**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформаційних систем та технологій**

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**
**за освітньо-професійною програмою «Інформаційні управляючі системи та технології»**
**спеціальності 126 «Інформаційні системи та технології»**
**на тему: «Інформаційна система для підтримки процесу повірки лічильників води. Підсистема «Оператор»»**

Виконав:
студент IV курсу, групи ІС-91
Купрієнко Вілен Дем'янович

_____

Керівник:
доцент, к.т.н., доцент
Жураковська Оксана Сергіївна _____

Рецензент:
доцент кафедри ІПІ, к.т.н., доцент
Лісовиченко Олег Іванович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Факультет інформатики та обчислювальної техніки**
**Кафедра інформаційних систем та технологій**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

# ЗАВДАННЯ

**на дипломний проєкт студенту**
**Купрієнку Вілену Дем`яновичу**

1. Тема проєкту «Інформаційна система для підтримки процесу повірки лічильників води. Підсистема «Оператор»», керівник проєкту Жураковська Оксана Сергіївна, к. т. н., доцент затверджені наказом по університету від «31» травня 2023 р. №2101-с

2. Термін подання студентом проєкту: «12» червня 2023 року

3. Вихідні дані до проєкту: Технічне завдання

4. Зміст пояснювальної записки:

*1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд наявних аналогів, постановка задачі;*
*2. Інформаційне забезпечення: вхідні та вихідні дані, опис структури бази даних;*
*3. Математичне забезпечення: змістовна та математична постановка задачі, обґрунтування та опис методу розв'язання;*
*4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів;*

*5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.*

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

1. Схема діяльності;
2. Схема варіантів використань системи ;
3. Схема класів програмного забезпечення ;
4. Схема послідовностей.

7. Дата видачі завдання 13 квітня 2023 р.

Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної області | 17 квітня 2023 | |
| 2 | Формалізація функціональної моделі | 21 квітня 2023 | |
| 3 | Огляд наявних аналогів | 23 квітня 2023 | |
| 4 | Розробка й опис структури бази даних | 25 квітня 2023 | |
| 5 | Опис математичної постановки задачі | 5 травня 2023 | |
| 6 | Огляд варіантів розв'язання задачі | 8 травня 2023 | |
| 7 | Розробка й опис технічного рішення | 19 травня 2023 | |
| 8 | Тестування готового проєкту | 26 травня 2023 | |
| 9 | Опис керівництва користувача | 31 травня 2023 | |

Студент Вілен КУПРІЄНКО

Керівник Оксана ЖУРАКОВСЬКА

# АНОТАЦІЯ

Структура та обсяг роботи. Дипломний проєкт на тему «Інформаційна система для підтримки процесу повірки лічильників води. Підсистема «Оператор»» складається з п’яти розділів, містить 12 рисунків, 10 таблиць, 5 додатків, 16 джерел.

Дипломний проєкт  присвячений розробці комплексу задач, що стосуються розробки користувацької частини процесу повірки лічильників.

У розділі інформаційного забезпечення описано вхідні та вихідні дані, розроблена схема таблиць бази даних.

Розділ математичного забезпечення присвячений формулюванню математичної задачі та огляду наявних методів її розв’язання.

У розділі програмного забезпечення описуються засоби розробки програмного забезпечення, обґрунтовуються обрані архітектурні рішення та розробляються специфікації функцій.

Технологічний розділ визначає мету проведення випробувань програмного продукту та описує їх результати.

Інформаційна система, оператор повірки лічильників, повірка лічильників води.

# ABSTRACT

Structure and scope of the work. The diploma project titled "Information System for Supporting the Water Meter Verification Process. Operator Subsystem" consists of five sections and includes 12 figures, 10 tables, 5 appendices, 16 sources.

The diploma project is dedicated to the development of a set of tasks related to the user part of the water meter verification process.

The information provision section describes the input and output data and presents the database table schema.

The mathematical provision section focuses on formulating the mathematical problem and reviewing existing methods for its solution.

The software provision section describes the software development tools, justifies the chosen architectural solutions, and develops functional specifications.

The technological section defines the purpose of conducting tests for the software product and describes their results.

Informational system, water meter verification operator, verification of water meter

| Номер рядка | Формат | Позначення | Найменування | Кільк. аркушів | Номер екзем. | Примітка |
|---|---|---|---|---|---|---|
| 1 | | | Документація загальна | | | |
| 2 | | | | | | |
| 3 | | | Знову зроблена | | | |
| 4 | | | | | | |
| 5 | А4 | ІС91.130БАК.004 ПЗ | Пояснювальна записка | 60 | | |
| 6 | | | | | | |
| 7 | А3 | ІС91.130БАК.004 Д1 | Схема діяльності | 1 | | |
| 8 | | | | | | |
| 9 | А3 | ІС91.130БАК.004 Д2 | Схема варіантів | 1 | | |
| 10 | | | використання системи | | | |
| 11 | | | | | | |
| 12 | А3 | ІС91.130БАК.004 Д3 | Схема потоку даних | 1 | | |
| 13 | | | | | | |
| 14 | А3 | ІС91.130БАК.004 Д4 | Схема послідовностей | 1 | | |
| 15 | | | | | | |
| 16 | | | | | | |
| 17 | | | | | | |
| 18 | | | | | | |
| 19 | | | | | | |
| 20 | | | | | | |
| 21 | | | | | | |
| 22 | | | | | | |
| 23 | | | | | | |
| 24 | | | | | | |
| 25 | | | | | | |
| 26 | | | | | | |
| 27 | | | | | | |
| 28 | | | | | | |

| | | | | | ІС91.130БАК.004 ТП | | |
|---|---|---|---|---|---|---|---|
| Зм. | Аркуш | № докум. | Підпис | Дата | | | |
| Розроб. | | Купрієнко В. Д. | | | «Інформаційна система для підтримки процесу повірки лічильників води. Підсистема «Оператор»» Відомість проєкту | Літ. | Аркуш 1 / Аркушів 1 |
| Керівн. | | Жураковська О.С. | | | | Т | |
| | | | | | | КПІ ім. Ігоря Сікорського Група ІС-91 | |
| | | | | | | | |
| Затв. | | | | | | | |

# Пояснювальна записка
# до дипломного проєкту
# на тему: «Інформаційна система для підтримки процесу повірки лічильників води. Підсистема «Оператор»»

Київ – 2023 року

ЗМІСТ

# ВСТУП

У сучасному світі інформація – один з найцінніших активів, яким можна володіти. Диджиталізація постійно розвивається і призводить до проникнення розумних систем в усі аспекти життя суспільства. Це й призводить до неодмінного збільшення обсягу інформації, яку потрібно обробляти та зберігати. Універсальні хмарні сервіси великих техно корпорацій надають усі спектри послуг, що можуть нам знадобитись. Однак, із ростом важливості зберігання та обробки інформації з'являється також проблема специфікації.

Професійна інформація, особливо в сенситивних галузях, вимагає особливого підходу до зберігання. Такі дані не можуть бути просто передані на загальнодоступні платформи, такі як «Google Drive», «Microsoft OneDrive» тощо.

Замість цього, компанії все частіше створюють власні рішення, які адаптовані до вирішення конкретних завдань, забезпечують зручний інтерфейс і спрощують процес виконання роботи певними людьми.

У контексті даної дипломної роботи розглядається сфера комунальних послуг, зокрема, повірка лічильників води. Водопостачальні компанії встановлюють лічильники води в житлових та комерційних будівлях, щоб контролювати споживання води. Однак, з часом лічильники можуть втрачати свою точність та потребувати обслуговування. Для ефективної та точної роботи необхідно вести систематичний моніторинг та перевірку цих лічильників.

Традиційний підхід до повірки лічильників залежить від кваліфікації операторів та їхньої здатності правильно інтерпретувати дані. Це може призвести до помилок та неправильних вимірювань, що впливає на надійність та точність інформації.

З метою покращення цього процесу та надійності вимірювання пропонується інформаційна система для підтримки процесу повірки лічильників води. Основна увага в дипломному проєкті зосереджена на розробці та реалізації підсистеми «Оператор», яка має на меті автоматизувати робочі процеси операторів та забезпечити їм ефективні інструменти для перевірки. Більш того,

система забезпечить зручність роботи з даними для середньостатистичного користувача.

У ході дослідження та розробки підсистеми «Оператор» ми використовували сучасні методи та технології, такі як бази даних, веброзробка та алгоритми оптимізації. Крім того, ми провели аналіз сучасних інформаційних систем водопостачальних компаній та врахували найкращі практики в цій галузі.

Після розробки системи ми провели випробування та аналіз ефективності. З впевненістю можемо сказати, що система має високу продуктивність та надійність.

Дипломний проєкт має на меті розв'язання глобальної проблеми - забезпечення раціонального використання водних ресурсів та коштів. Для досягнення цієї мети використані сучасні методи програмування, надійна база даних, а також механізми для забезпечення безпеки даних.

# 1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

## 1.1 Опис предметного середовища

Оператори працюють зі спеціалізованим обладнанням, яке використовується для прогону точного об’єму води крізь водомір та визначення його показників вимірювання. В результаті вони збирають значний обсяг даних про лічильник, про середовище його роботи та числові параметри вимірювань. Ці дані мають бути точно зафіксовані, збережені та доступні для подальшого аналізу на наступних етапах роботи глобальної системи повірки.

У подальшій роботі оператор не відіграє важливої ролі, а лише очікує на виконання роботи метрологом – спеціалістом з розрахунків. Метролог може як затвердити дані збережені оператором, так і змінити їх на підставі невідповідності даних відповідно до фотофіксації, яка також проводиться оператором під час експертизи.

Існує можливість, що дані були неправильно заповнені, або втрачені. В такому випадку метролог відхиляє протокол, тим самим змушуючи оператора перевірити усі дані та, у крайньому разі, провести повторну експертизу.

### 1.1.1 Опис процесу діяльності

Оскільки загальна система повірки є закритою, для надійності та правдивості даних повірки будь-який працівник має для початку авторизуватись у системі. Для отримання доступу до функціонала системи, оператор повинен увійти в систему ввівши свої персональні дані аутентифікації: логін, пароль.

Після успішного входу в систему, оператор отримує доступ до власної підсистеми без доступу до даних інших етапів обробки.

Оператор має можливість виконати наступні дії:

– створити протокол для внесення даних експертизи;

– заповнити протокол даними що були отримані в наслідок проведення експертизи;

– зберегти протокол у системі;

– переглянути створені оператором протоколи (варто зазначити, що оператор не може бачити створені іншим оператором протоколи).

Узагальнимо інформацію на діаграмі діяльності, яка наведена в графічних матеріалах (ІС91.130БАК.004 Д1).

### 1.1.2 Опис функціональної моделі

Оператор в межах системи здатний виконувати наступні функції:

– вхід та вихід з системи: система при вході оператора в неї має надавати усі дані до яких він має доступ і, відповідно, закрити доступ коли відбувся вихід;

– створити протокол: оператор має змогу створити протокол, після чого система надасть форму для заповнення даних оператору;

– заповнити протокол: система надає змогу вносити текстові та числові дані також завантажувати фотофіксації;

– перегляд попередніх експертиз: оператор має можливість переглянути свої попередні експертизи, що були завантажені в систему раніше без права зміни значень, окрім випадку коли протокол, попередньо, був відхилений метрологом.

Побудуємо діаграму варіантів використання зображеній у графічних матеріалах (ІС91.130БАК.004 Д2).

## 1.2 Огляд наявних аналогів

У рамках огляду наявних аналогів розглядаються наявні рішення та системи, які використовуються для підтримки процесу повірки лічильників води. Дослідження наявних аналогів допомагає з'ясувати, які функції та можливості уже існують на ринку, а також визначити переваги та недоліки кожного з них.

На основі проведеного аналізу можна виділити конкретний список функцій, необхідних для нашої системи, аби вона стала конкурентоспроможною.

Нижче наведені приклади наявних аналогів, які можуть бути використані для підтримки процесу повірки лічильників води.

### 1.2.1 Огляд аналога «Neptune Technology Group»

«Neptune Technology Group» надає інформаційні системи для керування водними ресурсами, включаючи системи для повірки лічильників води. Головна сторінка програми-аналога зображена на рисунку 1.1.

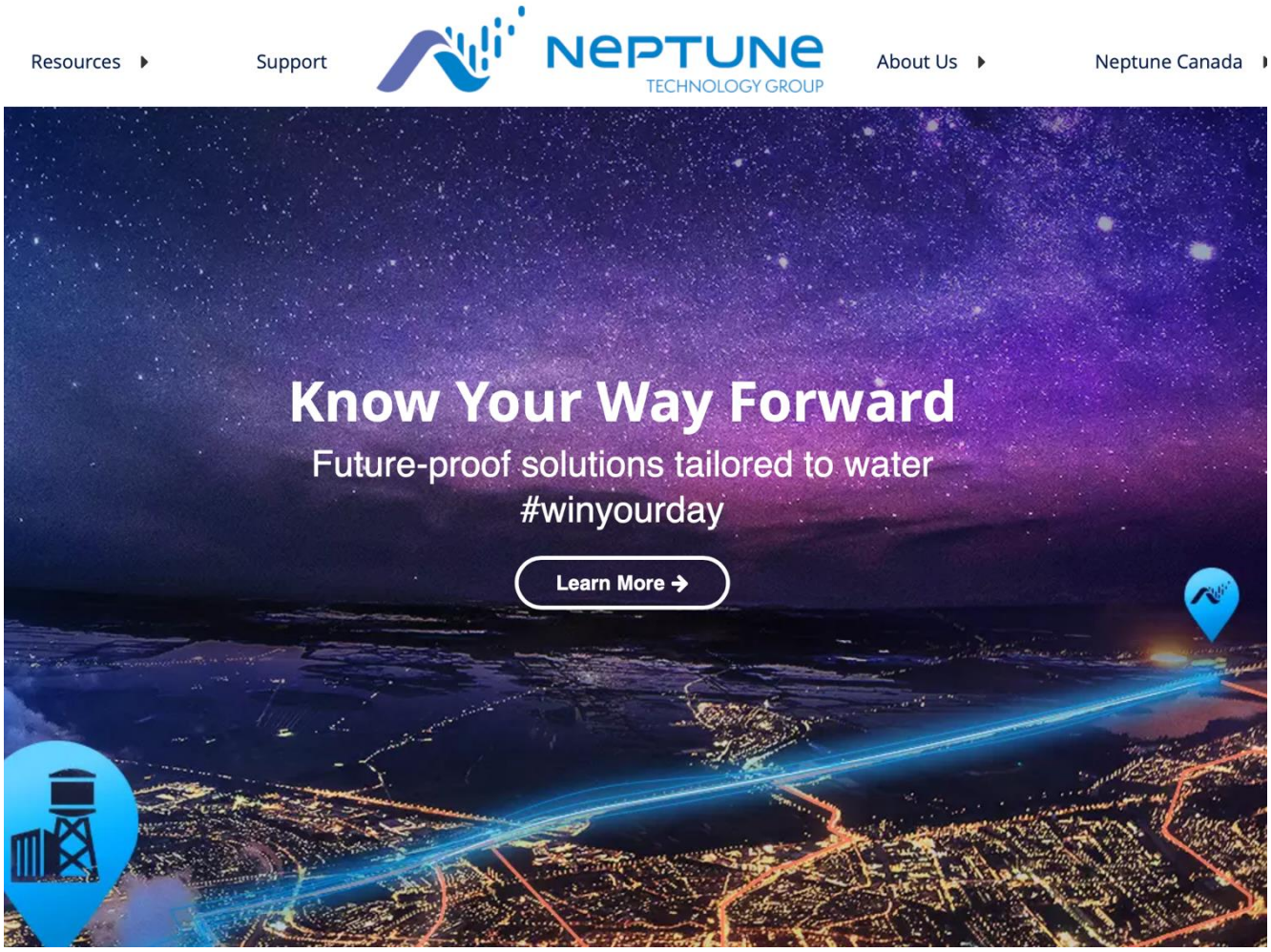


Рисунок 1.1 – Програма-аналог «Neptune Technology Group»

Переваги:

– широкий спектр продуктів і послуг для управління водними ресурсами;

– висока точність лічильників води та надійність системи повірки;

– інтеграція з іншими системами управління.

Недоліки:

– вища вартість в порівнянні з деякими іншими аналогами;

– обмежена доступність на деяких ринках.

### 1.2.2 Огляд аналога – «Kamstrup»

«Kamstrup» є постачальником рішень для управління водними ресурсами, включаючи системи для повірки лічильників води. Головна сторінка програми-аналога зображена на рисунку 1.2.

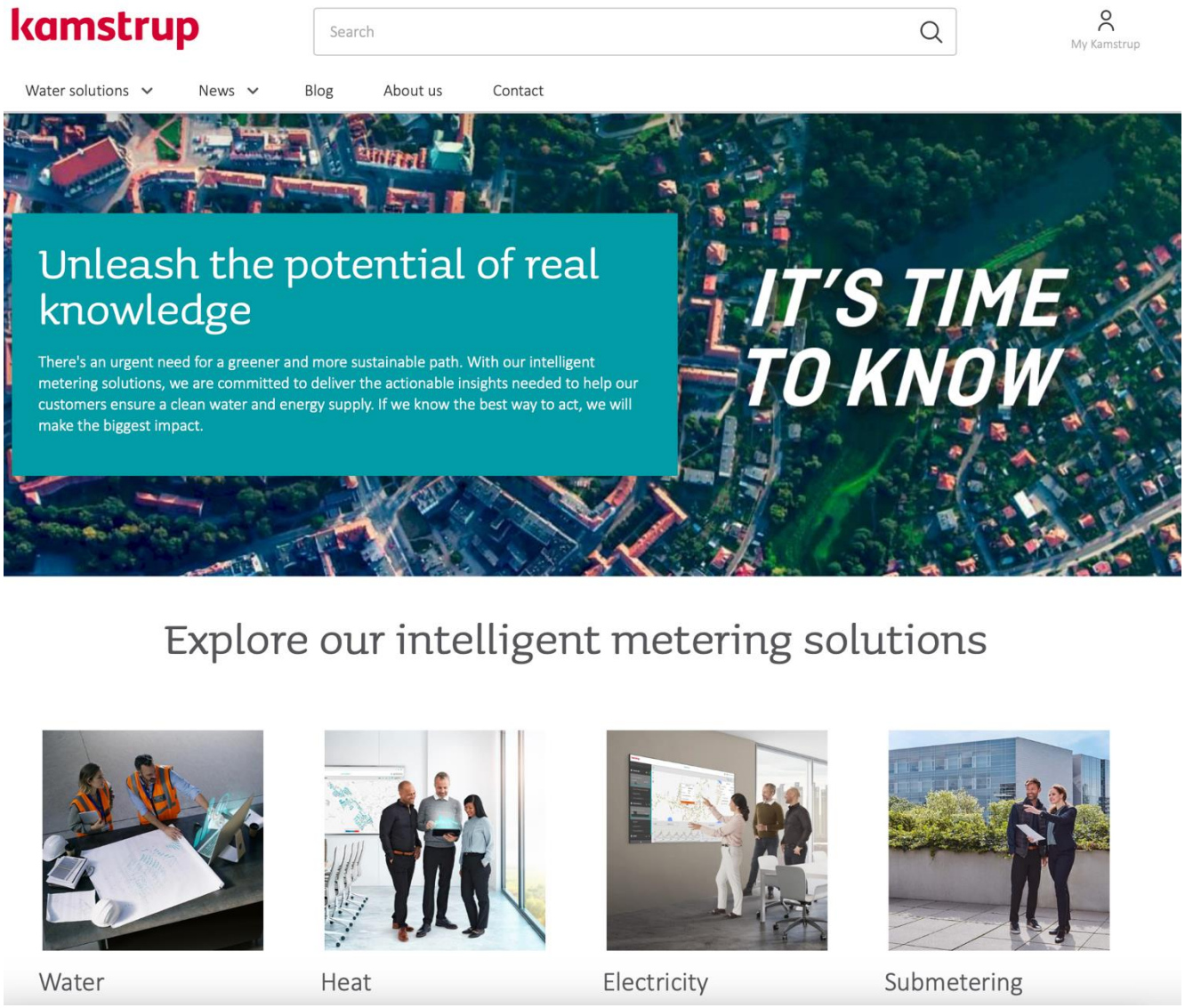


Рисунок 1.2 – Програма-аналог «Kamstrup»

Переваги:

– висока точність лічильників води та надійність системи повірки;
– інтеграція з розумними мережами та іншими технологіями;
– зручний інтерфейс користувача та високий рівень підтримки клієнтів.

Недоліки:

– вища вартість в порівнянні з деякими іншими аналогами;

– обмежена географічна наявність.

### 1.2.3 Огляд аналога «Itron»

«Itron» є світовим лідером у сфері розумних мереж та рішень для повірки лічильників води. Головна сторінка програми-аналога зображена на рисунку 1.3.

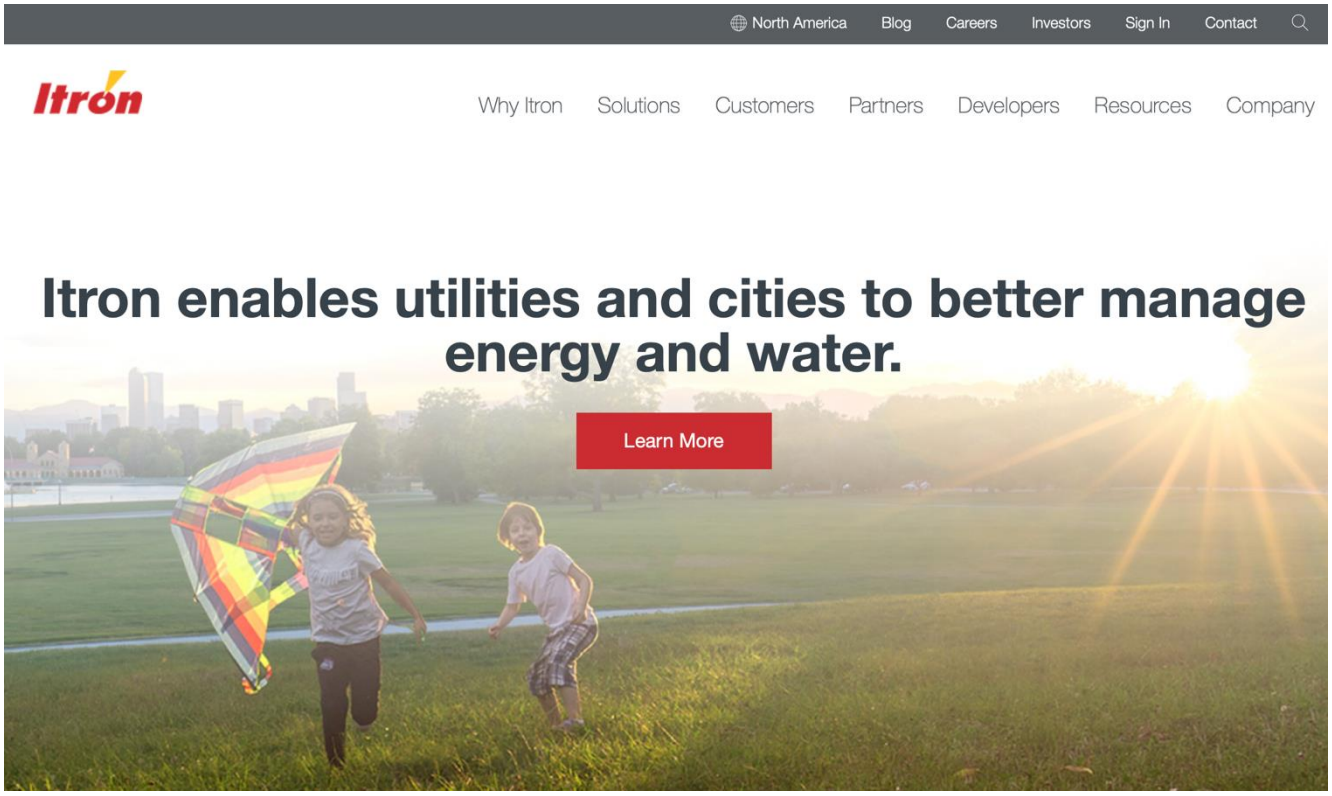


Рисунок 1.3 – Програма-аналог «Itron»

Переваги:

– широкий діапазон продуктів та рішень для управління водними ресурсами;

– застосування розумних технологій та аналітики даних;

– гнучкість та масштабованість системи для великих мереж лічильників води.

Недоліки:

– вища вартість в порівнянні з деякими іншими аналогами;

– складність налаштування та інтеграції з наявними системами.

### 1.2.4 Огляд аналога – «Badger Meter»

«Badger Meter» спеціалізується на розробці та виробництві лічильників води та систем для їх повірки. Головна сторінка програми-аналога зображена на рисунку 1.4.

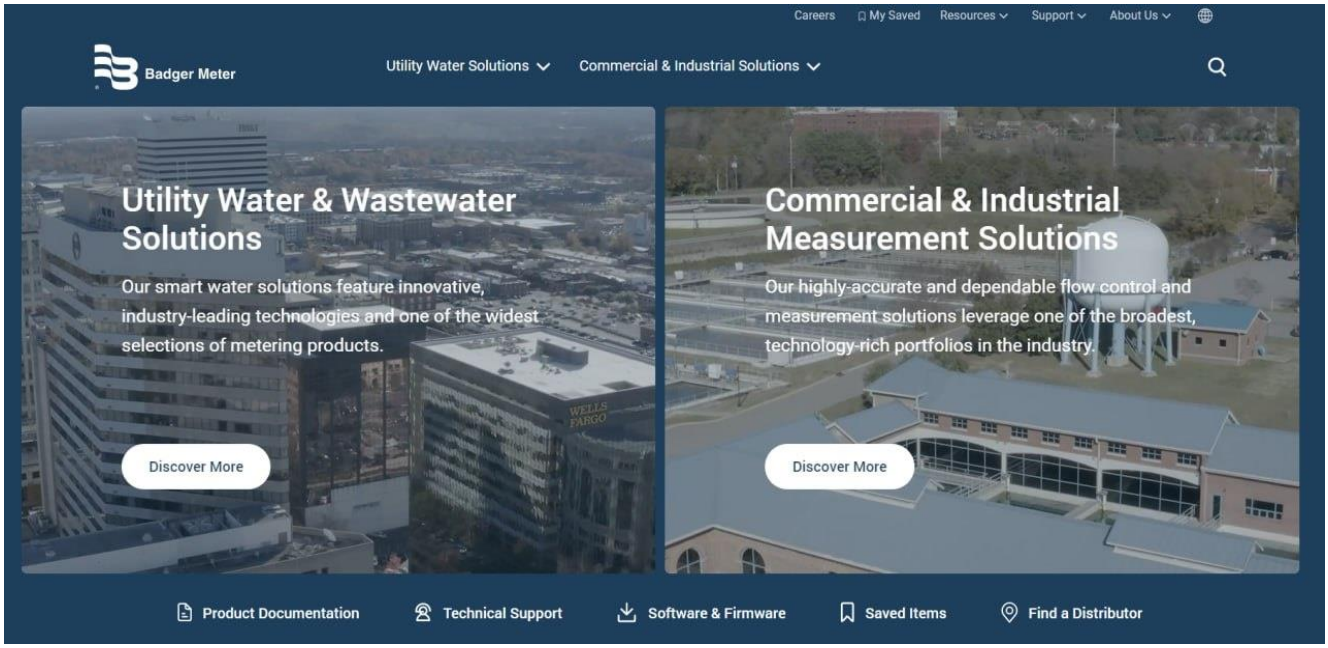


Рисунок 1.4 – Програма-аналог «Badger Meter»

Переваги:

- широкий асортимент лічильників води та додаткових послуг;
- довгий термін служби та висока точність продукції;
- гнучкість в роботі з різними типами водних систем.

Недоліки:

- обмежена функціональність у порівнянні з деякими конкурентами;
- вища вартість в деяких категоріях продуктів.

## 1.3 Постановка задачі

### 1.3.1 Призначення розробки

Призначенням розробки є створення системи підтримки роботи оператора та системи збереження даних отриманих оператором в ході експертизи.

### 1.3.2 Цілі та задачі розробки

Метою розробки є підвищення ефективності роботи оператора повірки лічильників холодної води шляхом спрощення процесу передачі даних повірок та забезпечення зручного доступу до цих даних усім учасникам процесу.

Завдання, необхідні для досягнення мети розробки:

– забезпечити зручну авторизацію оператора в системі за умови що глобальна система захисту має певний контроль аутентифікації;

– створення сутності «Оператор» як актора зі своїми особливостями та функціями, розробка схеми зберігання даних в базі даних;

– розробка зручного механізму завантаження даних;

– проєктування бази даних для зберігання інформації з експертиз;

– створення графічного інтерфейсу:

1) вікно авторизації;
2) вікно для перегляду протоколів;
3) вікно для створення протоколу повірки;
4) вікно для зміни пароля.

Висновок до розділу

У даному розділі було описано цілі та завдання розробки, які спрямовані на досягнення зручності, ефективності та точності взаємодії операторів з інформаційною системою для підтримки процесу повірки лічильників води. Підсистема «Оператор» має на меті покращення продуктивності роботи операторів, забезпечення точності та надійності даних, зручного доступу до попередніх експертиз та забезпечення безпеки даних.

Розробка підсистеми «Оператор» відіграє важливу роль у впровадженні інформаційної системи для підтримки процесу повірки лічильників води. Вона допомагає операторам ефективно виконувати свої обов'язки, забезпечуючи зручність та надійність завантаження та зберігання даних. Завдяки цій підсистемі оператори можуть легко оперувати даними доступними після проведення експертизи, що сприяє підвищенню ефективності та якості виконання своїх обов’язків метрологом.

Загалом, підсистема «Оператор» є необхідним компонентом інформаційної системи для підтримки повірки лічильників води, яка сприяє поліпшенню процесу повірки, забезпеченню точності та ефективності роботи операторів і підвищенню якості обслуговування користувачів.

# 2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

## 2.1 Вхідні дані

Як вже зазначалось, після проведення експертизи оператор отримує великий набір даних. Ці дані можуть бути подані у документі, на передньому плані лічильника або на дисплеї повірочної установки.

Ці дані можна розділити на три окремі складові:

1) інформація про водяний лічильник:
   - стандарт виготовлення;
   - номер;
   - тип;
   - рік виробництва;
   - технічна характеристика лічильника:
     а) клас;
     б) нормований потік;
     в) додаткова інформація.
   - об’єм;
   - температура води.
2) результати експертизи:
   - об’єм пропущений установкою (еталонний);
   - об’єм виміряний лічильник.
3) оточення:
   - вологість повітря;
   - температура повітря.

До кожного лічильника та тесту в експертизі мають додаватись фотофіксації для підтвердження даних.

## 2.2 Вихідні дані

Вихідними даними виступають заповнені форми протоколів, що зберігаються у базі даних.

## 2.3 Опис структури бази даних

Для визначення структури бази даних потрібно визначити яку інформацію та в якому вигляді ми маємо зберігати, які сутності можуть у ній існувати.

Розглянемо вимоги до даних що мають бути збережені:

– дані експертизи: обов’язкові дані без яких повірка не може бути виконана;

– характеристика лічильника: опис параметрів лічильника дає змогу чітко визначити особливості та підібрати якісні тести, що зможуть вказати на придатність лічильника;

– метадані: важливо знати коли та яким оператором був створений протокол, коли протокол перевірив метролог та, у випадку зміни метрологом якихось даних, дату зміни протоколу;

– статус протоколу: мітка що відображає на якому етапі знаходиться протокол.

Відповідно, маємо наступні сутності, що мають існувати в нашій базі даних:

– водяний лічильник (Water Meter);

– протокол (Verification Record);

– оператор (Operator);

– метролог (Metrologist).

У графічних матеріалах наведена діаграма потоку даних (ІС91.130БАК.004 Д3).

Дана діаграма дає змогу нам побачити що сутність «watermeter» є залежною до «verificationrecord», а сутності оператора та метролога є окремими, та лише взаємодіють з протоколом.

Колекція «WaterMeter» подано в таблиці 2.1

Таблиця 2.1 — Колекція WaterMeter

| WaterMeter | |
|---|---|
| name | type |
| _id | ObjectId Unique |
| technicalDescription | String |
| standardType | String (enum: ДСТУ_3580, etc.) |
| meterNumber | String Unique |
| type | String |
| manufactureYear | Number |
| volume | Number |
| alteredBy | ObjectId Unique ref(metrologists) |
| createdAt | Date |

Колекція «VerificationRecord» подано в таблиці 2.2.

Таблиця 2.2 — Колекція VerificationRecord

| VerificationRecord | |
|---|---|
| name | type |
| _id | ObjectId Unique |
| waterMeterId | ObjectId ref(water_meters) |
| photos | Array of paths to photos |
| environment | Structure<br>waterTemperature: Number<br>airTemperature: Number<br>airHumidity: Number |
| status | String (enum: pending, approved, declined) |

| | |
|---|---|
| tests | Structure<br>consumption: Number<br>permissibleError: Number<br>standardVolume: Number<br>initialValue: Number<br>finalValue: Number<br>meterVolume: Number<br>duration: Number<br>averageConsumption:Number<br>withinRange: Boolean<br>actualError: Number<br>isSuitable: Boolean |
| alteredBy | ObjectId optional |
| createdBy | ObjectId |
| createdAt | Date |
| alteredAt | Date optional |
| statusAlteredBy | ObjectId Optional ref(metrologists) |

Висновок до розділу

У даному розділі були описані вхідні та вихідні дані, які використовуються в системі для підтримки процесу повірки лічильників води з погляду оператора. Вхідні дані включають необхідну інформацію про лічильники води, що підлягають повірці, а також форму для збору даних під час експертизи. Вихідні дані представлені у вигляді заповненої форми з усіма деталями тестування та загальної інформації про лічильник.

Опис структури бази даних включає перелік вихідних документів, які формуються в результаті експертизи.

Розробка структури бази даних, а також визначення вхідних та вихідних даних є важливим кроком у забезпеченні ефективності та точності процесу повірки лічильників води. Чітко визначені вхідні дані дозволяють збирати

необхідну інформацію для експертизи, а вихідні дані забезпечують передачу результатів повірок та інших важливих даних. Структура бази даних дозволяє організувати зберігання та обробку цих даних зручним та ефективним способом.

# 3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

Розділ з математичного забезпечення присвячений розв'язанню задачі формування розкладу роботи операторів.

## 3.1 Змістовна постановка задачі

Завдання полягає в плануванні робочого дня операторського штату за умови що ми вже маємо усі заявки що плануються на обробку цього дня. Кожний оператор має свою власну продуктивність - час, необхідний для обробки однієї заявки, і вартість - грошову винагороду, яку він отримує за день роботи.

Для стимулювання якісного та швидкого виконання повірок операторами організація вирішила створити рейтинг на основі якого буде видаватись заробітна плата за робочий день.

Рейтинг буде враховувати відсоток кількості правильно заповнених протоколів до загальної кількості опрацьованих заявок, стаж, можливо, посада. На основі цього рейтингу стандартна денна заробітна плата буде змінюватись.

Варто також врахувати, що кожна заявка має бути оброблена протягом дня, загальна вартість роботи всіх операторів не повинна перевищувати заданий бюджет S, середній час обробки заявок повинен бути якомога меншим, щоб забезпечити високу швидкість реакції на заявки клієнтів.

## 3.2 Математична постановка задачі

Задачу можна сформулювати як задачу дискретного програмування з додатковими обмеженнями.

Спочатку визначимо деякі змінні та параметри:

- $N$ - кількість заявок;
- $M$ - кількість операторів;
- $t_i$ - час обробки однієї заявки і-м оператором ($i$ = 1, 2, ..., M);
- $s_i$ - вартість одного дня роботи і-го оператора ($i$ = 1, 2, ..., M);
- $x_i$- вектор розмірності $N$, де $x_{i,j} = \{0, 1\}$ і позначають чи оброблена заявка $j$ $i$-м оператором;

- $y_i$ - {0, 1} , позначає чи залучений $i$-й оператор;
- $S$ - максимальний бюджет на день;
- $P$ - максимальний активний обсяг персоналу на день.

Цільова функція:

$$\frac{\sum_{i=1}^{M} x_i * t_i}{N} \to min \tag{3.1}$$

Обмеження:

1. Всі заявки повинні бути оброблені:

$$\sum_{j=1}^{N}[\sum_{i=1}^{M} x_{i,j}] = N \tag{3.2}$$

2. Загальна вартість роботи всіх операторів не повинна перевищувати заданий бюджет:

$$\sum_{\mathrm{i}=1}^{M} y_i * s_i \leq S \tag{3.3}$$

3. Кожна заявка має бути оброблена лише одним оператором.

$$\sum_{i=1}^{M} x_{i,j} = 1,\ \mathrm{j} = \overline{1, N} \tag{3.4}$$

4. Максимальна кількість операторів за день:

$$\sum_{\mathrm{i}=1}^{M} y_i \leq P \tag{3.5}$$

5. Оператор може обробляти заявки лише якщо він залучений:

$$\forall \mathrm{j}, i x_{i,j} \leq y_i \tag{3.6}$$

## 3.3 Обґрунтування методу розв’язання

Розглянемо алгоритми динамічного програмування та евристичний жадібний алгоритм.

### 3.3.1 Динамічне програмування

Динамічне програмування - це метод розбиття великої задачі на менші підзадачі та збереження розв'язків цих підзадач для використання в майбутніх обчисленнях.

Такий підхід може бути ефективним для розв'язання нашої задачі, але його складність вимагає значних обчислювальних ресурсів, особливо для великих значень *N* та *M*.

Задача динамічного програмування вимагає, щоб ми визначили стан системи та перехід між станами. В контексті нашої задачі, ми можемо визначити стан як поточне виділення заявок та операторів, а перехід - це призначення заявки наступному оператору.

Ось базові кроки, які можна використати для розв'язання цієї задачі за допомогою динамічного програмування:

1) визначимо масив *DP*[*i*][*j*][*k*], де *i* – кількість залучених операторів, *j* – кількість оброблених заявок, а *k* – поточний бюджет. Кожний стан *DP*[*i*][*j*][*k*] представляє мінімальний середній час обробки заявки, якщо ми залучили *i* операторів, обробили *j* заявок та витратили *k* бюджету;

2) ініціалізація: *DP*[0][0][0] = 0, оскільки немає операторів, заявок і витраченого бюджету;

3) пройдемося по *DP*[*i*][*j*][*k*] та оновимо
*DP*[*i*+1][*j*+l][*k*+*s*[*i*]] = *min*(*DP*[*i*+1][*j*+l][*k*+*s*[*i*]], *DP*[*i*][*j*][*k*] + *t*[i]);

4) Знайдемо мінімальне значення в *DP*[M][N][k] для всіх *k* від 0 до *S*. Це буде нашим оптимальним розв’язком.

### 3.3.2 Жадібний алгоритм

Жадібний алгоритм - це метод, при якому ми обираємо найкращу можливість на кожному етапі. Він швидкий, легкий в реалізації та зазвичай потребує мало обчислювальних ресурсів, але він може не надавати оптимальне рішення для деяких задач.

Для нашої задачі, жадібний алгоритм може містити вибір оператора з найменшим часом обробки для кожної заявки, поки це не перевищує бюджет. Цей процес продовжується, поки всі заявки не будуть розподілені або до тих пір, поки не буде вичерпано бюджет.

Побудуємо наступну «жадібну» стратегію:

1) Обчислюємо ефективність кожного оператора як відношення кількості заявок, які оператор може обробити за день, до вартості цього оператора за день $E[i] = (8 / t[i]) / s[i]$ для всіх $i = \overline{1, M}$. Ефективність тут розуміється як виконання заявки за одиницю грошей;

2) Сортуємо операторів в порядку зростання їх ефективності;

3) Починаємо розподіляти заявки між операторами починаючи з найбільш ефективного, доки не вичерпаємо бюджет або не обробимо всі заявки.

Порівнюючи ці два алгоритми, ми можемо побачити, що кожен з них має свої переваги та недоліки.

Алгоритм динамічного програмування здатний знайти глобальне оптимальне рішення, але він вимагає більше обчислювальних ресурсів і часу. Він також вимагає складної логіки для його реалізації.

Жадібний алгоритм є простішим у реалізації, вимагає менше часу та обчислювальних ресурсів, але він може не завжди знаходити глобальний оптимум. Він може знайти лише локальне оптимальне рішення, особливо в ситуаціях, коли вартості та часи обробки заявок операторів сильно відрізняються.

Враховуючи, що ми плануємо розклад на один день, можемо стверджувати, що вибір алгоритму напряму буде залежати від обсягу заявок. Враховуючи обмеженість в часі планування розкладу на наступний день, ми можемо обрати жадібний алгоритм, у випадку, коли надійшла велика кількість заявок. У

протилежному випадку, нам може вистачити часу для того, щоб знайти оптимальний розподіл для операторів за допомогою динамічного програмування.

Оскільки завдання розподілу не є основним в організації повірок, то доцільно обрати жадібний алгоритм як швидкий та зручний спосіб розподілу операторів.

### 3.4 Опис методів розв'язання

Нехай ми маємо $N = 30$ заявок і $M = 5$ операторів та бюджет на день 2000(грн), враховуючи усі обмеження та дані про час обробки та вартість роботи операторів побудуємо таблицю 3.1 з детальною інформацією про операторів та їх ефективність.

Таблиця 3.1 — Характеристики операторів

| Оператор | Час обробки однієї заявки (години) | Вартість за день (грн) | Кількість заявок, які можна обробити за день | Ефективність |
|---|---|---|---|---|
| 1 | 1 | 400 | 8 | 0.02 |
| 2 | 2 | 350 | 4 | 0.0114 |
| 3 | 0.5 | 500 | 16 | 0.032 |
| 4 | 2 | 700 | 4 | 0.0057 |
| 5 | 1.5 | 600 | 5.33 | 0.0089 |

Згідно з цими даними, оператори можна розташувати за ефективністю в такому порядку: 3, 1, 5, 2, 4. Почнемо розподіляти заявки між ними, дотримуючись вказаного порядку.

1) оператор 3 (найефективніший) отримує 16 заявок. В бюджеті залишилось – 1500 грн, заявок лишилось – 14;

2) оператор 1 отримує 8 заявок. В бюджеті залишилось – 1100 грн, заявок лишилось – 6;

3) оператор 5 отримує 5 заявок. В бюджеті залишилось – 500 грн, заявок лишилось – 1;

4) оператор 2 отримує 1 заявку. В бюджеті залишилось – 150 грн, заявок лишилось – 0;

5) оператор 4 не може отримати заявки, оскільки в бюджеті недостатньо коштів для його найму, але й усі заявки вже розподілені.

Таким чином, всі заявки були розподілені між операторами з урахуванням максимально можливої ефективності в рамках наявного бюджету.

Висновок до розділу

Була розглянута задача дискретного програмування. Здійснено математичну постановку задачі, проаналізовано методи її розв’язання.

Алгоритми для розв'язання задачі розподілу заявок між операторами: алгоритм динамічного програмування та жадібний алгоритм.

Алгоритм динамічного програмування виявився більш точним у пошуку глобально оптимального рішення. Проте, його реалізація є складнішою, а також вимагає значних обчислювальних ресурсів і часу, що може бути критичним для великих обсягів даних.

Жадібний алгоритм, навпаки, характеризується швидкістю і простотою реалізації, з меншими вимогами до обчислювальних ресурсів. Втім, він може не завжди гарантувати знаходження глобально оптимального рішення.

Враховуючи специфіку задачі, де важливою є швидкість планування розкладу на короткотермінову перспективу (один день) і при цьому розподіл заявок не є основним аспектом в організації повірок, було вирішено, що оптимальним вибором для цієї задачі є жадібний алгоритм. Його використання дозволить забезпечити швидкий та зручний спосіб розподілу заявок серед операторів з адекватною точністю рішень та економією ресурсів, наведено приклад побудови розкладу.

# 4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

## 4.1 Засоби розробки

У даному розділі будуть розглянуті основні засоби розробки програмного забезпечення, які були використані під час розробки проєкту.

Серверна частина програмного забезпечення була реалізована з використанням мови програмування «TypeScript» та платформи «Node.js». «TypeScript» є розширенням мови «JavaScript», яке додає підтримку сучасних функціональних можливостей та покращену інтеграцію з інструментами розробки. «Node.js» є виконавчим середовищем для виконання «JavaScript» на сервері, що дозволяє розробникам створювати потужні та масштабовані серверні додатки.

Одним із ключових компонентів серверної частини є фреймворк «NestJS». «NestJS» - це модульний фреймворк для побудови ефективних та масштабованих вебдодатків з використанням «TypeScript». Він надає розробникам зручність та швидкість розробки, завдяки своїй структурі на основі модулів, провайдерів та контролерів. «NestJS» також підтримує використання різних шаблонів проєктування, таких як «MVC» («Model-View-Controller»), що сприяє організації та структуруванню коду.

Для хешування паролів використовувався алгоритм «argon2». «Argon2» є одним із найбільш безпечних алгоритмів хешування паролів, що забезпечує високий рівень безпеки шляхом витрати багато ресурсів обчислення.

Для забезпечення зручної та продуктивної взаємодії з базою даних була використана бібліотека «Mongoose». «Mongoose» - це об'єктно-документна моделювальна бібліотека для «Node.js», яка дозволяє зручно взаємодіяти з базою даних «MongoDB». Вона надає зручний API для визначення схем даних, виконання запитів та взаємодії з документами бази даних «MongoDB».

Для забезпечення декларативності була використана бібліотека «class-transformer». Це інструмент, який дозволяє легко і зручно перетворювати об'єкти одного класу в об'єкти іншого класу в мовах програмування «TypeScript» або

«JavaScript». «Class-transformer» забезпечує потужні механізми автоматичного перетворення об'єктів. Наприклад, валідацію, серіалізацію, десеріалізацію, копіювання та багато інших.

Бібліотека «class-transformer» в нашій системі була дуже корисною для розробки підсистеми «Оператор», який взаємодіє з об'єктами даних.

Основні можливості, які надала бібліотека:

– набір декораторів, які можна додати до класів та їх властивостей для визначення різних метаданих. Наприклад, декоратор «@Expose» дозволяє вказати, які властивості класу мають бути викладені під час серіалізації або десеріалізації;

– «class-transformer» надає простий спосіб перетворення об'єктів з одного класу в інший та навпаки. Ми використовували декоратори, щоб вказати, які властивості мають бути включені або виключені під час цього процесу;

– дозволила групувати властивості класу та працювати з ними як з окремими наборами. Це дозволяє проводити операції серіалізації, десеріалізації та валідації тільки для певних властивостей;

– дозволила використати декоратор «@Transform()» для створення функцій-трансформаторів, які застосовуються під час серіалізації та десеріалізації. Ці процеси були корисними для форматування даних та числових значень;

– дала можливість для поверхневого та глибокого клонування об'єктів. Це полегшило створити копії об'єктів з ідентичними значеннями властивостей;

– підтримувати використання декораторів для роботи з реляційними відношеннями між об'єктами. Усе для того, аби вказати взаємозв'язки типу один до одного, один до багатьох та багато до багатьох;

– «class-transformer» добре працює з наслідуванням класів. Завдяки цьому ми змогли виставити декоратори до базового класу і вони автоматично успадковували його підкласи.

Бібліотека «class-transformer» легко інтегрується з іншими бібліотеками, такими як «Mongoose» або «NestJS». Це дозволило нам використовувати всі

можливості «class-transformer» разом з іншими потужними інструментами для розробки інформаційної системи.

Бібліотека «class-validator» є корисним інструментом для розробки валідації даних в нашій системі. Вона дозволяє легко валідувати та перевіряти дані, які вводяться операторами системи, забезпечуючи їхню правильність та відповідність певним критеріям.

Основні можливості, які надала бібліотека «class-validator»:

– валідація введених даних, які вводять оператори системи при заповненні протоколів повірки лічильників води. Наприклад, валідувати числові значення, щоб переконатися, що вони входять в певний діапазон, або перевіряти, щоб рядки мали певну довжину чи формат, наприклад, для ідентифікаційних номерів лічильників;

– використання вбудованих валідаторів. Бібліотека надає широкий набір вбудованих валідаторів, які можна використовувати для різних типів даних. Наприклад, можна використовувати «IsNotEmpty» для перевірки на наявність значення, «IsNumber» для перевірки на числовий тип, або «IsEmail» для перевірки формату електронної пошти;

– створення власних валідаторів. Існує можливість також створювати власні валідатори для валідації специфічних полів чи правил інформаційної системи. Наприклад, можливо створити валідатор, який перевіряє унікальність ідентифікаційних номерів лічильників в базі даних;

– комплексна валідація. Бібліотека «class-validator» дозволяє виконувати складні правила валідації, які вимагають комбінації різних умов та валідаторів. Наприклад, створити валідатор, який перевіряє, щоб значення одного поля було більше за значення іншого поля, або перевіряє, щоб сума числових значень полів була менше певної межі;

– повідомлення про помилки. Бібліотека надає можливість настроювати повідомлення про помилки, які виводяться при невдалій валідації даних. Існує можливість встановлювати власні повідомлення або використовувати наявні шаблони для різних валідаторів.

– інтеграція з фреймворками. Бібліотека «class-validator» добре інтегрується з популярними фреймворками, такими як «Express», «NestJS», «Mongoose» і багатьма іншими. Її можна легко використовувати разом з цими фреймворками для валідації даних, що надходять до серверної частини вашої інформаційної системи;

– тестування валідації. Бібліотека надає можливості для тестування валідації даних. Ви можете створювати автоматичні тести, які перевірятимуть правильність валідації для різних сценаріїв та вхідних даних.

Використання «class-validator» дозволило нам легко та ефективно валідувати дані, які вводять оператори в систему. Бібліотека забезпечила коректність та консистентність даних, а також допомогла уникнути непередбачуваних помилок під час обробки цих даних.

В розробці системи ми також використали бібліотеку «reflect-metadata», яка є важливим інструментом для роботи з метаданими інтерфейсу об'єктів в TypeScript або JavaScript. Вона надає можливість отримати доступ до метаданих класів, властивостей і методів під час виконання програми. «reflect-metadata» дозволяє отримати інформацію про типи, декоратори та інші деталі, що може бути корисним при створенні інформаційної системи для підтримки процесу перевірки лічильників води.

Основні можливості, які надала бібліотека «reflect-metadata»:

– отримання інформації про типи. Бібліотека дозволяє отримати інформацію про типи параметрів, повернутих значень та інших даних. Це може бути корисно для автоматичної генерації форм, перевірки типів та інших операцій, які вимагають знання про типи даних;

– робота з декораторами. Бібліотека «reflect-metadata» дозволяє отримувати доступ до декораторів, що застосовуються до класів, властивостей або методів. Це дає можливість виконувати додаткові дії на основі цих декораторів, наприклад, валідувати дані або виконувати певні перетворення.

– динамічне створення об'єктів. Вона дозволяє динамічно створювати екземпляри класів і викликати методи з метаданими. Це може бути корисно для

динамічного створення об'єктів з використанням метаданих, збереження стану або виконання певних операцій в залежності від даних;

– автоматична генерація документації. Завдяки «reflect-metadata» ви можете отримувати інформацію про метадані класів, методів та властивостей. Це може бути корисним для автоматичної генерації документації інформаційної системи. Можна використовувати метадані для створення описів класів, методів і властивостей, а також для визначення типів параметрів і повернутих значень;

– підтримка DI-контейнерів. Бібліотека «reflect-metadata» допомагає забезпечити підтримку інверсії керування (DI) та контейнерів введення залежностей. Можливо використовувати метадані, щоб визначити залежності та їхні типи, що дозволить DI-контейнерам автоматично розрізняти та вводити залежності під час створення об'єктів;

– тестування та налагодження. Бібліотека допомагає забезпечити зручність під час тестування і налагодження інформаційної системи. Є можливість отримувати доступ до метаданих, щоб перевірити наявність декораторів, типів та іншої інформації, що допоможе при створенні тестів або розв’язанні проблем у процесі налагодження.

Загалом, використання бібліотеки «reflect-metadata» дозволило нам отримувати доступ до метаданих класів, методів і властивостей, що дозволяє здійснювати динамічні операції, створювати гнучкі механізми валідації, генерувати документацію та підтримувати DI-контейнери. Це посприяло розробці більш гнучкої, модульної та легко налагоджуваної інформаційної системи для підтримки процесу перевірки лічильників води в підсистемі «Оператор».

Також у розробці нашої системи ми використовували бібліотеку «lodash». Адже це одна з найпопулярніших «JavaScript»-бібліотек, яка надає корисні функції для роботи з масивами, рядками, об'єктами та колекціями даних. Вона пропонує набір утиліт для спрощення та оптимізації розробки вебдодатків.

Бібліотека «lodash» дозволила здійснювати пошук, фільтрацію, сортування, злиття та інші маніпуляції з масивами даних. Наприклад, завдяки їй ми змогли використовувати такі функції, як «sortBy», «groupBy». Також вона надала

набір функцій для маніпуляцій об'єктами, наприклад, клонування. Це корисно для створення копій об’єкта без посилання на оригінал.

Бібліотека спростила роботу з рядками, адже має функції розбиття рядків на підрядки, з’єднання, форматування та інші маніпуляції з рядками. «Lodash» дозволила працювати з функціями, а саме реалізовувати композицію, каріювання функцій, мемоїзацію та інші техніки функціонального програмування. Ми використали такі функції, як «compose», «curry», «memoize» та інші для покращення роботи з функціями в «JavaScript».

Отже, завдяки широкому спектру функцій, які надає «lodash», ми спростили та оптимізували обробку даних, зокрема масивів, рядків і об'єктів. Використання функцій «lodash», таких як фільтрація, сортування, злиття, групування, дозволило зручно та ефективно обробляти дані повірок лічильників води. Функції для роботи з об'єктами, такими як клонування і злиття, забезпечили безпеку та консистентність даних.

Загалом, використання «lodash» у нашій системі підвищило продуктивність, ефективність і читабельність коду, забезпечивши багато корисних функцій для роботи з даними. Вона дозволила нам зосередитися на розробці бізнес-логіки системи, мінімізуючи зусилля на реалізацію рутинних операцій обробки даних.

Також ми використовували бібліотеку «RxJS» у нашій системі.

Основні можливості, які вона нам надала:

– створення потоків даних. Ми використовували Observable-об'єкти, які дозволяють створювати потоки даних та працювати з ними. Кожен потік являв собою послідовність подій або значень, які могли бути отримані асинхронно. Я використовував ці потоки для отримання даних про лічильники води, взаємодії з користувачем та обробки результатів перевірки;

– керування подіями. За допомогою операторів «RxJS» ми могли обробляти події та редагувати їх в потоках даних. Це дозволяло нам створювати логіку обробки подій, таку як натискання кнопок, введення даних або отримання результатів перевірки лічильників. Ми використовували оператори, такі як «map", «filter» та «tap», для трансформації та фільтрації подій;

– асинхронні операції. «RxJS» дозволяє зручно працювати з асинхронними операціями, такими як запити до сервера чи операції з базою даних. За допомогою операторів, таких як «switchMap», «mergeMap» та «concatMap», ми могли здійснювати асинхронні операції та керувати послідовністю їх виконання. Це було особливо корисно при взаємодії з сервером для отримання даних про лічильники води;

– обробка помилок. «RxJS» надає засоби для обробки та керування помилками в потоках даних;

– за допомогою операторів, таких як «catchError», «retry» та «throwError», ми могли елегантно обробляти помилки, що виникають під час виконання операцій, і вживати відповідних дій для відновлення потоку або повідомлення користувача;

– комбінування та паралельна обробка даних. За допомогою «RxJS» я міг комбінувати та об'єднувати дані з різних джерел одночасно. Використовуючи оператори, такі як «merge», «combineLatest», «forkJoin», ми могли об'єднувати результати запитів до різних API або комбінувати дані з різних джерел в одну потужну потокову обробку;

– стрімлення та кешування даних. «RxJS» надавав зручні засоби для стрімлення та кешування даних. Я міг використовувати оператори, такі як «throttleTime», «debounceTime», «cache», для керування частотою подій, забезпечення ефективної роботи з потоками даних та зменшення навантаження на сервер;

– тестування. «RxJS» має вбудовану підтримку для тестування. Це дозволило нам легко створювати тестові сценарії для перевірки правильності роботи Observable-потоків, операторів та логіки обробки даних. За допомогою різноманітних функцій тестування, таких як «marble testing», «TestScheduler» ми могли ефективно валідувати та перевіряти поведінку потоків;

Використання бібліотеки «RxJS» у нашій системі дозволило нам зручно та ефективно працювати з асинхронними подіями, обробляти та комбінувати дані,

керувати станом системи та забезпечити ефективну обробку запитів. Вона підвищила якість, швидкість та розширюваність системи.

Клієнтська частина програмного забезпечення була написана з використанням мови програмування «TypeScript» та бібліотеки «React». «React» - це відкрита «JavaScript» бібліотека, що використовується для розробки інтерфейсів користувача. Вона була розроблена компанією «Facebook» і випущена у 2013 році. «React» заснований на компонентному підході, що дозволяє розбити інтерфейс на невеликі, самодостатні та повторно використовувані компоненти.

Під час розробки клієнтської частини було активно використано компонентний підхід, який дозволяє розбити інтерфейс на невеликі та повторно використовувані компоненти. Це полегшує управління додатком, реагування на якісь зміни та покращує читабельність коду.

Для того, аби у користувача був елегантний та сучасний інтерфейс, ми вирішили використати бібліотеку «Ant Design» або просто «antd». Це одна з найпопулярніших бібліотек для розробки інтерфейсу вебдодатків на основі «React». «Antd» пропонує широкий вибір готових компонентів і стилів, які можуть швидко створити сучасні та естетичні інтерфейси користувача. Ми використали більшість компонентів, а саме кнопки, форми, таблиці, меню, модальні вікна, каруселі.

Один із головних аспектів, який нам сподобався в «antd» - адаптивний дизайн. Завдяки цьому, інформаційна система, яку ми розробляли, стала доступною та зручною для користувачів незалежно від розміру їхніх пристроїв. Це було особливо важливо для підсистеми "Оператор", оскільки оператори можуть використовувати пристрої для доступу до систем, включаючи різні комп'ютери, планшети та смартфони.

«Antd» також забезпечує потужні можливості для роботи з формами та перевіркою даних. Ми з легкістю створювали складні форми з різноманітними полями введення. Валідація даних в «antd» допомогла забезпечити правильність та повноту введених даних, забезпечуючи надійність функціональності нашої

інформаційної системи.

В клієнтській частині ми використовували бібліотеку «react-router». Вона надала мені потужні інструменти для керування маршрутизацією в нашій інформаційній системі.

Основні можливості, які вона нам надала:

– визначення маршрутів. За допомогою «react-router» ми змогли визначати маршрути, які відповідають різним сторінкам та компонентам підсистеми. Ми використовували такі компоненти, як «<Route>» та «<Switch>», для визначення шаблонів URL та відповідних компонентів, які мають бути відображені при відповідному URL;

– навігація між сторінками. За допомогою «react-router» ми змогли створювати посилання та кнопки, які забезпечували навігацію між різними сторінками системи. Ми використовували компонент «<Link>», який дозволяв створювати посилання з автоматично згенерованим URL, що збігається з визначеним маршрутом;

– параметризовані маршрути. «React-router» дозволяє використовувати параметри в маршрутах для передачі динамічних даних. Це було особливо корисно при роботі зі сторінками деталей лічильників води. Були використані параметри в URL для ідентифікації конкретного лічильника та відображення його деталей;

– захист маршрутів. За допомогою «react-router» я міг забезпечити захист певних маршрутів від несанкціонованого доступу. Я використовував компонент «<PrivateRoute>», який перевіряв, чи користувач має достатні права доступу до конкретного маршруту. У разі несанкціонованого доступу користувач перенаправлявся на сторінку входу;

– анімація переходів. «React-router» надавав підтримку для анімації переходів між сторінками. Можливо було використовувати компонент «<CSSTransition>» з бібліотекою «react-transition-group» для створення плавних анімацій під час зміни сторінок.

Використання бібліотеки «react-router» у нашій системі спростило реалізацію маршрутизації, навігації та захисту маршрутів. Вона надала зручні інструменти, які дозволили користувачам легко переміщатися між різними сторінками та взаємодіяти з різними компонентами системи.

Для забезпечення зв'язку між клієнтською частиною та сервером ми обрали бібліотеку «axios». Це досить популярна бібліотека «JavaScript», яка надає простий та потужний інтерфейс для виконання HTTP-запитів. Одна з головних переваг «axios» - це простота використання. Бібліотека пропонує зрозумілий API, який дозволяє легко відправляти запити на сервер і обробляти отримані відповіді. Ми використовували бібліотеку «axios» для виконання запитів GET, POST, PUT та DELETE, щоб отримати та надсилати дані з сервера.

Однією з ключових можливостей «axios» є підтримка обіцянок (promises) та асинхронного коду. Це дозволяє зручно та ефективно обробляти асинхронні запити до сервера без блокування інтерфейсу користувача. Ми використовували обіцянки для керування запитами «axios» та обробки результатів асинхронних операцій.

Одна з важливих функцій «axios», яку ми використовували у своїй дипломній роботі, - це можливість налаштування інтерцепторів (interceptors). Перехоплювачі можуть перехоплювати та модифікувати запити та відповіді перед їх відправленням або після отримання. Це було корисно для додавання заголовків авторизації до запитів, обробки помилок або перенаправлення користувача на сторінку входу у випадку недійсного токена.

Також «axios» забезпечує зручний спосіб обробки помилок під час виконання запитів. Завдяки цьому ми змогли елегантно обробляти помилки, які виникли під час комунікації з сервером.

Загалом, використання бібліотеки «axios» у розробці інформаційної системи було надзвичайно корисним. Бібліотека надала потужний інструмент для здійснення HTTP-запитів до сервера, обробки відповідей та керування асинхронним кодом. Ми були задоволені її простим використанням та багатьма функціями, які закрили всі наші потреби в комунікації з сервером у контексті

розробки підсистеми «Оператор» для нашої інформаційної системи.

Отже, застосування цих засобів розробки дозволило забезпечити швидку та ефективну розробку серверної та клієнтської частин програмного забезпечення, а також організувати безпечну та зручну взаємодію з базою даних.

## 4.2 Вимоги до технічного забезпечення

### 4.2.1 Загальні вимоги

Для належної функціональності розробленої системи необхідно мати наступну конфігурацію технічних засобів:

– Комп'ютер з процесором Intel Core i3 або AMD, з тактовою частотою не менше 2,5 ГГц.

– Мінімум 2 ГБ оперативної пам'яті (RAM).

– Встановлений браузер, необхідний для запуску вебдодатків.

– Комп'ютерна периферія, така як мишка або тачпад, монітор та клавіатура.

Дані вимоги впливають на забезпечення правильної роботи системи та задоволення її вимог продуктивності.

## 4.3 Архітектура програмного забезпечення

### 4.3.1 Структура класів

Даний застосунок складається з багатьох класів, але серед них є, звичайно, основні. В таблиці 4.1 наведено опис цих основних класів.

Таблиця 4.1 – Опис основних класів застосунку

| Клас | Призначення |
|---|---|
| Operator | клас, який описує сутність «Оператор» |
| Metrologist | клас, який описує сутність «Метролог» |
| WaterMeter | клас, який описує сутність «Лічильник» |
| VerificationRecord | клас, який описує сутність «Протокол» |

| | |
|---|---|
| AuthService | клас-сервіс, який містить в собі логіку автентифікації |
| ActorsService | абстрактний клас-сервіс, який містить в собі базову логіку для всіх типів акторів |
| OperatorsService | клас-сервіс, який містить в собі логіку для роботи з сутністю «Оператор» |
| OperatorsController | клас, який містить в собі методи, які представляє API операторів |
| WaterMetersService | клас-сервіс, який містить в собі логіку для роботи з сутністю «Лічильник» |
| WaterMetersController | клас, який містить в собі методи, які представляє API лічильників |
| VerificationRecordsService | клас-сервіс, який містить в собі логіку для роботи з сутністю «Протокол» |
| VerificationRecordsController | клас, який містить в собі методи, які представляє API протоколів |

### 4.3.2 Схема послідовності

На даній схемі послідовності ми показали процеси серверної взаємодії оператора. Схема послідовності наявна у графічних додатках (ІС91.130БАК.004 Д4).

### 4.3.3 Діаграма компонентів

Структурна схема компонентів, яка зображує розбиття розробленого застосунку на логічні компоненти та зв’язки між ними представлена на Рисунку 4.1.

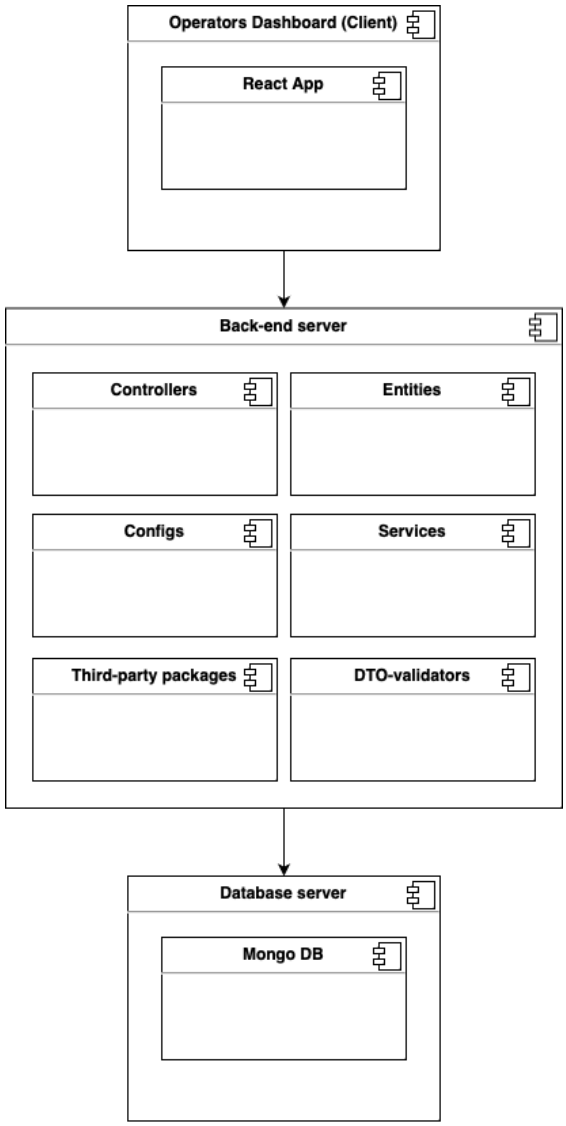


Рисунок 4.1 – Структурна схема компонентів

### 4.3.4 Специфікація функцій

Функції та методи, що мають найбільший вплив на функціонал підсистеми «Оператор» наведено у таблиці 4.2

Таблиця 4.2 – Специфікація функцій застосунку

| Назва функції | Специфікація функції |
|---|---|
| hashPassword() | Даний метод викликається при хешуванні пароля |
| verifyPassword() | Даний метод викликається при перевірці пароля на валідність |

| | |
|---|---|
| generateAccessToken() | Даний метод викликається при генерації access токена |
| generateRefreshToken() | Даний метод викликається при генерації refresh токена |
| generateTokenPair() | Даний метод викликається при генерації пари access та refresh токенів |
| verifyAccessToken() | Даний метод викликається при перевірці на валідність access токену |
| uploadFile() | Даний метод викликається при завантаженні файлу в систему |
| getVerificationRecordById() | Даний метод викликається, щоб отримати протокол повірки лічильників за id |
| getVerificationRecords() | Даний метод викликається, щоб отримати масив протоколів повірки лічильників за заданим фільтром |
| createVerificationRecord() | Даний метод викликається при створенні протоколу повірки лічильників води |
| signIn() | Даний метод буде викликано при вході користувача в систему |
| changePassword() | Даний метод буде викликано при зміні пароля користувачем |
| getByID() | Даний метод викликається, щоб отримати абстрактного актора за id |
| getByUsername() | Даний метод викликається, щоб отримати абстрактного актора за username |
| updateById() | Даний метод викликається, щоб оновити абстрактного актора за id |
| refreshAccess() | Даний метод викликається для оновлення доступу в систему |
| getWaterMeterById() | Даний метод викликається для отримання |

| | |
|---|---|
| | лічильника за id |
| createWaterMeter() | Даний метод викликається для створення лічильника |
| updateWaterMeter() | Даний метод викликається для оновлення лічильника |
| logout() | Даний метод викликається при виході користувача з системи |

Висновок до розділу

У цьому розділі було детально розглянуто використані засоби розробки програмного забезпечення для реалізації проєкту. Ми розглянули засоби реалізації серверної та клієнтської частини, а також базу даних. Ретельно обрали та обґрунтували використання конкретних інструментів, зокрема мови програмування «TypeScript» для серверної частини, «TypeScript» та бібліотеки «React» для клієнтської частини, а також бази даних «MongoDB».

Також були сформульовані вимоги до технічних засобів, необхідних для успішної роботи розробленого застосунку. Це включало конкретні характеристики комп'ютера, наявність сумісного браузера та використання комп'ютерної периферії.

Було детально описано архітектуру системи за допомогою структурних схем класів, схем діяльності та компонентів. Це надає розуміння структури та взаємодії компонентів в системі, сприяє зрозумінню архітектурних рішень та полегшує подальшу розробку та підтримку проєкту.

# 5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

## 5.1 Керівництво користувача

При відкритті вебзастосунку користувач буде перенаправлений на сторінку авторизації, зображену на рисунку 5.1. З цієї сторінки він матиме можливість увійти в систему, якщо він вже має обліковий запис. Для цього потрібно ввести свій логін та пароль, а потім натиснути кнопку «Увійти».

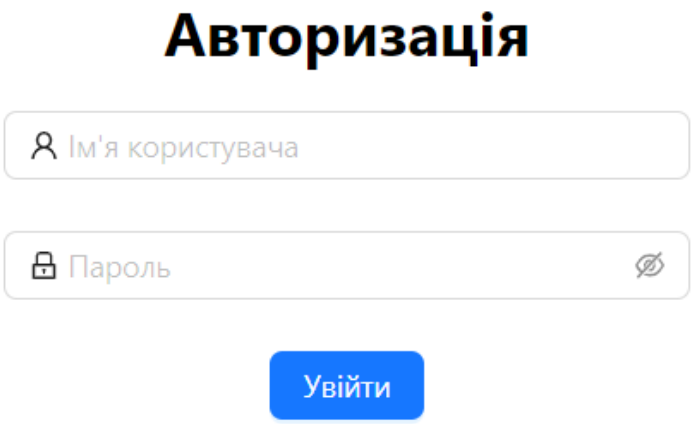


Рисунок 5.1 — Сторінка авторизації

З міркувань безпеки при введенні пароль закривається крапками, як показано на рисунку 5.2.

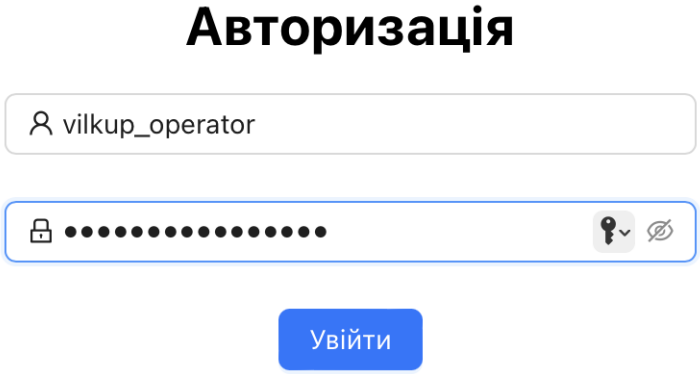


Рисунок 5.2 – Приховування пароля

У разі неуспішної авторизації користувач бачить сповіщення про те, що авторизація не була успішною. Даний інтерфейс користувача зображений на рисунку 5.3.

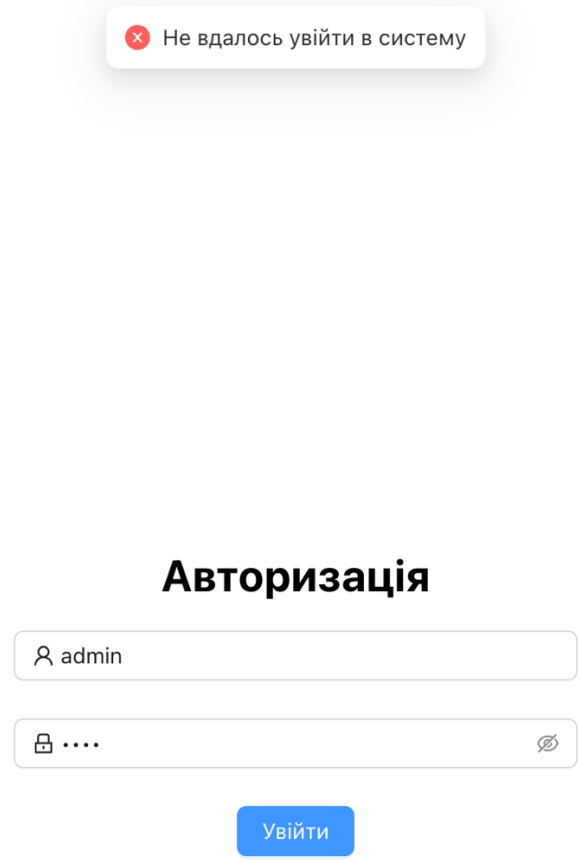


Рисунок 5.3 – Помилка авторизації

Після успішної авторизації користувач потрапляє на головну сторінку, на якій він має змогу обрати один з пунктів меню, яке зображене на рисунку 5.4.

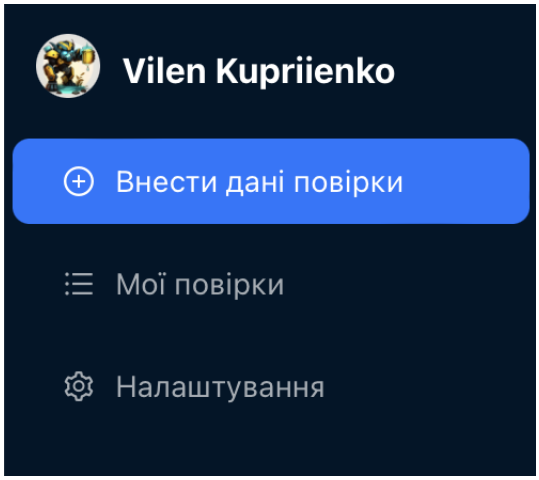


Рисунок 5.4 – Меню користувача

Пунктом меню за замовчуванням є «Внести дані повірки», при виборі якого користувач переходить на сторінку створення повірки, яка зображена на рисунку 5.5 (перший екран сторінки) та рисунку 5.6 (другий екран сторінки).

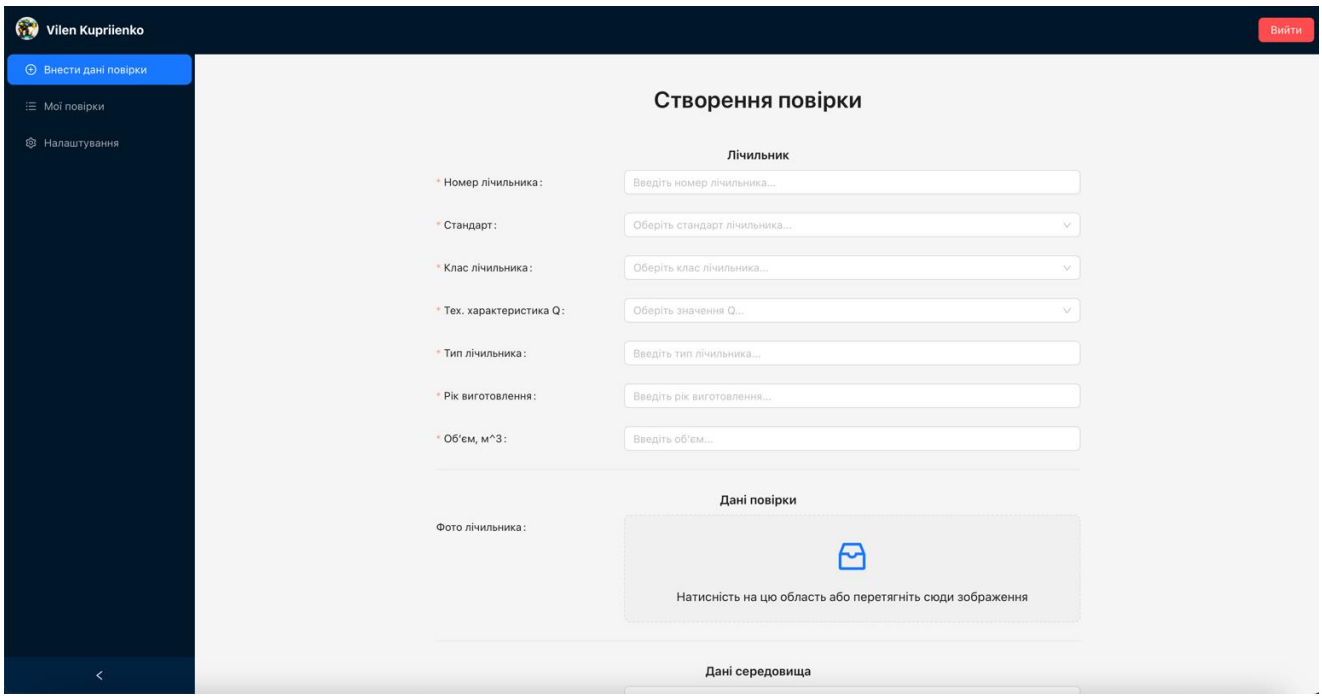


Рисунок 5.5 — Сторінка створення повірки (перший екран)

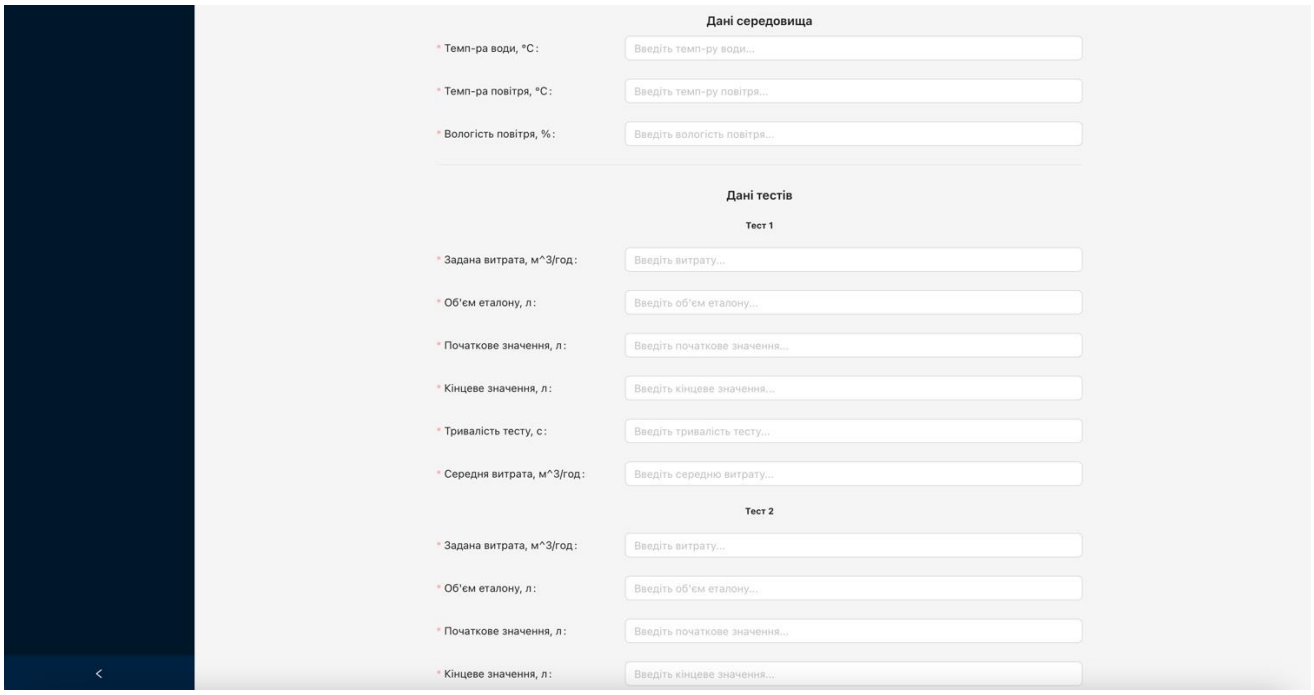


Рисунок 5.6 — Сторінка створення повірки (другий екран)

Для створення нової повірки користувач має внести у форму такі дані як:

Дані лічильника:

- номер лічильника
- стандарт
- клас лічильника
- тех. характеристика Q
- тип лічильника
- рік виготовлення
- об’єм, м^3

Дані повірки:

- фото лічильника

Дані середовища:

- температура води, °C
- температура повітря, °C
- вологість повітря, %

Дані трьох тестів:

- задана витрата, м^3/год
- об'єм еталона, л
- початкове значення, л
- кінцеве значення, л
- тривалість тесту, с
- середня витрата, м^3/год

Після введення всіх необхідних даних користувач має натиснути кнопку “Створити”. Заповнені дані відправляються на сервер.

В разі, якщо всі дані були введені правильно, користувач бачить повідомлення про успішне внесення даних про повірку, яке зображене на рисунку 5.7.

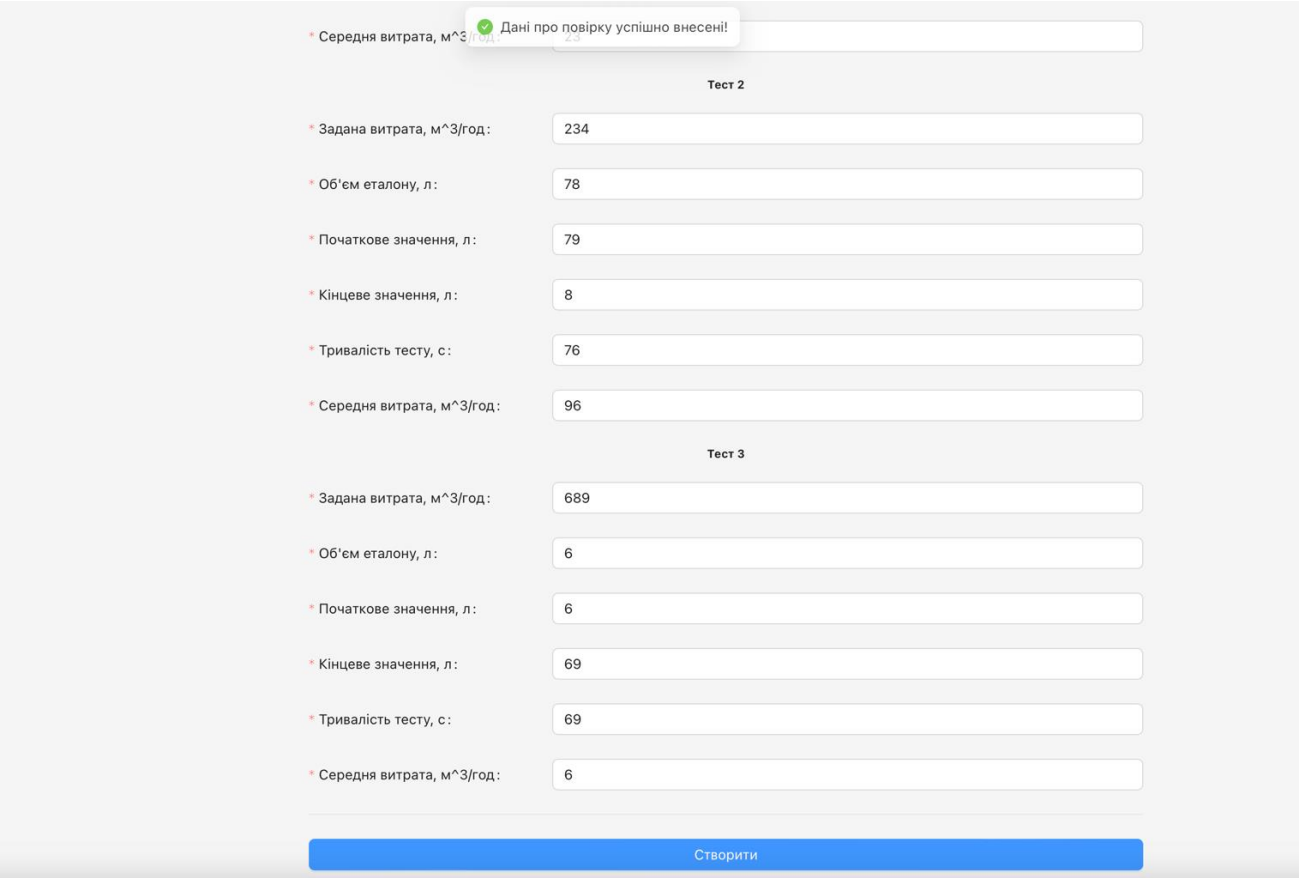


Рисунок 5.7 — Повідомлення про створення повірки

У разі, якщо дані були введені неправильно, користувач бачить які помилки та в яких полях він допустив, як зображено на рисунку 5.8.

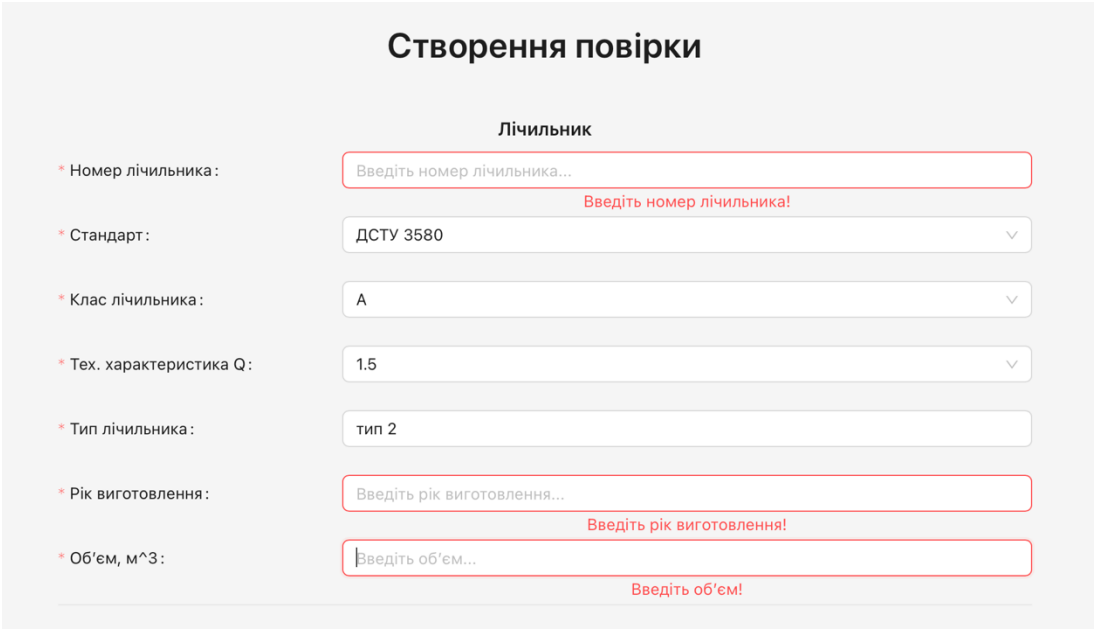


Рисунок 5.8 — Індикація помилок введення даних

Користувач може переглянути список створених ним повірок. Для цього він має у меню зверху зліва, як показано на рисунку 5.9, обрати пункт «Мої повірки» та натиснути на нього.

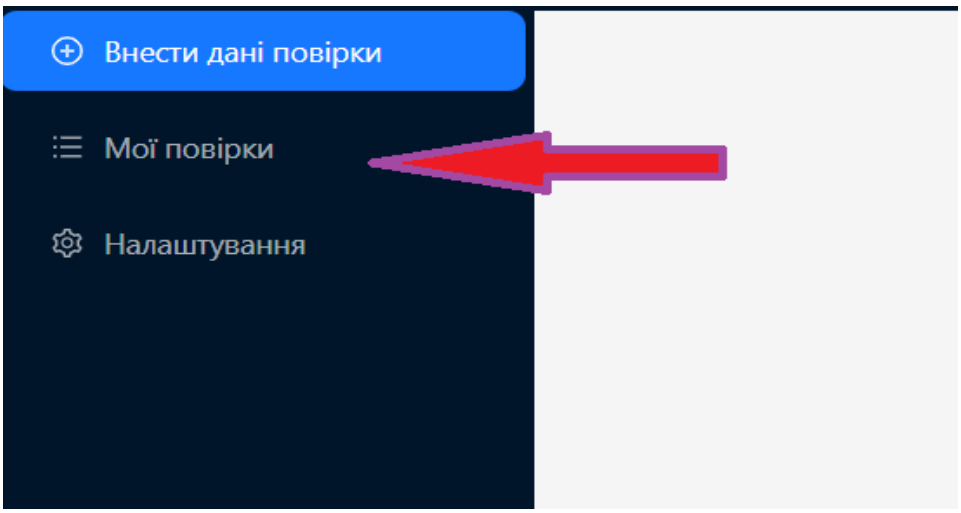


Рисунок 5.9 — Меню вибору перегляду повірок

Тепер користувач знаходиться на сторінці перегляду власних повірок, як це зображено на рисунку 5.10.

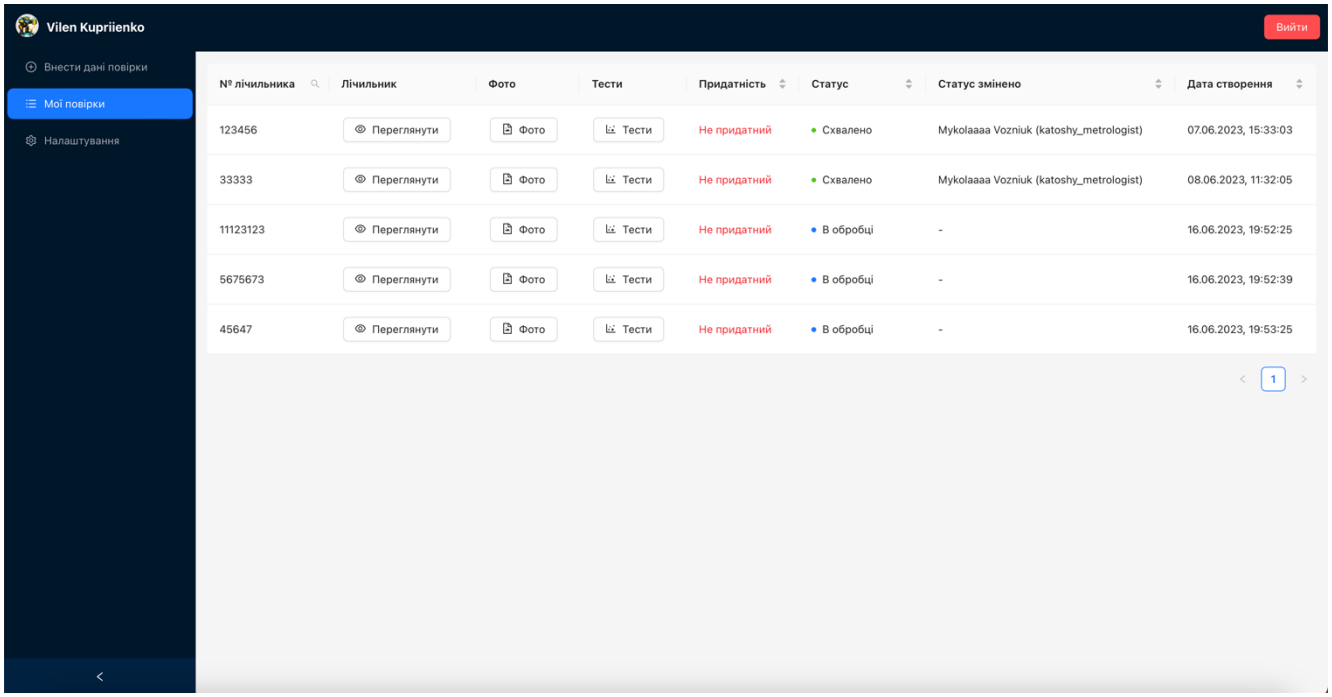


Рисунок 5.10 — Сторінка перегляду повірок

Тут він може переглянути дані лічильника, його фото, результати проведених тестів, результат повірки (придатний чи не придатний), статус обробки метрологом («схвалено», «в обробці» або «відхилено»).

Також користувач може переглянути ім’я та логін метролога, який змінив статус повірки, та дату створення повірки.

Користувачу доступний пошук повірок по номеру лічильника, до якого прив’язана повірка. Щоб ним скористатися, користувачу необхідно натиснути на іконку лупи в заголовку таблиці біля надпису «№ лічильника» і ввести номер або частину номера лічильника, як показано на рисунку 5.11. Повірки, що представлені в таблиці, будуть відфільтровані.

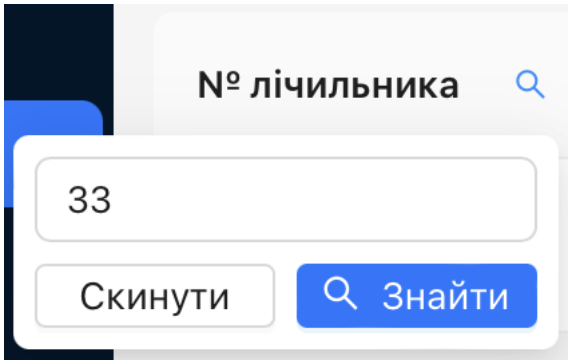


Рисунок 5.11 – Вікно пошуку повірок за номером лічильника

При натисканні на кнопку «Переглянути» в рядку таблиці в стовпці «Лічильник» користувач побачить модальне вікно, в якому він зможе переглянути дані лічильника, до якого прив’язана повірка, і яке зображено на рисунку 5.12.

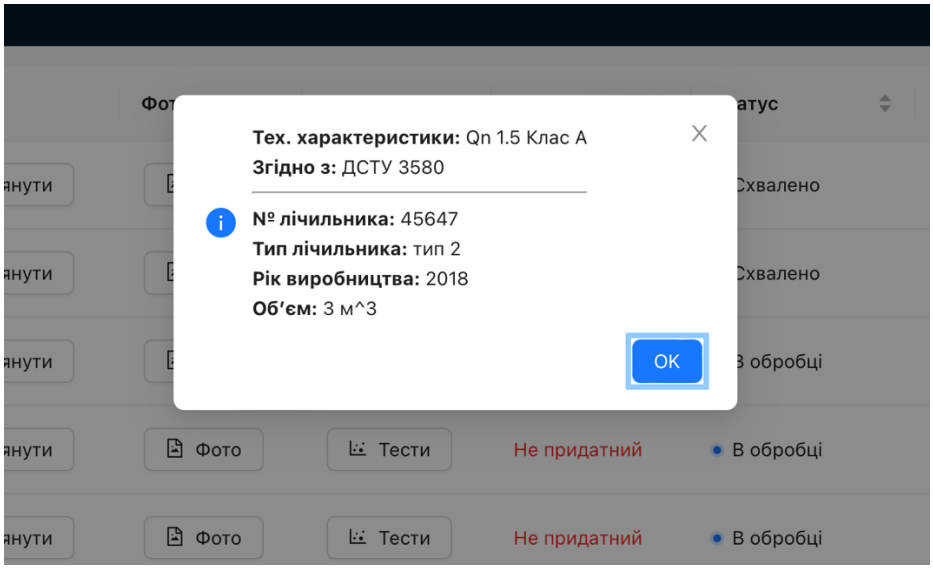


Рисунок 5.12 – Модальне вікно з даними лічильника

При натисканні на кнопку «Фото» в рядку таблиці користувач побачить модальне вікно, в якому він зможе переглянути фотографії лічильника, які були прикріплені до повірки, і яке зображено на рисунку 5.13.

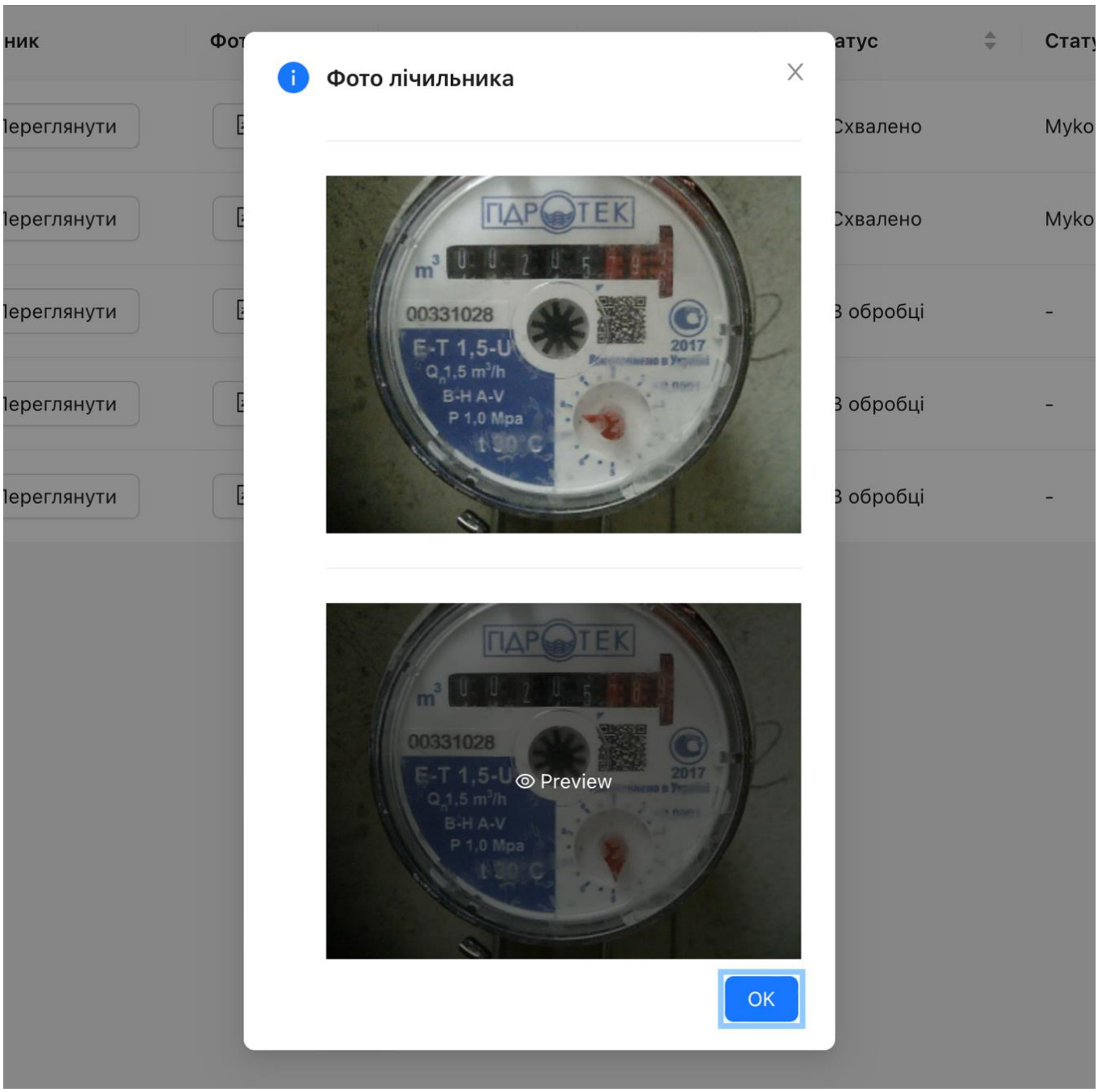


Рисунок 5.13 – Модальне вікно з фотографіями лічильника

Користувачу також доступне вікно для детального перегляду фото, яке можна відкрити натиснувши на слово «Preview», яке з’являється на фото після наведення курсора.

Користувач може наблизити чи віддалити фото, повернути його та інвертувати. Дане вікно зображене на рисунку 5.14.


Рисунок 5.14 – Вікно детального перегляду фото

При натисканні на кнопку «Тести» в рядку таблиці користувач побачить модальне вікно, в якому він зможе побачити тести повірки та дані про середовище, в якому вони були проведені. Дане модальне вікно зображено на рисунку 5.15.

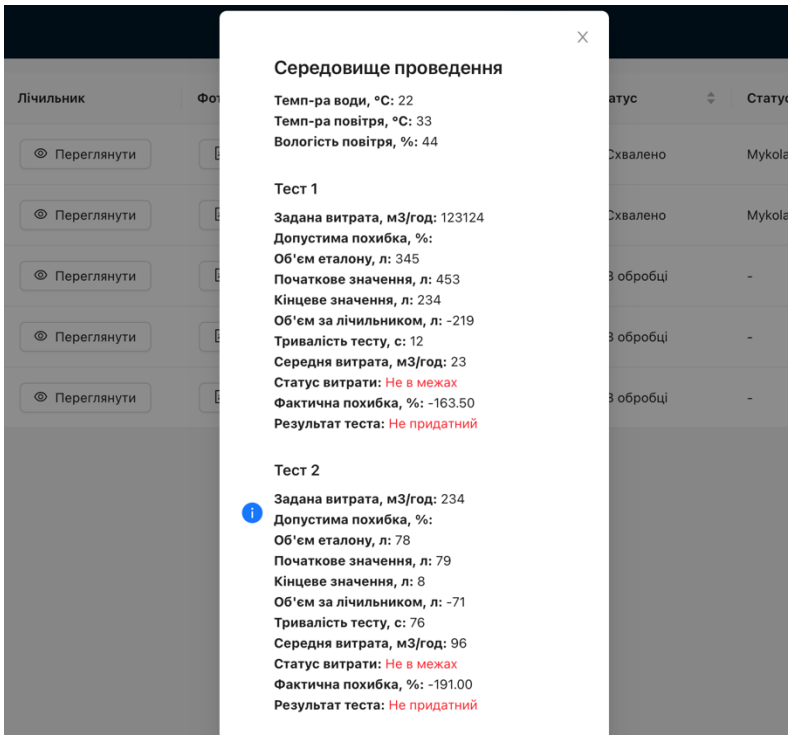


Рисунок 5.15 – Модальне вікно з даними тестів

Також користувач може перейти на сторінку налаштувань, натиснувши кнопку “Налаштування” як це показано на рисунку 5.16.

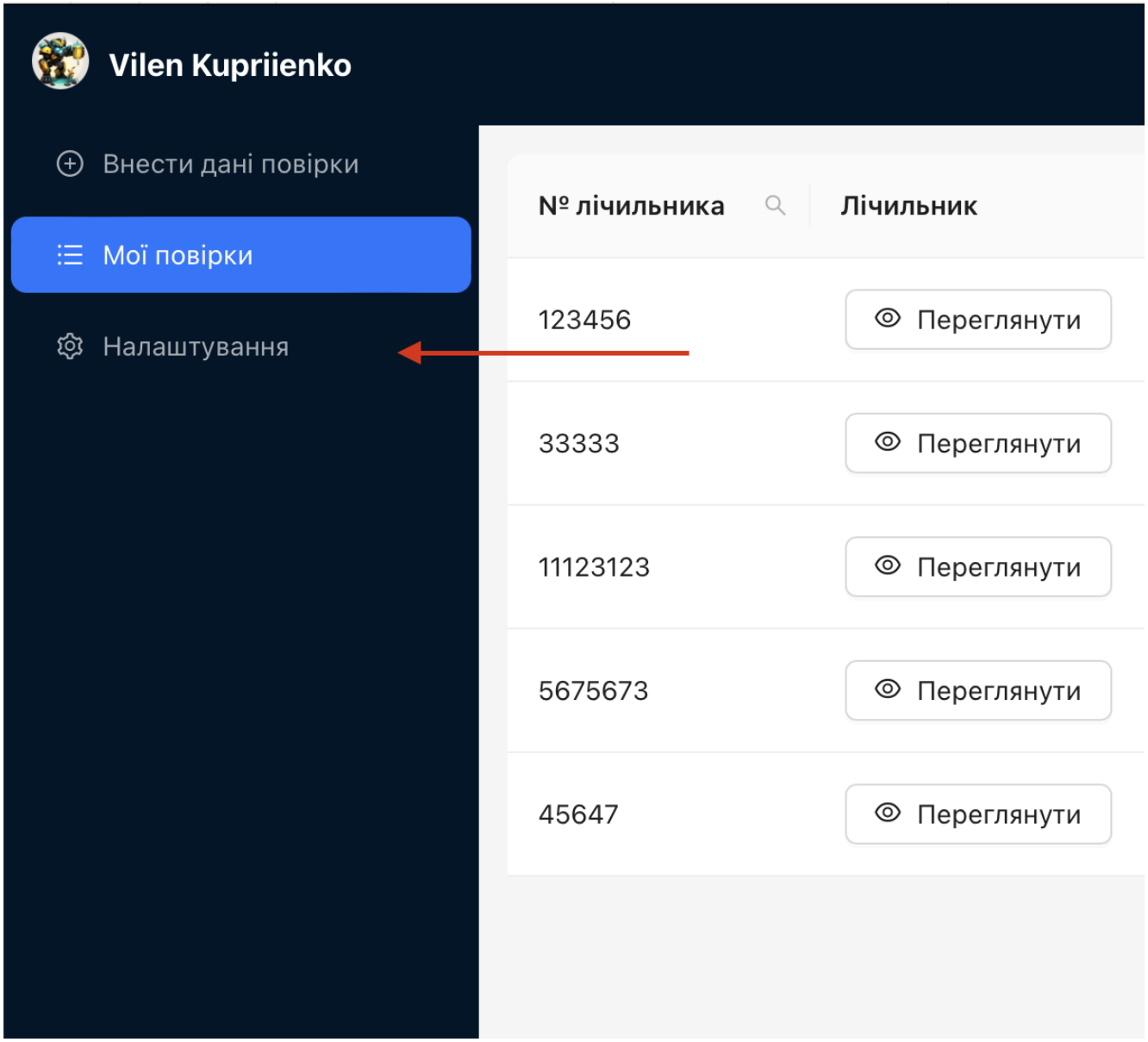


Рисунок 5.16 – Меню вибору налаштувань

Користувач потрапляє на сторінку налаштувань. Для зміни пароля користувач має ввести старий пароль і новий, на який хоче його замінити. Дана сторінка зображена на рисунку 5.17.

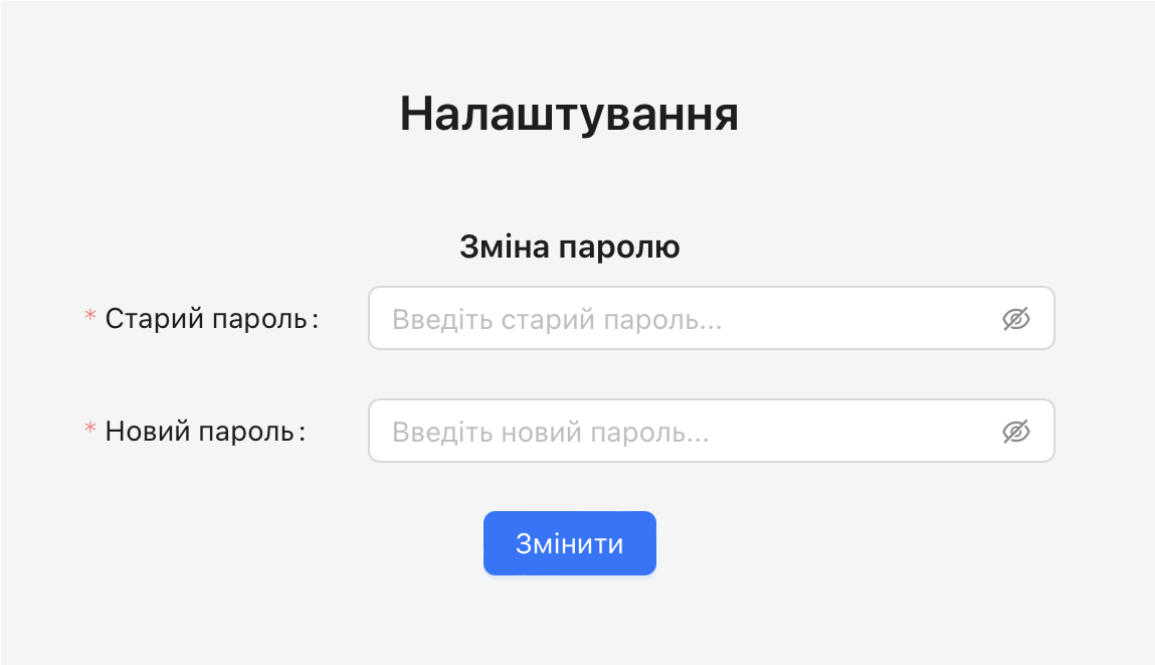


Рисунок 5.17 – Сторінка зміни пароля

Для виходу із системи користувачу необхідно натиснути на кнопку “Вихід”, як показано на рисунку 5.18, і користувача відправить на початкове вікно авторизації.

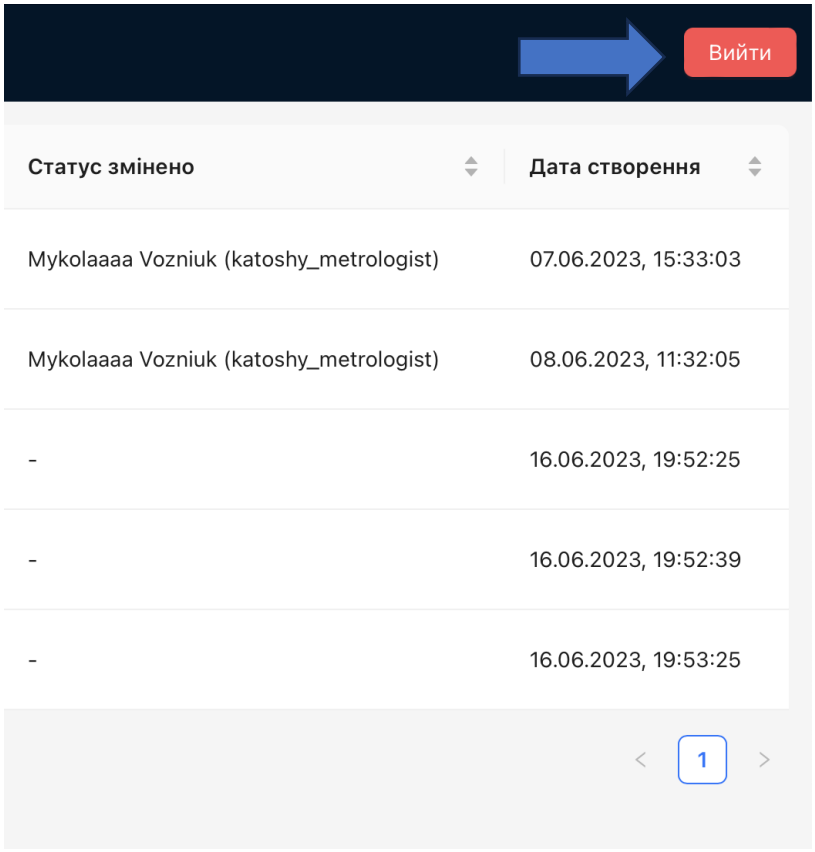


Рисунок 5.18 – Вихід із системи

## 5.2 Випробування програмного продукту

З метою перевірки відповідності програмного продукту вимогам, що визначені у технічному завданні, було проведено тестування інформаційної системи повірки лічильників підсистеми “Оператор”. В цьому розділі представлено опис тест-кейсів, які містять послідовність дій для їх виконання.

### 5.2.1 Мета випробувань

Для досягнення цілі випробувань було обрано перевірку функціональності підсистеми «Оператор» інформаційної системи повірки лічильників.

### 5.2.2 Загальні положення

Тестування програмного забезпечення є процесом перевірки й оцінки якості програмного продукту. Цей процес містить в собі виконання різних тестів з метою виявлення помилок, недоліків і невідповідностей до вимог і очікувань. Тестування програмного забезпечення має такі цілі як:

– виявлення помилок: Тестування дозволяє ідентифікувати дефекти, помилки, неочікувану поведінку або невідповідності функціональним або нефункціональним вимогам програми;

– перевірка відповідності вимогам: Тестування виконується для перевірки того, чи відповідає програмний продукт вимогам, які були визначені в технічному завданні або специфікації;

– підтвердження працездатності: Тестування допомагає переконатися, що програма працює так, як очікувалося, і виконує свої функції безперебійно;

– підвищення якості: Шляхом виявлення і виправлення помилок, тестування сприяє покращенню якості програмного забезпечення.

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;

– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

У даному розділі представлено висновки, отримані в результаті досліджень, що стосуються основних функцій повірки лічильника оператором. Результати цих досліджень представлені у таблицях 5.1 - 5.6.

Таблиця 5.1 – Тестування реєстрації користувача - оператора

| | |
|---|---|
| Ціль тесту | Перевірка авторизації користувача |
| Передумова | Користувач знаходиться на сторінці авторизації |
| Вхідні дані | Дані користувача |
| Послідовність кроків | - У полі «Ім’я користувача» ввести логін латиницею, наприклад, «Vilen»;<br>- У полі «Пароль» ввести пароль, який складається з латинських символів та чисел, наприклад, «qwert123»;<br>- Натиснути кнопку «Увійти». |
| Очікуваний результат | Користувач авторизований у системі |
| Система після тестування | Користувач знаходиться на сторінці створення повірки |

Таблиця 5.2 —Тестування створення протоколу повірки

| | |
|---|---|
| Ціль тесту | Перевірка створення протоколу повірки |
| Передумова | Користувач знаходиться на сторінці створення повірки |
| Вхідні дані | Параметри повірки |
| Послідовність кроків | - У полі «Номер лічильника» необхідно ввести номер |

| | |
|---|---|
| | лічильника води, наприклад, «11111»;<br>- У полі «Стандарт» вибрати один з двох варіантів, наприклад, ДСТУ 3580;<br>- У полі «Клас лічильника» вибрати один з трьох варіантів, наприклад, «А»;<br>- У полі «Тех. Характеристика Q» вибрати один з чотирьох варіантів, наприклад, 1.5;<br>- У полі «Тип лічильника» ввести номер лічильника води, наприклад, «11111»;<br>- У полі «Рік виготовлення» ввести рік виготовлення лічильника води, наприклад, «2000»;<br>- У полі «Номер лічильника» необхідно ввести номер лічильника води, наприклад, «11111»;<br>- У полі «Об’єм, м^3» ввести число об’єму лічильника води, наприклад, «300»;<br>- У поле «Фото лічильника» завантажити зображення лічильника води; |

| | |
|---|---|
| | - У полі «Темп-ра води, °С» ввести число, наприклад, «30»;<br>- У полі «Темп-ра повітря, °С» ввести число, наприклад, «30»;<br>- У полі «Вологість повітря, %» ввести число, наприклад, «60»;<br>- У полі «Задана витрата, м^3/год» ввести число, наприклад, «1»;<br>- У полі «Об'єм еталона, л» ввести число, наприклад, «1»;<br>- У полі «Початкове значення, л» ввести число, наприклад, «25»;<br>- У полі «Кінцеве значення, л» ввести число, наприклад, «26»;<br>- У полі «Тривалість тесту, с» ввести число, наприклад, «60»;<br>- У полі «Середня витрата, м^3/год» ввести число, наприклад, «60».<br>Натиснути кнопку «Створити» |
| Очікуваний результат | - Протокол доданий у систему |
| Система після тестування | - Користувач знаходиться на сторінці створення повірки<br>Дані про нову повірку збережені у базі даних. |

Таблиця 5.3– Тестування перегляду списку повірок

| | |
|---|---|
| Ціль тесту | Перевірка перегляду повірок |
| Передумова | Користувач знаходиться на сторінці створення повірки або на сторінці налаштувань |
| Вхідні дані | |
| Послідовність кроків | Натиснути кнопку «Мої повірки» |
| Очікуваний результат | Користувач знаходиться на сторінці перегляду повірок |
| Система після тестування | Користувач знаходиться на сторінці перегляду повірок |

Таблиця 5.4–Тестування перегляду налаштувань

| | |
|---|---|
| Ціль тесту | Перевірка перегляду налаштувань |
| Передумова | Користувач знаходиться на сторінці створення повірки або на сторінці перегляду повірок |
| Послідовність кроків | Натиснути кнопку «Налаштування» |
| Очікуваний результат | Користувач знаходиться на сторінці налаштувань |
| Система після тестування | Користувач знаходиться на сторінці налаштувань |

Таблиця 5.5 – Тестування зміни пароля

| | |
|---|---|
| Ціль тесту | Перевірка зміни пароля |
| Передумова | Користувач знаходиться на сторінці налаштувань |
| Вхідні дані | Паролі користувача |
| Послідовність кроків | - У полі «Старий пароль» ввести пароль, який складається з |

| | |
|---|---|
| | латинських символів та чисел, наприклад, «qwert123»;<br>- У полі «Новий пароль» ввести пароль, який складається з латинських символів та чисел, наприклад, «zxc123»;<br>- Натиснути кнопку «змінити». |
| Очікуваний результат | Користувач змінює пароль |
| Система після тестування | - Користувач знаходиться на сторінці налаштувань;<br>- Дані про новий пароль збережені у базі даних. |

Висновок до розділу

У цьому розділі було розглянуто важливі аспекти, пов'язані з розробкою та впровадженням програмного продукту. Були висвітлені ключові функції, які були реалізовані з урахуванням встановлених завдань. Користувачам була надана інформація щодо авторизації в системі, створення протоколів повірки, заповнення форм, навігації по застосунку, перегляду повідомлень та зміни пароля. Ця інформація допомагає користувачам ефективно використовувати програмний продукт та отримувати необхідну інформацію.

Також було представлено опис тестів та порядок їх виконання з метою перевірки відповідності програмного забезпечення функціональним вимогам, визначеним у технічному завданні. Цей етап є важливим для виявлення помилок, дефектів або некоректностей в програмному продукті та його вдосконалення перед випуском.

Завершуючи цей технологічний розділ, можна зробити висновок, що розроблений програмний продукт відповідає функціональним вимогам, які були визначені в технічному завданні. Керівництво користувача надає необхідну

інформацію для ефективного використання продукту. Випробування програмного продукту допомагають виявити та виправити помилки перед його впровадженням, забезпечуючи високу якість роботи програмного забезпечення.

# ВИСНОВКИ

У розробці інформаційної системи для підтримки процесу повірки лічильників велику роль відіграє підсистема «Оператор». Оператор виконує важливі завдання, пов'язані зі створенням протоколів повірок та їх заповненням.

Отже, в даній роботі було визначено ціль розробки застосунку, його призначення та функціонал. Також важливою частиною роботи був аналіз наявних аналогів, виявлення їх переваг та недоліків для внесення коректив у розробку нашої системи. На подальших етапах роботи з проєктом ми описали формат вхідних та вихідних даних, необхідних для роботи підсистеми «Оператор».

Далі в роботі була сформульована задача формування розкладу роботи операторів. Оскільки наша система допомагає операторам у виконанні їх обов'язків, ми вирішили розглянути й питання керування штатом операторів під час робочого процесу. Для розв’язання задачі було використано метод динамічного програмування.

Також ми створили зручний інтерфейс для операторів, що дозволяє ефективно працювати з повірками лічильників.

Розроблена інформаційна система складається з серверної та клієнтської частини, а також бази даних. Серверна частина була розроблена за допомогою мови програмування «TypeScript» та фреймворку «Nest». Вибір цих інструментів дозволяє забезпечити ефективну обробку запитів та забезпечити швидку та надійну роботу серверної частини системи. Для клієнтської частини ми також використовували мову «TypeScript» та бібліотеку «React». Для роботи з базою даних було застосовано «MongoDB». Ця нереляційна база даних забезпечує гнучкість і швидкість у роботі з даними. Використання "MongoDB" дозволяє ефективно зберігати та опрацьовувати дані про перевірку лічильників води.

Для більш детального розуміння структури діяльності системи була наведена схема, яка показує послідовність кроків та взаємодію компонентів системи. Також була розроблена діаграма компонентів, яка візуалізує взаємозв'язки та взаємодію між більшими компонентами системи.

Далі була наведена схема структури діяльності та діаграма компонентів. Були проведені та описані тестові сценарії, які покривають всі функціональні вимоги до підсистеми «Оператор». Тестування підтвердило ефективність та надійність розробленої інформаційної системи. Підсистема «Оператор» дозволяє операторам швидко та точно проводити перевірку лічильників води, забезпечуючи високу якість обслуговування споживачів та зменшуючи можливість помилок.

Розроблена система має потенціал для подальшого розширення та вдосконалення, наприклад, шляхом додавання нових функціональних елементів або інтеграції з іншими компонентами водопостачальних компаній.

ПЕРЕЛІК ПОСИЛАНЬ


1. TypeScript офіційна документація. [Електронний ресурс]. Режим доступу https://www.typescriptlang.org/docs/ 17.05.2023
2. Alain Chautard, "Mastering NestJS: Step-by-Step Guide to Building Enterprise-Grade Node.js Web Applications" (2020), 408 pages
3. John Papa, "TypeScript in Plain Language Volume 1: A Comprehensive Starting Up Guide" (2021), 293 pages
4. Christian Johansen, "Test-Driven JavaScript Development" (2013), 456 p.
5. Nir Kaufman, "React and React Native: A Complete Hands-On Guide to Modern Web and Mobile Development with React.js, 2nd Edition" (2020), 500 pages
6. Flavio Copes, "The React Handbook" (2021), 247 pages
7. Brad Traversy, "MERN Stack Front To Back: Full Stack React, Redux & Node.js" (2018), 346 pages
8. Nathanael Anderson, "Pro TypeScript: Application-Scale JavaScript Development" (2019), 404 p.
9. MongoDB офіційна документація. [Електронний ресурс]. Режим доступу https://docs.mongodb.com/ 17.05.2023
10. Frank Zammetti, "Learn React with TypeScript 3: Beginner's Guide to Modern React Web Development with TypeScript" (2018), 280 pages
11. Alex Banks, "React for Real: Front-End Code, Untangled" (2021), 344 pages
12. Caleb Curry, "TypeScript: Modern JavaScript Development" (2020), 314 pages
13. Тестування програмного забезпечення: підхід на основі методології Agile. [Електронний ресурс]. Режим доступу https://www.agilealliance.org/ 17.05.2023
14. Dr. Eduard Glatz, "Mastering MongoDB 4.x: Expert Techniques to Run High-Volume and Fault-Tolerant Database Solutions" (2018), 612 pages
15. Isaac Rabinovitch, "Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node" (2019), 820 pages

16. Robert D. Macredie, "Human-Computer Interaction: An Empirical Research Perspective" (2017), 508 pages
17. Tom Coleman, Sacha Greif, "Discover Meteor: Building Real-Time JavaScript Web Apps" (2015), 482 pages