Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

СИСТЕМА АВТОМАТИЧНОГО СПОВІЩЕННЯ ПРО ВІДКЛЮЧЕННЯ ЕЛЕКТРОЕНЕРГІЇ У ВОЛИНСЬКІЙ ОБЛАСТІ

SYSTEM OF AUTOMATIC NOTIFICATION OF POWER OUTAGES IN THE VOLYN REGION

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІс-21
Чижик Нікіта Миколайович

(підпис)

Керівник:
к.т.н., доцент
Багнюк Наталія Володимирівна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« _____ » червня 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Чижик Нікіта Миколайович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Система автоматичного сповіщення про відключення електроенергії у Волинській області*

Керівник роботи *к.т.н., доцент Багнюк Наталія Володимирівна*

затверджені наказом закладу вищої освіти від «28» грудня 2022 року № 982/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 01.06.2023р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Аналіз предметної області
Використовувані програмні засоби та технології
Програмна реалізація системи
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Існуючі рішення
Приклад графіку відключення
Опис процесу відправки сповіщень
Інтерфейс реєстрації користувача
Приклад оповіщення

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Багнюк Н.В.*		
Теоретичне дослідження та практична реалізація	*Багнюк Н.В.*		
Практична реалізація об'єкта проектування	*Багнюк Н.В.*		
Висновки			

7. Дата видачі завдання 01.11.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Обгрунтування теми*	До 15.11.2022 р.	Виконано
2.	*Огляд літератури із досліджуваної проблеми*	До 15.12.2022 р.	Виконано
3.	*Аналіз предметної області*	До 02.02.2023 р.	Виконано
4.	*Використовувані програмні засоби та технології*	До 02.03.2023 р.	Виконано
5.	*Програмна реалізація системи*	До 01.04.2023 р.	Виконано
6.	*Висновки та пропозиції*	До 02.04.2023 р.	Виконано
7.	*Формування списку використаних джерел*	До 15.04.2023 р.	Виконано
8.	*Формування додатків*	До 02.05.2023 р.	Виконано
9.	*Оформлення ілюстративного матеріалу*	До 15.05.2023 р.	Виконано
10.	*Нормоконтроль*	До 25.05.2023 р.	Виконано
11.	*Інструментальна перевірка на академічний плагіат*	До 01.06.2023 р.	Виконано
12.	*Представлення кваліфікаційної роботи бакалавра до захисту*	До 07.06.2023 р.	Виконано

Здобувач вищої освіти _____ (підпис) Чижик Н.М. (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (підпис) Багнюк Н.В. (прізвище, ініціали)

АНОТАЦІЯ

Чижик Н.М. Система автоматичного сповіщення про відключення електроенергії у волинській області. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2023.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, п'яти додатків.

Перший розділ присвячено огляду предметної олбасті, тут розглядаються основні поняття про системи сповіщення, розглянуто актуальність теми. Також в цьому розділі здійснено огляд програм-аналогів.

В другому розділі здійснено вибір та обгронтування засобів розробки. Обрано засоби: C#, Net core, Rabbit mq, Blazor та MS SQL. Розглянуто середу розробки Visual Studio.

Третій розділ присвячено опису розробленої система для автоматичного відправлення сповіщень, опис архітектури системи, ключових компонентів системи та інструкції з користування.

Об'єкт – технології створення автоматизованних інформаційно-пошукових система для автоматичного інформування цивільного населення про відключення електроенергії.

Предмет є процес автоматизації інформування та пошуку данних для відключення електроенергії та надання доступу до них.

Метою роботи є створення автоматизованого веб ресурсу, основним функціоналом якого є відправка сповіщення про відсутність електроенергії за вказаною адрессою та перегляд стану та історії відключень по адресі.

Ключові слова: автоматизація, система оповіщеня, .Net, Blazor, сповіщення у Telegram, сповіщення через Email, веб-застосунок, відключення електроенергії

ANNOTATION

Chyzhyk N.M. System of automatic notification of power outages in the Volyn region. Manuscript.

Qualifying work of a bachelor of EP "Computer Engineering" specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2023.

Qualification work consists of an introduction, three sections, conclusions, a references, five appendices.

The first section is dedicated to the overview of the subject matter, here the basic concepts of notification systems are considered, the relevance of the topic is considered. Also, in this section, a review of similar programs is carried out.

In the second section, the selection and testing of development tools was carried out. Selected tools: C#, Net core, Rabbit mq, Blazor and MS SQL. The Visual Studio development environment is considered.

The third is devoted to the description of the developed system for automatic sending of notifications, description of the system architecture, key components of the system and instructions for use.

The object – technologies for creating automated information and search systems for automatically informing the civilian population about power outages.

The subject is the process of automating information and finding data for power outages and providing access to them.

The aim of the work is to create an automated web resource, the main functionality of which is to send a notification about the lack of electricity at the specified address and view the status and history of outages at the address.

Keywords: automation, notification system, .Net, Blazor, Telegram notification, Email notification, web application, power outage.

ЗМІСТ

ВСТУП

Актуальність кваліфікаційної роботи підверджується необхідністю поширення своєчасної та актуальної інформації серед населення та бізнесу. Під час воєнного стану чи військових дій відключення електроенергії можуть виникати з різних причин, у тому числі через пошкодження інфраструктури чи навмисні дії. Надаючи точні оновлення в режимі реального часу про збої в електропостачанні, система сповіщень допомагає окремим особам і організаціям відповідно планувати свою діяльність. Автоматична система сповіщень, здатна передавати користувачеві зручним способом актуальну інформацію, залежно від його місця проживання або розташування виробничих потужностей підприємства. Зможе значно оптимізувати та буде відігравти важливу роль у плануванні графіку роботи та інших потреб споживачів електроенергі.

Метою кваліфікаційної роботи є створення автоматизованого веб ресурсу, основним функціоналом якого є відправка сповіщення про відсутність електроенергії за вказаною адрессою та перегляд стану та історії відключень по адресі.

Об'єкт дослідження: технології створення автоматизованних інформаційно-пошукових система для автоматичного інформування цивільного населення про відключення електроенергії.

Предметом дослідження є процес автоматизації інформування та пошуку данних для відключення електроенергії та надання доступу до них.

Завдання кваліфікаційної роботи:

– провести аналіз потреб для реалізації системи` сповіщення про відключення та існуючих аналогів;

– розробити структуру для автоматичної системи сповіщення;

– практично реалізувати систему сповіщення та веб інтерфейс до неї;

– розробити інструкцію користування веб інтерфейсом та системою сповіщення.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз стану проблеми відключення електроенергії

З початку повномасштабного вторгнення РФ на територію України було не одноразово скоєно військовий злочин «Удари по об'єктах критичної інфраструктури» що стало причиною планових та аварійних відключень електроенергії.

В результаті цього злочину населення та бізнес сфера в Україні стикнулись з тривалими відключеннями електроенергії. Під час війни відключення електроенергії можуть мати дуже серйозні наслідки для населення та інфраструктури. Наприклад, у разі відключення електроенергії в житлових будинках може відсутнім бути опалення, освітлення та доступ до гарячої води. Це може призвести до посилення холоду та зниження комфортності проживання, особливо в зимовий період. Також відключення електроенергії можуть суттєво ускладнити роботу медичних закладів та інших соціальних установ, що потребують стабільного електропостачання. Наприклад, відключення електроенергії може призвести до відключення приладів штучної вентиляції легенів, зниження температури зберігання медичних препаратів та інші подібні проблеми.

Відключення електроенергії також може призвести до зниження продуктивності та ефективності енергосистеми. У разі відключення електроенергії знижується потужність електростанцій, що може призвести до порушення стабільності енергосистеми та збільшення ймовірності аварій. Що в свою чергу, може призвести до недостатнього постачання електроенергії та палива для виробництва електроенергії. Це може призвести до зниження продуктивності та робочих місць у галузі енергетики, а також до збільшення вартості електроенергії для населення та промисловості.

Під час війни були введена практика встановлення квот можливого об'єму споживання електроенергії. Для розрахунку квот на обсяги споживання

електроенергії кожне обленерго подає в Укренерго загальний прогноз споживання по своїй області, який має враховувати міграційні процеси, релокацію підприємств, фактичні зміни в загальному рівні й структурі споживання та середньомісячні температури по регіону. Укренерго, своєю чергою, розраховує у відсотках питому вагу кожної області в загальному обсязі споживання. Для дотримання ліміту Волиньобленерго застосувала планові відключення. Це відключення окремих вулиць, будинків чи населених пунктів за запланованим графіком, який в «ідеальному» світі мали були заплановані рівномірно для всього населення. Коли під час планових відключень, перевищуються заданий ліміт (рисунок 1.1) водиться аварійні відключення, що б забезпечити виконання квот від Укренерго [2].

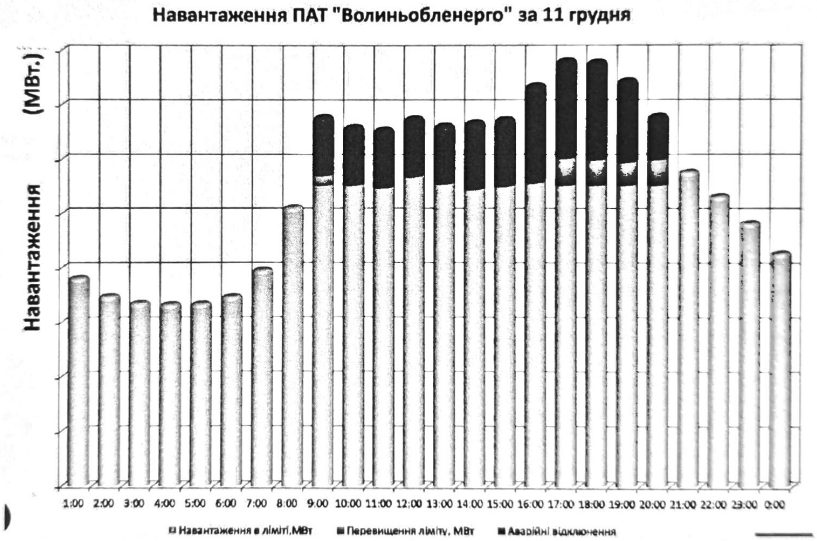

Рисунок 1.1 – Приклад перевищення ліміту для «Волиньобленерго» [2]

Але відключення електроенергії відбувались не тільки з початку повномасштабного вторгнення РФ, а і раніше. Наприклад, під час війни на сході України, яка почалася у 2014 році, відключення електроенергії відбуваються часто через знищення електромереж, дії бойовиків та інші проблеми. За даними Державної служби статистики України [4], у 2020 році у зоні АТО/ООС

(Антитерористична операція/Операція об'єднаних сил) зареєстровано 2418 випадків відключення електроенергії, що становить 7,5% від загальної кількості відключень в країні. В інших регіонах України також були планові та аварійні відключення. У відсутність бойових дій та терористичних актів, основні причини відключень наступні:

- Зношеність та недостатність інфраструктури: багато електричних мереж в Україні досить старі та потребують значного ремонту та модернізації, а також збільшення ємності для задоволення зростаючої потреби в електроенергії.
- Недостатність виробництва електроенергії: Україна має недостатню потужність виробництва електроенергії для задоволення пікової потреби. Це може призводити до відключень електроенергії в періоди пікового навантаження.
- Низька ефективність системи: Українська енергетична система має низьку ефективність та високу вартість виробництва електроенергії. Це може призводити до відключень електроенергії для зменшення надмірного споживання.
- Неплатоспроможність споживачів: багато споживачів в Україні не мають можливості оплатити свої рахунки за електроенергію, що призводить до заборгованостей перед енергетичними компаніями та може призвести до відключень електроенергії.
- Аварії: надзвичайні ситуації, такі як аварії на електромережах або виробничі аварії, можуть призводити до відключення електроенергії для запобігання подальшим аварійним ситуаціям та захисту споживачів.

Відключення електроенергії можуть мати значний вплив на бізнес. Особливо це стосується компаній, які потребують неперервного живлення для своєї діяльності, таких як виробництво, медичні установи, банки, інформаційні технології тощо.

Основні наслідки відключень електроенергії для бізнесу можуть включати наступне:

- Втрати продуктивності: Підприємства можуть зазнавати втрат продуктивності внаслідок відключень електроенергії, оскільки багато виробничих процесів потребують постійного електроживлення. Це може призвести до зниження продуктивності, відставання від графіку виробництва та невиконання замовлень.
- Втрати матеріальних цінностей: Відключення електроенергії може призвести до втрат матеріальних цінностей, таких як пошкодження обладнання, яке працює на електроенергії, або втрата продуктів, які потребують певної температури для зберігання.
- Фінансові втрати: Відключення електроенергії може вплинути на фінансову стійкість підприємства, оскільки компанії можуть зазнавати втрат доходів внаслідок зниження продуктивності та втрати продукції.
- Порушення договорів: Якщо відключення електроенергії триває довго, це може призвести до порушення договорів з клієнтами, які очікують своєчасної доставки товарів або послуг.
- Негативний вплив на репутацію: Відключення електроенергії може мати негативний вплив на репутацію підприємства серед клієнтів та інших зацікавлених сторін.

За даними Європейської Бізнес Асоціації (рисунок 1.2) під час масових атак на інфраструктуру більше 40% українського бізнесу довелось зменшити обсяги виробництва, 12% скоротили частину офісів/відділень/торгових точок та 66% змінили графік роботи, що є значним впливом на роботу та ефективність бізнесу та підприємств [1]. Через постійну зміну графіків відключень бізнесам постійно приходилось змінювати графік роботи. Тому для збільшення ефективності та автоматизації процесу отримання завчасно графіку відключень потрібна система

яка автоматично буде сповіщати про майбутні відключення та допоможе створити графік роботи для підприємства.

Рисунок 1.2 – Вплив відключень електроенергії на роботу бізнесу [1]

1.2 Порівняльний аналіз аналогічних існуючих систем для оповіщення

На момент написання роботи, Волиньобленерго не має системи автоматичного сповіщення про відключення електроенергії за певною адресою. Тому розглянемо приклади з інших областей та галузей.

Основні шляхи сповіщення користувача у сучасних автоматизованних системах є наступні:

- SMS-оповіщення: ця система оповіщення використовує текстові повідомлення для передачі інформації. Вона досить ефективна та швидка, оскільки більшість людей має мобільні телефони і перевіряє їх наявність повідомлень досить часто.
- Електронна пошта: електронна пошта є дуже популярною системою оповіщення, яка може бути використана для надсилання повідомлень

великій кількості людей. Вона також дозволяє відправляти додаткові деталі та прикріплення.

- Месенджери: такі як WhatsApp, Viber, Telegram, Slack тощо. Вони надають можливість відправляти текстові повідомлення, відео, фотографії та інші додатки, що робить їх більш різноманітними і зручними у використанні.
- Додатки на телефон: встановлення такого дадатку дозволяє відправляти сповіщення користувачу на його смартфоні. Досить ефективний та зручний варіант, так як користувач може обрати в налаштуваннях формат повідомлень, але потребує великих зусиль для розробки самого додатку.
- Системи автоматичних дзвінків: ці системи відправляють автоматичні дзвінки для оповіщення людей про надзвичайні ситуації. Вони дуже ефективні, оскільки майже кожен має телефон і може прийняти дзвінок.
- Соціальні мережи: такі як Facebook, Twitter тощо, дозволяють надсилати повідомлення тисячам людей одночасно. Вони можуть бути корисні для швидкого оповіщення про надзвичайні ситуації, але можуть бути менш ефективні, оскільки користувачи частно ігнорують або вимикають сповіщення в налаштуваннях.

Всі вище вказані види оповіщень є доступні в першу чергу на мобільний телефон, так як, сповіщення на мобільний пристрій є приоретним напрямком. В сучасному світі це є найпоширенішим девайсом серед населення та завдяки своїй мобільності, майже постійно знаходиться біля користувача. Але в кожного з вище вказаних видів сповіщень є свої плюси та недоліки. SMS-оповіщення є ефективним шляхом сповіщення, так як не потребують додатко щось завантажувати чи реєструватись, але оператор стягує оплату з відправника, що в свою чергу збільшує потенційну вартість утримання системи. Електронна пошта на відокрему від SMS-оповіщення є безкоштовним, але потребує реєстрації у

систему постачальника послуг пошти та завантаженя мобільного додатку для отримання сповіщень користувачем, за що постачальник може окремо стягувати оплату з користувача, також недоліком є швикдість роботи SMTP-систем, які використовується для відправки електронних листів, та існування феномену «загубленного» повідомлення. Системи автоматичних дзвінків є платними, та зазвичай дорожще за SMS-оповіщень, та можуть дратувати користувача, так як не існує можливості на тимчасово чи постійно відключити сповіщення. Для месенджерів та соціальні мереж має встановитись застосунон на мобільний пристрій, але сам застосунок є безкоштовний і вони надають інструменти для інтеграції, але месенджери мають перевагу коли мова заходить про сповіщення, так як це є частиною основного функціоналу.

Розглянемо на прикладі систему сповіщення про вікдлючення електронергії для споживачив міста Київ, які є клієнтами енергопостачальної компанії YASNO та водночас є абонентами мобільного оператору lifecell[14]. Споживачі мають можливість отримувати SMS-оповіщення про відключення, за вказаною ними групою споживачив, що визначає компанія YASNO за місцем проживання. Реєстрація відбувається через USSD-запит в lifecell, за визначеним кодом відправляється повідомлення «Активувати» та номер споживчої групи. Після реєстрації, SMS-оповіщення про стабілізаційні та екстрені відключення електроенергії будуть надходити з зазначенного номера. Користувачи отримуватимуть їх за 15 хвилин до початку так званої «сірої зони» — часового проміжку, коли можливе відключення світла.

Не доліками данної системи є відсутність вибору шляху як отримувати спровіщення, система обмеженна лише SMS-оповіщення, також відсутність користувацького інтерфесу, немає можливості переглянути історію оповіщень. Також серйозним недоліком є що користувачу потрібно знати та вказати групу споживача, замість адреси, що в свою чергу ускладнює користування системою, та створює певні не зручності. І система не доступна для споживачів у Волиньскій області.

Іншим прикладом системи оповіщення, це система миттєвого сповіщення (Emergency Alert System, EAS), яка використовується в США для широкомасштабного оповіщення громадян про надзвичайні ситуації, такі як пожежі, повені, теракти, тощо. Хоча система сповіщає про надзвичайні ситуатції, але вона є прикладом як можна точно та ефективно, за допомогою різних джерел сповіщати користувача.

Система EAS заснована на передачі повідомлень через радіо та телевізійні мережі, а також мобільні пристрої. Ця система автоматично активується, коли надходить повідомлення від національних органів управління надзвичайними ситуаціями або локальних органів управління екстреними ситуаціями. Вона забезпечує широке оповіщення, охоплюючи велику кількість радіо- та телевізійних станцій, а також мобільні пристрої, які мають можливість отримувати повідомлення через різні канали зв'язку. Також система забезпечує можливість відстеження стану оповіщення та повідомляє про будь-які проблеми з доставкою повідомлень. У випадку, якщо система EAS не використовується в режимі надзвичайних ситуацій, вона використовується для широкого сповіщення про інші події, такі як планові технічні роботи у мережі та інші події.

Але ця система також має відсутній графічний інтерфейс для кінцевого користувача, та відсутні будь які налаштування, які доступні користувачу.

1.3 Постановка задачі в результаті проведенного аналізу

Проаналізувавши проблему та її вплив на роботу бізнесу та населення, можна дійти до висновку, що автоматизована система сповіщення не тільки спростить життя для населлення, а і допоможе місцевому бізнесу та підприємствам завчасно планувати графік роботи, відповідно до графіку відключення. Та в разі зміни, бути вчасно повідомленим про зміну у графіку.

За відсутності подібних систем у Волиньскій області, проаналізувавши недоліки рішеннь для інших регіонів, можна сформувати вимоги до системи:

- Швидкість: система повинна бути швидкою та оперативною, щоб повідомлення про відключення електроенергії надходили миттєво.
- Надійність: система повинна бути надійною та стійкою до збоїв, щоб забезпечити правильну доставку повідомлень.
- Гнучкість: система повинна бути гнучкою та дозволяти настроювати параметри оповіщення, наприклад, час, коли відключення електроенергії передбачається, і кого оповіщати.
- Автоматичне керування: система повинна мати можливість автоматичного керування, щоб оповіщення надходили без участі людей.
- Безпека: система повинна бути безпечною та забезпечувати конфіденційність даних користувачів.
- Користувацький інтерфейс: система повинна мати зрозумілий та зручний користувацький інтерфейс для налаштування та керування оповіщеннями.
- Сумісність: система повинна бути сумісною з різними пристроями та мережами зв'язку, щоб забезпечити широку доступність користувачам.
- Відстеження історії: система повинна мати можливість для відстежування минулих оповіщенньь.

Однією з основних переваг цієї системи є можливість легкого розгортання на популярних серверних рішеннях, таких як Linux або Windows Server. Це означає, що користувачеві не потрібно встановлювати додаткові програми або компоненти на свій пристрій, оскільки весь функціонал системи доступний через сервер. Такий підхід має декілька переваг:

- Спрощена установка: Користувачеві не потрібно проводити складні процеси встановлення або конфігурації на своєму пристрої. Все необхідне вже налаштовано на сервері, і користувач може отримати доступ до системи швидко та без зайвих зусиль.

- Компактність: Оскільки весь функціонал системи зосереджений на сервері, це дозволяє зберегти простір на пристрої користувача. Великі обсяги даних та обчислювальні ресурси зберігаються та обробляються на сервері, а користувач працює з ними через інтерфейс системи.
- Збереження ресурсів: Користувачеві не потрібно мати потужний апаратний засіб або велику кількість пам'яті для виконання системи. Багато з обчислювальних задач та обробки даних здійснюються на сервері, а результати передаються на пристрій користувача, що дозволяє заощадити ресурси й забезпечити плавну роботу системи навіть на менш потужних пристроях.

Таким чином, розгортання цієї системи на серверних рішеннях дозволяє користувачеві насолоджуватися усіма перевагами та функціоналом системи без необхідності встановлювати додаткові програми на свій пристрій, забезпечуючи зручну та ефективну роботу з даними.

РОЗДІЛ 2
ВИКОРИСТОВУВАНІ ПРОГРАМНІ ЗАСОБИ ТА ТЕХНОЛОГІЇ

2.1 Інформаційна модель роботи системи

Перед початком реалізації програми важливо продумати структуру та алгоритм роботи системи. Алгоритм роботи програми, зображенний на інформаційній моделі (рисунок 2.1), має наступні кроки:

- Завантаження оновненого графіку: система повинна перевіряти наявність нового графіку та в разі його наявності завантажити його, зберегти та підготовити для обробки.
- Конвертація данних: на цьому етапі система повинна буде конвертувати файл з графіками та обробити, та повернути результат у зручному для збереження в базі данних форматі.
- Запис в базу данних: так як користувач може зареєструватись після оновлення графіку то система має зберегти структуровані данні в базу данних.
- Витягування налаштувань користувачів та створення об'єктів повідомлень у черги: потрібно витягнути налаштування користувачив, що б порівняти адреси з налаштувань та з нового графіку, та скорегувати час сповіщення, на заданий користувачам час.
- Створення черги повідомлень: що б оптимізувати відправку великої кількості повідомлень та в разі помилки відправки програми зберегти чергу повідомлень до рестарту програми, буде використува асинхронна черга івентів.
- Відправка повідомлень: модуль відправки має викликатись чергою повідмлень та за вибраним типом сповіщенням відправляти його кінцевому користувачу.

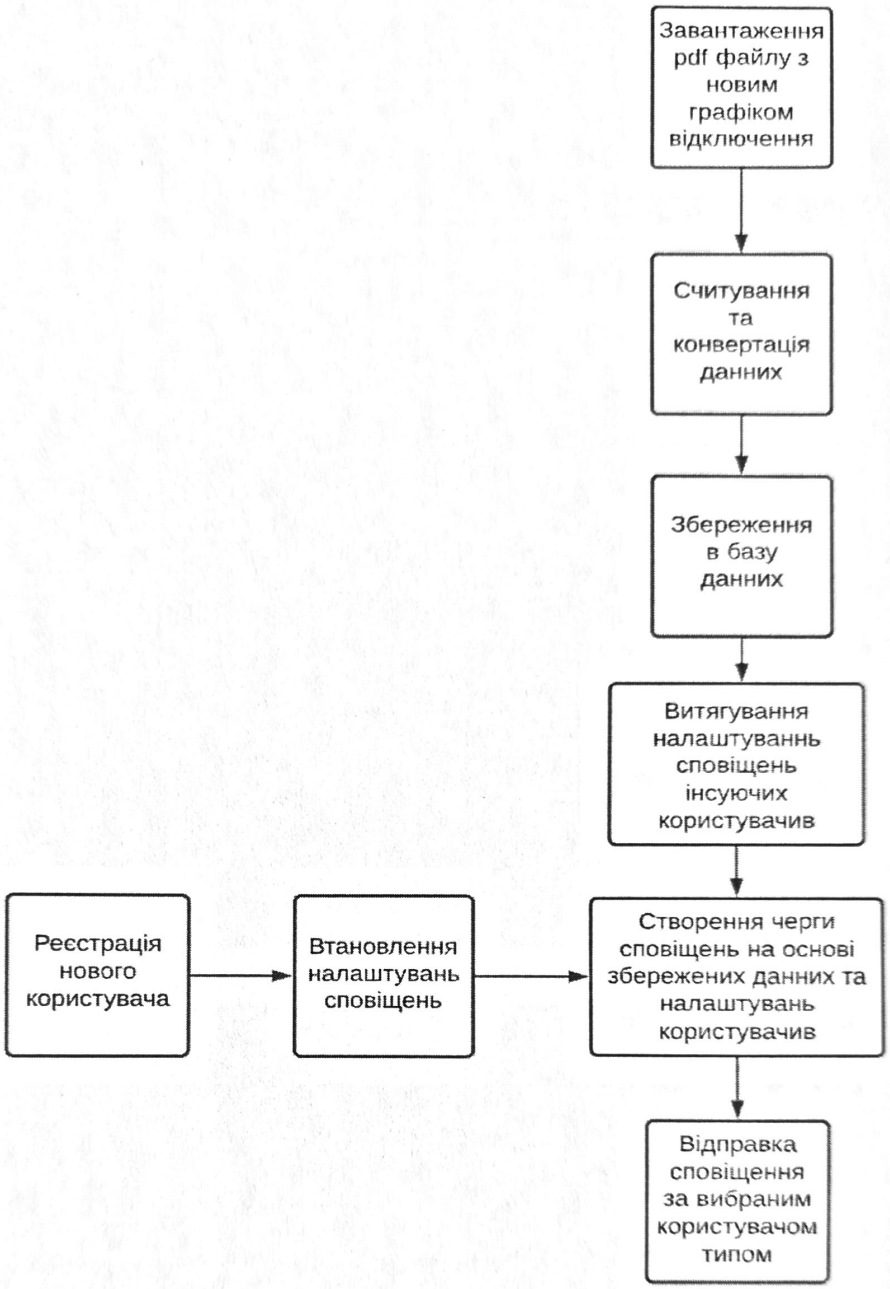

Рисунок 2.1 – Інформаційна модель роботи системи

2.2 Мова програмування C# та платформа Net Core

Net Core – це модульна платформа для розробки програмного забезпечення з відкритим вихідним кодом. Була випущена компанією Microsoft. Мовою програмування є C#.

NET Core [7] може працювати поверх крос-платформного середовища .NET, яке може бути розгорнута на основних популярних операційних системах: Windows, Mac OS, Linux. І таким чином,NET Core має можливість створювати крос-платформні додатки. І хоча Windows як середовище для розробки і розгортання програми досі переважає, але вже існує підтримка інших операційних систем. Тобто є можливість запускати веб-додатки не тільки на операційній системі Windows, але і на Linux і Mac OS. А для розгортання веб-додатка можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатка можуть завантажуватися як окремі модулі через пакетний менеджер Nuget.

Модуль Nuget є окремим ZIP-файл з розширенням .nupkg, які містять скомпільований код (DLL), інші файли, пов'язані з цим кодом, і описовий маніфест, що включає такі відомості, як номер версії пакета. Розробники, у яких є код, до якого потрібно надати загальний доступ, створюють пакети і публікують їх на закритих або відкритих вузлах. Користувачі отримують ці пакети з відповідних вузлів, додають їх у свої проекти, а потім викликають функції пакета в коді свого проекту. При цьому NuGet сам обробляє всі проміжні дані.

Net Core включає в себе фреймворк MVC [13], який об'єднує функціональність MVC, Web API і Web Pages. У попередніх версії платформи дані технології реалізувалися окремо і тому містили багато дублюючої функціональності. Зараз же вони об'єднані в одну програмну модель Net Core MVC.

Фактично фреймворк - це відокремлення .NET Framework, чия реалізація була оптимізована з урахуванням завдань декомпозиції. Хоча сценарії для .NET Native (мобільні пристрої) і Net 5 (серверна частина веб-розробки) порядком відрізнятися, для обох типів додатків є можливість розробити спільну базову бібліотеку класів (Base Class Library, BCL).

Набір API, доступний в .NET Core BCL[8], ідентичний для обох типів додатків: як для .NET Native, так і для Net 5. В основі BCL має дуже тонкий шар,

специфічний для середовища виконання .NET. На даний момент у нас є дві реалізації:

- одна, специфічна для середовища .NET Native,
- друга, специфічна для Core CLR, яка використовується в NET 5.

Однак цей шар не змінюється дуже часто. Він містить базові речі, наприклад String і Int32. Велика частина BCL - це чисті MSIL збірки, які можуть бути перевикористати як є. API для різних платформ не просто виглядають однаково, вони використовують одну і ту ж реалізацію. Наприклад, немає причин мати різні реалізації для колекцій.

Поверх BCL є API, специфічні для моделі додатків. Зокрема, .NET Native надає API, специфічні для клієнтської розробки під Windows, такі як WinRT interop. NET 5 теж додає API, наприклад, MVC, яке специфічно для серверної частини веб-розробки.

Додатки Net Core можна публікувати в двох різних режимах. Режим впливає на те, як користувач запускає додаток. При публікації автономного додатку до складу, окрім додатку і його залежностей, включається runtime і бібліотеки Net Core. Користувачі програми можуть запустити його на комп'ютері, на якому не встановлена runtime Net Core.

Переваги:

- Управління версією .NET. Ви керуєте тим, яка версія .NET розгортається разом з додатком.
- Націлювання на конкретні платформи. Оскільки необхідно опублікувати додаток для кожної платформи, ви знаєте, де буде виконуватися додаток. Якщо в .NET з'являється нова платформа, користувачі не зможуть запустити додаток на ній до випуску версії, призначеної для цієї платформи. Ви можете протестувати додаток на наявність проблем сумісності перед тим, як користувачі запустять додаток на новій платформі.

Недоліки:

- Більший розмір. Так як програма включає середовище виконання .NET і все залежності додатки, розмір файлу і необхідного простору на жорсткому диску буде більше, ніж для версії, яка залежить від середовища.
- Ускладнене оновлення версії .NET. Runtime .NET (поширювану з вашим додатком) можна оновити лише шляхом випуску нової версії програми. Але при необходімості.NET оновить критично важливі оновлення системи безпеки для бібліотеки платформи на комп'ютері, на якому виконується програма. Відповідальність за повну перевірку цього сценарію оновлення системи безпеки лежить на вас.

При публікації програми як залежного від середовища створюється додаток, яке включає тільки сам додаток і його залежності. Користувачам програми необхідно окремо встановити середу runtime Net Core.

Переваги:

- Невеликій розмір. Так як в публікації, тільки додаток і його залежності. Середу runtime і бібліотеки .NET встановлює користувач, і всі додатки спільно використовують встановлений runtime.
- Кросплатформені додатки. Ваша програма і бібліотеки на основі .NET працюють в інших операційних системах. Для програми не потрібно визначати цільову платформу.
- Використовує останню виправлену версію runtime. Додаток використовує найновішу версію runtime (в рамках цільового основного і додаткового сімейства .NET), встановлену в цільовій системі. Це означає, що додаток буде автоматично використовувати останню встановлену версію середовища виконання .NET.

Недоліки:

- Вимагає попередньої установки середовища виконання. Додаток може виконуватися, тільки якщо в системі, в якій воно розміщене, вже

встановлена цільова версія .NET. Є можливість налаштувати поведінку для додатка, або вимагати певну версію .NET.

- .NET може змінитися. Середовище і бібліотеки .NET можна оновлювати на комп'ютері, де виконується додаток. У рідкісних випадках це може змінити поведінку програми, якщо використовуються бібліотеки .NET, що характерно для більшості додатків.
- Обидва режими публікації за замовчуванням створюють виконавчий файл, який залежить від платформи. Додатки, що залежать від середовища, можуть створюватися без виконавчого файлу, і ці програми є крос-платформеними.

2.3 Черга повідомлень RabbitMq

RabbitMQ - це потужна відкрита система повідомлень на основі протоколу AMQP (Advanced Message Queuing Protocol), яка дозволяє розподіленій системі компонентів обмінюватися повідомленнями. Система була розроблена фірмою Rabbit Technologies Ltd. і стала відомою завдяки своїй широкій популярності та надійності [9].

RabbitMQ базується на ідеї черги повідомлень, де відправник відправляє повідомлення в чергу, а отримувач отримує його з неї. Це дозволяє розподіленій системі обробки повідомлень бути більш гнучкою та надійною, оскільки кожен компонент може працювати асинхронно та не залежить від роботи інших.

RabbitMQ включає в себе декілька ключових компонентів:

- Брокер повідомлень. Брокер відповідає за отримання, збереження та пересилання повідомлень між відправником та отримувачем.
- Відправник повідомлень. Відправник відправляє повідомлення до брокера, який зберігає його в черзі.
- Отримувач повідомлень. Отримувач отримує повідомлення з черги, яку він підписався на отримання.

- Черга повідомлень. Черга зберігає повідомлення від відправника, доки вони не будуть отримані отримувачем.
- Обмін повідомленнями. Обмін повідомленнями служить для маршрутизації повідомлень до відповідних черг.

RabbitMQ підтримує багато різних функцій, таких як точна доставка повідомлень, маршрутизація, транзакції та безпека. Вона також підтримує багато різних протоколів, таких як HTTP, MQTT та STOMP, що дозволяє легко інтегрувати її в різні системи. RabbitMQ підтримує розширення за допомогою плагінів, що дозволяє розширювати його функціональність та додавати нові протоколи та інші можливості.

Крім того, RabbitMQ дуже масштабовний і може обробляти великий обсяг повідомлень. Вона також проста у використанні та має велику кількість документації та прикладів, що дозволяє розробникам швидко та легко почати використовувати її в своїх проектах.

Незважаючи на всі переваги, RabbitMQ має деякі недоліки. Наприклад, він може бути досить повільним при обробці великих обсягів повідомлень. Також, його налаштування та конфігурування може бути складним завданням для новачків, оскільки вона має досить складну архітектуру.

Крім того, щоб використовувати RabbitMQ, потрібно мати знання протоколу AMQP та інших технологій, що може вимагати додаткової підготовки та навчання. Нарешті, якщо використовувати RabbitMQ в розподіленій системі, можуть виникнути проблеми з масштабуванням та керуванням мережею, що також потребує додаткової уваги та зусиль.

Незважаючи на те, що RabbitMQ є потужною та надійною системою повідомлень, вона також має деякі недоліки:

- Вимоги до ресурсів. RabbitMQ вимагає достатньо багато ресурсів, зокрема пам'яті та процесорного часу, для ефективної роботи.
- Складність конфігурування. Конфігурування та налаштування RabbitMQ може бути складним завданням для новачків.

- Стійкість до помилок. Якщо у RabbitMQ виникає проблема, то це може призвести до блокування черги та втрати повідомлень.
- Відсутність моніторингу. У RabbitMQ відсутній вбудований механізм моніторингу та аналізу роботи системи.
- Висока вартість. Для роботи з RabbitMQ необхідно мати достатньо великий бюджет на інфраструктуру та ресурси.
- Обмежена масштабованість. Якщо система RabbitMQ потребує масштабування, то це може бути складним та вимагати значних зусиль
- Але також варто зазначити і переваги:
- Надійність: RabbitMQ є надійною системою, яка дозволяє гарантувати доставку повідомлень між системами, навіть якщо одна з них тимчасово не працює.
- Масштабовність: RabbitMQ може легко масштабуватись в залежності від потреб вашої системи. Він може обробляти великі обсяги повідомлень та підтримувати велику кількість клієнтів.
- Гнучкість: RabbitMQ дозволяє налаштовувати повідомлення залежно від потреб вашої системи, що дозволяє вам вибирати різні стратегії для обробки повідомлень.
- Підтримка різних протоколів: RabbitMQ підтримує різні протоколи, такі як AMQP, MQTT та STOMP, що дозволяє вам використовувати його в різних сценаріях.
- Легкість використання: RabbitMQ досить простий у використанні та має велику кількість документації та прикладів, що дозволяє розробникам швидко та легко почати використовувати його в своїх проектах.
- Відкритий код: RabbitMQ є відкритим програмним забезпеченням, що дозволяє розробникам змінювати та доповнювати його функціональність за необхідності.

- Підтримка кластерів: RabbitMQ дозволяє створювати кластери для розподіленої обробки повідомлень, що дозволяє підвищувати продуктивність та забезпечувати високу доступність системи.

Отже, RabbitMQ є досить потужною та надійною системою для обміну повідомленнями між системами, що дозволяє розробникам створювати розподілені та масштабовані.

2.4 База данних MS SQL

Microsoft SQL Server (MS SQL) - це одна з найпопулярніших реляційних баз даних, розроблена корпорацією Microsoft [3]. Вона працює на Windows-платформах і забезпечує широкий набір функцій для управління даними.

MS SQL може бути використана для зберігання, організації і управління даними у різних масштабах від невеликої бази даних до великих корпоративних систем. Вона підтримує різні мови програмування, такі як C#, Java, Python, і відкриті стандарти, що дозволяє розробникам і адміністраторам легко інтегрувати її з іншими інструментами. Основні функції MS SQL:

- Створення і керування базами даних: MS SQL надає можливість створювати бази даних, таблиці, індекси, зберігати інформацію про користувачів та ролі, а також керувати правами доступу до даних.
- Запити та збережені процедури: збережені процедури - це набір SQL-запитів, які можна зберегти для майбутнього використання. Це дає можливість прискорити виконання запитів та забезпечити більшу безпеку та контроль над даними.
- Реплікація: MS SQL надає можливість копіювати дані з однієї бази даних в іншу з метою забезпечення доступу до даних на різних серверах.
- Резервне копіювання: MS SQL дозволяє створювати резервні копії баз даних для забезпечення захисту даних в разі їх втрати або пошкодження.
- Підтримка OLAP: MS SQL може бути використана як база даних для аналізу бізнес-даних, які зберігаються у великих обсягах. Для цього MS

SQL надає підтримку OLAP (Online Analytical Processing), що дозволяє аналізувати дані в режимі реального часу, здійснювати швидкий доступ до інформації та виконувати складні аналітичні запити.

- Підтримка транзакцій: MS SQL забезпечує підтримку транзакцій, що дозволяє виконувати групу запитів як єдину операцію, яка буде виконуватись атомарно, тобто або всі запити виконуються успішно, або ні один з них не виконується.
- Кластеризація: MS SQL підтримує можливість створення кластерів, що дозволяє розподілити навантаження між різними серверами, що зменшує навантаження на окремі сервери та забезпечує більшу доступність даних.
- Моніторинг та оптимізація: MS SQL надає різні інструменти для моніторингу та оптимізації баз даних. Наприклад, SQL Server Profiler дозволяє переглядати запити до бази даних, а SQL Server Management Studio надає можливість здійснювати адміністрування баз даних, настроювання, моніторинг та діагностику.

Крім того, Microsoft постійно розробляє та випускає нові версії MS SQL з покращеною функціональністю та підвищеною продуктивністю. Найновішою версією є SQL Server 2019, яка містить в собі багато нових функцій, таких як підтримка Big Data, інтеграція з Kubernetes та підтримка машинного навчання. Також, MS SQL є платформонезалежною системою, що дозволяє розгортати бази даних на різних операційних системах, таких як Windows, Linux та Docker. Це дозволяє користувачам вибирати оптимальну платформу для розгортання своєї бази даних в залежності від їх потреб.

Оскільки MS SQL є популярною системою керування базами даних, вона має велику спільноту користувачів та розробників, яка допомагає розв'язувати проблеми, надає поради щодо оптимізації та підтримки системи. MS SQL може бути використана для різних цілей, від невеликих веб-додатків до складних корпоративних систем, тому вона є однією з найбільш потужних і розширюваних реляційних баз даних на ринку. Узагальнюючи, MS SQL є потужною та

розширюваною реляційною базою даних, яка надає широкий спектр можливостей для розробки та адміністрування баз даних. MS SQL може бути використана для різних цілей, від невеликих веб-додатків до складних корпоративних систем. Це також є плюсом, якщо система матиме потенціал на розширення функіоналу.

2.5 Веб фреймворк Blazor

Так як back-end частина системи буде написана на C#, розумним вибором є використати фреймворк від Microsoft – Blazor.

Blazor - це фреймворк для створення веб-додатків, що дозволяє розробникам використовувати мову C# для написання клієнтської частини додатка, яка виконується в браузері, замість традиційних мов веб-розробки, таких як JavaScript або TypeScript [6].

Основна ідея Blazor полягає в тому, що весь код, написаний розробником на C#, компілюється в WebAssembly, що дозволяє запускати клієнтську частину додатка безпосередньо в браузері, без необхідності використовувати JavaScript. Одна з ключових особливостей Blazor - це можливість використовувати .NET-бібліотеки та фреймворки, такі як Entity Framework, ASP.NET Core, SignalR та багато інших, безпосередньо в браузері. Це забезпечує зручність та ефективність розробки, оскільки дозволяє розробникам використовувати знайомі інструменти та мови програмування для розробки веб-додатків.

Blazor має дві основні моделі програмування: Server-Side та Client-Side. Server-Side модель використовує SignalR для забезпечення взаємодії між клієнтом та сервером, тоді як Client-Side модель використовує WebAssembly для запуску коду безпосередньо в браузері. Переваги Blazor полягають у зручності та ефективності розробки, високій продуктивності та широких можливостях інтеграції з іншими .NET-бібліотеками та фреймворками. Blazor також дозволяє розробникам створювати багатоплатформні додатки, які працюють на різних пристроях та операційних системах, що забезпечує гнучкість та розширюваність проектів.

Додатково можна сказати, що Blazor підтримує різні способи взаємодії зі сторонніми бібліотеками та JavaScript-кодом, включаючи P/Invoke та JavaScript Interop. Це дозволяє розробникам легко інтегрувати Blazor з існуючим JavaScript-кодом або використовувати сторонні JavaScript-бібліотеки. Окрім цього, Blazor підтримує багатомовність, що дозволяє створювати міжнародні додатки зі зручною локалізацією. Також фреймворк надає засоби для розробки адаптивного дизайну та підтримує Progressive Web App (PWA).

Також однією з головних переваг Blazor є його безпека. Оскільки весь код виконується на боці клієнта, Blazor надає додаткові засоби захисту, такі як механізми перевірки дозволів та атрибутів безпеки, що дозволяють уникнути багатьох типів атак.

Узагалнюючи, Blazor є потужним інструментом для розробки веб-додатків, який дозволяє розробникам використовувати знайомі мови програмування та бібліотеки для створення клієнтської частини додатка, що працює безпосередньо в браузері. Це забезпечує ефективну та гнучку розробку веб-додатків з високою продуктивністю та безпекою.

2.6 Відправка сповіщень через Telegram

Telegram став важливим засобом комунікації та інформаційного обміну для українських військових, волонтерів та журналістів під час війни в Україні. У 2022 році, після початку повномасштабного вторгнення, на платформі було створено численні канали та групи, що дозволяли координувати дії військових на передовій, а також збирати гроші та зброю для потреб на фронті. Тому відправка сповіщень через телеграм є одним із пріорітетних шляхів відправки повідомлень.

Telegram Bot API - це програмний інтерфейс, що дозволяє створювати та керувати ботами в месенджері Telegram. API надає можливість розробникам створювати різноманітні функції для ботів, такі як обробка повідомлень, реакція на події в чаті, розсилка повідомлень, створення кнопок та інших інтерактивних елементів.

Для того, щоб створити бота в Telegram, розробник повинен зареєструвати його в месенджері та отримати API-ключ. Після цього можна створювати ботів з використанням різноманітних мов програмування, таких як Python, Java, PHP, Node.js, Net core та інші. Telegram Bot API містить багато різноманітних методів, що дозволяють розробникам створювати ботів з різноманітними функціями. Наприклад, метод sendMessage дозволяє боту відправляти повідомлення в чат, метод sendPhoto - відправляти фото, а метод sendAudio - аудіофайли. Також API містить методи для роботи з клавіатурами, зберіганням даних, отриманням інформації про користувачів та інше. За допомогою Telegram Bot API можна створювати ботів для різноманітних цілей, таких як повідомлення новин, створення ботів для взаємодії з користувачами в різних галузях, наприклад, бот для взаємодії з клієнтами відділу підтримки, бот для організації збору замовлень у магазині, та багато інших. Всі ці боти можуть бути легко інтегровані в Telegram, що робить їх дуже зручними та популярними серед розробників

Для того, щоб створити бота за допомогою Telegram Bot API, розробник повинен мати попередній досвід програмування та знання однієї з підтримуваних мов програмування. Для цього Telegram Bot API надає різноманітну документацію та приклади коду на різних мовах програмування.

Основні етапи створення бота за допомогою Telegram Bot API:

- Реєстрація бота в Telegram та отримання API-ключа (рисунок 2.2).
- Встановлення необхідних бібліотек та пакетів для мови програмування, яку ви використовуєте.
- Написання коду бота, включаючи функції та методи, які ви бажаєте реалізувати.
- Завантаження коду бота на сервер.
- Запуск бота та тестування його функціональності.

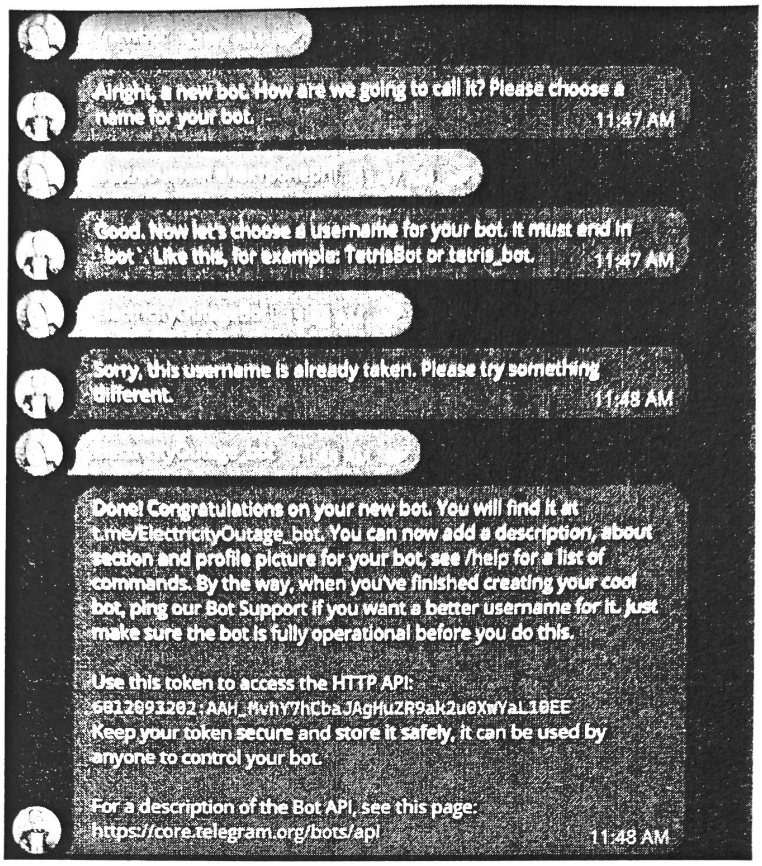

Рисунок 2.2 – Процес реєстрації Telegram Bot

Після створення бота за допомогою Telegram Bot API, розробник може почати взаємодіяти з ним та використовувати його для різноманітних цілей. Наприклад, створити бота для отримання оновлень новин, створити бота для відстеження замовлень в інтернет-магазині, або створити бота для організації голосових конференцій тощо.

Telegram Bot API дозволяє розробникам створювати ботів, які можуть стати великою допомогою в різних сферах життя. Наприклад, боти можуть бути корисними для бізнесу, медіа, навчання та інших галузей. Крім того, вони можуть допомагати користувачам в багатьох сферах, надавати інформацію, розважати, допомагати при розв'язанні різних завдань та інше.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Структура бази данних

Структура бази даних в серверній частині системи сповіщення складається з чотирьох таблиць, які забезпечують організацію та збереження даних. Усі таблиця є взаємопов'язаними (рисунок 3.1).

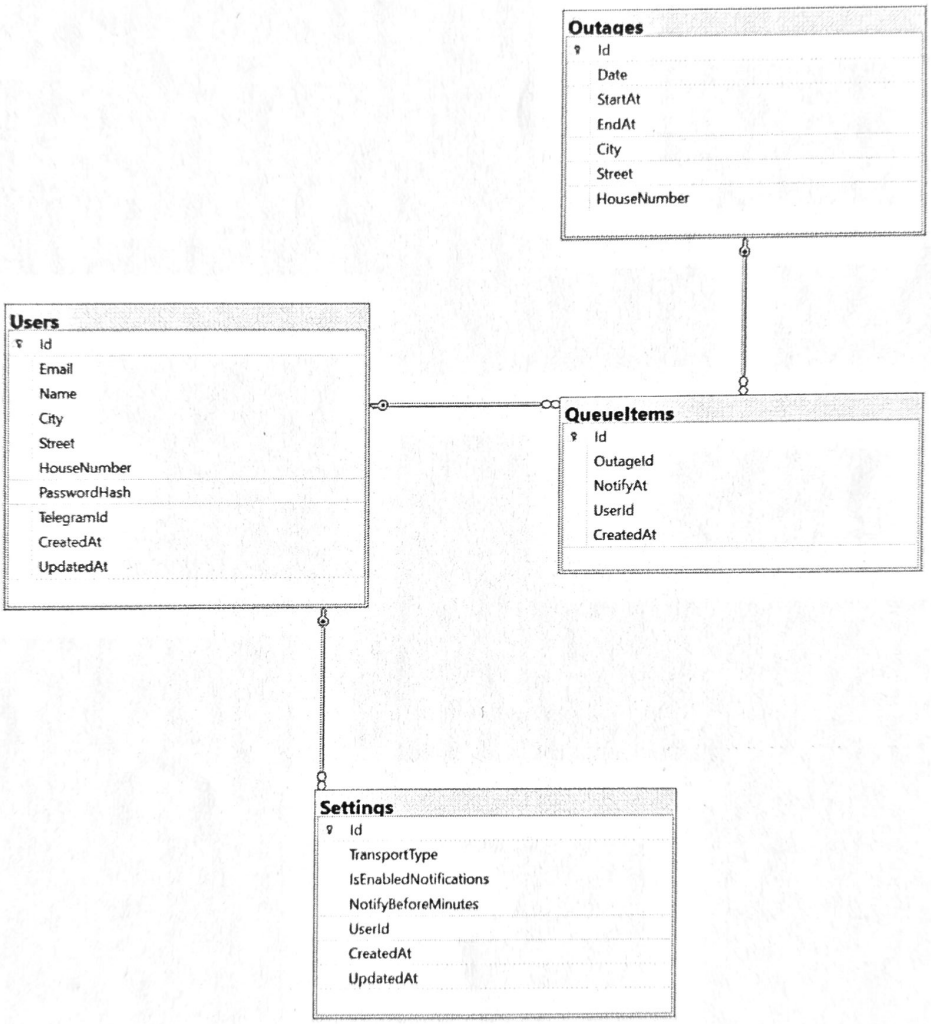

Рисунок 3.1 – Діаграма бази даних

В якості СУБД використано MS SQL Server. Система сповіщень може скористатися перевагами потужних функцій MS SQL Server, включаючи цілісність, безпеку та масштабованість даних, завдяки використанню його в якості базової технології баз даних. Ці чотири таблиці складають структуру бази даних, яка забезпечує ефективну організацію, зберігання та пошук даних, що призводить до безперебійної комунікації та оперативної доставки сповіщень користувачам. Система сповіщень на базі MS SQL Server пропонує надійний і систематичний метод підтримки ефективної комунікації та розповсюдження інформації, будь то сповіщення, оновлення або важлива інформація.

У таблиці «Users» міститься важлива інформація про користувачів системи, яка використовується для їх реєстрації та аутентифікації. Поля, що містяться в цій таблиці:

- Ідентифікатор користувача (ID): Це унікальне значення, яке ідентифікує кожного користувача в системі. Воно може бути автоматично генерованим або встановлюватись користувачем під час реєстрації.
- Ім'я (Name): Додавання окремих полів для імені дозволяє точніше зберігати особисті дані користувачів. Це може бути корисним, наприклад, при вітанні користувача у повідомленнях або на сторінках системи.
- Електронна пошта (Email): Поле електронної пошти дозволяє зберігати унікальну адресу кожного користувача. Воно може використовуватись для надсилання сповіщень, відновлення паролю та іншої комунікації з користувачем.
- Пароль (PasswordHash): Це важливе поле, яке зберігає захешований або зашифрований пароль користувача.
- Ідентифікатор телеграм чату (TelegramId): Поле потрібне для того що б зв’язати чат юзера після підключення бота. Це 64 число, яке автоматично генерується системою Telegram.

- Місто (City), Вулиця(Street) та Номер будинку(HouseNumber): Відокремлення міста, вулиці та будинку спрощує процес пошуку данної адресси на данних з графіків відключень.
- Дата реєстрації (CreatedAt): Це поле вказує дату та час реєстрації кожного користувача в системі. Воно може бути використане для відстеження активності користувачів та аналізу динаміки зростання користувацької бази.
- Дата останньою зміни (UpdatedAt): Це поле вказує дату та час останньої зміни данних користувача. Воно може бути використане для відстеження активності користувачів та аналізу динаміки зростання користувацької бази.

Таблиця «Settings» має інформацію про на налаштування встановленні користувачем та допоміжні поля UserId, CreatedAt та UpdatedAt. До ключових полів налаштувань відносяться:

- Параметр NotifyBeforeMinutes вказує, за скільки хвилин користувач бажає бути поінформованим про подію або явище заздалегідь. Система сповістить користувача, наприклад, за 30 хвилин до події, якщо користувач встановить значення 30.
- Поле IsEnabledNotifications, показує, чи користувач увімкнув сповіщення. Користувач отримуватиме сповіщення, якщо воно має значення «так» або «істина», і навпаки, якщо воно має значення «ні» або «брехня», сповіщення не будуть ввімкнені.
- TransportType: У цьому полі відображається обраний користувачем тип сповіщень. Він може бути налаштований на надсилання електронних листів, Telegram або інший тип сповіщень якій може бути доданий у систему. Завдяки цьому користувач може вибрати, які канали зв'язку він хоче використовувати для отримання сповіщень.

Таблиця «QueueItems» це допоміжна таблиця має інформацію про елементи черги. Також ця таблиця використовується для історії сповіщення для

користувача. Ця таблиця має поле NotifyAt, яке має в собі точний час відправки сповіщення та допоміжні поля OutageId(посилання на об'єкт відключення), UserId та CreatedAt.

Таблиця «Outages» це таблиця з інформацію детальною інформацією про місці відключення та про його тривалість. Адреса розбивається так само, як адреса вказується користувачем (City, Street, HouseNumber). Інформація про тривалість зберігається у двох полях таблиці. StartAt і EndAt – відповідно дата початку і дата завершення відповідно.

3.2 Опис алгоритму автоматичною відправки повідомлень

Алгоритм роботи автоматичною відправки повідомлень складається з послідовного виклику функцій для обробки та відправки повідомлень. Робота програми виділення в цикли (рисунок 3.2).

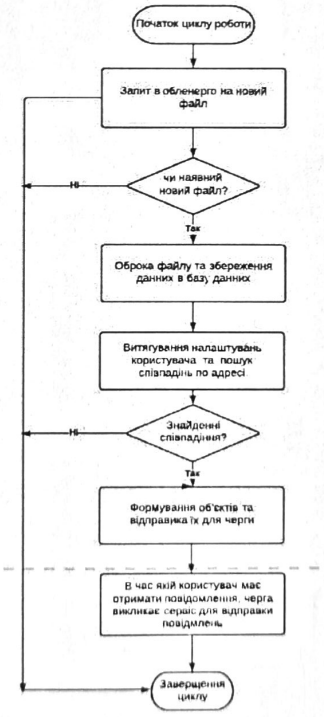

Рисунок 3.2 – Модель роботи циклу

За виклики циклу відповідає клас TimerService (рисунок 3.3) це сервіс якій працює за патерном програмування «Singleton». Це означає, що під першого виклику класу, створюється екземпляр классу, який буде існувати до кінця виконання програми. Це потрібно для того що б екземпляр класу не перестворювався при повторному виклику. Під час ініціалізації екземпляру класу, за допомогою функції InitSettings налаштовується час періоди виклику циклу.

```
public class TimerService : IDisposable
{
    private readonly IConfiguration _configuration;
    private readonly ILogger<TimerService> _logger;
    private readonly CycleService _cycleService;

    private int _cycleTimeoutInMinutes = 0;
    private bool _isStartedTimer;
    private System.Timers.Timer _timer;

    0 usages
    public TimerService(IConfiguration configuration, ILogger<TimerService> logger, CycleService cycleService)
    {
        _configuration = configuration;
        _logger = logger;
        _cycleService = cycleService;
        InitSettings();
    }

    1 usage
    private void InitSettings()
    {
        var timerConfiguration string? = _configuration["Timer:CycleTimeout"];
        if (int.TryParse(timerConfiguration, out var result int))
        {
            _logger.LogInformation( message: $"Timer configuration value in minutes {result}");
            _cycleTimeoutInMinutes = result;
        }
    }

    0 usages
    public void StartTimer()
    {
        _timer = new System.Timers.Timer( interval: _cycleTimeoutInMinutes * 60 * 1000);
        _timer.Elapsed += TimerOnElapsed;
        _timer.Enabled = true;
        _timer.AutoReset = true;
    }

    1 usage
    private void TimerOnElapsed(object? sender, ElapsedEventArgs e)
    {
        _logger.LogInformation( message: $"Timer elapsed");
        _cycleService.RunCycle();
    }

    0 usages
    public void StopTimer()
    {
        _timer.Stop();
    }

    public void Dispose()
    {
        _timer.Dispose();
    }
}
```

Рисунок 3.3 – Програмний код TimerService

3.3 Реалізація отримання нових данних

Першим кроком у роботі циклу програми є отримання данних з сайту Волиньобленерго. Процес відбувається натсупним чинном, викликається функція DownloadReportForDay з класу ReportService (рисунок 3.4). Функція приймає параметр c# класу DateTime, в цьому параметрі вказується день, за має бути завантажений файл.

Принцип роботи функції є наступний. Перевіряється, чи день за якій ми хочемо завантажити файл, є сьоднішнім, оскільки від цього залежить шлях де він буде знаходитись на сайті Волиньобленерго. Потім іде перевірка чи файл вже існує на цю дату. Якщо ні то виконується завантаження файлу у створенну дерикторію під вказаний день.

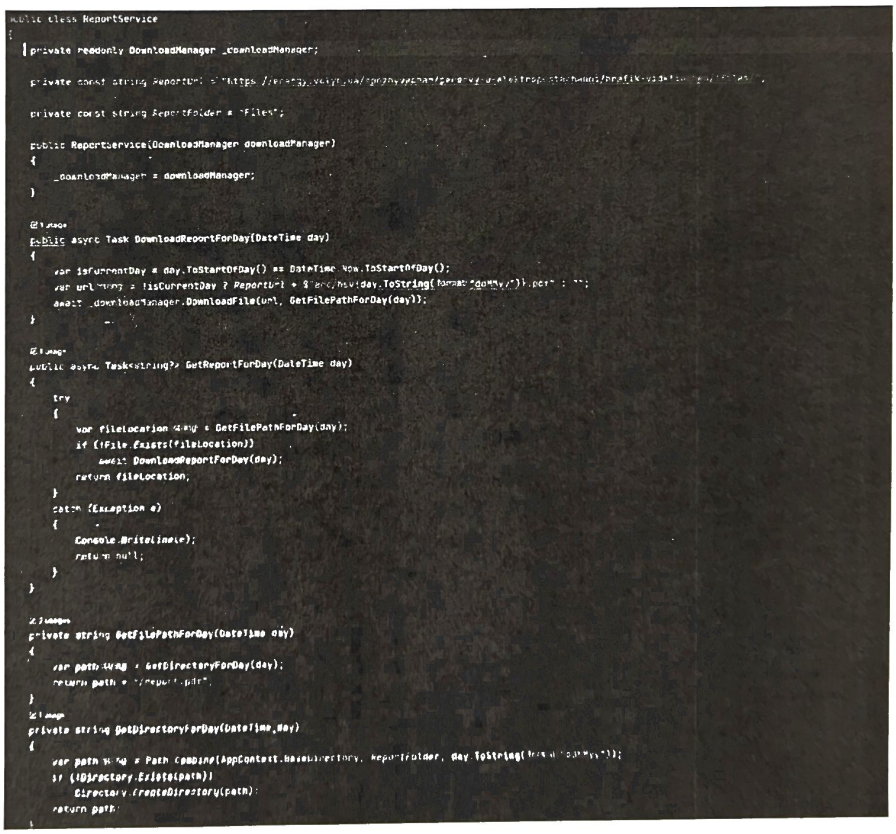

Рисунок 3.4 – Програмний код класу ReportService

3.4 Реалізація роботи з чергою RabbitMq

Для забезпечення ефективної комунікації та обміну даними між різними компонентами системи в реальному часі, а також для гарантії надійності та стійкості до збоїв, використовується сервіс черги RabbitMQ. Цей сервіс заснований на протоколі AMQP (Advanced Message Queuing Protocol) і використовується в багатьох сферах, включаючи розподілені системи, мікросервісну архітектуру, хмарні рішення та багато іншого.

Однією з головних переваг RabbitMQ є можливість відправки повідомлень з значним часом життя. Це означає, що повідомлення можуть бути збережені в черзі протягом тривалого часу, доки вони не будуть оброблені або доставлені до відповідного компонента системи [10]. Це особливо корисно у випадку, коли отримувач повідомлення тимчасово недоступний або система перезавантажується.

Використання черги RabbitMQ дозволяє забезпечити надійну доставку повідомлень. Коли відправник надсилає повідомлення в чергу, воно не втрачається, навіть якщо приймач на даний момент недоступний або відсутній. RabbitMQ зберігає повідомлення в черзі та намагається доставити його приймачеві, коли він стає доступним. Крім того, RabbitMQ забезпечує можливість підтвердження доставки, що дозволяє відправнику отримати підтвердження про успішне отримання повідомлення та обробку його отримувачем.

Однією з істотних переваг RabbitMQ є його гнучкість та можливість налаштування під різні потреби системи. Він підтримує різні сценарії обробки повідомлень, включаючи публікацію-підписку, черги повідомлень з пріоритетами та маршрутизацію повідомлень. Це дає можливість розробникам налаштовувати поведінку черги відповідно до специфічних вимог системи і забезпечувати оптимальну обробку повідомлень. Крім того, RabbitMQ є масштабованим рішенням, що може бути розгорнутим у режимі кластеру. Це означає, що ви

можете створити кластер з кількох вузлів RabbitMQ, що спільно обробляють навантаження та забезпечують високу доступність. Кластеризація дозволяє розподілити навантаження між вузлами, збільшити продуктивність та забезпечити резервування, що гарантує безперебійну роботу системи навіть у разі збоїв окремих вузлів.

Consumer (споживач) на діаграмі (рисунок 3.5) елемент «Формування елементів черги» і Producer (виробник) на діаграмі (рисунок 3.5) елемент «Отримання повідомлення» є двома ключовими поняттями в системі RabbitMQ. Consumer є складовою частиною додатка, який отримує повідомлення з черги RabbitMQ для їх обробки, а Producer (рисунок 3.6) який відправляє повідомлення в чергу RabbitMQ.

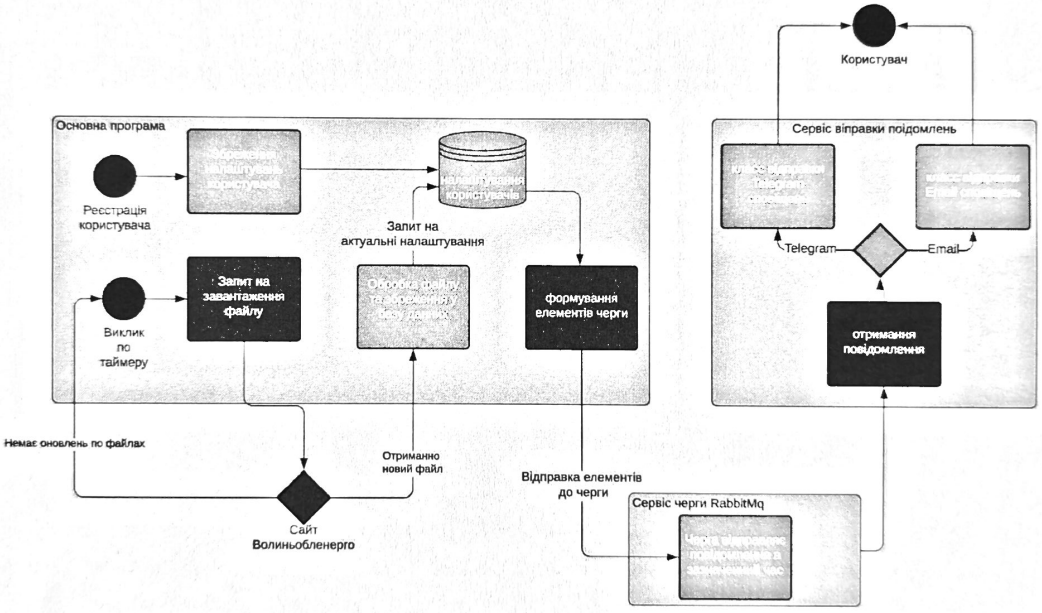

Рисунок 3.5 – Діаграма комунікації системи

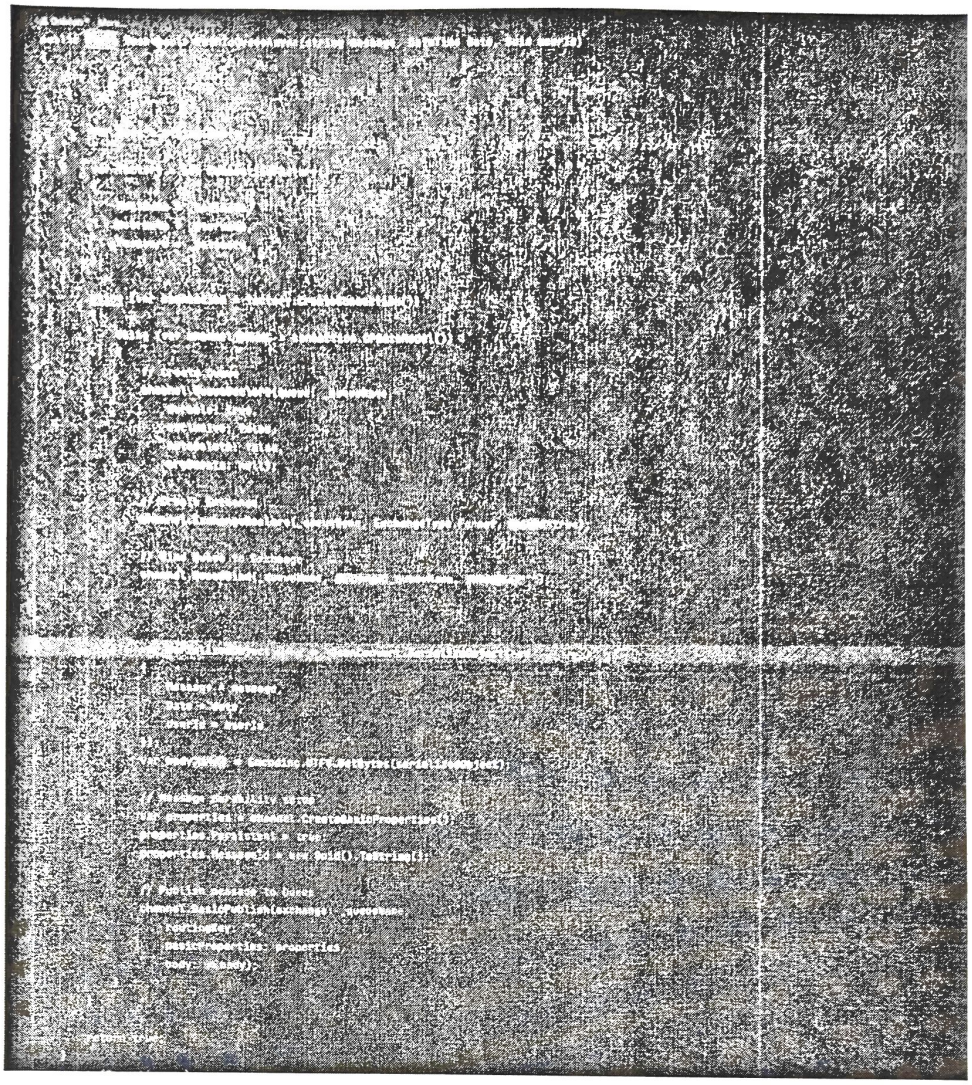

Рисунок 3.6 – Реалізація Producer

3.5 Реалізація відправки сповіщень

Сервіс відправки сповіщень реалізований у формі патерну Фабричний метод [5], використання данного патерну має певні переваги:

- Розширюваність: Фабричний метод дозволяє легко додавати нові типи сповіщень без необхідності змінювати існуючий код. Кожен новий тип сповіщення може бути представлений підкласом, який реалізує спільний інтерфейс. Це забезпечує гнучкість і дозволяє легко розширювати функціональність системи сповіщень.

- Локалізація: Використання фабричного методу полегшує роботу з локалізацією сповіщень. Кожен тип сповіщення може мати свої власні локалізовані рядки для різних мов або регіонів. Фабричний метод дозволяє створити відповідний тип сповіщення з правильними локалізованими рядками, забезпечуючи таким чином ефективну підтримку локалізації.
- Інкапсуляція: Використання фабричного методу дозволяє інкапсулювати процес створення об'єктів сповіщень. Клієнтський код, який відправляє сповіщення, не повинен знати про конкретний клас сповіщення. Він просто використовує фабричний метод для створення об'єкта сповіщення, не вдаючись у деталі його реалізації.
- Тестування: Фабричний метод полегшує тестування системи сповіщень. Замість того, щоб створювати об'єкти сповіщень безпосередньо в коді тестів, можна використовувати фабричний метод для створення тестових подвійників або замінників об'єктів сповіщень. Це полегшує налаштування тестів та забезпечує більшу контрольованість у випадку тестування різних сценаріїв.
- Оптимізація ресурсів: Фабричний метод може бути використаний для впровадження оптимізаційних стратегій. Наприклад, в залежності від умови або конфігурації системи, фабричний метод може повернути пул попередньо створених об'єктів сповіщень замість створення нових об'єктів кожного разу. Це може покращити продуктивність системи і ефективно використовувати ресурси.

Ці переваги роблять патерн «фабричний метод» корисним інструментом для розробки системи сповіщень з різними типами. Він сприяє легкості розширення, локалізації, інкапсуляції, тестування та оптимізації ресурсів.

Для реалізації патерну фабричного методу для різних типів сповіщень, спочатку створюється новий клас, який представляє певний вид сповіщень і наслідується від абстрактного класу AbstractTransport (рисунок 3.7).

AbstractTransport виступає в якості базового класу, в якому оголошені абстрактні методи, які будуть реалізовані в похідних класах.

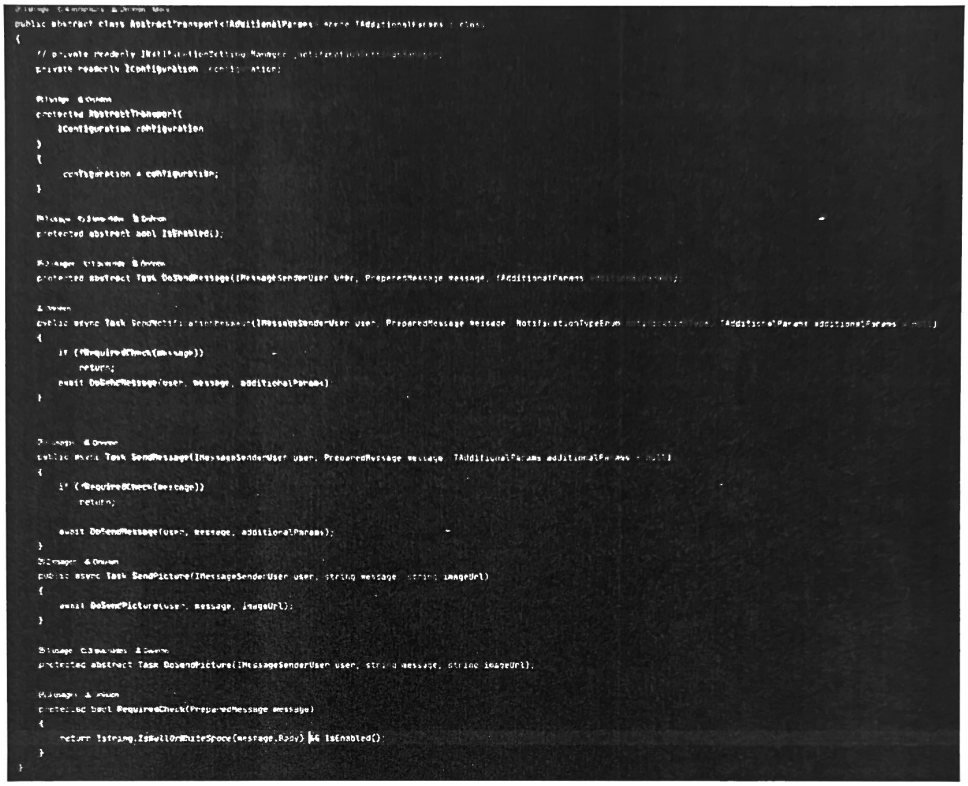

Рисунок 3.7 – Код абстрактного транспорту

Клас AbstractTransport може містити загальну логіку і функціональність, що є спільною для всіх типів сповіщень. Він також може включати абстрактний метод для надання шаблону, який нащадкові класи повинні реалізувати. Цей абстрактний метод може бути, наприклад, методом «відправити», який буде викликатися для відправки сповіщень. Коли потрібно створити новий тип сповіщення, створюється похідний клас від AbstractTransport. У цьому похідному класі реалізуються всі абстрактні методи, оголошені в AbstractTransport, а також можуть бути додані додаткові методи або перевантажені існуючі методи за потреби. Цей похідний клас представляє конкретну реалізацію сповіщення з використанням певного виду транспорту. При використанні патерну фабричного

методу клієнтський код, який відправляє сповіщення, викликає метод фабрики замість прямого створення об'єкту сповіщення. Фабричний метод, який знаходиться в реалізації похідного класу, створює об'єкт відповідного типу сповіщення і повертає його клієнтському коду. Тим самим забезпечується інкапсуляція процесу створення об'єктів сповіщень.

Це дозволяє релалізувати відправку сповіщень через Telegram [12] та через Email [11] інкапсульовано від базової логіки, та спрощує додавання нових типів сповіщень шляхом створення нових похідних класів від AbstractTransport і реалізації необхідних методів. Це дозволяє легко розширювати функціональність системи сповіщень без впливу на існуючий код.

3.6 Інструкція реєстрації та налаштування сповіщень

Для початку використання програми користувачу потрібно (рисунок 3.7):

1. Зайти на веб посилання, де знаходиться система та натиснути кнопку зареєструватися (рисунок 3.8).

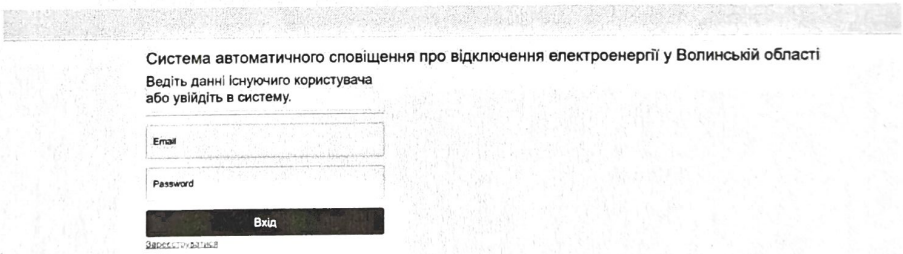

Рисунок 3.8 – Інтерфейс входу у систему

2. Наступним кроком користувач повинен внести свої данні, Email на якій користувач бажає отримувати повідомлення, Ім'я, детальну інформацію про адресу за якою користувач бажає отримувати сповіщення та

створити пароль для входу у систему. Після завершення вводу данних користувач має натиснути на кнопку зареєструватись (рисунок 3.9).

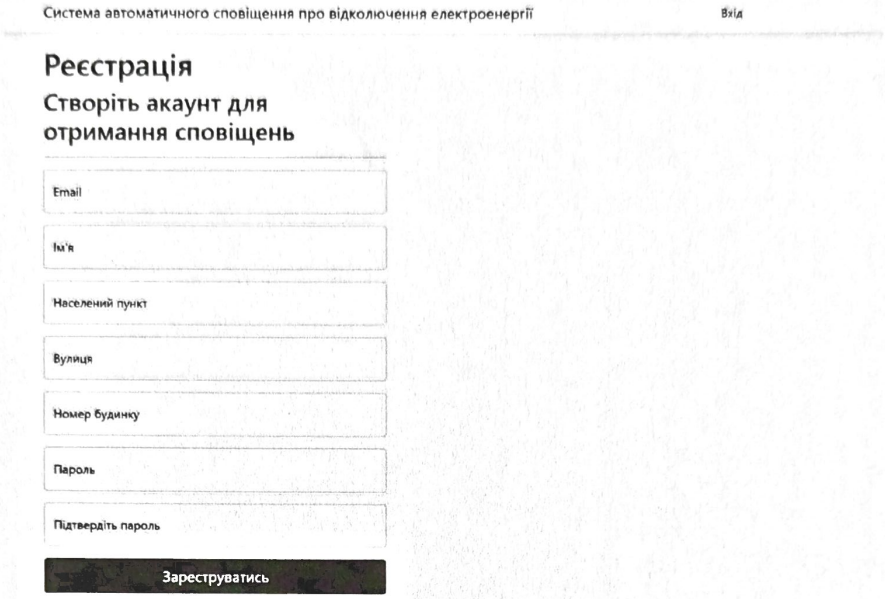

Рисунок 3.9 – Інтерфейс реєстрації користувача

3. Після реєстрації користувача направить на стартову сторінку системи (рисунок 3.10), на якій користувач побачить навігаційне мені на лівій частині інтерфейсу. На правій частині, користувач побачить розташування вказаної адреси на карті та список історій сповіщень.
4. Для підключення та зміни налаштувань сповіщень користувач потрібний перейти на сторінку «Налаштування».
5. На данній сторінці (рисунок 3.11) користувач може виставити основні налаштування роботи системи. Користувач може вибрати час за скільки часу до сповіщення та вибрати вид сповіщень, яких він отримувати. Для отримування сповіщень через Email достатньо просто поставити відмітити відповідний чекбокс. Для Telegram-бота сповіщень окрім відповідної відмітки також потрібно підключити Telegram, для цього

потрібно натиснути на посилання під чекбоксом «Відкрити Telegram-бота».

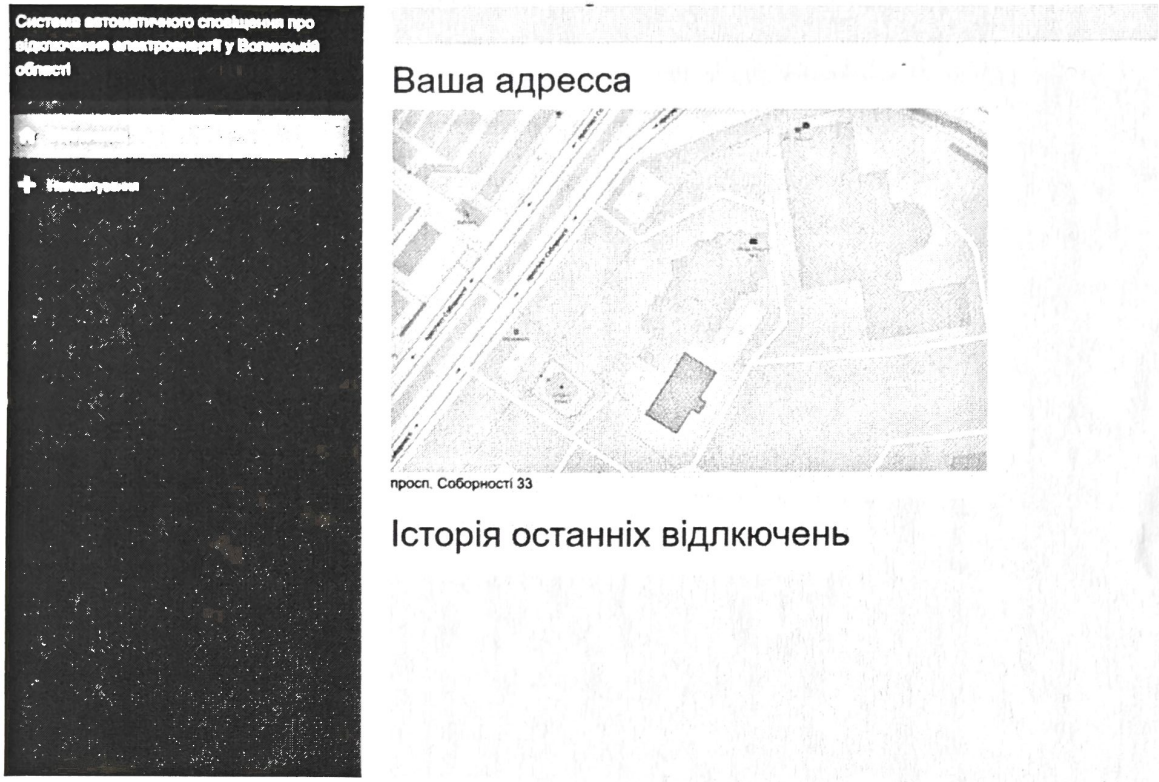

Рисунок 3.10 – Інтерфейс стартової сторінки системи

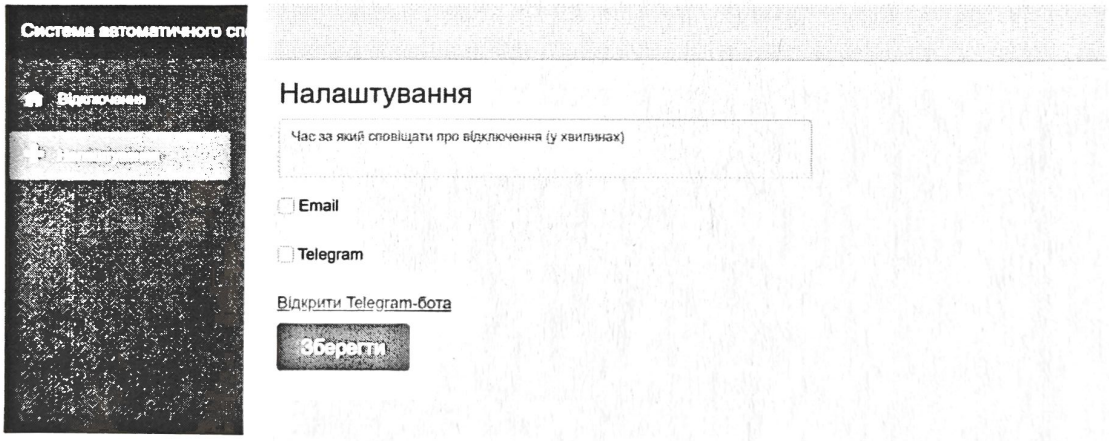

Рисунок 3.11 – Інтерфейс сторінки налаштувань

6. Перейшовши по посиланню, користувач має натиснути на кнопку старт (рисунок 3.12), після чого він отримає у відповідь повідомлення про завершення реєстрації.

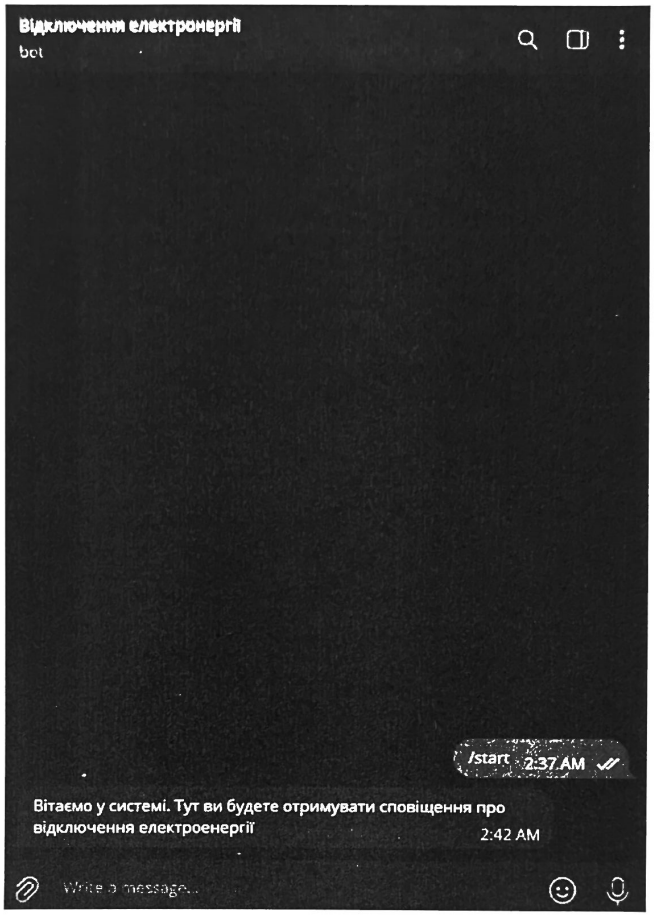

Рисунок 3.12 – Інтерфейс реєстрації у телеграм боті

7. Після реєстрації, в залежності від вибраних налаштувань, користувач буде отримувати сповіщення(рисунок 3.13 та рисунок 3.14) про відключення по вказаній адресі.

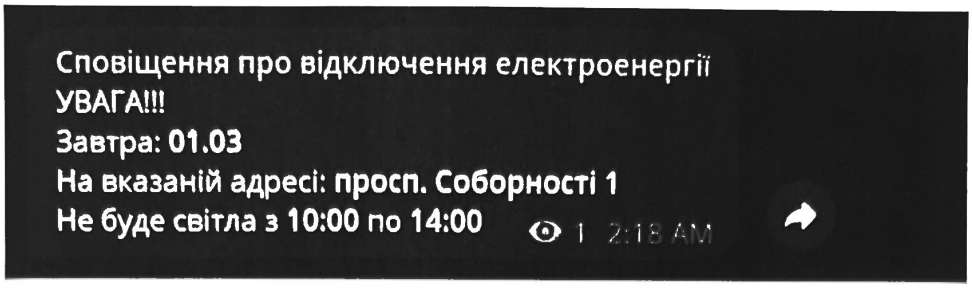

Рисунок 3.13 – Приклад сповішеня у телеграм бот

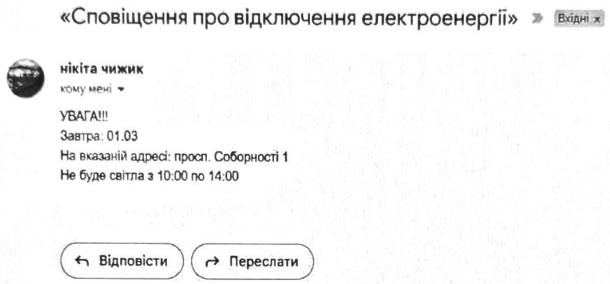

Рисунок 3.14 – Приклад сповішеня через Email

Після успішної завершення процесу реєстрації, користувач отримує повний доступ до системи і може вільно користуватись всіма її функціями та можливостями. Він має можливість налаштувати свої особисті налаштування залежно від своїх потреб та вимог. Одним з важливих функціональних елементів системи є можливість отримувати сповіщення про відключення. Користувач може вказати певні налаштування, за якими будуть надсилатися сповіщення, що пов'язані з відключенням.

ВИСНОВКИ

В результаті даної роботи спроектовано та реалізовано систему автоматичного сповіщення про відключення електроенергії у Волинській області.

Проведено аналіз предметної області, висновки з якого доводять, що автоматизація процесу отримання інформації про планові відключення буде вкрай важлива для оптимізації процесів планування роботи підприємств та зможе спростити та покращити досвід отримання інформації про майбутні планові відключення для кінцевих користувачів.

На основі аналізу системи-аналога для споживачів компанії Yasno та користувачів мобільного оператору lifecell визначено, що основними недоліками даної системи є відсутність можливості вказувати безпосередньо саму адресу споживання та данна система обмеженна смс-повідомленнями, що в свою чергу обмежує користувачів у доступі до сповіщень на інших пристроях, окрім мобільного телефону. Також в даній системи відсутній веб інтерфейс, на якому користувач мав би можливість перегляду історії сповіщень та налаштувань самих сповіщень.

На основі цих недоліків аналога, спроектовано і розроблено структуру та архітектуру програми, яка має можливість сповіщати користувача за допомогою інших типів сповіщень та має в наявності веб інтерфейс для налаштувань даної системи. Також обрано інструменти та засоби які потрібні для реалізації системи з відповідними можливостями.

Отже в результаті роботи розроблено систему сучасних автоматизованих сповіщень для цивільного населення про планові відключення електроенергії у Волинській області, за допомогою даної системи кінцевий користувач матиме можливість отримувати сповіщення про відключення електроенергії через email та telegram і інформативний інтерфейс для перегляду історії та актуального графіку відключень по вибраний користувачем адресі. Також розробленно повну інструкцію з реєстрації та налаштувань спровіщень для кінцевого користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. European Business Association. European Business Association. URL: https://eba.com.ua/ (дата звернення: 24.02.2023).
2. Волиньобленерго URL: https://energy.volyn.ua/. (дата звернення: 20.01.2023).
3. Учасники проектів Вікімедіа. Microsoft SQL Server – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Microsoft_SQL_Server (дата звернення: 29.12.2022).
4. Державна служба статистики – URL: https://www.ukrstat.gov.ua (дата звернення: 29.12.2022)..
5. Фабричний метод. Refactoring and Design Patterns. URL: https://refactoring.guru/uk/design-patterns/factory-method (дата звернення: 01.03.2023).
6. ASP.NET Core Blazor. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-gb/aspnet/core/blazor/?view=aspnetcore-7.0 (date of access: 08.02.2023).
7. Contributors to Wikimedia projects. .NET - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/.NET (дата звернення: 15.12.2022).
8. Messaging that just works – RabbitMQ. Messaging that just works – RabbitMQ. URL: https://www.rabbitmq.com/. (дата звернення: 19.01.2023).
9. RabbitMQ Delayed Messages 101: How to Delay & Schedule Messages Made Easy - Learn | Hevo. Learn | Hevo. URL: https://hevodata.com/learn/rabbitmq-delayed-message (дата звернення: 09.02.2023).
10. Send email from a printer, scanner, or app - Google Workspace Admin Help. Google Help. URL:

https://support.google.com/a/answer/176600?hl=en (дата звернення: 16.03.2023).

11. Telegram Bot API. Telegram APIs. URL: https://core.telegram.org/bots/api (дата звернення: 09.03.2023).
12. Tutorial: Create a web API with ASP.NET Core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio. (date of access: 16.03.2023).
13. YASNO. URL: https://yasno.com.ua/news/yasno_news/YASNO_and_lifecell_will_remind_Kyiv_residents_about_the_time_of_probable_power_outages_via_SMS (дата звернення: 11.01.2023).

ДОДАТКИ

Додаток А

Фрагмент коду відправки Telegram сповіщення

```
using System.Threading.Tasks;
using Microsoft.Extensions.Configuration;
using EOS.Transports.Business.StoreService;
using EOS.Transports.Domain.Messaging.Notifications;
using EOS.Transports.TelegramBot;

namespace EOS.Transports
{
   public class TelegramTransport : AbstractTransport
   {
      private readonly ITelegramBot _telegramBot;
      private readonly string _botKey;
      public TelegramTransport(
         ITelegramBot telegramBot,
         IConfiguration configuration
      ) : base(
         configuration)
      {
         _telegramBot = telegramBot;
         _botKey = configuration["Telegram:Key"];
      }

      protected override bool IsEnabled() => true;

      protected override bool CanSend(NotificationSettings notificationSettings) =>
notificationSettings.IsTelegramEnabled;
```

```
        protected override async Task DoSendPicture(IMessageSenderUser user, string
message, string imageUrl)
        {
            await _telegramBot.SendPicture(user, imageUrl, message, _botKey);
        }

        protected override async Task DoSendMessage(IMessageSenderUser user,
PreparedMessage preparedMessage)
        {
            await _telegramBot.SendMessage(user, preparedMessage.Body, _botKey);
        }
    }
}
```

Додаток Б

Фрагмент коду сторінки налаштувань користувача

```
@page "/settings"

   <main role="main" class="pb-1">

        <h3>Налаштування</h3>
      <div class="row">
        <div class="col-md-4">
          <section>
            <form id="account" >
              <div class="form-floating mb-3">
                <input class="form-control" aria-required="true"  type="number"
data-val="true"  id="Input_Email" name="Input.Email" value="">
                <label class="form-label" for="Input_Email">Час за який сповіщати
про відключення (у хвилинах)</label>
                <span class="text-danger field-validation-valid" data-valmsg-
for="Input.Email" data-valmsg-replace="true"></span>
              </div>
              <div class="checkbox mb-3">
                <label class="form-label" for="Input_RememberMe">
                  <input class="form-check-input" type="checkbox" data-val="true"
data-val-required="The Remember me? field is required." id="Input_RememberMe"
name="Input.RememberMe" value="true">
                  Email
                </label>
              </div>
              <div class="checkbox mb-3">
```

```
                    <label class="form-label" for="Input_RememberMe">
                        <input class="form-check-input" type="checkbox" data-val="true"
data-val-required="The Remember me? field is required." id="Input_RememberMe"
name="Input.RememberMe" value="true">
                        Telegram
                    </label>
                </div>
                <a class="mb-1" href="http://t.me/ElectricityOutage_bot">Відкрити
Telegram-бота</a>
                <div>
                    <button id="login-submit" class="w-25 btn btn-lg btn-primary mt-
2">Зберегти</button>
                </div>
                </form>
            </section>
        </div>
    </div>


  </main>
</div>
```

Додаток В

Фрагмент коду сторінки налаштувань користувача

```
public class EOS.ParserCP
{

   private const int _startDataCol = 2;
   private const int _startDataRow = 1;

   public static List<Statistic> GetStatistic(string filePath, string sheetName)
   {
      var data = GetDataTable(filePath, sheetName);
      var res = AnalyzeTable(data);
      return res;
   }

   private static List<Statistic> AnalyzeTable(DataTable data)
   {
      var res = new List<Statistic>();
      var Locations = GetLocationData(data);
      foreach (DataRow row in data.Rows.Cast<DataRow>())
      {
         if (DateTime.TryParse(row.ItemArray[_startDataRow].ToString(), out var
reportDate))
         {
            var LocationIndex = 0;
            for (var i = _startDataCol; i < data.Columns.Count - 1; i+= 3)
            {
               var startDateCol = data.Columns[i];
               var enddateCol = data.Columns[i + 1];
```

```
                if (int.TryParse(row[startDateCol].ToString(), out var startDate) &&
                    int.TryParse(row[enddateCol].ToString(), out var enddate))
                {
                    res.AddRange(GetData(reportDate,Locations[LocationIndex], startDate,
enddate));
                }
                LocationIndex++;
            }
        }
    }
    return res;
}

private static IEnumerable<Statistic> GetData(DateTime reportDate, LocationData
Location, int startDate, int enddate)
{
    var res = new List<Statistic>();
    var enddatePerDay = enddate / Location.Days;
    var roomPerDay = startDate / Location.Days;
    foreach (var day in Enumerable.Range(1, Location.Days))
    {
        res.Add(new Statistic
        {
            ReportDate = reportDate,
            StayDate = new DateTime(Location.Year, Location.Location, day),
            StartDate = roomPerDay,
            enddate = enddatePerDay,
        });
    }
```

```
        res[0].StartDate += startDate % Location.Days;
        res[0].enddate += enddate % Location.Days;
        return res;
    }


    private static List<LocationData> GetLocationData(DataTable dataTable)
    {
        var res = new List<LocationData>();
        var row = dataTable.Rows[_startDataRow - 1];
        foreach (DataColumn col in dataTable.Columns)
        {
            var rowText = row[col].ToString();
            if (string.IsNullOrEmpty(rowText) || !DateTime.TryParse(rowText, out var
date))
                continue;
            var data = GetDataByLocation(date.Location);
            res.Add(new LocationData
            {
                Year = data.Year,
                Location = date.Location,
                Days = data.DayCount
            });
        }
        return res;
    }

    private static DataTable GetDataTable(string filePath, string sheetNames)
```

```
    {
        using var edr = ExcelDataReader.Create(filePath);
        //it's a magic fix for hidden sheet
        var isRead = edr.TryOpenWorksheet(sheetNames);
        do
        {
            if (edr.WorksheetName != sheetNames)
                continue;
            var dataTable = new DataTable();
            dataTable.Load(edr);
            return dataTable;
        } while(edr.NextResult());
        return null;
    }
}
```

Додаток Г

Фрагмент коду відправки Email-сповіщення

```
public class EmailTransport : AbstractTransport
  {
    private readonly IConfiguration _configuration;

    public EmailTransport(
      // INotificationSettingsManager notificationSettingsManager,
      IConfiguration configuration
    ) : base(
      // notificationSettingsManager,
      configuration)
    {
      _configuration = configuration;
    }

    // protected override bool IsEnabled() => TenantProperties.EmailFeatureEnabled;
    protected override bool IsEnabled() => true;


    protected override bool CanSend(NotificationSettings notificationSettings)
    {
      return notificationSettings.IsEmailEnabled;
    }

    protected override Task DoSendPicture(IMessageSenderUser user, string message,
string imageUrl)
    {
      throw new NotImplementedException();
```

```
    }

    // protected override bool IsCashierTransportAllowed(CashierTransportsSettings
cashierTransportsSettings)
    // {
    //     return cashierTransportsSettings.Email;
    // }

    protected override Task DoSendMessage(IMessageSenderUser user,
PreparedMessage message)
    {
      if (user is AppIdentityUser appIdentityUser)
        return SendMessageWithAttachments(appIdentityUser, message, null);
      throw new ArgumentException("EmailTransport cant work with non
AppIdentityUser user", nameof(user));
    }
    public Task SendMessageWithAttachments(AppIdentityUser user,
PreparedMessage message, List<string> attachments)
    {
      return SendMessageWithAttachments(user.Email, message, attachments);
    }

    public Task SendMessageWithAttachments(string userEmail, PreparedMessage
message, List<string> attachments)
    {
      if (!RequiredCheck(message))
        return Task.CompletedTask;
```

```
        var client = new SmtpClient(_configuration["EmailConnection:Server"],
_configuration.GetValue<int>("EmailConnection:Port"))
        {
            UseDefaultCredentials = false,
            EnableSsl = true,
            Credentials = new
NetworkCredential(_configuration["EmailConnection:UserName"],
_configuration["EmailConnection:Password"]),
            DeliveryMethod = SmtpDeliveryMethod.Network
        };
        var mailMessage = new MailMessage
        {
            From = new MailAddress(_configuration["EmailConnection:UserName"]),
            Body = message.Body,
            Subject = message.Subject,
            IsBodyHtml = true
        };
        attachments?.ForEach(file =>
        {
            mailMessage.Attachments.Add(new Attachment(file));
        });

        mailMessage.To.Add(userEmail);
        client.SendCompleted += (s, e) => {
            client.Dispose();
            mailMessage.Dispose();
        };

        return client.SendMailAsync(mailMessage)    }}
```

Додаток Д

Фрагмент коду класу RabbitMqConsumer

```
public class ConsumerQueueService : IConsumerQueueService
  {
    private readonly IConfiguration _configuration;
    private readonly IDeviceService _deviceService;
    private readonly ILogger<ConsumerQueueService> _logger;

    ConnectionFactory _factory { get; set; }
    IConnection _connection { get; set; }
    IModel _channel { get; set; }

    private readonly string _hostname;
    private readonly string _queueName;
    private readonly string _exchangeName;
    private readonly string _exchangeQueueKey;
    private readonly string _username;
    private readonly string _password;

    public ConsumerQueueService(
      IOptions<RabbitMqConfModel> rabbitMqOptions,
      IConfiguration configuration,
      IDeviceService deviceService,
      ILogger<ConsumerQueueService> logger
    )
    {
      _hostname = rabbitMqOptions.Value.Hostname;
      _queueName = rabbitMqOptions.Value.QueueName;
      _exchangeName = rabbitMqOptions.Value.ExchangeName;
```

```
    _exchangeQueueKey = rabbitMqOptions.Value.ExchangeQueueKey;
    _username = rabbitMqOptions.Value.UserName;
    _password = rabbitMqOptions.Value.Password;
    _configuration = configuration;
    _deviceService = deviceService;
    // _logger = logger;
}

public Task RunListenerTaskQueue()
{
    try
    {
        _factory = new ConnectionFactory()
        {
            HostName = _hostname,
            UserName = _username,
            Password = _password
        };

        _connection = _factory.CreateConnection();
        _channel = _connection.CreateModel();

        _channel.QueueDeclare(queue: _queueName,
            durable: true,
            exclusive: false,
            autoDelete: false,
            arguments: null);

        _channel.BasicQos(prefetchSize: 0, prefetchCount: 1, global: false);
```

```
            var consumer = new EventingBasicConsumer(_channel);

            Console.WriteLine(" [*] Consumer Queue Service waiting for messages.");

            consumer.Received += (m,e) => _ = ReceivedCallBack(m,e);

            _channel.BasicConsume(
               queue: _queueName,
               autoAck: false,
               consumer: consumer
            );
         }
         catch (Exception e)
         {
            // _logger.LogError(e.Message, e);
            Console.WriteLine($"{e.Message} | {e.StackTrace}");
         }
         return Task.CompletedTask;
      }

      private async Task ReceivedCallBack(object model, BasicDeliverEventArgs ea)
      {
         try
         {
            var body = ea.Body.ToArray();
            var modelBody =
JsonConvert.DeserializeObject<RabbitBodyModel>(Encoding.UTF8.GetString(body));
            Console.WriteLine(" [x] Consumer Queue Service received message {0}",
modelBody.ImageName);
```

```
            MessageSender.SendMessage(modelBody);

            if (modelBody.TaskId != 0)
                _channel.BasicAck(
                ea.DeliveryTag,
                false);
        }
        catch (Exception e)
        {
            Console.WriteLine($"{e.Message} | {e.StackTrace}");
            // _logger.LogError(e.Message, e);
        }
    }
}
```