перейти до сторінки редагування певного опитування та натиснути на кнопку видалення питання. Після натискання на кнопку з'являється вікно підтвердження, де користувач підтверджує видалення.

3.2.10.3 Обробка

Після натискання кнопки підтвердження видалення, клієнт виконує запит на видалення питання у вигляді редагування опитування на сервер, після чого отримує результат запиту.

3.2.10.4 Вихідні дані

Кнопка видалення буде відображена на сторінці редагування опитування біля кожного питання. Після видалення питання, користувач побачить повідомлення та питання зникне з опитування.

3.2.10.5 Обробка помилок

У разі помилки при запиті або неправильно введених даних користувач побачить відповідне повідомлення.

3.2.10.6 Обмеження

Авторизований користувач, створене опитування та питання в ньому.

3.2.11 Редагування запитань в опитуванні

3.2.11.1 Вступ

Користувачу доступна функція редагування питання в опитуваннях. Він може редагувати створене питання змінюючи варіанти відповіді та переклади для певних обраних мов.

3.2.11.2 Вхідні дані

Для редагування створеного питання користувачу потрібно перейти до сторінки редагування певного опитування та натиснути на редагування відповідного питання. Після натискання на кнопку з'являється вікно редагування питання, де користувач змінює текст питання та змінює, додає або видаляє відповідні варіанти відповідей і їх типи, а також змінити чи додати текст питання на обраних мовах опитування.

3.2.11.3 Обробка

Рисунок В.14 – Сторінка 14 специфікації пз

Після натискання кнопки збереження питання, клієнт виконує запит на оновлення даниї певного питання на сервер, після чого отримує результат запиту на зміну даних.

3.2.11.4 Вихідні дані

Кнопка редагування питання буде відображена на сторінці опитування біля певного питання. Після успішного редагування, користувач побачить повідомлення про це, і на сторінці опитування буде відображене щойно відредаговане питання .

3.2.11.5 Обробка помилок

У випадку неуспішного додавання питання користувачеві буде відображена помилка з текстом причини результату запиту

3.2.11.6 Обмеження

Авторизований користувач, створене опитування та питання в ньому.

3.2.12 Оплата підписки за користування послугами

3.2.12.1 Вступ

Щоб користуватися функціями активації опитувань на інших сайтах, користувачу потрібно купити підписку.

3.2.12.2 Вхідні дані

Щоб активувати підписку, користувачу потрібно натиснути на кнопку активації, відбувається редірект на вікно оплати PayPal, в якій користувач вводить свої дані для оплати та натискає на кнопку «сплатити».

3.2.12.3 Обробка

Після натискання кнопки оплати підписки користувача відбувається запит на створення підписки PayPal, після чого відбувається перехід на іншу сторінку з оплатою на PayPal систему і посилається запит на оплату підписки з відповідним станом оплати.

3.2.12.4 Вихідні дані

Після успішно проведеної оплати користувач може побачити стан активації підписки у своєму профілі з відповідною датою кінця статусу підписки.

3.2.12.5 Обробка помилок

У випадку неуспішної оплати, сторонній сервіс повідомляє користувача про результат оплати, при цьому на сайті буде відображена помилка про статус неуспішної оплати підписки.

Рисунок В.15 – Сторінка 15 специфікації пз

3.2.12.6 Обмеження

Авторизований користувач, діюча картка PayPal.

3.2.13 Перегляд статистики по опитуванню

3.2.13.1 Вступ

Користувачу доступна функція перегляду статистики по обраному опитуванню. Він може переглядати та фільтрувати дані, отримані з відгуків клієнтів, використовуючи візуальні діаграми та засоби фільтрації.

3.2.13.2 Вхідні дані

Для перегляду статистики опитування користувачу потрібно перейти до сторінки статистики і натиснути на відповідну кнопку для перегляду статистики, після чого користувач буде переправлений на сторінку з відображеними діаграмами, які користувач може обрати, а також змінити поля та фільтрацію за допомогою відповідних засобів на сторінці.

3.2.13.3 Обробка

Після натискання кнопки на перегляд статистики, клієнт виконує запит на отримання даних по обраному питанню на високонавантажений сервер, після чого формуються відповідні дані для їх перегляду на сторінці.

3.2.13.4 Вихідні дані

Кнопка перегляду статистики опитування відображена на сторінці опитування біля основної інформації. Після успішного отримання даних, користувач побачить сторінку з відображеними даними у вигляді діаграм, і на цій ж сторінці користувач зможе відфільтрувати дані за допомогою наявних інструментів.

3.2.13.5 Обробка помилок

У випадку неуспішного отримання даних опитування користувачеві буде відображена помилка з текстом причини результату запиту

3.2.13.6 Обмеження

Авторизований користувач, опитування та відгуки на це опитування.

3.2.14 Перегляд таргетингів

3.2.14.1 Вступ

Користувач може переглядати усі створені таргетинги для опитувань двома способами: при створенні опитування, коли користувач обирає таргетинг, та окремо зайшовши на сторінку усіх таргетингів.

3.2.14.2 Вхідні дані

Рисунок В.16 – Сторінка 16 специфікації пз

Перегляд таргетінгів можливий після переходу на сторінку усіх таргетингів та під час створення опитування у вигляді форми.

3.2.14.3 Обробка

В обох випадках процес обробки однаковий, а саме: клієнт відправляє запит про усі створені таргетинги на сервер та, після отримання відповіді, відображає список таргетингів.

3.2.14.4 Вихідні дані

Список таргетінгів буде відображено на екрані користувача під час створення опитування та після переходу на сторінку усіх тагретингів.

3.2.14.5 Обробка помилок

У випадку помилок клієнт повідомить користувача про те, що сталася помилка з текстом відповідним до типу помилки. Якщо не було створено жодних таргетингів, то про це буде повідомлено користувача.

3.2.14.6 Обмеження

Авторизований користувач

3.2.15 Створення таргетингів

3.2.15.1 Вступ

Для налаштування аудиторії, на яку буде націлено опитування, користувач зможе створювати таргетинги, обравши цільові країни.

3.2.15.2 Вхідні дані

Для створення таргетингу користувачу потрібно натиснути на відповідну кнопку та заповнити форму, де потрібно обрати цільові країни та придумати назву таргетингу.

3.2.15.3 Обробка

Під час підтвердження та відправки форми створення таргетингу, клієнт посилає запит на сервер, де створюється таргетинг, та відображає на клієнті результат залежно від відповіді.

3.2.15.4 Вихідні дані

У випадку успішного створення, про це буде повідомлено користувача.

3.2.15.5 Обробка помилок

Якщо під час створення було отримано помилку – вона відображається на екрані користувача.

3.2.15.6 Обмеження

Рисунок В.17 – Сторінка 17 специфікації пз

Авторизований користувач.

3.3 **Класи/Об’єкти**

3.3.1 User

3.4.1.1 Атрибути

- Email – text
- Hashed password – text
- First name – text
- Last name – text
- Role – text

3.3.2 SurveyUnit

3.3.2.1 Атрибути

- Name – text
- One survey take per device – int
- Maximum surveys per device – int
- Message after no surveys – bool
- Hide after no surveys – bool

3.3.3 UnitAppearance

3.3.3.1 Атрибути

- Type – text
- Params – text
- State – text
- Name – text
- Template code – text
- Default params – text

3.3.4 Survey

3.3.4.1 Атрибути

- Name – text
- DateBy – text

Рисунок В.18 – Сторінка 18 специфікації пз

- Params – text
- Countries – text
- QuestionType – text
- OrderNumber – int
- Translate – text

3.4 **Нефункціональні вимоги**

3.4.1 **Продуктивність**

Будь-яка сторінка повинна бути завантажена менш ніж за 5 секунд;

3.4.2 **Надійність**

Система не повинна виходити з ладу, коли користувач вводить неправильні параметри.

3.4.3 **Доступність**

Система повинна бути доступною для використання з будь-якого місця.

3.4.4 **Безпека**

- Конфіденційні дані повинні бути захищені;
- Паролі мають бути хешованими.

3.4.5 **Придатність до обслуговування**

Повинна бути панель адміністратора з реалізованими деякими функціональними вимогами.

3.4.6 **Портативність**

Сервіс має бути доступним для використання з мобільних браузерів.

3.5 **Зворотні вимоги**

Веб-сервіс не надає жодних додаткових конфіденційних даних користувача, окрім тих, які вводить користувач.

3.6 **Обмеження дизайну**

- всі користувацькі інтерфейси повинні бути перекладені українською мовою;
- колір # 2196F3 використовується для панелей (наприклад, головного меню);

Рисунок В.19 – Сторінка 19 специфікації пз

- колір #FFFFFF для тла або тексту, якщо текст написаний на темних кольорах;
- колір #000000 для тексту, якщо він написаний на світлих кольорах;
- Pure Survey використовує шрифти: Impact для заголовків, Century Gothic для підзаголовків;
- дизайн має бути мінімалістичним.

3.7 **Вимоги до логічної бази даних**

Будемо використовувати MS SQL для основної функціональності системи, оскільки важлива безпека та цілісність даних та СКБД Vertica для збереження даних стстистики опитувань.

Рисунок В.20 – Сторінка 20 специфікації пз

ДОДАТОК Г

Приклад програмного коду (PaymentController)

```
using Diploma.Backend.Application.Dto.Request;
using Diploma.Backend.Application.Dto.Response;
using Diploma.Backend.Application.Helpers;
using Diploma.Backend.Application.Services;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json.Linq;
using System.Net;
using System.Security.Claims;

namespace Diploma.Backend.API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class PaymentController : ControllerBase
    {
        private readonly IPaymentService _payPalService;
        public PaymentController(IPaymentService payPalService)
        {
            _payPalService = payPalService;
        }

        [HttpPost]
        [Route("create-subscription")]
        public async Task<IActionResult> CreateSubscription([FromBody]
PayPalSubscriptionRequestShort request)
        {
            var identity = HttpContext.User.Identity as ClaimsIdentity;
            var user = CurrentUserRetriever.GetCurrentUser(identity);
            if (user.Error != null)
            {
                return Unauthorized(user);
            }

            var response = await _payPalService.CreateSubscription(request,
user.Data);
            if (response.Error != null)
            {
                return BadRequest(response);
            }
            return Ok(response);
        }

        [HttpPost]
        [Route("cancel/{id}")]
        public async Task<IActionResult> CancelSubscription([FromBody]
PayPalCancelSubscriptionRequest request, [FromRoute] string id)
        {
            var response = await _payPalService.CancelSubscription(id, request);
            if (response.Error != null)
            {
                return BadRequest(response);
            }
            return Ok(response);
        }

        [HttpGet]
        [Route("subscription/{id}")]
        public async Task<IActionResult> GetSubscription([FromRoute] string id)
        {
            var response = await _payPalService.GetSubscription(id);
            if (response.Error != null)
```

```
            {
                return BadRequest(response);
            }
            return Ok(response);
        }

        [HttpPost]
        [Route("activate/{id}")]
        public async Task<IActionResult> ActivateSubscription([FromRoute] string
id, [FromBody] ActivateSubscriptionRequest request)
        {
            var response = await _payPalService.ActivateSubscription(id,
request);
            if (response.Error != null)
            {
                return BadRequest(response);
            }
            return Ok(response);
        }

        [HttpPost]
        [Route("paypal-webhook")]
        public async Task<IActionResult> ReceiveWebhook()
        {
            try
            {
                string requestBody;
                using (var reader = new StreamReader(Request.Body))
                {
                    requestBody = await reader.ReadToEndAsync();
                }

                var request = JObject.Parse(requestBody);
                var eventType = request.Value<string>("event_type");
                var subscriptionId =
request.SelectToken("resource.id")?.Value<string>();

                if (eventType == "BILLING.SUBSCRIPTION.EXPIRED" || eventType ==
"BILLING.SUBSCRIPTION.CANCELLED")
                {
                    HandleSubscriptionExpiration(subscriptionId);
                }
                return Ok("Webhook received and processed");
            }
            catch (Exception)
            {
                return Ok("Webhook received and processed");
            }
        }

        private void HandleSubscriptionExpiration(string? subscriptionId)
        {
            _payPalService.HandleExpiration(subscriptionId);
        }
    }
}
```

Приклад програмного коду (PaymentService)

```
using Diploma.Backend.Application.Dto.Request;
using Diploma.Backend.Application.Dto.Response;
using Diploma.Backend.Application.Helpers;
using Diploma.Backend.Application.Mappers;
using Diploma.Backend.Application.Repositories.Payment;
```

```
using Diploma.Backend.Application.Repositories.Payment.Proxies;
using Diploma.Backend.Application.Services;
using Diploma.Backend.Domain.Common;
using Diploma.Backend.Domain.Enums;
using Diploma.Backend.Domain.Models;
using Microsoft.Extensions.Configuration;
using System.Text;

namespace Diploma.Backend.Application.Services.Payment.impl
{
    public class PaymentService : IPaymentService
    {
        private readonly IPaymentProxy _payPalProxy;
        private readonly IPaymentRepository _paymentRepository;

        public PaymentService(IPaymentProxy paymentProxy, IPaymentRepository
paymentRepository)
        {
            _payPalProxy = paymentProxy;
            _paymentRepository = paymentRepository;
        }

        public async Task<BaseResponse<PayPalCancelSubscriptionResponse>>
CancelSubscription(string id, PayPalCancelSubscriptionRequest request)
        {
            var token = await _payPalProxy.GetTokenAsync();
            var cancelResponse = await _payPalProxy.CancelSubscriptionAsync(id,
request, token.access_token);

            return
BaseResponseGenerator.GenerateValidBaseResponse(cancelResponse);
        }

        public async Task<BaseResponse<PayPalSubscriptionResponse>>
CreateSubscription(PayPalSubscriptionRequestShort request, User jwtUser)
        {
            var existingSubscription =
_paymentRepository.GetSubscriptionByUserId(jwtUser.Id);
            if (existingSubscription != null)
                return
BaseResponseGenerator.GenerateBaseResponseByErrorMessage<PayPalSubscriptionRespo
nse>(ErrorCodes.SubscriptionAlreadyExists.ToString());

            var token = await _payPalProxy.GetTokenAsync();
            var product = await
_payPalProxy.CreateProductAsync(token.access_token);
            var plan = await _payPalProxy.CreatePlanAsync(token.access_token,
product.Id);
            var subscrResponse = await
_payPalProxy.CreateSubscriptionAsync(request, token.access_token, plan.Id);
            var user = _paymentRepository.GetUserById(jwtUser.Id);

            if (user == null)
                 return
BaseResponseGenerator.GenerateBaseResponseByErrorMessage<PayPalSubscriptionRespo
nse>(ErrorCodes.UserNotFound.ToString());

            var subscription = new Subscription
            {
                User = user,
                UserId = user.Id,
                SubscriptionId = subscrResponse.id,
            };

            await _paymentRepository.AddSubscriptionAsync(subscription);
```

```
            return
BaseResponseGenerator.GenerateValidBaseResponse(subscrResponse);
        }

        public async Task<BaseResponse<PayPalSubscriptionResponse>>
GetSubscription(string id)
        {
            var token = await _payPalProxy.GetTokenAsync();
            var subscrResponse = await _payPalProxy.GetSubscriptionAsync(id,
token.access_token);
            return
BaseResponseGenerator.GenerateValidBaseResponse(subscrResponse);
        }

        public async Task<BaseResponse<PayPalSubscriptionResponse>>
ActivateSubscription(string id, ActivateSubscriptionRequest request)
        {
            var token = await _payPalProxy.GetTokenAsync();
            var subscrResponse = await _
payPalProxy.ActivateSubscriptionAsync(id, request, token.access_token);
            var subscription = _paymentRepository.GetSubscriptionById(id);
            _paymentRepository.UpdateSubscription(subscription);
            return B
aseResponseGenerator.GenerateValidBaseResponse(subscrResponse);
        }

        public async Task HandleExpiration(string id)
        {
            var subscription = _paymentRepository.GetSubscriptionById(id);
            if (subscription != null)
            {
                await _paymentRepository.DeleteSubscriptionAsync(subscription);
            }
        }
    }
}
```

Приклад програмного коду (PayPalProxy)

```
using Diploma.Backend.Application.Dto.Request;
using Diploma.Backend.Application.Dto.Response;
using Diploma.Backend.Application.Repositories.Payment.Proxies;
using Diploma.Backend.Infrastructure.PayPal.Facades;
using Diploma.Backend.Infrastructure.PayPal.Helpers;
using Diploma.Backend.Infrastructure.PayPal.Models;
using Microsoft.Extensions.Configuration;
using System.Text;

namespace Diploma.Backend.Infrastructure.PayPal.Proxies.impl
{
    public class PayPalProxy : IPaymentProxy
    {
        private readonly IPayPalFacade _payPalFacade;
        private readonly PayPalConfig _configuration;
        private readonly IConfiguration _configurationExtension;

        public PayPalProxy(IConfiguration configuration, IPayPalFacade
payPalFacade)
        {
            _configuration = new PayPalConfig(configuration);
            _payPalFacade = payPalFacade;
            _configurationExtension = configuration;
        }
```

```
        public async Task<PayPalAccessTokenResponse> GetTokenAsync()
        {
            var tokenAuthBasic =
Convert.ToBase64String(Encoding.ASCII.GetBytes($"{_configuration.ClientId}:{_con
figuration.ClientSecret}"));
            var tokenResponse = _payPalFacade.Post<PayPalAccessTokenResponse>(
                _configuration.TokenUrl,
                new Dictionary<string, string> { { "grant_type",
"client_credentials" } },
                null,
                new Dictionary<string, string> { { "Authorization", $"Basic {
tokenAuthBasic}" } },
                "application/x-www-form-urlencoded"
            );

            return await Task.FromResult(tokenResponse);
        }

        public async Task<PayPalSubscriptionResponse>
CreateSubscriptionAsync(PayPalSubscriptionRequestShort request, string token,
string planId)
        {
            var requestPayPal = new
PayPalRequestGenerator(_configurationExtension).GenerateSubscriptionRequest();
            requestPayPal.plan_id = planId;
            requestPayPal.start_time = DateTime.UtcNow.AddMinutes(1);
            requestPayPal.application_context.return_url = request.ReturnUrl;
            requestPayPal.application_context.cancel_url = request.CancelUrl;
            return await
Task.FromResult(_payPalFacade.Post<PayPalSubscriptionResponse>(
                _configuration.CreateSubscriptionUrl,
                requestPayPal,
                null,
                new Dictionary<string, string> { { "Authorization", $"Bearer
{token}" } }
            ));
        }

        public async Task<PayPalSubscriptionResponse>
GetSubscriptionAsync(string id, string token)
        {
            var url = _configuration.GetSubscriptionUrl.Replace("{id}", id);
            return await
Task.FromResult(_payPalFacade.Get<PayPalSubscriptionResponse>(
                url,
                null,
                null,
                new Dictionary<string, string> { { "Authorization", $"Bearer
{token}" } }
            ));
        }

        public async Task<PayPalCancelSubscriptionResponse>
CancelSubscriptionAsync(string id, PayPalCancelSubscriptionRequest request,
string token)
        {
            var cancelSubscriptionUrl =
_configuration.CancelSubscriptionUrl.Replace("{id}", id);
            return await
Task.FromResult(_payPalFacade.Post<PayPalCancelSubscriptionResponse>(
                cancelSubscriptionUrl,
                request,
                null,
                new Dictionary<string, string> { { "Authorization", $"Bearer
{token}" } }
            ));
        }
```

```
        public async Task<PayPalSubscriptionResponse>
ActivateSubscriptionAsync(string id, ActivateSubscriptionRequest request, string
token)
        {
            var url = _configuration.ActivateSubscriptionUrl.Replace("{id}",
id);
            return await
Task.FromResult(_payPalFacade.Post<PayPalSubscriptionResponse>(
                url,
                request,
                null,
                new Dictionary<string, string> { { "Authorization", $"Bearer
{token}" } }
            ));
        }

        public async Task<PayPalPlanResponse> CreatePlanAsync(string token,
string productId)
        {
            var request = new
PayPalRequestGenerator(_configurationExtension).GeneratePlanRequest();
            request.ProductId = productId;
            return await Task.FromResult(_payPalFacade.Post<PayPalPlanResponse>(
                _configuration.CreatePlanUrl,
                request,
                null,
                new Dictionary<string, string> { { "Authorization", $"Bearer
{token}" } }
            ));
        }

        public async Task<PayPalProductResponse> CreateProductAsync(string
token)
        {
            var requestPayPal = new
PayPalRequestGenerator(_configurationExtension).GenerateProductRequest();
            return await
Task.FromResult(_payPalFacade.Post<PayPalProductResponse>(
                _configuration.CreateProductUrl,
                requestPayPal,
                null,
                new Dictionary<string, string> { { "Authorization", $"Bearer
{token}" } }
            ));
        }
    }
}
```

Приклад програмного коду (PayPalFacade)

```
using RestSharp;
using Microsoft.Extensions.Configuration;
using Diploma.Backend.Infrastructure.PayPal.Models;
using Newtonsoft.Json;
using Diploma.Backend.Domain.Enums;

namespace Diploma.Backend.Infrastructure.PayPal.Facades.impl
{
    public class PayPalFacade : IPayPalFacade
    {
        private readonly IRestClient restClient;

        public PayPalFacade(IConfiguration provideConfiguration,
                            IRestClient restClient)
```

```
        {
            this.restClient = restClient;
            var configuration = new PayPalConfig(provideConfiguration);
            restClient.Timeout = configuration.TimeOut;
            restClient.BaseUrl = new Uri(configuration.BaseUrl);
        }

        public T Post<T>(string url,
                         object body = null,
                         Dictionary<string, string> urlParams = null,
                         Dictionary<string, string> headers = null,
                         string contentType = "application/json"
                         ) where T : new()
        {
            return Action<T>(url, Method.POST, body, urlParams, headers,
contentType);
        }

        public T Get<T>(string url,
                         object body = null,
                         Dictionary<string, string> urlParams = null,
                         Dictionary<string, string> headers = null,
                         string contentType = "application/json"
                         ) where T : new()
        {
            return Action<T>(url, Method.GET, body, urlParams, headers,
contentType);
        }

        public T Patch<T>(string url,
                   object body = null,
                   Dictionary<string, string> urlParams = null,
                   Dictionary<string, string> headers = null,
                   string contentType = "application/json"
                   ) where T : new()
        {
            return Action<T>(url, Method.PATCH, body, urlParams, headers,
contentType);
        }

         private T Action<T>(string url, Method method, object body,
Dictionary<string, string> queryParams, Dictionary<string, string> headers,
string contentType) where T : new()
        {
            var request = new RestRequest(url, method);

            AddQueryParams(ref request, queryParams);
            AddHeaders(ref request, headers);
            AddBody(ref request, body, contentType);

            var response = restClient.Execute(request);

            if (!response.IsSuccessful)
            {
                throw new
Exception(ErrorCodes.PayPalCommunicationError.ToString());

              retun string.IsNullOrEmpty(response.Content) ? new T() :
JsonConvert.DeserializeObject<T>(response.Content);
        }

         private void AddBody(ref RestRequest request, object body, string
contentType)
        {
            if (body != null)
            {
                switch (contentType)
```

```
                {
                    case "application/json":
                        AddJsonBody(ref request, body);
                        break;
                    case "application/x-www-form-urlencoded":
                        AddFormBody(ref request, body as Dictionary<string,
string>);
                        break;
                }
            }
        }

         private void AddFormBody(ref RestRequest request, Dictionary<string,
string> body)
        {
            if (body != null && body.Any())
            {
                foreach (var parameter in body)
                {
                    request.AddParameter(parameter.Key, parameter.Value);
                }
            }

              request.AddHeader("Content-Type", "application/x-www-form-
urlencoded");
        }

         private void AddJsonBody(ref RestRequest request, object body)
        {
            if (body != null)
            {
                var jsonBody = body is string ? body :
JsonConvert.SerializeObject(body);
                   request.AddParameter("application/json", jsonBody,
ParameterType.RequestBody);
            }
            request.AddHeader("Content-Type", "application/json");
        }

         private void AddQueryParams(ref RestRequest request, Dictionary<string,
string> urlParams)
        {
            if (urlParams != null)
            {
                foreach (var key in urlParams.Keys)
                {
                    request.AddParameter(key, urlParams[key],
ParameterType.QueryString);
                }
            }
        }

         private void AddHeaders(ref RestRequest request, Dictionary<string,
string> headers)
        {
            if (headers != null)
            {
                foreach (var key in headers.Keys)
                {
                    request.AddParameter(key, headers[key],
ParameterType.HttpHeader);
                }
            }
        }
    }
}
```