НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

Інститут комп’ютерних систем

Кафедра інженерії програмного забезпечення

КУРСОВИЙ ПРОЕКТ

Розробка інформаційних систем

на тему: Міський водопровід

Студентки Андрусенко О. В.

групи АС-221

Спеціальність 121 Інженерія програмного забезпечення

Керівник Кунгурцев О.Б.

Доцент

Одеса - 2024

**АНОТАЦІЯ**

У цій курсовій роботі було розроблено графічний додаток на мові програмування C#. Суть розробки полягає в створенні інформаційної системи, яка відповідає вимогам, визначеним у завданні. Самостійна робота студента в рамках дисципліни «Аналіз вимог до програмного забезпечення» включає розробку вимог до інформаційної системи, предметна галузь якої коротко визначена в завданні. В рамках дисципліни «Конструювання програмного забезпечення» студенти виконують цикл лабораторних робіт з побудови модулів програмного продукту відповідно до вимог, сформульованих під час вивчення дисципліни «Аналіз вимог до програмного забезпечення». В ході виконання курсового проекту з дисципліни «Розробка інформаційних систем» студенти завершують створення програмних модулів, розробляють відповідну базу даних і підтверджують виконання вимог системним тестуванням.

Метою даного проекту є практичне застосування теоретичних знань для побудови повноцінного програмного продукту, який забезпечить авторизацію користувачів, введення та редагування даних в базі даних, отримання даних на основі запитів, реєстрацію нових клієнтів, подання заявок на ремонтні роботи, призначення бригад на ремонтні роботи, а також закупівлю та списання матеріалів.

Завданням проектування є створення функціонального, надійного та зручного у використанні програмного забезпечення, яке дозволить автоматизувати діяльність у визначеній предметній області, де раніше не використовувались подібні системи автоматизації.

**ЗМІСТ**

**1 ВСТУП**

**2 ВИЗНАЧЕННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ**

**2.2 Документ «Бачення»**

**2.2.1 Введення**

На цьому етапі маємо програмне забезпечення для міського водопроводу, головна ціль якого є автоматизація таких процесів як реєстрація клієнтів, підключення до водопровідної мережі, розрахунки зі споживачами води, перегляд та редагування даних, організація ремонтних робіт, спрямованих на усунення аварій, закупівля та фіксування списаних матеріалів для виконання ремонтних робіт та підключення.

Автоматизація цих процесів дозволить безперебійно працювати в системі водопостачання, зменшить суттєво ризик непорозумінь між системою та зацікавленими особами.

**2.2.2 Зацікавлені особи та їх потреби**

До зацікавлених осіб в зазначеній предметній області відноситься менеджер системи водопостачання та клієнт.

Потреба менеджера полягає у можливості переглядати дані про клієнтів, формувати розрахунки за воду, призначати бригади для виконання ремонтних робіт, закупівля матеріалів та їх списання.

Потреба клієнта полягає у можливості реєстрації в системі та внесення змін в особисті дані, подання заявки на підключення до системи, подання заявки щодо аварії водопроводу, переглядати відповідні заявки та розраховуватись з платежами.

**2.2.3 Основні виконавці**

У системі міського водопостачання виявлено два основних виконавця: менеджер системи та клієнт.

Відповідні профілі для кожного з виконавців, де розглянуто, яку позицію займає в системі, їх відповідальності та критерії, які надаватимуть успіх.

Таблиця 2.1 – “Профіль Менеджера”

|  |  |
| --- | --- |
| Типовий представник | Менеджер |
| Опис | Менеджер займається керуванням всією діяльністю організації з обслуговування водопостачання. |
| Тип | Керівник |
| Відповідальності | Внесення додаткової інформації в заявках, формування рахунків за воду, призначення бригад для виконання ремонтних робіт, здійснення закупівель та контроль списання матеріалів. |
| Критерій успіху | Ефективне управління ресурсами та персоналом, задоволення клієнтів та забезпечення надійності системи, своєчасне вирішення проблем та звернень клієнтів |

Таблиця 2.2 – “Профіль Клієнта”

|  |  |
| --- | --- |
| Типовий представник | Клієнт |
| Опис | Є споживачем послуг водопостачання та має потребу в наданні цих послуг. Він може бути приватним споживачем або представником організації. |
| Тип | Користувач |

Продовження таблиці 2.2

|  |  |
| --- | --- |
| Відповідальності | Реєстрація в системі та зберігання актуальних особистих даних, подання заявки на підключення до системи водопостачання, подання звернень щодо аварій, оплата рахунків за воду. |
| Критерій успіху | Своєчасне внесення інформації та заявок в систему, задоволення власних потреб у водопостачанні. |

**2.2.3 Загальні вимоги до системи**

У цьому пункті буде пояснено, яких вимог система повинна дотримуватись.

Таблиця 2.3 – Загальні вимоги до системи

|  |  |
| --- | --- |
| Можливість реєстрації користувачів | Система повинна дозволяти клієнтам та менеджерам реєструватися в системі з обліком рівня доступу та прав користувачів. |
| Облік клієнтів та розрахунків | Система повинна зберігати інформацію про клієнтів та їх розрахунки за спожиту воду. Можливість внесення змін до особистих даних клієнта. |
| Керування ремонтними роботами | Система повинна дозволяти менеджерам призначати бригади для виконання ремонтних робіт. Облік матеріалів, використаних під час ремонтних робіт. |

Продовження таблиці 2.3

|  |  |
| --- | --- |
| Можливість отримання інформації за запитами | Користувачам повинна надаватися можливість здійснювати запити до системи для отримання інформації про свої розрахунки, ремонтні роботи тощо. |
| Авторизація користувачів з різними рівнями доступу | Система повинна мати можливість авторизації користувачів з різними рівнями доступу (наприклад, клієнти, менеджери, робітники). Надання відповідних прав доступу залежно від ролі користувача. |

**2.2.4 Словник предметної області**

У цьому пункті складено словник предметної області «Міський водопровід», де зібрано базові поняття по даній темі для загального розуміння про що йдеться в документі.

Словник предметної області:

- Клієнт: Особа або організація, яка користується послугами системи водопостачання.

- Менеджер: Особа, відповідальна за управління та координацію діяльності водопостачальної системи.

- Бригада: Група працівників, яка призначається для виконання ремонтних або обслуговуючих робіт.

- Постачальник: Організація або постачальник, який постачає матеріали для водопостачання.

- Розрахунки з клієнтом: Облік та розрахунок вартості спожитої води клієнтом.

- Матеріал: предмет, який використовується під час ремонтних робіт або підключення до водопостачальної мережі.

- Підключення до водопровідної мережі: Процес приєднання нового споживача до водопровідної системи.

- Система керування водопостачанням: Програмне забезпечення, розроблене для автоматизації процесів постачання води та управління бізнесом.

- Заявка на ремонт або аварію: Повідомлення від клієнта про несправності або аварії в системі водопостачання.

- Авторизація користувачів: Процес перевірки та підтвердження ідентифікації користувача для надання доступу до системи.

- Списання матеріалів: Облік та відображення в системі використаних матеріалів під час ремонтних робіт.

**2.2.5 Діаграма прецедентів**

У цьому пункті зображено діаграму прецедентів та акторів системи, які мають певні можливості для взаємодії з системою (див. рис. 2.5.1).



Рисунок 2.5.1 – Діаграма прецедентів

На рис. 2.5.1 зображено функції системи, які вона може виконати та її акторів.

**2.2.6 Програмний модуль, що підлягає реалізації**

У цьому пункті позначені можливості та функції розроблюваної системи, які реалізовані студентом практично.

До програмного модуля, що підлягає реалізації, входить:

1. Можливість авторизації користувачів та розподілення можливостей та прав доступу відповідно до їх повноважень.

2. Можливість введення та редагування даних в базі даних.

3. Можливість отримання даних на основі запитів до бази даних.

4. Функціональність для реєстрації нових клієнтів.

5. Функціональність для подання заявок на ремонтні роботи та усунення аварій.

6. Функціональність для призначення бригад на ремонтні роботи.

7. Функціональність для закупівлі та списання матеріалів.

**2.2.7 Опис двох прецедентів, відповідно до програмного модулю**

У цьому пункті описано про два важливі прецеденти для системи «Міський водопровід».

Важливі прецеденти для системи, які були взяті: підключення клієнта до водопостачальної системи та усунення аварій. Далі бути йти мова про опис прецедентів:

Прецедент №1

Назва прецеденту: підключення клієнта до водопостачальної системи.

Основне діюче обличчя: менеджер.

Область дії: система керування водопостачанням.

Учасники й інтереси:

- Менеджер – бажає підключити нового клієнта до водопостачальної системи;

- Клієнт – бажає підключитися до системи водопостачання.

Передумова: Менеджер авторизований у системі.

Мінімальні гарантії: клієнт отримує відповідь про можливість або неможливість підключення до системи.

Гарантія успіху: клієнт успішно підключений до системи водопостачання.

Тригер: клієнт бажає підключитися до водопостачальної системи та заповнює заявку.

Основний сценарій:

1. Клієнт входить у систему керування водопостачанням. Система підтверджує вхід.

2. Клієнт заповнює заявку на підключення до водопостачальної системи. Система приймає заявку та перевіряє дані клієнта, включаючи зону обслуговування. Система підтверджує, що клієнт знаходиться в зоні обслуговування. Система реєструє заявку на підключення.

3. Менеджер переглядає заявку на підключення. Система надає дані щодо заявки на підключення.

4. Менеджер запускає форму редагування заявки. Система надає форму для редагування.

5. Менеджер визначає вільну дату та час. Система фіксує заплановану дату та час проведення робіт з підключення.

6. Менеджер призначає бригаду для виконання робіт з підключення. Система перевіряє чи вільна бригада в заплановану дату та час.

7. Менеджер перевіряє наявність необхідних матеріалів для підключення. Система надає інформацію про ці матеріали.

8. Менеджер фіксує списання матеріалів для виконання робіт. Система зберігає дані про списані матеріали.

9. Менеджер відправляє рахунок за заявку клієнту. Система формує рахунок клієнту. Клієнт оплачує рахунок за заявку на підключення.

Розширення:

2а. Система повідомляє клієнта, що підключення неможливе через відсутність технічних можливостей або зону обслуговування.

2а1. Клієнт отримує повідомлення про неможливість підключення. Прецедент завершений.

6a. Система повідомляє менеджера про відсутність доступних бригад для виконання робіт.

6a1. Менеджер повідомляє клієнта про затримку у виконанні робіт. Прецедент завершений.

7б. Система повідомляє менеджера про відсутність необхідних матеріалів.

7б1. Менеджер замовляє необхідні матеріали. Повернення до пункту 7 після отримання матеріалів.

Прецедент №2

Назва прецеденту: усунення аварії.

Основне діюче обличчя: менеджер.

Область дії: система керування водопостачанням.

Учасники й інтереси:

- Менеджер – бажає призначити бригаду для усунення аварії та списати використані матеріали.

- Клієнт – бажає повідомити про аварію та отримати її усунення.

Передумова: менеджер авторизований у системі.

Мінімальні гарантії: менеджер отримує підтвердження про призначення бригади та списання матеріалів.

Гарантія успіху: аварія успішно усунена, матеріали списані.

Тригер: клієнт подає заявку на усунення аварії.

Основний сценарій:

1. Клієнт входить у систему керування водопостачанням. Система підтверджує вхід.

2. Клієнт вибирає опцію подачі заявки на усунення аварії. Клієнт заповнює форму заявки. Клієнт надсилає заповнену заявку до системи. Система приймає заявку та реєструє її.

3. Система повідомляє менеджера про нову заявку на усунення аварії. Менеджер отримує заявку на усунення аварії. Менеджер призначає бригаду для виконання ремонтних робіт. Менеджер перевіряє наявність необхідних матеріалів для проведення ремонтних робіт.

4. Система генерує рахунок за ремонтні роботи. Клієнт оплачує рахунок.

5. Менеджер фіксує списання матеріалів для виконання робіт у системі.

6. Бригада виконує роботи по усуненню аварії.

7. Менеджер оновлює статус заявки на "Оновлено". Система фіксує завершення робіт та успішне виконання завдання. Система повідомляє клієнта про успішне усунення аварії.

Розширення:

3a. Система повідомляє менеджера про відсутність доступних бригад для виконання робіт.

3a1. Менеджер повідомляє клієнта про затримку у виконанні робіт. Прецедент завершений.

3б. Система повідомляє менеджера про відсутність необхідних матеріалів.

3б1. Менеджер замовляє необхідні матеріали. Повернення до пункту 7 після отримання матеріалів.

**3 МОДЕЛЬ КОНЦПЕПТУАЛЬНИХ КЛАСІВ**

**3.1 Аналіз концептуальних класів для прецеденту «Підключення клієнта до водопостачальної системи»**

У цьому підрозділі буде розібрано більше детально сценарії двох важливих прецедентів. Де на основі кожного кроку, який проходить користувач та система, визначиться клас, який буде виконувати ці кроки. Поки що буде розглянуто лише перший з прецедентів «Підключення клієнта до водопостачальної мережі».

Таблиця 3.1 - Аналіз концептуальних класів для прецеденту «Підключення клієнта до водопостачальної системи»

|  |  |
| --- | --- |
| Пункт прецеденту | Концептуальний клас |
| 1. Клієнт входить у систему керування водопостачанням. Система підтверджує вхід. | LoginForm |
| 2. Клієнт заповнює заявку на підключення до водопостачальної системи. Система приймає заявку та перевіряє дані клієнта, включаючи зону обслуговування. | UserForm |
| 3. Система підтверджує, що клієнт знаходиться в зоні обслуговування. Система реєструє заявку на підключення. | UserForm |
| 4. Менеджер переглядає заявку на підключення. | AdminForm |

Продовження таблиці 3.1

|  |  |
| --- | --- |
| 5. Менеджер призначає бригаду для виконання робіт з підключення. Менеджер перевіряє наявність необхідних матеріалів для підключення. Система фіксує заплановану дату та час проведення робіт з підключення. | Brigade, Material, ConnectionRequest |
| 6. Система генерує рахунок для клієнта. Клієнт оплачує рахунок. | Invoice, UserForm |
| 7. Менеджер фіксує списання матеріалів для виконання робіт у системі. | WriteOffMaterials |

**3.2 Аналіз концептуальних класів для прецеденту «Усунення аварії»**

У цьому підрозділі буде виявлено концептуальні класи для кожного кроку сценарію прецеденту «Усунення аварії».

Таблиця 3.2 - Аналіз концептуальних класів для прецеденту «Усунення аварії»

|  |  |
| --- | --- |
| 1. Клієнт входить у систему керування водопостачанням. Система підтверджує вхід. | LoginForm |
| 2. Клієнт вибирає опцію подачі заявки на усунення аварії. Клієнт заповнює форму заявки. Клієнт надсилає заповнену заявку до системи. Система приймає заявку та реєструє її. | UserForm |

Продовження таблиці 3.2

|  |  |
| --- | --- |
| 3. Система повідомляє менеджера про нову заявку на усунення аварії. Менеджер отримує заявку на усунення аварії. Менеджер призначає бригаду для виконання ремонтних робіт. Менеджер перевіряє наявність необхідних матеріалів для проведення ремонтних робіт. | AdminForm, Brigade, Material, EmergencyRepairRequest |
| 4. Система генерує рахунок за ремонтні роботи. Клієнт оплачує рахунок. | Invoice, UserForm |
| 5. Менеджер фіксує списання матеріалів для виконання робіт у системі. | WriteOffMaterials |
| 6. Бригада виконує роботи по усуненню аварії. | Brigade |
| 7. Менеджер оновлює статус заявки на "Оновлено". Система фіксує завершення робіт та успішне виконання завдання. Система повідомляє клієнта про успішне усунення аварії. | EmergencyRepairRequest |

**3.3 Діаграма концептуальних класів**

У відповідності до аналізу концептуальних даних, виходячи з даних прецедентів, нижче наведено діаграму концептуальних класів з вказанням зв’язку між класами.

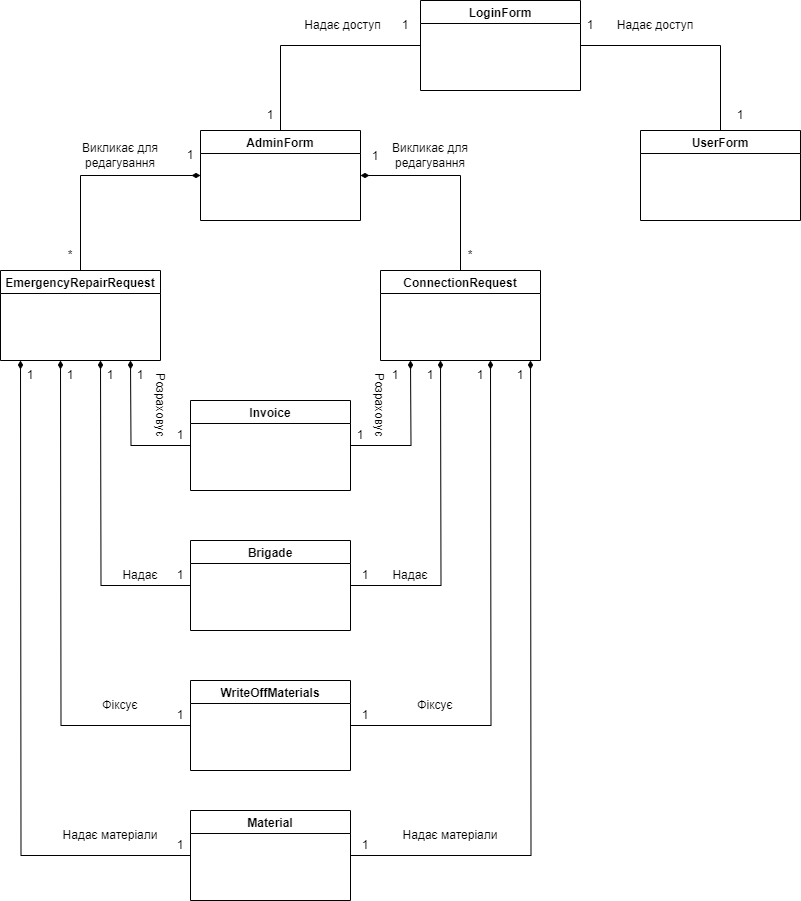


Рисунок 3.3.1 - Концептуальна модель класів

На цьому рис. 3.3.1 зображено концептуальні класи двох важливих прецедентів системи «Міського водопроводу». Між класами зображені зв’язки, які вказують на їх можливості та взаємодії.

**4 ДІАГРАМИ ВЗАЄМОДІЇ**

**4.1 Діаграма послідовностей**

Нижче наведено діаграму послідовностей для предметної області: Міський водопровід та прецеденту «Підключення клієнта до водопостачальної системи».

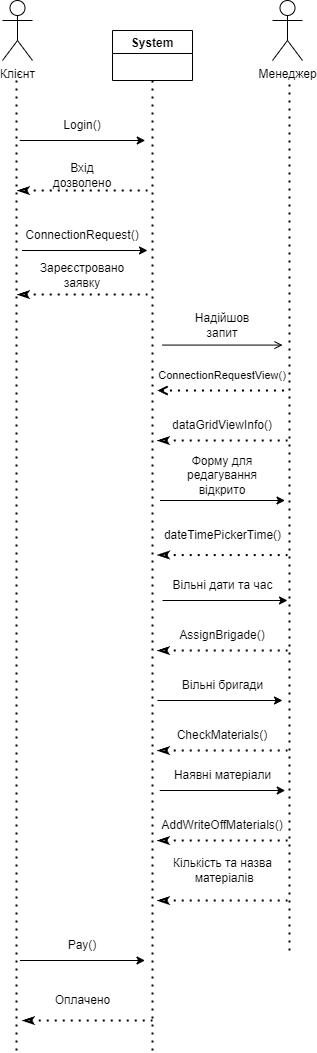


Рисунок 4.1.1 - Діаграма послідовностей

На рис. 4.1.1 зображено діаграму послідовностей, де вказана взаємодія акторів та системи, їх повідомлення між собою.

**4.2 Діаграми кооперації**

4.2.1 Проектне рішення Login

Клієнт входить у систему керування водопостачанням. Система підтверджує вхід.

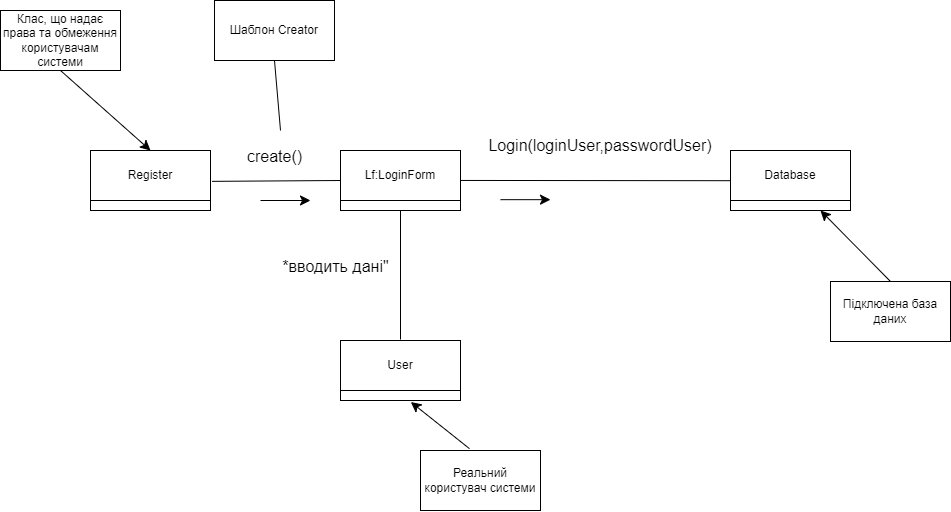


Рисунок 4.1 – Проектне рішення Login

4.2.2 Проектне рішення ConnectionRequest

Клієнт заповнює заявку на підключення до водопостачальної системи. Система приймає заявку та перевіряє дані клієнта, включаючи зону обслуговування. Система підтверджує, що клієнт знаходиться в зоні обслуговування. Система реєструє заявку на підключення.

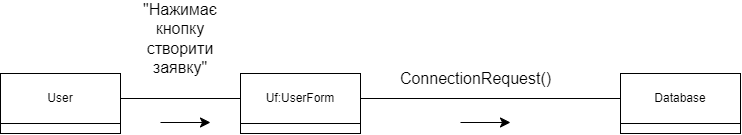


Рисунок 4.2 - Проектне рішення ConnectionRequest

4.2.3 Проектне рішення ConnectionRequestView

Менеджер переглядає заявку на підключення. Система надає дані щодо заявки на підключення.

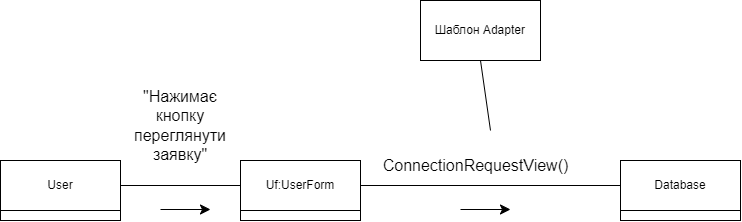


Рисунок 4.3 - Проектне рішення ConnectionRequestView

4.2.4 Проектне рішення dataGridInfoView

Менеджер запускає форму редагування заявки. Система надає форму для редагування.

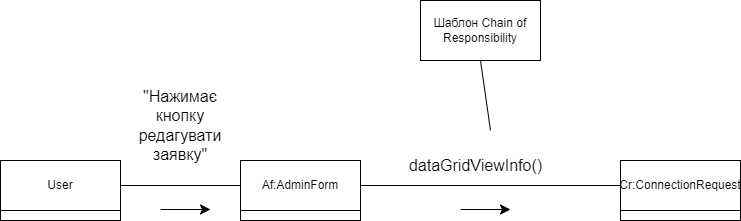


Рисунок 4.4 - Проектне рішення dataGridInfoView

4.2.5 Проектне рішення dateTimePickerTime

Менеджер визначає вільну дату та час. Система фіксує заплановану дату та час проведення робіт з підключення.

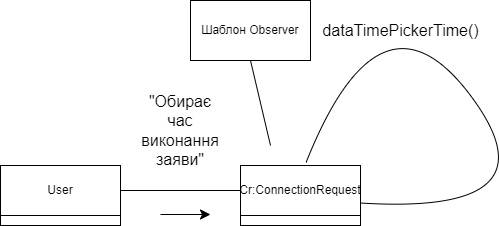


Рисунок 4.5 - Проектне рішення dataTimePickerTime

4.2.6 Проектне рішення AssignBrigade

Менеджер призначає бригаду для виконання робіт з підключення. Система перевіряє чи вільна бригада в заплановану дату та час.

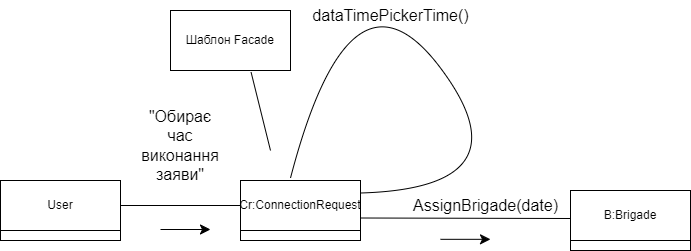


Рисунок 6 - Проектне рішення AssignBrigade

4.2.7 Проектне рішення CheckMaterials

Менеджер перевіряє наявність необхідних матеріалів для підключення. Система надає інформацію про ці матеріали.

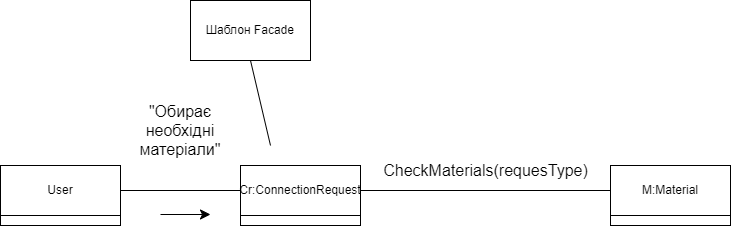


Рисунок 4.7 - Проектне рішення CheckMaterials

4.2.8 Проектне рішення AddWriteOffMaterials

Менеджер фіксує списання матеріалів для виконання робіт. Система зберігає дані про списані матеріали.

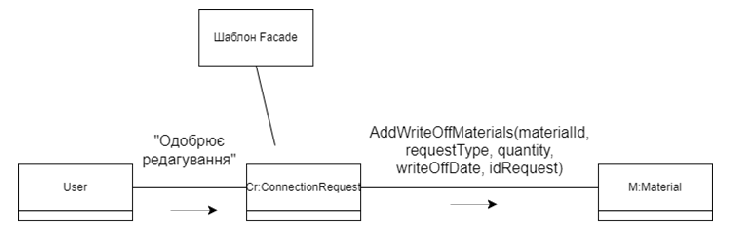


Рисунок 4.8 - Проектне рішення AddWriteOffMaterials

4.2.9 Проектне рішення Pay

Менеджер відправляє рахунок за заявку клієнту. Система формує рахунок клієнту. Клієнт оплачує рахунок за заявку на підключення.

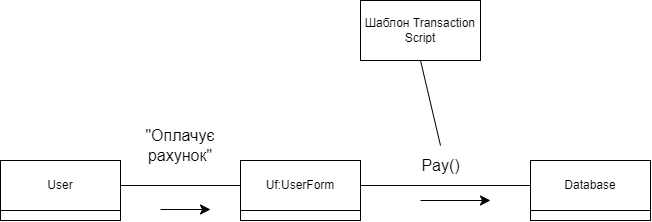


Рисунок 4.9 - Проектне рішення Pay

**5 МОДЕЛЬ ДАНИХ**

**5.1 Аналіз предметної області**

Після проведення аналізу предметної області «Міський водопровід», було сформульовано набір функцій, які мають бути виконані системою (див. табл. 5.1).

Таблиця 5.1 – Технічні вимоги «Міський водопровід»

|  |  |
| --- | --- |
| Вимоги замовника | Технічні вимоги |
| 1. Інформація про клієнтів | 1.1 Інформація про клієнтів містить: прізвище, ім’я та по-батькові клієнта, номер телефону, адресу проживання. |
| 2. Інформація про робітників | 2.1 Інформація про робітників містить: прізвище, ім’я та по-батькові робітника, номер телефону, дату працевлаштування, дату звільнення. |
| 3. Інформація про авторизацію клієнтів та робітників | 3.1 Інформація про авторизацію користувачів системи містить: логін та пароль користувача. |
| 4. Інформації про бригади | 4.1 Інформація про бригади містить: дату створення бригади. |
| 5. Інформація про посади | 5.1 Інформація про посади містить: назву посади, опис посади. |
| 6. Інформація про заявки на підключення до системи | 6.1 Інформація про заявки на підключення до системи містить: дату створення, статус заявки, деталі, дату виконання, інформацію про бригаду (id), суму заявки, номер телефону та адресу проживання клієнта. |

Продовження таблиці 5.1

|  |  |
| --- | --- |
| 7. Інформація про заявки на усунення аварій | 7.1 Інформація про заявки на усунення аварій містить: дату створення, статус заявки, деталі, дату виконання, інформацію про бригаду (id), суму заявки, номер телефону та адресу проживання клієнта. |
| 8. Інформація про матеріали | 8.1 Інформація про матеріали містить:  назву матеріалу. |
| 9. Інформація про матеріали для заявок | 9.1 Інформація про матеріали для заявок містить: кількість матеріалу. |
| 10. Інформація про постачальників | 10. Інформація про постачальників містить: ім'я, номер телефону та адресу знаходження. |
| 11. Інформація про фіксовані ціни за заявки | 11. Інформація про фіксовані ціни за заявки містить: суму заявки. |
| 12. Інформація про рахунки споживання води | 12. Інформація про рахунки споживання води містить: дату створення, ціну за куб води, кількість кубів води, дату оплати, статус рахунку. |
| 13. Інформація про списання матеріалів | 13. Інформація про списанні матеріали містить: тип заявки, кількість матеріалу, дату списання. |
| 14. Інформація про закупівлі матеріалів | 14. Інформація про закупівлі матеріалів містить: кількість матеріалу, дату закупівлі, ціну. |

**5.2 Концептуальна модель даних**

Виходячи з предметної області «Міський водопровід», мною була розроблена відповідна концептуальна модель даних (див. рис. 5.2), яка буде враховувати всі необхідності системи для зазначеної предметної області.

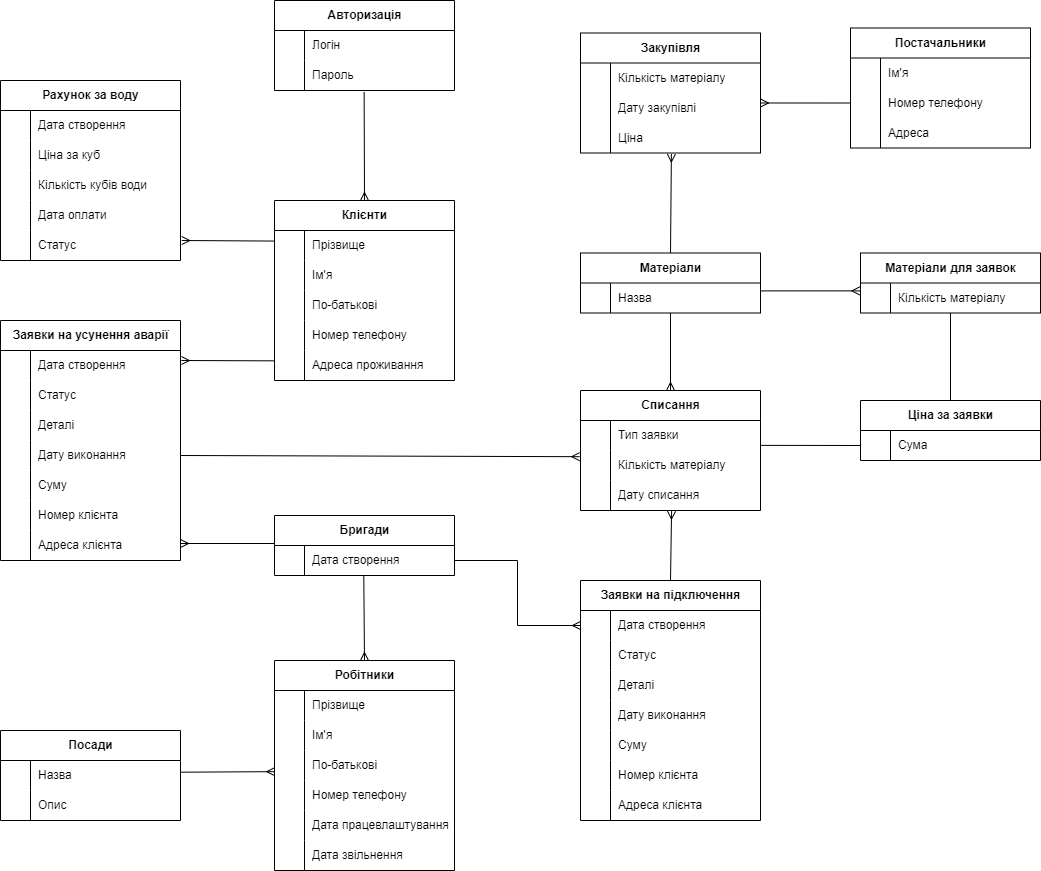


Рисунок 5.2 - Концептуальна модель даних системи "Міського водопроводу"

На рис. 5.2 зображено концептуальну модель даних, на ній показано зв’язки між таблицями створюваної інформаційної системи.

**5.3 Реляційна модель даних**

Беручи за основу концептуальну модель даних відповідної предметної області, мною було розроблено реляційну модель даних (див. рис 5.3).

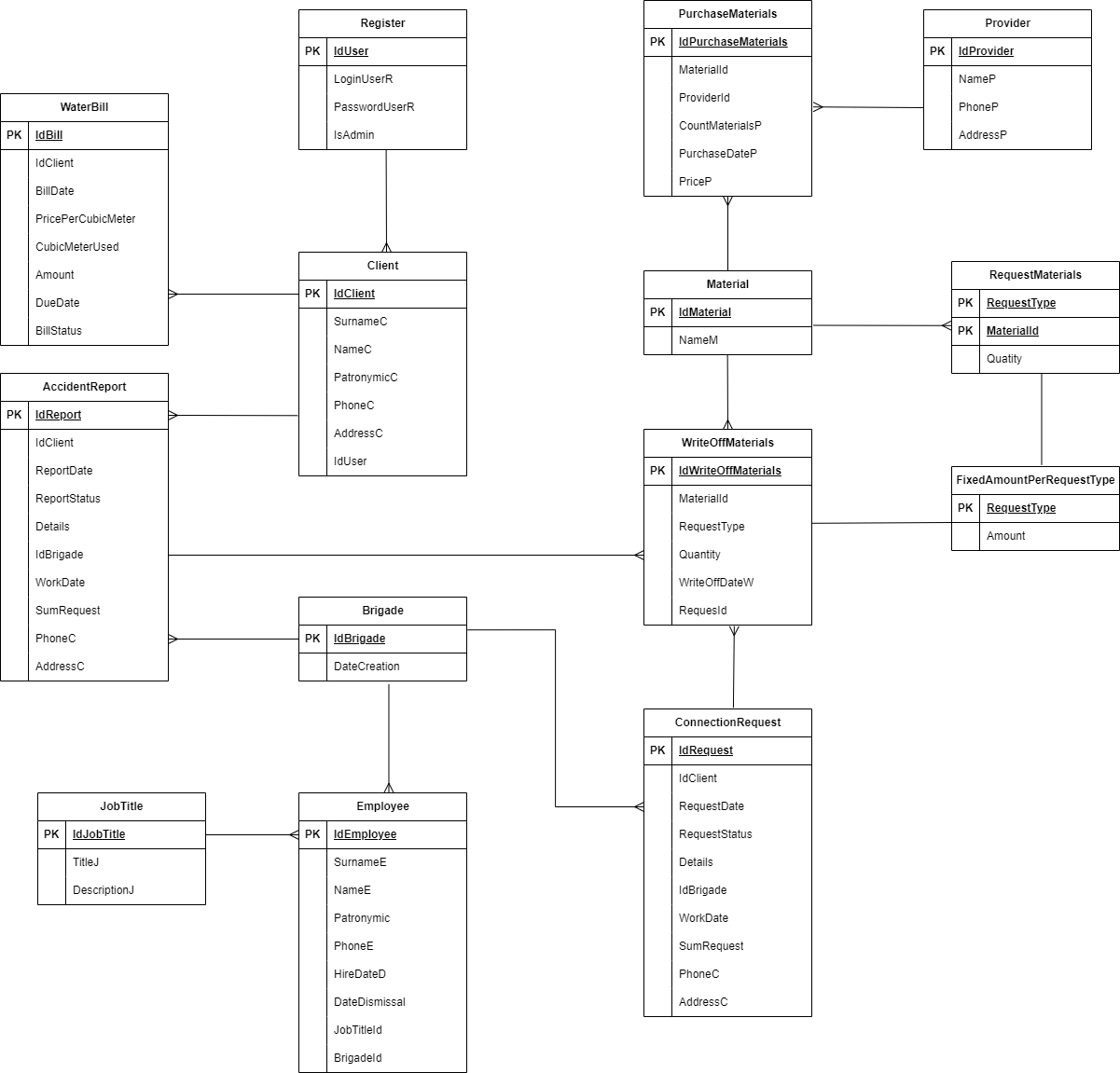


Рисунок 5.3 - Реляційна модель для системи "Міського водопроводу"

На рис. 5.3 зображено реляційну модель, де показано більш детально як пов’язані між собою таблиці створюваної інформаційної системи.

**6 ДІАГРАМА ПРОГРАМНИХ КЛАСІВ**

**6.1 Аналіз діаграм взаємодій для побудови програмних класів**

У цьому підрозділі буде показано вміст програмних класів, які формують інформаційну систему «Міський водопровід».

Проаналізувавши всі діаграми взаємодії, маємо такі діаграми класів:

- клас LoginForm

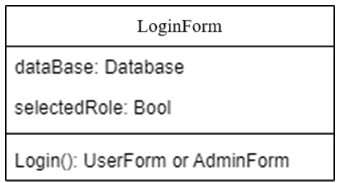


Рисунок 6.1 - Програмний клас LoginForm

- клас UserForm



Рисунок 6.2 - Програмний клас UserForm

- клас AdminForm

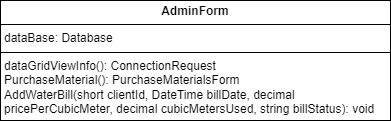


Рисунок 6.3 - Програмний клас AdminForm

- клас ConnectionRequest

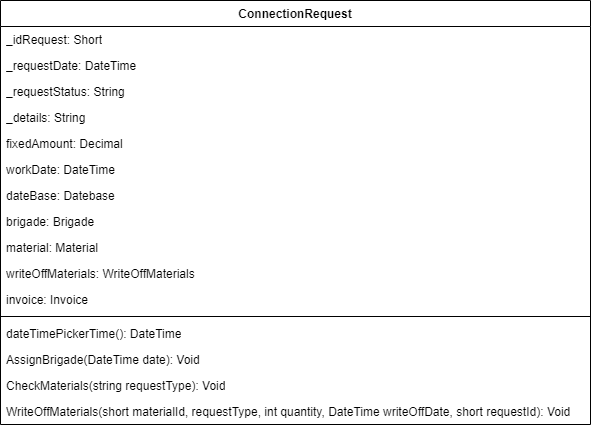


Рисунок 6.4 - Програмний клас ConnectionRequest

- клас EmergencyRepairRequest

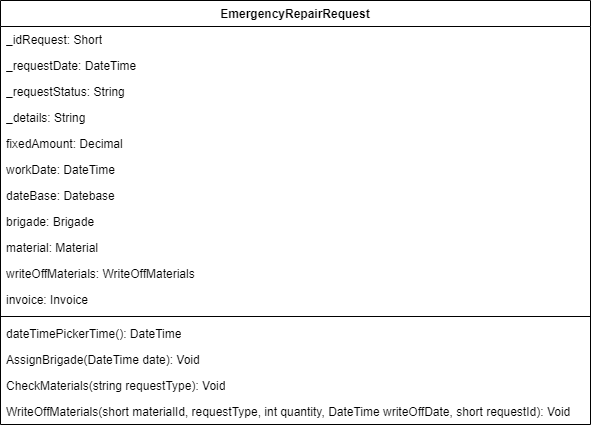


Рисунок 6.5 - Програмний клас EmergencyRepairRequest

- клас Invoice

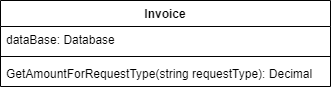


Рисунок 6.6 - Програмний клас Invoice

- клас Brigade



Рисунок 6.7 - Програмний клас Brigade

- клас WriteOffMaterials

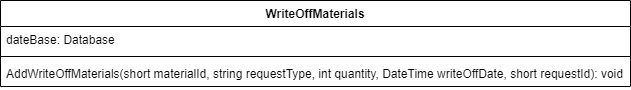


Рисунок 6.8 - Програмний клас WriteOffMaterials

- клас Material



Рисунок 6.9 - Програмний клас Material

**6.2 Діаграма програмних класів**

Після аналізу та побудови діаграм для кожного класу інформаційної системи «Міський водопровід». Можемо побудувати загальну діаграму класів, де будуть зображені зв’язки між класами.

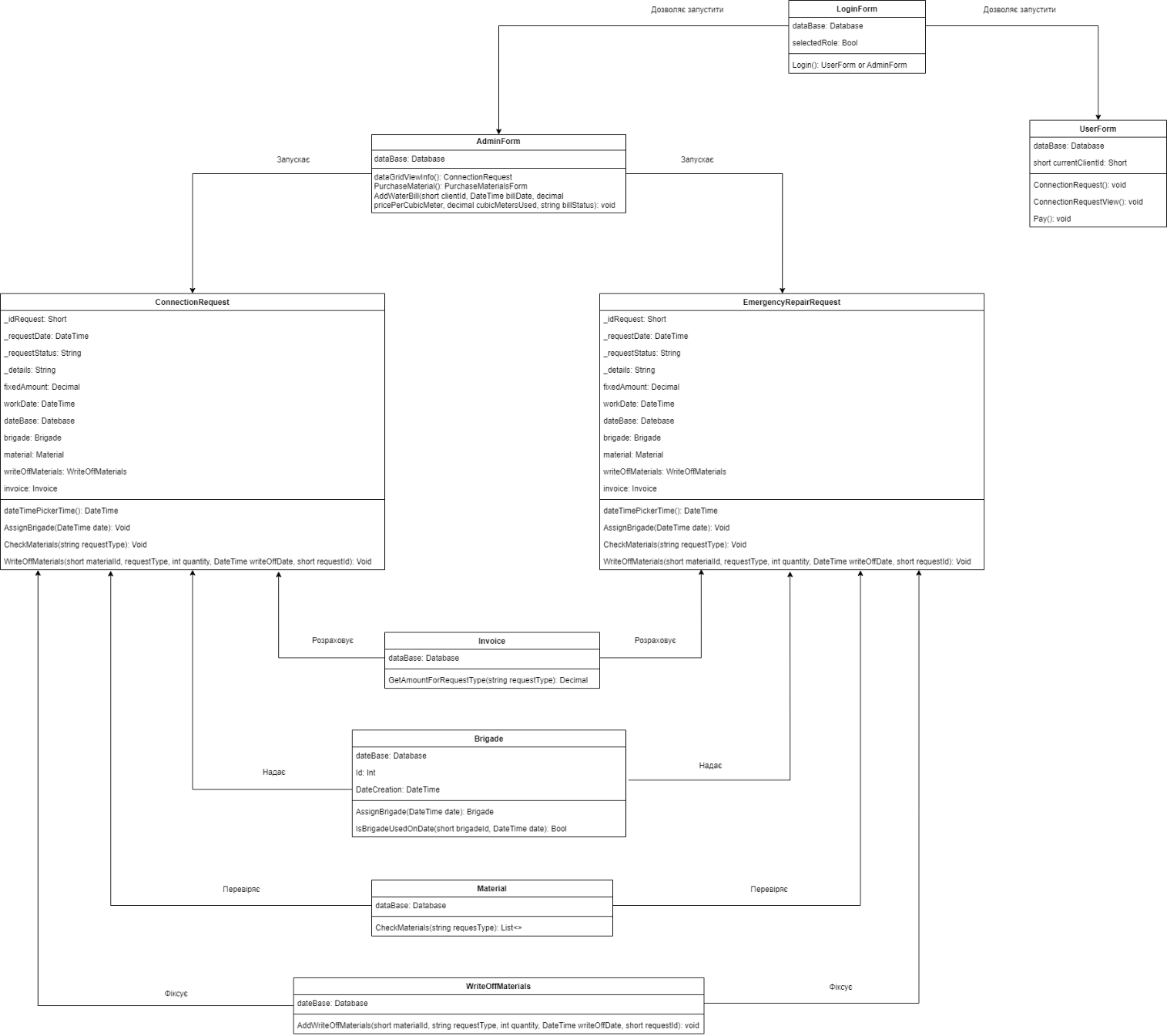


Рисунок 6.10 - Загальні діаграма програмних класів

На рис. 6.10 зображено діаграму програмних класів. Кожен клас має зв’язок з іншим, яких поєднує лінія.

Після аналізу та побудови діаграм, розберемо детально кожен клас:

- LoginForm: відповідає за авторизацію користувачів. Має атрибути dataBase (об’єкт класу бази даних) та selectedRole (обрана роль). Методи: Login().

Обрані атрибути відіграють таку роль, де об’єкт бази даних надає доступ до створеної бази даних для розроблюваної системи, та «обрана роль» відповідає за те, які права має користувач у цій системі. Метод реалізовує перевірку даних, чи є вони валідними, аби увійти в систему. Клас LoginForm має агрегацію з класами UserForm та AdminForm. Так як клас для авторизації використовує об’єкти класів, перечислених вище, аби відкрити їх форми(вікна). Відповідно життєвий цикл не залежить одне від одного.

- UserForm: клас, який відповідає за взаємодію між користувачем та системою. Має такі атрибути database (об’єкт класу бази даних) та currentClientId (ідентифікатор клієнта). Методи: ConnectionRequest(), ConnectionRequestView() та Pay().

Обрані атрибути відіграють таку роль, де об’єкт бази даних надає доступ до створеної бази даних для розроблюваної системи, та «ідентифікатор клієнта» визначає певного користувача, який взаємодіє з системою. Методи відіграють таку роль, де ConnectionRequest() додає запис до бази даних про створену заявку на підключення до водопостачальної системи. ConnectionRequestView() дозволяє користувачеві переглянути дані створеної заявки. Pay() відповідає за оплату рахунків, які має користувач. Клас UserForm немає зв’язку ні з ким, окрім LoginForm.

- AdminForm: клас, який відповідає за взаємодію користувача з системою. Має такий атрибут database (об’єкт класу бази даних). Методи: dataGridViewInfo(),

PurchaseMaterial() та AddWaterBill().

Обраний атрибут відіграє за зв’язок між системою та базою даних. Методи відіграють таку роль, де dataGridViewInfo() запускає вікна для редагування запитів, які надіслав клієнт. PurchaseMaterial() відповідає за закупівлю матеріалів, якщо необхідні закінчились. AddWaterBill() створює рахунок за воду клієнтам системи. Має зв’язки з класами EmergencyRepairRequest та ConnectionRequest, ідентифікувати можна як композиція. Бо перечислені класи мають короткий життєвий цикл та не можуть існувати без класу AdminForm.

- EmergencyRepairRequest: клас, який є вікном редагування заявки на усунення аварії. Має такі атрибути \_idRequest (ідентифікатор заявки), \_requestDate (дата створення заявки), requestStatus (статус заявки), \_details (деталі заявки), fixedAmount (фіксована сума за заявку), workDate (дата виконання), dataBase (об’єкт класу бази даних), bridage (об’єкт класу бригади), material (об’єкт класу матеріалу), writeOffMaterials (об’єкт класу списання) та invoice (об’єкт класу рахунок). Методи: dateTimePickerTime(), AssignBrigade(), CheckMaterials() та WriteOffMaterials().

Обрані атрибути відіграють таку роль, де «ідентифікатор заявки» відповідає за збереження ідентифікатору заявки, «дата створення заявки» зберігає дату створення, «статус заявки» зберігає статус, «деталі заявки» зберігають інформацію про заявку, «фіксована сума заявки» зберігатиме суму за певний тип заявки, «дата виконання» зберігає дату виконання, «об’єкт бази даних» зв’язує програму та базу даних, «об’єкт класу бригади» допомагає викликати функцію з класу бригади, «об’єкт класу матеріалу» допомагає викликати функцію з класу матеріал, «об’єкт класу списання» допомагає викликати функцію з класу списання та «об’єкт класу рахунок» допомагає викликати функцію з класу рахунок. Методи відіграють таку роль, де dateTimePickerTime() реагує на взаємодію користувача з обранням дати виконання заявки. AssignBrigade() звертається за допомогою «об’єкту класу бригада» до класу бригади для отримання вільних бригад в обраний час користувачем. CheckMaterials() викликається «об’єктом класу матеріал», відповідно клас матеріал повертає список матеріалів за типом заявки. WriteOffMaterials() за допомогою «об’єкту класу списання» викликає метод класу списання для запису обраних матеріалів користувачем та видалення їх з таблиці, яка містить список необхідних матеріалів. EmergencyRepairRequest має зв’язки з класами Brigade, Material, WriteOffMaterials та Invoice. Відповідно ці зв’язки можна ідентифікувати як композиція.

- ConnectionRequest: клас, який є вікном редагування заявки на підключення до системи.   
 Так як клас ConnectionRequest ідентичний з класом EmergencyRepairRequest, тому опис атрибутів та методів описані в класі EmergencyRepairRequest.

- Invoice: клас, який відповідає за отримання суми на основі типу заявки клієнта. Має такий атрибут dataBase (об’єкт класу бази даних). Метод: GetAmountForRequestType().

Обраний метод відповідає за отримання суми на основі типу заявки. Invoice немає зв’язку, окрім класів EmergencyRepairRequest та ConnectionRequest, з якими має зв’язок композиції.

- Brigade: клас, який відповідає за бригади в системі. Має такі атрибути dataBase (об’єкт класу бази даних), id (ідентифікатор бригади) та DateCreation (дата створення бригади). Методи: AssignBrigade() та IsBrigadeUsedOnDate().

Обрані атрибути відіграють таку роль, де «ідентифікатор бригади» зберігає ідентифікатор, «дата створення бригади» зберігає дату створення. Методи відіграють так роль, де AssignBrigade() знаходить всі бригади в базі даних та передає отримані дані до методу IsBrigadeUsedOnDate(), який у свою чергу перевіряє чи є вільні бригади у зазначений день користувачем. Brigade немає зв’язку, окрім класів EmergencyRepairRequest та ConnectionRequest, з якими має зв’язок композиції.

- WriteOffMaterials: клас, який фіксує списування матеріалів на виконання певної заявки. Містить такий атрибут dataBase (об’єкт класу бази даних) та метод: AddWriteOffMaterials().

Обраний метод AddWriteOffMaterials() записує дані про певну заявку та про списані матеріали. WriteOffMaterials не містить зв’язків, окрім EmergencyRepairRequest та ConnectionRequest, з якими має зв’язок композиції.

- Material: клас, який надає список необхідних матеріалів. Має такий атрибут dataBase (об’єкт класу бази даних) та метод CheckMaterials (надає список необхідних матеріалів). Material не містить зв’язків, окрім EmergencyRepairRequest та ConnectionRequest, з якими має зв’язок композиції.

**7 ПРОГРАМНІ КЛАСИ**

**7.1 Програмний код до класу LoginForm**

У цьому підрозділі буде показано загальний вигляд класу LoginForm в програмному коді. Лише будуть основні методи, які згадуються в цій роботі. Весь код програмного класу можна переглянути в Додатку А.

Код класу LoginForm:

- атрибути та деякі елементи в класі;

public partial class LoginForm : Form

{

Database dataBase = new Database();

private bool selectedRole;

public LoginForm()

{

InitializeComponent();

// Вирівнюємо текст-бокс з паролем

this.textBoxLock.AutoSize = false;

this.textBoxLock.Size = new Size(this.textBoxUser.Width, 32);

// Підписуємося на подію завантаження форми

this.Shown += new EventHandler(LoginForm\_Shown);

}

public LoginForm(bool selectedRole)

{

InitializeComponent();

this.selectedRole = selectedRole;

// Вирівнюємо текст-бокс з паролем

this.textBoxLock.AutoSize = false;

this.textBoxLock.Size = new Size(this.textBoxUser.Width, 32);

// Підписуємося на подію завантаження форми

this.Shown += new EventHandler(LoginForm\_Shown);

}

Поданий код вище показує, які атрибути та елементи присутні в класі. Також зазначимо, що цей клас містить вікно для взаємодії з користувачем системи.

- метод Login() виконаний так у програмному коді;

// Запит до бази даних, авторизація до програми

private void buttonLogin\_Click(object sender, EventArgs e)

{

var loginUser = textBoxUser.Text;

var passwordUser = textBoxLock.Text;

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

string isAdminValue = selectedRole ? "1" : "0";

string queryString = $"SELECT iduser, loginuserr, passworduserr, isadmin FROM register WHERE loginuserr = '{loginUser}' AND passworduserr = '{passwordUser}' AND isadmin = {isAdminValue}";

SqlCommand command = new SqlCommand(queryString, dataBase.getConnection());

adapter.SelectCommand = command;

adapter.Fill(table);

if (table.Rows.Count == 1)

{

short userId = Convert.ToInt16(table.Rows[0]["iduser"]); // Отримати ідентифікатор користувача з результатів запиту

SqlConnection connection = dataBase.getConnection();

// Відкриваємо підключення до бази даних

dataBase.openConnection(connection);

// Здійснюємо другий запит для отримання ідентифікатора клієнта

string clientQueryString = $"SELECT IdClient FROM Client WHERE IdUser = {userId}";

SqlCommand clientCommand = new SqlCommand(clientQueryString, connection);

short clientId = Convert.ToInt16(clientCommand.ExecuteScalar());

// Закриваємо підключення до бази даних після виконання запиту

dataBase.closeConnection(connection);

if (selectedRole == false)

{

MessageBox.Show("Авторизація пройшла!", "Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);

UserForm userForm = new UserForm(clientId);

this.Hide();

userForm.ShowDialog();

this.Close();

}

else

{

MessageBox.Show("Авторизація пройшла!", "Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);

AdminForm adminForm = new AdminForm();

this.Hide();

adminForm.ShowDialog();

this.Close();

}

}

else

{

MessageBox.Show("Обліковий запис не існує для даної ролі!", "Помилка авторизації!", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

Поданий код вище показує метод Login(), де можна побачити витягання даних введених користувачем, відповідно ці дані йдуть у базу даних для перевірки їх валідності. Якщо дані не валідні, отримаємо відповідне повідомлення, інакше користувач матиме доступ до іншого вікна.

**7.2 Програмний код до класу UserForm**

У цьому підрозділі буде показано загальний вигляд класу UserForm в програмному коді. Лише будуть основні методи, які згадуються в цій роботі. Весь код програмного класу можна переглянути в Додатку А.

Код класу LoginForm:

- атрибути та деякі елементи в класі;

public partial class UserForm : Form

{

private short currentClientId;

private Database dataBase = new Database();

public UserForm(short clientId)

{

currentClientId = clientId;

InitializeComponent();

}

Поданий код вище показує, які атрибути та елементи присутні в класі UserForm.

- метод ConnectionRequest() виглядає так у програмному коді;

// Заявка на підключення до системи

private void buttonConnectionRequest\_Click(object sender, EventArgs e)

{

string details = "Підключення до системи водопостачання";

SqlConnection connection = dataBase.getConnection();

dataBase.openConnection(connection);

using (connection)

{

SqlTransaction transaction = connection.BeginTransaction();

try

{

// Отримання адреси та номера телефону клієнта за його ID

string getAddressPhoneQuery = "SELECT AddressC, PhoneC FROM Client WHERE IdClient = @idClient";

SqlCommand getAddressPhoneCommand = new SqlCommand(getAddressPhoneQuery, connection, transaction);

getAddressPhoneCommand.Parameters.AddWithValue("@idClient", currentClientId);

SqlDataReader reader = getAddressPhoneCommand.ExecuteReader();

string address = "";

string phone = "";

if (reader.Read())

{

address = reader["AddressC"].ToString();

phone = reader["PhoneC"].ToString();

}

reader.Close();

// Запит на створення заявки

string insertRequestQuery = @"

INSERT INTO ConnectionRequest (IdClient, RequestDate, RequestStatus, Details, PhoneC, AddressC)

VALUES (@idClient, @requestDate, @status, @details, @phone, @address)";

SqlCommand commandRequest = new SqlCommand(insertRequestQuery, connection, transaction);

commandRequest.Parameters.AddWithValue("@idClient", currentClientId);

commandRequest.Parameters.AddWithValue("@requestDate", DateTime.Now);

commandRequest.Parameters.AddWithValue("@status", "Новий");

commandRequest.Parameters.AddWithValue("@details", details);

commandRequest.Parameters.AddWithValue("@phone", phone);

commandRequest.Parameters.AddWithValue("@address", address);

commandRequest.ExecuteNonQuery();

transaction.Commit();

MessageBox.Show("Заявка успішно створена!", "Успішно", MessageBoxButtons.OK, MessageBoxIcon.Information);

buttonConnectionRequest.Enabled = false; // Зробити кнопку недоступною після успішного створення заявки

}

catch (Exception ex)

{

transaction.Rollback();

MessageBox.Show("Не вдалося створити заявку! " + ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

finally

{

dataBase.closeConnection(connection);

}

}

}

Поданий код вище зображує створення заявки на підключення до системи. Отримуємо відповідні дані про клієнта та записуємо в базі даних у таблицю, яка відповідає за заявки на підключення.

- метод ConnectionRequestView() виглядає так у програмному коді;

// Показ записів з бд на DataGridView (Заявка на підключення)

private void buttonConnectionRequestView\_Click(object sender, EventArgs e)

{

LoadConnectionRequests(dataGridViewInfo);

}

private void LoadConnectionRequests(DataGridView dgw)

{

using (SqlConnection connection = dataBase.getConnection())

{

dataBase.openConnection(connection);

string query = @"

SELECT

CR.RequestDate,

CR.RequestStatus,

CR.Details,

CR.IdBrigade,

CR.WorkDate,

CR.SumRequest,

C.SurnameC,

C.NameC,

C.PatronymicC,

CR.PhoneC,

CR.AddressC

FROM

ConnectionRequest CR

INNER JOIN

Client C ON CR.IdClient = C.IdClient

WHERE

CR.IdClient = @idClient";

SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@idClient", currentClientId);

SqlDataReader reader = command.ExecuteReader();

dgw.DataSource = null; // Очистимо дані, щоб видалити попередні записи

DataTable dataTable = new DataTable(); // Створимо новий DataTable для збереження результатів запиту

dataTable.Load(reader); // Завантажимо дані з SqlDataReader до DataTable

dgw.DataSource = dataTable; // Прив'яжемо DataTable до DataGridView

// Змінимо назви стовпців

dgw.Columns["RequestDate"].HeaderText = "Дата заявки";

dgw.Columns["RequestStatus"].HeaderText = "Статус заявки";

dgw.Columns["Details"].HeaderText = "Деталі";

dgw.Columns["IdBrigade"].HeaderText = "ID Бригади";

dgw.Columns["WorkDate"].HeaderText = "Дата виконання";

dgw.Columns["SumRequest"].HeaderText = "Сума заявки";

dgw.Columns["SurnameC"].HeaderText = "Прізвище";

dgw.Columns["NameC"].HeaderText = "Ім'я";

dgw.Columns["PatronymicC"].HeaderText = "По батькові";

dgw.Columns["PhoneC"].HeaderText = "Телефон";

dgw.Columns["AddressC"].HeaderText = "Адреса";

reader.Close();

dataBase.closeConnection(connection);

}

}

Поданий код вище показує як програмно було побудовано метод ConnectionRequestView(). Спочатку отримуємо дані з бази даних, підготовлюємо їх для нормально вигляду у вікні програми. Потім зображуються дані певного користувача.

- метод Pay() виглядає так у програмному коді;

// Оплата рахунків

private void buttonPay\_Click(object sender, EventArgs e)

{

DateTime paymentDate = DateTime.Now;

List<short> paidBillIds = UpdateUnpaidBillsForClient(currentClientId, paymentDate);

List<short> updatedConnectionRequestIds = UpdateUnpaidConnectionRequestsForClient(currentClientId, paymentDate);

List<short> updatedAccidentReportIds = UpdateUnpaidAccidentReportsForClient(currentClientId, paymentDate);

string message = "";

if (paidBillIds.Count > 0)

{

message += "Оплачено рахунки з ID: " + string.Join(", ", paidBillIds) + "\n";

}

else

{

message += "Рахунків для оплати не знайдено.\n";

}

if (updatedConnectionRequestIds.Count > 0)

{

message += "Оплачено заявки на підключення з ID: " + string.Join(", ", updatedConnectionRequestIds) + "\n";

}

else

{

message += "Заявок на підключення для оплати не знайдено.\n";

}

if (updatedAccidentReportIds.Count > 0)

{

message += "Оплачено звіти про аварії з ID: " + string.Join(", ", updatedAccidentReportIds);

}

else

{

message += "Звітів про аварії для оплати не знайдено.";

}

MessageBox.Show(message, "Інформація", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

private List<short> UpdateUnpaidBillsForClient(short clientId, DateTime paymentDate)

{

List<short> paidBillIds = new List<short>();

using (SqlConnection connection = dataBase.getConnection())

{

string query = @"

UPDATE WaterBill

SET

DueDate = @PaymentDate,

BillStatus = 'Оплачено'

WHERE

IdClient = @ClientId

AND BillStatus = 'Неоплачено'";

SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@PaymentDate", paymentDate);

command.Parameters.AddWithValue("@ClientId", clientId);

dataBase.openConnection(connection);

int rowsAffected = command.ExecuteNonQuery();

if (rowsAffected > 0)

{

// Якщо хоч один рядок був змінений, додаємо ID рахунку до списку оплачених рахунків

paidBillIds.Add(clientId);

}

dataBase.closeConnection(connection);

}

return paidBillIds;

}

private List<short> UpdateUnpaidAccidentReportsForClient(short clientId, DateTime paymentDate)

{

List<short> updatedReportIds = new List<short>();

using (SqlConnection connection = dataBase.getConnection())

{

string query = @"

UPDATE AccidentReport

SET ReportStatus = 'Оплачено'

WHERE IdClient = @ClientId

AND SumRequest > 0";

SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@ClientId", clientId);

dataBase.openConnection(connection);

int rowsAffected = command.ExecuteNonQuery();

if (rowsAffected > 0)

{

// Додаємо ID звіту про аварію до списку оновлених звітів

updatedReportIds.Add(clientId);

}

dataBase.closeConnection(connection);

}

return updatedReportIds;

}

private List<short> UpdateUnpaidConnectionRequestsForClient(short clientId, DateTime paymentDate)

{

List<short> updatedRequestIds = new List<short>();

using (SqlConnection connection = dataBase.getConnection())

{

string query = @"

UPDATE ConnectionRequest

SET RequestStatus = 'Оплачено'

WHERE IdClient = @ClientId

AND SumRequest > 0";

SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@ClientId", clientId);

dataBase.openConnection(connection);

int rowsAffected = command.ExecuteNonQuery();

if (rowsAffected > 0)

{

// Додаємо ID заявки на підключення до списку оновлених заявок

updatedRequestIds.Add(clientId);

}

dataBase.closeConnection(connection);

}

return updatedRequestIds;

}

Поданий код вище показує як реалізована оплата рахунків клієнта. Спочатку перевіряється, чи є рахунки у клієнта чи оновлені заявки. Якщо так, то оновлюємо дані в записах. Якщо немає, то виводиться відповідне повідомлення.

**7.3 Програмний код до класу AdminForm**

У цьому підрозділі буде показано загальний вигляд класу AdminForm в програмному коді. Лише будуть основні методи, які згадуються в цій роботі. Весь код програмного класу можна переглянути в Додатку А.

Код класу AdminForm:

- атрибути та деякі елементи в класі;

public partial class AdminForm : Form

{

private Database dataBase = new Database();

public AdminForm()

{

InitializeComponent();

}

Поданий код вище показує, які є атрибути в класі AdminForm.

- метод dataGridViewInfo() виглядає так у програмному коді;

// Виклик форми редагування даних

private void dataGridViewInfo\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

// Перевірка назви стовпця, щоб визначити, яка заявка вибрана

if (dataGridViewInfo.Columns[e.ColumnIndex].Name == "IdRequest")

{

short idRequest = Convert.ToInt16(dataGridViewInfo.Rows[e.RowIndex].Cells["IdRequest"].Value);

DateTime requestDate = Convert.ToDateTime(dataGridViewInfo.Rows[e.RowIndex].Cells["RequestDate"].Value);

string requestStatus = dataGridViewInfo.Rows[e.RowIndex].Cells["RequestStatus"].Value.ToString();

string details = dataGridViewInfo.Rows[e.RowIndex].Cells["Details"].Value.ToString();

if (requestStatus.Equals("Оплачено"))

{

MessageBox.Show("Редагування цієї заявки не дозволено, оскільки статус 'Оплачено'.", "Повідомлення", MessageBoxButtons.OK, MessageBoxIcon.Information);

return;

}

ConnectionRequest requestDetailsForm = new ConnectionRequest(idRequest, requestDate, requestStatus, details);

requestDetailsForm.ShowDialog();

}

else if (dataGridViewInfo.Columns[e.ColumnIndex].Name == "IdReport")

{

short idReport = Convert.ToInt16(dataGridViewInfo.Rows[e.RowIndex].Cells["IdReport"].Value);

DateTime reportDate = Convert.ToDateTime(dataGridViewInfo.Rows[e.RowIndex].Cells["ReportDate"].Value);

string reportStatus = dataGridViewInfo.Rows[e.RowIndex].Cells["ReportStatus"].Value.ToString();

string details = dataGridViewInfo.Rows[e.RowIndex].Cells["Details"].Value.ToString();

if (reportStatus.Equals("Оплачено"))

{

MessageBox.Show("Редагування цієї заявки не дозволено, оскільки статус 'Оплачено'.", "Повідомлення", MessageBoxButtons.OK, MessageBoxIcon.Information);

return;

}

EmergencyRepairRequest reportDetailsForm = new EmergencyRepairRequest(idReport, reportDate, reportStatus, details);

reportDetailsForm.ShowDialog();

}

}

}

Поданий код вище демонструє вигляд методу dataGridViewInfo(), де в залежності, що переглядає користувач (заявки на усунення або на підключення), запуститься відповідна форма для редагування заявки.

- метод PurchaseMaterial() виглядає так у програмному коді;

// Закупівля матеріалів

private void buttonPurchaseMaterial\_Click(object sender, EventArgs e)

{

PurchaseMaterialsForm purchaseMaterialsForm = new PurchaseMaterialsForm();

purchaseMaterialsForm.ShowDialog();

}

Поданий код вище показує як працює метод PurchaseMaterial(). Після того як користувач натиснить кнопку в нього відкриється форма для закупівлі матеріалів. Повний клас форми буде знаходитись у додатку А.

- метод AddWaterBill() виглядає так у програмному коді;

// Метод для додавання рахунку в базу даних для певного клієнта

private void AddWaterBill(short clientId, DateTime billDate, decimal pricePerCubicMeter, decimal cubicMetersUsed, string billStatus)

{

using (SqlConnection connection = dataBase.getConnection())

{

string query = @"

INSERT INTO WaterBill (IdClient, BillDate, PricePerCubicMeter, CubicMetersUsed, BillStatus)

VALUES (@IdClient, @BillDate, @PricePerCubicMeter, @CubicMetersUsed, @BillStatus)";

SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@IdClient", clientId);

command.Parameters.AddWithValue("@BillDate", billDate);

command.Parameters.AddWithValue("@PricePerCubicMeter", pricePerCubicMeter);

command.Parameters.AddWithValue("@CubicMetersUsed", cubicMetersUsed);

command.Parameters.AddWithValue("@BillStatus", billStatus);

dataBase.openConnection(connection);

command.ExecuteNonQuery();

dataBase.closeConnection(connection);

}

}

Поданий код вище демонструє функціонал додавання рахунку певному клієнту.

**7.4 Програмний код до класу EmergencyRepairRequest**

У цьому підрозділі буде показано загальний вигляд класу EmergencyRepairRequest в програмному коді. Лише будуть основні методи, які згадуються в цій роботі. Весь код програмного класу можна переглянути в Додатку А.

Код класу EmergencyRepairRequest:

- атрибути та деякі елементи в класі;

public partial class EmergencyRepairRequest : Form

{

private short \_idRequest;

private DateTime \_requestDate;

private string \_requestStatus;

private string \_details;

private decimal fixedAmount;

public DateTime workDate;

Database dataBase = new Database();

Brigade brigade = new Brigade();

Material material = new Material();

WriteOffMaterials writeOffMaterials = new WriteOffMaterials();

Invoice invoice = new Invoice();

// Глобальна колекція для зберігання вибраних матеріалів

List<(short MaterialId, string RequestType, int Quantity)> selectedMaterials = new List<(short MaterialId, string RequestType, int Quantity)>();

public EmergencyRepairRequest()

{

InitializeComponent();

}

Поданий код вище показує скільки атрибутів містить клас.

- методи dateTimePickerTime\_ValueChanged та AssignBrigade() виглядають так у програмному коді;

private void AssignBrigade()

{

DateTime selectedDate = dateTimePickerDate.Value.Date;

TimeSpan selectedTime = dateTimePickerTime.Value.TimeOfDay;

DateTime dateTimeCombined = selectedDate.Add(selectedTime);

Brigade[] freeBrigades = brigade.AssignBrigade(dateTimeCombined);

comboBoxBrigade.Items.Clear();

foreach (Brigade brigade in freeBrigades)

{

comboBoxBrigade.Items.Add(brigade.Id);

}

}

private void dateTimePickerTime\_ValueChanged(object sender, EventArgs e)

{

AssignBrigade();

}

Поданий код вище демонструє 2 методи відразу. Так як було реалізовано час через елемент WinForms DataTimePicker, то було зроблено подію, яка при встановлені дати та часу користувачем, викликає метод AssignBrigade(), де отримаємо вільні бригади на цей день та час.

- методи CheckMaterials() виглядають так у програмному коді;

private void EmergencyRepairRequest\_Load(object sender, EventArgs e)

{

string requestType = "Усунення аварій";

selectedMaterials = material.CheckMaterials(requestType);

if (selectedMaterials.Count == 0)

{

MessageBox.Show("Потрібно здійснити закупку необхідних матеріалів!", "Повідомлення", MessageBoxButtons.OK, MessageBoxIcon.Information);

this.Hide();

this.Close();

return;

}

comboBoxNeedMaterial.Items.Clear();

foreach (var material in selectedMaterials)

{

// Кожен елемент матеріалу може бути доданий як рядок до комбобоксу

comboBoxNeedMaterial.Items.Add($"{material.RequestType}: {material.Quantity}");

}

fixedAmount = invoice.GetAmountForRequestType(requestType);

}

Поданий код вище показує реалізацію методу CheckMaterials(), де об’єкт класу Materials викликає його ж метод, аби перевірити чи є необхідні матеріали для заявки.

- методи WriteOffMaterials() виглядають так у програмному коді;

// Фіксація списання матеріалів

private void WriteOffMaterials(short materialId, string requestType, int quantity, DateTime writeOffDate, short requestId)

{

WriteOffMaterials writeOffMaterials = new WriteOffMaterials();

writeOffMaterials.AddWriteOffMaterials(materialId, requestType, quantity, writeOffDate, requestId);

}

private void buttonUpdate\_Click(object sender, EventArgs e)

{

if (!AreRequiredFieldsFilled())

{

return;

}

UpdateEmergencyRepairRequest(\_idRequest);

comboBoxNeedMaterial\_SelectedIndexChanged(sender, e);

}

private void comboBoxNeedMaterial\_SelectedIndexChanged(object sender, EventArgs e)

{

// Отримати вибраний елемент

string selectedItem = comboBoxNeedMaterial.SelectedItem.ToString();

// Знайти відповідний матеріал у списку

var selectedMaterial = selectedMaterials.FirstOrDefault(m => $"{m.RequestType}: {m.Quantity}" == selectedItem);

if (selectedMaterial != default)

{

short materialId = selectedMaterial.MaterialId;

string requestType = selectedMaterial.RequestType;

int quantity = selectedMaterial.Quantity;

DateTime writeOffDate = dateTimePickerDate.Value.Date.Add(dateTimePickerTime.Value.TimeOfDay);

short idRequest = \_idRequest;

WriteOffMaterials(materialId, requestType, quantity, writeOffDate, idRequest);

this.Close();

}

else

{

// Обробка випадку, коли матеріал не знайдено

MessageBox.Show("Матеріал не знайдено у списку.");

}

}

Поданий вище код показує реалізацію методу WriteOffMaterials, де при натиснені кнопки відбувається виклик обробника подій, отримуємо дані з нього, та передаємо дані до іншого методу, який знаходиться в класі WriteOffMaterials.

**7.5 Програмний код до класу ConnectionRequest**

У цьому підрозділі буде показано загальний вигляд класу ConnectionRequest в програмному коді. Лише будуть основні методи, які згадуються в цій роботі. Весь код програмного класу можна переглянути в Додатку А. Так як ConnectionRequest є ідентичним класу EmergencyRepairRequest, то будуть зображені коди методів без пояснень.

Код класу ConnectionRequest:

- атрибути та деякі елементи в класі;

public partial class ConnectionRequest : Form

{

private short \_idRequest;

private DateTime \_requestDate;

private string \_requestStatus;

private string \_details;

private decimal fixedAmount;

public DateTime workDate;

Database dataBase = new Database();

Brigade brigade = new Brigade();

Material material = new Material();

WriteOffMaterials writeOffMaterials = new WriteOffMaterials();

Invoice invoice = new Invoice();

// Глобальна колекція для зберігання вибраних матеріалів

List<(short MaterialId, string RequestType, int Quantity)> selectedMaterials = new List<(short MaterialId, string RequestType, int Quantity)>();

- методи dateTimePickerTime\_ValueChanged та AssignBrigade() виглядають так у програмному коді;

private void AssignBrigade()

{

DateTime selectedDate = dateTimePickerDate.Value.Date;

TimeSpan selectedTime = dateTimePickerTime.Value.TimeOfDay;

DateTime dateTimeCombined = selectedDate.Add(selectedTime);

Brigade[] freeBrigades = brigade.AssignBrigade(dateTimeCombined);

comboBoxBrigade.Items.Clear();

foreach (Brigade brigade in freeBrigades)

{

comboBoxBrigade.Items.Add(brigade.Id);

}

}

private void dateTimePickerTime\_ValueChanged(object sender, EventArgs e)

{

AssignBrigade();

}

- методи CheckMaterials() виглядають так у програмному коді;

private void ConnectionRequest\_Load(object sender, EventArgs e)

{

string requestType = "Підключення до системи водопостачання";

selectedMaterials = material.CheckMaterials(requestType);

if (selectedMaterials.Count == 0)

{

MessageBox.Show("Потрібно здійснити закупку необхідних матеріалів!", "Повідомлення", MessageBoxButtons.OK, MessageBoxIcon.Information);

this.Hide();

this.Close();

return;

}

comboBoxNeedMaterial.Items.Clear();

foreach (var material in selectedMaterials)

{

// Кожен елемент матеріалу може бути доданий як рядок до комбобоксу

comboBoxNeedMaterial.Items.Add($"{material.RequestType}: {material.Quantity}");

}

fixedAmount = invoice.GetAmountForRequestType(requestType);

}

- методи WriteOffMaterials() виглядають так у програмному коді;

private void WriteOffMaterials(short materialId, string requestType, int quantity, DateTime writeOffDate, short requestId)

{

//WriteOffMaterials writeOffMaterials = new WriteOffMaterials();

writeOffMaterials.AddWriteOffMaterials(materialId, requestType, quantity, writeOffDate, requestId);

}

private void buttonUpdate\_Click(object sender, EventArgs e)

{

if (!AreRequiredFieldsFilled())

{

return;

}

UpdateConnectionRequest(\_idRequest);

comboBoxNeedMaterial\_SelectedIndexChanged(sender, e);

}

private void comboBoxNeedMaterial\_SelectedIndexChanged(object sender, EventArgs e)

{

// Отримати вибраний елемент

string selectedItem = comboBoxNeedMaterial.SelectedItem.ToString();

// Знайти відповідний матеріал у списку

var selectedMaterial = selectedMaterials.FirstOrDefault(m => $"{m.RequestType}: {m.Quantity}" == selectedItem);

if (selectedMaterial != default)

{

// Викликати функцію WriteOffMaterials

short materialId = selectedMaterial.MaterialId;

string requestType = selectedMaterial.RequestType;

int quantity = selectedMaterial.Quantity;

DateTime writeOffDate = dateTimePickerDate.Value.Date.Add(dateTimePickerTime.Value.TimeOfDay);

short idRequest = \_idRequest; // Використовується з глобальної змінної

WriteOffMaterials(materialId, requestType, quantity, writeOffDate, idRequest);

// Закрити форму або виконати інші необхідні дії

this.Close();

}

else

{

// Обробка випадку, коли матеріал не знайдено (якщо необхідно)

MessageBox.Show("Матеріал не знайдено у списку.");

}

}

**7.6 Програмний код до класу Invoice**

У цьому підрозділі буде показано загальний вигляд класу Invoice в програмному коді.

Код класу Invoice:

internal class Invoice

{

private Database dataBase = new Database();

public decimal GetAmountForRequestType(string requestType)

{

decimal fixedAmount = 0;

using (SqlConnection connection = dataBase.getConnection())

{

dataBase.openConnection(connection);

string query = @"

SELECT Amount

FROM FixedAmountPerRequestType

WHERE RequestType = @requestType";

SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@requestType", requestType);

object result = command.ExecuteScalar();

if (result != DBNull.Value)

{

fixedAmount = Convert.ToDecimal(result);

}

dataBase.closeConnection(connection);

}

return fixedAmount;

}

}

Поданий код показує всю функціональність класу Invoice, який виконує розрахунок ціни для певного типу заявки. На основі запиту отримуємо потрібно інформацію та передаємо до класів, які відповідають за редагування заявок: EmergencyRepairRequest та ConnectionRequest.

**7.7 Програмний код до класу Brigade**

У цьому підрозділі буде показано загальний вигляд класу Brigade в програмному коді.

Код класу Brigade:

internal class Brigade

{

private Database dataBase = new Database();

public int Id { get; set; }

public DateTime DateCreation { get; set; }

public Brigade() { }

public Brigade(int id, DateTime dateCreation)

{

Id = id;

DateCreation = dateCreation;

}

public Brigade[] AssignBrigade(DateTime date)

{

List<Brigade> freeBrigades = new List<Brigade>();

using (SqlConnection connection = dataBase.getConnection())

{

dataBase.openConnection(connection);

string query = @"

SELECT IdBrigade, DateCreation

FROM Brigade";

SqlCommand command = new SqlCommand(query, connection);

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

short id = reader.GetInt16(0);

DateTime dateCreation = reader.GetDateTime(1);

if (!IsBrigadeUsedOnDate(id, date))

{

freeBrigades.Add(new Brigade(id, dateCreation));

}

}

}

dataBase.closeConnection(connection);

}

return freeBrigades.ToArray();

}

private bool IsBrigadeUsedOnDate(short brigadeId, DateTime date)

{

string query = @"

SELECT COUNT(\*)

FROM (

SELECT WorkDate, IdBrigade FROM ConnectionRequest

UNION ALL

SELECT WorkDate, IdBrigade FROM AccidentReport

) AS CombinedRequests

WHERE WorkDate = @date AND IdBrigade = @brigadeId";

using (SqlConnection connection = dataBase.getConnection())

{

dataBase.openConnection(connection);

SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@date", date);

command.Parameters.AddWithValue("@brigadeId", brigadeId);

// Журналювання для перевірки параметрів

Console.WriteLine($"Checking if brigade {brigadeId} is used on date {date}");

int count = (int)command.ExecuteScalar();

dataBase.closeConnection(connection);

// Журналювання для перевірки результату

Console.WriteLine($"Brigade {brigadeId} usage count on {date}: {count}");

return count > 0;

}

}

}

Поданий вище код представляє реалізацію класу Brigade, який виконує запит до бази даних, аби отримати вільні бригади на певну дату та час.

**7.8 Програмний код до класу WriteOffMaterials**

У цьому підрозділі буде показано загальний вигляд класу WriteOffMaterials в програмному коді.

Код класу WriteOffMaterials:

internal class WriteOffMaterials

{

private Database dataBase = new Database();

public void AddWriteOffMaterials(short materialId, string requestType, int quantity, DateTime writeOffDate, short requestId)

{

using (SqlConnection connection = dataBase.getConnection())

{

dataBase.openConnection(connection);

SqlTransaction transaction = connection.BeginTransaction();

try

{

// Додаємо запис

string insertQuery = @"INSERT INTO WriteOffMaterials (MaterialId, RequestType, Quantity, WriteOffDateW, RequestId)

VALUES (@MaterialId, @RequestType, @Quantity, @WriteOffDate, @RequestId)";

SqlCommand insertCommand = new SqlCommand(insertQuery, connection, transaction);

insertCommand.Parameters.AddWithValue("@MaterialId", materialId);

insertCommand.Parameters.AddWithValue("@RequestType", requestType);

insertCommand.Parameters.AddWithValue("@Quantity", quantity);

insertCommand.Parameters.AddWithValue("@WriteOffDate", writeOffDate);

insertCommand.Parameters.AddWithValue("@RequestId", requestId);

insertCommand.ExecuteNonQuery();

// Видаляємо матеріал

string deleteQuery = @"DELETE FROM RequestMaterials

WHERE MaterialId = @MaterialId AND RequestType = @RequestType";

SqlCommand deleteCommand = new SqlCommand(deleteQuery, connection, transaction);

deleteCommand.Parameters.AddWithValue("@MaterialId", materialId);

deleteCommand.Parameters.AddWithValue("@RequestType", requestType);

deleteCommand.ExecuteNonQuery();

transaction.Commit();

}

catch (Exception ex)

{

transaction.Rollback();

throw new Exception("Error while updating the database.", ex);

}

finally

{

dataBase.closeConnection(connection);

}

}

}

}

Поданий вище код демонструє реалізацію класу WriteOffMaterials, який спрацьовує лише тоді, коли користувач відредагував форму, відповідно ввів всі необхідні дані для виконання запиту. У цей час цей клас робить запис у бази даних, та видаляє з необхідних матеріалів той запис, який буде прикріплений до заявки.

**7.9 Програмний код до класу Material**

У цьому підрозділі буде показано загальний вигляд класу Material в програмному коді.

Код класу Material:

internal class Material

{

private Database dataBase = new Database();

public List<(short MaterialId, string RequestType, int Quantity)> CheckMaterials(string requestType)

{

List<(short MaterialId, string RequestType, int Quantity)> requiredMaterials = new List<(short MaterialId, string RequestType, int Quantity)>();

using (SqlConnection connection = dataBase.getConnection())

{

dataBase.openConnection(connection);

string query = @"

SELECT m.IdMaterial, rm.RequestType, rm.Quantity

FROM Material m

INNER JOIN RequestMaterials rm ON m.IdMaterial = rm.MaterialId

WHERE rm.RequestType = @requestType";

SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@requestType", requestType);

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

short materialId = Convert.ToInt16(reader["IdMaterial"]);

string requestTypeFromDb = reader["RequestType"].ToString();

int quantity = Convert.ToInt16(reader["Quantity"]);

requiredMaterials.Add((materialId, requestTypeFromDb, quantity));

}

reader.Close();

dataBase.closeConnection(connection);

}

return requiredMaterials;

}

}

Поданий вище код демонструє реалізацію класу Material, де на основі переданого типу заявки шукає записи в базі даних, які підходять під цей тип заявки.

**8 ЗАПИТИ ДО БАЗИ ДАНИХ**

У цьому розділі буде представлено 9 запитів, які відповідають таким вимогам:

а) вибір з декількох таблиць із сортуванням. Текст умови запиту:  
Вибрати список клієнтів разом з датами створення їхніх заявок на підключення та відповідними бригадами, відсортований за датою заявки на підключення в порядку спадання.

Запит:

SELECT

C.SurnameC, C.NameC, C.PatronymicC,

CR.RequestDate, CR.RequestStatus,

B.DateCreation

FROM

Client C

JOIN ConnectionRequest CR ON C.IdClient = CR.IdClient

JOIN Brigade B ON CR.IdBrigade = B.IdBrigade

ORDER BY

CR.RequestDate DESC;

б) Завдання умови відбору з використанням предиката LІKE. Текст умови запиту: Знайти всіх клієнтів, чиє прізвище починається на букву "Л".

Запит:

SELECT

C.SurnameC, C.NameC, C.PatronymicC, C.PhoneC

FROM

Client C

WHERE

C.SurnameC LIKE 'Л%';

в) Завдання умови відбору з використанням предиката BETWEEN. Текст умови запиту: Отримати список заявок на підключення, створених у 2023 році.

Запит:

SELECT

CR.IdRequest, C.SurnameC, C.NameC, CR.RequestDate, CR.RequestStatus

FROM

Client C

JOIN ConnectionRequest CR ON C.IdClient = CR.IdClient

WHERE

CR.RequestDate BETWEEN '2023-01-01' AND '2023-12-31';

г) Агрегатна функція без угруповання. Текст умови запиту: Порахувати загальну кількість клієнтів, які зареєструвалися у системі (мають IdUser).

SELECT

COUNT(\*) AS TotalClients

FROM

Client

WHERE

IdUser IS NOT NULL;

ґ) Агрегатна функція з угрупованням. Текст умови запиту: Порахувати кількість заявок на підключення для кожної дати створення бригади.

Запит:

SELECT

B.DateCreation, COUNT(\*) AS TotalRequests

FROM

Brigade B

JOIN ConnectionRequest CR ON B.IdBrigade = CR.IdBrigade

GROUP BY

B.DateCreation;

д) Використання предиката ALL або ANY. Текст умови запиту: Знайти посаду і робітника, який має найбільшу кількість заявок на підключення.

Запит:

SELECT

J.TitleJ, E.SurnameE, E.NameE, COUNT(\*) AS TotalRequests

FROM

Employee E

JOIN JobTitle J ON E.JobTitleId = J.IdJobTitle

JOIN Brigade B ON E.BrigadeId = B.IdBrigade

JOIN ConnectionRequest CR ON B.IdBrigade = CR.IdBrigade

GROUP BY

J.TitleJ, E.SurnameE, E.NameE

HAVING

COUNT(\*) >= ALL (SELECT COUNT(\*)

FROM ConnectionRequest

GROUP BY IdBrigade);

е) Корельований підзапит. Текст умови запиту: Для кожної посади визначити робітника, який має максимальну кількість заявок на підключення.

Запит:

SELECT

J.TitleJ, E.SurnameE, E.NameE

FROM

Employee E

JOIN JobTitle J ON E.JobTitleId = J.IdJobTitle

WHERE

E.IdEmployee = (SELECT TOP 1 IdEmployee

FROM Employee E2

WHERE E2.JobTitleId = J.IdJobTitle

ORDER BY (SELECT COUNT(\*)

FROM ConnectionRequest CR

WHERE CR.IdBrigade = E2.BrigadeId) DESC);

є) Запит на заперечення. Текст умови запиту: найти робітників, які не брали участь у заявках на підключення протягом останнього тижня.

Запит:

-- Використання LEFT JOIN:

SELECT

E.SurnameE, E.NameE

FROM

Employee E

LEFT JOIN ConnectionRequest CR ON E.BrigadeId = CR.IdBrigade

AND CR.RequestDate BETWEEN '2023-06-01' AND '2023-06-07'

WHERE

CR.IdRequest IS NULL;

-- Використання предиката IN:

SELECT

E.SurnameE, E.NameE

FROM

Employee E

WHERE

E.BrigadeId NOT IN (SELECT CR.IdBrigade

FROM ConnectionRequest CR

WHERE CR.RequestDate BETWEEN '2023-06-01' AND '2023-06-07');

-- Використання предиката EXISTS:

SELECT

E.SurnameE, E.NameE

FROM

Employee E

WHERE

NOT EXISTS (SELECT 1

FROM ConnectionRequest CR

WHERE CR.IdBrigade = E.BrigadeId

AND CR.RequestDate BETWEEN '2023-06-01' AND '2023-06-07');

ж) Операція об'єднання UNION із включенням коментарю в кожен рядок. Текст умови запиту: Отримати список бригад з коментарем про кількість виконаних робіт.

Запит:

SELECT

B.IdBrigade,

'Має виконані роботи' AS Коментар

FROM

Brigade B

JOIN

ConnectionRequest CR ON B.IdBrigade = CR.IdBrigade

UNION

SELECT

B.IdBrigade,

'Не має виконаних робіт' AS Коментар

FROM

Brigade B

WHERE

B.IdBrigade NOT IN (

SELECT

CR.IdBrigade

FROM

ConnectionRequest CR

);

**9 ТЕСТУВАННЯ**