Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ РОБОТИ КОМУНАЛЬНИХ СЛУЖБ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-20-1

Губін Д.І.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Безсонов О.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Губіну Дмитру Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка застосунку для організації роботи комунальних служб

затверджена наказом університету від 20 травня 2024 року № 464 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 3 червня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, дані інтернет-мережі, бібліотека з відкритим кодом React.

4. Перелік питань, що потрібно опрацювати в роботі

1. Аналітичний огляд мов програмування та бібліотек для розробки застосунку організації роботи комунальних служб.
2. Моделювання застосунку організації роботи комунальних служб.
3. Програмна реалізація застосунку.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми організації роботи комунальних служб, постановка задачі, тестові зображення.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.04.2024	
2	Аналіз завдання, підбір літератури	11.04.23-17.04.24	
3	Аналіз літератури з досліджуваної проблеми	18.04.23-20.04.24	
4	Аналіз технічних засобів	21.04.23-30.04.24	
5	Моделювання застосунку	01.05.23-14.05.24	
6	Програмна реалізація	15.05.23-23.05.24	
7	Оформлення пояснювальної записки	24.05.23-26.05.24	
8	Перевірка на плагіат	27.05.24	
9	Рецензування	28.05.24	
10	Підготовка презентації та доповіді	29.05.23-30.05.24	
11	Занесення роботи в електронний архів	02.06.24	
12	Попередній захист кваліфікаційної роботи	12.06.24	

Дата видачі завдання 08 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Безсонов О.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 56 с., 14 рис., 24 джерел.

ВЕБСАЙТ, ОТОЧЕННЯ DOCKER ТА DOCKER COMPOSE, NEXT JS, CSS, МОВА ПРОГРАМУВАННЯ C#, REACT, JAVASCRIPT ОТОЧЕННЯ NOTE.JS, РЕЛЯЦІЙНА БАЗА ДАНИХ POSTGRESQL

Об’єктом роботи є створення вебсайту для управління комунальними послугами та проблемами. Буде використовуватися Docker, Docker Compose, JavaScript, Node.js, Express, React, PostgreSQL для розробки системи.

Метою роботи є створення надійної та ефективної системи, що дозволяє користувачам легко та швидко повідомляти про проблеми, керувати ними та відслідковувати їх вирішення.

Користувач може додавати звіти про проблеми та переглядати свої звіти. У результаті роботи створено вебсайт для управління комунальними послугами з усіма базовими функціями.

WEBSITE, DOCKER AND DOCKER COMPOSE ENVIRONMENT, NEXT JS, CSS, PROGRAMMING LANGUAGE C#, JAVASCRIPT LIBRARY REACT, JAVASCRIPT ENVIRONMENT NODE.JS, RELATIONAL DATABASE POSTGRESQL

The object of the work is to create a website for managing communal services and issues. Docker, Docker Compose, JavaScript, Node.js, Express, React, PostgreSQL will be used to develop the system.

The aim of the work is to create a reliable and efficient system that allows users to easily and quickly report issues, manage them. Users can add reports about issues, view their reports, and receive notifications about the status of issue resolution. As a result of the work, a website for managing communal services with all basic functions has been created.

ЗМІСТ

Вступ 7
1 Аналіз предметної області та постановка задачі 10
1.1 Огляд останніх досліджень і публікацій 10
1.2 Вибір засобів реалізації 11
1.3 Загальний огляд мов програмування 12
1.4 Загальний огляд бібліотек та фреймворків 16
1.5 Постановка задачі 19
2 Обґрунтування вибраних методів 20
2.1 База данних 20
2.2 Методики фронтенд частини 26
2.3 Методики бекенд частини 29
3 Програмна реалізація 34
3.1 Структура бази даних 34
3.2 Реалізація бекенд частини 38
3.3 Реалізація фронтенд частини 43
3.4 Перспективи подальшої роботи 49
Висновки 52
Перелік джерел посилання 53

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ВЗ – вебзастосунок

HTML – HyperText Markup Language (мова розмітки гіпертексту)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

CRUD-операції (Create, Read, Update, Delete)

ВСТУП

Комунальні послуги відіграють важливу роль у забезпеченні чистоти і порядку в містах. Вони охоплюють широкий спектр діяльності, включаючи утримання вулиць, прибирання сміття, ремонт інфраструктури, водопостачання, освітлення, озеленення та багато інших аспектів, що забезпечують комфортне життя міських жителів. Для ефективного виконання своїх обов'язків комунальним службам необхідно оперативно отримувати інформацію про проблеми, що виникають у міському середовищі. Без швидкого і точного обміну інформацією комунальні служби можуть зіткнутися з численними труднощами, такими як затримки у вирішенні проблем, підвищення витрат на ремонт і обслуговування, а також зниження рівня задоволеності громадян.

Створення спеціалізованих електронних платформ для обміну такою інформацією сприяє більш швидкому і ефективному реагуванню на проблеми. Такі платформи дозволяють автоматизувати процес збору, обробки і передачі інформації, що значно зменшує час реакції на запити громадян. Вони також забезпечують прозорість процесів і підвищують рівень довіри між громадянами і комунальними службами. Впровадження таких платформ може значно покращити якість обслуговування, підвищити ефективність роботи комунальних служб і зменшити витрати на управління міською інфраструктурою.

Інтернет-платформи для комунальних послуг стають дедалі популярнішими в Україні. З розвитком цифрових технологій та зростанням кількості інтернет-користувачів, обсяг взаємодії між громадянами і комунальними службами через інтернет постійно збільшується. Завдяки зручності та доступності інтернет-платформ громадяни можуть легко і швидко повідомляти про проблеми, залишати відгуки та пропозиції, а також отримувати інформацію про статус вирішення їхніх запитів. Це сприяє

підвищенню якості життя в містах, покращенню стану міської інфраструктури та підвищенню рівня задоволеності громадян.

Вебсайт для допомоги комунальним службам у виявленні проблем дозволяє громадянам оперативно повідомляти про проблеми, що потребують уваги. Це може бути повідомлення про поламаний ліхтар, розбиту дорогу, витік води або будь-яку іншу проблему, яка потребує втручання комунальних служб. Такий вебсайт також може надавати додаткову інформацію про комунальні послуги, включаючи графіки робіт, контакти відповідальних осіб та новини про стан міської інфраструктури.

Відсутність таких платформ може бути стратегічним недоліком для комунальних служб, оскільки уповільнює процес виявлення і вирішення проблем. Без можливості швидкого обміну інформацією комунальні служби змушені покладатися на традиційні методи, такі як телефонні дзвінки або письмові запити, що значно збільшує час реакції та ускладнює управління ресурсами. Це може призвести до погіршення якості обслуговування, зниження рівня довіри громадян і збільшення витрат на управління міською інфраструктурою.

Швидкість передачі інформації великому колу людей: Вебсайти дозволяють миттєво поширювати інформацію серед великої кількості користувачів, що забезпечує оперативність і ефективність реагування на проблеми.

Поліпшення іміджу комунальних служб і підвищення довіри громадян: Прозорість і відкритість процесів, доступність інформації та оперативне реагування на запити громадян сприяють підвищенню рівня довіри до комунальних служб.

Наявність зворотного зв’язку з громадянами: Вебсайти надають можливість громадянам залишати відгуки та пропозиції, що допомагає комунальним службам покращувати якість своїх послуг і оперативно реагувати на потреби населення.

Ефективна реклама і залучення громадян до підтримання порядку в місті: Вебсайти можуть використовуватися для інформування громадян про важливі події, акції та ініціативи, спрямовані на покращення міського середовища. Це сприяє активній участі громадян у підтриманні чистоти і порядку в місті.

Таким чином, розробка та впровадження вебсайту для допомоги комунальним службам є важливим кроком до підвищення ефективності управління міською інфраструктурою.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд останніх досліджень і публікацій

Сайти, спрямовані на допомогу комунальним службам у виявленні та вирішенні міських проблем, стають дедалі актуальнішими. Прикладом такого ресурсу є сайт 1562.kharkivrada.gov.ua, який дозволяє подавати заявки на вирішення міських проблем. Проте, цей сайт не є достатньо популярним серед жителів Харкова через його складність та недостатню орієнтованість на простих громадян. Мій проект передбачає створення нового вебсайту, який буде більш зручним та ефективним для використання звичайними мешканцями міста.

Значна частина населення не знає про існування сайту 1562 або не довіряє міським властям через старі звички та складність системи подачі заявок. Без використання веб-платформ процес подачі заявок значно ускладнюється: утворюються великі черги, людям доводиться їхати далеко від дому, щоб подати заявку, що призводить до затримок в обробці заявок. Якщо людина не може відразу дістатися до місця подачі заявки, проблема може залишитися нерозв’язаною.

Розробка нового вебсайту для допомоги комунальним службам у виявленні проблем має на меті вирішити ці проблеми. Новий ресурс забезпечить швидку обробку заявок, можливість подати їх з будь-якого місця за наявності інтернету, автоматизацію для уникнення помилок та швидку сортування заявок по відповідним службам.

Основні функції сайту включатимуть завантаження фотографій проблем, присвоєння їм імені та опису, можливість вказати місце, де це сталося, та автоматичне зазначення часу подачі заявки. Це дозволить мешканцям Харкова оперативно повідомляти про проблеми, а комунальним

службам – швидко реагувати на запити і допомагати громадянам. Такий підхід сприятиме покращенню роботи міста та вирішенню його проблем.

Цільова аудиторія сайту – всі жителі Харкова, які зможуть подавати заявки для вирішення міських проблем, а також комунальні служби, які будуть оперативно реагувати на ці запити. Вебсайт допоможе створити краще розуміння про роботу міста та надасть можливість для швидкого вирішення міських проблем, таких як встановлення світлофорів або усунення неполадок.

Для бекенду проекту буде використано ASP.NET Core, C#, Visual Studio; для бази даних – PostgreSQL та Docker; для фронтенду – React, EF Core, Visual Studio Code. Вибір цих технологій обумовлений їхньою актуальністю, підтримкою розробників та ефективністю.

Метою роботи є створення надійного та ефективного вебсайту, який допоможе вирішувати проблеми громадян та покращувати місто, роблячи його більш безпечним та комфортним для життя.

1.2 Вибір засобів реалізації

Зараз у веброзробника стоїть безліч завдань, починаючи від створення інтерактивних сайтів для розваг до серйозних бізнес-проєктів, які потребують підвищеної надійності та захисту від несанкціонованого доступу. Для їх виконання потрібні правильно підібрані інструменти мови програмування, фреймворки або системи управління контентом, які стають все більш актуальними.

На сьогодні існує багато мов програмування, і перевага кожної з них залежить від конкретного контексту завдання. Вибір мови визначається рівнем знань програміста та їх достатністю для реалізації проєкту.

1.3 Загальний огляд мов програмування

Мова програмування – це формалізована система символів, призначена для створення комп'ютерних програм. Кожна мова має свій унікальний набір команд, синтаксис та семантику, які визначають, як виглядатиме програма та які дії виконуватиме комп'ютер. На наведеному графіку показана популярність мов програмування для розробки вебсайтів (рис. 1.1).

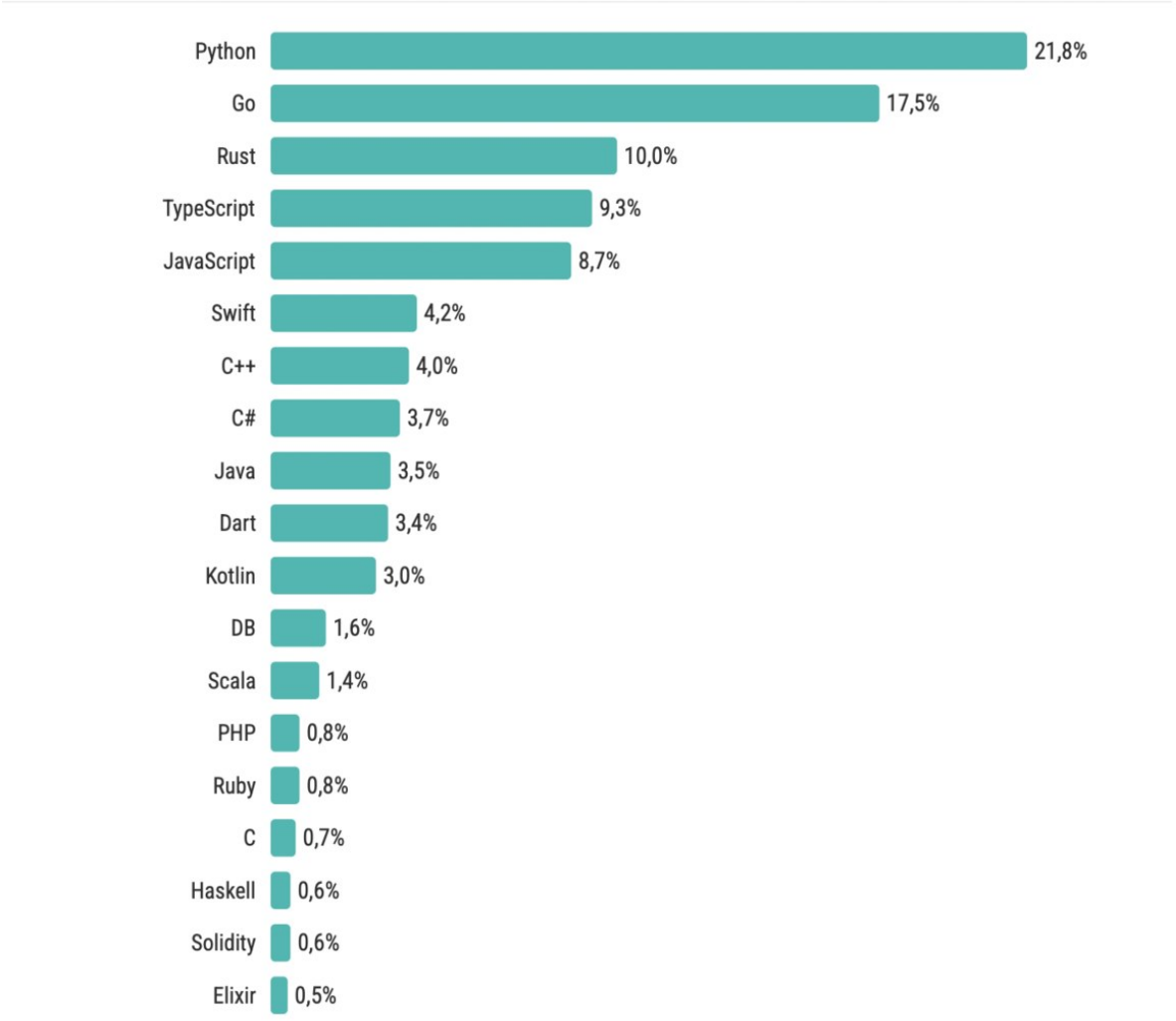

Рисунок 1.1 – Графік популярності мов програмування для веброзробки [1]

Можна розбити всі мови вебпрограмування на дві категорії: клієнтські та серверні мови. Мова клієнта використовується для створення програм, що

працюють на клієнтській стороні, тобто у браузері, а серверна мова для програм, що працюють на сервері.

Єдина мова програмування для клієнтської частини вебсторінок – це JavaScript, яка використовується разом з бібліотекою React. JavaScript забезпечує інтерактивність і динамічність вебсайтів. Основною особливістю JavaScript є те, що елементи вебсередовища можуть змінюватись без перезавантаження сторінки, що дозволяє змінювати колір фону, зображення, створювати нові вікна або відображати повідомлення.

JavaScript і React були обрані через такі переваги як те що JavaScript є універсальною мовою програмування, яка використовується для створення інтерактивних елементів на вебсторінках. Вона підтримується всіма сучасними браузерами, що робить її невід'ємною частиною веброзробки. Використання JavaScript дозволяє реалізувати асинхронне програмування за допомогою технологій AJAX, що забезпечує динамічне оновлення контенту без перезавантаження сторінки.

React є однією з найпопулярніших бібліотек для побудови інтерфейсів користувача. Вона була створена компанією Facebook і має велику спільноту розробників. React використовує компонентний підхід, що дозволяє розбивати інтерфейс на незалежні, повторно використовувані компоненти. Це спрощує розробку та підтримку коду, а також підвищує продуктивність завдяки використанню віртуального DOM.

Крім того, для створення інтерфейсу використовуються мови розмітки HTML і CSS. HTML є мовою розмітки гіпертексту для створення структури вебсайтів, а CSS – це формальна мова, яка визначає зовнішній вигляд документів, написаних на HTML. CSS використовується для встановлення шрифтів та макетів на сторінці, а також інших важливих аспектів вигляду сторінок.

ASP.NET Core, яка базується на C# і є потужним інструментом для розробки вебзастосунків. Вибір ASP.NET Core обумовлений її високою продуктивністю, безпекою та масштабованістю. Ця мова дозволяє створювати

надійні серверні програми, що можуть обробляти великі обсяги даних і забезпечувати високу продуктивність.

PostgreSQL використовується як основна база даних для зберігання і обробки даних. Це потужна реляційна база даних, яка забезпечує високу продуктивність і надійність. PostgreSQL підтримує складні запити і гарантує цілісність даних, що є критично важливим для нашого проєкту.

Docker і Docker Compose використовуються для контейнеризації застосунку, що дозволяє ізолювати середовище застосунку і забезпечити його стабільну роботу незалежно від конфігурації системи. Використання Docker спрощує розгортання і масштабування застосунок.

C# є основною мовою програмування для розробки серверної частини нашого застосунка. Ця мова забезпечує високу продуктивність і безпеку, а також підтримує різні парадигми програмування, що робить її потужним і гнучким інструментом.

Visual Studio є основним середовищем розробки, яке забезпечує потужні інструменти для написання, налагодження та тестування коду. Це інтегроване середовище розробки підтримує інтеграцію з багатьма іншими інструментами і технологіями, що спрощує процес розробки.

Додаткові інструменти та фреймворки які були обрані для написання проекту. Для роботи з базою даних використовується Entity Framework. Це об’єктно-реляційний маппер (ORM), який забезпечує зручну роботу з базами даних на основі C#. Entity Framework дозволяє розробникам працювати з базами даних, використовуючи об’єктно-орієнтований підхід, що значно спрощує розробку і підтримку коду.

Переваги ASP.NET Core, PostgreSQL та Docker:

– ASP.NET Core забезпечує високу продуктивність завдяки своїй легковаговій та модульній архітектурі. Вона підтримує розробку кросплатформених застосунків, що дозволяє розгортати вебзастосунки на різних операційних системах, включаючи Windows, Linux та macOS. Безпека

також є ключовим аспектом ASP.NET Core, зокрема підтримка захисту від CSRF, XSS та інших атак;

– PostgreSQL є потужною та надійною базою даних, яка забезпечує високий рівень продуктивності та масштабованості. Вона підтримує транзакції ACID, що гарантує цілісність даних, а також має потужний механізм розширень, що дозволяє адаптувати базу даних до специфічних вимог проекту;

– Docker дозволяє створювати ізольовані середовища для розробки та розгортання застосунків, що забезпечує стабільність та передбачуваність роботи програм. Використання контейнерів спрощує масштабування застосунків та управління залежностями, а також дозволяє швидко розгортати нові версії програмного забезпечення.

Таким чином, вибрані технології забезпечують ефективну, надійну та масштабовану реалізацію нашого вебсайту для допомоги комунальним службам у виявленні та вирішенні міських проблем. Використання ASP.NET Core, React, PostgreSQL, Docker та інших інструментів дозволяє створити високоякісний вебзастосунок, що відповідає сучасним вимогам та стандартам веброзробки.

Проте, важливо також враховувати інші аспекти, такі як забезпечення безпеки даних, зручність користування, адаптивність інтерфейсу та ефективне тестування, щоб створений продукт був максимально корисним та надійним.

Безпека даних є ключовим аспектом будь-якого вебзастосунку. Для цього використовуються такі заходи, як аутентифікація та авторизація користувачів, шифрування даних, захист від SQL-ін’єкцій, а також регулярне оновлення програмного забезпечення для запобігання вразливостям.

Зручний та адаптивний інтерфейс забезпечує кращий користувацький досвід. Вебсайт має бути інтуїтивно зрозумілим і доступним для всіх користувачів, включаючи людей з обмеженими можливостями. Використання сучасних технологій, таких як React, дозволяє створити інтерфейс, який легко адаптується під різні розміри екранів та пристрої.

Тестування є невід’ємною частиною процесу розробки. Воно включає модульне тестування окремих компонентів, інтеграційне тестування для перевірки взаємодії між ними та тестування продуктивності для оцінки здатності системи витримувати високі навантаження.

Після впровадження важливо забезпечити постійний моніторинг системи для виявлення та виправлення можливих проблем. Регулярне оновлення та підтримка системи дозволяють зберегти її продуктивність та безпеку на високому рівні.

Наявність ефективної системи підтримки користувачів є важливим аспектом успішного функціонування вебсайту. Це включає створення довідкових матеріалів, надання можливості зворотного зв’язку та оперативне вирішення питань та проблем користувачів.

Ці додаткові аспекти забезпечують створення комплексного рішення, яке не тільки відповідає технічним вимогам, але й є зручним та безпечним для користувачів

1.4 Загальний огляд бібліотек та фреймворків

Кожен фреймворк має свій унікальний набір функцій та інтерфейсів, що допомагають зменшити кількість коду, необхідного для розробки програм. На графіку показана популярність фреймворків для розробки вебсайтів, що дозволяє розробникам вибрати найбільш підходящий для їхнього проєкту. Фреймворки можуть бути створені для різних мов програмування, таких як Python, Java, Ruby, PHP та інших, що дозволяє розробникам працювати з різними технологіями. Це робить фреймворки дуже популярними серед розробників вебзастосунків та вебсайтів. Однією з головних переваг використання фреймворків є прискорення розробки. Завдяки готовим структурам та інтерфейсам, розробникам не потрібно писати код з нуля. Крім того, фреймворки зазвичай відрізняються високою надійністю та безпекою,

оскільки вони підтримуються та оновлюються розробниками. Використання фреймворків допомагає розробникам зосередитися на вирішенні конкретних завдань, замість витрачання часу на написання загальних функцій та алгоритмів з нуля. На відміну від мов програмування, бібліотеки є наборами готових функцій та інструментів, які можна використовувати у програмуванні для скорочення часу розробки та спрощення процесу програмування. Бібліотеки надають можливість розробникам використовувати готові рішення та функції для вирішення певних завдань, замість написання коду з нуля. Сьогодні існує багато різних бібліотек для різних мов програмування, що дозволяє розробникам створювати складні та потужні програми з меншими витратами часу та зусиль.

Наведена нижче діаграма показує популярність бібліотечних мов і фреймворків для розробки клієнтської частини вебсайту (рис. 1.2).

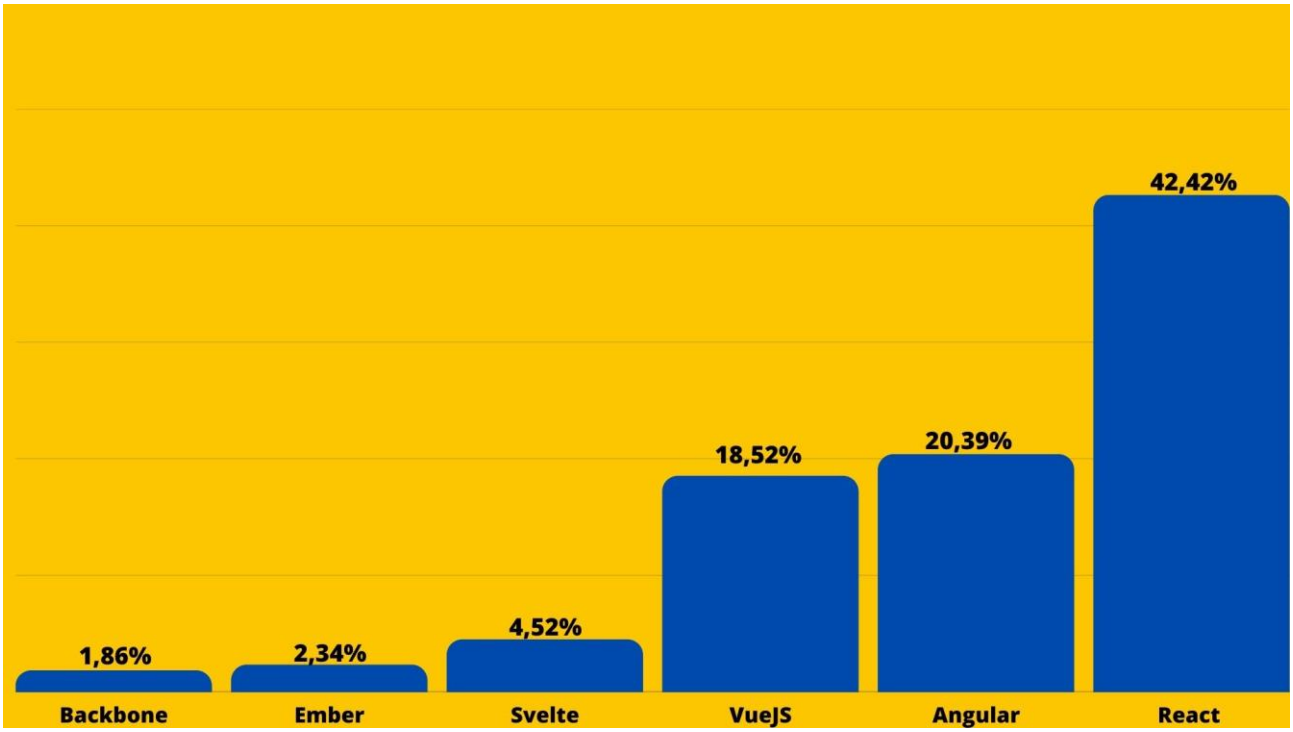

Рисунок 1.2 – Графік популярності бібліотек та фреймворків для розробки клієнтської частини вебсайтів [2]

React є однією з найбільш потужних та популярних бібліотек для розробки вебінтерфейсів у сфері IT. Її використання дозволяє розробникам

створювати динамічні та взаємодіючі інтерфейси швидко та ефективно. За допомогою React можна створювати складні компоненти з низкою залежністю, що дозволяє розробникам легко перевикористовувати код та підтримувати його. React також має велику спільноту, яка активно працює над розвитком бібліотеки та надає багато корисних «інструментів для розробників».

React – це бібліотека JavaScript для створення користувацьких інтерфейсів, розроблена компанією Facebook. Вона дозволяє розробникам створювати багаторазові компоненти, які ефективно оновлюються та рендеряться, забезпечуючи високу продуктивність застосунків. React використовує концепцію віртуального DOM, яка мінімізує операції з реальним DOM і таким чином підвищує швидкість роботи застосунків. React може бути використаний як основа для мобільних застосунків за допомогою React Native. Бібліотека є дуже гнучкою і може бути інтегрована з іншими фреймворками або використовуватись окремо.

React дозволяє створювати багаторазові компоненти, які можна легко комбінувати і використовувати в різних частинах програми. Це сприяє організації коду і полегшує його супровід та тестування. Використання віртуального DOM дозволяє React ефективно оновлювати лише ті частини інтерфейсу, які зазнали змін. Це значно покращує продуктивність застосунку, оскільки зменшує кількість операцій з реальним DOM. React забезпечує передбачуване управління станом завдяки однонаправленому потоку даних, що спрощує відстеження змін у застосунку і полегшує відлагодження. Можна легко інтегрувати з іншими бібліотеками або фреймворками. Це дозволяє розробникам використовувати React у поєднанні з іншими інструментами та технологіями для створення складних застосунків. Має велику і активну спільноту розробників, що забезпечує наявність численних ресурсів, бібліотек та інструментів, які можуть допомогти у вирішенні різноманітних задач.

Завдяки цим перевагам, React стає оптимальним вибором для розробки сучасних вебзастосунків, які потребують високу продуктивність, масштабованість та зручність у розробці.

1.5 Постановка задачі

Основою функціональності вебсайту для допомоги комунальним службам є взаємопов’язані компоненти, які ретельно розроблені та розподілені на окремі функціональні блоки. Сайт надає користувачам можливість повідомляти про різні міські проблеми, завантажувати фотографії, додавати описи та координати, що забезпечує швидке реагування комунальних служб. Використовуватимуться сучасні технології, щоб забезпечити надійну та ефективну обробку заявок.

Об’єктом роботи є створення вебсайту для допомоги комунальним службам у виявленні та вирішенні міських проблем. Для розробки системи буде використовуватися Docker, Docker Compose, JavaScript, Node.js, Express, React, PostgreSQL та C# з використанням ASP.NET Core.

Метою роботи є створення надійного та ефективного ресурсу, що дозволяє мешканцям легко та швидко повідомляти про проблеми, які потребують уваги комунальних служб.

Для досягнення мети необхідно вирішити такі завдання:

– завантаження фотографій проблем;

– додавання опису та координат місця проблеми;

– автоматичне зазначення часу подачі заявки;

– каталог заявок з можливістю фільтрації за статусом та типом проблеми;

– інструменти для моніторингу та аналізу вирішення проблем.

2 ОБҐРУНТУВАННЯ ВИБРАНИХ МЕТОДІВ

2.1 База даних

Бази даних відіграють критичну роль у сучасних інформаційних системах, забезпечуючи зберігання, організацію і управління великими обсягами даних. У даному розділі буде розглянуто, що таке бази даних, їх використання, плюси і мінуси, структура бази даних, що використовується в проекті, і процес запису даних у базу даних.

Бази даних використовуються для зберігання і управління даними в багатьох сферах, включаючи бізнес, освіту, медицину, уряд і розваги. Вони дозволяють організаціям ефективно зберігати, обробляти та аналізувати інформацію. Наприклад, бази даних використовуються для зберігання інформації про клієнтів, облікових записів, продуктів, фінансових транзакцій, медичних записів і багато іншого.

З переваг баз даних я можу навести такі приклади:

– організація даних: Бази даних дозволяють організовувати дані в структурованому вигляді, що полегшує їх зберігання, пошук і обробку;

– цілісність даних: СУБД забезпечують механізми для підтримки цілісності даних, включаючи контроль доступу, забезпечення унікальності даних і підтримку зв’язків між таблицями;

– масштабованість: Бази даних можуть масштабуватися для підтримки великих обсягів даних і одночасної роботи багатьох користувачів;

– безпека: СУБД забезпечують високий рівень безпеки даних, включаючи аутентифікацію, авторизацію і шифрування;

– відновлення даних: СУБД забезпечують механізми для відновлення даних у випадку збою, включаючи резервне копіювання і відновлення після аварії.

Але є і таки недоліки:

– Складність налаштування і управління: Налаштування і управління базою даних можуть вимагати значних зусиль і технічних знань;

– Вартість: Впровадження і підтримка СУБД можуть бути дорогими, особливо для великих організацій;

– Проблеми продуктивності: Зі збільшенням обсягів даних і кількості користувачів можуть виникати проблеми з продуктивністю, що вимагає оптимізації і налаштування.

У проекті використовується реляційна база даних PostgreSQL. Вибір PostgreSQL обумовлений її високою продуктивністю, надійністю, а також широкими можливостями для масштабування та забезпечення цілісності даних. PostgreSQL підтримує транзакції ACID (Atomicity, Consistency, Isolation, Durability), що гарантує надійне зберігання та обробку даних.

Архітектура бази даних побудована на основі моделі клієнт-сервер. Клієнтська частина взаємодіє з базою даних через API, розроблений на основі ASP.NET Core. Це забезпечує високу продуктивність і безпеку передачі даних між клієнтами і сервером. База даних розгорнута в контейнері Docker, що забезпечує легкість розгортання і масштабування.

Модель даних проекту складається з однієї таблиці, яка відображають основні сутності системи.

Це таблиця Файли (Files)**:** таблиця містить інформацію про завантажені файли, включаючи їх опис, тип та шлях зберігання.

Приклад цієї таблиці наведено в наступному рисунку.

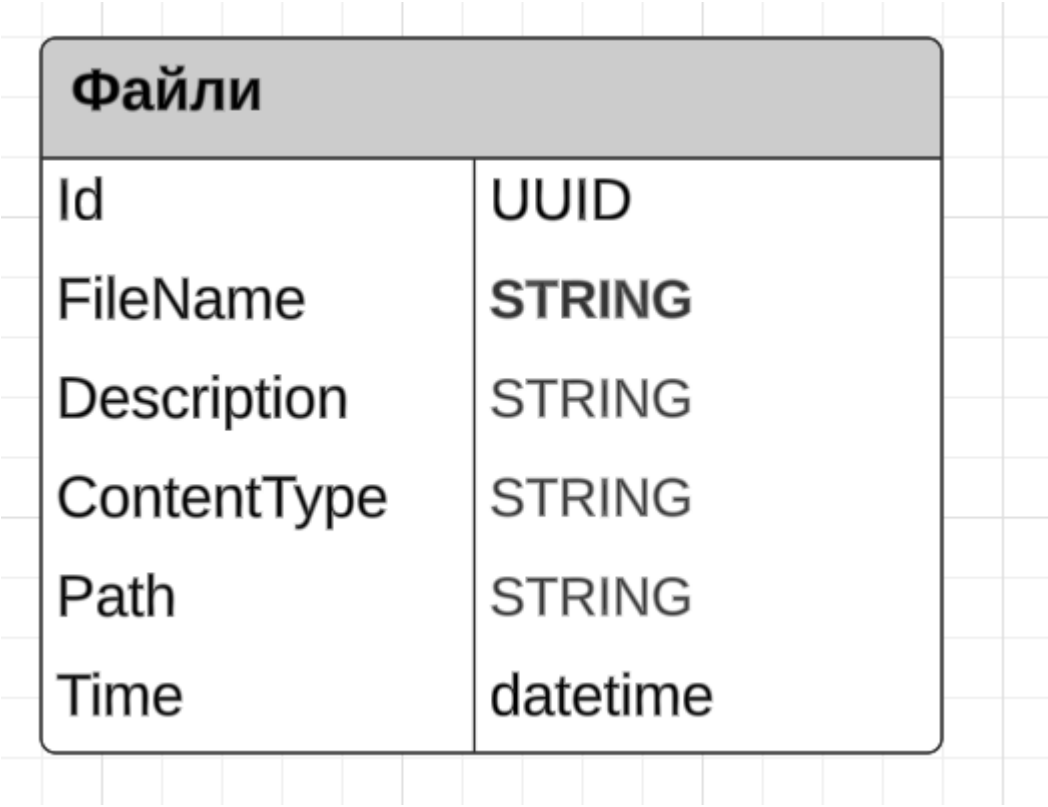

Рисунок 2.1 – Таблиця Files, яка використана в проекті

Для налаштування і створення бази даних використовувались міграції Entity Framework (EF Core). EF Core дозволяє автоматизувати процес створення і оновлення структури бази даних за допомогою кодогенерації. Основні етапи включають:

– Ініціалізація проекту: створення нового проекту і додавання необхідних залежностей, включаючи EF Core та PostgreSQL;

– Створення моделей даних: визначення моделей даних у коді, які будуть відображати структуру таблиць у базі даних. Моделі даних визначаються за допомогою класів C#, які відображають структуру таблиць у базі даних;

– Налаштування контексту даних: конфігурація контексту даних, який встановлює зв’язок між моделями даних і базою даних. Контекст даних є ключовим елементом у EF Core, оскільки він дозволяє взаємодіяти з базою даних через моделі даних;

– Виконання міграцій: створення і застосування міграцій для оновлення структури бази даних у відповідності до моделей даних. Міграції дозволяють автоматично оновлювати структуру бази даних при зміні моделей даних.

Першим кроком є створення нового проекту і додавання необхідних залежностей. Це дозволяє швидко налаштувати проект і розпочати розробку.

Після ініціалізації проекту необхідно визначити моделі даних, які будуть відображати структуру таблиць у базі даних. Моделі даних визначаються за допомогою класів, які відображають структуру таблиць у базі даних.

Після створення моделей даних необхідно налаштувати контекст даних, який встановлює зв’язок між моделями даних і базою даних. Контекст даних є ключовим елементом у EF Core, оскільки він дозволяє взаємодіяти з базою даних через моделі даних.

Після налаштування контексту даних необхідно створити і застосувати міграції для оновлення структури бази даних у відповідності до моделей даних. Це дозволяє автоматично оновлювати структуру бази даних при зміні моделей даних.

Взаємодія з базою даних реалізована через сервісний шар, який використовує репозиторій для роботи з даними. Це дозволяє абстрагуватися від конкретної реалізації бази даних і забезпечує легкість у підтримці та розширенні функціональності системи.

Основні функції взаємодії включають:

– створення записів: додавання нових звітів про проблеми та завантажених файлів;

– читання даних: отримання даних про звіти і файли для відображення у веб-інтерфейсі;

– оновлення записів: внесення змін у існуючі записи, такі як статус обробки звітів;

– видалення записів: видалення застарілих або некоректних записів з бази даних.

Запис у базу даних є важливим етапом у процесі взаємодії з користувачем. Наприклад, користувач може зареєструвати нову проблему через форму на вебсайті, яка буде збережена у базі даних. Це дозволяє комунальним службам швидко отримувати інформацію про проблеми і реагувати на них.

Читання даних з бази даних є ключовим елементом для відображення інформації користувачам. Наприклад, комунальні служби можуть переглядати всі зареєстровані проблеми і деталі щодо них, включаючи завантажені файли. Це дозволяє їм ефективно управляти процесом вирішення проблем.

Оновлення записів у базі даних дозволяє комунальним службам вносити зміни у статус обробки проблем у реальному часі. Це дозволяє користувачам отримувати актуальну інформацію про стан вирішення їхніх проблем.

Видалення записів з бази даних дозволяє підтримувати актуальність і чистоту даних. Наприклад, користувач може видалити помилково зареєстровану проблему або комунальні служби можуть видалити записи, які більше не є актуальними.

Для забезпечення цілісності та безпеки даних у базі даних використовуються наступні методи:

– обмеження цілісності: використання первинних та зовнішніх ключів для забезпечення правильності зв’язків між таблицями;

– транзакції: використання транзакцій для гарантування цілісності даних при виконанні складних операцій;

– резервне копіювання: регулярне створення резервних копій бази даних для захисту від втрати даних;

– Цілісність даних забезпечується за допомогою первинних та зовнішніх ключів, які гарантують правильність зв’язків.

Транзакції забезпечують цілісність даних при виконанні складних операцій, гарантуючи, що всі зміни даних будуть виконані або скасовані як єдине ціле. Це дозволяє уникнути неповних або некоректних змін у базі даних.

Резервне копіювання забезпечує захист від втрати даних, дозволяючи відновити базу даних у випадку збою. Регулярне створення резервних копій є важливим елементом забезпечення безпеки даних.

Для оптимізації продуктивності бази даних використовуються наступні методи:

– Індексація: створення індексів на часто використовуваних полях для прискорення пошуку і фільтрації даних;

–Кешування: використання кешування для зберігання часто запитуваних даних у пам’яті, що зменшує навантаження на базу даних;

– Оптимізація запитів: аналіз і оптимізація SQL-запитів для зменшення часу їх виконання.

Індексація дозволяє значно підвищити швидкість виконання запитів до бази даних. Наприклад, створення індексу на полі «Name» у таблиці «Звіти» дозволяє швидко знаходити звіти за назвою.

Кешування дозволяє зменшити навантаження на базу даних, зберігаючи часто запитувані дані у пам’яті. Це дозволяє зменшити кількість звернень до бази даних і підвищити продуктивність системи.

Оптимізація запитів включає аналіз і оптимізацію SQL-запитів для зменшення часу їх виконання. Це включає використання найефективніших методів запитів і уникнення надмірно складних запитів, які можуть сповільнювати роботу системи.

У даному розділі було розглянуто базу даних, використану у проекті для розробки вебсайту, призначеного для допомоги комунальним службам. Вибір PostgreSQL обумовлений її високою продуктивністю, надійністю і широкими можливостями для масштабування. Описано архітектуру бази даних, модель даних, процес створення і налаштування бази даних, інтеграцію з веб-застосунком, методи забезпечення цілісності та безпеки даних, а також оптимізацію продуктивності. Завдяки використаним підходам і методам, розроблена система є надійною, ефективною і готовою до розширення у майбутньому.

2.2 Методики фронтенд частини

Фронтенд частина мого проєкту була реалізована з використанням сучасних технологій та бібліотек, що забезпечують високу продуктивність і зручність у розробці та використанні застосунку. Основною технологією для створення користувацького інтерфейсу став React, який надає компонентний підхід до розробки. Це дозволяє розбивати інтерфейс на незалежні, повторно використовувані компоненти, що суттєво спрощує процес розробки та підтримки коду. Кожен компонент інкапсулює свою логіку та стилі, що забезпечує модульність і легкість у тестуванні.

React, розроблений компанією Facebook, став стандартом де-факто для розробки користувацьких інтерфейсів завдяки своїй гнучкості, високій продуктивності та підтримці великої спільноти розробників. React дозволяє створювати динамічні веб-застосунки, що реагують на зміни даних у реальному часі, завдяки концепції віртуального DOM. Віртуальний DOM дозволяє ефективно оновлювати лише ті частини інтерфейсу, які змінилися, що значно покращує продуктивність застосунку.

Основні компоненти у нашому проєкті включають:

– компонент «Список проблем»: Відповідає за відображення всіх проблем, про які повідомили користувачі. Він отримує дані з сервісного шару і відображає їх у вигляді списку;

– компонент «Форма додавання проблеми»: Дозволяє користувачам додавати нові проблеми. Включає поля вводу для опису проблеми, завантаження зображень і валідацію форми для перевірки коректності введених даних;

– кожен компонент взаємодіє з сервісами, які інкапсулюють логіку роботи з API. Це дозволяє відокремити логіку отримання даних від компонентів і зробити код більш чистим та повторно використовуваним.

На наступному рисунку зображено приклад роботи додавання проблеми

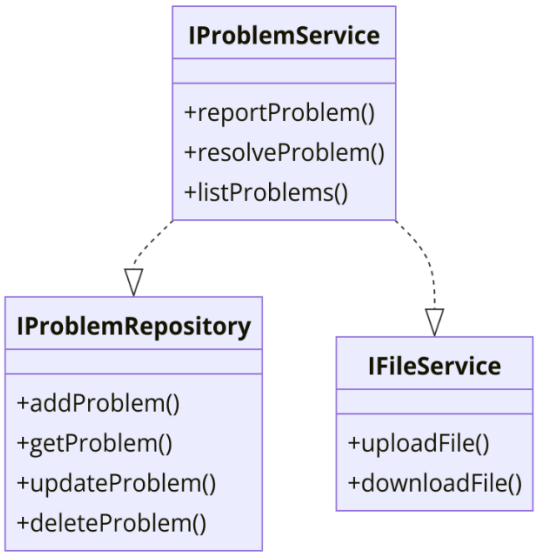

Рисунок 2.2 – Діграма-приклад реалізації додавання проблеми

Для створення стилів та інтерфейсних компонентів було використано бібліотеку Ant Design. Ant Design надає набір високоякісних React-компонентів та стилів, що забезпечують професійний вигляд та зручність використання застосунку. Використання Ant Design дозволило значно скоротити час розробки, оскільки багато компонентів вже були готові до використання і потребували лише мінімальної настройки.

Проєкт організований за компонентним принципом. Основні розділи включають компоненти, сервіси, сторінки та стилі. Компоненти є основними будівельними блоками інтерфейсу. Наприклад, компонент «Список проблем» відображає всі проблеми, про які повідомили користувачі. Інший важливий компонент – «Форма додавання проблеми». Ця форма дозволяє користувачам повідомляти про нові проблеми, включає поля вводу для опису проблеми та завантаження зображень, а також валідацію форми для перевірки коректності введених даних.

Сторінки є високорівневими компонентами, які об’єднують кілька дрібніших компонентів для відображення цілих екранів застосунку. Наприклад, головна сторінка включає в себе «Список проблем» та «Форму додавання проблеми». Це дозволяє розробляти та підтримувати застосунок у

вигляді окремих модулів, що спрощує тестування та подальше розширення функціональності.

Обґрунтування вибору технологій базується на їх перевагах та застосовності до завдань проєкту. React був обраний за його компонентний підхід та віртуальний DOM, які забезпечують високу продуктивність та спрощують розробку. JavaScript є основною мовою програмування завдяки своїй гнучкості та потужним можливостям. HTML та CSS забезпечують чітку структуру та стилізацію інтерфейсу, що робить код підтримуваним та зрозумілим.

Використання цих технологій дозволяє створити надійний та ефективний фронтенд-застосунок, який легко підтримувати та розвивати. HTML та CSS дозволяють створити чистий та підтримуваний код, який легко модифікувати та розширювати. Завдяки цим рішенням, фронтенд-застосунок відповідає сучасним стандартам веб-розробки та забезпечує високий рівень взаємодії з користувачами.

Окрім того, важливим аспектом у розробці фронтенду було забезпечення адаптивності та кросбраузерної сумісності застосунку. Це було досягнуто завдяки використанню сучасних веб-технологій та дотриманню стандартів веб-розробки. Наприклад, CSS Flexbox та Grid Layout використовувалися для створення гнучких та адаптивних макетів, що забезпечують коректне відображення інтерфейсу на різних пристроях та екранах. Також було проведено тестування застосунку у різних браузерах для забезпечення сумісності та коректної роботи у всіх популярних середовищах.

Ще одним важливим аспектом стало забезпечення високої продуктивності фронтенд застосунку. Було використано методи оптимізації завантаження сторінок, такі як lazy loading для завантаження компонентів за потребою, мініфікація та обфускація JavaScript-коду, а також кешування статичних ресурсів. Це дозволило зменшити час завантаження сторінок та покращити загальний користувацький досвід.

Lazy loading дозволяє завантажувати компоненти лише тоді, коли вони потрібні. Це значно зменшує початковий обсяг даних, що передаються клієнту, і прискорює час завантаження сторінки. Наприклад, компоненти, які не є критичними для першого відображення сторінки, можуть бути завантажені асинхронно після початкового рендерингу.

Мініфікація JavaScript-коду зменшує його розмір, видаляючи пробіли, коментарі та скорочуючи імена змінних. Обфускація додатково ускладнює читання коду, що захищає його від несанкціонованого копіювання та аналізу. Ці методи разом допомагають зменшити час завантаження і підвищити безпеку застосунку.

Кешування статичних ресурсів, таких як CSS, JavaScript і зображення, дозволяє браузеру зберігати ці файли локально і використовувати їх повторно при наступних завантаженнях сторінки. Це значно зменшує кількість запитів до сервера і підвищує швидкість завантаження сторінок.

У результаті використання всіх цих технологій та методів, фронтенд-застосунок став надійним, продуктивним та зручним у використанні. Він забезпечує високий рівень взаємодії з користувачами, легкість у підтримці та розширенні, а також відповідність сучасним стандартам веб-розробки та безпеки.

2.3 Методики бекенд частини

Бекенд частина мого проєкту була реалізована з використанням сучасних технологій та бібліотек, що забезпечують високу продуктивність, надійність і зручність у розробці та підтримці застосунка. Основними технологіями, що використовуються в моєму проєкті, є .NET Core, Entity Framework Core та PostgreSQL. Вибір цих технологій був зумовлений їхніми потужними можливостями, гнучкістю та широкою підтримкою з боку спільноти розробників.

Вибір технологій:

−.NET Core був обраний як основний фреймворк для розробки серверної частини завдяки його кросплатформенності, високій продуктивності та активній підтримці. Цей фреймворк дозволяє створювати застосунки, які можуть працювати на різних операційних системах, таких як Windows, Linux і macOS. Це робить його ідеальним вибором для сучасних веб- застосунків, які повинні бути доступні на різних платформах;

− Entity Framework Core був використаний як ORM (Object-Relational Mapping) для взаємодії з базою даних. Цей інструмент надає зручні механізми для роботи з даними, дозволяючи розробникам використовувати об’єкти .NET для взаємодії з базою даних. Entity Framework Core забезпечує високу продуктивність і гнучкість, дозволяючи легко керувати змінами в структурі даних і виконувати міграції;

− PostgreSQL був обраний як система управління базами даних завдяки своїй високій продуктивності, надійності та підтримці розширених можливостей, таких як зберігання JSON-даних, повнотекстовий пошук та реплікація. PostgreSQL є вільно розповсюджуваним програмним забезпеченням з відкритим вихідним кодом, що робить його економічно вигідним рішенням для розробки.

Бекенд мого проєкту організований за принципом багатошарової архітектури, що дозволяє розділити логіку застосунку на кілька шарів і поліпшити модульність та тестованість коду. Основні шари застосунка включають:

− шар даних: містить сутності даних і контекст бази даних, які використовуються для взаємодії з базою даних.

− сервісний шар: містить бізнес-логіку застосунку і забезпечує виконання основних операцій з даними.

− шар контролерів: реалізує API-інтерфейс для взаємодії з фронтенд-частиною застосунка і обробки вхідних HTTP-запитів.

До основних компонентів належать:

− контекст бази даних (ProblemsDbContext). Цей клас використовується для взаємодії з базою даних і визначення наборів даних (DbSet), які представляють таблиці в базі даних. Контекст бази даних конфігурується за допомогою Entity Framework Core і включає визначення сутностей, таких як ProblemEntity і FileEntity. Контекст бази даних відповідає за підключення до бази даних, конфігурацію зв’язків між сутностями і забезпечення збереження даних;

– сервісний шар містить бізнес-логіку застосунка. Наприклад, сервіс ProblemService надає методи для створення, читання, оновлення та видалення проблем, а також взаємодіє з репозиторіями для виконання операцій з базою даних. Використання сервісного шару дозволяє ізолювати бізнес-логіку від контролерів і забезпечити чистий код, який легко тестувати та підтримувати.

Основні методи, що реалізовані в сервісах, включають:

– створення нової проблеми: метод, який приймає дані про проблему, зберігає їх у базі даних і повертає результат операції;

– отримання списку проблем: метод, який витягує з бази даних усі проблеми або проблеми, що відповідають певним критеріям;

– оновлення проблеми: метод, який приймає оновлені дані про проблему і зберігає їх у базі даних;

– видалення проблеми: метод, який видаляє проблему з бази даних за заданим ідентифікатором.

Контролери обробляють вхідні HTTP-запити і повертають відповіді клієнтам. Наприклад, контролер ProblemController надає кінцеві точки API для управління проблемами, такі як створення нової проблеми, отримання списку всіх проблем і оновлення існуючих проблем. Контролери забезпечують зв’язок між фронтендом і бізнес-логікою, приймаючи запити від клієнтів, викликаючи відповідні методи сервісів і повертаючи результати. Основні кінцеві точки включають:

– POST /problems: кінцева точка для створення нової проблеми яка викликає метод сервісу для збереження даних у базі.

– GET /problems: кінцева точка для отримання списку проблем яка викликає метод сервісу для отримання даних з бази.

– PUT /problems/{id}: кінцева точка для оновлення проблеми яка викликає метод сервісу для оновлення даних у базі.

– DELETE /problems/{id}: кінцева точка для видалення проблеми яка викликає метод сервісу для видалення даних з бази.

Вибір .NET Core як основного фреймворка був обумовлений його високою продуктивністю, кросплатформенністю і активною підтримкою з боку спільноти. Цей фреймворк забезпечує надійну основу для створення серверних застосунків і легко інтегрується з іншими інструментами і бібліотеками.

Entity Framework Core був обраний як ORM завдяки своїй гнучкості і зручності використання. Цей інструмент дозволяє розробникам працювати з базою даних, використовуючи об’єкти .NET, що спрощує процес розробки і управління даними. Entity Framework Core також надає потужні механізми для міграції даних, що полегшує управління змінами в структурі бази даних.

PostgreSQL був обраний як СУБД завдяки своїй високій продуктивності, надійності і підтримці розширених можливостей. Ця система управління базами даних забезпечує ефективне зберігання і обробку даних, а також надає можливості для масштабування і реплікації.

У процесі розробки бекенд частини застосунка були використані кілька важливих методик, що забезпечили високу якість і ефективність коду:

− багатошарова архітектура: розподіл логіки застосунку на кілька шарів забезпечує кращу організацію коду, модульність і легкість у тестуванні. Кожен шар виконує свою роль, що полегшує розуміння та підтримку коду;

− використання ORM Entity Framework Core: забезпечує зручність роботи з базою даних, дозволяючи розробникам використовувати об’єкти .NET для взаємодії з базою даних. Це знижує кількість помилок, пов’язаних з прямими запитами до бази даних, і спрощує управління даними;

− міграції даних: використання механізмів міграції Entity Framework Core дозволяє легко керувати змінами в структурі бази даних. Це забезпечує безперервний розвиток застосунку без втрати даних і мінімізації ризиків під час оновлень;

− репозиторій і сервіси: використання шаблонів «репозиторій» і «сервіс» дозволяє ізолювати доступ до даних і бізнес-логіку від контролерів. Це забезпечує чистий код, який легко тестувати і підтримувати, а також підвищує модульність застосунка;

− обробка виключень: впровадження механізмів обробки виключень забезпечує надійність і стабільність роботи застосунка. Це дозволяє вчасно виявляти і обробляти помилки, забезпечуючи кращий користувацький досвід;

− тестування: всі основні компоненти і функції були ретельно протестовані для забезпечення їх коректної роботи. Використання модульного тестування дозволяє виявляти помилки на ранніх етапах розробки, що знижує витрати на виправлення помилок і забезпечує стабільність застосунка;

− оптимізація продуктивності: впровадження методів оптимізації запитів до бази даних і зменшення кількості викликів до сервера дозволило підвищити продуктивність застосунка. Це забезпечує швидкий відгук застосунка на дії користувача і покращує загальний користувацький досвід.

Використання сучасних технологій і методик, таких як багатошарова архітектура, ORM Entity Framework Core і високопродуктивна СУБД PostgreSQL, дозволило створити надійний і ефективний серверний застосунок. Ці рішення забезпечують високу продуктивність, зручність використання і підтримки коду, а також відповідність сучасним стандартам веб-розробки. Завдяки цим підходам, застосунок забезпечує високий рівень взаємодії з користувачами і стабільність у роботі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Структура бази даних

Програмна реалізація бази даних мого проекту базується на використанні сучасних інструментів та методик для забезпечення високої продуктивності, надійності та зручності в роботі з даними. Основними технологіями, що використовуються, є Entity Framework Core для ORM та PostgreSQL як система управління базами даних.

Основні аспекти реалізації бази даних включають:

− створення сутностей даних;

− конфігурація контексту бази даних;

− використання міграцій для управління змінами в структурі бази даних;

− методи роботи з даними.

Сутності даних представляють таблиці в базі даних. У моєму проекті основними сутностями є ProblemEntity і FileEntity. Ці сутності визначають структуру даних, що зберігаються в таблицях бази даних. Кожна сутність має набір полів, що відповідають стовпцям таблиці. Наприклад, сутність FileEntity містить такі поля, як Id, FileName, Description, ContentType, Path та Time.

Контекст бази даних є центральним компонентом для взаємодії з базою даних. Він містить визначення наборів даних (DbSet), що представляють таблиці в базі даних, а також конфігурації зв’язків між сутностями. Контекст бази даних відповідає за підключення до бази даних, виконання запитів і збереження змін. Наприклад, ProblemsDbContext включає визначення DbSet для сутностей ProblemEntity і FileEntity. Цей контекст також налаштовує з’єднання з базою даних PostgreSQL, що дозволяє легко інтегрувати і використовувати можливості цієї СУБД.

Міграції використовуються для керування змінами в структурі бази даних. Вони дозволяють автоматично застосовувати зміни до бази даних, що

забезпечує безперервний розвиток застосунку без втрати даних. Міграції забезпечують можливість додавання нових таблиць, зміни існуючих структур даних та оновлення зв’язків між сутностями. Цей підхід дозволяє зберігати цілісність даних та забезпечує гнучкість у розробці.

Методи роботи з даними реалізовані у сервісному шарі та забезпечують виконання основних CRUD-операцій (Create, Read, Update, Delete). Основні методи включають:

− створення записів: методи для додавання нових записів до бази даних. Наприклад, метод створення нової проблеми додає новий запис до таблиці Problems. Цей метод отримує дані від клієнта, перевіряє їх і створює новий запис у базі даних за допомогою контексту бази даних;

− читання записів: методи для отримання даних з бази даних. Це включає отримання списку всіх записів або вибіркових даних за певними критеріями. Наприклад, метод для отримання списку всіх проблем виконує запит до бази даних і повертає результати у вигляді колекції об’єктів;

− оновлення записів: методи для оновлення існуючих записів у базі даних. Наприклад, метод оновлення проблеми змінює дані в таблиці Problems. Цей метод отримує оновлені дані від клієнта, перевіряє їх і оновлює відповідний запис у базі даних;

− видалення записів: методи для видалення записів з бази даних. Це включає видалення записів за певними критеріями або ідентифікаторами. Наприклад, метод видалення проблеми видаляє запис з таблиці Problems за заданим ідентифікатором.

У процесі реалізації бази даних у моєму проекті були використані наступні методики:

− багатошарова архітектура: розподіл логіки управління даними на кілька шарів забезпечує кращу організацію коду, модульність і легкість у тестуванні. Кожен шар виконує свою роль, що полегшує розуміння та підтримку коду.

− використання ORM Entity Framework Core: забезпечує зручність роботи з базою даних, дозволяючи розробникам використовувати об’єкти .NET для взаємодії з базою даних. Це знижує кількість помилок, пов’язаних з прямими запитами до бази даних, і спрощує управління даними.

− міграції даних: використання механізмів міграції Entity Framework Core дозволяє легко керувати змінами в структурі бази даних. Це забезпечує безперервний розвиток застосунку без втрати даних і мінімізації ризиків під час оновлень.

− конфігурація зв’язків між сутностями: використання зв’язків між сутностями забезпечує цілісність даних і зручність роботи з пов’язаними таблицями. Наприклад, зв’язок між сутностями ProblemEntity і FileEntity дозволяє легко отримувати файли, пов’язані з певною проблемою.

− оптимізація запитів: впровадження методів оптимізації запитів до бази даних дозволяє підвищити продуктивність застосунка. Це включає використання індексів, оптимізацію SQL-запитів і зменшення кількості викликів до бази даних.

− тестування: всі основні компоненти і функції були ретельно протестовані для забезпечення їх коректної роботи. Використання модульного тестування дозволяє виявляти помилки на ранніх етапах розробки, що знижує витрати на виправлення помилок і забезпечує стабільність застосунка.

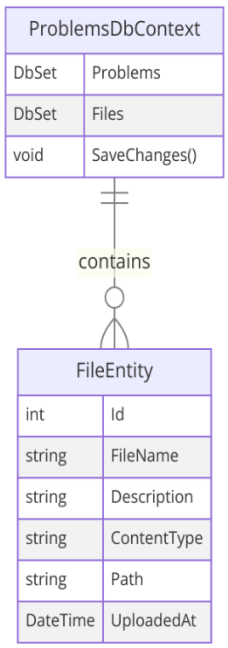

Рисунок 3.1 – Діграма-приклад роботи бази даних з файлами

У моєму проекті реалізація бази даних включає створення та налаштування контексту бази даних, створення сутностей даних, налаштування міграцій та реалізацію методів для роботи з даними.

Контекст бази даних (ProblemsDbContext) забезпечує основний механізм взаємодії з базою даних. Він містить визначення DbSet для кожної сутності, що дозволяє легко виконувати операції CRUD. Контекст налаштовує з’єднання з базою даних PostgreSQL, використовуючи конфігураційні параметри.

Сутність даних, така як і FileEntity, визначає структуру таблиць бази даних і включають атрибути, що відповідають стовпцям таблиць. Вона забезпечують зручний спосіб роботи з даними на рівні об’єктів .NET.

Міграції використовуються для створення та оновлення структури бази даних. Вони автоматично генерують SQL-запити для внесення змін до бази даних, що забезпечує зручність і безпеку при оновленні структури даних.

Методи роботи з даними реалізовані у сервісному шарі. Вони забезпечують виконання основних операцій CRUD, використовуючи контекст

бази даних. Наприклад, метод створення нової проблеми отримує дані від клієнта, перевіряє їх, створює новий об’єкт ProblemEntity і додає його до бази даних за допомогою контексту. Метод читання проблем виконує запит до бази даних і повертає список проблем у вигляді колекції об’єктів.

Використання сучасних технологій і методик, таких як багатошарова архітектура, ORM Entity Framework Core і високопродуктивна СУБД PostgreSQL, дозволило створити надійну і ефективну базу даних для мого застосунку. Ці рішення забезпечують високу продуктивність, зручність використання і підтримки коду, а також відповідність сучасним стандартам веб-розробки. Завдяки цим підходам, база даних застосунку забезпечує надійне зберігання і обробку даних, що є критично важливим для стабільної і ефективної роботи застосунка.

3.2 Реалізація бекенд частини

Програмна реалізація бекенд частини є критичною для забезпечення функціональності веб- застосунків. Бекенд відповідає за обробку запитів від клієнтів, взаємодію з базою даних, а також за бізнес-логіку застосунку. У цьому розділі буде детально розглянуто процес реалізації бекенд частини проекту, використані технології, структуру коду, а також приклади.

У проекті використовується стек технологій, який включає:

– ASP.NET Core - фреймворк для створення веб- застосунків та API;

– Entity Framework Core - ORM (Object-Relational Mapping) для взаємодії з базою даних PostgreSQL;

− PostgreSQL - реляційна база даних для зберігання даних;

− AutoMapper - бібліотека для автоматичного маппінгу між об’єктами;

− Dependency Injection - вбудований механізм для впровадження залежностей.

Структура проекту організована таким чином, щоб забезпечити модульність, зручність в підтримці та масштабованість:

− Controllers - містять контролери для обробки HTTP-запитів;

− Services - містять бізнес-логіку застосунку;

− Repositories - містять код для взаємодії з базою даних;

− Models - містять моделі даних, які відображають структуру бази даних;

− DTOs (Data Transfer Objects) - містять об’єкти для передачі даних між шаром контролерів і сервісів;

− Configurations - містять конфігураційні файли для налаштування застосунку.

Контролери відповідають за обробку HTTP-запитів і виклик відповідних методів сервісів для виконання бізнес-логіки. Наприклад, ProblemController відповідає за роботу з проблемами. Контролер отримує запити від клієнтів, передає їх до сервісного шару для обробки і повертає відповідь клієнту.

ProblemController отримує запит від клієнта на створення нової проблеми. Контролер перевіряє вхідні дані, передає їх до сервісу для обробки і створення нової проблеми. Після успішного створення, контролер повертає відповідь клієнту з інформацією про створену проблему.

На наступній діаграмі це можна подивитися наглядно(рис. 3.2)

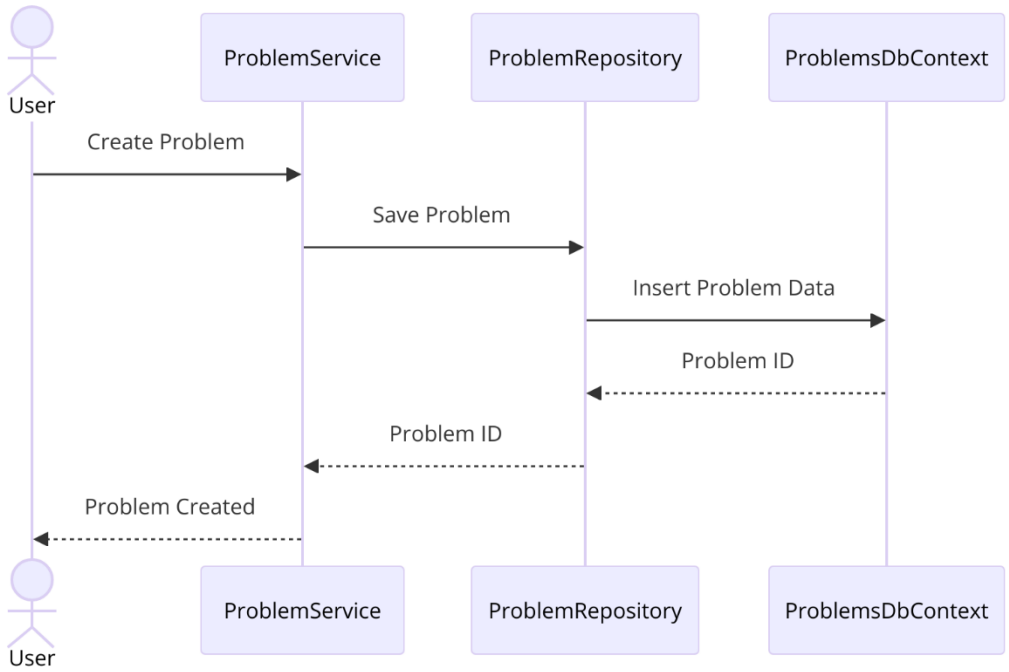

Рисунок 3.2 – Діаграма послідовності для створення проблеми

Сервіси містять бізнес-логіку застосунку і виконують операції, такі як отримання даних з бази даних, обробка даних і взаємодія з іншими сервісами. Наприклад, ProblemService надає методи для створення, читання, оновлення та видалення проблем. Сервіси використовують репозиторії для взаємодії з базою даних.

Репозиторії відповідають за взаємодію з базою даних, включаючи виконання CRUD-операцій (створення, читання, оновлення, видалення даних). Репозиторій є проміжним шаром між сервісами та контекстом бази даних. Наприклад, ProblemRepository реалізує методи для роботи з таблицею проблем у базі даних.

ProblemRepository отримує об’єкт проблеми від сервісу, використовує контекст бази даних для додавання нового запису в таблицю Problems і зберігає зміни. Після цього повертає збережений об’єкт назад до сервісу.

Моделі даних відображають структуру таблиць у базі даних. Наприклад, ProblemEntity представляє сутність проблеми і містить поля, які відповідають стовпцям у таблиці Problems. Моделі використовуються контекстом бази даних для створення і управління таблицями.

Модель ProblemEntity містить поля Id, Title, Description, CreatedDate, Status та інші, які відображають структуру таблиці Problems у базі даних. Модель використовується для створення нових записів у таблиці та для читання даних з бази.

DTOs використовуються для передачі даних між різними шарами застосунку. Вони допомагають зменшити кількість даних, що передаються, і забезпечують безпеку, оскільки дозволяють контролювати, які дані будуть видимі клієнту. Наприклад, ProblemDTO містить тільки ті поля, які необхідні для передачі інформації про проблему від сервісу до контролера і назад.

ProblemDTO містить поля Id, Title, Description, CreatedDate, які використовуються для передачі даних про проблему між сервісом і контролером. DTO дозволяє уникнути передачі зайвих даних, які не потрібні для виконання операцій.

Приклад такої роботи можна побачити на наступній діаграмі (рис. 3.3).

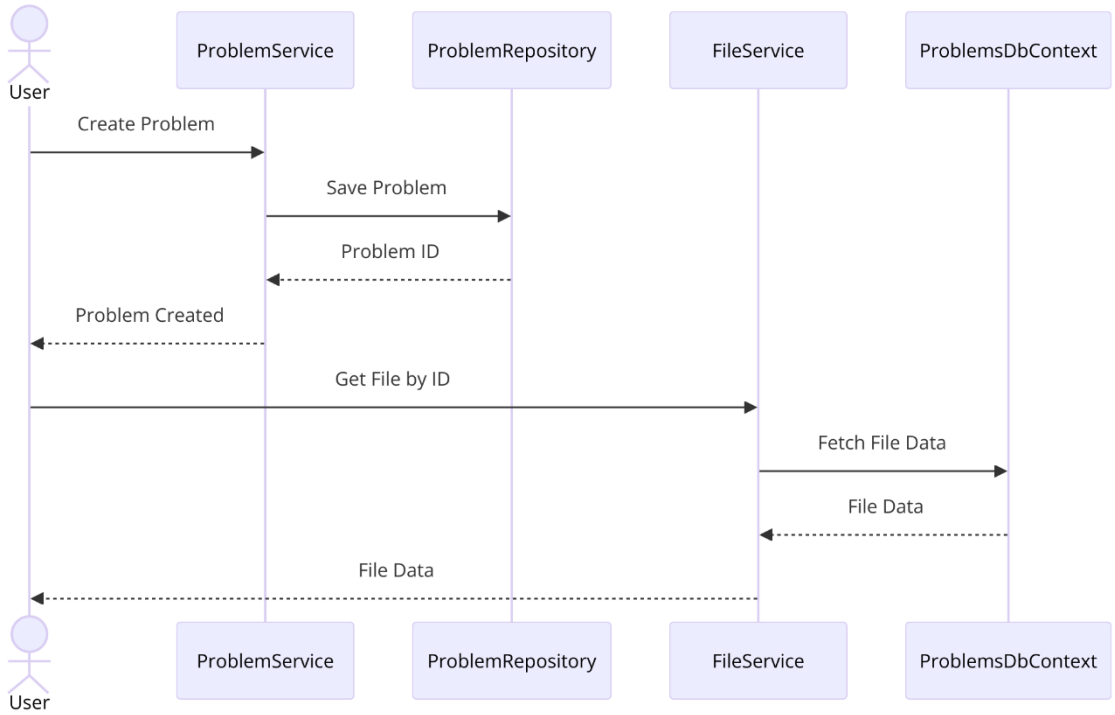

Рисунок 3.3 – Діаграма послідовності для отримання файлу за ідентифікатором

Конфігураційні файли містять налаштування застосунку, такі як рядки підключення до бази даних, параметри автентифікації і авторизації, налаштування для зовнішніх сервісів тощо. Ці файли забезпечують гнучкість і легкість у налаштуванні застосунку без необхідності змінювати код.

Конфігураційний файл містить налаштування рядка підключення до бази даних PostgreSQL, що використовується контекстом бази даних для встановлення з’єднання. Файл також може містити інші параметри, такі як налаштування для обробки помилок, логування та інші параметри конфігурації застосунку.

Кожен з компонентів бекенд частини виконує свою специфічну функцію і взаємодіє з іншими компонентами для забезпечення цілісності та ефективності роботи застосунка. Нижче наведено приклад взаємодії основних компонентів:

Контролери отримують HTTP-запити від клієнтів, викликають відповідні методи сервісів і повертають відповіді клієнтам.

Сервіси виконують бізнес-логіку, використовуючи методи репозиторіїв для взаємодії з базою даних.

Репозиторії використовують контекст бази даних для виконання операцій з даними.

Контекст бази даних (ProblemsDbContext) взаємодіє з сутностями даних (ProblemEntity, FileEntity) для забезпечення доступу до бази даних і виконання CRUD-операцій.

Реалізація бекенд частини у проекті включає створення та налаштування контексту бази даних, створення сутностей даних, налаштування міграцій, реалізацію методів для роботи з даними, створення контролерів, сервісів та репозиторіїв.

Контекст бази даних (ProblemsDbContext) забезпечує основний механізм взаємодії з базою даних. Він містить визначення DbSet для кожної сутності, що дозволяє легко виконувати операції CRUD. Контекст налаштовує

з’єднання з базою даних PostgreSQL, використовуючи конфігураційні параметри.

Сутності даних визначають структуру таблиць бази даних і включають атрибути, що відповідають стовпцям таблиць. Вони забезпечують зручний спосіб роботи з даними на рівні об’єктів .NET.

Міграції використовуються для створення та оновлення структури бази даних. Вони автоматично генерують SQL-запити для внесення змін до бази даних, що забезпечує зручність і безпеку при оновленні структури даних.

Методи роботи з даними реалізовані у сервісному шарі. Вони забезпечують виконання основних операцій CRUD, використовуючи контекст бази даних. Наприклад, метод створення нової проблеми отримує дані від клієнта, перевіряє їх, створює новий об’єкт ProblemEntity і додає його до бази даних за допомогою контексту. Метод читання проблем виконує запит до бази даних і повертає список проблем у вигляді колекції об’єктів.

Контролери обробляють вхідні HTTP-запити і викликають відповідні методи сервісів для виконання бізнес-логіки. Наприклад, ProblemController відповідає за обробку запитів, пов’язаних з проблемами, таких як створення, отримання, оновлення та видалення проблем.

Таким чином, програмна реалізація бекенд частини включає використання сучасних технологій та підходів для забезпечення функціональності і масштабованості застосунку. У даному розділі було розглянуто структуру проекту, використані технології. Це забезпечує високу продуктивність, надійність і зручність у роботі з даними.

3.3 Реалізація фронтенд частини

Реалізація фронтенд частини проєкту є ключовим етапом розробки будь-якого веб-застосунку. Вона забезпечує зручний інтерфейс для взаємодії користувача з системою. В даному розділі детально розглянуто структуру,

функціональні можливості та принципи роботи фронтенд частини, заснованої на скріншотах та описах, наданих користувачем.

Фронтенд частина включає наступні основні компоненти:

– навігаційна панель (рис. 3.4). Містить посилання на головну сторінку та розділ з проблемами;

Рисунок 3.4 – Навігаційна панель

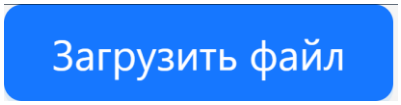

Рисунок 3.5– Кнопка для завантаження файлів.

– список файлів (рис 3.6). Відображає завантажені файли з їх назвами, описами та датами завантаження;

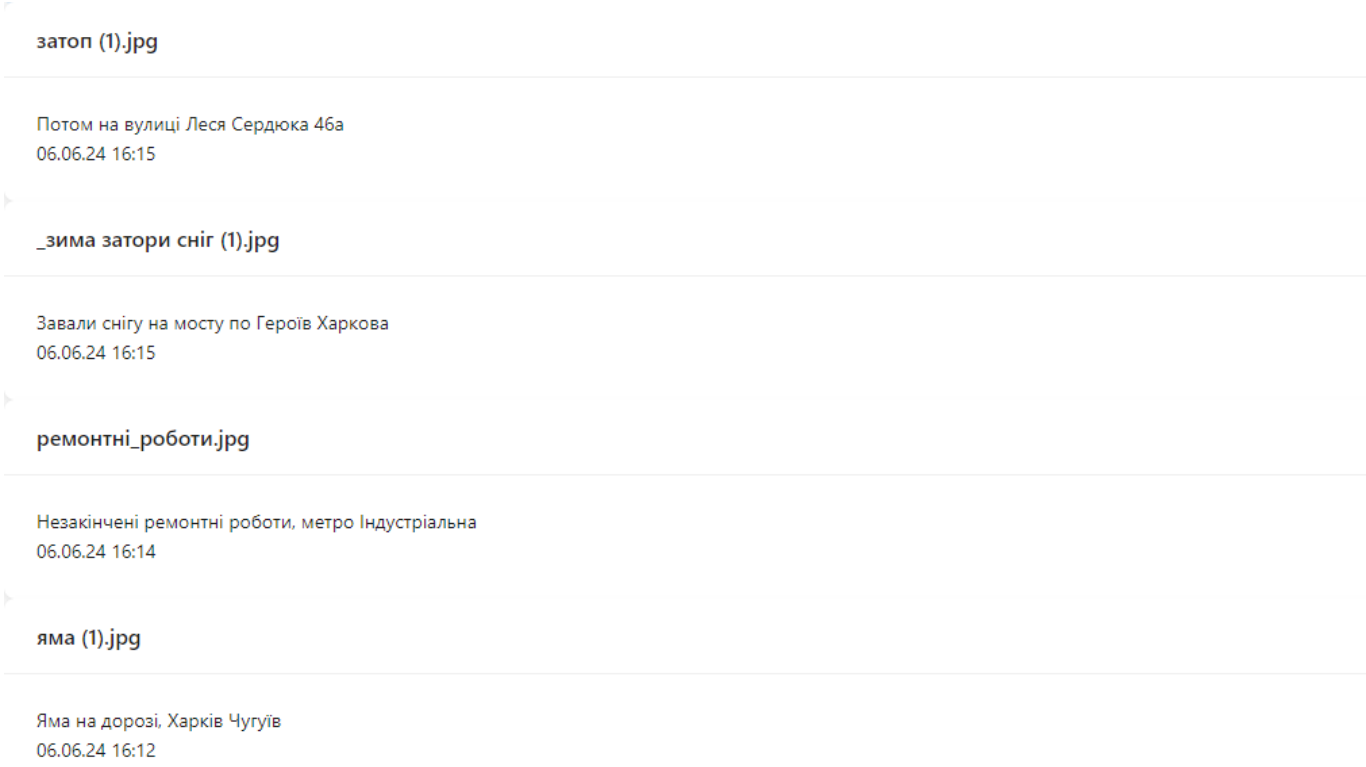

Рисунок 3.6 – Список файлів

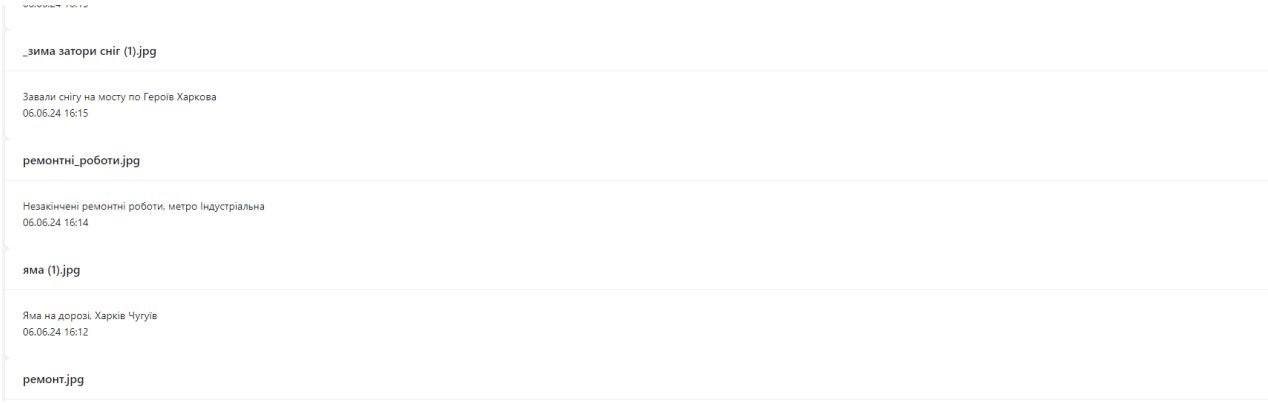

Рисунок 3.7 – Можливість прокручування для перегляду всіх файлів.

Рисунок 3.8 – Можливість дивитися фотографії, опис та час відправки

– футер (рис. 3.9). Містить інформацію про сайт або додаткові посилання.

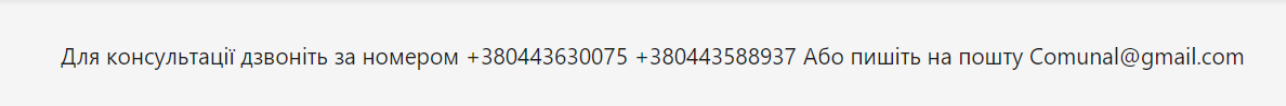

Рисунок 3.9 – Футер

Для реалізації фронтенд частини використовувалися сучасні вебтехнології:

– HTML5: Основна мова для створення структури веб-сторінки;

– CSS3: Використовується для стилізації елементів інтерфейсу;

– JavaScript (ES6): Додає інтерактивність та забезпечує динамічну взаємодію з сервером;

– React: Бібліотека для побудови користувацьких інтерфейсів, яка дозволяє створювати компонентну архітектуру.

Навігаційна панель забезпечує швидкий доступ до основних розділів сайту. Вона включає:

– головна сторінка (Home): Посилання для повернення на основну сторінку сайту;

– розділ «Problems»: Перехід до сторінки з переліком завантажених файлів;

– кнопка «Загрузить файл»: Відкриває діалогове вікно для завантаження нових файлів.

Список файлів відображає всі завантажені файли з наступними характеристиками:

– назва файлу: Ім’я завантаженого файлу, наприклад «IMG_0027.jpg»;

– опис: Короткий опис або коментар до файлу;

– дата і час завантаження: Інформація про те, коли файл був завантажений.

Файли відображаються в вигляді списку, що дозволяє користувачам швидко переглядати та знаходити необхідну інформацію.

Навігаційна панель реалізована за допомогою компонентів React. Ось основні елементи цього компонента:

– компонент NavBar: Відповідає за відображення навігаційної панелі;

– елемент NavLink: Використовується для створення посилань на різні розділи сайту;

– кнопка завантаження: Викликає функцію, що відкриває діалогове вікно для вибору файлів.

– список файлів реалізовано за допомогою компонента FileList, який отримує дані про файли з сховища.

Основні елементи:

– компонент FileItem: Відповідає за відображення окремого файлу, включаючи назву, опис та дату завантаження;

– функція map: Використовується для відображення списку файлів, що зберігаються в стані застосунку;

– футер реалізований як простий компонент, що містить текст або посилання. Він забезпечує додаткову інформацію про сайт або проект.

Для обміну даними з сервером використовуються запити HTTP.

Основні методи включають:

– GET-запити: Використовуються для отримання списку файлів з сервера;

– POST-запити: Використовуються для завантаження нових файлів на сервер;

Запити обробляються за допомогою бібліотеки axios, яка спрощує роботу з HTTP-запитами у JavaScript.

Для забезпечення оновлень в реальному часі використовуються WebSockets. Це дозволяє миттєво оновлювати список файлів при завантаженні нових або видаленні існуючих.

Компонент NavBar є головною навігаційною панеллю, яка забезпечує користувачам можливість швидко переміщатися між основними розділами сайту.

Основні методи та властивості:

– componentDidMount(): Викликається при монтуванні компонента. Виконує GET-запит до сервера для отримання списку файлів;

– renderFileItems(): Використовує функцію map для створення списку компонентів FileItem на основі даних з сховища;

– Компонент FileItem представляє окремий файл у списку файлів;

– render(): Відповідає за відображення назви файлу, опису та дати завантаження;

– handleDeleteClick(): Метод, що викликається при натисканні кнопки видалення. Виконує DELETE-запит до сервера для видалення файлу.

– компонент Footer реалізований для відображення інформації про сайт або проект у нижній частині сторінки.

При завантаженні сторінки компонент NavBar відображає навігаційну панель, яка дозволяє користувачам переходити між розділами сайту. Компонент FileList виконує GET-запит до сервера для отримання списку файлів і зберігає дані в сховищі. Компонент FileItem відображає кожен файл у вигляді окремого елемента списку.

Коли користувач натискає кнопку «Загрузить файл», викликається метод handleUploadClick(), який відкриває діалогове вікно для вибору файлу. Після вибору файлу компонент виконує POST-запит до сервера для завантаження файлу. Новий файл додається до списку файлів у сховищі, і список файлів оновлюється автоматично.

Коли користувач натискає кнопку видалення поруч з файлом, викликається метод handleDeleteClick(), який виконує DELETE-запит до сервера для видалення файлу. Видалений файл видаляється зі списку файлів у сховищі, і список файлів оновлюється автоматично.

Реалізація фронтенд частини проєкту є критичним етапом, що забезпечує зручність користувачів та ефективність роботи системи. Використання сучасних технологій і компонентного підходу дозволяє створити гнучкий та масштабований інтерфейс, який легко підтримувати та розширювати.

3.4 Перспективи подальшої роботи

Перспективи подальшої роботи над розробкою вебзастосунку для організації роботи комунальних служб є досить широкими і включають кілька напрямків, які можуть суттєво покращити функціональність та ефективність системи. Один з ключових напрямків – розширення функціональності застосунку. Наприклад, можна додати модуль для управління проектами комунальних служб, таких як ремонт доріг або модернізація інфраструктури, що дозволить планувати та моніторити виконання робіт. Також варто надати користувачам можливість не лише повідомляти про проблеми, але й брати

участь у їх вирішенні, створюючи спільноти або групи, а також запровадити систему рейтингу для користувачів та служб, що сприятиме підвищенню якості обслуговування та залученню активних громадян.

Інтеграція з іншими системами також є важливим напрямком. Впровадження API для обміну даними з іншими міськими системами, такими як транспортні системи, системи моніторингу якості повітря, енергетичні системи тощо, дозволить створити єдиний портал для громадян, де вони зможуть отримати всю необхідну інформацію та послуги. Інтеграція з платіжними системами надасть можливість здійснювати оплати за комунальні послуги безпосередньо через вебзастосунок, що значно підвищить його зручність.

Використання нових технологій, таких як штучний інтелект та Інтернет речей (IoT), може суттєво підвищити ефективність системи. Штучний інтелект дозволить розробити інтелектуальні алгоритми для аналізу та прогнозування потреб у комунальних послугах, виявлення аномалій та оптимізації процесів. Впровадження чат-ботів та віртуальних асистентів надасть можливість автоматизувати рутинні завдання та підтримку користувачів. Використання IoT дозволить створити сенсорні мережі для моніторингу стану міської інфраструктури, наприклад, для контролю рівня сміття у контейнерах або стану доріг, що забезпечить оперативне реагування на проблеми.

Підвищення безпеки та конфіденційності даних є ще одним важливим напрямком. Впровадження багатофакторної аутентифікації для захисту облікових записів користувачів, а також технологій шифрування для захисту даних під час передачі та зберігання, значно підвищить рівень захищеності системи. Регулярні аудити безпеки допоможуть виявляти та усувати можливі вразливості, забезпечуючи стабільність та надійність роботи вебзастосунку.

Покращення користувацького досвіду є важливим аспектом для подальшого розвитку системи. Регулярні опитування та тестування користувачів дозволять виявляти проблеми та збирати зворотний зв’язок для впровадження необхідних змін. Забезпечення адаптивності інтерфейсу для

коректного відображення на різних пристроях та екранах, оптимізація швидкодії та зручності використання на мобільних пристроях також сприятимуть покращенню користувацького досвіду. Впровадження методів оптимізації завантаження сторінок, таких як lazy loading, мініфікація та обфускація JavaScript-коду, а також кешування статичних ресурсів, дозволить зменшити час завантаження сторінок та підвищити загальну продуктивність застосунку.

Загалом, перспективи подальшої роботи над вебзастосунком для організації роботи комунальних служб є дуже широкими. Розширення функціональності, інтеграція з іншими системами, використання нових технологій, підвищення безпеки та покращення користувацького досвіду дозволять створити ефективну та зручну систему, яка задовольнятиме потреби як комунальних служб, так і громадян.

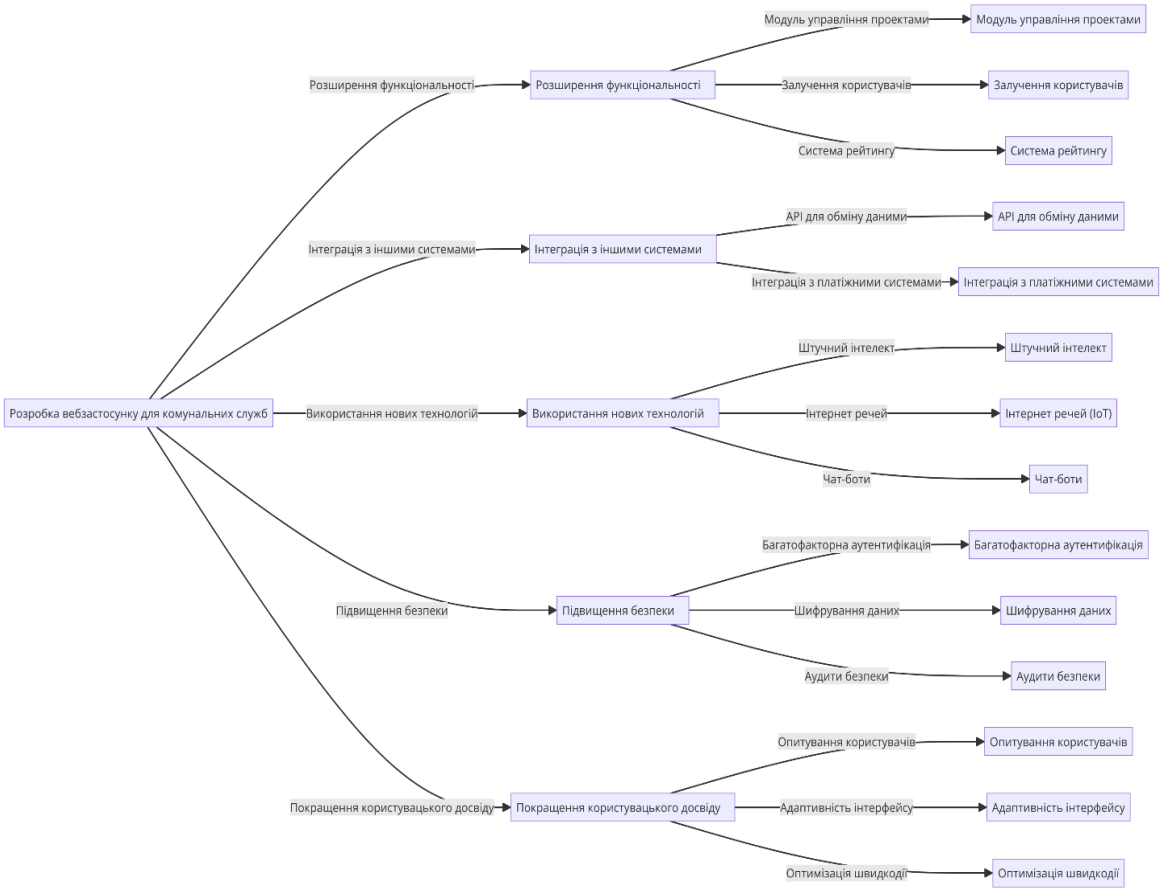

Рисунок 3.10 – Рисунок можливих шляхів зростання та покращення застосунку

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено і реалізовано вебзастосунок для організації роботи комунальних служб. Застосунок успішно здійснює усі базові функції, такі як додавання звітів про проблеми, завантаження фотографій, перегляд заявок, а також управління станом вирішення проблем.

У ході роботи були використані сучасні технології, такі як ASP.NET Core, C#, JavaScript, React, Docker, Docker Compose, PostgreSQL. Це дозволило реалізувати зручний та масштабований інтерфейс користувача та ефективну обробку даних на серверній стороні. Використання цих технологій забезпечило високу продуктивність, безпеку та надійність системи.

Основні досягнення проекту включають:

– фронтенд: Розроблено зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам легко повідомляти про проблеми, завантажувати фотографії та переглядати статус своїх заявок;

– бекенд: Реалізовано надійну серверну частину, що забезпечує обробку та зберігання даних у базі даних PostgreSQL. Використання ASP.NET Core та Docker дозволило забезпечити стабільність і масштабованість системи;

– безпека: Забезпечено захист даних користувачів за допомогою сучасних методів автентифікації та авторизації.

Застосунок має можливість масштабування та розширення, що дозволяє додавати новий функціонал та покращувати існуючий без втрати продуктивності та швидкості роботи. Це робить систему гнучкою і готовою до впровадження нових вимог та змін у майбутньому.

Отже, розроблений вебзастосунок є функціональним та надійним інструментом для організації роботи комунальних служб, який задовольняє всі необхідні потреби користувачів і сприяє покращенню міської інфраструктури.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Rabotiahov, A., Kobylin, O., Dudar, Z., & Lyashenko, V. (2018, February). Bionic image segmentation of cytology samples method. In 2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET) (pp. 665-670). IEEE.

2. Кобилін, О. А., & Творошенко, І. С. (2021). Методи цифрової обробки зображень.

3. Работягов, А. В., Ляшенко, В. В., & Кобылин, О. А. (2016). Сегментация сложных изображений цитологических препаратов.

4. Lyashenko, V., Mohammad, A., & Kobylin, O. (2015). Experiments with Fusion of Images with Use of Wavelet Transformation in Problems of the Text Information Analysis.

5. Kobylin, O., Vyskrebentseva, S., & Petrova, R. (2019). Обробка даних, що містять пропуски в задачах кластеризації. Системи управління, навігації та зв'язку. Збірник наукових праць, 5(57).

6. Oleg, K., Sergii, M., & Mykhailo, S. (2017, October). Video Clustering via Multidimensional Time-Series Analysis. In Proceedings of the 9th International Conference on Information Management and Engineering (pp. 60-63). ACM.

7. Mashtalir, S., Mashtalir, V., & Stolbovyi, M. (2018, August). Representative Based Clustering of Long Multivariate Sequences with Different Lengths. In 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP) (pp. 545-548). IEEE.

8. Bodyanskiy, Y., Kobylin, I., Rashkevych, Y., Vynokurova, O., & Peleshko, D. (2018, February). Hybrid fuzzy-clustering algorithm of unevenly and asynchronously spaced time series in computer engineering. In 2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET) (pp. 930-935). IEEE.

9. Bodyanskiy, Y., Vynokurova, O., Kobylin, I., & Kobylin, O. (2016). Adaptive fuzzy clustering of short time series with unevenly distributed observations in Data Stream Mining tasks. Information Technology and Management Science, 19(1), 23-28.

10. Lyashenko V., Kobylin O., Selevko O. (2020) Wavelet Analysis and Contrast Modification in the Study of Cell Structures Images. International Journal of Advanced Trends in Computer Science and Engineering. 9(4). – 4701-4706.

11. Mashtalir, V., Ruban, I., & Levashenko, V. (Eds.). (2019). Advances in Spatio-Temporal Segmentation of Visual Data (Vol. 876). Springer Nature.

12. Kobylin, O., & Lyashenko, V. (2016). Contrast Modification as a Tool to Study the Structure of Blood Components.

13. Kobylin, O. A., Gorokhovatskyi, V. O., Tvoroshenko, I. S., & Peredrii, O. O. (2020). The application of non-parametric statistics methods in image classifiers based on structural description components. Telecommunications and Radio Engineering, 79(10).

14. Kobylin, O., & Lyashenko, V. (2014). Comparison of standard image edge detection techniques and of method based on wavelet transform.

15. Кобилін, О. А., & Творошенко, І. С. (2021). Методи цифрової обробки зображень.

16. Gorokhovatskiy, V. A., Kobylin, O. A., & Kulikov, Y. A. (2015). Application of Granulation of Feature Descriptions in Structural Image Recognition. Telecommunications and Radio Engineering, 74(6).

17. Kuzminska, O., Mazorchuk, M., Morze, N., & Kobylin, O. (2019, June). Digital learning environment of ukrainian universities: The main components to influence the competence of students and teachers. In International Conference on Information and Communication Technologies in Education, Research, and Industrial Applications (pp. 210-230). Springer, Cham.

18. Kinoshenko, D., Kobylin, O., Mashtalir, S., & Stolbovyi, M. (2019, March). Metric video retrieval speedup by irrelevant data elimination. In Eleventh

International Conference on Machine Vision (ICMV 2018) (Vol. 11041, pp. 176-183). SPIE.

19. Бодянский, Е. В., Винокурова, Е. А., Пелешко, Д. Д., Кобылин, И. О., & Кобылин, О. А. (2017). Нечёткая кластеризация временных рядов с неравномерными и асинхронными тактами квантования. Системи обробки інформації, (5), 47-54.

20. Lyashenko, V., Matarneh, R., Kobylin, O., & Putyatin, Y. (2016). Contour detection and allocation for cytological images using Wavelet analysis methodology.

21. Lyashenko, V., Kobylin, O., & Ahmad, M. A. (2014). General methodology for implementation of image normalization procedure using its wavelet transform.

22. Gorokhovatskyi V., Tvoroshenko I., Kobylin O., and Vlasenko N. (2023) Search for visual objects by request in the form of a cluster representation for the structural image description, Advances in Electrical and Electronic Engineering, 21(1), pp. 19-27.

23. Yakovleva, O., Kovtunenko, A., Liubchenko, V., Honcharenko, V., & Kobylin, O. (2023). Face Detection for Video Surveillance-based Security System (COLINS-2023). In CEUR Workshop Proceedings (Vol. 3403, pp. 69-86).

24. Tvoroshenko I., Gorokhovatskyi V., Kobylin O., and Tvoroshenko A. (2023) Application of deep learning methods for recognizing and classifying culinary dishes in images, International Journal of Academic and Applied Research, 7(9), pp. 57–70.