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

Комп'ютерно – інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: «Розробка web сервісу обліку користувачів газопостачальної компанії з використанням Python та РБД»

Виконав(ла): студент(ка) IV курсу, групи СПс-42
спеціальності 121− Інженерія програмного забезпечення
(шифр і назва спеціальності)

	(підпис)	Соловій Т.Б. (прізвище та ініціали)
Керівник	(підпис)	Цебрій О.Р. (прізвище та ініціали)
Нормоконтроль	(підпис)	Стоянов Ю.М. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Петрик М.Р. (прізвище та ініціали)
Рецензент	(підпис)	(прізвище та ініціали)

Тернопіль
2024

РЕФЕРАТ

Бакалаврська робота на тему «Розробка web сервісу обліку користувачів газопостачальної компанії з використанням Python та РБД» містить 50 сторінок, 1 таблицю, 22 рисунків, 1 додаток. Перелік посилань нараховує 13 найменувань.

Актуальність теми дослідження зумовлено необхідністю підвищення ефективності управління споживанням газу шляхом впровадження сучасних інформаційних технологій, зокрема, web сервісів для обліку користувачів. Це дозволить забезпечити точний облік, автоматизацію процесів та покращення взаємодії між газопостачальною компанією та її клієнтами.

Метою роботи є розробка та впровадження web сервісу обліку користувачів газопостачальної компанії на основі Python та реляційних баз даних (РБД), що дозволить автоматизувати облік та покращити якість обслуговування клієнтів.

Для досягненя мети необхідно вирішити наступні завдання:

- Визначити основні вимоги до web сервісу для обліку користувачів газопостачальної компанії;
- Розробити архітектуру та дизайн системи;
- Впровадити інструменти для обробки та зберігання даних за допомогою Python та РБД;
- Провести тестування та аналіз ефективності розробленого сервісу;
- Надати рекомендації щодо вдосконалення та розширення функціональності системи.

Об’єктом дослідження є процеси обліку користувачів газопостачальної компанії.

Предметом дослідження є методи та засоби розробки web сервісу для обліку користувачів з використанням Python та РБД.

У кваліфікаційній роботі було використано аналітичний метод (для вивчення поточного стану області), методи кількісних і якісних оцінок, методи економічного та технічного аналізу (для визначення ефективності запропонованого рішення та оцінювання його впливу на процеси обліку), а також методи бібліографічного аналізу (для вивчення літератури та передового досвіду у цій сфері).

Наукова новизна одержаних результатів дослідження полягає у розробці теоретико-методичних та практичних рекомендацій з впровадження web сервісу обліку користувачів газопостачальної компанії на основі Python та РБД. Запропонована модель системи дозволяє автоматизувати процеси обліку та забезпечує інтерактивний доступ до даних для користувачів і адміністрації компанії.

Практичне значення одержаних результатів полягає у можливості впровадження розробленого web сервісу в газопостачальних компаніях для автоматизації процесів обліку споживачів, що сприятиме покращенню ефективності управління та якості обслуговування клієнтів.

ABSTRACT

A bachelor's thesis on "Development of a web service for accounting for users of a gas supply company using Python and a database" contains 50 pages, 1 table, 22 figures, 1 appendix. The list of references includes 13 titles.

The relevance of the research topic is due to the need to improve the efficiency of gas consumption management through the introduction of modern information technologies, in particular, web services for user accounting. This will ensure accurate accounting, process automation and improved interaction between the gas supply company and its customers.

The aim of the work is to develop and implement a web service for accounting for users of a gas supply company based on Python and relational databases (RDBMS), which will automate accounting and improve the quality of customer service.

To achieve the goal, the following tasks need to be solved:

- Define the basic requirements for a web service for accounting for users of a gas supply company;
- Develop the architecture and design of the system;
- Implement tools for processing and storing data using Python and databases;
- Test and analyze the effectiveness of the developed service;
- Provide recommendations for improving and expanding the functionality of the system.

The object of research is the processes of accounting for users of a gas supply company.

The subject of the study is the methods and tools for developing a web service for user accounting using Python and a database.

The qualification work used the analytical method (to study the current state of the field), methods of quantitative and qualitative assessments, methods of economic and technical analysis (to determine the effectiveness of the proposed solution and assess its impact on accounting processes), as well as methods of bibliographic analysis (to study the literature and best practices in this area).

The scientific novelty of the obtained research results is the development of theoretical, methodological and practical recommendations for the implementation of a web service for accounting for users of a gas supply company based on Python and a database. The proposed model of the system allows automating accounting processes and provides interactive access to data for users and company administration.

The practical significance of the obtained results lies in the possibility of implementing the developed web service in gas supply companies to automate customer accounting processes, which will improve management efficiency and customer service quality.

ЗМІСТ

ВСТУП

Розвиток інформаційних технологій значно вплинув на всі сфери діяльності людини, включаючи сферу енергетики та газопостачання. В сучасних умовах підвищення ефективності управління підприємствами та організаціями вимагає застосування інноваційних підходів до обліку і управління користувачами. Однією з таких інновацій є розробка та впровадження web сервісів, які дозволяють автоматизувати процеси обліку, спрощувати взаємодію з клієнтами та підвищувати загальну ефективність роботи компанії.

Актуальність дослідження зумовлена необхідністю модернізації існуючих систем обліку користувачів у газопостачальних компаніях. Існуючі системи часто мають ряд недоліків, таких як складність у пошуку оплат по адресі, недостатня інтеграція з сучасними платіжними системами та відсутність ефективного розподілу ролей серед користувачів. Враховуючи ці проблеми, розробка нового web сервісу на базі сучасних технологій, таких як Python та реляційні бази даних (РБД), є надзвичайно важливою і своєчасною.

Метою даної роботи є створення ефективного web сервісу для обліку користувачів газопостачальної компанії, який забезпечить покращення процесу обліку, спрощення пошуку оплат, можливість прямої оплати через систему, а також реалізацію механізмів розподілу користувачів на ролі з відповідними правами доступу.

Завдання дослідження включають:

- Аналіз предметної області та існуючих систем обліку користувачів у газопостачальних компаніях;
- Розробку концепції та структури web сервісу з використанням мови програмування Python та реляційної бази даних;
- Проектування та реалізацію функціоналу для розподілу користувачів на ролі та забезпечення інформаційної безпеки;

- Тестування та впровадження розробленого web сервісу у практичну діяльність газопостачальної компанії;

Наукова новизна одержаних результатів полягає у створенні нової системи обліку користувачів, яка враховує сучасні вимоги до інформаційної безпеки, зручності використання та інтеграції з платіжними системами. Практичне значення роботи полягає у можливості використання розробленого web сервісу для покращення процесів обліку та управління користувачами в газопостачальних компаніях, що сприятиме підвищенню ефективності їхньої діяльності та покращенню якості обслуговування клієнтів.

ПЕРЕЛІК СКОРОЧЕНЬ

HTML (HyperText Markup Language) — мова розмітки гіпертексту — мова для створення структури веб-сторінок.

CSS (Cascading Style Sheets) — каскадні таблиці стилів — мова для опису вигляду веб-сторінок.

API (Application Programming Interface) — інтерфейс програмування додатків — набір інструментів для взаємодії програмних компонентів.

VS Code (Visual Studio Code) — візуальний студійний код — редактор коду, розроблений Microsoft.

ORM (Object-Relational Mapping) — об’єктно-реляційне відображення — технологія для перетворення даних між несумісними типами систем у об’єктно-орієнтованих мовах програмування.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Принцип взаємодії користувачів в онлайн системах газопостачання

Сучасні онлайн системи газопостачання є складними багатофункціональними платформами, які забезпечують ефективну взаємодію між користувачами та компаніями, що надають послуги газопостачання. Принципи взаємодії користувачів у таких системах ґрунтуються на низці ключових аспектів, які забезпечують зручність, безпеку та ефективність процесів обліку та управління [1].

Перш за все, важливим елементом будь-якої онлайн системи є розподіл ролей та доступу. У системі газопостачання це питання вирішується за рахунок чіткого визначення категорій користувачів, кожна з яких має свої права та обов’язки. Адміністратори мають повний доступ до всіх функцій системи, можуть керувати користувачами, налаштовувати параметри системи, переглядати та редагувати дані. Оператори регіональних центрів мають обмежений доступ, можуть здійснювати облік користувачів та платежів у своєму регіоні, переглядати відповідну інформацію та надавати підтримку клієнтам. Кінцеві споживачі газу, які є звичайними користувачами, можуть переглядати свої дані, здійснювати оплату, подавати заявки на технічне обслуговування та отримувати інформацію про стан свого рахунку.

Забезпечення інформаційної безпеки є критично важливим аспектом роботи онлайн систем газопостачання. Для цього впроваджуються різноманітні заходи, такі як аутентифікація та авторизація користувачів. Кожен користувач має проходити процедуру аутентифікації для доступу до системи, що дозволяє визначити його особу та підтвердити його права. Авторизація визначає рівень доступу до інформації та функцій системи, що дозволяє запобігти несанкціонованому доступу до конфіденційних даних. Шифрування даних, які передаються між користувачем і системою, є ще одним важливим заходом, що забезпечує захист від несанкціонованого доступу. Логування дій користувачів

дозволяє фіксувати всі дії у системі, що забезпечує можливість відстеження та аналізу у разі інцидентів.

Зручність та доступність є одними з основних вимог до онлайн систем газопостачання. Система повинна мати інтуїтивний інтерфейс, що дозволяє користувачам легко знаходити потрібну інформацію та здійснювати необхідні дії. Важливою характеристикою є також мультиплатформність, що забезпечує доступність системи з різних пристроїв, включаючи комп'ютери, планшети та смартфони. Це дозволяє користувачам взаємодіяти з системою у зручний для них спосіб. Цілодобовий доступ до системи є ще одним важливим аспектом, що дозволяє користувачам отримувати доступ до своїх даних та здійснювати операції у будь-який час [2-3].

Для підвищення ефективності роботи онлайн систем газопостачання впроваджується автоматизація ряду процесів. Це включає автоматичне нарахування та облік оплат, що зменшує ймовірність помилок та спрощує процес управління фінансами. Інтеграція з різними платіжними системами дозволяє користувачам здійснювати оплату безпосередньо через систему, що спрощує процес взаємодії та підвищує його зручність.

Таким чином, принципи взаємодії користувачів в онлайн системах газопостачання спрямовані на забезпечення зручності, безпеки та ефективності процесів обліку та управління. Це сприяє покращенню якості обслуговування клієнтів та підвищенню загальної ефективності роботи газопостачальних компаній.

1.2. Огляд та аналіз аналогових систем

Сучасні онлайн системи газопостачання в Україні забезпечують важливі функції для ефективного управління та обліку споживання газу. Для аналізу обрано декілька українських онлайн систем, які мають свої переваги та недоліки, що

дозволяє оцінити їх функціональність та визначити можливості для покращення. Нижче представлено огляд трьох таких систем.

«104.ua» є однією з найбільш відомих систем для обліку та управління споживанням газу в Україні. Вона дозволяє користувачам реєструвати свої рахунки, подавати показники лічильника, переглядати історію сплат та здійснювати онлайн-платежі [4] (рис. 1.1).

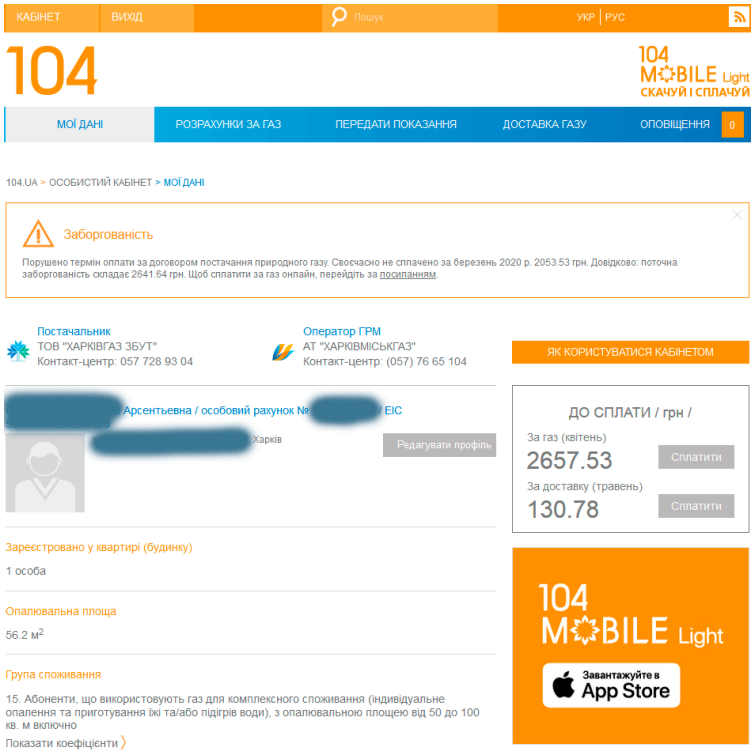

Рисунок 1.1 — Інтерфейс системи «104.ua»

Переваги:

- Простий та зручний інтерфейс: користувачам легко взаємодіяти з системою;
- Можливість подачі показників лічильника онлайн: спрощує процес для споживачів;
- Інтеграція з популярними платіжними системами: дозволяє здійснювати оплату безпосередньо через систему;

- Підтримка мобільних додатків: користувачі можуть використовувати систему з будь-якого пристрою.

Недоліки:

- Обмежена функціональність для операторів регіональних центрів: ускладнює управління обліком на рівні регіонів;
- Відсутність розширених функцій для аналітики та звітності: обмежує можливості для управління даними;
- Недостатній рівень персоналізації інтерфейсу: може бути незручним для різних категорій користувачів.

«MyGas» від Нафтогаз є ще однією популярною системою для обліку споживання газу. Вона надає можливість користувачам реєструвати свої рахунки, переглядати дані про споживання, подавати показники лічильника та здійснювати онлайн-платежі [5] (рис. 1.2).

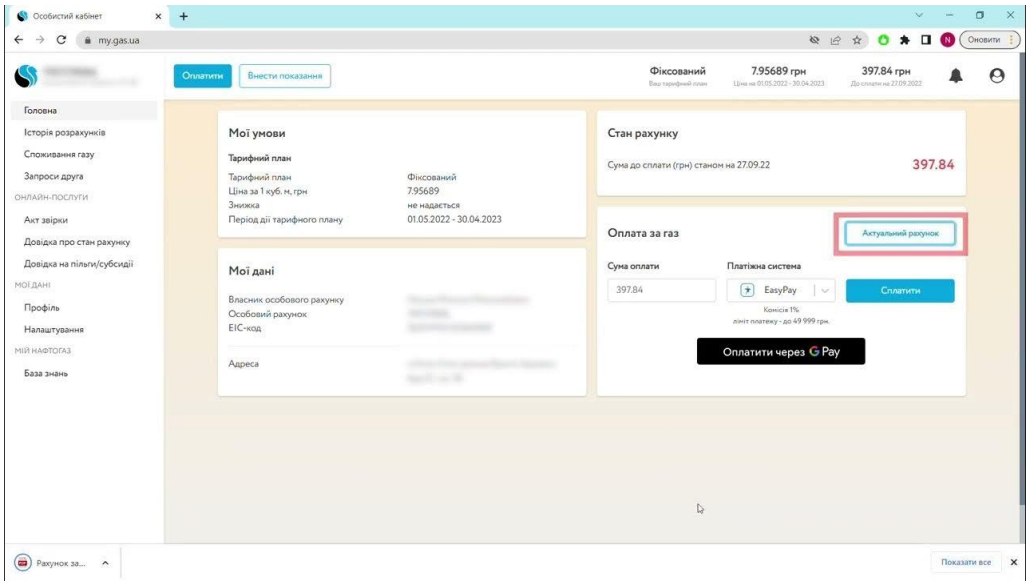

Рисунок 1.2 — Інтерфейс системи «MyGas»

Переваги:

- Наявність мобільного додатку: дозволяє користувачам здійснювати операції з будь-якого пристрою;

- Підтримка багатомовного інтерфейсу: робить систему доступною для ширшої аудиторії;
- Автоматичне нагадування про подачу показників лічильника та сплату рахунків: підвищує зручність для користувачів.

Недоліки:

- Відсутність можливості управління користувачами та їхніми правами доступу: обмежує функціональність для адміністраторів;
- Обмежений набір функцій для аналізу даних та формування звітів: ускладнює управління та контроль за споживанням газу;
- Недостатній рівень захисту даних: може призвести до проблем з безпекою та конфіденційністю інформації.

Обидві розглянуті системи мають свої переваги та недоліки. «104.ua» та «MyGas» забезпечують основні функції для обліку та управління споживанням газу, проте всі системи мають обмеження щодо функціональності для операторів та адміністраторів, а також недостатній рівень аналітики та персоналізації. Ці недоліки відкривають можливості для розробки нової системи, яка буде забезпечувати розширену функціональність, підвищений рівень безпеки та зручності для користувачів, що зробить її більш ефективною та конкурентоспроможною на ринку.

1.3. Формування задач розробки

В результаті аналізу аналогових систем для обліку та управління споживанням газу виявлено декілька аспектів, які потребують вдосконалення. Ці моменти були враховані при розробці нової системи «GasInfo», що дозволить створити більш функціональну та зручну платформу.

Однією з основних задач розробки є чіткий розподіл ролей між користувачами та адміністраторами системи. Це дозволить забезпечити, безпеку та контроль доступу. Адміністратори матимуть повний доступ до налаштувань системи, управління користувачами та перегляду детальних звітів, тоді як звичайні користувачі зможуть взаємодіяти лише з власними даними та базовими функціями системи. Підвищення ефективності: Розподіл ролей сприятиме більш ефективному управлінню системою та оперативному вирішенню проблем, що виникають. Користувачі бачитимуть лише ті функції, які їм потрібні, що зробить інтерфейс менш перевантаженим та більш інтуїтивно зрозумілим.

Ще одним важливим аспектом є спрощення інтерфейсу системи, щоб він був зрозумілим та легко використовуваним для всіх категорій користувачів. Всі елементи інтерфейсу мають бути розташовані логічно та зручно, щоб користувачі могли швидко знайти необхідну інформацію та функції. Відмова від зайвих елементів, що можуть відволікати або заплутувати користувачів. Усі важливі функції повинні бути доступними з головного екрану.

Для забезпечення більшої прозорості та зручності користування, в систему «GasInfo» будуть додані елементи візуалізації даних. Візуалізація споживання газу, витрат та змін тарифів у вигляді графіків та діаграм дозволить користувачам легше розуміти та аналізувати свої дані.

Користувачі зможуть бачити актуальну інформацію про споживання газу та витрати в режимі реального часу.

Врахування цих аспектів дозволить створити систему «GasInfo», яка буде більш функціональною, зручною та ефективною у порівнянні з існуючими аналогами, забезпечуючи високу якість обслуговування та задоволення потреб користувачів.

РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1. Визначення вимог до системи

Вимоги до системи є ключовими аспектами, які визначають її функціональність, продуктивність, зручність використання та інші важливі характеристики. Вимоги поділяються на дві основні категорії: функціональні та нефункціональні. Функціональні вимоги описують конкретні функції або послуги, які система повинна виконувати. Вони визначають, що система повинна робити для задоволення потреб користувачів. Нефункціональні вимоги описують критерії, які можуть використовуватися для оцінки роботи системи, але не є безпосередньо пов'язаними з конкретною функціональністю. Вони охоплюють аспекти, такі як продуктивність, безпека, зручність використання та інші.

Нижче наведено таблицю з прикладами функціональних та нефункціональних вимог до системи «GasInfo» (див. табл. 2.1).

Таблиця 2.1

Вимоги до системи «GasInfo»

Функціональні вимоги	Нефункціональні вимоги
1. Реєстрація користувачів	1. Високий рівень безпеки даних
2. Авторизація користувачів	2. Система повинна бути доступною 24/7
3. Створення лічильників користувачами	3. Інтерфейс повинен бути інтуїтивно зрозумілим
4. Створення квитанцій користувачами	4. Час відгуку системи не більше 2 секунд
5. Введення даних для лічильників	5. Підтримка різних браузерів
6. Введення даних для квитанцій	6. Адаптивний дизайн для різних пристроїв
7. Видалення лічильників та квитанцій	7. Легкість інтеграції з іншими системами
8. Перегляд статистики по графікам та діаграмам	8. Висока надійність системи (мінімум простоїв)
9. Динаміка використання газу по датам	9. Легкість в обслуговуванні та оновленні
10. Динаміка цін за куб газу	10. Масштабованість системи

Продовження таблиці 2.1

11. Використання кожного лічильника	11. Захист від несанкціонованого доступу
12. Перегляд даних у особистому профілі	12. Чітка документація для користувачів
13. Відображення дати реєстрації користувача	13. Відповідність нормативним вимогам
14. Відображення загальної кількості лічильників	14. Можливість персоналізації інтерфейсу
15. Відображення загальної кількості квитанцій	15. Висока продуктивність під навантаженням

Визначення вимог до системи «GasInfo» є критично важливим етапом розробки, який забезпечує чітке розуміння функцій та характеристик, необхідних для задоволення потреб користувачів. Функціональні вимоги описують конкретні операції та сервіси, які система повинна виконувати, тоді як нефункціональні вимоги визначають стандарти якості, продуктивності, безпеки та зручності використання. Усі ці вимоги разом створюють основу для успішної розробки ефективної та конкурентоспроможної системи.

Для кращого розуміння взаємодії користувачів з системою, доцільно використовувати діаграми варіантів використання (Use Case Diagrams). Ці діаграми показують, як різні ролі (користувачі та адміністратори) взаємодіють з системою, включаючи всі можливі сценарії використання. Вони допомагають візуалізувати, які функції доступні для кожної ролі та як ці функції взаємодіють між собою (рис. 2.1). Діаграма варіантів використання для системи «GasInfo» ілюструє взаємодію користувачів та адміністраторів з різними функціональними можливостями системи. Вона відображає ключові сценарії використання системи, розподіляючи їх між двома основними акторами: користувачами та адміністраторами. Користувачі мають доступ до основних функцій, необхідних для управління своїми лічильниками та квитанціями. Це включає можливості реєстрації та входу в систему, створення та видалення лічильників і квитанцій, а також перегляд особистих даних та статистики використання газу. Важливою частиною є можливість вибору субсидій і пільг під час створення лічильників та квитанцій, що

дозволяє користувачам оптимізувати свої витрати. Адміністратори системи мають доступ до функцій управління користувачами, включаючи перегляд списку всіх користувачів, пошук та видалення користувачів. Крім того, адміністратори можуть переглядати та підтверджувати інформацію щодо лічильників та квитанцій користувачів, забезпечуючи тим самим контроль і точність даних у системі.

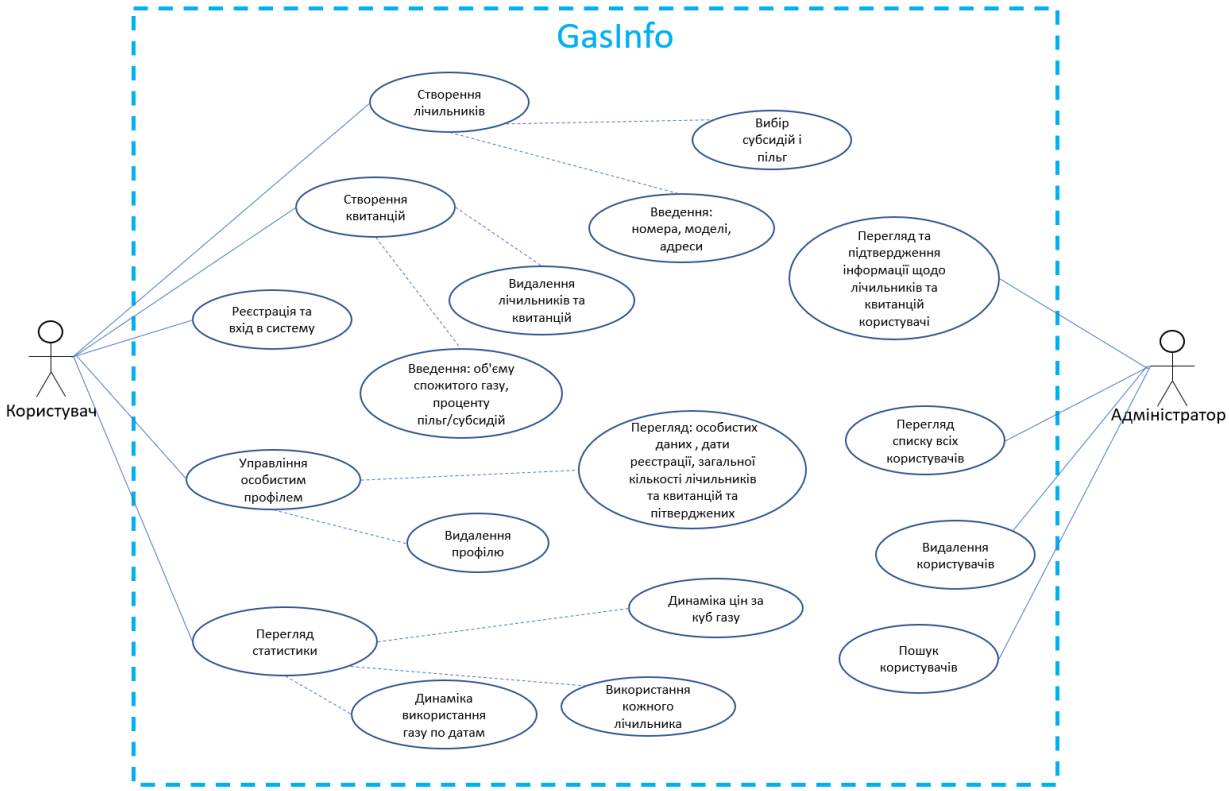

Рисунок 2.1 — Діаграма варіантів використання

Діаграма відображає основні сценарії, такі як реєстрація та вхід в систему, створення та видалення лічильників і квитанцій, перегляд статистики та управління особистим профілем для користувачів. Використання стрілок та асоціацій на діаграмі показує, як різні сценарії взаємопов’язані. Ця діаграма варіантів використання наочно демонструє функціональні можливості системи «GasInfo», допомагаючи зрозуміти взаємодію користувачів з системою та ролі адміністраторів у підтримці її функціонування.

2.2. Архітектура та ключові компоненти системи

Архітектура системи «GasInfo» базується на використанні веб-фреймворка Flask для Python, а також мов програмування JavaScript, HTML та CSS для реалізації інтерфейсу користувача. Основна мета такої архітектури забезпечити гнучкість, масштабованість та зручність використання системи для обліку та управління споживанням газу.

Архітектурні компоненти системи:

1. Клієнтська частина (Front-end)
    - Інтерфейс користувача: Розроблений за допомогою HTML, CSS та JavaScript, інтерфейс забезпечує зручний доступ користувачів до функцій системи, таких як реєстрація, вхід, створення лічильників та квитанцій, перегляд статистики, та управління особистим профілем.
    - Візуалізація даних: Використовуються графіки та діаграми для надання користувачам наочної інформації про споживання газу, динаміку цін та обсяги споживання по кожному лічильнику;
2. Серверна частина (Back-end)
    - Flask фреймворк: Відповідає за обробку запитів від клієнтської частини, управління сесіями користувачів, обробку даних та взаємодію з базою даних.
    - База даних: Зберігає всю інформацію про користувачів, лічильники, квитанції, та інші дані. Для забезпечення швидкого доступу та надійного зберігання використовується реляційна база даних;
3. Адміністративна частина
    - Інтерфейс адміністратора: Забезпечує доступ до списку користувачів, можливість видалення користувачів, пошук користувачів та підтвердження інформації щодо лічильників та квитанцій;

- Управління ролями: Реалізовано чіткий розподіл ролей між користувачами та адміністраторами, що забезпечує безпеку та правильний доступ до функцій системи.

2.3. Створення бази даних та визначення залежностей

При розробці системи «GasInfo» важливим етапом є створення бази даних, яка буде зберігати всю необхідну інформацію про користувачів, лічильники та квитанції. Правильна структура бази даних забезпечує ефективність та зручність роботи з даними, а також гарантує безпеку та цілісність інформації. База даних складається з трьох основних таблиць: users, meters та receipts. Кожна таблиця виконує свою функцію і містить певні поля, які забезпечують зберігання необхідної інформації (рис. 2.2).

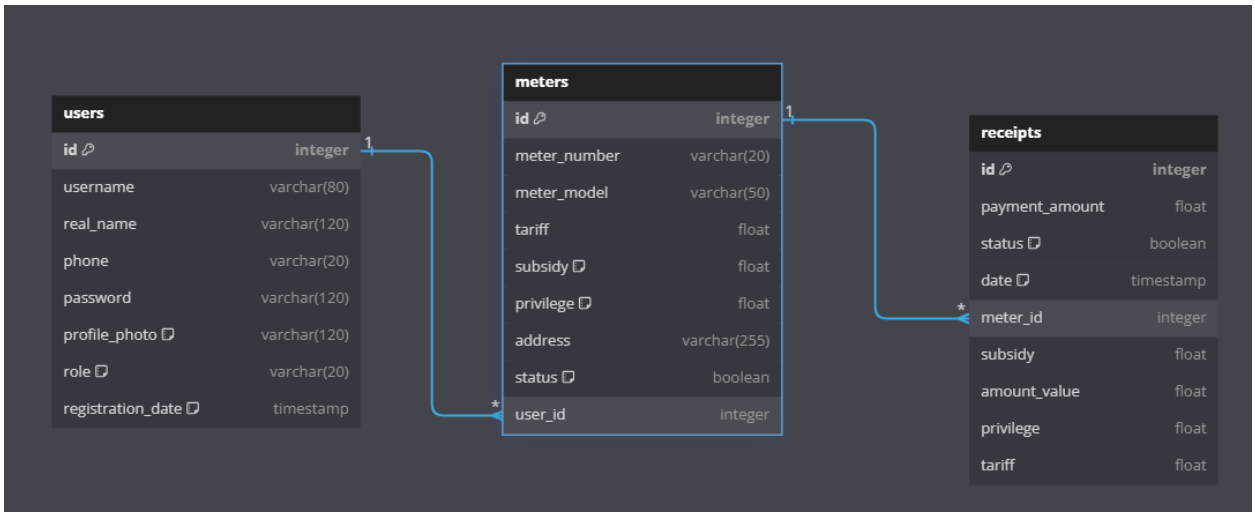

Рисунок 2.2 — Структура бази даних

Таблиця users, яка зберігає інформацію про користувачів системи, та містить наступні поля:

- id: Унікальний ідентифікатор користувача (первинний ключ);
- username: Ім’я користувача (унікальне);

- real_name: Справжнє ім’я користувача;
- phone: Номер телефону користувача;
- password: Пароль користувача;
- profile_photo: Шлях до фотографії профілю користувача (за замовчуванням user.png);
- role: Роль користувача в системі (за замовчуванням user);
- registration_date: Дата реєстрації користувача (за замовчуванням поточна дата).

Таблиця meters, зберігає інформацію про лічильники газу, зареєстровані користувачами, містить наступні поля:

- id: Унікальний ідентифікатор лічильника (первинний ключ);
- meter_number: Номер лічильника (унікальний);
- meter_model: Модель лічильника;
- tariff: Тариф на газ;
- subsidy: Субсидія (за замовчуванням 0);
- privilege: Пільга (за замовчуванням 0);
- address: Адреса встановлення лічильника;
- status: Статус підтвердження лічильника (за замовчуванням false);
- user_id: Ідентифікатор користувача, до якого прив’язаний лічильник (зовнішній ключ).

Таблиця receipts, зберігає інформацію про квитанції, створені користувачами для лічильників, містить такі поля:

- id: Унікальний ідентифікатор квитанції (первинний ключ);
- payment_amount: Сума оплати;
- status: Статус підтвердження квитанції (за замовчуванням false);
- date: Дата створення квитанції (за замовчуванням поточна дата);
- meter_id: Ідентифікатор лічильника, до якого відноситься квитанція (зовнішній ключ);
- subsidy: Субсидія (може бути відсутня);

- amount_value: Обсяг спожитого газу;
- privilege: Пільга (може бути відсутня);
- tariff: Тариф на газ.

Залежності між таблицями визначаються за допомогою зовнішніх ключів. Поле user_id в таблиці meters встановлює зв’язок між таблицями users і meters. Це означає, що кожен лічильник прив’язаний до конкретного користувача. Поле meter_id в таблиці receipts встановлює зв’язок між таблицями meters і receipts. Це означає, що кожна квитанція прив’язана до конкретного лічильника.

Структура бази даних системи «GasInfo» забезпечує ефективне зберігання та управління даними користувачів, лічильників та квитанцій. Завдяки чітким залежностям між таблицями та визначенню ролей користувачів і адміністраторів, система забезпечує необхідну функціональність для обліку та контролю споживання газу, а також зручність та безпеку використання.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Вибір інструментів реалізації

При розробці системи «GasInfo» було використано кілька ключових технологій та інструментів, які забезпечили ефективну реалізацію функціональності, зручний інтерфейс користувача та надійне зберігання даних. Нижче описані основні інструменти та бібліотеки, які були використані в проекті.

Flask є легким мікрофреймворком для веб-розробки на мові програмування Python. Він забезпечує необхідні інструменти та можливості для створення веб-додатків, дозволяючи розробникам зосередитися на основній логіці без надмірної складності [6]. Основні переваги Flask включають:

- Легкість та гнучкість: Flask не накладає жорстких обмежень на структуру проекту, що дозволяє легко адаптуватися до потреб конкретного додатка;
- Розширюваність: Завдяки численним плагінам та бібліотекам, Flask можна легко розширювати, додаючи нові можливості;
- Простота у використанні: Зручний API та зрозуміла документація роблять Flask доступним навіть для новачків.

Мова програмування Python була обрана як основна мова для реалізації серверної частини системи завдяки своїм численним перевагам:

- Простота та читабельність коду: Python відомий своєю зрозумілою та легкою для читання синтаксичною структурою;
- Широкий набір бібліотек: Наявність великої кількості бібліотек для роботи з даними, веб-розробки, машинного навчання та інших областей;
- Швидкий розвиток та підтримка спільноти: Python має велику спільноту розробників, що забезпечує постійну підтримку та розвиток мови.

Мова програмування JavaScript була використана для реалізації динамічної поведінки на стороні клієнта. Вона забезпечує інтерактивність та динамічну взаємодію з користувачем, дозволяючи створювати інтерактивні інтерфейси.

HTML (HyperText Markup Language) та CSS (Cascading Style Sheets) були використані для створення структури та стилізації веб-сторінок. HTML забезпечує основу для розміщення контенту, тоді як CSS відповідає за візуальне оформлення та розташування елементів на сторінці.

Visual Studio Code (VS Code) було обрано як основне середовище розробки для проекту «GasInfo» завдяки своїм численним перевагам [7]:

- Легкість та швидкість: VS Code є легким та швидким редактором коду, що забезпечує зручне і ефективне середовище для розробки;
- Розширюваність: Велика кількість розширень, що дозволяють додати підтримку різних мов програмування, інструментів та бібліотек;
- Інтеграція з Git: Вбудована підтримка системи контролю версій Git дозволяє легко відстежувати зміни в коді та співпрацювати з іншими розробниками;
- Зручність роботи: Зрозумілий інтерфейс та численні функції для підвищення продуктивності, такі як автозаповнення коду, інтелектуальні підказки та налагодження.

Для реалізації різних функцій у системі «GasInfo» були використані наступні бібліотеки [7-13]:

- SQLAlchemy: ORM (Object-Relational Mapping) для Python, що забезпечує інтеграцію з базою даних. Використовується для роботи з моделями бази даних (User, Meter, Receipt) та виконання SQL-запитів і маніпуляцій з даними;
- Werkzeug: Бібліотека для роботи з веб-додатками, яка забезпечує безпечну обробку імен файлів при завантаженні на сервер;
- os: Бібліотека для роботи з файловою системою;
- datetime: Бібліотека для роботи з датами та часом;
- collections: Бібліотека, що забезпечує зручні способи роботи з колекціями даних, такі як defaultdict;

- Swal: Бібліотека для створення модальних вікон та повідомлень, що покращують взаємодію з користувачем, забезпечуючи більш інтуїтивно зрозумілий інтерфейс;
- Chart.js.: Бібліотека для створення графіків та діаграм.

Ці бібліотеки та інструменти були обрані для забезпечення максимальної функціональності, гнучкості та безпеки системи «GasInfo».

3.2. Структура проєкту та ключові елементи системи

При розробці системи «GasInfo» була створена організована структура файлів і тек для забезпечення логічного розподілу компонентів системи (рис. 3.1). Така структура сприяє легкості в навігації та підтримці коду, а також забезпечує зручність розробки та розширення функціональності.

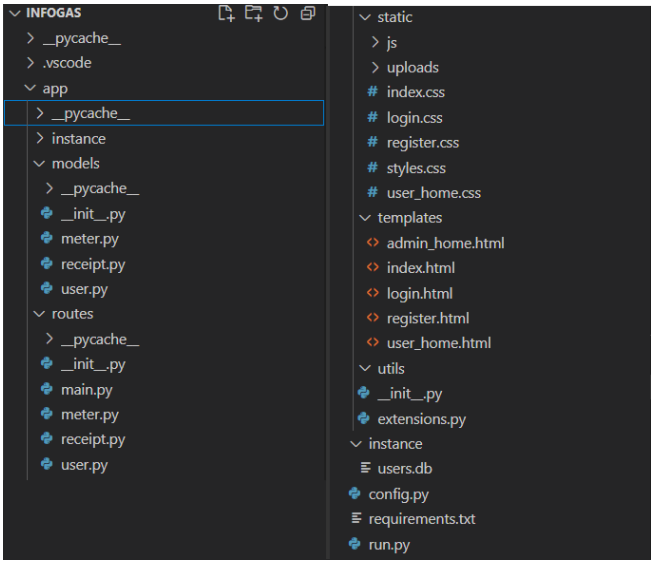

Рисунок 3.1 — Структура файлів та тек у проєкті

У кореневій теці знаходяться такі підтеки:

1. Тека app/: Ця тека містить основні модулі додатку. Вона організована наступним чином:

- models/: Ця підтека містить файли, що описують моделі бази даних. Кожна модель відповідає за взаємодію з конкретною таблицею бази даних:
    - __init__.py: Ініціалізаційний файл, що дозволяє Python розпізнавати цю теку як пакет;
    - meter.py: Модель для лічильників;
    - receipt.py: Модель для квитанцій;
    - user.py: Модель для користувачів.
- routes/: Ця підтека містить файли, що описують маршрути (роути) додатку. Кожен файл відповідає за обробку запитів для певної частини функціональності:
    - __init__.py: Ініціалізаційний файл для пакетів маршрутів;
    - main.py: Основні маршрути додатку;
    - meter.py: Маршрути для роботи з лічильниками;
    - receipt.py: Маршрути для роботи з квитанціями;
    - user.py: Маршрути для роботи з користувачами.

2. Тека instance/: Ця тека використовується для зберігання конфіденційної інформації та файлів бази даних:

- users.db: Файл бази даних SQLite.

3. Тека static/: У цій теці зберігаються статичні файли, такі як CSS, JavaScript та зображення:

- js/: Підтека для JavaScript файлів:
    - admin.js: Скрипти для адміністративної частини додатку;
    - register.js: Скрипти для сторінки реєстрації;
    - user.js: Скрипти для сторінки користувача.
- uploads/: Підтека для збереження завантажених файлів:
    - system_icon/: Зображення системних іконок;
    - uploads_file/: Завантажені файли користувачів;
    - user_icon/: Іконки користувачів.
- index.css: CSS стилі для головної сторінки;

- login.css: CSS стилі для сторінки входу;
- register.css: CSS стилі для сторінки реєстрації;
- styles.css: Загальні стилі для всього додатку.
- user_home.css: CSS стилі для домашньої сторінки користувача.

4. Тека templates/: Ця тека містить HTML шаблони, що використовуються для відображення веб-сторінок користувачам:

- admin_home.html: Шаблон для домашньої сторінки адміністратора;
- index.html: Головний шаблон для домашньої сторінки;
- login.html: Шаблон для сторінки входу;
- register.html: Шаблон для сторінки реєстрації;
- user_home.html: Шаблон для домашньої сторінки користувача.

5. Тека utils/: Містить допоміжні файли та модулі, що використовуються для підтримки основної логіки додатку:

- __init__.py: Ініціалізаційний файл для пакетів утиліт;
- extensions.py: Файл для розширень додатку, таких як налаштування бази даних.

6. Інші важливі файли в кореневій теці:

- config.py: Файл конфігурації, який містить налаштування додатку, такі як параметри бази даних та секретні ключі;
- requirements.txt: Файл залежностей, що описує необхідні для проекту бібліотеки Python;
- run.py: Головний файл для запуску додатку.

Опис ключових елементів системи:

- Моделі (models): Ці файли визначають структуру бази даних. Наприклад, user.py містить модель користувача, meter.py містить модель лічильника, а receipt.py модель квитанції. Вони використовуються для взаємодії з базою даних через ORM SQLAlchemy;
- Маршрути (routes): Файли в цій теці визначають маршрути додатку, обробляють запити користувачів та повертають відповідні відповіді.

Наприклад, main.py містить основні маршрути додатку, а user.py, meter.py і receipt.py містять маршрути для роботи з користувачами, лічильниками та квитанціями відповідно;

- Шаблони (templates): HTML файли, що використовуються для відображення веб-сторінок користувачам. Вони містять шаблони для різних сторінок додатку, таких як головна сторінка, сторінки входу та реєстрації, домашня сторінка користувача та адміністратора;
- Статичні файли (static): CSS, JavaScript та інші файли, які використовуються для стилізації та додавання динамічної поведінки на веб-сторінках. Вони допомагають забезпечити зручний та привабливий інтерфейс користувача;
- Утиліти (utils): Допоміжні файли, що містять різні функції та розширення для підтримки основної логіки додатку. Наприклад, extensions.py може містити налаштування бази даних або інші корисні утиліти;
- Конфігураційний файл (config.py): Містить налаштування додатку, такі як параметри бази даних, секретні ключі тощо.

Організована структура проекту «GasInfo» забезпечує чіткий розподіл функціональних елементів, що сприяє зручності розробки, підтримки та розширення додатку. Використання логічно структурованих тек і файлів дозволяє легко орієнтуватися в коді та забезпечувати високу якість розробки програмного забезпечення.

3.3. Опис та програмна реалізація основних функцій

Першим з чим зіткнеться користувач відкривши систему це авторизація. Функція реєстрації користувача відповідає за створення нового облікового запису.

Вона обробляє дані, введені користувачем у форму реєстрації, перевіряє їх на валідність та зберігає у базі даних:

```
@main.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        realname = request.form['realname']
        phone = request.form['phone']
        password = request.form['password']
        existing_user = User.query.filter_by(username=username).first()
        existing_phone = User.query.filter_by(phone=phone).first()
        if existing_user or existing_phone:
            flash('Користувач з таким логіном або номером телефону вже існує.', 'error')
            return render_template('register.html')
```

У цій частині коду здійснюється обробка POST-запиту форми реєстрації. Спочатку зчитуються дані з форми, після чого перевіряється, чи не існує користувач з таким же ім’ям користувача або номером телефону.

Далі відбувається перевірка валідності номера телефону та інших полів форми. Якщо дані коректні, завантажується фото профілю або встановлюється значення за замовчуванням:

```
            if not (phone.startswith('+') and len(phone) == 13 and phone[1:].isdigit()):
                flash('Номер телефону повинен починатися з +, містити тільки цифри та мати довжину 12 символів без +.', 'error')
                return render_template('register.html')
            if 'profile_photo' in request.files and request.files['profile_photo'].filename != '':
                profile_photo = save_profile_image(request.files['profile_photo'])
            else:
                profile_photo = 'user.png'

            if len(username) < 8 or len(password) < 8 or len(realname) < 8:
                flash('Поля Логін, Пароль та ПІБ мають містити не менше 8 символів!', 'error')
            elif not username or not password or not realname or not phone:
                flash('Заповніть всі необхідні поля!', 'error')
            else:
```

Якщо всі перевірки пройдено успішно, створюється новий об’єкт користувача і додається до бази даних. Після успішної реєстрації система перенаправляє користувача на домашню сторінку відповідно до його ролі:

```
                role = 'admin' if username == 'admin_admin' and password == 'admin_admin' else 'user'
                new_user = User(
```

```
                username=username,
                real_name=realname,
                phone=phone,
                password=password,
                profile_photo=profile_photo,
                role=role,
                registration_date=datetime.now()
            )
            db.session.add(new_user)
            db.session.commit()
            flash('Реєстрація пройшла успішно!', 'success')
            session.clear()
            session['user_id'] = new_user.id
            return redirect(url_for(f'user.{new_user.role}_home'))
    return render_template('register.html')
```

Щоб візуально відобразити роботу цього коду можна використати діаграму послідовності. Ці діаграми показують послідовність взаємодій між різними компонентами системи під час виконання певних операцій, таких як реєстрація та вхід користувача. Вони відображають, як користувач взаємодіє з системою через інтерфейс користувача, які запити надсилаються до серверу та які відповіді отримуються (рис. 3.2).

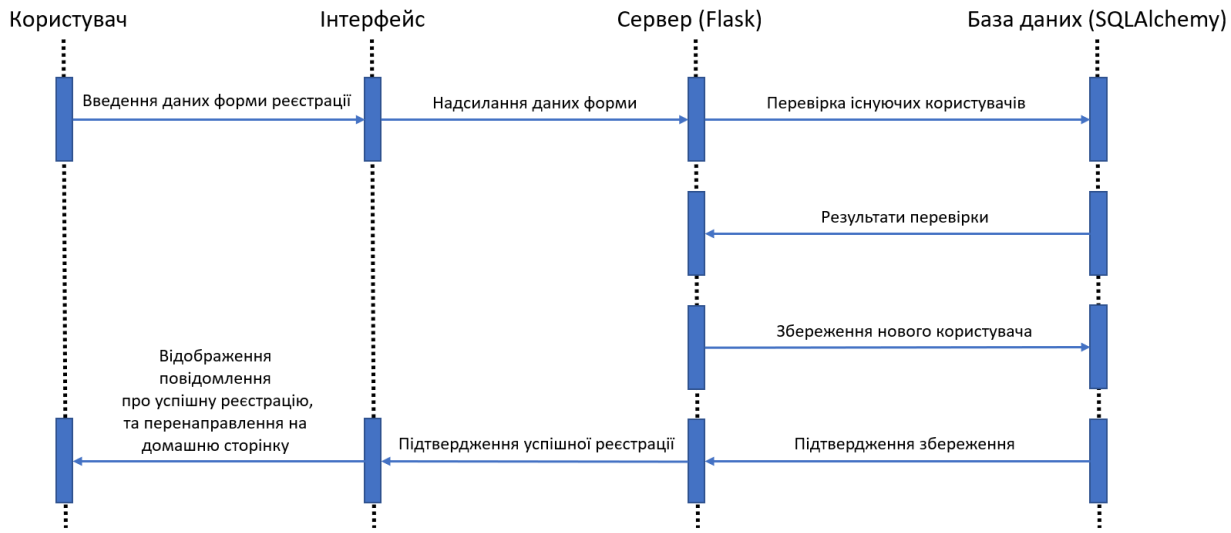

Рисунок 3.2 — Діаграма послідовності для процесу реєстрації

Дана діаграма послідовності добре відображає послідовність дій та взаємодію між різними компонентами системи під час процесу реєстрації.

Функція входу користувача дозволяє існуючим користувачам входити в систему, перевіряючи їхні облікові дані. У цій частині коду обробляється POST-запит форми

входу. Спочатку зчитуються введені дані, перевіряється їх валідність та здійснюється пошук користувача в базі даних. Якщо користувача знайдено, викликається функція user_login, яка здійснює вхід користувача в систему:

```
@main.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if len(username) < 8 or len(password) < 8:
            flash('Поля Логін та Пароль мають містити не менше 8 символів!',
'error')
            return render_template('login.html')
        user            =            User.query.filter_by(username=username,
password=password).first()
        if not user:
            flash('Помилка входу! Перевірте дані та спробуйте ще раз.',
'error')
            return render_template('login.html')
        user_login(user)
        flash('Вхід успішний!', 'success')
        return redirect(url_for(f'user.{user.role}_home'))
    return render_template('login.html')
```

Наступна функція зберігає ID користувача в сесії, що дозволяє ідентифікувати його у подальших запитах:

```
def user_login(user):
    db.session.commit()
    session['user_id'] = user.id
```

Для відображення домашніх сторінок користувачів та адміністраторів використовуються наступні маршрути:

```
@user.route('/user_home')
def user_home():
    if 'user_id' in session:
        user_id = session['user_id']
        user = User.query.get(user_id)
        if user:
            profile_photo = user.profile_photo
            return       render_template('user_home.html',       user=user,
profile_photo=profile_photo)
    return "Користувача не знайдено, будь-ласка увійдіть ще раз!", 404
```

Цей маршрут обробляє запит до домашньої сторінки користувача. Якщо користувач аутентифікований, його дані витягуються з бази даних і передаються у шаблон для відображення. Аналогічний маршрут для домашньої сторінки адміністратора, який виконує ті ж дії, але перенаправляє до іншого шаблону:

```
@user.route('/admin_home')
def admin_home():
    if 'user_id' in session:
        user_id = session['user_id']
        user = User.query.get(user_id)
        if user:
            profile_photo = user.profile_photo
            return render_template('admin_home.html', user=user,
profile_photo=profile_photo)
    return "Користувача не знайдено, будь-ласка увійдіть ще раз!", 404
```

Для візуалізації принципу роботи входу в систему також можна використати діаграму послідовності (рис. 3.3).

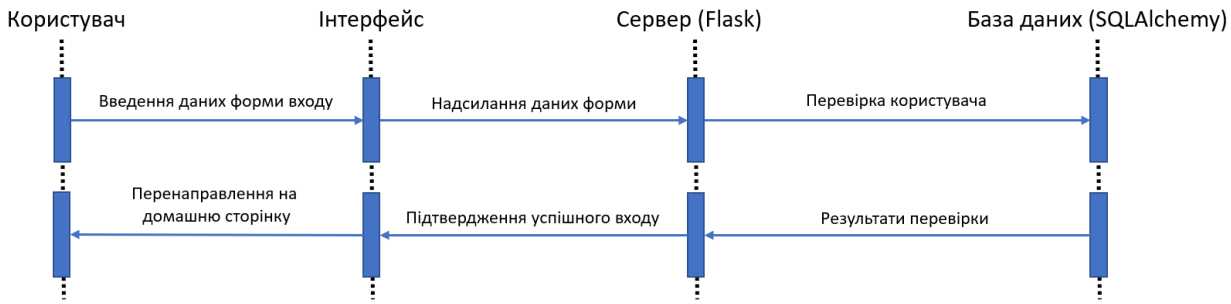

Рисунок 3.3 — Діаграма послідовності для процесу входу

Ці функції забезпечують основну логіку авторизації користувачів в системі «GasInfo», включаючи реєстрацію нових користувачів та вхід існуючих. Правильна реалізація цих функцій забезпечує безпеку та зручність використання системи для всіх користувачів.

Функція для додавання нового лічильника дозволяє користувачам додавати інформацію про новий лічильник до системи. Спочатку відбувається перевірка аутентифікації користувача. Якщо користувач не аутентифікований, повертається помилка. Дані з форми додаються у змінні. Перевіряється, чи існує лічильник з таким номером у базі даних. Якщо перевірка пройдена, створюється новий об’єкт лічильника і додається до бази даних:

```
@meter.route('/add-meter', methods=['POST'])
def add_meter():
    if 'user_id' not in session:
        return jsonify({'error': 'Not authenticated'}), 401
    user_id = session['user_id']
    meter_number = request.form['meter_number']
    meter_model = request.form['meter_model']
```

```
        tariff = float(request.form['tariff'])
        subsidy = 1.0 if request.form.get('subsidy') == 'on' else 0.0
        privilege = 1.0 if request.form.get('privilege') == 'on' else 0.0
        address = request.form['address']
```

Потім перевіряється, чи існує лічильник з таким номером у базі даних:

```
        if Meter.query.filter_by(meter_number=meter_number).first():
            return jsonify({'error': 'Meter with that number already exists'}),
400
        new_meter = Meter(
            user_id=user_id,
            meter_number=meter_number,
            meter_model=meter_model,
            tariff=tariff,
            subsidy=subsidy,
            privilege=privilege,
            address=address,
            status=False
        )
        db.session.add(new_meter)
        db.session.commit()

        return jsonify({'success': 'Лічильник додано успішно'})
```

Функція для отримання всіх лічильників, зареєстрованих користувачем. Вона використовується для відображення лічильників на інтерфейсі користувача. Якщо користувач не аутентифікований, повертається помилка. З бази даних витягується користувач за його ID. Створюється список лічильників користувача і повертається у вигляді JSON.

```
@meter.route('/get-meters')
def get_meters():
    if 'user_id' not in session:
        return jsonify({'error': 'Not authenticated'}), 401
    user_id = session['user_id']
    user = User.query.get(user_id)
    if not user:
        return jsonify({'error': 'User not found'}), 404
    meters = [{
        'meter_number': meter.meter_number,
        'meter_model': meter.meter_model,
        'tariff': meter.tariff,
        'subsidy': meter.subsidy,
        'privilege': meter.privilege,
        'address': meter.address,
        'status': meter.status
    } for meter in user.meters]
    return jsonify(meters)
```

Функція для видалення лічильника дозволяє користувачам видаляти свої лічильники разом з усіма пов’язаними квитанціями. Витягуються дані користувача

та лічильника з бази даних. Видаляються всі квитанції, пов’язані з лічильником, та сам лічильник:

```
@meter.route('/delete-meter/<string:meter_number>', methods=['DELETE'])
def delete_meter(meter_number):
    if 'user_id' not in session:
        return jsonify({'error': 'Not authenticated'}), 401
    user_id = session['user_id']
    user = User.query.get(user_id)
    if not user:
        return jsonify({'error': 'User not found'}), 404
    meter = Meter.query.filter_by(user_id=user_id,
meter_number=meter_number).first()
    if not meter:
        return jsonify({'error': 'Meter not found or does not belong to you'}),
404
    try:
        receipts = Receipt.query.filter_by(meter_id=meter.id).all()
        for receipt in receipts:
            db.session.delete(receipt)
        db.session.delete(meter)
        db.session.commit()
        return jsonify({'success': 'Meter and associated receipts deleted
successfully'}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({'error': 'Failed to delete the meter and associated
receipts', 'message': str(e)}), 500
```

Для взаємодії з інтерфейсом використовуються функції JavaScript, які забезпечують відправлення запитів на сервер, обробку відповідей та оновлення інтерфейсу. JavaScript функція обробляє подію надіслання форми, відправляє дані на сервер і обробляє відповідь:

```
document.getElementById('add-meter-form').addEventListener('submit',
function(event) {
    event.preventDefault();
    const formData = new FormData(this);
    fetch('/add-meter', {
        method: 'POST',
        body: formData
    })
    .then(response => response.json())
    .then(data => {
        if (data.success) {
            Swal.fire('Успіх', 'Лічильник додано успішно!',
'success').then(() => {
                loadMeters();
                closeForm();
            });
        } else {
            Swal.fire('Помилка', data.error, 'error');
        }
    })
    .catch(error => {
        console.error('Error:', error);
```

```
        });
    });
```

Завантаження лічильників, JavaScript функція відправляє запит на сервер для отримання списку лічильників та оновлює таблицю на сторінці:

```
    function loadMeters() {
        fetch('/get-meters')
        .then(response => response.json())
        .then(meters => {
            const tableBody = document.getElementById('meters-table').querySelector('tbody');
            tableBody.innerHTML = '';
            meters.forEach(meter => {
                const row = document.createElement('tr');
                row.innerHTML = `
                    <td>${meter.meter_number}</td>
                    <td>${meter.meter_model}</td>
                    <td>${meter.tariff.toFixed(2)}</td>
                    <td>${meter.subsidy === 1.0 ? 'Так' : 'Ні'}</td>
                    <td>${meter.privilege === 1.0 ? 'Так' : 'Ні'}</td>
                    <td>${meter.address}</td>
                    <td>${meter.status ? 'Активний' : 'Неактивний'}</td>
                    <td><button onclick="confirmDeletion('${meter.meter_number}')" class="delete-meter-button">Видалити</button></td>
                `;
                tableBody.appendChild(row);
            });
        })
        .catch(error => console.error('Error loading meters:', error));
    }
```

Підтвердження та видалення лічильника, JavaScript функції для підтвердження видалення лічильника та відправлення запиту на сервер для видалення:

```
    function confirmDeletion(meterNumber) {
        Swal.fire({
            title: 'Ви впевнені?',
            text: `Ви дійсно бажаєте видалити лічильник з номером ${meterNumber}?`,
            icon: 'warning',
            showCancelButton: true,
            confirmButtonColor: '#ab0a0a',
            cancelButtonColor: '#007bff',
            confirmButtonText: 'Так, видалити!',
            cancelButtonText: 'Ні, скасувати'
        }).then((result) => {
            if (result.isConfirmed) {
                deleteMeter(meterNumber);
            }
        });
    }
    function deleteMeter(meterNumber) {
        fetch(`/delete-meter/${meterNumber}`, { method: 'DELETE' })
        .then(response => response.json())
```

```
        .then(data => {
            if (data.success) {
                Swal.fire('Видалено!', 'Лічильник успішно видалено.', 'success');
                loadMeters();
            } else {
                Swal.fire('Помилка', 'Не вдалося видалити лічильник.', 'error');
            }
        })
        .catch(error => console.error('Error:', error));}
```

Ці функції забезпечують основну логіку для роботи з лічильниками в системі «GasInfo», включаючи додавання, отримання та видалення лічильників. Аналогічні функції використовуються і для взаємодії з квитанціями. Правильна реалізація цих функцій забезпечує зручність та ефективність використання системи для всіх користувачів.

Наступним варто розглянути реалізацію системи для аналізу даних користувачів. Тут основну роль виконує функція get_analytics_data відповідає за збирання та обробку аналітичних даних користувача, таких як споживання газу по місяцях, динаміка цін на газ, статуси оплат квитанцій та статистика споживання газу по лічильниках. Вона повертає ці дані у форматі JSON, щоб їх можна було використовувати на стороні клієнта для візуалізації.

Спочатку йде перевірка аутентифікації користувача, Якщо користувач не аутентифікований, повертається помилка:

```
@main.route('/analytics/data')
def get_analytics_data():
    if 'user_id' not in session:
        return jsonify({'error': 'Not authenticated'}), 401
```

Зчитування даних користувача, витягуються дані користувача з бази даних за його ID. Якщо користувача не знайдено, повертається помилка:

```
   user_id = session['user_id']
    user = User.query.get(user_id)
    if not user:
        return jsonify({'error': 'User not found'}), 404
```

Збір квитанцій користувача, отримуються всі квитанції для даного користувача:

```
        receipts = Receipt.query.join(Meter).filter(Meter.user_id == user_id).all()
```

Аналіз даних, розраховується споживання газу по місяцях, динаміка зміни ціни газу, статуси оплати квитанцій та статистика споживання газу по лічильниках:

```
        monthly_gas_usage = defaultdict(float)
        for receipt in receipts:
            month = receipt.date.strftime('%m')
            monthly_gas_usage[month] += receipt.amount_value
        price_trends = [{'date': receipt.date.strftime('%Y-%m-%d'), 'tariff': receipt.tariff} for receipt in receipts]
        payment_status_count = defaultdict(int)
        for receipt in receipts:
            payment_status_count[str(receipt.status)] += 1
        meter_usage_stats = defaultdict(float)
        for receipt in receipts:
            meter_usage_stats[receipt.meter.meter_number] += receipt.amount_value
```

Формування та повернення даних у форматі JSON: Дані формуються у відповідний формат і повертаються користувачу:

```
        data = {
            'monthly_gas_usage': [{'month': month, 'total_gas': total_gas} for month, total_gas in monthly_gas_usage.items()],
            'price_trends': price_trends,
            'payment_status_count': [{'status': status, 'count': count} for status, count in payment_status_count.items()],
            'meter_usage_stats': [{'meter_number': meter_number, 'total_gas': total_gas} for meter_number, total_gas in meter_usage_stats.items()]
        }
        return jsonify(data)
```

Функції JavaScript забезпечують завантаження аналітичних даних з сервера та їх візуалізацію за допомогою бібліотеки Chart.js. Завантаження аналітичних даних, тут основна функція loadAnalyticsData відправляє запит на сервер для отримання аналітичних даних та обробляє отриману відповідь:

```
    function loadAnalyticsData() {
        fetch('/analytics/data')
        .then(response => response.json())
        .then(data => {
            const monthlyGasLabels = data.monthly_gas_usage.map(item => {
                if (item.meter_number !== undefined) {
                    return `${item.month} (${item.meter_number})`;
                } else {
                    return item.month;
                }
            });
            createChart('monthlyGasUsageChart', 'Графік споживання газу за місяць', 'line', data.monthly_gas_usage, 'total_gas', 'month', 'Споживання газу (куб. м)', monthlyGasLabels);
            createChart('priceTrendsChart', 'Динаміка цін на газ', 'line', data.price_trends, 'tariff', 'date', 'Тариф за куб. м');
            const meterStatusLabels = data.meter_usage_stats.map(item => `Лічильник ${item.meter_number}`);
```

```
            createChart('meterStatusChart',    'Статус    лічильників',    'pie',
data.meter_usage_stats, 'total_gas', 'meter_number', '', meterStatusLabels);
        })
        .catch(error => {
            console.error('Error loading analytics data:', error);});}
```

Створення графіків, для цього створена функція createChart яка створює графік за допомогою Chart.js, використовуючи передані дані:

```
    function createChart(chartId, title, type, data, valueField, labelField,
yAxisLabel, labels) {
        const ctx = document.getElementById(chartId).getContext('2d');
        const values = data.map(item => item[valueField]);
        const chart = new Chart(ctx, {
            type: type,
            data: {
                labels: labels || data.map(item => item[labelField]),
                datasets: [{
                    label: yAxisLabel,
                    data: values,
                    backgroundColor: type === 'pie' ? ['#FF6384', '#36A2EB',
'#FFCE56'] : 'rgba(54, 162, 235, 0.2)',
                    borderColor: type === 'pie' ? ['#FF6384', '#36A2EB',
'#FFCE56'] : 'rgba(54, 162, 235, 1)',
                    borderWidth: 1
                }]
            },
            options: {
                responsive: true,
                maintainAspectRatio: false,
                scales: type !== 'pie' ? {
                    y: {
                        beginAtZero: true,
                        title: {
                            display: !!yAxisLabel,
                            text: yAxisLabel}}} : {}});}
```

Ці функції забезпечують завантаження аналітичних даних з сервера, обробку даних та створення графіків для візуалізації використання газу, динаміки цін та статистики по лічильниках.

3.4. Тестування та демонстрація отриманих результатів

Процес тестування та демонстрації роботи системи «GasInfo» розпочинається з головної сторінки, яку користувач бачить першим при відвідуванні інформаційної системи (рис. 3.4).

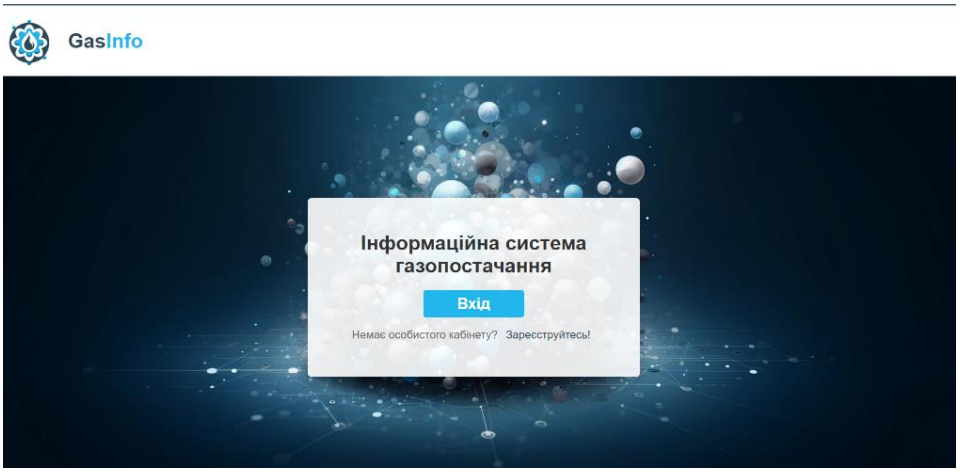

Рисунок 3.4 — Головна сторінка системи «GasInfo»

На головній сторінці відображається інформаційна панель системи газопостачання з кнопкою «Вхід». Також є можливість зареєструвати новий обліковий запис, якщо користувач ще не має особистого кабінету, натиснувши на посилання «Зареєструйтесь!». Якщо користувач натисне кнопку «Вхід» на головній сторінці, йому буде представлена форма для введення логіну та паролю (рис. 3.5).

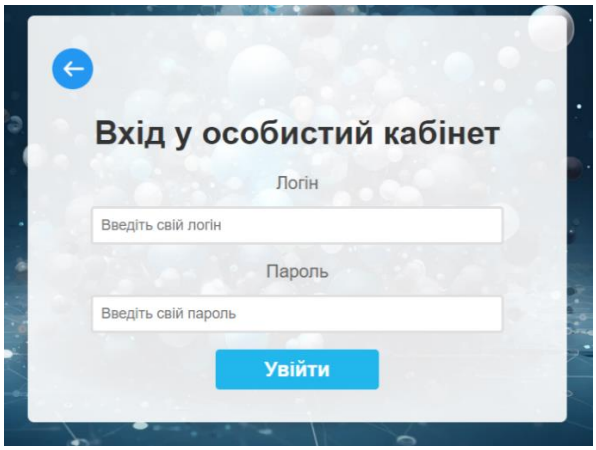

Рисунок 3.5 — Форма входу

Форма входу дозволяє користувачам ввести свої облікові дані для доступу до особистого кабінету. Після введення логіну та паролю користувач натискає кнопку «Увійти» для автентифікації.

Якщо користувач натисне на посилання «Зареєструйтеся!» на головній сторінці, відкриється форма реєстрації нового облікового запису (рис. 3.6).

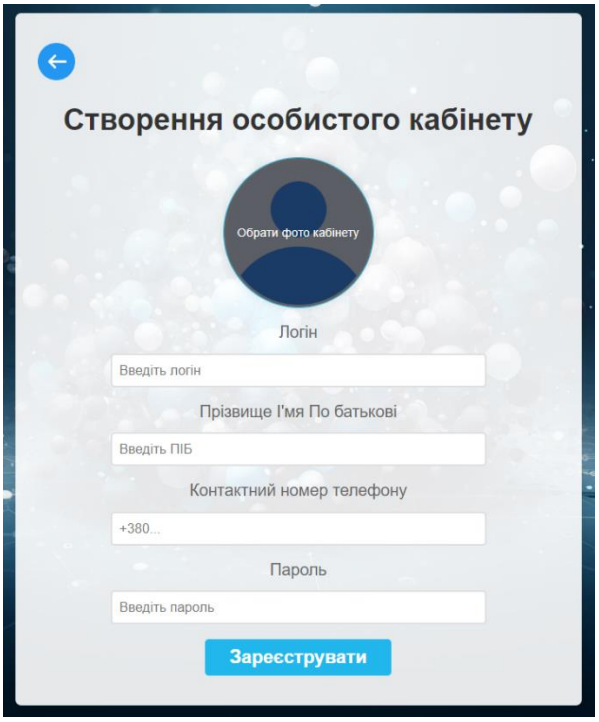

Рисунок 3.6 — Форма реєстрації

У формі реєстрації користувач може ввести свої дані, такі як логін, повне ім’я, номер телефону та пароль. Крім того, якщо користувач наведе курсор на круглу іконку профілю, вона затемниться і з’явиться текст, що пропонує обрати фото для особистого кабінету.

Після успішного введення даних для входу, користувача буде перенаправлено на головну сторінку системи «GasInfo». На цій сторінці користувач може бачити навігаційне меню з декількома елементами, що дозволяють переходити між різними розділами системи (рис. 3.7).

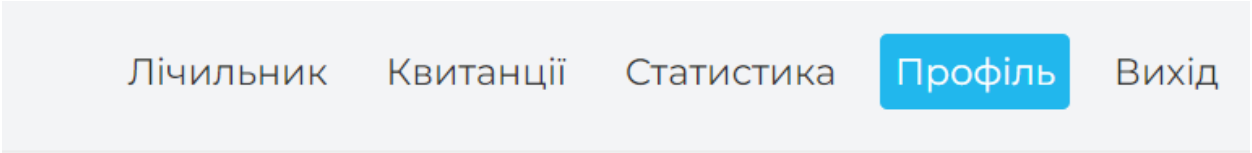

Рисунок 3.7 — Навігаційне меню системи

Навігаційне меню включає наступні елементи:

- Лічильник;
- Квитанції;
- Статистика;
- Профіль;
- Вихід

Користувач може натиснути на будь-який з цих елементів для переходу до відповідного розділу системи, елементи якщо на них навести змінюють свій колір що візуально підкреслює дії користувача.

Якщо користувач натисне на елемент меню «Профіль», то він перейде на сторінку профілю (рис. 3.8).

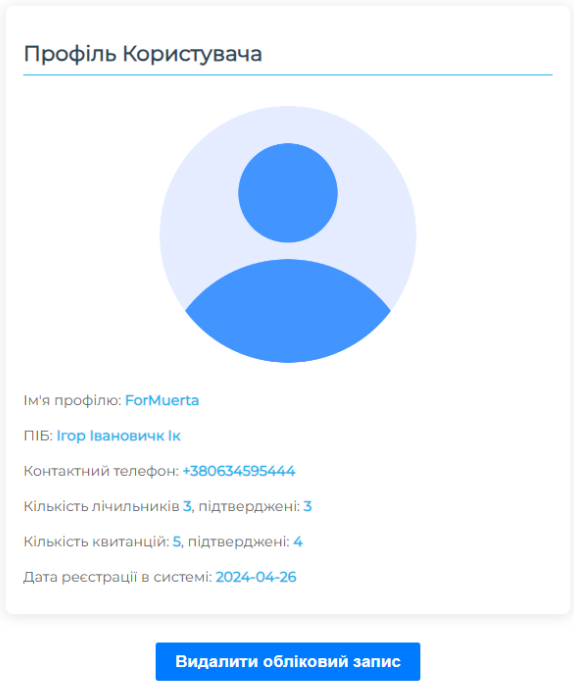

Рисунок 3.8 — Інформація про користувача

Тут відображається основна інформація про користувача:

- Назва профілю;
- ПІБ;
- Контактний телефон;
- Кількість лічильників та квитанцій, а також статус їх підтвердження;

- Дата реєстрації в системі.

Крім того, на цій сторінці є кнопка для видалення облікового запису. Якщо користувач натисне на кнопку «Видалити обліковий запис», відкриється форма підтвердження видалення (3.9).

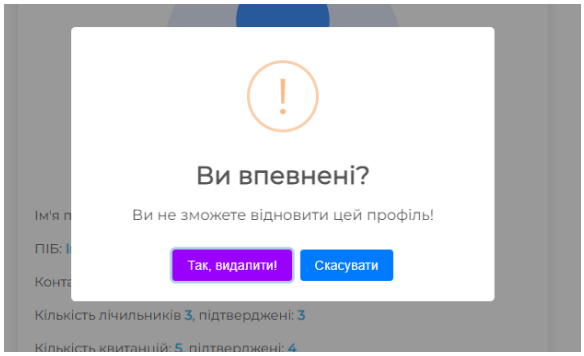

Рисунок 3.9 — Форма підтвердження видалення профілю

Форма підтвердження містить повідомлення про те, що користувач не зможе відновити свій профіль після видалення. Також є дві кнопки: «Так, видалити!» та «Скасувати», що дозволяють підтвердити або відмінити операцію видалення облікового запису.

Якщо користувач перейде до розділу «Лічильники» у навігаційному меню, він побачить таблицю з існуючими лічильниками, що містить такі дані (рис. 3.10):

- Номер лічильника;
- Модель лічильника;
- Тариф;
- Наявність субсидії;
- Наявність пільги;
- Адреса;
- Статус;
- Кнопка для видалення лічильника.

Ця таблиця дозволяє користувачам переглядати та керувати своїми лічильниками.

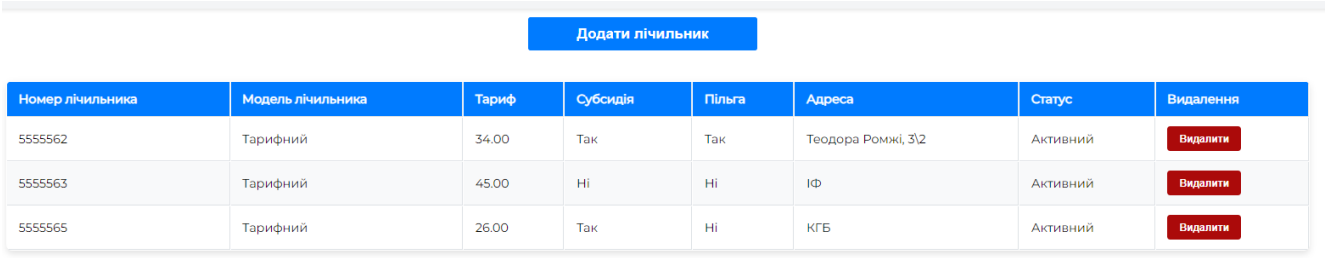

Рисунок 3.10 — Таблиця лічильників

Якщо користувач натисне на кнопку «Додати лічильник», відкриється форма для введення даних нового лічильника (рис. 3.11).

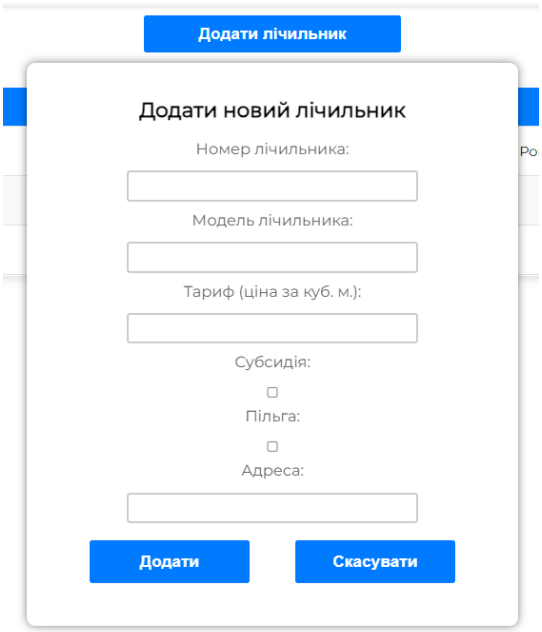

Рисунок 3.11 — Форма додавання лічильника

Форма включає наступні поля:

- Номер лічильника;
- Модель лічильника;
- Тариф (ціна за куб. м);
- Опції для вибору субсидії та пільги;
- Адреса.

Після заповнення форми користувач може натиснути кнопку «Додати» для збереження нового лічильника, або «Скасувати» для відміни дії. Якщо користувач

натисне на кнопку «Видалити» у відповідному записі в таблиці лічильників, відкриється форма підтвердження видалення (рис. 3.12).

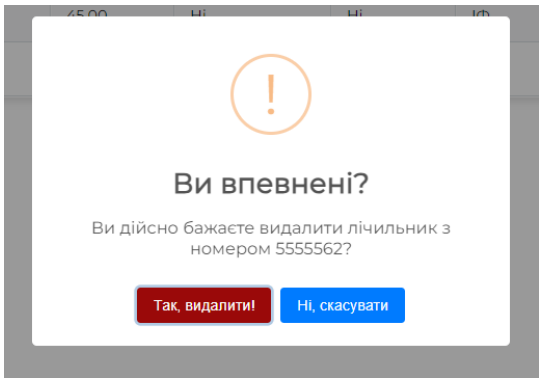

Рисунок 3.12 — Форма підтвердження видалення лічильника

Форма містить запитання «Ви впевнені?» з текстом, що уточнює номер лічильника, який користувач хоче видалити. Є дві кнопки: «Так, видалити!» для підтвердження видалення і «Ні, скасувати» для відміни дії.

При переході до розділу «Квитанції» користувач бачить таблицю з квитанціями (3.13).

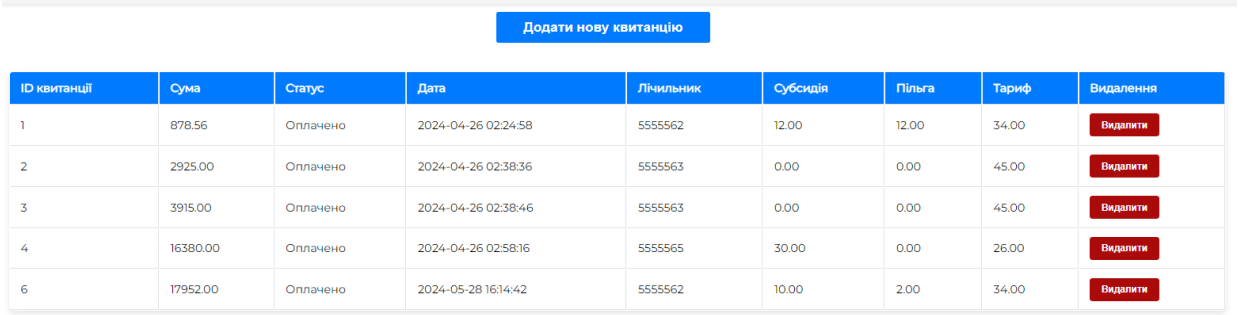

Рисунок 3.13 — Таблиця квитанцій

У таблиці відображаються такі дані:

- ID квитанції;
- сума;
- статус;
- дата;

- номер лічильника;
- субсидія;
- пільга;
- тариф;
- кнопка для видалення.

Якщо користувач натисне на кнопку «Додати квитанцію», відкриється форма для введення даних нової квитанції (рис. 3.14).

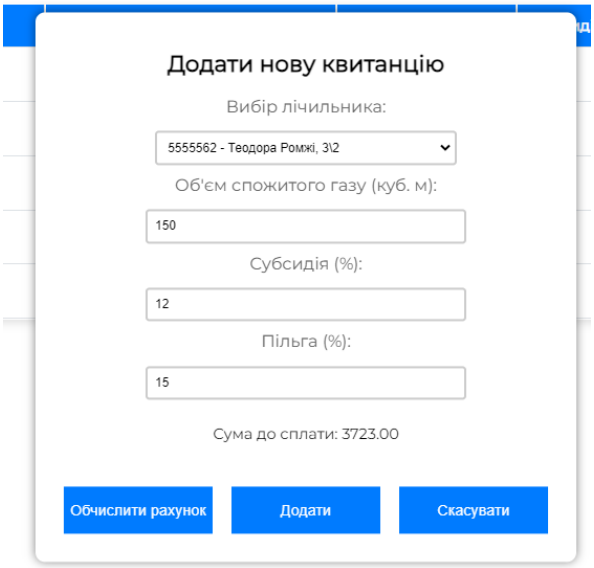

Рисунок 3.14 — Форма добавлення квитанції

Користувач обирає лічильник, вводить об’єм спожитого газу, відсоток субсидії та пільги. Якщо для лічильника не були вказані субсидія або пільги, відповідні поля будуть неактивними. Коли користувач введе всі дані він може обчислити суму до оплати натиснувши на кнопку «Обчислити рахунок» Видалення квитанцій працює так само, як і для лічильників, через форму підтвердження.

При переході до розділу «Статистика» користувач бачить два графіки та одну діаграму. Перший графік показує споживання газу по місяцях, другий графік відображає динаміку тарифів на газ (рис. 3.15).

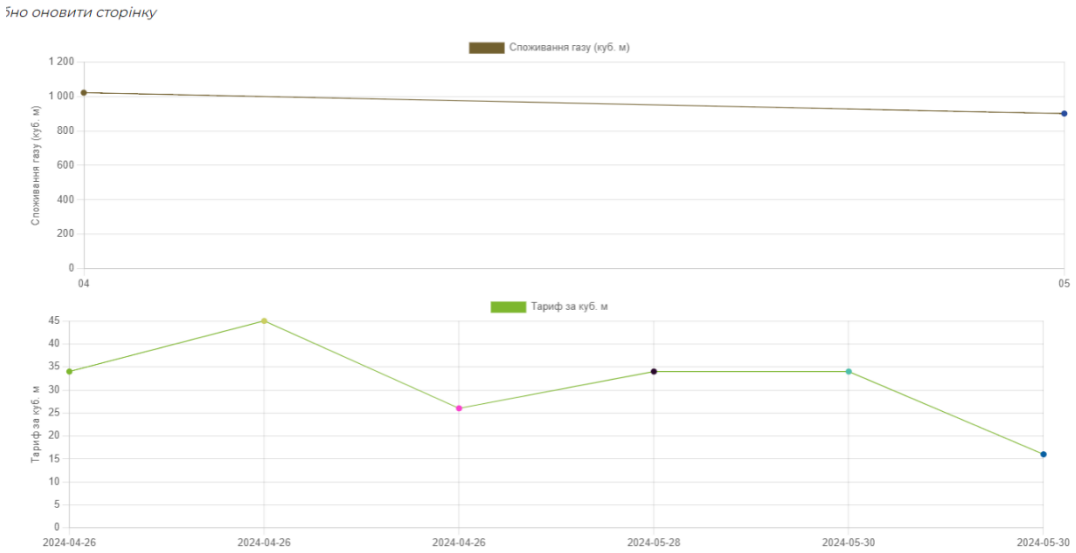

Рисунок 3.15 — Графіки візуалізації даних

Кругова діаграма демонструє використання газу за різними лічильниками, причому кольори обираються випадковим чином, а при наведенні на елемент діаграми відображається точне значення споживаного газу (рис. 3.16).

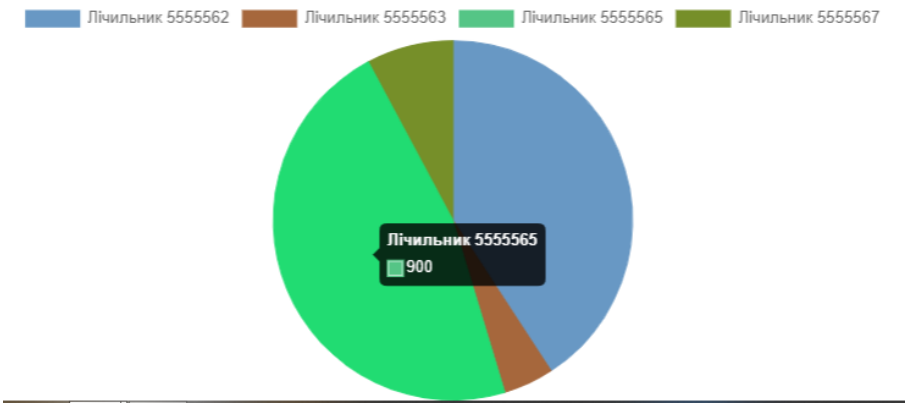

Рисунок 3.16 — Кругова діаграма візуалізації даних

При вході в систему під роллю адміністратора користувач побачить таблицю з усіма зареєстрованими користувачами (рис. 3.17).

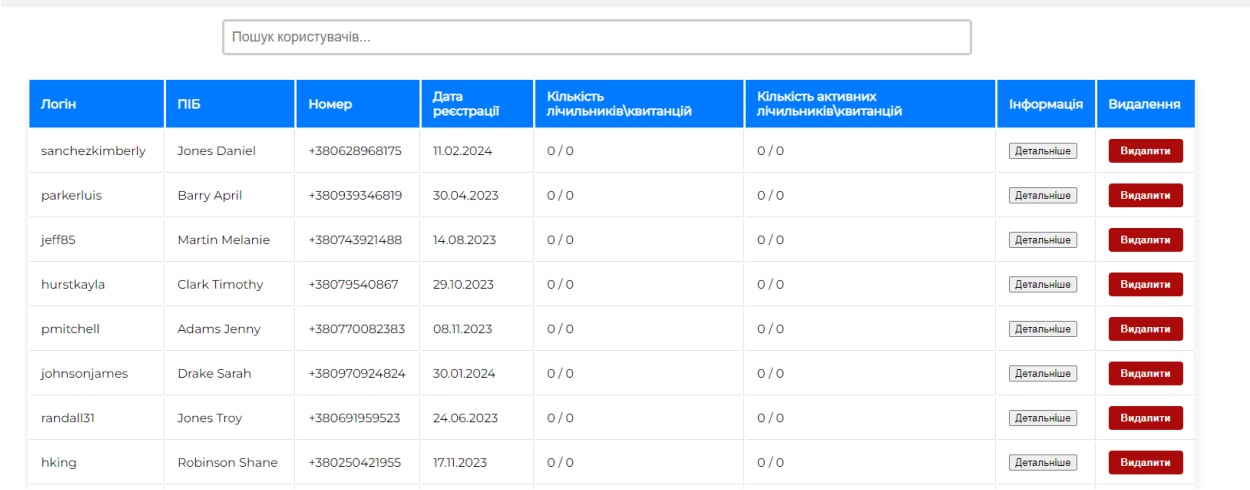

Рисунок 3.17 — Таблиця користувачів

Таблиця містить стовпці:

- логіну;
- ПІБ;
- номер телефону;
- дату реєстрації;
- кількістю лічильників та квитанцій;
- кількість активних лічильників і квитанцій;
- кнопка детальніше;
- кнопка видалення.

При натисканні на кнопку «Детальніше» відкривається форма з детальною інформацією про користувача, його лічильники та квитанції. Адміністратор може бачити всі дані користувача, включаючи статус лічильників та квитанцій. У формі також є кнопка для підтвердження даних та її закриття (3.18). Кнопка «Видалити» дозволяє адміністратору видалити користувача з системи, через форму підтвердження.

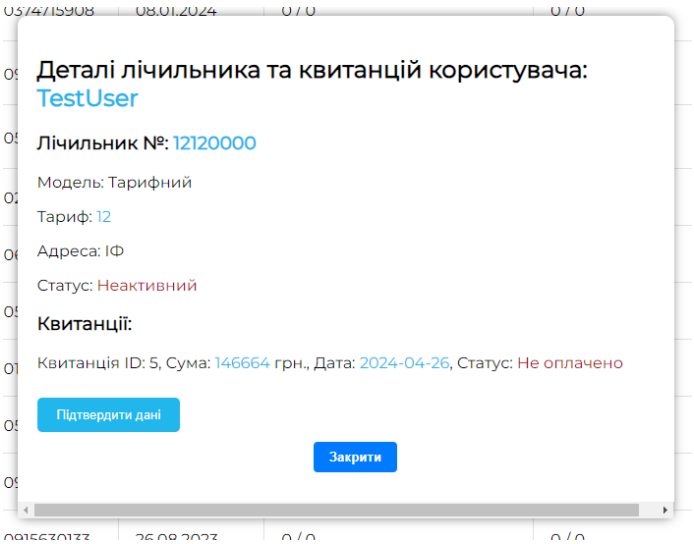

Рисунок 3.18 — Форма детальної інформації користувача

Пошук користувачів здійснюється за допомогою поля введення у верхній частині таблиці, що дозволяє швидко знайти потрібного користувача за будь-яким критерієм.

Після проведення ретельного тестування всі елементи та функції системи «GasInfo» працюють коректно. Реєстрація, вхід, управління лічильниками та квитанціями, перегляд статистики, а також панель адміністратора функціонують без помилок. Таким чином, тестування можна вважати успішним.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Природні загрози та характер проявів і дій на об’єкти економіки

До природних загроз відносяться стихійні явища, які являють безпосередню загрозу для життя та здоров’я людей.

Природні небезпеки підпорядковуються деяким загальним закономірностям:

– для кожного виду небезпек характерна певна просторова приуроченість;

– встановлено, що чим більша інтенсивність (потужність) небезпечного явища, тим рідше воно трапляється;

– кожному виду небезпек передують певні специфічні ознаки (передвісники);

– за всієї непередбачуваності тієї чи іншої природної небезпеки, її прояв може бути передбачений;

– у багатьох випадках можуть бути передбачені пасивні та активні захисні заходи від природних небезпек [14].

Природні загрози [15,16,17]:

1) Гідросферні:

– повені паводки: великі кількості опадів або танення снігу можуть призвести до затоплення територій та промислових зон, що може призвести до затримок в постачанні товарів та послуг, збитків внаслідок затоплення приміщень та обладнання. Повені – це найпоширеніша природна небезпека;

– цунамі – це гравітаційні хвилі дуже великої довжини, які виникають у результаті зсуву вверх або вниз великих ділянок дна під час сильних підводних землетрусів, рідше вулканічних виверженнь.

2) Літосферні небезпеки:

– землетруси можуть спричинити руйнування будівель та інфраструктури, перерви у виробництві та доставці товарів, а також можуть призвести до загибелі або травмування працівників;

– селі – короткочасні бурхливі паводки на гірських річках, що мають характер грязекам’яних потоків. Причинами селів можуть бути землетруси, сильні снігопади, дощі, інтенсивне танення снігу. Основна небезпека – велика кінематична енергія грязьових та водяних потоків, швидкість руху яких може досягати 15 км/год.

– снігова лавина – це сніговий обвал, маса снігу, що падає чи сповзає із гірських схилів під впливом якої-небудь дії і захоплює на своєму шляху нові маси снігу. Однією із спонукальних причин лавини може бути землетрус. Снігові лавини поширені у гірських районах.

3) Атмосферні:

– урагани та торнадо: ці стихійні явища можуть спричинити руйнування будівель та об'єктів, перерви у виробництві та транспорті, а також можуть призвести до великих збитків внаслідок руйнування майна.

– інші стихійні лиха, такі як суховії, засухи.

4) Пожежі. Можуть пошкодити будівлі, обладнання та товари, що може призвести до збитків у виробництві та торгівлі, а також можуть мати негативний вплив на довкілля та здоров'я працівників.

4.2 Аналіз умов праці за показниками шкідливості та небезпечності чинників виробничого середовища, важкості та напруженості трудового процесу

Робота оператора ПК (бухгалтер газової служби) передбачає виконання посадових обов’язків у приміщенні операторської, обладнаному персональними комп’ютерами (далі ПК) з візуальними дисплейними терміналами (далі ВДТ), шафами управління (серверні) та іншим електрообладнанням.

На основі аналізу роботи існуючого обладнання і посадових обов’язків фахівця у приміщенні обладнаному ПК з ВДТ, виявлені небезпечні та шкідливі

виробничі фактори, здатні привести до травм або ушкодження здоров'я працівників.

Основні небезпечні та шкідливі виробничі фактори:

– можливість ураження електричним струмом;

– механічне травмування внаслідок нераціонального розташування робочих місць, що є порушенням вимог ергономіки;

– підвищене нервово-психічне навантаження, швидка втома;

– незадовільні параметри мікроклімату в робочих приміщеннях;

– неправильні або нераціональні дії персоналу в умовах надзвичайних ситуацій;

– можливість загоряння, в зв'язку з порушенням правил протипожежної безпеки;

– монітор ПК є джерелом практично всіх видів електромагнітного випромінювання;

– від постійної роботи за клавіатурою ПК розвиток хронічного «тунельного синдрому».

Таблиця 4.1 – Вхідні умови для оцінки умов праці

Параметр	Фактичне значення
1. Санітарно-гігієнічні показники	
Мікроклімат	24 °С
Відносна вологість	45 %
Швидкість руху повітря	0,1 м/с
Освітлення	300 лк
Шум (дБ)	52,7дБ
Теплове випромінювання	25 Вт/м^2
Рентгенівське випромінювання	7 мР/год
2. Показники важкості праці	
Стереотипні робочі рухи (кількість за зміну): - за локального навантаження - за загального навантаження	 мінімальні мінімальні
Робоча поза	статична
Нахили корпуса	-
Переміщення у просторі, км: по горизонталі; по вертикалі	 мінімальні мінімальні

Продовження таблиці 4.1

3. Показники напруженості праці	
Інтелектуальне навантаження	невисока
Розподіл функцій за ступенем складності завдання	є
Характер виконуваної роботи	монотанна
Навантаження на зоровий аналізатор	висока
Спостереження за екраном відеотерміналу (годин за зміну)	6-7
Монотонність навантажень	висока
Режим праці (трив. роб. дня)	6
Наявність перерв та їх тривалість	10 хв. кожної роб. години

За хімічним фактором – 1 клас (підклас) умов праці (оптимальні умови).

Категорія робіт за енергозатратами (від 172 до 232 Вт) – середня, категорія IIа.

За температурними умовами праці (при 24 ºС) – 1 клас (підклас) умов праці (оптимальні умови).

За відносною вологістю (при 60%) – 1 клас (підклас) умов праці (оптимальні умови).

За рівнем шуму (при 52,7 дБ) – 1 клас (підклас) умов праці (оптимальні умови).

Вібрація загальна, еквівалентний скоригований рівень віброускорення, вісь Z (при 50 дБ) – 1 клас (підклас) умов праці (оптимальні умови).

За освітленістю і підрозрядом зорової роботи (штучне освітлення) (300 лк) – 1 клас (підклас) умов праці (оптимальні умови).

Електричні поля промислової частоти 50 Гц – 2 клас (підклас) умов праці (допустимі умови).

Робота оператора ПК за енерговитратами відноситься до категорії легких робіт Іа, Іб.

Під час оцінки умов праці на робочому місці оператора ПК значущими є такі показники:

– стереотипні робочі рухи (кількість за зміну): у разі локального навантаження (за участю м'язів кистей та пальців рук) – належить до класу 1 (до 20000). У випадку загального навантаження (при роботі з переважною участю м'язів рук та плечового поясу) – належить до класу 1 (до 10000);

– робоча поза – належить до класу 2 (періодичне перебування в незручній позі (робота з поворотом) тулуба, незручним розташуванням кінцівок) та/або фіксованій позі (неможливість зміни взаєморозташування різних частин тіла відносно одна одної) до 25 % часу зміни.);

– нахили корпуса (вимушені, більше 30), кількість за зміну - належить до класу 1 (до 50);

– переміщення у просторі (переходи, обумовлені технологічним процесом протягом зміни), км: по горизонталі - належить до класу 1 (до 4); по вертикалі – до класу 1 (до 2).

Отже за даними атестації робоче місце за показниками важкості трудового процесу відноситься до класу 2 – допустимий, середнє фізичне навантаження.

Характер виконуваної роботи належить до класу 2 (робота за встановленим графіком з можливим його коректуванням у ході діяльності).

Навантаження на зоровий аналізатор (при відстані від очей працюючого до об'єкта розрізнення не більше 0,5 м), за тривалості зосередженого спостереження (% часу зміни)) – належить до класу 2 (5,0...1,1 мм більше 50 % часу; 1,0...0,3 мм до 50 % часу; менше 0,3 мм до 25 %).

Спостереження за екранами відеотерміналів (годин на зміну) – належить до класу 2 (більше 2 годин).

Монотонність праці. Кількість елементів (прийомів, необхідних для реалізації простого завдання або в операціях, які повторюються багаторазово) – належить до класу 2 (5...3 прийоми).

Режим праці (фактична тривалість робочого дня (год.)) – належить до класу 1 (6 годин) (оптимальні умови).

Наявність регламентованих перерв та їх тривалість – належить до класу 1 (тривалістю 10 хвилин після кожною години роботи за ВДТ) (оптимальні умови).

Отже за даними атестації робоче місце за показниками напруженості трудового процесу відноситься легкого ступеня.

Результати аналізу умов праці на робочому місці наведені в таблиці 6.1.

Таблиця 4.2 – Аналіз умов праці на робочому місці оператора ПК

Фактори виробничого середовища та трудового процесу	Класи умов праці						
	оптимальний	допустимий	шкідливий				небезпечний
	1	2	3.1	3.2	3.3	3.4	4
Хімічні	+						
Біологічні	+						
Фізичні:							
шум	+						
вібрація	+						
інфразвук	+						
ультразвук	+						
неіонізуючі випромінювання	+						
іонізуючі випромінювання	+						
мікроклімат	+						
атмосферний тиск	+						
освітленість	+						
іонізація повітря	+						
важкість праці	+						
напруженість праці		+					
Загальна оцінка умов праці							
за ступенем шкідливості і небезпечності_			оптимальні				
за ступенем травмобезпечності			безпечні				

ВИСНОВКИ

У кваліфікаційній роботі проведено дослідження предметної області онлайн систем газопостачання. У першому розділі було розглянуто принципи взаємодії користувачів в таких системах, проведено огляд та аналіз існуючих аналогів, що дозволило сформувати задачі для розробки власної системи.

У другому розділі виконано проектування системи, зокрема визначено вимоги до неї, розроблено архітектуру та ключові компоненти, створено базу даних і визначено залежності між її елементами.

У третьому розділі здійснено вибір інструментів реалізації, описано структуру проекту та ключові елементи системи. Проведено програмну реалізацію основних функцій, таких як реєстрація та вхід користувачів, управління лічильниками та квитанціями, перегляд статистики. Завершено тестування системи та продемонстровано отримані результати.

В результаті виконання роботи було створено інформаційну систему «GasInfo», яка забезпечує ефективне управління даними користувачів, лічильниками та квитанціями, а також надає зручний інтерфейс для взаємодії користувачів із системою.

У подальшому система може збільшувати свій функціонал, оптимізувати роботу та ставати ефективнішою і зручнішою для користувачів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Українці можуть обирати постачальника газу онлайн – завдяки ПриватБанку [Електронний ресурс] // gre4kainfo. – 2020. – Режим доступу до ресурсу: https://gre4ka.info/zhyttia/61436-ukraintsi-mozhut-obyraty-postachalnyka-hazu-onlain-zavdiaky-pryvatbanku/.

2. Особистий кабінет – найкращий онлайн-помічник при оплаті за розподіл газу та передачі показників [Електронний ресурс] // kopachivska-gromada. – 2024. – Режим доступу до ресурсу: https://kopachivska-gromada.gov.ua/news/1711708624/.

3. В онлайн-сервісах 104.ua зареєструвалися 277 тисяч користувачів Чернігівщини [Електронний ресурс] // interfax. – 2022. – Режим доступу до ресурсу: https://interfax.com.ua/news/press-release/794816.html.

4. Компанія «104.ua» [Електронний ресурс] // «104.ua». – 2024. – Режим доступу до ресурсу: https://about.104.ua/ua/.

5. Нафтогаз [Електронний ресурс] // Нафтогаз. – 2024. – Режим доступу до ресурсу: https://gas.ua/uk/home.

6. Flask [Електронний ресурс] // Flask. – 2024. – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/3.0.x/.

7. Visual Studio Code [Електронний ресурс] // Microsoft. – 2024. – Режим доступу до ресурсу: https://code.visualstudio.com/.

8. SQLAlchemy [Електронний ресурс] // SQLAlchemy. – 2024. – Режим доступу до ресурсу: https://www.sqlalchemy.org/.

9. Werkzeug [Електронний ресурс] // Werkzeug. – 2024. – Режим доступу до ресурсу: https://werkzeug.palletsprojects.com/en/3.0.x/.

10. os — operating system interfaces [Електронний ресурс] // Python. – 2024. – Режим доступу до ресурсу: https://docs.python.org/uk/3/library/os.html.

11. datetime — Basic date and time types¶ [Електронний ресурс] // Python. – 2024. – Режим доступу до ресурсу: https://docs.python.org/uk/3/library/datetime.html.

12. collections — Container datatypes [Електронний ресурс] // Python. – 2024. – Режим доступу до ресурсу: https://docs.python.org/uk/3/library/collections.html.

13. SweetAlert2 [Електронний ресурс] // SweetAlert2. – 2024. – Режим доступу до ресурсу: https://sweetalert2.github.io/.

14.Наказом МОЗ від 08.04.2014 № 248, зареєстрованим у Мін'юсті 06.05.2014 за № 472/25249, затверджено Державні санітарні норми та правила «Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу».

15.Стиценко Т.Є., Пронюк Г.В., Сердюк Н.М., Хондак І.І. «Безпека життєдіяльності»: навч. посібник / Т.Є Стиценко, Г.В. Пронюк, Н.М. Сердюк, І.І. Хондак. – Харків: ХНУРЕ, 2018. 336 с.

16.Безпека життєдіяльності та охорона праці. Навчальний посібник для здобувачів вищої освіти із галузі знань 08 «Право», спеціальність 081 «Право». Львів: Львівський державний університет внутрішніх справ, 2022. 219 с.

17.Безпека життєдіяльності : навч. посібник / уклад. : М. Г. Русанов, Н. В. Чернікова, І. М. Шепель ; Комунальний заклад «Харківська гуманітарнопедагогічна академія» Харківської обласної ради. – Харків : 2019. – 100 с.

18.ДСанПіН 3.3.2.007-98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин.

19.НПАОП 0.00-7.15-18 Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями.

20.ДБН В.2.5-28-2018 Природне і штучне освітлення» передбачене природне та штучне освітлення.

21.ДСанПіН 3.3.6.096-2002 Державні санітарні норми і правила при роботі з джерелами електромагнітних полів.

22.ДСТУ-Н Б А.3.2-1:2007 Система стандартів безпеки праці. Настанова щодо визначення небезпечних і шкідливих факторів та захисту від їх впливу при виробництві будівельних матеріалів і виробів та їх використанні в процесі зведення та експлуатації об`єктів будівництва.

23.ДСН 3.3.6.037-99 Санітарні норми виробничого шуму, ультразвуку та інфразвуку.

24.Наказ від 08.04.2014 № 248 Про затвердження Державних санітарних норм та правил Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу.

25. Глова Б., Мудрик І. Застосування глибокого навчання у нейромаркетингових дослідженнях впливу несвідомих реакцій на поведінку споживачів // Тези доповідей третьої міжнародної конференції з обробки потокових даних, DSMP 2020, 2020. – С. 337-340. – IEEE. – DOI: 10.1109/DSMP.2020.9204192.

26. Бойко І., Стоянов Ю., Цуприк Х., Приведа О., Кінах І. Моделювання тунельного струму в наноструктурах з урахуванням ефекту електрон-електронної взаємодії // Матеріали міжнародної науково-технічної конференції з комп'ютерних наук та інформаційних технологій, 2020. – Т. 1. – С. 411-414. – IEEE. – DOI: 10.1109/STC-CSIT.2020.9322038.

Додатки

ДОДАТОК А

Лістинг системи «GasInfo»

Клієнтська частина:

Сторінки:

- admin_home.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="icon" href="{{ url_for('static',
filename='uploads/system_icon/favicon.png') }}" type="image/x-icon">
    <title>Головна</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='user_home.css') }}">
    <link
href="https://fonts.googleapis.com/css?family=Montserrat&display=swap"
rel="stylesheet">
</head>
<body>
    <div class="header">
        <div class="logo-and-title">
            <img src="{{ url_for('static',
filename='uploads/system_icon/favicon.png') }}" alt="Logo" class="logo">
            <h1><span style="color: #2e4957;">Gas</span><span style="color:
#21b7ed;">Info</span></h1>
        </div>
        <nav class="header-nav">
            <a href="{{ url_for('main.logout') }}">Вихід</a>
        </nav>
    </div>
    <div class="search-container">
        <input type="text" id="searchInput" placeholder="Пошук
користувачів...">
    </div>
            <div id="receipts-table-container">
                <table id="receipts-table">
                    <thead>
                        <tr>
                            <th>Логін</th>
                            <th>ПІБ</th>
                            <th>Номер</th>
                            <th>Дата реєстрації</th>
                            <th>Кількість лічильників\квитанцій</th>
                            <th>Кількість активних лічильників\квитанцій</th>
                            <th>Інформація</th>
                            <th>Видалення</th>
                        </tr>
                    </thead>
                    <tbody id="users-table-body">
                        <!-- Table rows will be dynamically added here -->
                    </tbody>
                </table>
            </div>
            <div id="user-data-form-container" style="display: none;">
```

```
                    <form id="user-data-form">
                    </form>
                    <div class="button-container-form">
                        <button           type="button"    id="descr-user-button"
onclick="closeDataUsertForm()">Закрити</button>
                    </div>
                </div>
            </div>
        </div>
        <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
        <script src="{{ url_for('static', filename='js/admin.js') }}"></script>
        <script src="{{ url_for('static', filename='js/user.js') }}"></script>
        <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
        <script>
            var userId = "{{ user.id }}";
            </script>
    </body>
    </html>
```

- index.html:

```
    <!DOCTYPE html>
    <html>
    <head>
        <title>Вхід\Реєстрація</title>
        <link      rel="icon"      href="static/uploads/system_icon/favicon.png"
type="image/x-icon">
        <link rel="stylesheet" href="{{ url_for('static', filename='index.css')
}}">
    </head>
    <body>
        <div class="header">
            <img     src="static/uploads/system_icon/favicon.png"     alt="Logo"
class="logo">
            <h1><span   style="color:   #2e4957;">Gas</span><span   style="color:
#21b7ed;">Info</span></h1>
        </div>
        <div             class="background-image"             style="background:
url('static/uploads/system_icon/background.png')    no-repeat    center    center;
background-size: cover;">
            <div class="container">
                <h1>Інформаційна система газопостачання</h1>
                <a href="/login" class="btn-link">
                    <button class="btn-large">Вхід</button>
                </a>
                <p>Немає особистого кабінету? <a href="/register" class="btn-link
register-link">Зареєструйтесь!</a></p>
            </div>
        </div>
    </body>
    </html>
```

- login.html:

```
    <!DOCTYPE html>
    <html lang="uk">

    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Вхід</title>
        <link      rel="icon"      href="static/uploads/system_icon/favicon.png"
type="image/x-icon">
```

```
        <link   rel="stylesheet"   type="text/css"   href="{{   url_for('static',
filename='login.css') }}">
    </head>

    <body>
        <div class="header">
            <img     src="static/uploads/system_icon/favicon.png"     alt="Logo"
class="logo">
            <h1><span   style="color:   #2e4957;">Gas</span><span   style="color:
#21b7ed;">Info</span></h1>
        </div>
        <div             class="background-image"             style="background:
url('static/uploads/system_icon/background.png')    no-repeat    center    center;
background-size: cover;">
            <div class="container">
                    <nav class="navigation">
                        <a href="{{ url_for('main.index') }}">
                            <img            src="{{            url_for('static',
filename='uploads/system_icon/back.png') }}" alt="Go back to homepage">
                        </a>
                    </nav>
                    <main>
                        <form method="POST">
                            <h1>Вхід у особистий кабінет</h1>
                            <div class="form-group">
                                <label for="username" class="label">Логін</label>
                                <input type="text" id="username" name="username"
placeholder="Введіть свій логін" required>
                            </div>
                            <div class="form-group">
                                <label                                  for="password"
class="label">Пароль</label>
                                <input       type="password"       id="password"
name="password" placeholder="Введіть свій пароль" required>
                            </div>
                            <div class="submit-button">
                                <button type="submit">Увійти</button>
                            </div>
                        </form>
                        <div class="flashes">
                            {% with messages = get_flashed_messages() %}
                                {% if messages %}
                                    <ul class="messages">
                                        {% for message in messages %}
                                            <li>{{ message }}</li>
                                        {% endfor %}
                                    </ul>
                                {% endif %}
                            {% endwith %}
                        </div>
                    </main>
            </div>
        </div>
    </body>

    </html>
```

- register.html:

```
    <!DOCTYPE html>
    <html lang="uk">

    <head>
        <meta charset="UTF-8">
```

```
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Реєстрація</title>
        <link      rel="icon"      href="static/uploads/system_icon/favicon.png"
type="image/x-icon">
        <link   rel="stylesheet"   type="text/css"   href="{{   url_for('static',
filename='register.css') }}">
    </head>

    <body>
        <div class="header">
            <img     src="static/uploads/system_icon/favicon.png"     alt="Logo"
class="logo">
            <h1><span   style="color:   #2e4957;">Gas</span><span   style="color:
#21b7ed;">Info</span></h1>
        </div>
        <div             class="background-image"             style="background:
url('static/uploads/system_icon/background.png')    no-repeat    center    center;
background-size: cover;">
            <div class="container">
                <nav class="navigation">
                    <a href="{{ url_for('main.index') }}">
                        <img              src="{{              url_for('static',
filename='uploads/system_icon/back.png') }}" alt="Go back to homepage">
                    </a>
                </nav>
                <h1>Створення особистого кабінету</h1>
                <main>
                    <figure class="image-container">
                        <img  src="static/uploads/user_icon/user.png"  alt="User
Profile Picture" id="user-image">
                        <figcaption        id="image-overlay">Обрати        фото
кабінету</figcaption>
                    </figure>

                    <form method="POST" enctype="multipart/form-data">
                        <div class="form-group">
                            <label for="username" class="label">Логін</label>
                            <input   type="text"   id="username"   name="username"
placeholder="Введіть логін" required>
                        </div>
                        <div class="form-group">
                            <label for="realname" class="label">Прізвище І'мя По
батькові</label>
                            <input   type="text"   id="realname"   name="realname"
placeholder="Введіть ПІБ" required>
                        </div>
                        <div class="form-group">
                            <label  for="phone"  class="label">Контактний  номер
телефону</label>
                            <input    type="text"    id="phone"    name="phone"
placeholder="+380..." required>
                        </div>
                        <div class="form-group">
                            <label for="password" class="label">Пароль</label>
                            <input type="password" id="password" name="password"
placeholder="Введіть пароль" required>
                        </div>
                        <input           type="file"           id="profile_photo"
name="profile_photo">
                        <button             type="submit"             class="btn-
register">Зареєструвати</button>
                    </form>
                    <div class="flashes">
```

```
                        {% with messages = get_flashed_messages() %}
                        {% if messages %}
                        <ul class="messages">
                            {% for message in messages %}
                            <li>{{ message }}</li>
                            {% endfor %}
                        </ul>
                        {% endif %}
                        {% endwith %}
                    </div>
                </main>
            </div>
        </div>

        <script     src="{{     url_for('static',     filename='js/register.js')
}}"></script>
    </body>

    </html>
```

- user_home.html:

```
    <!DOCTYPE html>
    <html lang="uk">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link           rel="icon"           href="{{           url_for('static',
filename='uploads/system_icon/favicon.png') }}" type="image/x-icon">
        <title>Головна</title>
        <link         rel="stylesheet"         href="{{         url_for('static',
filename='user_home.css') }}">
        <link
href="https://fonts.googleapis.com/css?family=Montserrat&display=swap"
rel="stylesheet">
    </head>
    <body>
        <div class="header">
            <div class="logo-and-title">
                <img                  src="{{                  url_for('static',
filename='uploads/system_icon/favicon.png') }}" alt="Logo" class="logo">
                <h1><span  style="color:  #2e4957;">Gas</span><span  style="color:
#21b7ed;">Info</span></h1>
            </div>
            <nav class="header-nav">
                <a href="#" onclick="showContent('meter')">Лічильник</a>
                <a href="#" onclick="showContent('check')">Квитанції</a>
                <a href="#" onclick="showContent('stat')">Статистика</a>
                <a href="#" onclick="showContent('profile')">Профіль</a>
                <a href="{{ url_for('main.logout') }}">Вихід</a>
            </nav>
        </div>
        <div id="dynamic-content" class="hidden">
            <div id="meter" class="content">
                <div class="button-container">
                    <button    class="add_meter"    id="open-form-button">Додати
лічильник</button>
                </div>
                <div id="form-container" style="display: none;">
                    <form id="add-meter-form">
                        <h2>Додати новий лічильник</h2>
                        <label for="meter-number">Номер лічильника:</label>
                        <input  type="text"  id="meter-number"  name="meter_number"
pattern="\d*" required>
```

```
                        <label for="meter-model">Модель лічильника:</label>
                        <input  type="text"  id="meter-model"  name="meter_model"
required>

                        <label for="tariff">Тариф (ціна за куб. м.):</label>
                        <input    type="number"    id="tariff"    name="tariff"
step="0.01" required>

                        <label for="subsidy">Субсидія:</label>
                        <input type="checkbox" id="subsidy" name="subsidy">

                        <label for="privilege">Пільга:</label>
                        <input type="checkbox" id="privilege" name="privilege">

                        <label for="address">Адреса:</label>
                        <input type="text" id="address" name="address" required>

                        <div class="button-container-form">
                            <button         type="submit"         class="submit-
button">Додати</button>
                            <button      type="button"      class="cancel-button"
onclick="closeForm()">Скасувати</button>
                        </div>
                    </form>
                </div>
                <div id="meters-table-container">
                    <table id="meters-table">
                        <thead>
                            <tr>
                                <th>Номер лічильника</th>
                                <th>Модель лічильника</th>
                                <th>Тариф</th>
                                <th>Субсидія</th>
                                <th>Пільга</th>
                                <th>Адреса</th>
                                <th>Статус</th>
                                <th>Видалення</th>
                            </tr>
                        </thead>
                        <tbody>
                        </tbody>
                    </table>
                </div>
            </div>

            <div id="check" class="content">
                <div class="button-container">

                    <button      class="add_receipt"      id="open-receipt-form-
button">Додати нову квитанцію</button>
                </div>
                <div id="receipts-table-container">
                    <table id="receipts-table">
                        <thead>
                            <tr>
                                <th>ID квитанції</th>
                                <th>Сума</th>
                                <th>Статус</th>
                                <th>Дата</th>
                                <th>Лічильник</th>
                                <th>Субсидія</th>
                                <th>Пільга</th>
```

```
                                <th>Тариф</th>
                                <th>Видалення</th>
                            </tr>
                        </thead>
                        <tbody>
                            <!-- Дані будуть додані динамічно -->
                        </tbody>
                    </table>
                </div>
                <div id="receipt-form-container" style="display: none;">
                    <form id="add-receipt-form">
                        <h2>Додати нову квитанцію</h2>
                        <label for="meter-select">Вибір лічильника:</label>
                        <select         id="meter-select"         name="meter_id"
required></select>

                        <label  for="amount-value">Об'єм  спожитого  газу  (куб.
м):</label>
                        <input          type="number"          id="amount-value"
name="amount_value" step="0.01" required>

                        <label for="subsidy-value">Субсидія (%):</label>
                        <input  type="number"  id="subsidy-value"  name="subsidy"
disabled>

                        <label for="privilege-value">Пільга (%):</label>
                        <input         type="number"         id="privilege-value"
name="privilege" disabled>

                        <p>Сума      до      сплати:      <span      id="payment-
amount">0.00</span></p>

                        <div class="button-container-form">
                            <button      type="button"      id="calculate-button"
class="calculate-button" onclick="calculateReceipt()">Обчислити рахунок</button>
                            <button       type="submit"       id="submit-receipt"
class="submit-button">Додати</button>
                            <button       type="button"       id="cancel-receipt"
class="cancel-button" onclick="closeReceiptForm()">Скасувати</button>
                        </div>

                    </form>
                </div>
            </div>
            <div id="stat" class="content">
                <h1>Статистика</h1>: <p> <i> для оновлення даних потрібно оновити
сторінку</i></p>
                <div class="chart-container">
                    <canvas id="monthlyGasUsageChart"></canvas>
                </div>
                <div class="chart-container">
                    <canvas id="priceTrendsChart"></canvas>
                </div>
                <div class="chart-container">
                    <canvas id="meterStatusChart"></canvas>
                </div>
            </div>
            <div id="profile" class="content">
                <div class="profile-card">
                    <h2>Профіль Користувача</h2>
                    <div class="profile-photo">
                        <img id="profile-photo" src="" alt="Profile Photo">
                    </div>
```

```
                    <p>Ім'я профілю: <span id="profile-username"></span></p>
                    <p>ПІБ: <span id="profile-real-name"></span></p>
                    <p>Контактний телефон: <span id="profile-phone"></span></p>
                    <p>Кількість лічильників <span id="meter-count"></span>,
підтверджені: <span id="active-meters-count"></span> </p>
                    <p>Кількість квитанцій: <span id="receipts-count"></span>,
підтверджені: <span id="paid-receipts-count"></span> </p>
                    <p>Дата реєстрації в системі: <span id="profile-registration-
date"></span></p>
                </div>
                <div class="button-container">
                    <button class="delete-account" id="delete-account">Видалити
обліковий запис</button>
                </div>
            </div>
        </div>
        <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
        <script src="{{ url_for('static', filename='js/register.js')
}}"></script>
        <script src="{{ url_for('static', filename='js/user.js') }}"></script>
        <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
        <script>
            var userId = "{{ user.id }}";
            </script>
    </body>
    </html>
```

Скрипти:

- admin.js:

```
function deleteUser(username) {
    Swal.fire({
        title: 'Ви не зможете відновити цю інформацію!',
        text: `Ви дійсно хочете видалити користувача: ${username}?`,
        icon: 'warning',
        showCancelButton: true,
        confirmButtonColor: '#ab0a0a',
        cancelButtonColor: '#007bff',
        confirmButtonText: 'Так, видалити!',
        cancelButtonText: 'Ні, відмінити'
    }).then((result) => {
        if (result.isConfirmed) {
            fetch(`/delete-user/${username}`, {
                method: 'DELETE'
            }).then(response => response.json())
            .then(data => {
                Swal.fire(
                    'Видалено!',
                    `Користувач ${username} був успішно видалений.`,
                    'success'
                );
                loadUsers();
            })
            .catch(error => {
                Swal.fire(
                    'Помилка!',
                    'Не вдалося видалити користувача: ' + error.message,
                    'error'
                );
            });
```

```
            }
        });
    }
    document.addEventListener("DOMContentLoaded", function() {
        loadUsers();
        let user_id_in;
        document.getElementById('searchInput').addEventListener('keyup',
function() {
            const searchValue = this.value.toLowerCase();
            const tableRows = document.querySelectorAll('#users-table-body tr');

            tableRows.forEach(row => {
                const    isVisible    =    Array.from(row.cells).some(cell    =>
cell.textContent.toLowerCase().includes(searchValue));
                row.style.display = isVisible ? '' : 'none';
            });
        });
    });

    document.getElementById('searchInput').addEventListener('keyup', function() {
        const searchValue = this.value.toLowerCase();
        const tableRows = document.querySelectorAll('#users-table-body tr');

        tableRows.forEach(row => {
            const     isVisible     =     Array.from(row.cells).some(cell     =>
cell.textContent.toLowerCase().includes(searchValue));
            row.style.display = isVisible ? '' : 'none';
        });
    });

    function loadUsers() {
        fetch('/users')
        .then(response => response.json())
        .then(data => {
            const tableBody = document.getElementById('users-table-body');

            if (!tableBody) {
                console.error("Couldn't find the table body element.");
                return;
            }
            tableBody.innerHTML = '';
            data.forEach(user => {
                const             registrationDate             =             new
Date(user.registration_date).toLocaleDateString('uk-UA');

                const row = document.createElement('tr');
                row.innerHTML = `
                    <td>${user.username}</td>
                    <td>${user.real_name}</td>
                    <td>${user.phone}</td>
                    <td>${registrationDate}</td>
                    <td>${user.inactive_meters_count}                          /
${user.inactive_receipts_count} </td>
                    <td>${user.active_meters_count}                           /
${user.active_receipts_count} </td>
                    <td><button       class="descr-user-button"       data-user-
id="${user.id}" data-user-name="${user.real_name}">Детальніше</button></td>
                    <td><button    id="open-user-data-form"  class="delete-user-
button" data-username="${user.username}">Видалити</button></td>
                `;
                tableBody.appendChild(row);
            });
            document.querySelectorAll('.descr-user-button').forEach(button => {
```

```
                button.addEventListener('click', function() {
                    const userId = this.getAttribute('data-user-id');
                    user_id_in =userId
                    const  realName  =  this.getAttribute('data-user-name');  //
Отримання імені користувача
                    document.getElementById('user-data-form-
container').style.display = 'block';
                    fetch(`/user/${userId}/meters-details`)
                    .then(response => response.json())
                    .then(data => {
                        displayMeterDetails(data,  realName);  //  Передача  імені
користувача
                    })
                    .catch(error => {
                        console.error('Error fetching meter details:', error);
                    });
                });
            });
            document.querySelectorAll('.delete-user-button').forEach(button => {
                button.addEventListener('click', function() {
                    const username = this.getAttribute('data-username');
                    deleteUser(username);
                });
            });
        })
        .catch(error => {
            console.error('Error fetching users data:', error);
        });
    }
    function closeDataUsertForm() {
        var formContainer = document.getElementById('user-data-form-container');
        console.log(formContainer);  //  Перевірте,  що  ви  отримуєте  правильний
елемент
        if (formContainer) {
            formContainer.style.display = 'none';
        } else {
            console.error("Can't find the form container element.");
        }
    }
    function displayMeterDetails(data, realName) {
        const    formContainer    =    document.getElementById('user-data-form-
container');
        const form = document.getElementById('user-data-form');
        let showConfirmButton = false;  // Змінна для відстеження статусу

        form.innerHTML = `<h2>Деталі лічильника та квитанцій користувача:<br><span
style="color: #21b7ed;">${realName}</span></h2>`;


        if (data.length === 0) {
            form.innerHTML += '<p>Дані відсутні</p>';
        } else {
            data.forEach(meter => {
                const meterInfo = document.createElement('div');
                meterInfo.innerHTML = `
                    <h3>Лічильник          №:          <span          style="color:
#21b7ed;">${meter.meter_number}</span></h3>
                    <p>Модель: ${meter.meter_model}</p>
                    <p>Тариф:                  <span                 style="color:
#21b7ed;">${meter.tariff}</span></p>
                    <p>Адреса: ${meter.address}</p>
                    <p>Статус:    ${meter.status    ?    '<span    style="color:
#0aab22;">Активний</span>' : '<span style="color: #ab0a0a">Неактивний</span>'}</p>
```

```
                    <h3>Квитанції:</h3>
                `;
                if (!meter.status) {
                    showConfirmButton = true;  // Якщо статус лічильника false
                }

                meter.receipts.forEach(receipt => {
                    if (!receipt.status) {
                        showConfirmButton = true;  // Якщо статус квитанції false
                    }
                    const receiptInfo = document.createElement('p');
                    receiptInfo.innerHTML = `Квитанція ID: ${receipt.id}, Сума: <span style="color: #21b7ed;">${receipt.payment_amount}</span> грн., Дата: <span style="color: #21b7ed;">${receipt.date}</span>, Статус: ${receipt.status ? '<span style="color: #0aab22;">Оплачено</span>' : '<span style="color: #ab0a0a">Не оплачено</span>'}`;
                    meterInfo.appendChild(receiptInfo);
                });

                form.appendChild(meterInfo);
            });
        }

        if (showConfirmButton) {
            const confirmButton = document.createElement('button');
            confirmButton.textContent = 'Підтвердити дані';
            confirmButton.className = 'confirm-button';
            confirmButton.onclick = function(event) {
                event.preventDefault();
                Swal.fire({
                    title: 'Ви впевнені?',
                    text: "Ви не зможете відмінити цю дію!",
                    icon: 'warning',
                    showCancelButton: true,
                    confirmButtonColor: '#ab0a0a',
                    cancelButtonColor: '#007bff',
                    confirmButtonText: 'Так, підтвердити!',
                    cancelButtonText: 'Ні, скасувати'
                }).then((result) => {
                    if (result.isConfirmed) {
                        fetch(`/user/${user_id_in}/confirm-data`, {
                            method: 'POST',
                            headers: {
                                'Content-Type': 'application/json'
                            },
                            body: JSON.stringify({userId: userId})
                        })
                        .then(response => response.json())
                        .then(data => {
                            if(data.error) {
                                Swal.fire('Помилка!', data.error, 'error');
                            } else {
                                Swal.fire('Підтверджено!', data.message, 'success');
                                closeDataUsertForm();
                            }
                        })
                        .catch(error => {
                            Swal.fire('Помилка!', 'Не вдалося підтвердити дані: ' + error.message, 'error');
                        });
                    }
                });
```

```
            };
            form.appendChild(confirmButton);
        }
        formContainer.style.display = 'block';
    }
```

- register.js:

```
    document.addEventListener("DOMContentLoaded", function() {
        const userImage = document.getElementById("user-image");
        const profilePhotoInput = document.getElementById("profile_photo");

        document.getElementById("image-overlay").addEventListener("click",
function() {
            profilePhotoInput.click();
        });

        profilePhotoInput.addEventListener("change", function() {
            const selectedFile = profilePhotoInput.files[0];
            if (selectedFile) {
                const url = URL.createObjectURL(selectedFile);
                userImage.src = url;
            }
        });
    });
```

- user.js:

```
    function fetchUserInfo() {
        fetch('/user/get-user-info')
            .then(response => response.json())
            .then(data => {
                if (data.error) {
                    alert(data.error);
                    return;
                }
                document.getElementById('profile-username').textContent         =
data.username;
                document.getElementById('profile-real-name').textContent        =
data.real_name;
                document.getElementById('profile-phone').textContent            =
data.phone;
                document.getElementById('profile-registration-date').textContent
= data.registration_date;
                document.getElementById('profile-photo').src                    =
data.profile_photo_url;
                document.getElementById('meter-count').textContent              =
data.total_meters;
                document.getElementById('active-meters-count').textContent      =
data.active_meters;
                document.getElementById('receipts-count').textContent           =
data.receipts_count;
                document.getElementById('paid-receipts-count').textContent      =
data.paid_receipts_count;
            })
            .catch(error => {
                console.error('Error fetching user info:', error);
            });
    }
    function showContent(contentId) {
        const dynamicContent = document.getElementById('dynamic-content');
        if (!dynamicContent) {
```

```
            console.error("Couldn't find the element with ID 'dynamic-content'.");
            return;
        }

        dynamicContent.style.display = 'block';

        const contentSections = dynamicContent.querySelectorAll('.content');
        contentSections.forEach(section => {
            section.style.display = 'none';
        });

        const selectedContent = document.getElementById(contentId);
        if (!selectedContent) {
            console.error(`Couldn't find the element with ID '${contentId}'.`);
            return;
        }

        selectedContent.style.display = 'block';
        if (contentId === 'profile') {
            fetchUserInfo();
        }
        if (contentId === 'meter') {
            loadMeters();
        }
        if (contentId === 'check') {
            loadMetersDropdown();
            loadReceipts();
        }
        if (contentId === 'stat') {
            loadAnalyticsData();
        }

    }

    document.getElementById('delete-account').addEventListener('click',
function() {
        Swal.fire({
            title: 'Ви впевнені?',
            text: "Ви не зможете відновити цей профіль!",
            icon: 'warning',
            showCancelButton: true,
            confirmButtonColor: '#9a00fd',
            cancelButtonColor: '#007bff',
            confirmButtonText: 'Так, видалити!',
            cancelButtonText: 'Скасувати'
        }).then((result) => {
            if (result.isConfirmed) {
                fetch('/delete-user', { method: 'POST' })
                .then(response => response.json())
                .then(data => {
                    if (data.message) {
                        Swal.fire('Видалено!',                    data.message,
'success').then((result) => {
                            if (result.isConfirmed) {
                                window.location.href = data.redirect;
                            }
                        });
                    } else {
                        Swal.fire('Помилка!', data.error, 'error');
                    }
                });
            }
        });
```

```
    });


    document.getElementById('open-form-button').addEventListener('click',
function() {
        console.log("Button clicked!");
        document.getElementById('form-container').style.display = 'block';
    });

    function closeForm() {
        document.getElementById('form-container').style.display = 'none';
    }

    document.getElementById('add-meter-form').addEventListener('submit',
function(event) {
        event.preventDefault();
        const formData = new FormData(this);
        fetch('/add-meter', {
            method: 'POST',
            body: formData
        })
        .then(response => response.json())
        .then(data => {
            if (data.success) {
                Swal.fire('Успіх',       'Лічильник       додано       успішно!',
'success').then(() => {
                    loadMeters();
                    closeForm();
                });
            } else {
                Swal.fire('Помилка', data.error, 'error');
            }
        })
        .catch(error => {
            console.error('Error:', error);
        });
    });

    function loadMeters() {
        fetch('/get-meters')
        .then(response => response.json())
        .then(meters => {
            const       tableBody       =       document.getElementById('meters-
table').querySelector('tbody');
            tableBody.innerHTML = '';
            meters.forEach(meter => {
                const row = document.createElement('tr');
                row.innerHTML = `
                    <td>${meter.meter_number}</td>
                    <td>${meter.meter_model}</td>
                    <td>${meter.tariff.toFixed(2)}</td>
                    <td>${meter.subsidy === 1.0 ? 'Так' : 'Ні'}</td>
                    <td>${meter.privilege === 1.0 ? 'Так' : 'Ні'}</td>
                    <td>${meter.address}</td>
                    <td>${meter.status ? 'Активний' : 'Неактивний'}</td>
                    <td><button
onclick="confirmDeletion('${meter.meter_number}')"            class="delete-meter-
button">Видалити</button></td>
                `;
                tableBody.appendChild(row);
            });
        })
```

```
            .catch(error => console.error('Error loading meters:', error));
        }



        function confirmDeletion(meterNumber) {
            Swal.fire({
                title: 'Ви впевнені?',
                text: `Ви дійсно бажаєте видалити лічильник з номером
${meterNumber}?`,
                icon: 'warning',
                showCancelButton: true,
                confirmButtonColor: '#ab0a0a',
                cancelButtonColor: '#007bff',
                confirmButtonText: 'Так, видалити!',
                cancelButtonText: 'Ні, скасувати'
            }).then((result) => {
                if (result.isConfirmed) {
                    deleteMeter(meterNumber);
                }
            });
        }


        function deleteMeter(meterNumber) {
            fetch(`/delete-meter/${meterNumber}`, { method: 'DELETE' })
            .then(response => response.json())
            .then(data => {
                if (data.success) {
                    Swal.fire('Видалено!', 'Лічильник успішно видалено.', 'success');
                    loadMeters();
                } else {
                    Swal.fire('Помилка', 'Не вдалося видалити лічильник.', 'error');
                }
            })
            .catch(error => console.error('Error:', error));
        }




        function loadMetersDropdown() {
            fetch('/get-meters')
            .then(response => response.json())
            .then(meters => {
                const select = document.getElementById('meter-select');
                select.innerHTML = ''; // Clear existing options before adding new
ones
                meters.forEach(meter => {
                    const    option    =    new    Option(`${meter.meter_number}    -
${meter.address}`, meter.meter_number);
                    option.dataset.tariff = meter.tariff;
                    option.dataset.subsidy = (meter.subsidy === 1.0).toString(); //
Convert to string "true" or "false"
                    option.dataset.privilege = (meter.privilege === 1.0).toString();
// Convert to string "true" or "false"
                    select.appendChild(option);
                });
                select.dispatchEvent(new Event('change')); // Trigger change event to
initialize form state
            })
            .catch(error => console.error('Error loading meters:', error));
```

```
    }

    document.getElementById('meter-select').addEventListener('change', function()
{
        const selectedOption = this.options[this.selectedIndex];
        const subsidyInput = document.getElementById('subsidy-value');
        const privilegeInput = document.getElementById('privilege-value');

        // Enable or disable subsidy input based on the selected meter's data
        subsidyInput.disabled = !(selectedOption.dataset.subsidy === 'true');
        subsidyInput.value = subsidyInput.disabled ? '0' : ''; // Reset value if
disabled
        subsidyInput.placeholder = subsidyInput.disabled ? 'Субсидії недоступні'
: 'Введіть % субсидії';

        // Enable or disable privilege input based on the selected meter's data
        privilegeInput.disabled = !(selectedOption.dataset.privilege === 'true');
        privilegeInput.value = privilegeInput.disabled ? '0' : ''; // Reset value
if disabled
        privilegeInput.placeholder = privilegeInput.disabled ? 'Пільги недоступні'
: 'Введіть % пільги';
    });

    document.getElementById('open-receipt-form-
button').addEventListener('click', function() {
        document.getElementById('receipt-form-container').style.display        =
'block';
    });

    function closeReceiptForm() {
        document.getElementById('receipt-form-container').style.display = 'none';
    }

    document.getElementById('add-receipt-form').addEventListener('submit',
function(event) {
        event.preventDefault();
        const       selectedMeter       =       document.querySelector('#meter-
select').selectedOptions[0];
        const       paymentAmount       =       document.getElementById('payment-
amount').textContent;
        const data = {
            meter_select: selectedMeter.value,
            amount_value: document.getElementById('amount-value').value,
            payment_amount: paymentAmount,
            subsidy_value: document.getElementById('subsidy-value').value,
            privilege_value: document.getElementById('privilege-value').value
        };

        fetch('/add-receipt', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(data)
        })
        .then(response => {
            if (!response.ok) {
                throw new Error('Network response was not ok.');
            }
            return response.json();
        })
        .then(data => {
            if (data.success) {
```

```
                closeReceiptForm();
                loadReceipts();
            } else {
                alert('Помилка: ' + data.error);
            }
        })
        .catch(error => {
            console.error('Error:', error);
            alert('Помилка: Не вдалося відправити дані');
        });
    });

    function calculateReceipt() {
        const selectedMeter = document.querySelector('#meter-
select').selectedOptions[0];
        const amountValue = parseFloat(document.getElementById('amount-
value').value) || 0; // Ensures a default of 0 if parse fails
        const tariff = parseFloat(selectedMeter.dataset.tariff) || 0; // Default
to 0 if parsing fails

        // Retrieve subsidy and privilege values, use 0 as default if parse fails
or if the respective checkbox is unchecked
        const subsidy = selectedMeter.dataset.subsidy ?
parseFloat(document.getElementById('subsidy-value').value) || 0 : 0;
        const privilege = selectedMeter.dataset.privilege ?
parseFloat(document.getElementById('privilege-value').value) || 0 : 0;

        // Calculate the amounts for subsidy and privilege
        const subsidyAmount = (tariff * amountValue * (subsidy / 100));
        const privilegeAmount = (tariff * amountValue * (privilege / 100));

        // Calculate the final payment amount
        const paymentAmount = (tariff * amountValue) - subsidyAmount -
privilegeAmount;

        // Update the payment amount text, ensuring it never shows NaN
        document.getElementById('payment-amount').textContent =
isNaN(paymentAmount) ? '0.00' : paymentAmount.toFixed(2);
    }
    function loadReceipts() {
        fetch('/get-all-receipts')
        .then(response => {
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            return response.json();
        })
        .then(receipts => {
            console.log('Receipts loaded:', receipts);
            const tableBody = document.getElementById('receipts-
table').querySelector('tbody');
            tableBody.innerHTML = ''; // Clear the table body before adding new
rows
            receipts.forEach(receipt => {
                const row = document.createElement('tr');
                row.innerHTML = `
                    <td>${receipt.id}</td>
                    <td>${receipt.payment_amount.toFixed(2)}</td>
                    <td>${receipt.status ? 'Оплачено' : 'Не оплачено'}</td>
                    <td>${receipt.date}</td>
                    <td>${receipt.meter_number}</td>
                    <td>${receipt.subsidy.toFixed(2)}</td>
                    <td>${receipt.privilege.toFixed(2)}</td>
```

```
                    <td>${receipt.tariff.toFixed(2)}</td>
                    <td><button   onclick="confirmReceiptDeletion(${receipt.id})"
class="delete-receipt-button">Видалити</button></td>
                `;
                tableBody.appendChild(row);
            });
        })
        .catch(error => {
            console.error('Error loading receipts:', error);
            alert('Error loading receipts: ' + error.message);
        });
    }

    function confirmReceiptDeletion(receiptId) {
        Swal.fire({
            title: 'Ви впевнені?',
            text: "Ви дійсно хочете видалити цю квитанцію?",
            icon: 'warning',
            showCancelButton: true,
            confirmButtonColor: '#ab0a0a',
            cancelButtonColor: '#007bff',
            confirmButtonText: 'Так, видалити!',
            cancelButtonText: 'Ні, скасувати'
        }).then((result) => {
            if (result.isConfirmed) {
                deleteReceipt(receiptId);
            }
        });
    }

    function deleteReceipt(receiptId) {
        fetch(`/delete-receipt/${receiptId}`, { method: 'DELETE' })
        .then(response => response.json())
        .then(data => {
            if (data.success) {
                Swal.fire('Видалено!', 'Квитанція успішно видалена.', 'success');
                loadReceipts();
            } else {
                Swal.fire('Помилка', data.error, 'error');
            }
        })
        .catch(error => {
            console.error('Error:', error);
            alert('Помилка: Не вдалося видалити квитанцію');
        });
    }

    function loadAnalyticsData() {
        fetch('/analytics/data')
        .then(response => response.json())
        .then(data => {
            // Перший графік (лінійний графік) для використання газу за місяць
            const monthlyGasLabels = data.monthly_gas_usage.map(item => {
                if (item.meter_number !== undefined) {
                    return `${item.month} (${item.meter_number})`;
                } else {
                    return item.month;
                }
            });
            createChart('monthlyGasUsageChart',  'Графік споживання газу за
місяць',  'line',  data.monthly_gas_usage,  'total_gas',  'month',  'Споживання газу
(куб. м)', monthlyGasLabels);
```

```
            // Другий графік (лінійний графік) для динаміки цін на газ
            createChart('priceTrendsChart', 'Динаміка цін на газ', 'line',
data.price_trends, 'tariff', 'date', 'Тариф за куб. м');

            // Круговий графік для статусів лічильників
            const meterStatusLabels = data.meter_usage_stats.map(item =>
`Лічильник ${item.meter_number}`);
            createChart('meterStatusChart', 'Статус лічильників', 'pie',
data.meter_usage_stats, 'total_gas', 'meter_number', '', meterStatusLabels);
        })
        .catch(error => {
            console.error('Error loading analytics data:', error);
        });
    }


    function getRandomColor() {
        const letters = '0123456789ABCDEF';
        let color = '#';
        for (let i = 0; i < 6; i++) {
            color += letters[Math.floor(Math.random() * 16)];
        }
        return color;
    }

    function createChart(chartId, title, type, data, valueField, labelField,
yAxisLabel, labels) {
        const ctx = document.getElementById(chartId).getContext('2d');
        const values = data.map(item => item[valueField]); // Отримуємо значення
для графіка
        const colors = values.map(() => getRandomColor()); // Генеруємо випадкові
кольори для кожного значення

        // Створюємо графік за допомогою Chart.js
        const chart = new Chart(ctx, {
            type: type,
            data: {
                labels: labels || data.map(item => item[labelField]), //
Використовуємо передані підписи, якщо вони є, в іншому випадку використовуємо дані
з масиву
                datasets: [{
                    label: yAxisLabel,
                    data: values,
                    backgroundColor: colors,
                    borderColor: colors,
                    borderWidth: 1
                }]
            },
            options: {
                responsive: true,
                maintainAspectRatio: false,
                scales: type !== 'pie' ? {
                    y: {
                        beginAtZero: true,
                        title: {
                            display: !!yAxisLabel,
                            text: yAxisLabel
                        }
                    }
                } : {}
            }
        });
    }
```

Серверна частина:

Маршрути:

- main.py:

```
from flask import render_template, request, session, flash, redirect, url_for,Blueprint
from flask import current_app as app
from flask import Blueprint, request, jsonify, abort
from app.extensions import db
from app.models import User, Meter, Receipt
import os
from werkzeug.utils import secure_filename
from datetime import datetime
from sqlalchemy.sql import extract
from collections import defaultdict
main = Blueprint('main', __name__)
@main.route('/')
def index():
    return render_template('index.html')
def user_login(user):
    db.session.commit()
    session['user_id'] = user.id

@main.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        realname = request.form['realname']
        phone = request.form['phone']
        password = request.form['password']
        existing_user = User.query.filter_by(username=username).first()
        existing_phone = User.query.filter_by(phone=phone).first()
        if existing_user:
            flash('Користувач з таким логіном вже існує. Спробуйте інший!', 'error')
            return render_template('register.html')

        if existing_phone:
            flash('Користувач з таким номером телефону вже існує.', 'error')
            return render_template('register.html')
        if not (phone.startswith('+') and len(phone) == 13 and phone[1:].isdigit()):
            flash('Номер телефону повинен починатися з +, містити тільки цифри та мати довжину 12 символів без +.', 'error')
            return render_template('register.html')
        if 'profile_photo' in request.files and request.files['profile_photo'].filename != '':
            profile_photo = save_profile_image(request.files['profile_photo'])
        else:
            profile_photo = 'user.png'

        if len(username) < 8 or len(password) < 8 or len(realname) < 8:
            flash('Поля Логін, Пароль та ПІБ мають містити не менше 8 символів!', 'error')
        elif not username or not password or not realname or not phone:
            flash('Заповніть всі необхідні поля!', 'error')
        else:
            role = 'admin' if username == 'admin_admin' and password == 'admin_admin' else 'user'
```

```
                new_user = User(
                    username=username,
                    real_name=realname,
                    phone=phone,
                    password=password,
                    profile_photo=profile_photo,
                    role=role,
                    registration_date=datetime.now()  # Додаємо фактичну дату
реєстрації
                )
                db.session.add(new_user)
                db.session.commit()
                flash('Реєстрація пройшла успішно!', 'success')
                session.clear()
                session['user_id'] = new_user.id
                return redirect(url_for(f'user.{new_user.role}_home'))

        return render_template('register.html')


    def save_profile_image(image):
        filename = secure_filename(image.filename)
        file_path = os.path.join(app.config['USER_ICON'], filename)
        image.save(file_path)
        return filename

    @main.route('/login', methods=['GET', 'POST'])
    def login():
        if request.method == 'POST':
            username = request.form['username']
            password = request.form['password']

            if len(username) < 8 or len(password) < 8:
                flash('Поля Логін та Пароль мають містити не менше 8 символів!',
'error')
                return render_template('login.html')

            user = User.query.filter_by(
                username=username, password=password).first()

            if not user:
                flash('Помилка входу! Перевірте дані та спробуйте ще раз.',
'error')
                return render_template('login.html')
            user_login(user)
            flash('Вхід успішний!', 'success')
            return redirect(url_for(f'user.{user.role}_home'))

        return render_template('login.html')

    @main.route('/logout')
    def logout():
        if 'user_id' in session:
            user_id = session['user_id']
            user = User.query.get(user_id)
        session.clear()
        return redirect(url_for('main.index'))


    @main.route('/analytics/data')
    def get_analytics_data():
        if 'user_id' not in session:
            return jsonify({'error': 'Not authenticated'}), 401
```

```
        user_id = session['user_id']
        user = User.query.get(user_id)
        if not user:
            return jsonify({'error': 'User not found'}), 404

        # Отримання всіх квитанцій для користувача
        receipts = Receipt.query.join(Meter).filter(Meter.user_id ==
user_id).all()

        # Споживання газу по місяцях
        monthly_gas_usage = defaultdict(float)
        for receipt in receipts:
            month = receipt.date.strftime('%m')
            monthly_gas_usage[month] += receipt.amount_value

        # Динаміка зміни ціни газу
        price_trends = [{'date': receipt.date.strftime('%Y-%m-%d'), 'tariff':
receipt.tariff} for receipt in receipts]

        # Статуси оплати квитанцій
        payment_status_count = defaultdict(int)
        for receipt in receipts:
            payment_status_count[str(receipt.status)] += 1

        # Статистика споживання газу по лічильникам
        meter_usage_stats = defaultdict(float)
        for receipt in receipts:
            meter_usage_stats[receipt.meter.meter_number] += receipt.amount_value

        data = {
            'monthly_gas_usage': [{'month': month, 'total_gas': total_gas} for
month, total_gas in monthly_gas_usage.items()],
            'price_trends': price_trends,
            'payment_status_count': [{'status': status, 'count': count} for
status, count in payment_status_count.items()],
            'meter_usage_stats': [{'meter_number': meter_number, 'total_gas':
total_gas} for meter_number, total_gas in meter_usage_stats.items()]
        }

        return jsonify(data)

    @main.route('/users')
    def users():
        # Отримання користувачів з роллю не адміністратора
        users = User.query.filter_by(role='user').all()

        # Створення списку користувачів з їхньою інформацією
        users_info = []
        for user in users:
            # Підрахунок активних і неактивних лічильників
            active_meters_count = len([m for m in user.meters if m.status == True])
            inactive_meters_count = len([m for m in user.meters if m.status ==
False])

            # Підрахунок активних і неактивних квитанцій
            active_receipts_count =
db.session.query(Receipt).join(Meter).filter(Meter.user_id == user.id,
Receipt.status == True).count()
            inactive_receipts_count =
db.session.query(Receipt).join(Meter).filter(Meter.user_id == user.id,
Receipt.status == False).count()
```

```
            # Створення словника з інформацією про користувача
            user_info = {
                'id': user.id,
                'username': user.username,
                'real_name': user.real_name,
                'phone': user.phone,
                'registration_date': user.registration_date.strftime("%Y-%m-%d"),
                'active_meters_count': active_meters_count,
                'inactive_meters_count': inactive_meters_count,
                'active_receipts_count': active_receipts_count,
                'inactive_receipts_count': inactive_receipts_count
            }
            users_info.append(user_info)

        return jsonify(users_info)

    @main.route('/delete-user/<string:username>', methods=['DELETE'])
    def delete_user(username):
        if 'user_id' not in session:
            return jsonify({'error': 'Not authenticated'}), 401

        # Припускаємо, що ви зберігаєте ID користувача в сесії
        current_user = User.query.get(session['user_id'])

        if not current_user or current_user.role != 'admin':
            return jsonify({'error': 'Unauthorized'}), 403

        user_to_delete = User.query.filter_by(username=username).first()
        if user_to_delete:
            db.session.delete(user_to_delete)
            db.session.commit()
            return jsonify({'message': 'Профіль видалено!', 'redirect':
url_for('main.index')}), 200
        else:
            return jsonify({'error': 'User not found'}), 404


    @main.route('/user/<int:user_id>/meters-details', methods=['GET'])
    def get_meters_details(user_id):
        if 'user_id' not in session:
            return jsonify({'error': 'User not logged in'}), 401

        # Перевірка чи користувач у сесії є адміністратором
        current_user = User.query.get(session['user_id'])
        if not current_user or current_user.role != 'admin':
            return jsonify({'error': 'Unauthorized'}), 403

        user = User.query.get(user_id)
        if not user:
            return jsonify({'error': 'User not found'}), 404

        meters = Meter.query.filter_by(user_id=user_id).all()
        meters_info = []

        for meter in meters:
            receipts_info = [{
                'id': receipt.id,
                'payment_amount': receipt.payment_amount,
                'status': receipt.status,
                'date': receipt.date.strftime('%Y-%m-%d'),
                'subsidy': receipt.subsidy,
                'amount_value': receipt.amount_value,
                'privilege': receipt.privilege,
```

```
                'tariff': receipt.tariff
            } for receipt in meter.receipts]

            meters_info.append({
                'id': meter.id,
                'meter_number': meter.meter_number,
                'meter_model': meter.meter_model,
                'tariff': meter.tariff,
                'subsidy': meter.subsidy,
                'privilege': meter.privilege,
                'address': meter.address,
                'status': meter.status,
                'receipts': receipts_info
            })

        return jsonify(meters_info)

    @main.route('/user/<int:user_id>/confirm-data', methods=['POST'])
    def confirm_user_data(user_id):
        if 'user_id' not in session:
            return jsonify({'error': 'Unauthorized'}), 403

        # Отримуємо користувача з сесії
        current_user = User.query.get(session['user_id'])

        # Перевіряємо, чи користувач є адміністратором
        if not current_user or current_user.role != 'admin':
            return jsonify({'error': 'Unauthorized'}), 403

        try:
            # Знаходимо всі лічильники та квитанції для заданого user_id зі
статусом False
            meters = Meter.query.filter_by(user_id=user_id, status=False).all()
            for meter in meters:
                print(f"Changing status of Meter ID: {meter.id}, Current Status:
{meter.status}")
                meter.status = True
                for receipt in meter.receipts:
                    if not receipt.status:
                        print(f"Changing  status  of  Receipt  ID:  {receipt.id},
Current Status: {receipt.status}")
                        receipt.status = True

            db.session.commit()  # Збереження змін у базі даних
            return jsonify({'message': 'Data confirmed successfully'}), 200
        except Exception as e:
            db.session.rollback()  # Відкіт змін у разі помилки
            return jsonify({'error': str(e)}), 500
```

- meter.py:

```
    from  flask  import  render_template,  request,  session,  flash,  redirect,
url_for,Blueprint
    from flask import current_app as app
    from flask import Blueprint, request, jsonify, abort
    from app.extensions import db
    from app.models import User, Meter, Receipt
    import os
    from werkzeug.utils import secure_filename
    from datetime import datetime
    meter = Blueprint('meter', __name__)

    @meter.route('/add-meter', methods=['POST'])
    def add_meter():
```

```
        if 'user_id' not in session:
            return jsonify({'error': 'Not authenticated'}), 401

        user_id = session['user_id']
        meter_number = request.form['meter_number']
        meter_model = request.form['meter_model']
        tariff = float(request.form['tariff'])
        subsidy = 1.0 if request.form.get('subsidy') == 'on' else 0.0
        privilege = 1.0 if request.form.get('privilege') == 'on' else 0.0
        address = request.form['address']

        if Meter.query.filter_by(meter_number=meter_number).first():
            return jsonify({'error': 'Meter with that number already exists'}),
400

        new_meter = Meter(
            user_id=user_id,
            meter_number=meter_number,
            meter_model=meter_model,
            tariff=tariff,
            subsidy=subsidy,
            privilege=privilege,
            address=address,
            status=False
        )
        db.session.add(new_meter)
        db.session.commit()

        return jsonify({'success': 'Лічильник додано успішно'})

    @meter.route('/get-meters')
    def get_meters():
        if 'user_id' not in session:
            return jsonify({'error': 'Not authenticated'}), 401

        user_id = session['user_id']
        user = User.query.get(user_id)
        if not user:
            return jsonify({'error': 'User not found'}), 404

        meters = [{
            'meter_number': meter.meter_number,
            'meter_model': meter.meter_model,
            'tariff': meter.tariff,
            'subsidy': meter.subsidy,
            'privilege': meter.privilege,
            'address': meter.address,
            'status': meter.status
        } for meter in user.meters]

        return jsonify(meters)


    @meter.route('/delete-meter/<string:meter_number>', methods=['DELETE'])
    def delete_meter(meter_number):
        if 'user_id' not in session:
            return jsonify({'error': 'Not authenticated'}), 401

        user_id = session['user_id']
        user = User.query.get(user_id)
        if not user:
            return jsonify({'error': 'User not found'}), 404
```

```
        meter = Meter.query.filter_by(user_id=user_id,
meter_number=meter_number).first()
        if not meter:
            return jsonify({'error': 'Meter not found or does not belong to you'}),
404

        try:
            # Видалення всіх квитанцій, пов'язаних з видаленим лічильником
            receipts = Receipt.query.filter_by(meter_id=meter.id).all()
            for receipt in receipts:
                db.session.delete(receipt)
            db.session.delete(meter)
            db.session.commit()
            return jsonify({'success': 'Meter and associated receipts deleted
successfully'}), 200
        except Exception as e:
            db.session.rollback()
            return jsonify({'error': 'Failed to delete the meter and associated
receipts', 'message': str(e)}), 500
```

- receipt.py:

```
    from flask import render_template, request, session, flash, redirect,
url_for,Blueprint
    from flask import current_app as app
    from flask import Blueprint, request, jsonify, abort
    from app.extensions import db
    from app.models import User, Meter, Receipt
    import os
    from werkzeug.utils import secure_filename
    from datetime import datetime
    receipt = Blueprint('receipt', __name__)

    @receipt.route('/add-receipt', methods=['POST'])
    def add_receipt():
        if 'user_id' not in session:
            return jsonify({'error': 'Not authenticated'}), 401

        data = request.get_json()  # Получение данных JSON
        user_id = session['user_id']

        meter_number = data.get('meter_select')
        amount_value = float(data.get('amount_value', 0))  # Безопечное
преобразование
        payment_amount = float(data.get('payment_amount', 0))
        submitted_subsidy = float(data.get('subsidy_value', 0))
        submitted_privilege = float(data.get('privilege_value', 0))

        # Поиск метра по номеру
        meter = Meter.query.filter_by(user_id=user_id,
meter_number=meter_number).first()
        if not meter:
            return jsonify({'error': 'Meter not found'}), 404

        # Only apply subsidy and privilege if they are available for this meter
        final_subsidy = submitted_subsidy if meter.subsidy == 1.0 else 0.0
        final_privilege = submitted_privilege if meter.privilege == 1.0 else 0.0

        new_receipt = Receipt(
            payment_amount=payment_amount,
            status=False,  # Статус оплаты по умолчанию False
            date=datetime.utcnow(),
            meter_id=meter.id,
```

```
            subsidy=final_subsidy,
            privilege=final_privilege,
            tariff=meter.tariff,
            amount_value=amount_value
        )

        db.session.add(new_receipt)
        db.session.commit()

        return jsonify({'success': 'Receipt added successfully'})


    @receipt.route('/get-all-receipts')
    def get_all_receipts():
        if 'user_id' not in session:
            return jsonify({'error': 'Not authenticated'}), 401

        user_id = session['user_id']
        # Join Receipts with Meter where Meter's user_id matches the session
user_id
        receipts = Receipt.query.join(Meter).filter(Meter.user_id ==
user_id).all()
        receipts_data = [{
            'id': receipt.id,
            'payment_amount': receipt.payment_amount,
            'status': " Не оплачено" if receipt.status else "Оплачено",
            'date': receipt.date.strftime('%Y-%m-%d %H:%M:%S'),
            'meter_number': receipt.meter.meter_number,
            'subsidy': receipt.subsidy,
            'privilege': receipt.privilege,
            'tariff': receipt.tariff
        } for receipt in receipts]

        return jsonify(receipts_data)
    @receipt.route('/delete-receipt/<int:receipt_id>', methods=['DELETE'])
    def delete_receipt(receipt_id):
        if 'user_id' not in session:
            return jsonify({'error': 'Not authenticated'}), 401

        # Fetch the receipt and its related meter from the database
        receipt = Receipt.query.get(receipt_id)
        if not receipt:
            return jsonify({'error': 'Receipt not found'}), 404

        # Check if the related meter belongs to the logged-in user
        if receipt.meter.user_id != session['user_id']:
            return jsonify({'error': 'Unauthorized access'}), 403

        try:
            db.session.delete(receipt)
            db.session.commit()
            return jsonify({'success': 'Receipt deleted successfully'}), 200
        except Exception as e:
            db.session.rollback()   # Roll back in case of any issues during
deletion
            return jsonify({'error': 'Failed to delete the receipt', 'message':
str(e)}), 500
```

- user.py:

```
    from flask import render_template, request, session, flash, redirect,
url_for,Blueprint
    from flask import current_app as app
```

```
from flask import Blueprint, request, jsonify, abort
from app.extensions import db
from app.models import User, Meter, Receipt
import os
from werkzeug.utils import secure_filename
from datetime import datetime
from sqlalchemy.sql import extract
from collections import defaultdict
user = Blueprint('user', __name__)
@user.route('/user_home')
def user_home():
    if 'user_id' in session:
        user_id = session['user_id']
        user = User.query.get(user_id)
        if user:
            profile_photo = user.profile_photo

            return render_template('user_home.html', user=user,
profile_photo=profile_photo)
        else:
            return "Користувача не знайдено, будь-ласка увійдіть ще раз!", 404
    else:
        return "Користувача не знайдено, будь-ласка увійдіть ще раз!", 404

@user.route('/admin_home')
def admin_home():
    if 'user_id' in session:
        user_id = session['user_id']

        user = User.query.get(user_id)
        if user:

            profile_photo = user.profile_photo
            return render_template('admin_home.html', user=user,
profile_photo=profile_photo)
        else:
            return "Користувача не знайдено, будь-ласка увійдіть ще раз!", 404
    else:
        return "Користувача не знайдено, будь-ласка увійдіть ще раз!", 404

@user.route('/get-user-info')
def get_user_info():
    if 'user_id' not in session:
        return jsonify({'error': 'User not logged in'}), 401

    user_id = session['user_id']
    current_user = User.query.get(user_id)  # Use a different variable name
instead of 'user'
    if not current_user:
        return jsonify({'error': 'User not found'}), 404

    try:
        meters = current_user.meters
        total_meters = len(meters)
        active_meters = sum(1 for meter in meters if meter.status)

        receipts_count = sum(len(meter.receipts) for meter in meters)
        paid_receipts_count = sum(1 for meter in meters for receipt in
meter.receipts if receipt.status)

        profile_photo_url = url_for('static', filename='uploads/user_icon/' +
current_user.profile_photo if current_user.profile_photo else
'uploads/user_icon/user.png')
```

```
            return jsonify({
                'username': current_user.username,
                'real_name': current_user.real_name,
                'phone': current_user.phone,
                'registration_date': current_user.registration_date.strftime("%Y-
%m-%d"),
                'profile_photo_url': profile_photo_url,
                'total_meters': total_meters,
                'active_meters': active_meters,
                'receipts_count': receipts_count,
                'paid_receipts_count': paid_receipts_count
            })
        except Exception as e:
            print(f"Error processing request: {e}")
            return jsonify({'error': 'Error processing request'}), 500



    @user.route('/delete-user', methods=['POST'])
    def delete_user():
        if 'user_id' not in session:
            return jsonify({'error': 'Not authenticated'}), 401

        user_id = session['user_id']
        user = User.query.get(user_id)
        if user:
            db.session.delete(user)
            db.session.commit()
            session.clear()
            return   jsonify({'message':   'Профіль   видалено!',   'redirect':
url_for('main.index')}), 200
        else:
            return jsonify({'error': 'User not found'}), 404
```

Моделі:

- meter.py:

```
from datetime import datetime
from app.extensions import db
class Meter(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    meter_number = db.Column(db.String(20), unique=True, nullable=False)
    meter_model = db.Column(db.String(50), nullable=False)
    tariff = db.Column(db.Float, nullable=False)
    subsidy = db.Column(db.Float, default=0)
    privilege = db.Column(db.Float, default=0)
    address = db.Column(db.String(255), nullable=False)
    status = db.Column(db.Boolean, default=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    receipts = db.relationship('Receipt', backref='meter', lazy=True)
    def __repr__(self):
        return f"<Meter {self.meter_number} Model {self.meter_model}>"
```

- receipt.py:

```
from datetime import datetime
from app.extensions import db

class Receipt(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    payment_amount = db.Column(db.Float, nullable=False)
    status = db.Column(db.Boolean, default=False)
```

```
        date = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
        meter_id     =     db.Column(db.Integer,     db.ForeignKey('meter.id'),
nullable=False)
        subsidy = db.Column(db.Float, nullable= True)
        amount_value= db.Column(db.Float, nullable= False)
        privilege = db.Column(db.Float, nullable= True)
        tariff = db.Column(db.Float, nullable=False)
        def __repr__(self):
            return f"<Receipt {self.id} for Meter {self.meter_id}>"
```

- user.py:

```
from datetime import datetime
from app.extensions import db

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    real_name = db.Column(db.String(120), nullable=False)
    phone = db.Column(db.String(20), nullable=False)
    password = db.Column(db.String(120), nullable=False)
    profile_photo      =      db.Column(db.String(120),      nullable=True,
default='user.png')
    role = db.Column(db.String(20), nullable=False, default='user')
    registration_date = db.Column(db.DateTime, default=datetime.utcnow)
    meters = db.relationship('Meter', backref='user', lazy=True)
    def __repr__(self):
        return f"<User '{self.username}'>"
```

ДОДАТОК Б

Диск з роботою