Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Центр післядипломної освіти
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)
Програмна система обліку роботи мережі АЗС
(тема)

Виконав:
студент 2 курсу, групи ПЗПІпз-22-1
Мезенцева Д.С.
(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 Русакова Н.Є.
(посада, прізвище, ініціали)

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

2024 р.

Харківський національний університет радіоелектроніки

Центр післядипломної освіти
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 Мезенцевій Дар'ї Сергіївні
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система обліку роботи мережі АЗС
Затверджена наказом по університету від наказ 93Стз від 17.06.2024
2. Термін подання студентом роботи до екзаменаційної комісії 20.07.2024
3. Вихідні дані до роботи Розробити веб-застосунок для управління АЗС з авторизацією користувачів, відновленням паролю, пошуком, додаванням та редагуванням співробітників, додаванням транзакцій та звітом по ним, з цінами на пальне та знижками з використанням технології Blazor та мови програмування C#, середовище розробки VS Code
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	26.05.2024	*виконано*
2	Створення специфікації ПЗ	01.06.2024	*виконано*
3	Проектування ПЗ	15.06.2024	*виконано*
4	Розробка ПЗ	18.06.2024	*виконано*
5	Тестування ПЗ	20.06.2024	*виконано*
6	Оформлення пояснювальної записки	05.07.2024	*виконано*
7	Підготовка презентації та доповіді	10.07.2024	*виконано*
8	Попередній захист	18.07.2024	*виконано*
9	Нормоконтроль, рецензування	18.07.2024	*виконано*
10	Здача роботи у електронний архів	18.07.2024	*виконано*
11	Допуск до захисту у зав. кафедри	19.07.2024	*виконано*

Дата видачі завдання 06 травня 2024р.

Студент (ка)_____ Мезенцева Д.С.
(підпис)

Керівник роботи _____ доц. кафедри ПІ Русакова Н.Є.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 68 стор., 20 рис., 3 табл., 10 джерел.

АЗС, ВЕБ ЗАСТОСУНОК, КОЛОНКА, ПАЛЬНЕ, СТАНЦІЯ, ТРАНЗАКЦІЯ, BLAZOR, C#, POSTGRESQL

Об’єктом дослідження є процес оптимізації обліку та управління АЗС.

Об’єктом розробки є веб-застосунок для обліку роботи мережі АЗС.

Метою розробки є створення програмного застосунку для обліку роботи АЗС, управління транзакціями та формування звітності.

Методи розробки базуються на мові програмування C#, технології .NET Core, технології Blazor[1] та бази даних PostgreSQL.

У результаті був проведений аналіз предметної області, виявлені основні проблеми ринку, проведено проектування системи, створено веб-застосунок для співробітників АЗС, проведено тестування створеного рішення.

GAS STATION, WEB APPLICATION, COLUMN, FUEL, STATION, TRANSACTION, BLAZOR, C3, POSTGRESQL

The object of the research is the process of optimization of gas station accounting and management.

The object of development is a web application for accounting for the gas station network.

The goal of thework is to develop a web application for gas station operation accounting, transaction management and reporting.

Development methods are based on C# programming language, .NET Core technology, Blazor technology[1] and PostgreSQL database.

As a result, an analysis of the subject area was analyzed, the main market problems were identified, the system was designed, a web application was developed and the created solution was tested.

Я, Мезенцева Дар’я Сергіївна, студентка гр. ПЗПІпз-22-1, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система обліку роботи мережі АЗС», 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 ознайомлена з 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

ПЕРЕЛІК СКОРОЧЕНЬ

АЗС - Автозаправна станція.

CRUD - Create, Read, Update, Delete

API - Application Programming Interface

HTTP/HTTPS - Hypertext Transfer Protocol / Hypertext Transfer Protocol Secure

SDLC – Software Development Lifecycle

REST – Representational State Transfer

TCP – Transmission Control Protocol

UML – Unified Modelling Language

ВСТУП

Сьогоднішні АЗС стикаються зі складними викликами, пов'язаними з необхідністю точного обліку руху пального та ефективного управління запасами. Традиційні методи обліку, що ґрунтуються на ручних операціях та паперових документах, часто виявляються неефективними та вразливими до помилок. Також потрібно багато місця щоб зберігати всі папери, що в свою чергу уповільнює та ускладнює процес пошуку потрібної інформації.

В цьому контексті програмні системи обліку надають можливість автоматизувати багато процесів. Це дозволяє забезпечити точний облік кількості проданого пального, виробничих операцій та фінансових транзакцій. Така програма для обліку АЗС може централізувати всю інформацію, зробити її легко доступною для користувачів та зменшити ризик людських помилок. Використання цієї програми знижує потребу у великих фізичних архівах, оскільки вся інформація зберігається в електронному вигляді, що значно спрощує її зберігання та доступ до неї. Це не лише економить простір, але й сприяє більш ефективному управлінню даними, дозволяючи швидко знаходити необхідні записи та генерувати звіти. Крім того, програмні системи можуть впроваджувати аналітичні засоби для виявлення тенденцій у споживанні, прогнозування попиту та оптимізації запасів. Це дозволяє АЗС реагувати на зміни у ринкових умовах швидше та ефективніше, забезпечуючи конкурентні переваги.

Метою роботи є розробка програмної системи, яка складається з веб-застосунку. Веб-застосунок дозволяє користувачу вести облік АЗС станцій, здійснювати транзакції та управляти даними про пальне через зручний інтерфейс. Для розробки продукту використовувалася технологія Blazor та мова програмування C#. Відповідно до технологій використовувалось середовище розробки Visual Studio.

Основне завдання веб-застосунку – надання користувачу можливості вести облік станцій АЗС, включаючи управління інформацією про колонки, пальне та транзакції.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Аналіз сучасних АЗС (Автозаправних станцій) вимагає огляду різних аспектів, таких як технологічні інновації, конкуренція, екологічна стійкість та споживчі звички. В умовах війни АЗС стало краєугольним каменем, тому що машини стали майже не єдиним засобом пересування, особливо в дальні кутки України. Важливо забезпечувати пальним як цивільних, так і військових.

Автозаправні станції (АЗС) відіграють критичну роль у забезпеченні транспорту пальним. В сучасних умовах важливо мати ефективну систему управління, яка дозволяє контролювати запаси пального, управляти фінансовими транзакціями, підтримувати обладнання та забезпечувати аналітику для прийняття рішень. Ці завдання стають дедалі важливішими з огляду на зростання конкуренції та вимог до ефективності операцій.

Основними аспектами предметної галузі являються фінансові транзакції, аналітика та звітність, клієнтське обслуговування та управління співробітниками. Нижче розглянемо детальніше кожен з цих аспектів.

Управління фінансовими транзакціями є ключовим аспектом роботи АЗС. Це включає облік продажів пального, що дозволяє точно відстежувати обсяги реалізованого продукту та доходи. Крім того, необхідно ефективно управляти грошовими потоками, щоб забезпечити стабільність фінансових операцій та мінімізувати ризики. Генерація фінансових звітів та аналіз прибутковості допомагає керівництву приймати обґрунтовані рішення щодо оптимізації операцій та підвищення рентабельності бізнесу.

Аналітика та звітність є важливими компонентами управління АЗС, що дозволяють детально аналізувати продажі та визначати найбільш популярні види пального. Виявлення тенденцій у споживанні пального сприяє більш точному прогнозуванню попиту та плануванню закупівель. Підготовка звітів для керівництва забезпечує прозорість операційної діяльності та підтримує прийняття стратегічних рішень, спрямованих на розвиток бізнесу та підвищення його конкурентоспроможності.

Забезпечення високого рівня обслуговування клієнтів є пріоритетним завданням для АЗС. Впровадження програм лояльності та акцій допомагає залучити та утримати клієнтів, підвищуючи їх задоволеність та довіру до сервісу. Це також сприяє зростанню обсягів продажів та поліпшенню іміджу компанії. Орієнтація на потреби клієнтів і надання якісного сервісу є важливими факторами успіху в конкурентному середовищі.

Управління персоналом на АЗС включає кілька важливих аспектів. По-перше, необхідно ефективно планувати та координувати зміни співробітників, щоб забезпечити безперебійну роботу станції та оптимальне використання трудових ресурсів. Прив'язка співробітників до конкретних АЗС дозволяє краще контролювати їхню роботу та забезпечувати необхідну підтримку. Ведення обліку робочого часу та ефективності роботи персоналу допомагає оцінювати продуктивність та вживати заходів для її підвищення, що в кінцевому рахунку сприяє підвищенню загальної ефективності роботи АЗС.

Аналіз основних аспектів предметної галузі для управління автозаправними станціями (АЗС) демонструє важливість комплексного підходу до організації роботи. Кожен з аспектів – управління фінансовими транзакціями, аналітика та звітність, клієнтське обслуговування, а також управління співробітниками – відіграє критичну роль у забезпеченні ефективності та успіху бізнесу.

Впровадження нової веб-системи управління АЗС, що враховує всі ці аспекти, дозволить значно покращити ефективність операцій, підвищити задоволеність клієнтів та сприяти зростанню прибутковості компанії.

1.2 Виявлення та вирішення проблем

На ринку існує кілька систем для управління АЗС, що пропонують різні набори функцій для вирішення описаних вище завдань. Розглянемо їх у таблиці порівнянь аналогів – таблиці 1.

На ринку існує кілька потужних рішень для управління мережею АЗС, кожне з яких має свої переваги та недоліки.

Таблиця 1 – Порівняння аналогів систем управління АЗС (створено самостійно)

Назва системи	Переваги	Недоліки
Petrosoft	Комплексні рішення для управління АЗС, інтеграція з обладнанням, фінансові звіти та аналітика	Може бути складним у налаштуванні, висока вартість
FuelForce	Контроль доступу до пального, моніторинг витрат, звіти про транзакції	Обмежений функціонал для малих АЗС, можливі проблеми з інтеграцією
PDI (Professional Data Solutions)	Управління запасами пального, динамічне ціноутворення, розширена аналітика продажів	Висока вартість, може вимагати складних інтеграцій
OPW Fuel Management System	Моніторинг резервуарів, автоматизація транзакцій, інтеграція з POS-системами	Обмежений користувацький інтерфейс, може потребувати додаткового навчання
SkyBitz Petroleum Logistic	Моніторинг рівня пального в реальному часі, оптимізація логістики, аналітика	Висока вартість, обмежений функціонал для невеликих мереж АЗС

Petrosoft і PDI надають комплексні рішення з розширеними можливостями аналітики та інтеграцією, але вони можуть бути дорогими та складними в налаштуванні. На прикладі Petrosoft ми бачимо продаж трьох продуктів для однієї АЗС, такі як управління бек-офіс системою, додатковий сервіс для обробки рахунків-фактур та програму аналітики запобігання втратам (див рис. 1.1). Це може бути забагато для однієї АЗС. FuelForce і OPW Fuel Management Systems пропонують хороші можливості для моніторингу та управління, але їхній функціонал може бути обмеженим для малих АЗС і вимагати додаткового навчання. FuelForce використовують додаткові трекінгові системи для підрахунку

пального, а також для обліку електричних зарядних станцій, які останнім часом сталі популярні (див рис. 1.2).

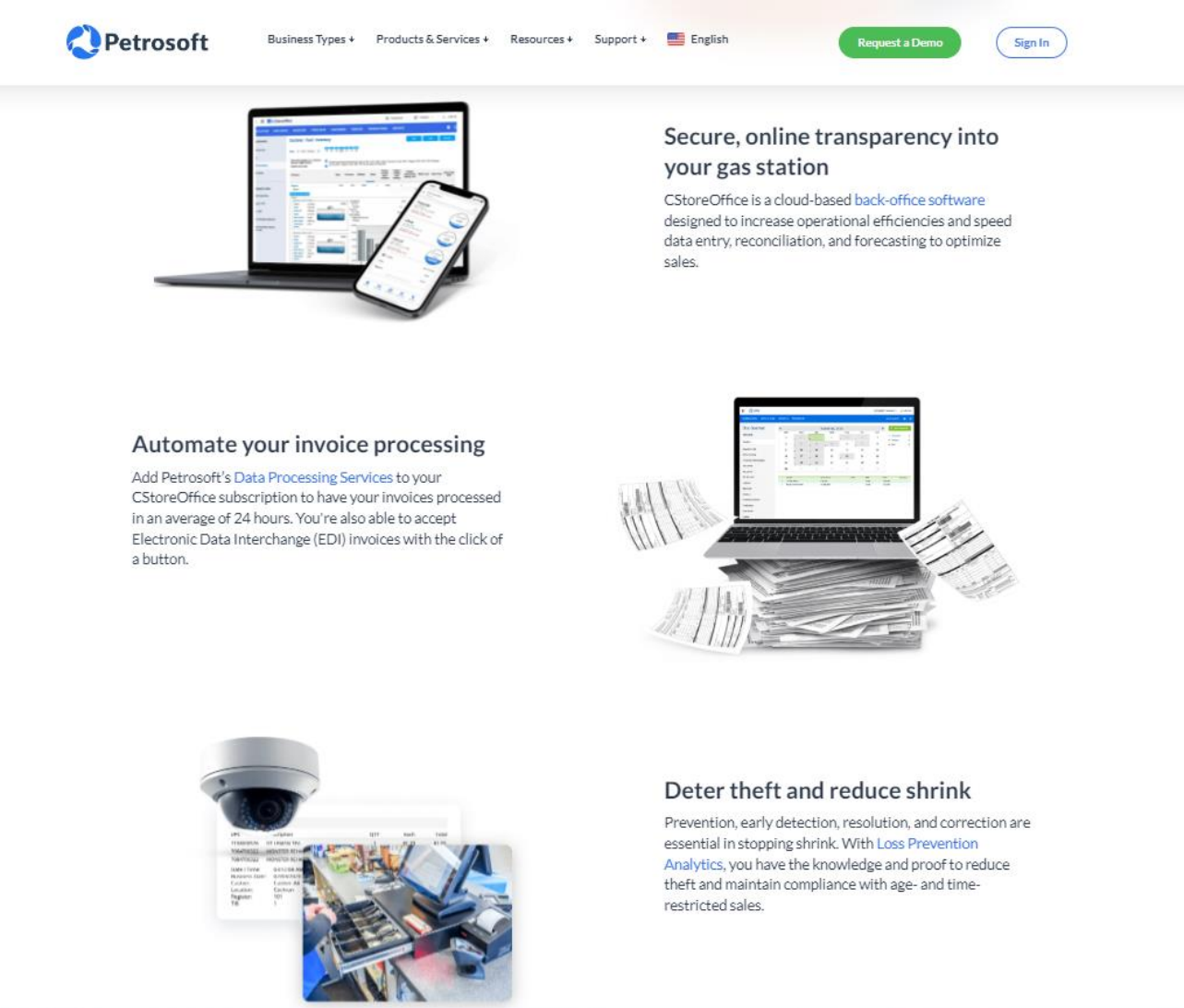


Рисунок 1.1 – Сторінка з пропозиціями для АЗС на сайті Petrosoft (за даними [2])

SkyBitz Petroleum Logistics спеціалізується на моніторингу та оптимізації логістики, що може бути корисно для великих мереж, але може бути надмірним для невеликих станцій.

Виділимо 4 критерії – управління фінансовими транзакціями, аналітика та звітність, клієнтське обслуговування та управління співробітниками. Оцінимо задоволення цими критеріями в таблиці порівняння (див. табл.2).

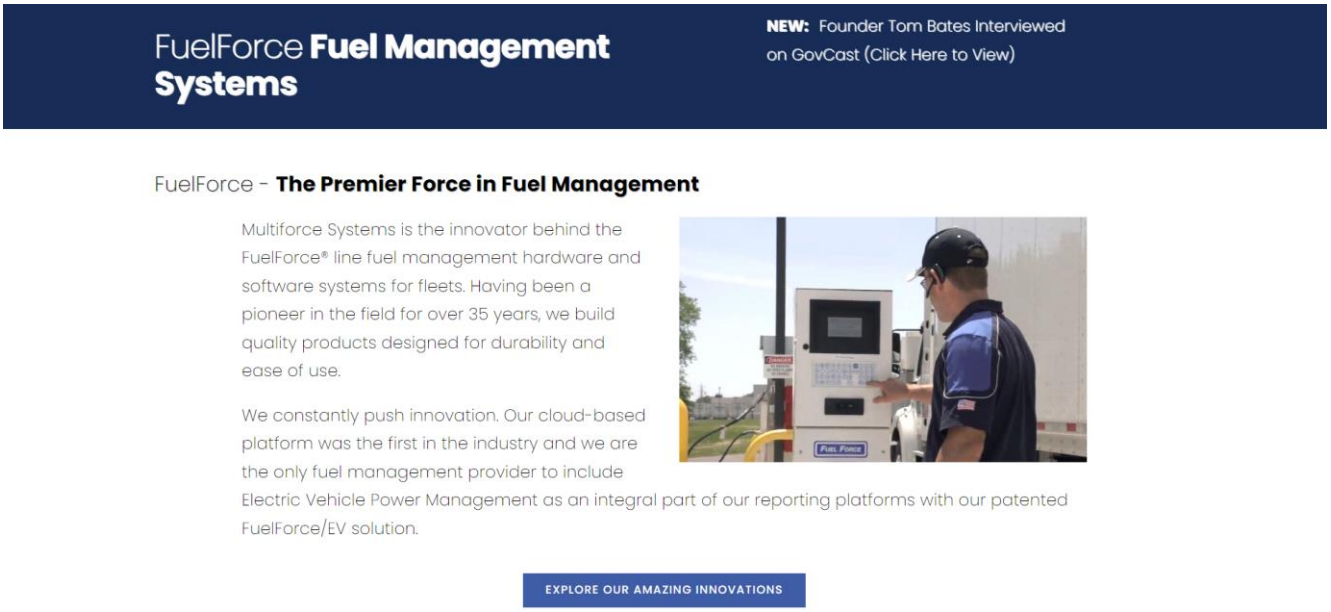


Рисунок 1.2 – Система обліку пального FuelForce (за даними [3])

Таблиця 2 – Порівняння задоволення критеріям систем управління АЗС (таблиця створена самостійно)

Назва компанії	Управління фінансовими транзакціями	Аналітика та звітність	Клієнтське обслуговування	Управління співробітниками
Petrosoft	Високий	Високий	Середній	Середній
FuelForce	Високий	Середній	Низький	Низький
PDI	Високий	Високий	Середній	Середній
OPW	Високий	Високий	Низький	Низький
SkyBitz	Середній	Середній	Низький	Низький

Аналіз існуючих рішень показує, що на ринку є кілька потужних систем управління АЗС, які забезпечують високий рівень управління фінансовими транзакціями та аналітикою. Однак, більшість з них мають обмежені можливості для управління клієнтським обслуговуванням і персоналом. Це підкреслює потребу в розробці нового веб-застосунку, який би враховував ці аспекти та пропонував більш комплексне рішення з функціоналом для управління співробітниками, оптимізації операційних процесів та покращення взаємодії з клієнтами.

Кожна з розглянутих систем, таких як Petrosoft, FuelForce, PDI, OPW Fuel Management Systems та SkyBitz Petroleum Logistics, пропонує широкий спектр функціональних можливостей, які спрямовані на ефективне управління АЗС, моніторинг запасів пального, управління транзакціями та забезпечення аналітики продажів. SkyBitz Petroleum Logistics навіть дає змогу відстежувати вантажівки з пальним та контролювати їхні маршрути в режимі реального часу, що значно підвищує ефективність логістичних операцій та безпеку транспортування пального (див рис. 1.3). Але натомість не пропонує жодного рішення для управління персоналом. Тож під час аналізу конкурентних рішень були виявлені як переваги, так і недоліки кожної з систем.

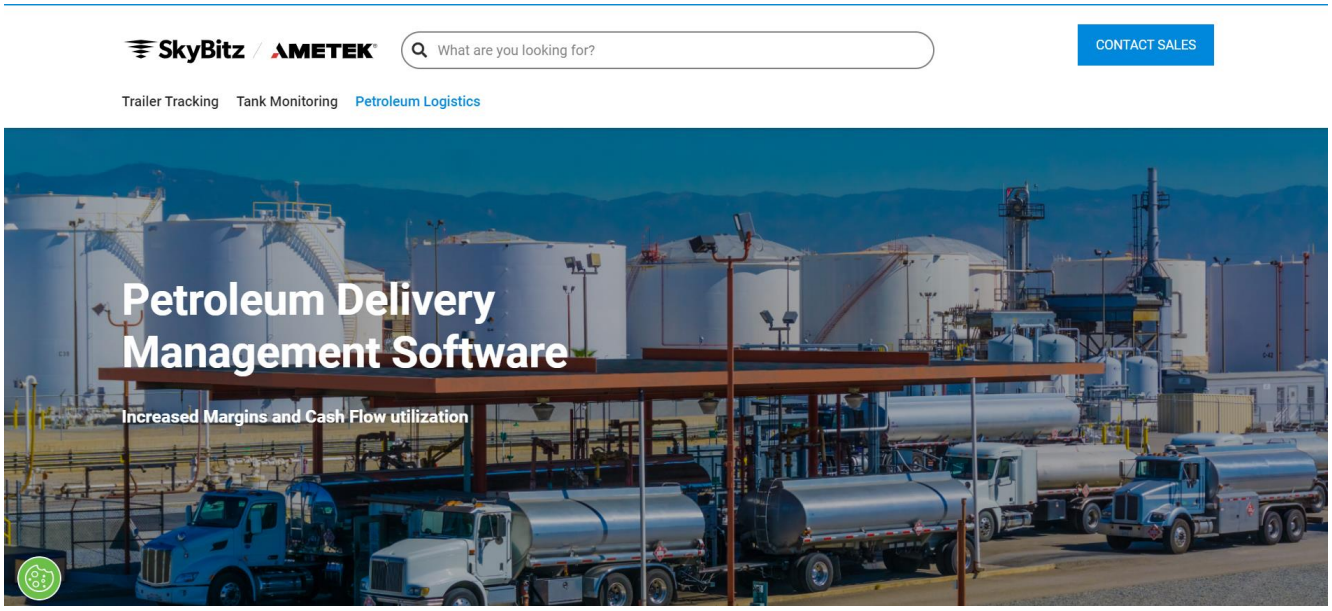


Рисунок 1.3 - SkyBitz Petroleum Logistics (за даними [4])

Розглядаючи існуючі аналоги програмних систем обліку роботи мережі АЗС, можна виділити декілька ключових переваг, які роблять ці продукти популярними серед користувачів. Аналогічні системи значно знижують потребу в ручному введенні даних та обробці інформації, що зменшує кількість помилок та підвищує ефективність роботи. Багато з існуючих систем надають можливість моніторингу та управління операціями АЗС в режимі реального часу, що дозволяє швидко реагувати на зміни та приймати обґрунтовані рішення. Сучасні аналоги зазвичай мають інтуїтивно зрозумілий інтерфейс, що спрощує процес навчання персоналу

та знижує ймовірність помилок під час експлуатації. Багато систем підтримують інтеграцію з бухгалтерськими програмами, системами управління запасами та іншими бізнес-процесами, що забезпечує єдину інформаційну середу для компанії. Існуючі продукти часто включають розширені можливості для створення звітів та проведення аналізу даних, що допомагає керівникам отримувати необхідну інформацію для стратегічного планування. Забезпечення високого рівня безпеки даних є пріоритетом для багатьох аналогічних систем, що захищає конфіденційну інформацію від несанкціонованого доступу. Багато сучасних рішень пропонують мобільні додатки або веб-інтерфейси, які дозволяють керувати АЗС з будь-якого місця, що забезпечує додаткову гнучкість для користувачів. Більшість аналогів надають комплексну підтримку користувачів, включаючи навчання, технічну підтримку та регулярні оновлення системи. Ці переваги роблять існуючі аналогічні програмні продукти потужними інструментами для ефективного управління АЗС та сприяють покращенню операційних процесів.

Хоча існуючі аналоги програмних систем обліку роботи мережі АЗС мають багато переваг, вони також мають кілька недоліків, які можуть вплинути на їх ефективність та прийнятність для користувачів. Багато сучасних рішень можуть бути дорогими в плані початкового впровадження та налаштування, що може бути непосильним для малих та середніх підприємств. Іноді інтеграція з існуючими системами та інфраструктурою може бути складною та вимагати значних ресурсів та часу, що може затримувати запуск нової системи. Деякі системи мають обмежені можливості налаштування під конкретні потреби бізнесу, що може знижувати їхню ефективність у різних умовах роботи. Складні інтерфейси та численні функції можуть вимагати тривалого навчання персоналу, що може знижувати продуктивність на початкових етапах використання системи. Деякі рішення можуть не справлятися з ростом бізнесу або змінами в його структурі, що може призвести до необхідності заміни системи або значних оновлень. Деякі аналоги можуть не мати певних специфічних функцій, які важливі для окремих АЗС, що знижує їхню цінність для таких користувачів. Деякі постачальники можуть повільно випускати оновлення або надавати технічну підтримку, що може

призвести до проблем з безпекою та стабільністю системи. Використання систем від сторонніх постачальників може призвести до залежності від їхнього графіка оновлень, рівня підтримки та розвитку продукту. Використання сторонніх систем може викликати занепокоєння щодо конфіденційності даних, особливо якщо ці дані зберігаються на серверах постачальника. Підтримка та обслуговування складних систем може бути дорогим, що впливає на загальні операційні витрати компанії. Ці недоліки підкреслюють важливість ретельного вибору та оцінки програмної системи обліку роботи мережі АЗС перед її впровадженням, щоб забезпечити максимальну відповідність потребам бізнесу та мінімізувати можливі ризики.

Ці висновки підтверджують необхідність створення власного веб-застосунку для обліку роботи мережі АЗС, який зможе врахувати потреби користувачів, бути більш доступним у впровадженні та використанні. Використання сучасних технологій, таких як Blazor та мова програмування C#, дозволить створити ефективний та надійний інструмент для управління АЗС, який забезпечить високу продуктивність та зручність роботи. Таким чином, результати аналізу конкурентних систем підтверджують актуальність та доцільність розробки власного веб-застосунку, який відповідатиме сучасним вимогам ринку та забезпечить якісне управління мережею АЗС.

Аналіз предметної галузі показує, що існує потреба в ефективних та доступних системах управління для мереж АЗС. Незважаючи на наявність на ринку потужних рішень, їхні висока вартість та складність налаштування можуть бути суттєвими бар’єрами для впровадження, особливо для малих та середніх мереж АЗС. Це підкреслює актуальність розробки нового веб-застосунку для обліку роботи мережі АЗС, який буде враховувати специфічні потреби користувачів та забезпечувати інтуїтивно зрозумілий інтерфейс. Важливою функціональністю нового застосунку буде управління змінами співробітників та їх прив'язка до конкретних АЗС, що дозволить підвищити ефективність роботи персоналу та оптимізувати операційні процеси.

1.3 Постановка задач

Розробка веб-застосунку для обліку роботи мережі автозаправних станцій має на меті створення ефективної системи управління фінансовими транзакціями, аналітикою та звітністю, клієнтським обслуговуванням і управлінням співробітниками.

Тож, система повинна надавати наступні функціональності:

- підтримка роботи користувачів з наступнимим ролями: адміністратор та співробітник;
- вхід до системи;
- зміна паролю;
- створення транзакції;
- формування звіту по транзакціям;
- додавання інформації про нову станцію;
- редагування та видалення інформації про станції;
- додавання інформації про нової колонки;
- редагування та видалення інформації про колонки;
- додавання інформації про пальне;
- додавання ціни пального;
- додавання інформації про акції;
- додавання інформації про нового працівника;
- редагування інформації про існуючого працівника;
- видалення інформації про працівника.

Всі зазначені вище функції повинні бути реалізовані у веб-застосунку. Застосунок повинен бути реалізований, використовуючи клієнт-серверну архітектуру.

Інтерфейс користувача повинен відповідати принципам юзабіліті, тобто бути зрозумілим, мінімалістичним, показувати всі важливі опції веб-застосунку.

Система повинна бути захищеною: зберігати паролі користувача у захешованому стані. Для автентифікації користувача повинен бути використаний

JSON Web Token (JWT). Взаємодія між компонентами застосунку повинна здійснюватися за допомогою HTTPS-протоколу.

Запит користувача не має бути довший за 2 секунди, за цей час запит повинен коректно обробитися на серверній частині та повернути бажаний результат на сайті.

У якості бази даних буде застосовуватися PostgreSQL [5], який був обраний як основна система управління базами даних (СУБД) з-за своєї стабільності, надійності та розширених можливостей для роботи з даними. Його відкритість та широкий функціонал роблять PostgreSQL[6] ідеальним вибором для системи, що обробляє великий обсяг даних та потребує швидкого та надійного доступу до них.

Веб-застосунок має бути реалізований, використовуючи наступні технології: мова програмування C# з використанням технології Blazor.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Перед початом роботи над програмною системою потрібно чітко сформувати вимоги до неї. Вимоги у загальному випадку – це сукупність властивостей, якими повинна володіти система, що буде створена.

2.1 Призначення розробки

Головною метою майбутнього веб-застосунку є спрощення управління автозаправними станціями (АЗС), а також покращення продуктивності роботи працівників АЗС. На відміну від своїх аналогів, розроблювана система матиме зрозуміліший інтерфейс та набагато більше функціональних можливостей для адміністраторів та співробітників АЗС.

Система буде дуже корисною для ведення обліку пального та цін на нього. Вона дозволить оперативно управляти запасами пального, контролювати його надходження та витрати, а також моніторити цінову політику у режимі реального часу. Завдяки можливості додавання та видалення знижок, адміністратори зможуть легко адаптуватися до ринкових умов, пропонуючи клієнтам вигідні умови та акції.

Переглядаючи статистику продажів та попиту на різні види пального та товарів, можна буде робити обґрунтовані рішення щодо асортименту і цінової політики. Система надасть детальні звіти про динаміку продажів, що дозволить сфокусуватися на продуктах, які користуються найбільшим попитом, і більше розвивати саме ці напрямки послуг.

Для працівників АЗС веб-застосунок стане незамінним інструментом у повсякденній роботі. Вони зможуть швидко обслуговувати клієнтів, ефективно управляти запасами пального, а також слідкувати за своєю продуктивністю. Система допоможе організувати роботу на станції, плануючи графіки зміни працівників, автоматизуючи рутинні операції та знижуючи можливість помилок.

Більше того, адміністратору буде надано контроль над кадрами, щоб краще контролювати робочі процеси на АЗС. Він зможе відстежувати завантаженість кожного працівника, що допоможе уникнути простоїв та забезпечити рівномірний розподіл робочого навантаження. Аналізуючи статистику роботи, адміністратор

зможе приймати обґрунтовані рішення щодо покращення ефективності персоналу та оптимізації процесів.

Впровадження даної системи сприятиме загальному підвищенню ефективності управління АЗС, забезпечить прозорість та контроль над операційними процесами, покращить взаємодію з клієнтами та сприятиме розвитку бізнесу за рахунок оптимізації ресурсів та зниження витрат.

2.2 Вимоги до програмного продукту

Програмна система повинна бути реалізована у вигляді веб-застосунку та бути доступною на всіх актуальних браузерах таких як Google Chrome, Safari, Edge.

Інтерфейс повинен бути зручним для кожного користувача та інтуїтивно зрозумілий. Не повинен бути перевантаженим великою кількістю елементів.

Дані з серверної частини повинні швидко завантажуватись та відображатись на клієнтській частині.

У системі буде присутнє розподілення користувачів на ролі: адміністратор та співробітник.

Програмна система має вирішувати описані у пункті 1.1 задачі. Вимоги до можливостей кожного з компонентів у початковій версії наведені нижче.

Для розробки серверної частини системи управління АЗС використовується платформа .NET Core[7], яка є високопродуктивною, кросплатформенною та добре підходить для створення надійних веб-сервісів.

.NET Core підтримує широкий спектр бібліотек та фреймворків, що забезпечує гнучкість і ефективність у розробці застосунку.

Серверна частина виконує кілька ключових завдань:

- реєстрація та авторизація користувачів (адміністраторів та співробітників);
- використання JSON Web Token (JWT) для забезпечення безпечної аутентифікації та авторизації;
- хешування паролів для безпеки;
- формування звітів і статистики для адміністраторів;
- управління інформацією про станції, колонки та транзакції;

- підтримка CRUD операцій для користувачів системи.

Для розробки клієнтської частини системи обрано Blazor, який дозволяє писати клієнтський код за допомогою C# та .NET. Blazor забезпечує високий рівень інтеграції з серверною частиною, що спрощує розробку та підтримку застосунку. Використання Blazor дозволяє знизити складність проекту завдяки можливості спільного використання коду між сервером і клієнтом.

Клієнтська частина забезпечує зручний та інтуїтивно зрозумілий інтерфейс для всіх користувачів системи (адміністраторів та співробітників), включаючи функції реєстрації та авторизації, перегляд та управління станціями та колонками, створення та перегляд звітів по транзакціях, а також додавання нових транзакцій.

Для управління базами даних використовується PostgreSQL, яка є реляційною системою управління базами даних, відомою своєю стабільністю, надійністю та підтримкою складних операцій. PostgreSQL забезпечує високу продуктивність та гнучкість завдяки своїй розширеній мові запитів SQL, що дозволяє ефективно працювати з великими обсягами даних.

Інтерфейс користувача розробляється з використанням Fluent UI, який надає набір готових компонентів та дизайнів для створення сучасного, доступного та інтерактивного інтерфейсу. Fluent UI допомагає забезпечити єдність дизайну застосунку, підвищити користувацький досвід та зменшити час розробки завдяки використанню перевірених і стандартизованих компонентів.

Система управління АЗС також включає низку функціональних обмежень та вимог для забезпечення належного рівня безпеки та зручності використання. Зокрема, користувачі повинні бути авторизовані для доступу до основних функцій системи, співробітники можуть бути зареєстровані тільки адміністраторами. Якщо користувач не авторизований або не зареєстрований, він не має доступу до функціоналу веб-застосунку.

Зареєстрований та авторизований користувач повинен мати доступ до всього функціоналу системи. Адміністратор повинен мати змогу керувати кожним працівником. Тобто приймати на роботу, звільняти з посади, редагувати його персональні дані, додавати або видаляти його посаду.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проектування ПЗ

Для кращого розуміння системи було використано UML (Unified Modeling Language) – мову моделювання, основна мета якої – визначити стандартний спосіб візуалізації того, як була розроблена система. UML діаграми спрощують розуміння системи шляхом візуалізації її структури та поведінки за допомогою різних типів діаграм.

У роботі було вирішено використати Use-case діаграму, діаграма активносіті, діаграму розгортання веб-системи, діаграму послідовності та діаграму баз даних.

Використання цієї мови спрощує спілкування між командами, між замовником та командами, бо в цьому випадку відпадає необхідність пояснювати код і статичну структуру системи.

В уніфікованій мові моделювання діаграма прецендентів або use case діаграма може узагальнити деталі користувачів системи або акторів та їх взаємодії з системою. В діаграмі актори мають дві ролі – адмін та співробітник.

Суть діаграми полягає в тому, що проектована система представляється у вигляді акторів, які взаємодіють із системою за допомогою варіантів використання. Варіант використання описує послуги, що система пред’являє актору. Кожен варіант використання визначає певний набір дій, який надається системою при діалозі з актором. Деякі актори мають спільні дії, наприклад логін в систему. Деякі дії можуть належати тільки одному акторові, наприклад адміністратор має змогу додавати фніормацію про нового співробітника або ж видалити її. При цьому нічого не говориться про те, яким способом буде реалізована взаємодія акторів із системою тож немає обмежень реалізації.

Як видно на рисунку 3.1, користувачі системи можуть мати одну з двох ролей.

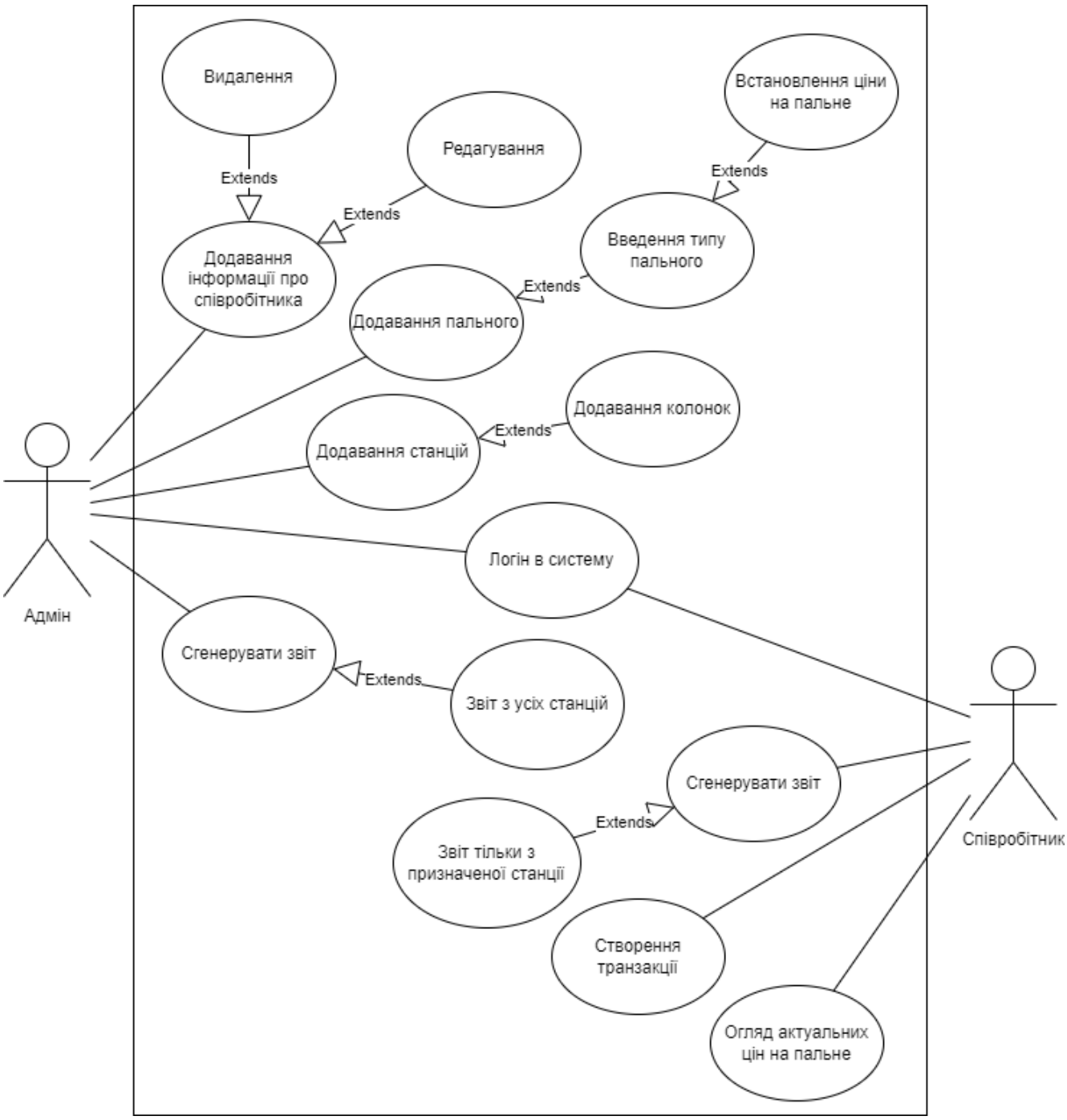


Рисунок 3.1 – Use-case діаграма веб-застосунку (рисунок виконаний самостійно)

Співробітник має увійти до системи, бо інакше в нього не буде доступу до функціоналу. Основний функціонал співробітника – це створення транзакцій, також можна подивитись актуальні ціни на пальне. Як і адміністратор у співробітника є доступ до звіту, але тільки з тієї станції, до якої у співробітника є доступ. Загальна діаграма активності наведена на рис.3.2.

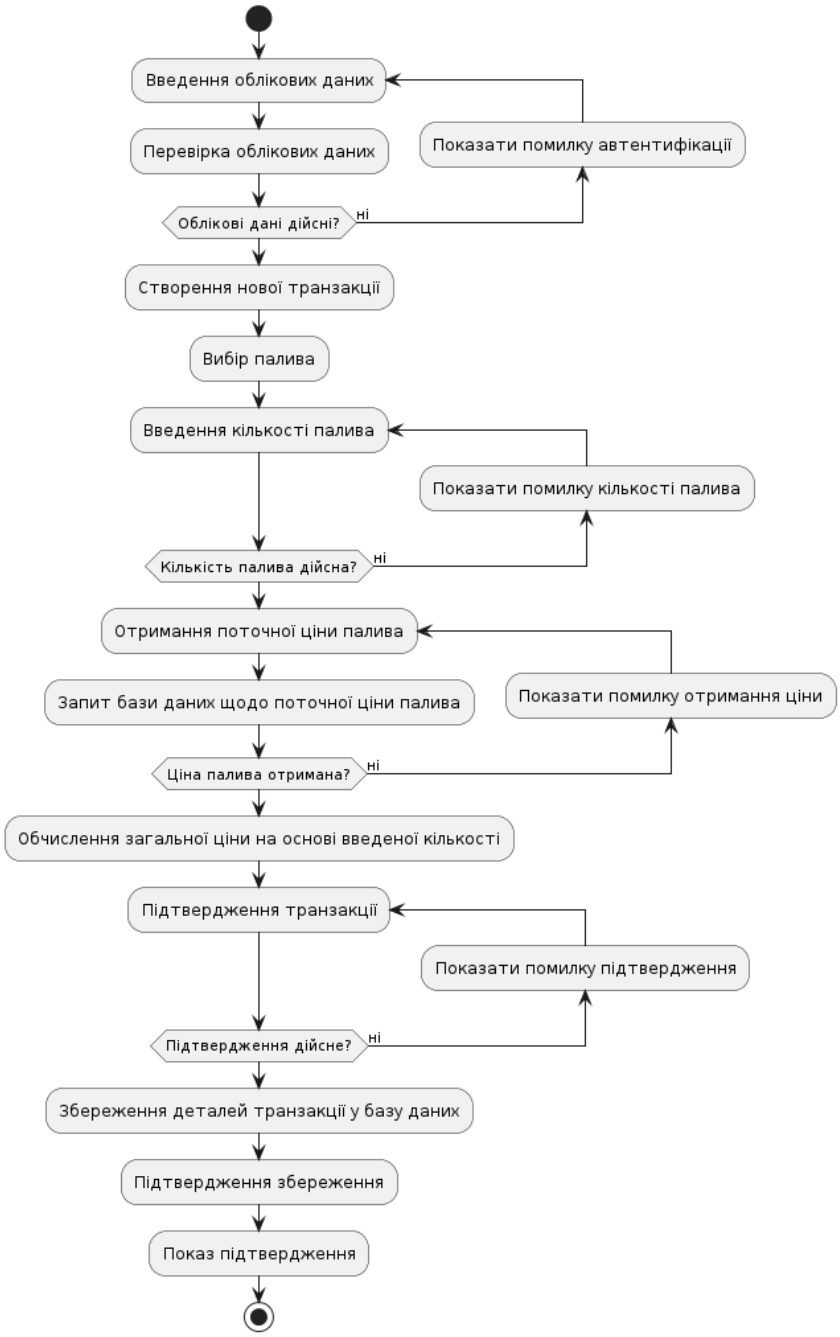


Рисунок 3.2 – Діаграма активності (рисунок виконаний самостійно)

У адміністратора є доступ до звітів з усих станцій мережі АЗС. Адміністратор має змогу додати співробітника, а також редагувати інформацію про нього, або навіть видалити. Також адміністратору доступно додавання пального до системи. Він може обрати тип пального та встановити ціну на пальне.

Адміністратор може додавати нові станції та колонки.

Також для опису системи було використано діаграму активності що використовується для моделювання динамічних аспектів системи. Вона відображає

потік управління або даних від однієї активності до іншої і може використовуватися для опису бізнес-процесів, алгоритмів або робочих процесів.

Початковою точкою є вхід в систему та введення облікових даних. Якщо дані не дійсні, то система покаже помилку в автентифікації.

Далі користувач ініціює транзакцію, вибираючи паливо. Якщо пального недостатньо, з'являється повідомлення про помилку. Якщо пального вистачає, система обчислює його вартість. Після цього користувач підтверджує створення транзакції і якщо транзакція зберігається в базу даних, то користувач бачить підтвердження збереження. Якщо ж ні – помилку.

3.2 Проєктування архітектури програмного забезпечення

Архітектура програмного забезпечення відноситься до фундаментальних структур програмної системи та дисципліни створення таких структур і систем. Кожна структура містить програмні елементи, зв'язки між ними, властивості як елементів, так і відносин.

Архітектура програмного забезпечення полягає в тому, щоб зробити фундаментальний структурний вибір, який дорого коштує змінювати після впровадження. Вибір архітектури програмного забезпечення включає конкретні структурні параметри з можливостей у дизайні програмного забезпечення. Документування архітектури програмного забезпечення полегшує взаємодію між зацікавленими сторонами, фіксує ранні рішення щодо проектування високого рівня та дозволяє повторно використовувати компоненти дизайну між проектами.

Для проектування архітектури було обрано клієнт-серверний шаблон. Клієнт-серверна архітектура – це обчислювальна модель, в якій сервер розміщує, постачає та керує більшістю ресурсів і послуг, які споживає клієнт. Цей тип архітектури має один або кілька клієнтських комп’ютерів, під’єднаних до центрального сервера через мережу або Інтернет. Це архітектура комп’ютерної мережі, в якій багато клієнтів (віддалені процесори) запрошують і отримують послугу від централізованого сервера (хост комп’ютера). Клієнтські комп’ютери забезпечують інтерфейс, який дозволяє користувачеві комп’ютера запрошувати

послуги сервера та відображати результати, які повертає сервер. Сервери чекають надходження запитів від клієнтів системи, обробляють запити, а потім відповідають на них.

Сервер у клієнт-серверній архітектурі, крім звичайного забезпечення одночасного доступу до даних, може також виконувати багато спеціалізованих задач. До переваг даної архітектури відносять зменшення навантаження на клієнтські місця, що, призводить до зменшення вимог до апаратно-програмного забезпечення клієнтів та зниження вартості системи в цілому.

Система складається з наступних рівнів:

- клієнтський рівень (для роботи з системою користувач застосовує стандартне програмне забезпечення для доступу в інтернет – браузер. Це лишає його необхідності завантажувати спеціальні програми. Браузер формує запит та надсилає його до серверної частини системи);
- середній рівень – рівень обробки запиту (серверна частина викликає серверні програмні модулі, які виконують обробку запиту і, в разі потреби, звертаються до серверу даних);
- рівень даних – сервер даних (виконує операції з даними, що зберігаються в системі. Зокрема, він може обрати дані з інформаційної бази відповідно до запиту серверної частини. Дані, з якими працює сервер даних, найчастіше організовані у вигляді реляційної бази даних, як і в нашому випадку).

Діаграма розгортання показує архітектуру виконання системи, включаючи вузли, такі як апаратне або програмне середовище виконання, і проміжне програмне забезпечення, що з'єднує їх (див. рис. 3.3). Діаграми розгортання зазвичай використовуються для візуалізації фізичного обладнання та програмного забезпечення системи. Використовуючи його, ви можете зрозуміти, як система буде фізично розгорнута на обладнанні. Діаграми розгортання допомагають моделювати апаратну топологію системи в порівнянні з іншими типами діаграм UML, які переважно описують логічні компоненти системи.

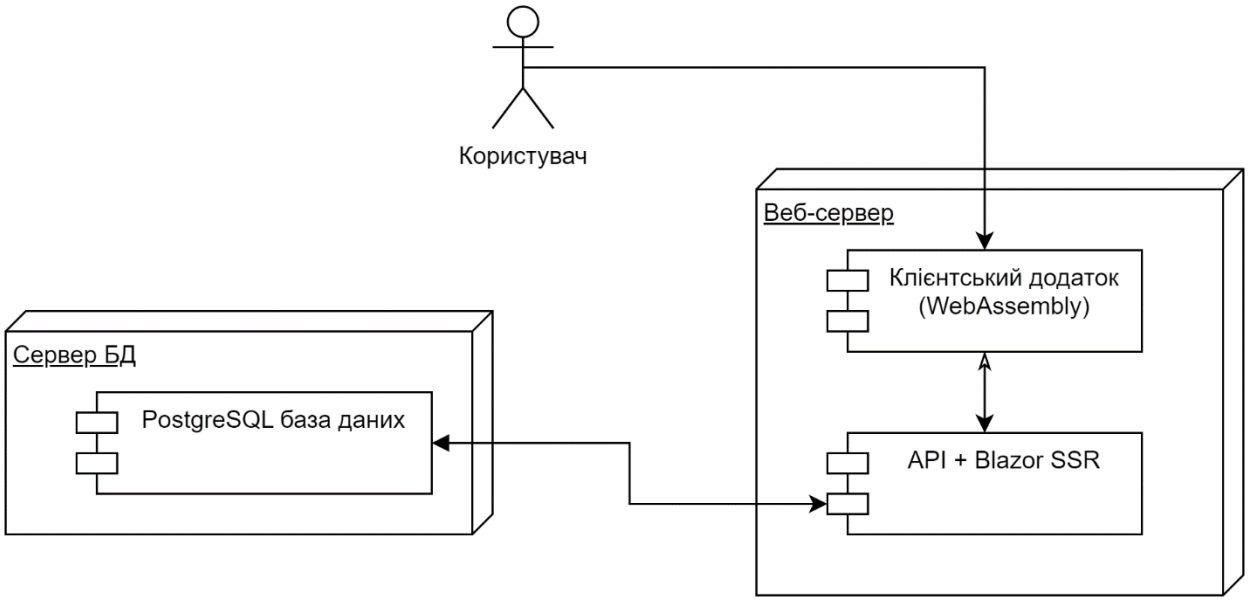


Рисунок 3.3 – Діаграма розгортання веб-системи (рисунок виконаний самостійно)

Для того, щоб отримати результат, з веб-сторінки WebAssembly посилає запит на API[8]. API в свою чергу звертається до бази даних (Database) з запитом на отримання даних, а далі, в зворотному напрямку, компоненти передають кінцевий результат користувачеві, тобто база даних надсилає дані до серверної частини, а потім ці дані передаються до клієнтської частини та відображаються на сторінці.

Компоненти даної системи взаємодіють між собою, використовуючи REST API. REST — це набір архітектурних обмежень, а не протокол чи стандарт, який повинен відповідати наступним критеріям:

- уніфікований інтерфейс. Усі запити API для одного ресурсу мають виглядати однаково, незалежно від того, звідки надходить запит;
- роз’єднання клієнт-серверу. У дизайні REST API клієнтські та серверні програмні частини повинні бути повністю незалежними один від одного. Єдина інформація, яку клієнтська частина програми має знати, це URI

запитуваного ресурсу. Аналогічно, серверна частина програми не повинна змінювати клієнтську програму, крім передачі її запитаним даним через HTTP протокол;

- відсутність стану. API REST не мають стану, тобто кожен запит повинен містити всю інформацію, необхідну для його обробки. Іншими словами, REST API не вимагають жодних сеансів на стороні сервера. Серверним програмам не дозволяється зберігати будь-які дані, пов’язані із запитом клієнта;
- можливість кешування. Якщо можливо, ресурси повинні бути кешовані на стороні клієнта або сервера. Відповіді сервера також повинні містити інформацію про те, чи дозволено кешування для доставленого ресурсу. Мета полягає в тому, щоб підвищити продуктивність на стороні клієнта, одночасно збільшуючи масштабованість на стороні сервера;
- багатошарова архітектура системи. У REST API виклики та відповіді проходять через різні рівні. У комунікаційному циклі може бути кілька різних посередників. API REST мають бути розроблені так, щоб ні клієнт, ні сервер не могли визначити, чи він спілкується з кінцевою програмою чи посередником.

REST — це набір рекомендацій, які можна впроваджувати за потреби, що робить REST API швидшими та легшими, з підвищеною масштабованістю — ідеально підходить для Інтернету речей (IoT) та розробки мобільних додатків.

Також було використано діаграму послідовностей для візуалізації взаємодії об'єктів у системі з плином часу (див рис. 3.4). Вона показує, як і в якій послідовності об'єкти обмінюються повідомленнями.

Діаграма послідовності представляє інформацію в горизонтальному і вертикальному вирівнюванні. Об'єкти, які надсилають повідомлення один одному, зображуються у вигляді блоків, вирівняних у верхній частині сторінки зліва направо, причому кожен об'єкт займає стовпчик простору на сторінці, обмежений вертикальною лінією, що тягнеться донизу сторінки. Повідомлення, які надсилаються від одного об'єкта до іншого, зображені у вигляді горизонтальних

стрілок. Порядок повідомлень представлений у послідовності зверху вниз і зліва направо, починаючи з першого повідомлення у верхньому лівому кутку сторінки, а наступні повідомлення розташовуються праворуч і нижче.

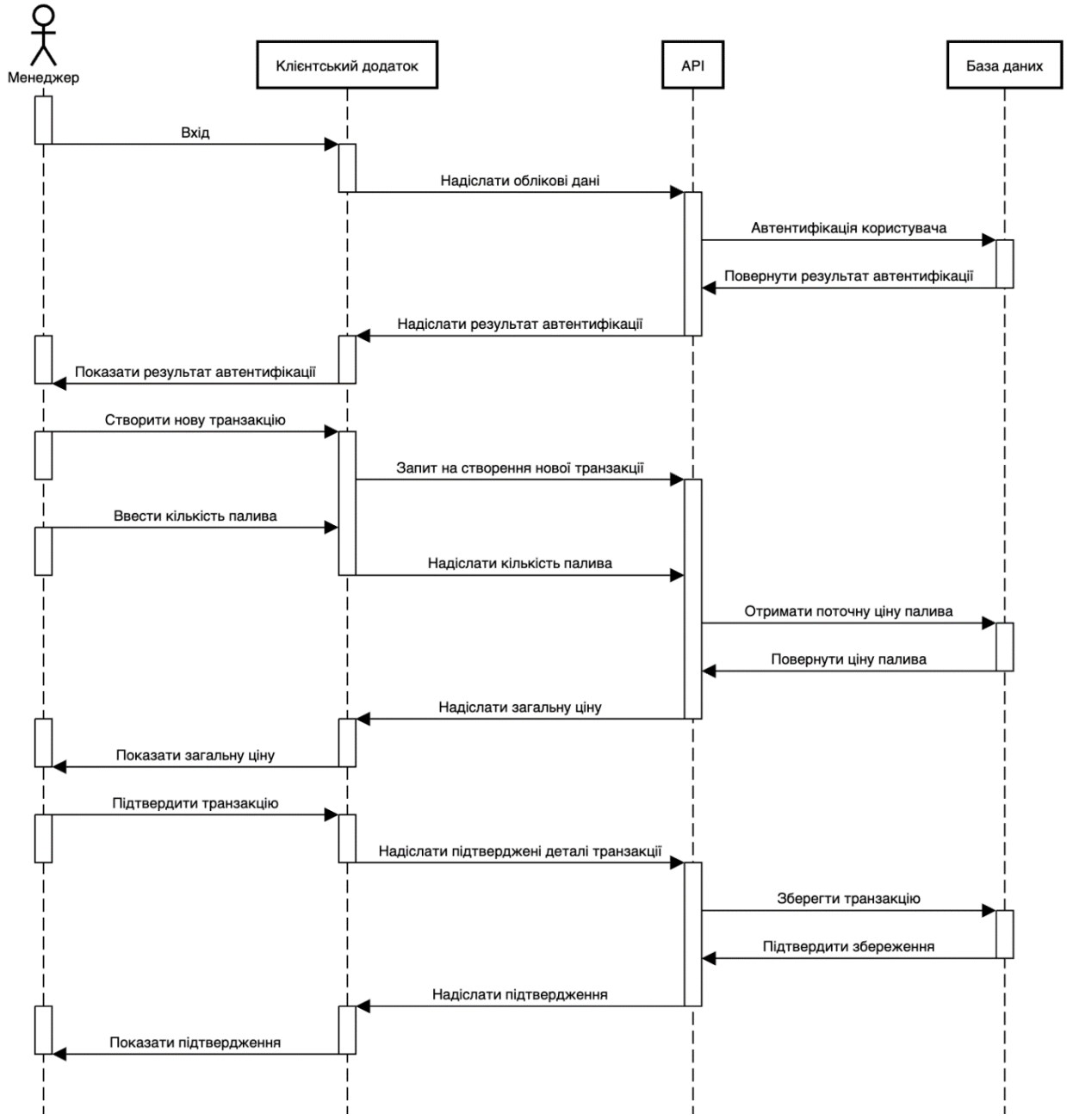


Рисунок 3.4 – Діаграма послідовностей (рисунок виконаний самостійно)

Це діаграма взаємодії, яка описує процес взаємодії між менеджером, клієнтським додатком, API та базою даних у системі. По суті, це кроки, які відбуваються під час автентифікації користувача та створення нової транзакції в застосунку.

Спершу менеджер входить в систему через клієнтський застосунок. Потім клієнтський застосунок надсилає облікові дані до API, яке в свою чергу звертається до бази даних для автентифікації користувача. Результат автентифікації повертається через API клієнтському застосунку, який відображає його менеджеру.

Після цього менеджер створює нову транзакцію, введенням кількості палива в клієнтському застосунку. Застосунок надсилає ці дані до API, яке знову взаємодіє з базою даних для отримання поточної ціни палива. Потім API надсилає загальну ціну клієнтському застосунку, який відображає її менеджеру.

Після підтвердження менеджером транзакції, застосунок надсилає підтверджені деталі транзакції до API, яке зберігає їх у базі даних. Після успішного збереження база даних надсилає підтвердження клієнтському застосунку, який відображає його менеджеру.

Отже, ця діаграма відображає основні кроки взаємодії між різними компонентами системи під час автентифікації користувача та створення нової транзакції.

3.3 Проектування структури зберігання даних

На етапі проектування було прийнято рішення використовувати PostgreSQL як основну систему управління базами даних (СУБД) та EF.Core[9] в якості ORM. Це рішення було обґрунтоване кількома ключовими факторами:

- надійність та стабільність: PostgreSQL є однією з найнадійніших СУБД, яка успішно використовується в промислових масштабах. Її архітектура забезпечує високий рівень стабільності та захисту даних.
- масштабованість: PostgreSQL підтримує різні методи масштабування, включаючи горизонтальне та вертикальне масштабування, що дозволяє ефективно обробляти великі обсяги даних.
- розширюваність: Завдяки модульній архітектурі та підтримці розширень, PostgreSQL легко адаптується під специфічні потреби проекту. Це дозволяє інтегрувати додаткові функціональні можливості без значних змін в основному коді.

– підтримка складних запитів: PostgreSQL[10] володіє потужним SQL-двигуном, який дозволяє виконувати складні запити, включаючи обробку геопросторових даних, повнотекстовий пошук, а також аналіз даних у реальному часі.

У ході формуванні вимог та аналізу предметної області була створена діаграма бази даних (див рис. 3.5).

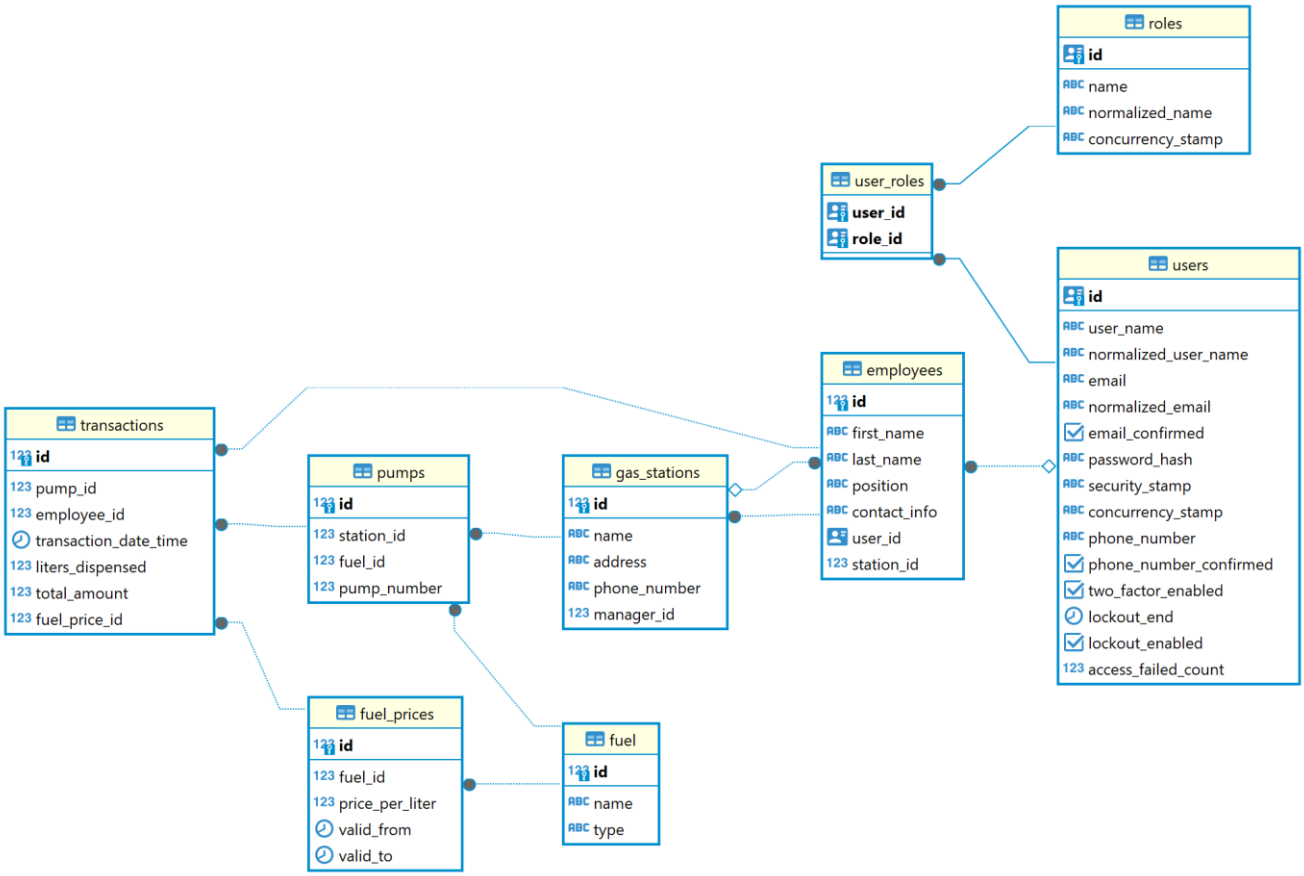


Рисунок 3.5 – Діаграма бази даних (рисунок виконаний самостійно)

Для забезпечення ефективного зберігання та обробки даних було розроблено кілька основних таблиць, кожна з яких виконує певну функцію. Нижче наведено опис ключових таблиць та їх взаємозв'язків.

Таблиця roles зберігає інформацію про різні ролі, які можуть бути присвоєні користувачам. Основні поля включають унікальний ідентифікатор ролі, назву ролі, нормалізовану назву для унікальності та поле для забезпечення одночасного доступу. Одна роль може бути присвоєна багатьом користувачам, що реалізується через таблицю user_roles.

Таблиця users містить інформацію про зареєстрованих користувачів системи. Основні поля включають унікальний ідентифікатор користувача, ім'я користувача, нормалізоване ім'я для унікальності, електронну адресу, нормалізовану електронну адресу для унікальності, підтвердження електронної адреси, хеш пароля, поле для забезпечення безпеки, поле для забезпечення одночасного доступу, номер телефону, підтвердження номера телефону, двофакторну аутентифікацію, кінець періоду блокування, активність блокування та кількість невдалих спроб доступу. Один користувач може мати кілька ролей через таблицю user_roles і може бути пов'язаний з працівником через таблицю employees.

Таблиця user_roles забезпечує зв'язок між користувачами та їх ролями, реалізуючи концепцію багато-до-багатьох. Основні поля включають ідентифікатор користувача та ідентифікатор ролі. Ця таблиця пов'язана з таблицями users та roles, дозволяючи одному користувачу мати кілька ролей і одній ролі належати багатьом користувачам.

Таблиця fuel містить інформацію про різні види пального. Основні поля включають унікальний ідентифікатор виду пального, назву та тип пального. Для одного виду пального може бути декілька цін, що зберігаються в таблиці fuel_prices, а також один вид пального може бути доступний на кількох колонках, що зберігаються в таблиці pumps.

Таблиця fuel_prices містить інформацію про ціни на пальне. Основні поля включають унікальний ідентифікатор ціни, ідентифікатор пального, ціну за літр, дату початку та закінчення дії ціни. Вона пов'язана з таблицею fuel, де для одного виду пального може бути декілька цінових записів. Також, одна ціна може бути використана в кількох транзакціях, що зберігаються в таблиці transactions.

Таблиця employees містить інформацію про працівників. Основні поля включають унікальний ідентифікатор працівника, ім'я, прізвище, посаду, контактну інформацію, ідентифікатор користувача та ідентифікатор автозаправної станції. Один працівник може бути пов'язаний з одним користувачем у таблиці users та з однією автозаправною станцією у таблиці gas_stations.

Таблиця gas_stations містить інформацію про автозаправні станції. Основні поля включають унікальний ідентифікатор станції, назву, адресу, номер телефону та ідентифікатор менеджера. Одну станцію може керувати один менеджер з таблиці employees, і на одній станції може бути багато колонок з таблиці pumps.

Таблиця pumps містить інформацію про паливні колонки. Основні поля включають унікальний ідентифікатор колонки, ідентифікатор станції, ідентифікатор пального та номер колонки. Одна станція може мати багато колонок, і одна колонка може видавати різні види пального з таблиці fuel.

Таблиця transactions містить інформацію про транзакції з продажу пального. Основні поля включають унікальний ідентифікатор транзакції, ідентифікатор колонки, ідентифікатор працівника, дату та час транзакції, кількість виданого пального, загальну суму транзакції та ідентифікатор ціни пального. Одна колонка може мати багато транзакцій, один працівник може обробляти багато транзакцій, одна ціна може бути використана у багатьох транзакціях.

3.4 Приклади найцікавіших алгоритмів та методів

Одним з цікавих алгоритмів що може бути використаним в застосунку є алгоритм обробки та аналізу даних з подальшою генерацією звіту. Алгоритм починається з збору даних про всі транзакції, які зберігаються в базі даних. Це можуть бути дані про продаж палива, обсяги, суми, інформація про співробітників, які обслуговували транзакції тощо. Для цього використовується бібліотека LINQ, що полегшує вибірку з БД за допомогою використання готових методів.

```
var transactions = await _dbContext.Transactions
    .Include(t => t.Employee)
    .Include(t => t.Pump)
    .ThenInclude(p => p.Station)
    .Include(t => t.Pump)
    .ThenInclude(p => p.Fuel)
    .ThenInclude(f => f.Prices)
    .ToListAsync(cancellationToken);
```

Після отримання необхідних даних, алгоритм розраховує загальну статистику щодо проведених транзакцій. Це включає середній обсяг проданого палива, максимальний та мінімальний обсяг продажів, середню суму та

максимальну, мінімальну суму транзакцій, та будь-які інші показники що є необхідними для бізнесу.

Для додаткового аналізу інформації, алгоритм також групує транзакції за типами палива. Для кожного типу палива розраховуються такі параметри, як кількість транзакцій і загальний дохід, що надійшов від цих транзакцій. Це дозволяє системі управління АЗС отримати детальний уявлення про те, які види палива є найбільш вигідними і популярними серед клієнтів.

```
  var commonStatistics = await _dbContext.Transactions
            .GroupBy(_ => 1)
            .Select(g => new TransactionStatisticsDto
            {
                AvgLiters = g.Average(t => t.LitersDispensed),
                MaxLiters = g.Max(t => t.LitersDispensed),
                MinLiters = g.Min(t => t.LitersDispensed),
                AvgAmount = g.Average(t => t.TotalAmount),
                MaxAmount = g.Max(t => t.TotalAmount),
                MinAmount = g.Min(t => t.TotalAmount)
            })
            .ToListAsync(cancellationToken);

        var statisticsByFuelType = (await _dbContext.Transactions
                .Include(t => t.Pump)
                .ThenInclude(t => t.Fuel)
                .ThenInclude(f => f.Prices)
                .ToListAsync(cancellationToken))
            .GroupBy(t => t.Pump.Fuel)
            .Select(g => new GroupingByFuelTypeDto
            {
                FuelType = g.Key.Name,
                Price = g.Key.Prices
                    .First(p => p.ValidFrom <= DateTime.UtcNow && p.ValidTo
>= DateTime.UtcNow)
                    .PricePerLiter,
                TotalTransactions = g.Count(),
                Income = g.Sum(t => t.TotalAmount)
            });
```

Завершальним кроком алгоритму є генерація звіту у форматі Excel, що містить розділи для загальної статистики, статистики за типом палива та деталізованих даних про кожну транзакцію. Цей звіт дозволяє керівникам АЗС та аналітикам здійснювати швидкий аналіз ефективності та приймати обґрунтовані управлінські рішення на основі даних.

```
using var package = new ExcelPackage();

var commonStatisticsWorksheet = package.Workbook.Worksheets.Add("Статистика
(загальна)");
```

```
commonStatisticsWorksheet.Cells["A1"].LoadFromCollection(commonStatistics,
true, TableStyles.Medium13);

commonStatisticsWorksheet.Cells[commonStatisticsWorksheet.Dimension.Address
].AutoFitColumns();

        var reportWorksheet = package.Workbook.Worksheets.Add("Статистика
(за типом пального)");

reportWorksheet.Cells["A1"].LoadFromCollection(statisticsByFuelType, true,
TableStyles.Medium14);

reportWorksheet.Cells[reportWorksheet.Dimension.Address].AutoFitColumns();

        var transactionsWorksheet =
package.Workbook.Worksheets.Add("Транзакції");
        transactionsWorksheet.Cells["A1"].LoadFromCollection(reportData,
true, TableStyles.Medium15);

transactionsWorksheet.Cells[transactionsWorksheet.Dimension.Address].AutoFi
tColumns();
        transactionsWorksheet.Column(9).Style.Numberformat.Format = "yyyy-
MM-dd HH:mm";

        return await package.GetAsByteArrayAsync(cancellationToken);
```

3.5 Створення UI / UX або іншого дизайну системи

Створення користувацького інтерфейсу (UI) та користувацького досвіду (UX) стає ключовим етапом у розробці будь-якого програмного забезпечення. UI визначає, які елементи та функції будуть доступні користувачеві та як вони будуть представлені на екрані. Його мета - забезпечити зручну та ефективну взаємодію між користувачем і програмою. UX, з іншого боку, це враження, яке виникає у користувача під час використання програми. Воно охоплює всі аспекти взаємодії, включаючи зручність використання, швидкість реакції програми та загальне задоволення від використання.

Прототип - це візуальне відображення ідеї програмного продукту перед його фактичною реалізацією. Це може бути простим набором макетів, що демонструють розміщення та функції елементів, або інтерактивним додатком, який дозволяє взаємодіяти з продуктом на ранніх стадіях розробки. Прототипи допомагають виявити потенційні проблеми, уникнути непорозумінь між командою розробників

та замовниками, а також забезпечити розуміння та підтримку від кінцевих користувачів.

Створення UI / UX та прототипів є важливими етапами у розробці програмного забезпечення з численними перевагами. Вони дозволяють командам розробників краще розуміти потреби та очікування користувачів, відкривають можливості для інновацій та вдосконалення дизайну продукту, а також допомагають уникнути непотрібних витрат часу та ресурсів на пізніші етапи розробки.

На рисунку 3.6 зображено макет сторінки співробітників, де адміністратор може переглянути інформацію про всіх співробітників в системі. Також зображено інформацію про хедер, головне бокове меню та інші елементи.

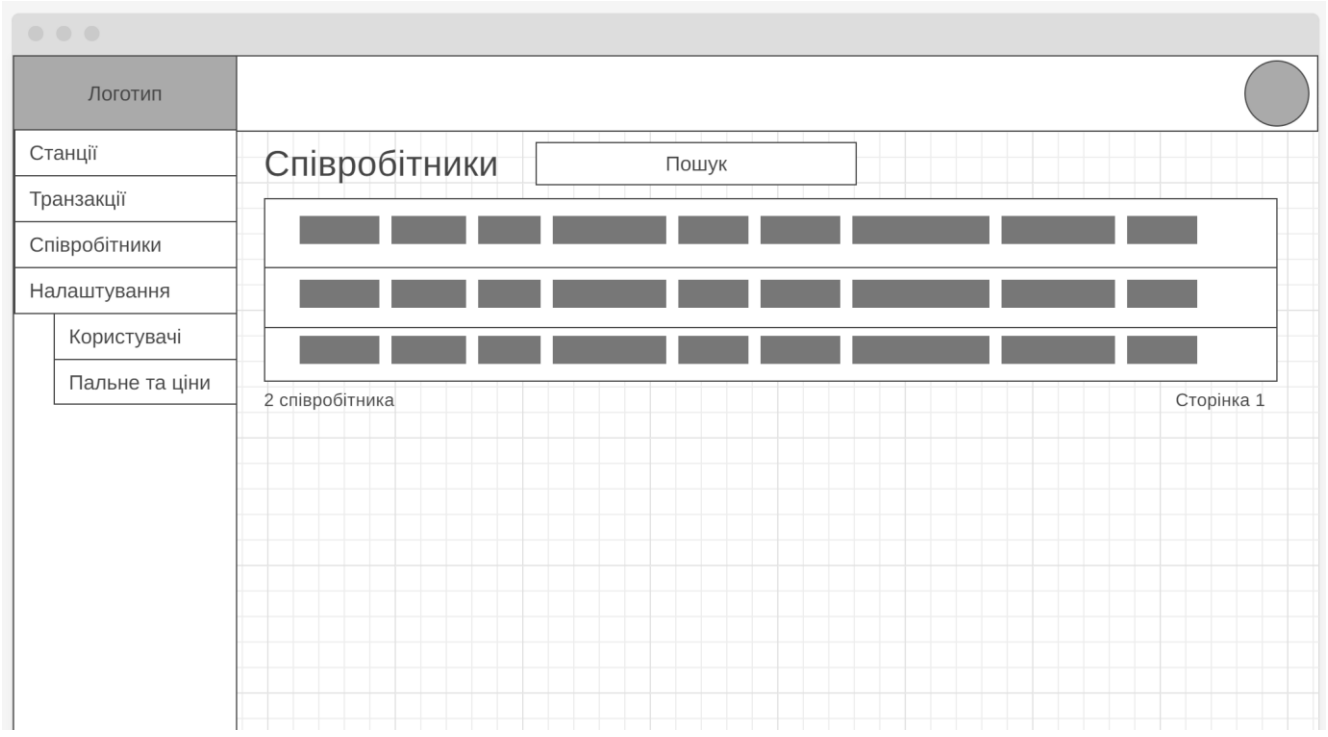


Рисунок 3.6 – Прототип сторінки співробітників (рисунок виконаний самостійно)

На рисунку 3.7 зображено дизайн сторінки співробітників створений з використанням бібліотеки Blazor FluentUI.

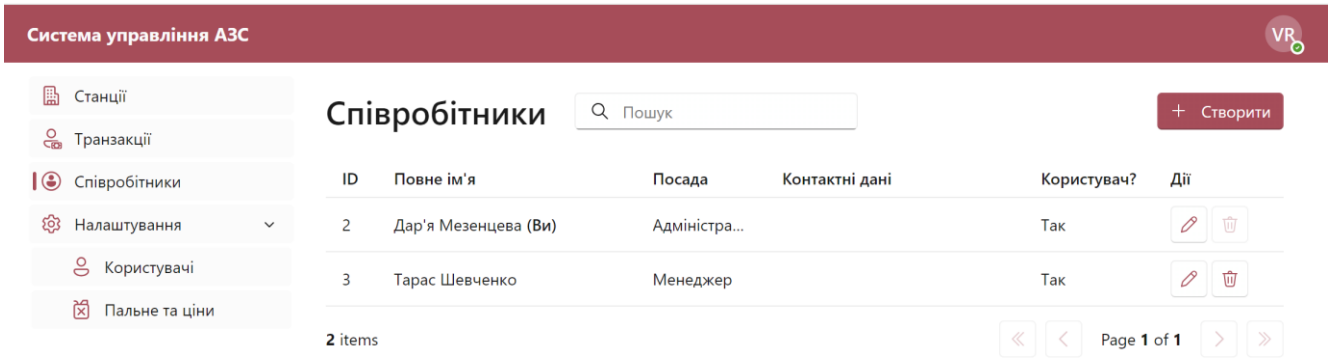


Рисунок 3.7 – Дизайн сторінки співробітників (рисунок виконаний самостійно)

Отже, створення якісних UI/UX та прототипів є важливими етапами у розробці програмного забезпечення. Завдяки цьому підходу, як показано на прикладі макетів сторінки співробітників, дизайнери можуть показати замовникам як буде виглядати майбутній продукт, а розробники можуть створювати інноваційні та вдосконалені продукти, що знижує ризики і витрати на пізніших етапах розробки.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Опис реалізації серверної частини

Після проведення аналізу предметної області та формування чітких вимог щодо програмного рішення, а також, після проектування системи, було розпочато безпосередньо розробку веб-застосунку.

Для початку було розроблено доменний рівень нашої системи. Доменний рівень — це частина програмної системи, що відповідає за бізнес-логіку та управління даними, відображаючи реальний світ у контексті предметної області. Починати з нього важливо, бо це забезпечує чітку структуру та зосередження на ключових функціях системи. В результаті цього етапу було створені сутності (див. рис. 4.1) що описують нашу систему.

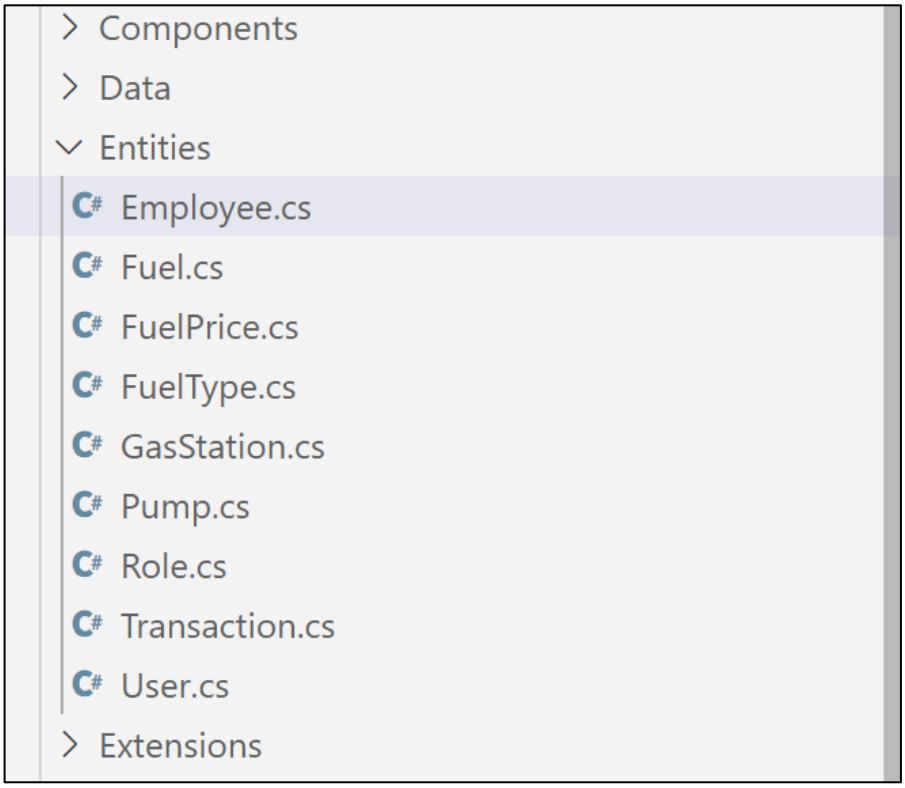


Рисунок 4.1 – Сутності доменного рівня (рисунок виконаний самостійно)

Після цього було розроблено інфраструктурний рівень нашої системи, що відповідає за взаємодію з базою даних. Так як в якості ORM було використано

EntityFramework Core, то було створено AppDbContext – контекст бази даних (див. рис. 4.2), що представляє собою таблиці БД у вигляді DbSet.

```
public class AppDbContext : IdentityDbContext<User, Role, Guid>, IAppDbContext
{
    public DbSet<Employee> Employees { get; set; } = null!;

    public DbSet<Fuel> Fuel { get; set; } = null!;

    public DbSet<GasStation> GasStations { get; set; } = null!;

    public DbSet<Pump> Pumps { get; set; } = null!;

    public DbSet<Transaction> Transactions { get; set; } = null!;

    public DbSet<User> Users { get; set; } = null!;

    public DbSet<Role> Roles { get; set; } = null!;

    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
    {

    }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());

        base.OnModelCreating(builder);
    }

    public async Task SaveChangesAsync()
    {
        await base.SaveChangesAsync().ConfigureAwait(false);
    }
}
```

Рисунок 4.2 – Контекст бази даних (рисунок виконаний самостійно)

У проекті було використано патерн Репозиторій. Це було зроблено для того, щоб ізолювати шар доступу до даних від бізнес-логіки застосунку, що спрощує тестування та зменшує залежність від конкретної реалізації зберігання даних. Для цього було встановлено nuget пакет Ardalis.Specification та зроблено відповідні конфігурації.

Також було використано патерн Специфікації. Цей патерн дозволяє визначати бізнес-правила та критерії для пошуку та фільтрації даних, що забезпечує гнучкість та чистоту коду. Також це полегшить нам функціональне розширення системи, так як таку логіку можна перевикористати (див. рис. 4.3).

```
using Ardalis.Specification;
using GasStationManagement.Entities;

namespace GasStationManagement.Specifications;

public sealed class StationDefaultSpec : Specification<GasStation>
{
    public StationDefaultSpec(string? searchTerm = null)
    {
        Query.Include(s => s.Manager);

        if (string.IsNullOrEmpty(searchTerm))
        {
            return;
        }

        Query.Search(x => x.Name.ToLower(), $"%{searchTerm.ToLower()}%");
        Query.Search(x => x.Address.ToLower(), $"%{searchTerm.ToLower()}%");
        Query.Search(x => x.PhoneNumber!.ToLower(), $"%{searchTerm.ToLower()}%");
    }
}
```

Рисунок 4.3 – Приклад реалізованої специфікації (рисунок виконаний самостійно)

Для генерації звітів у форматі Excel (.xlsx) було використано бібліотеку OfficeOpenXml (див. рис. 4.4). Ця бібліотека надає можливість програмно створювати, змінювати і зберігати документи у форматі Excel безпосередньо з програми, що дозволяє автоматизувати процес генерації звітів і забезпечити їхню сумісність із стандартами Excel.

Для вибірки даних з бази даних та їхньої агрегації використано стандартні можливості EntityFramework Core та LINQ to Entity.

Для забезпечення безпеки та контролю доступу було використано стандартні можливості фреймворку .NET Core. Це дозволило легко і швидко налаштувати аутентифікацію та авторизацію всього за допомогою декількох рядків коду.

```
            })
            .ToListAsync(cancellationToken);

        var statisticsByFuelType = await _dbContext.Transactions
            .Include(t => t.Pump)
            .ThenInclude(t => t.Fuel)
            .GroupBy(t => t.Pump.Fuel)
            .Select(g => new GroupingByFuelTypeDto
            {
                FuelType = g.Key.Name,
                Price = g.Key.Prices
                    .First(p => p.ValidFrom <= DateTime.UtcNow && p.ValidTo >= DateTime.UtcNow)
                    .PricePerLiter,
                TotalTransactions = g.Count(),
                Income = g.Sum(t => t.TotalAmount)
            })
            .ToListAsync(cancellationToken);

        using var package = new ExcelPackage();

        var commonStatisticsWorksheet = package.Workbook.Worksheets.Add("Статистика (загальна)");
        commonStatisticsWorksheet.Cells["A1"].LoadFromCollection(commonStatistics, true, TableStyles.Medium13);
        commonStatisticsWorksheet.Cells[commonStatisticsWorksheet.Dimension.Address].AutoFitColumns();

        var reportWorksheet = package.Workbook.Worksheets.Add("Статистика (за типом пального)");
        reportWorksheet.Cells["A1"].LoadFromCollection(statisticsByFuelType, true, TableStyles.Medium14);
        reportWorksheet.Cells[reportWorksheet.Dimension.Address].AutoFitColumns();

        var transactionsWorksheet = package.Workbook.Worksheets.Add("Транзакції");
        transactionsWorksheet.Cells["A1"].LoadFromCollection(reportData, true, TableStyles.Medium15);
        transactionsWorksheet.Cells[transactionsWorksheet.Dimension.Address].AutoFitColumns();
        transactionsWorksheet.Column(9).Style.Numberformat.Format = "yyyy-MM-dd HH:mm";

        return await package.GetAsByteArrayAsync(cancellationToken);
    }
}
```

Рисунок 4.4 – Реалізація звітів в Excel форматі (рисунок виконаний самостійно)

4.2 Опис реалізації клієнтської частини

Клієнтська частина веб-застосунку створена за допомогою Blazor, фреймворку від Microsoft, що дозволяє писати C# код для веб-інтерфейсів. Це забезпечує зручність розробки і високу продуктивність завдяки використанню WebAssembly. Також було використано бібліотеку готових компонентів Fluent UI.

Для цього в головному файлі застосунку було доданий рядок конфігурації builder.Services.AddFluentUIComponents(), а також імпортовано збірки бібліотеки в файлі _Imports.razor:

```
@using Microsoft.FluentUI.AspNetCore.Components
@using Microsoft.FluentUI.AspNetCore.Components.Extensions
@using Microsoft.FluentUI.AspNetCore.Components.DesignTokens
@using static Microsoft.AspNetCore.Components.Web.RenderMode
```

Крім того, було розроблено декілька макетів (див. рис. 4.5). Макет – це шаблон, що визначає загальну структуру сторінок, включаючи такі елементи як заголовок, навігаційне меню та «футер». Один макет використовується для авторизації (див. рис. 4.6), забезпечуючи мінімальний інтерфейс з полями для введення логіна та пароля. Інший макет застосовується для самого застосунку після входу в систему, включаючи більш розширене меню (див. рис. 4.7), панелі інструментів та інші елементи, необхідні для роботи користувача.

```
@inject IHttpContextAccessor Accessor
@inject SignInManager<User> SignInManager
@inject NavigationManager NavigationManager

<PageTitle>Управління АЗС</PageTitle>

<FluentLayout>

    <FluentHeader>
        Система управління АЗС
        <FluentSpacer />

        <div>
            <FluentProfileMenu Status="@PresenceStatus.Available"
                               HeaderLabel="Ваш акаунт"
                               Class="border-round-lg"
                               OnHeaderButtonClick="Logout"
                               HeaderButton="Вийти"
                               Initials="@UserInfo.Initials"
                               FullName="@UserInfo.FullName"
                               EMail="@UserInfo.Email"
                               Style="min-width: 330px;">
                <FooterTemplate />
            </FluentProfileMenu>
        </div>
    </FluentHeader>

    <FluentStack Orientation="Orientation.Horizontal" Width="100%" Class="px-4 pt-2">
        <NavMenu />

        <FluentBodyContent Class="p-3">
            @Body
        </FluentBodyContent>
    </FluentStack>
</FluentLayout>

<FluentToastProvider />
<FluentDialogProvider />
```

Рисунок 4.5 – Головний макет застосунку (рисунок виконаний самостійно)

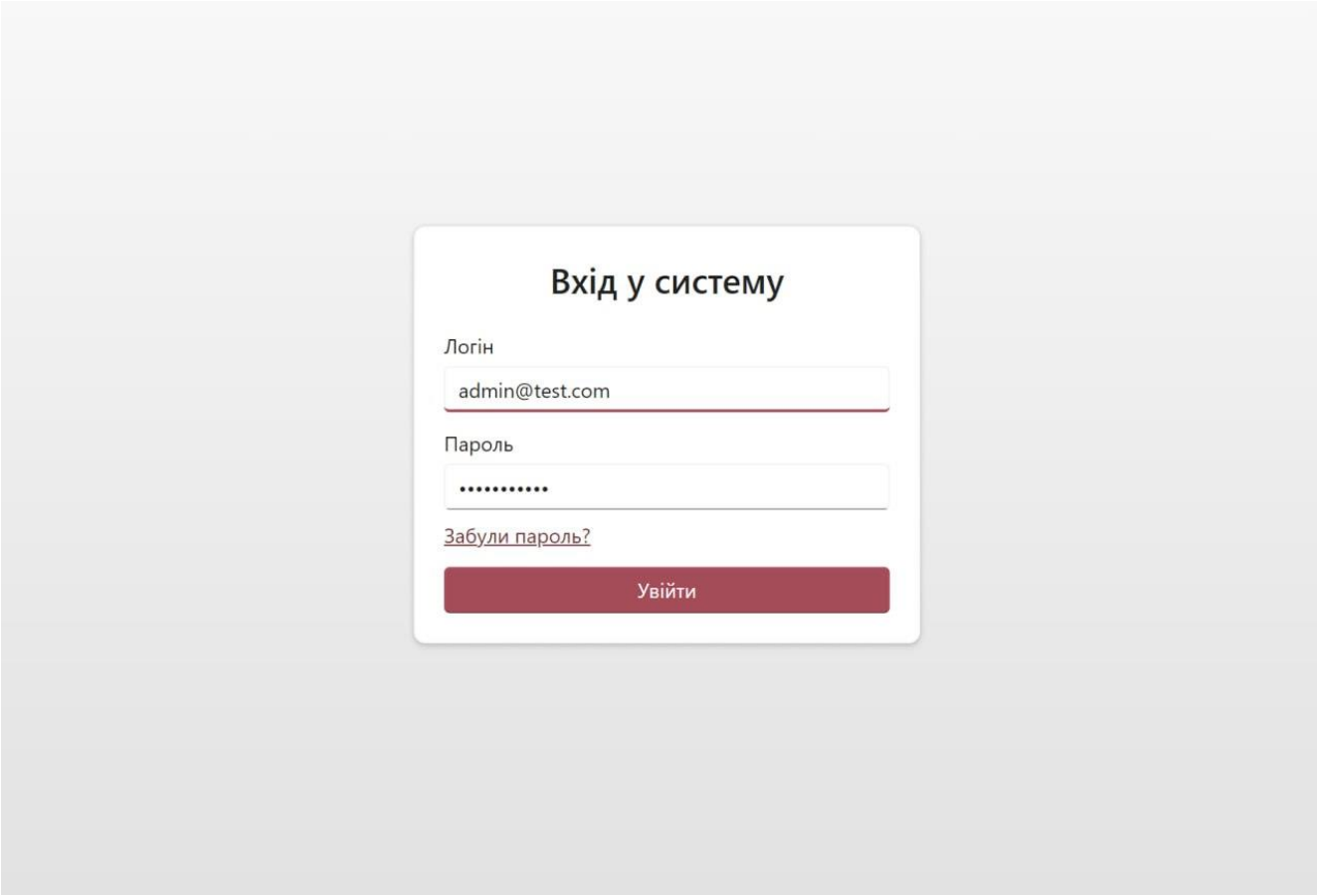


Рисунок 4.6 – Сторінка входу в систему (рисунок виконаний самостійно)

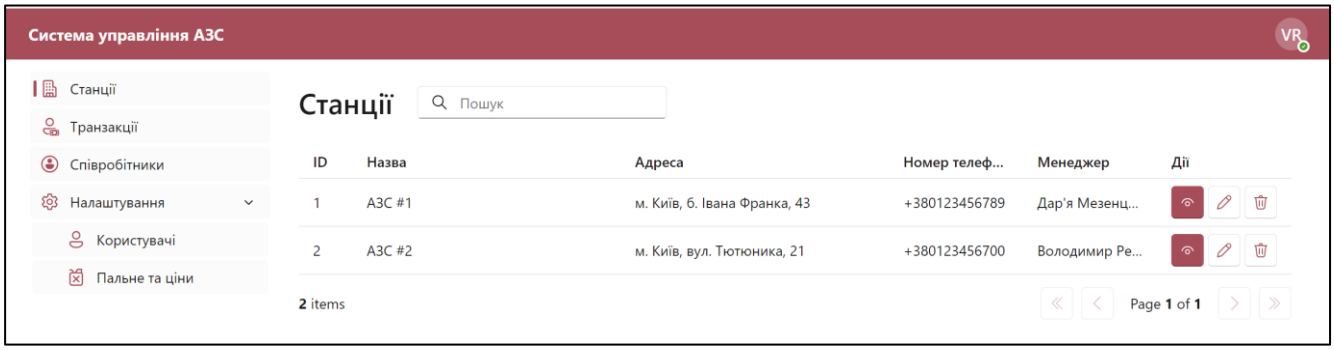


Рисунок 4.7 – Сторінка станцій з головним макетом застосунку (рисунок виконаний самостійно)

Зайшовши до системи користувач перенапрявляється на головну сторінку, що за замовчуванням являє собою список доступних станцій.

Принцип побудови додатків, заснованих на Blazor, полягає у використанні компонентів і сторінок. Компоненти - це основні будівельні блоки інтерфейсу, які можуть включати HTML, CSS та C# код. Сторінки представляють собою окремі

компоненти, пов'язані з URL-адресами, що дозволяє створювати навігацію в додатку.

Було розроблено сторінки для управління станціями, транзакціями, співробітниками, користувачами, а також для перегляду цін на пальне. Сторінка для станцій дозволяє керувати інформацією про заправні станції. Сторінка для транзакцій забезпечує доступ до історії фінансових операцій і дає змогу генерувати звіт (див. рис. 4.8). Сторінка для співробітників надає можливість додавати та редагувати дані співробітників. Сторінка для користувачів дозволяє керувати обліковими записами клієнтів. Сторінка з цінами на пальне забезпечує перегляд та оновлення поточних цін.

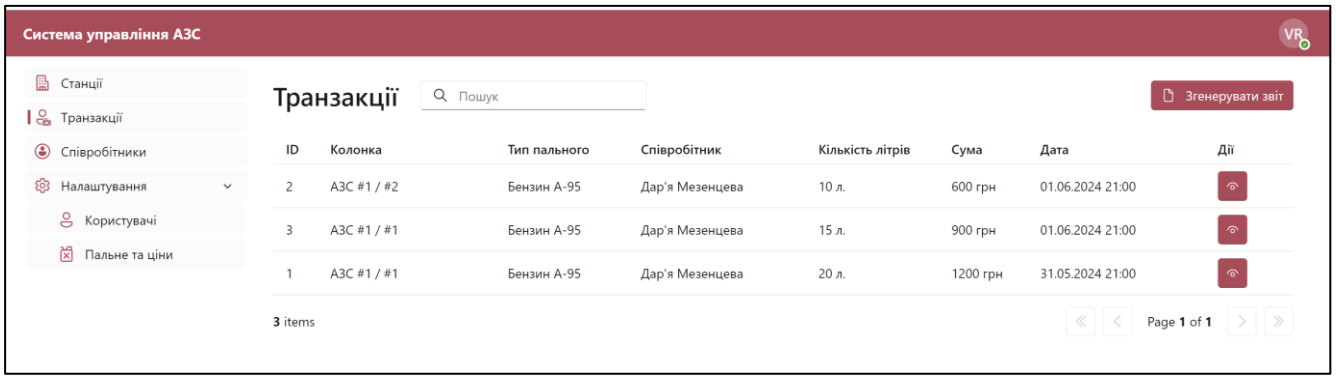


Рисунок 4.8 – Сторінка зі списком транзакцій (рисунок виконаний самостійно)

Після натискання на кнопку «Згенерувати звіт» виконується запит на серверну частину та викликається метод GenerateReportAsync (див. рис. 4.9) та користувачу повертається згенерований звіт.

```
private async Task GenerateReport()
{
    var report = await TransactionService.GenerateReportAsync();

    const string contentType = "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";
    var fileName = $"transactions_{DateTime.Now.ToString("dd_MM_yyyy_HH_mm_ss")}.xlsx";

    await Js.InvokeVoidAsync("BlazorDownloadFile", fileName, contentType, report);
}
```

Рисунок 4.9 – Реалізація методу генерації звіту на клієнтській стороні (рисунок виконаний самостійно)

В отриманому файлі відображається загальна статистика (див. рис. 4.10) де вказано середня, мінімальна та максимальна кількість літрів, а також статистика щодо витраченої суми.

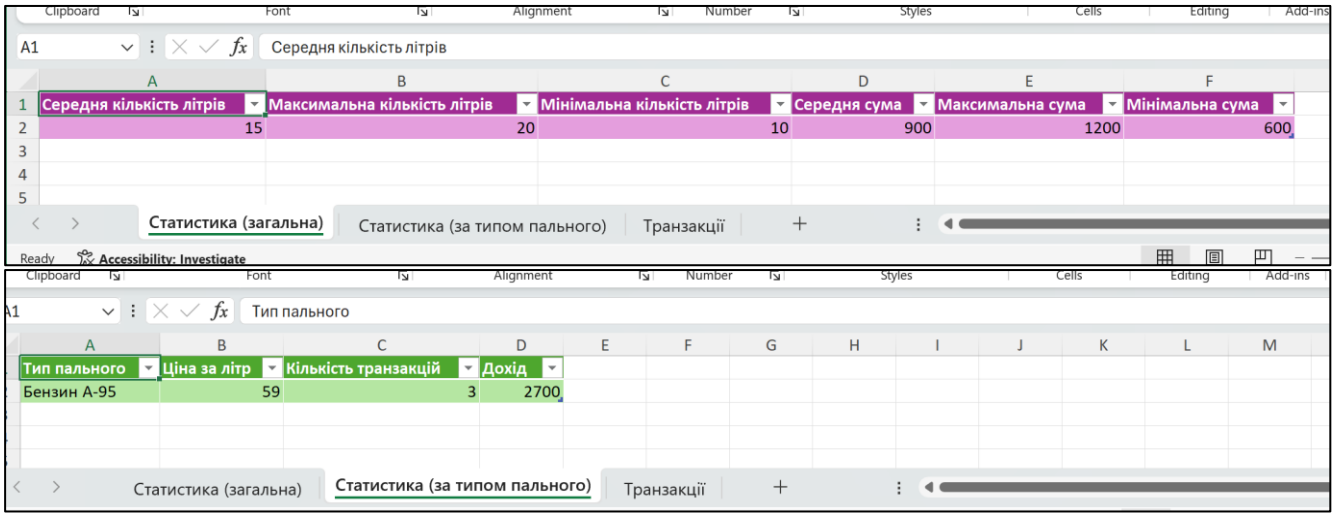


Рисунок 4.10 – Дані звіту (загальна статистика, за типом пального) (рисунок виконаний самостійно)

Крім того, у файлі відображається список всіх транзакцій (див. рис. 4.11) що входять до даного звіту та впливають на загальну статистику та статистику за типом пального.

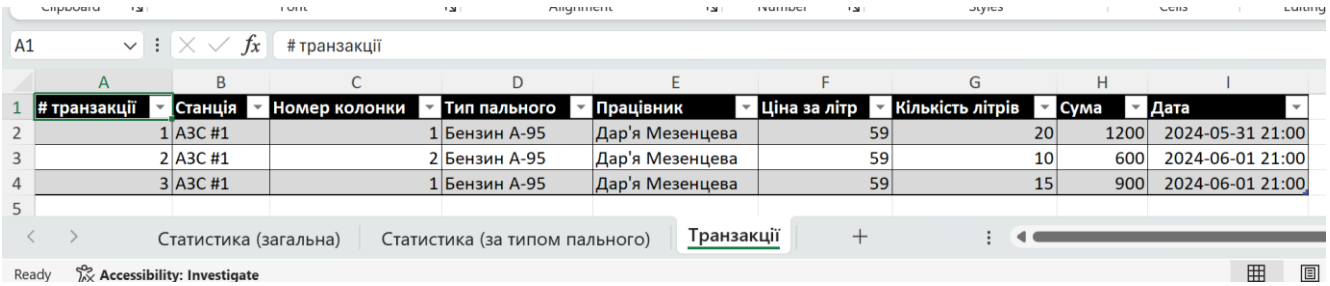


Рисунок 4.11 – Дані звіту (список транзакцій) (рисунок виконаний самостійно)

5 ТЕСТУВАННЯ РОЗРОБЛЕННОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування важлива частина SDLC – процесу, який використовується індустрією програмного забезпечення для проектування, розробки і тестування високоякісного програмного забезпечення. Основна мета тестування полягає у перевірки відповідності розробленого ПЗ до поставлених вимог та виявленні дефектів до випуску програми на ринок.

Тестування важливе на всіх етапах розробки, оскільки помилки програмного забезпечення можуть бути дорогими або навіть небезпечними. Помилки програмного забезпечення потенційно можуть призвести до грошових і людських втрат, втрат часу а також до зниження довіри користувачів до системи.

Помилки в програмному забезпеченні можуть призвести до прямих фінансових втрат через збої в роботі системи, що може спричинити втрату доходів або додаткові витрати на усунення несправностей. Наприклад, програмний збій при оплаті транзакцій може призвести до невиконаних транзакцій або навіть до втрати коштів клієнтів. Виправлення помилок на пізніх етапах розробки вимагає більше часу та зусиль, ніж їх виявлення та усунення на ранніх етапах. Це може затримати випуск продукту та знизити його конкурентоспроможність. Користувачі також очікують надійної та стабільної роботи програмного забезпечення. Наявність частих помилок та збоїв може призвести до втрати довіри до продукту, а отже, і до втрати клієнтів. Компанії, чиє програмне забезпечення часто стикається з проблемами, можуть втратити свою репутацію на ринку. Це може призвести до втрати клієнтів і партнерів, а також ускладнити залучення нових користувачів.

Тестування розробленого веб-застосунку здійснювалося за допомогою ручних методів. Основні етапи включали:

- функціональне тестування - тестування в якому було перевірено основні функції системи;
- нефункціональне тестування – тестування в якому було перевірено зручність використання системи;

− регресійне тестування - тестування системи після виправлення багів та внесення змін для виявлення нових дефектів.

Функціональне тестування включало перевірку основних функцій веб-застосунку, таких як:

− авторизація;

− реєстрація;

− зміна паролю;

− редагування даних профілю;

− додавання інформації про працівника;

− видалення інформації про працівника;

− редагування даних про працівника;

− додавання інформації про станцію;

− додавання інформації про колонку;

− додавання інформації про вид пального;

− додавання ціни пального;

− зміни цін на пальне;

− створення транзакції;

− генерація звіту;

− пошук;

− можливість вийти із системи.

Було створено сьют тест-кейсів з якого були відібрані тест кейси для регресії для перевірки вже працюючої системи після виправлення багів. Приклад тест-кейсу можна переглянути в таблиці 5.1.

Таблиця 5.1 – Тест-кейс №1 (таблиця виконана самостійно)

ID :		1	
Назва тест-кейсу:		Вхід у систему з відновленням паролю	
Створений ким:		Мезенцева Дар'я Сергіївна	
Дата створення:		20.06.2024	
Передумова			
№	Опис кроку	Очікуваний результат	Висновок
1	Відкрити веб-застосунок	Сторінка відкрита	Пройдено
2	Користувач має обліковий запис	Обліковий запис існує	Пройдено

Кінець таблиці 5.1

Вхід у систему з відновленням паролю			
№	Опис кроку	Очікуваний результат	Висновок
1	Ввести валідний Email в поле «Логін»	Email введен в поле «Логін»	Пройдено
2	Натиснути на кнопку «Увійти»	Отримано помилку «Пароль не введено»	Пройдено
3	Ввести неправильний пароль в поле «Пароль»	Введено неправильний пароль в поле «Пароль»	Пройдено
4	Настиснути на кнопку «Увійти»	Отримано помилку «Невірний пароль»	Пройдено
5	Натиснути на кнопку «Забули пароль»	Нове модальне вікно відкрито з повідомленням «Для відновлення паролю введіть ваш Email», полем «Email» та кнопкою «Відновити пароль»	Пройдено
6	Ввести email та натиснути кнопку «Відновити пароль»	На пошту прийшов лист з посиланням на оновлення паролю	Пройдено
7	Відкрити лист та перейти по посиланню	Нова вкладка в браузері відкрилась з двома полями -введенням та підтвердженням нового паролю, кнопка «Оновити пароль»	Пройдено
8	Ввести новий пароль та підтвердити його, натиснути кнопку «Оновити пароль»	Користувач бачить повідомлення про успішне оновлення паролю	Пройдено
9	Зайти на сторінку входу у веб-застосунок	Веб-застосунок відкрито	Пройдено
10	Ввести логін та новий пароль	Логін та пароль введено	Пройдено
11	Натиснути кнопку «Увійти»	Вхід в застосунок успішний	Пройдено
Результат тестування			
Тестувальниця: Мезенцева Д.С.	Дата тестування: 21.06.2024	Результат тесту: ПРОЙДЕНО	

У процесі тестування застосунку було виявлено проблему пов’язану з механізмом авторизації, а саме повідомленням після неправильної спроби входу в систему. Система відображала точну помилку, що дозволяло зловмисникам зрозуміти чи існує обліковий запис з вказаним логіном. Це дуже сильно впливало на безпеку та створювало додаткові ризики того що дані акаунту можуть будуть скомпрометовані шляхом підбору паролю, та могло призвести до отримання несанкціонованого доступу до системи. Для усунення даної проблеми було змінено текст повідомлення з «Невірний пароль» (див рис. 5.1) на більш загальний «Невірний логін чи пароль» (див рис. 5.2).

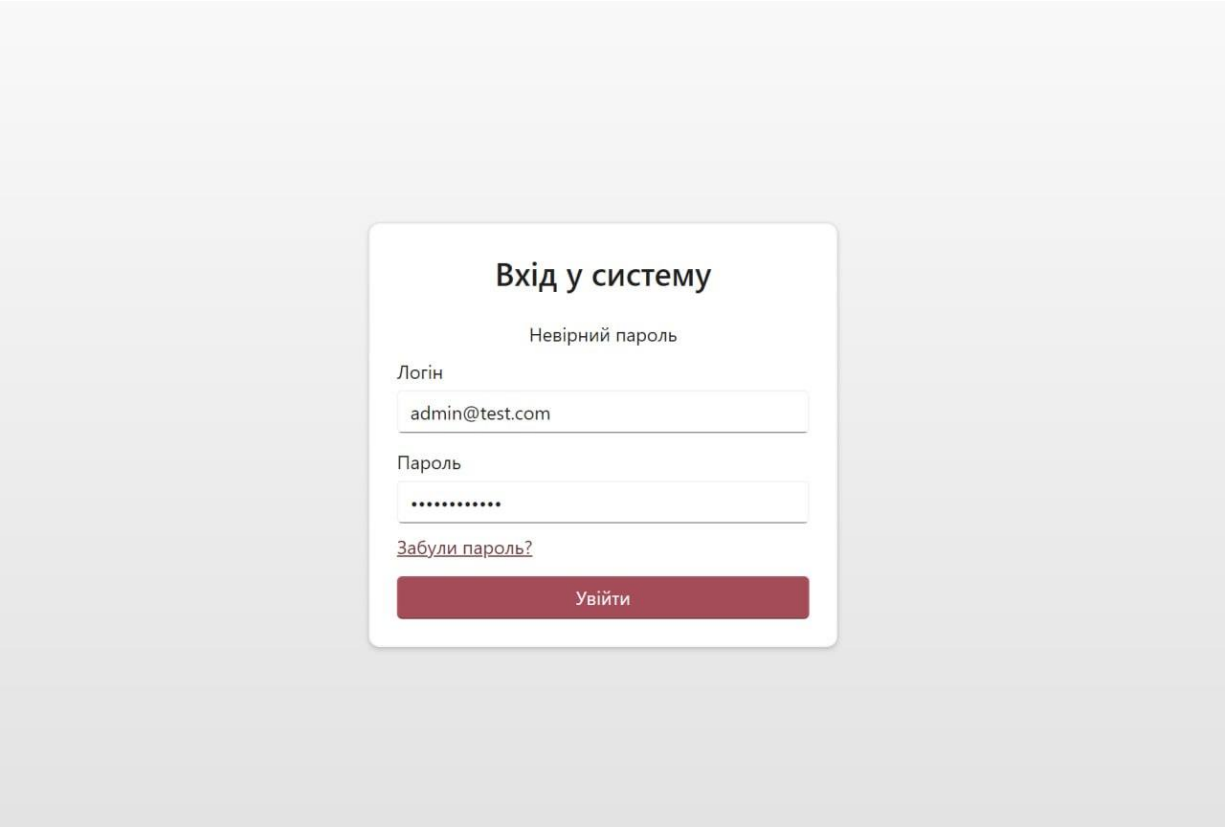


Рисунок 5.10 – Баг на скріні логін (рисунок виконаний самостійно)

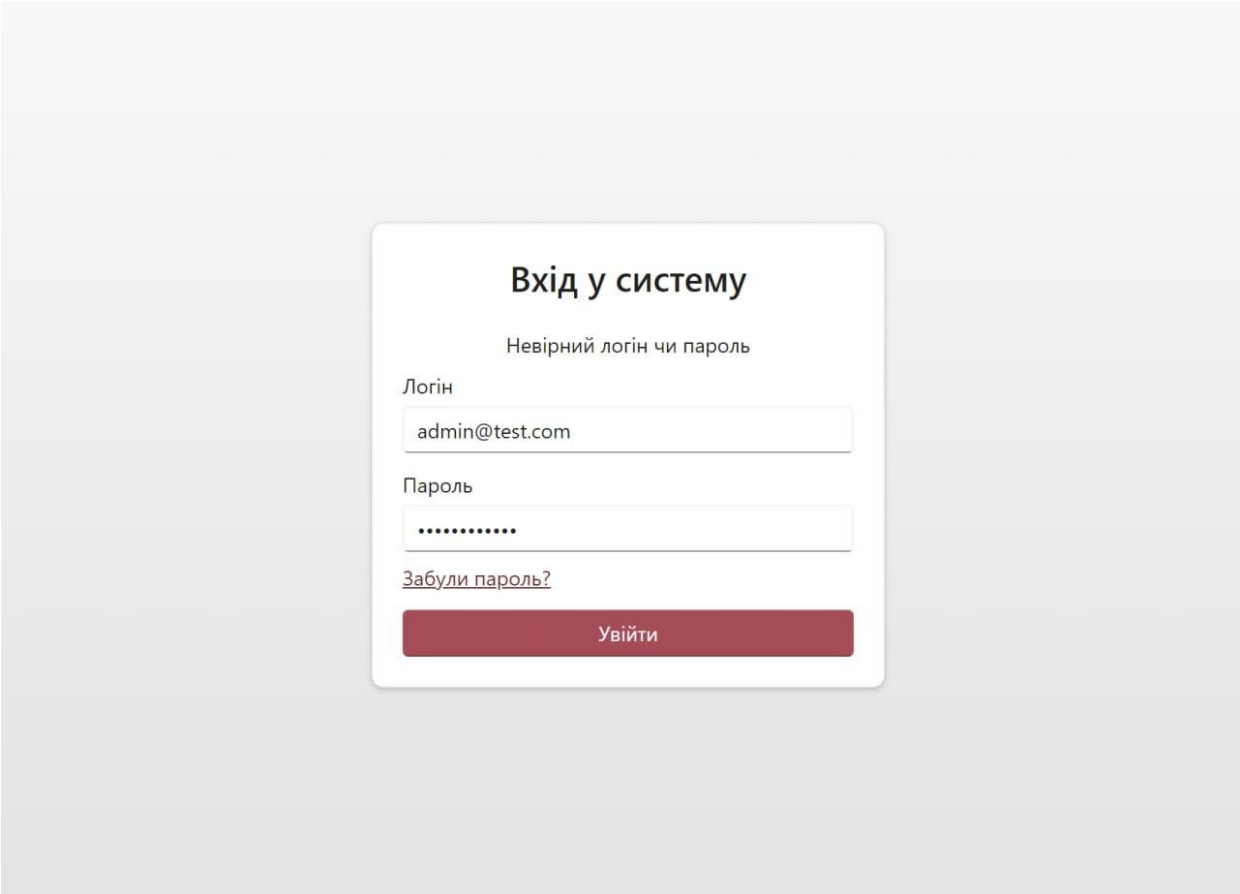


Рисунок 5.10 – Виправлений баг на скріні логін (рисунок виконаний самостійно)

Крім цього, було виявлено баг який дозволяв додавати колонки з однаковим порядковим номером, що впливало на правильність роботи системи. Проблема полягала у відсутності контролю за унікальністю на рівні бази даних. Для вирішення цієї проблеми було додано унікальний індекс (unique constraint) на поле station_id в таблиці pumps. Це дозволило автоматично перевіряти унікальність номерів перед додаванням нових записів і забезпечило відсутність дублів:

```
CREATE UNIQUE INDEX unique_station_id ON pumps (station_id)
```

Додатково, для уникнення схожих проблем в інших частинах системи, було проведено перевірку всіх існуючих таблиць БД на відсутність унікальних індексів на необхідних полях.

Ручне тестування веб-застосунку виявило та усунуло критичні проблеми, що підвищило стабільність і безпеку системи. Функціональне тестування перевірило основні функції, нефункціональне забезпечило зручність використання, а регресійне виявило нові дефекти після виправлень. Основні проблеми включали небезпечні повідомлення при авторизації, які дозволяли зловмисникам дізнатися про існування облікових записів, та відсутність унікальних індексів у базі даних, що призводило до дублювання колонок. Вирішення цих проблем підвищило надійність і готовність системи до використання.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було спроектовано та розроблено програмну систему, спрямовану на полегшення взаємодії між клієнтом та працівником автозаправної станції (АЗС). У рамках роботи було проведено аналіз предметної галузі, а також аналіз конкурентів із виявленням їх переваг та недоліків. На основі отриманих даних було сформовано вимоги до програмного продукту, спроектовано архітектуру програмного забезпечення, реалізовано веб-додаток та виконано тестування.

Для зберігання даних було спроектовано та нормалізовано базу даних, яка відображає всі зв'язки між сутностями обраної предметної галузі. Отримана програмна система є багатофункціональною у використанні працівників. Вона має інтуїтивно зрозумілий інтерфейс, що дозволяє працювати з програмою користувачам з різним рівнем комп'ютерної грамотності, має середні вимоги до апаратного та програмного забезпечення, і забезпечує досить високу швидкість роботи.

Розроблений програмний продукт можна вдосконалити шляхом розширення його наявного функціоналу, наприклад, додавання бонусної системи для клієнтів АЗС, клієнтської частини застосунку та створення нових статистик для покращення контролю над процесом обслуговування.

Усі поставлені вимоги до веб-додатку було виконано. Отже, результатом даної кваліфікаційної роботи став веб-додаток, який спрощує взаємодію між працівником АЗС та клієнтом.

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАННЯ

1. Blazor. URL: https://docs.microsoft.com/en-us/aspnet/core/blazor/ (дата звертання 23.05.2024).

2. Petrosoft web site. URL: https://petrosoftinc.com/optimize-gas-stations/ (дата звертання 23.05.2024).

3. FuelForce web site. URL: https://www.fuelforce.com/ (дата звертання 23.05.2024).

4. Sky Bitz web site. URL: https://www.skybitz.com/petroleum-logistics (дата звертання 23.05.2024).

5. PostgreSQL. URL: https://www.postgresql.org/docs/ (дата звертання 23.05.2024).

6. PostgreSQL Tutorial. URL: https://www.postgresqltutorial.com/ (дата звертання 23.05.2024).

7. .NET Core. URL: https://dotnet.microsoft.com/ (дата звертання 23.05.2024).

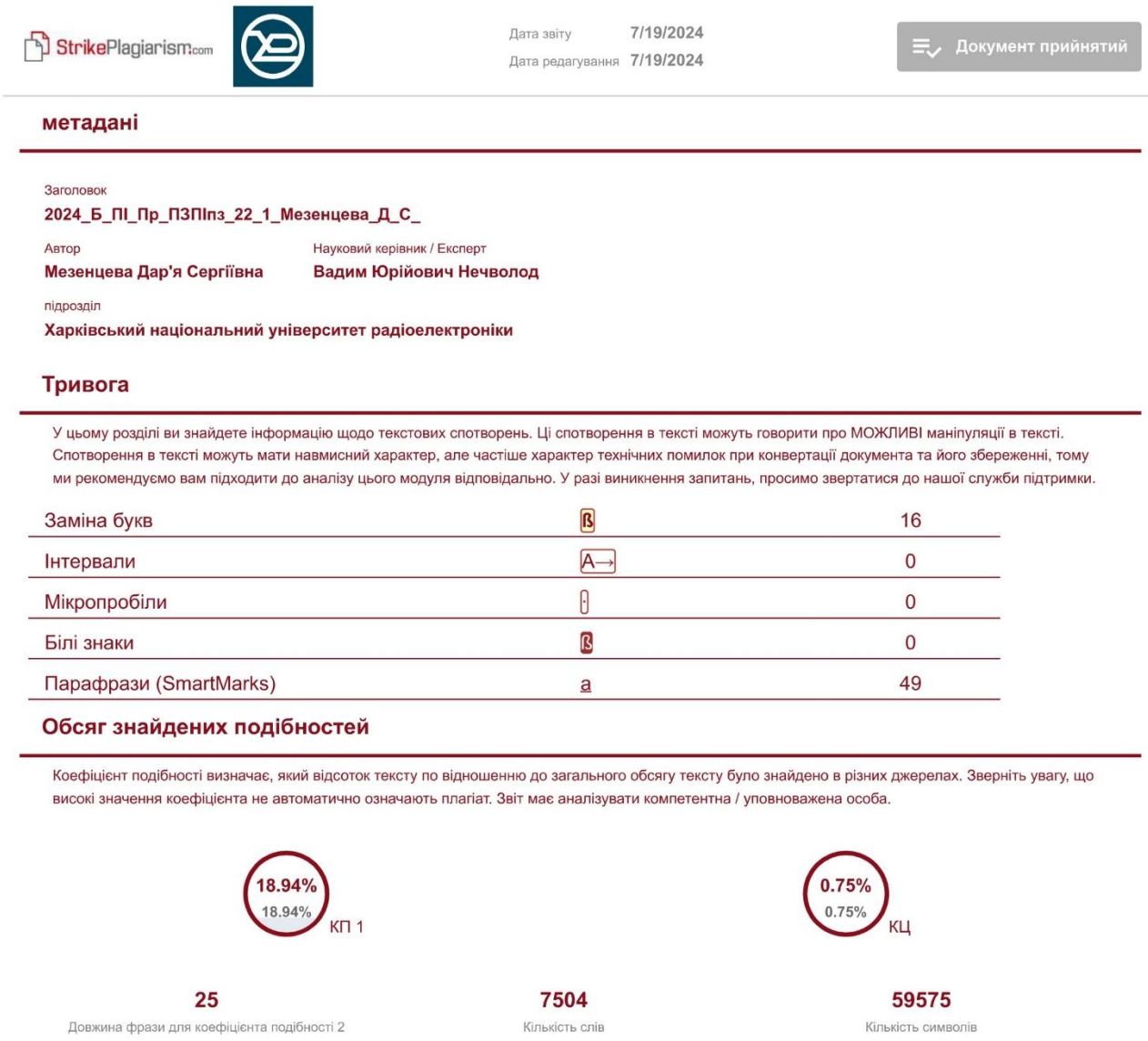
8. Microsoft Docs: .NET API Browser. URL: https://docs.microsoft.com/en-us/dotnet/api/ (дата звертання 23.05.2024).

9. Microsoft Docs: Entity Framework Core. URL: https://docs.microsoft.com/en-us/ef/core/ (дата звертання 23.05.2024).

10. NpgSql GitHub Repository. URL: https://github.com/npgsql/npgsql (дата звертання 23.05.2024).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



StrikePlagiarism.com

Дата звіту 7/19/2024
Дата редагування 7/19/2024

Документ прийнятий

метадані

Заголовок
2024_Б_ПІ_Пр_ПЗПІпз_22_1_Мезенцева_Д_С_

Автор
Мезенцева Дар'я Сергіївна

Науковий керівник / Експерт
Вадим Юрійович Нечволод

підрозділ
Харківський національний університет радіоелектроніки

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	16
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	49

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

18.94% 18.94% КП 1

0.75% 0.75% КЦ

25
Довжина фрази для коефіцієнта подібності 2

7504
Кількість слів

59575
Кількість символів

ДОДАТОК Б

Слайди презентації

Кваліфікаційна робота

Програмна система обліку роботи мережі АЗС

Виконала
ст. гр. ПЗПІпз-22-1
Мезенцева Дар'я

Керівник
доц. кафедри ПІ
Русакова Н.Є.

Мета роботи

Розробка системи для управління та обліку АЗС.

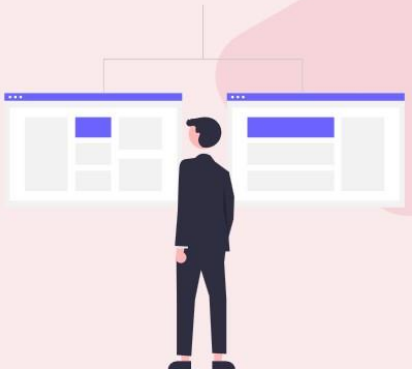
Забезпечення можливості легкого здійснення транзакцій, автоматизації процесу звітності та оптимізації управління співробітниками.

Актуальність

- Зростання потреби в ефективних системах управління для мереж АЗС.
- Висока вартість та складність налаштування існуючих рішень на ринку.
- Необхідність забезпечення інтуїтивно зрозумілого інтерфейсу та специфічних функцій для користувачів.

Конкуренти

Назва компанії	Управління фінансовими транзакціями	Аналітика та звітність	Клієнтське обслуговування	Управління співробітниками
Petrosoft	Високий	Високий	Середній	Середній
FuelForce	Високий	Середній	Низький	Низький
PDI	Високий	Високий	Середній	Середній
OPW	Високий	Високий	Низький	Низький
SkyBitz	Середній	Середній	Низький	Низький



5

Постановка задачі

Завдання полягає в розробці веб-додатку для управління АЗС з метою автоматизації їх роботи.

Співробітник

- авторизація
- зміна та відновлення паролю
- перегляд інформації про станцію
- перегляд та створення транзакцій
- формування звіту по транзакціям

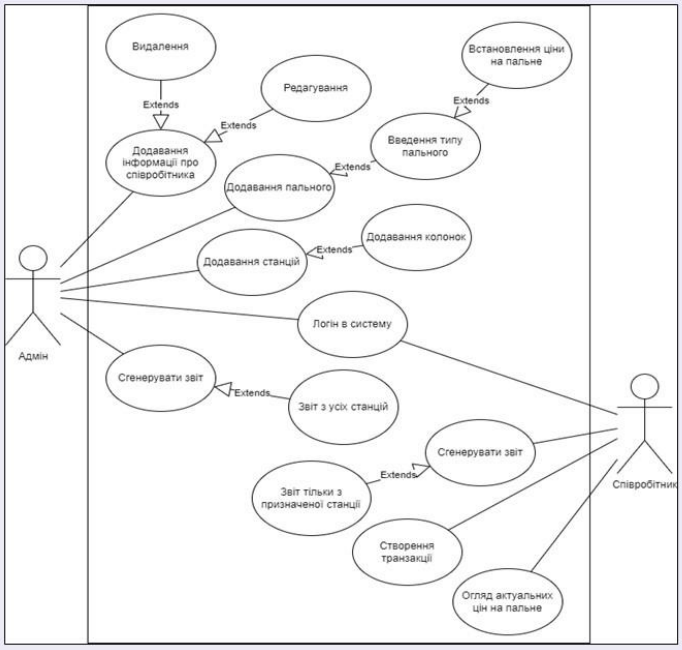
Адміністратор

Всі функції співробітника, а також:

- управління співробітниками та користувачами
- управління станціями так колонками
- управління типами пального
- управління цінами на пальне

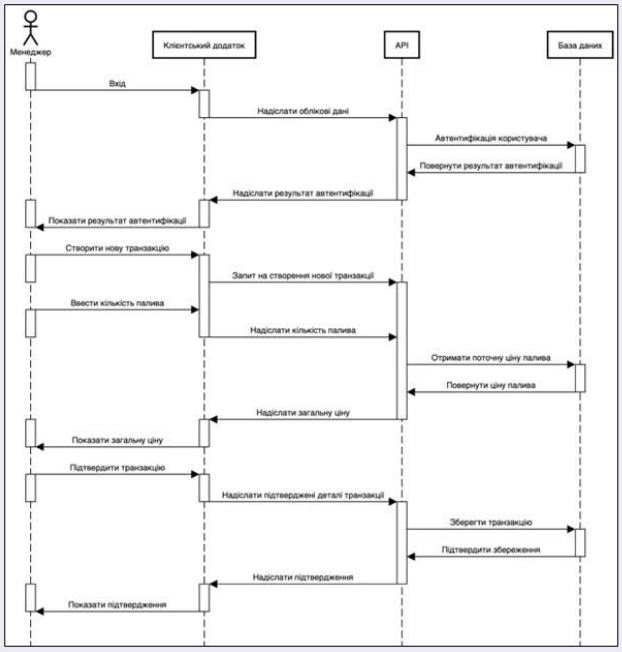
Сценарії використання (use-case)

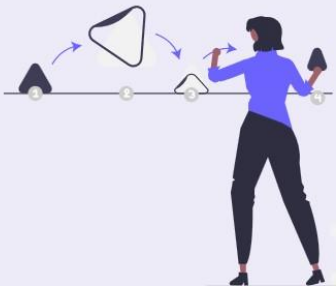
6



Діаграма послідовностей (нова транзакція)

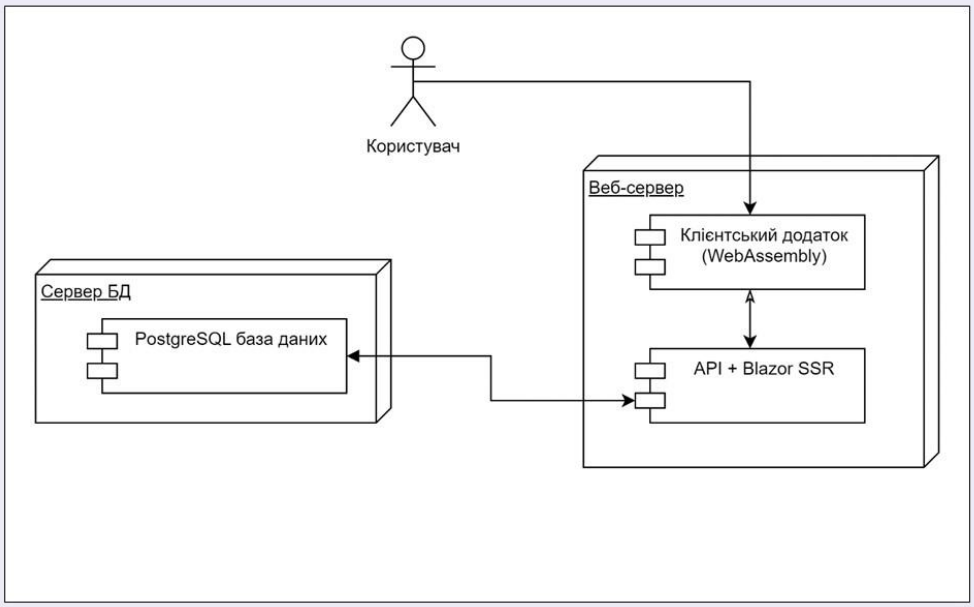
7





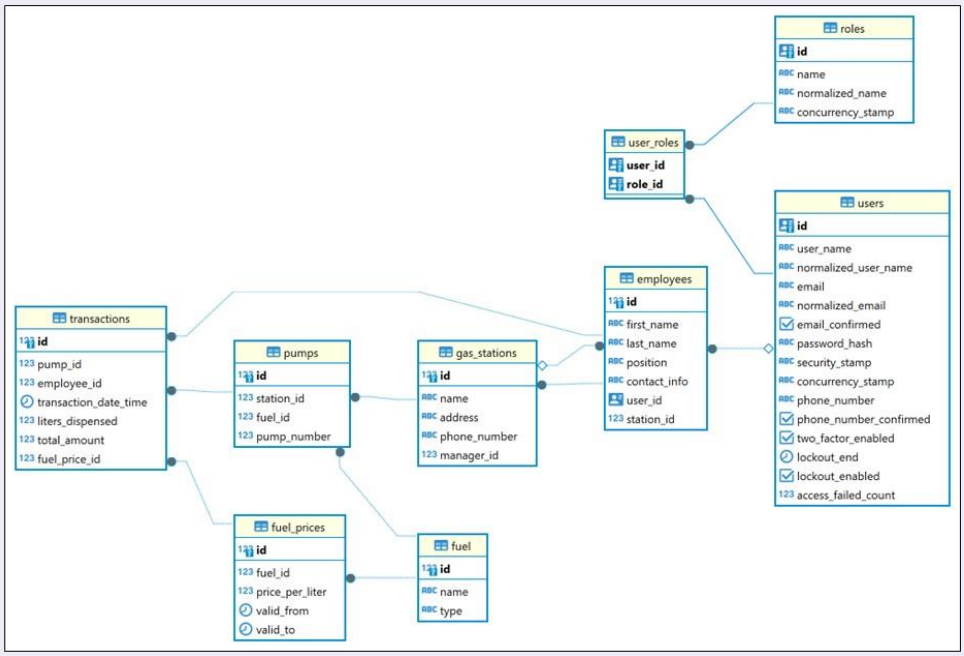
Інфраструктура (компоненти)

8



База даних

9



10

Використані технології

HTML

.NET Core

Blazor

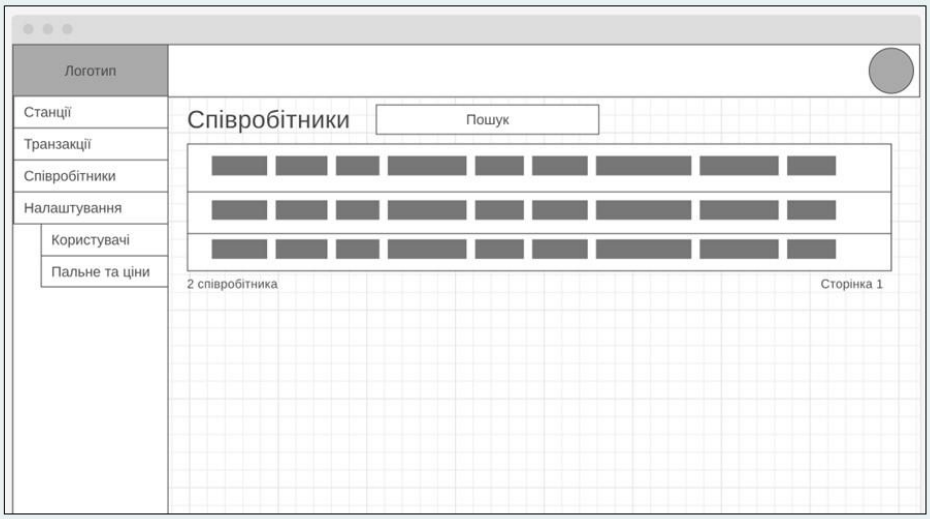
CSS

Fluent UI

PostgreSQL

11

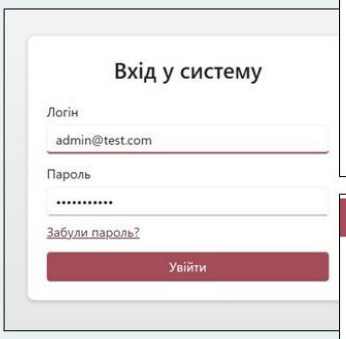
Прототип дизайну

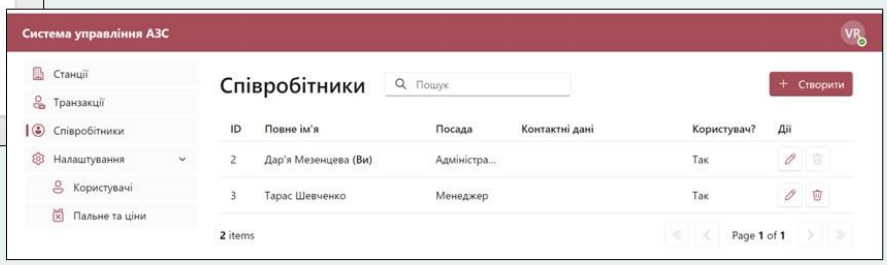


Для створення прототипу використано сервіс wireframe.cc

12

Дизайн системи





Тестування

13

Тест-кейс №1

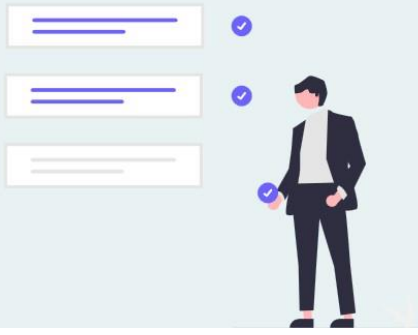
ID :		1	
Назва тест-кейсу:		Вхід у систему з відновленням паролю	
Створений ким:		Мезенцева Дар'я Сергіївна	
Дата створення:		20.06.2024	
Передумова			
№	Опис кроку	Очікуваний результат	Висновок
1	Відкрити веб-застосунок	Сторінка відкрита	Пройдено
2	Користувач має обліковий запис	Обліковий запис існує	Пройдено
Вхід у систему з відновленням паролю			
№	Опис кроку	Очікуваний результат	Висновок
1	Ввести валідний Email в поле «Логін»	Email введен в поле «Логін»	Пройдено
2	Натиснути на кнопку «Увійти»	Отримано помилку «Пароль не введено»	Пройдено
3	Ввести неправильний пароль в поле «Пароль»	Введено неправильний пароль в поле «Пароль»	Пройдено
4	Настиснути на кнопку «Увійти»	Отримано помилку «Невірний пароль»	
5	Натиснути на кнопку «Забули пароль»	Нове модальне вікно відкрито з повідомленням «Для відновлення паролю введіть ваш Email», полем «Email» та кнопкою «Відновити пароль»	
6	Ввести email та натиснути кнопку «Відновити пароль»	На пошту прийшов лист з посиланням на оновлення паролю	Пройдено
7	Відкрити лист та перейти по посиланню	Нова вкладка в браузері відкрилась з двома полями -введенням та підтвердженням нового паролю, кнопка «Оновити пароль»	Пройдено
8	Ввести новий пароль та підтвердити його, натиснути кнопку «Оновити пароль»	Користувач бачить повідомлення про успішне оновлення паролю	Пройдено
9	Зайти на сторінку входу у веб-застосунок	Веб-застосунок відкрито	Пройдено
10	Ввести логін та новий пароль	Логін та пароль введено	Пройдено
11	Натиснути кнопку «Увійти»	Вхід в застосунок успішний	Пройдено
Результат тестування			
Тестувальниця: Мезенцева Д.С.	Дата тестування: 21.06.2024	Результат тесту: ПРОЙДЕНО	

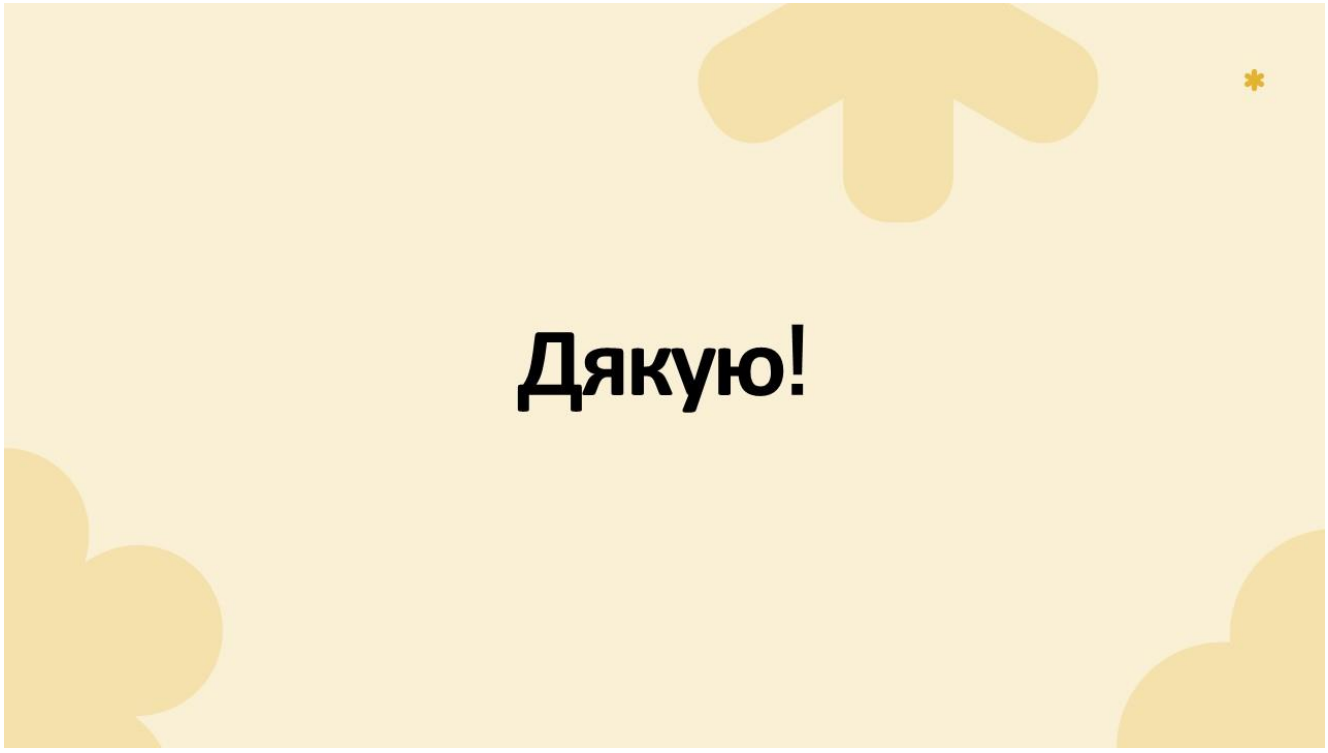
14

Висновки

Було спроектовано та розроблено програмну систему, спрямовану на полегшення взаємодії між клієнтом та працівником автозаправної станції (АЗС).

Було проведено аналіз предметної галузі, а також аналіз конкурентів із виявленням їх переваг та недоліків, сформовано вимоги до програмного продукту, спроектовано архітектуру програмного забезпечення, реалізовано веб-застосунок та виконано тестування.




Дякую!

ДОДАТОК В

Приклад кодів програми

Файл TransactionService.cs (серверна частина)

```
using GasStationManagement.Entities;
using GasStationManagement.Interfaces;
using GasStationManagement.Models.Reports;
using GasStationManagement.Specifications;
using Microsoft.EntityFrameworkCore;
using OfficeOpenXml;
using OfficeOpenXml.Table;

namespace GasStationManagement.Services;

public class TransactionService : ITransactionService
{
    private readonly IRepository<Transaction> _transactionRepository;
    private readonly IAppDbContext _dbContext;

    public TransactionService(IRepository<Transaction>
transactionRepository, IAppDbContext dbContext)
    {
        _transactionRepository = transactionRepository;
        _dbContext = dbContext;
    }

    public async Task<List<Transaction>>
GetTransactionsAsync(CancellationToken cancellationToken = default)
    {
        var spec = new TransactionDefaultSpec();
        return await _transactionRepository.ListAsync(spec,
cancellationToken);
    }

    public async Task<byte[]> GenerateReportAsync(CancellationToken
cancellationToken = default)
    {
        var transactions = await _dbContext.Transactions
            .Include(t => t.Employee)
            .Include(t => t.Pump)
            .ThenInclude(p => p.Station)
            .Include(t => t.Pump)
            .ThenInclude(p => p.Fuel)
            .ThenInclude(f => f.Prices)
            .ToListAsync(cancellationToken);

        var reportData = transactions.Select(t => new
TransactionReportItemDto
        {
            TransactionId = t.Id,
            StationName = t.Pump.Station.Name,
            PumpNumber = t.Pump.PumpNumber,
            LitersDispensed = t.LitersDispensed,
            Employee = $"{t.Employee.FirstName} {t.Employee.LastName}",
```

```
            PricePerLiter = t.Pump.Fuel.Prices
                .First(p => p.ValidFrom <= DateTime.UtcNow && p.ValidTo >=
DateTime.UtcNow)
                .PricePerLiter,
            FuelType = t.Pump.Fuel.Name,
            TotalAmount = t.TotalAmount,
            TransactionDateTime = t.TransactionDateTime
        });

        var commonStatistics = await _dbContext.Transactions
            .GroupBy(_ => 1)
            .Select(g => new TransactionStatisticsDto
            {
                AvgLiters = g.Average(t => t.LitersDispensed),
                MaxLiters = g.Max(t => t.LitersDispensed),
                MinLiters = g.Min(t => t.LitersDispensed),
                AvgAmount = g.Average(t => t.TotalAmount),
                MaxAmount = g.Max(t => t.TotalAmount),
                MinAmount = g.Min(t => t.TotalAmount)
            })
            .ToListAsync(cancellationToken);

        var statisticsByFuelType = (await _dbContext.Transactions
                .Include(t => t.Pump)
                .ThenInclude(t => t.Fuel)
                .ThenInclude(f => f.Prices)
                .ToListAsync(cancellationToken))
            .GroupBy(t => t.Pump.Fuel)
            .Select(g => new GroupingByFuelTypeDto
            {
                FuelType = g.Key.Name,
                Price = g.Key.Prices
                    .First(p => p.ValidFrom <= DateTime.UtcNow && p.ValidTo
>= DateTime.UtcNow)
                    .PricePerLiter,
                TotalTransactions = g.Count(),
                Income = g.Sum(t => t.TotalAmount)
            });

        using var package = new ExcelPackage();

        var commonStatisticsWorksheet =
package.Workbook.Worksheets.Add("Статистика (загальна)");

commonStatisticsWorksheet.Cells["A1"].LoadFromCollection(commonStatistics,
true, TableStyles.Medium13);

commonStatisticsWorksheet.Cells[commonStatisticsWorksheet.Dimension.Address
].AutoFitColumns();

        var reportWorksheet = package.Workbook.Worksheets.Add("Статистика
(за типом пального)");

reportWorksheet.Cells["A1"].LoadFromCollection(statisticsByFuelType, true,
TableStyles.Medium14);

reportWorksheet.Cells[reportWorksheet.Dimension.Address].AutoFitColumns();
```

```
        var transactionsWorksheet =
package.Workbook.Worksheets.Add("Транзакції");
        transactionsWorksheet.Cells["A1"].LoadFromCollection(reportData,
true, TableStyles.Medium15);

transactionsWorksheet.Cells[transactionsWorksheet.Dimension.Address].AutoFi
tColumns();
        transactionsWorksheet.Column(9).Style.Numberformat.Format = "yyyy-
MM-dd HH:mm";

        return await package.GetAsByteArrayAsync(cancellationToken);
    }
}
```

Файл AppDbContext.cs (серверна частина)

```
using System.Reflection;
using GasStationManagement.Entities;
using GasStationManagement.Interfaces;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace GasStationManagement.Data;

public class AppDbContext : IdentityDbContext<User, Role, Guid>,
IAppDbContext
{
    public DbSet<Employee> Employees { get; set; } = null!;

    public DbSet<Fuel> Fuel { get; set; } = null!;

    public DbSet<GasStation> GasStations { get; set; } = null!;

    public DbSet<Pump> Pumps { get; set; } = null!;

    public DbSet<Transaction> Transactions { get; set; } = null!;

    public DbSet<User> Users { get; set; } = null!;

    public DbSet<Role> Roles { get; set; } = null!;

    public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options)
    {

    }

    protected override void OnModelCreating(ModelBuilder builder)
    {

builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());

        base.OnModelCreating(builder);
    }

    public async Task SaveChangesAsync()
    {
        await base.SaveChangesAsync().ConfigureAwait(false);
    }
```

```
}
```

Файл Login.razor (клієнтська частина)

```
@page "/login"
@layout LoginLayout

@using System.ComponentModel.DataAnnotations
@using GasStationManagement.Components.Layout
@using GasStationManagement.Data.Constants
@using GasStationManagement.Entities
@using GasStationManagement.Identity
@using Microsoft.AspNetCore.Identity

@inject SignInManager<User> SignInManager
@inject ILogger<Login> Logger
@inject IdentityRedirectManager RedirectManager

<PageTitle>Вхід у систему</PageTitle>

<FluentCard Width="350px" Class="flex flex-column align-items-center gap-2
h-auto">
    <h3>Вхід у систему</h3>
    <StatusMessage Message="@_errorMessage" />
    <EditForm Model="Input" class="w-full" method="post"
OnValidSubmit="LoginUser" FormName="login">
        <div class="w-full flex flex-column gap-2">
            <div class="w-full">
                <FluentTextField Class="w-full" Name="Input.Email"
Autofocus="true" @bind-Value="@Input.Email" Label="Логін"
TextFieldType="TextFieldType.Email"/>
            </div>
            <div class="w-full">
                <FluentTextField Class="w-full" Name="Input.Password"
@bind-Value="@Input.Password" Label="Пароль"
TextFieldType="TextFieldType.Password"/>
            </div>
        </div>
        <FluentButton Type="ButtonType.Submit" Class="mt-2 w-full"
Appearance="Appearance.Accent">Увійти</FluentButton>
    </EditForm>
</FluentCard>

@code {
    private string? _errorMessage;

    [CascadingParameter]
    private HttpContext HttpContext { get; set; } = default!;

    [SupplyParameterFromForm]
    private InputModel Input { get; set; } = new();

    [SupplyParameterFromQuery]
    private string? Redirect { get; set; }

    protected override async Task OnInitializedAsync()
    {
```

```
        if (HttpContext.User.Identity?.IsAuthenticated is true)
        {
            RedirectManager.RedirectTo(Redirect ?? "/");
        }
    }

    public async Task LoginUser()
    {
        var result = await SignInManager.PasswordSignInAsync(Input.Email,
Input.Password, true, false);

        if (result.Succeeded)
        {
            Logger.LogInformation("User {Login} logged in.", Input.Email);
            RedirectManager.RedirectTo(Redirect ?? "/");
        }
        else
        {
            _errorMessage = "Невірний логін чи пароль";
        }
    }

    private sealed class InputModel
    {
        [Required]
        [EmailAddress]
        public string Email { get; set; } = UserCredentials.Admin.Login;

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; } =
UserCredentials.Admin.Password;
    }
}
```

Файл Program.cs (серверна частина)

```
using System.Reflection;
using GasStationManagement.Components;
using GasStationManagement.Data;
using GasStationManagement.Entities;
using GasStationManagement.Identity;
using GasStationManagement.Interfaces;
using GasStationManagement.Services;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.FluentUI.AspNetCore.Components;
using OfficeOpenXml;

ExcelPackage.LicenseContext = LicenseContext.NonCommercial;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorComponents()
    .AddInteractiveServerComponents();
```

```
builder.Services.AddDbContext<AppDbContext>(options =>

options.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnect
ion")));

builder.Services
    .AddIdentity<User, Role>()
    .AddDefaultTokenProviders()
    .AddRoleManager<RoleManager<Role>>()
    .AddUserManager<UserManager<User>>()
    .AddSignInManager<SignInManager<User>>()
    .AddEntityFrameworkStores<AppDbContext>();

builder.Services.AddScoped<IAppDbContext>(provider =>
provider.GetService<AppDbContext>()!);

builder.Services.ConfigureApplicationCookie(options =>
{
    options.Cookie.HttpOnly = true;
    options.ExpireTimeSpan = TimeSpan.FromMinutes(30);

    options.LoginPath = "/login";
    options.ReturnUrlParameter = "redirect";
    options.AccessDeniedPath = "/access-denied";
    options.SlidingExpiration = true;
});

builder.Services.AddAutoMapper(Assembly.GetExecutingAssembly());

builder.Services.AddCascadingAuthenticationState();

builder.Services.AddScoped<IdentityRedirectManager>();
builder.Services.AddScoped<AuthenticationStateProvider,
IdentityRevalidatingAuthenticationStateProvider>();

builder.Services.AddScoped<DbInitializer>();

builder.Services.AddScoped(typeof(IRepository<>), typeof(Repository<>));
builder.Services.AddScoped<IStationService, StationService>();
builder.Services.AddScoped<ITransactionService, TransactionService>();
builder.Services.AddScoped<IFuelService, FuelService>();
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IEmployeeService, EmployeeService>();
builder.Services.AddScoped<IProcessingService, ProcessingService>();

builder.Services.AddFluentUIComponents();
builder.Services.AddHttpClient();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
```

```
app.UseStaticFiles();

app.UseRouting();
app.MapAdditionalIdentityEndpoints();

app.UseAuthentication();
app.UseAuthorization();

app.UseAntiforgery();

app.MapRazorComponents<App>()
    .AddInteractiveServerRenderMode();

using var scope = app.Services.CreateScope();
var dbContext = scope.ServiceProvider.GetRequiredService<AppDbContext>();
var dbInitializer =
scope.ServiceProvider.GetRequiredService<DbInitializer>();

await dbContext.Database.MigrateAsync();
await dbInitializer.InitAsync();

app.Run();
```