Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Веб-сервіс для автоматизації інформаційних процесів спільноти мешканців одного чи кількох територіально поєднаних будинків "Житловий комплекс". Back-end
(тема)

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

Волосніков М.С.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Кравець Н.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Волоснікову Миколі Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Веб-сервіс для автоматизації інформаційних процесів спільноти мешканців одного чи кількох територіально поєднаних будинків "Житловий комплекс". Back-end

Затверджена наказом по університету від 20.05. 2024р. № 471 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 01.06.2024

3. Вихідні дані до роботи Розробити BackEnd додаток веб-сервісу для житлових комплексів, який дозволить полегшити отримання необхідних сервісів та комунікацію між мешканцями

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проєктування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	08.04.2024	*виконано*
2	Створення специфікації ПЗ	14.04.2024	*виконано*
3	Проєктування ПЗ	18.04.2024	*виконано*
4	Розробка ПЗ	20.04.2024	*виконано*
5	Тестування ПЗ	07.05.2024	*виконано*
6	Оформлення пояснювальної записки	10.05.2024	*виконано*
7	Підготовка презентації та доповіді	24.05.2024	*виконано*
8	Попередній захист	04.06.2024	
9	Нормоконтроль, рецензування	02.06.2024	
10	Здача роботи у електронний архів	02.06.2024	
11	Допуск до захисту у зав. кафедри	04.06.2024	

Дата видачі завдання 8 квітня 2024р.

Студент (ка) _____ Волосніков М.С.
(підпис)

Керівник роботи _____ доц. кафедри ПІ Кравень Н.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра: 71 с., 14 рис., 13 джерел.

АДМІНІСТРУВАННЯ, ВЕБ СЕРВІС, ЖИТЛОВІ КОМПЛЕКСИ, КОМУНАЛЬНІ ПОСЛУГИ, МОВА ПРОГРАМУВАННЯ C#, ФРЕЙМВОРК .NET, SIGNALR, JETBRAINS RIDER

Об’єкт розробки – BackEnd частина веб-сервісу для житлових комплексів.

Мета розробки – створення API для роботи з веб-сервісом для покращення функціонування житлових комплексів.

Метод рішення – середовище розробки JetBrains Rider, фреймворк .NET, мова програмування C#, платформа Azure.

У результаті розробки створено BackEnd додаток для веб-сервісу, який дозволяє полегшити отримання необхідних сервісів в житловому комплексі.

ADMINISTRATION, WEB SERVICE, RESIDENTIAL COPLEXES, COMMUNAL SERVICES, C# PROGRAMMING LANGUAGE, .NET FRAMEWORK, SIGNALR, JETBRAINS RIDER

The object of Development – the design and development of the BackEnd part of a web service for residential complexes.

The purpose of Development – create an API for interacting with a web service to improve the functioning of residential complexes.

Solution Method – JetBrains Rider development environment, .NET framework, C# programming language, Azure platform.

As a result of development a BackEnd application for a web service has been created, which facilitates access to services within the residential complex.

Я, Волосніков Микола Сергійович, студент гр. ПЗПІ-20-1, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Веб-сервіс для автоматизації інформаційних процесів спільноти мешканців одного чи кількох територіально поєднаних будинків "Житловий комплекс". Back-end», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

У сучасному світі, в якому технології розвиваються невпинно, неможливо недооцінювати важливість розробки програмних систем для покращення якості щоденного життя, в тому числі для сфери житлово-комунального господарства. Це галузь, яка в умовах, сьогодення стикається зі складними викликами та потребує вдосконалення процесів управління та обслуговування, щоб забезпечити комфорт та задоволення потреб споживачів.

Перш за все, розробка веб-сервісу для житлових комплексів дозволить автоматизувати велику кількість процесів, пов'язаних з обліком, плануванням та наданням послуг ЖКГ. Вона дозволить оптимізувати робочі процеси, зменшуючи час і зусилля, витрачені на завдання, пов’язані з адмініструванням, і спростить взаємодію між мешканцями та управляючими компаніями чи ОСББ, а також надасть можливість прямої комунікації мешканцям територіально поєднаних будинків.

По-друге, така система покращить контроль за станом житлового комплексу. Вона забезпечить можливість відстеження технічного стану будівель, систем опалення, водопостачання та інших комунікацій, що дозволить вчасно виявляти та усувати несправності, запобігаючи виникненню аварійних ситуацій.

Крім того, веб-сервіс для житлових комплексів сприятиме підвищенню якості обслуговування мешканців. Завдяки цифровій платформі мешканці матимуть змогу швидко та зручно звертатися із заявами, відстежувати їх статус та отримувати оперативну інформацію про надані послуги. Це зробить процес обслуговування більш прозорим і ефективним, сприяючи задоволенню клієнтів та підвищенню рівня їх довіри до управління. Крім того такий веб-сервіс надасть змогу адміністрації

житлових комплексів створювати оголошення та проводи голосування стосовно важливих питань.

Нарешті, розробка такої веб-сервісу є ключовим елементом цифрової трансформації у цій сфері. Вона дозволить суттєво підвищити ефективність управління ресурсами, зменшити витрати та ризики, пов'язані з недоліками у системі управління.

Отже, метою роботи є розробка BackEnd API для веб-сервісу для житлових комплексів. Галузю використання такої системи будуть територіально поєднанні будинки, яким вона допоможе покращити якість життя мешканців та сприятиме створенню більш ефективної та стабільної інфраструктури.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Житлово-комунальне господарство стикається з численними тенденціями та викликами, які впливають на його функціонування та розвиток.

Однією з головних тенденцій є стрімкий розвиток технологій у цій галузі, оскільки впровадження сучасних інформаційних технологій дозволяє автоматизувати багато процесів управління, забезпечуючи більшу ефективність та зручність для мешканців. Наприклад, системи "розумного" управління житловими будинками дозволяють зменшити витрати на комунальні послуги та підвищити комфорт проживання.

Ще однією тенденцією є зростання екологічних вимог до житлово-комунального господарства. У зв'язку зі збільшенням обсягів викидів та відходів, виникає потреба впровадження екологічних технологій та підходів до управління відходами. Сприяє цьому і розвиток концепції "зеленого будівництва", яка передбачає зменшення негативного впливу будівництва на навколишнє середовище.

Наступною тенденцією є зростання кількості міського населення та концентрація людей у великих містах. Це ставить існуючі комунальні системи та інфраструктуру під підвищений тиск, що вимагає їх постійного модернізації та розвитку.

Зараз житлово-комунальне господарство також стикається з рядом викликів, серед яких особливо важливим є питання сталого розвитку. Сучасне суспільство ставить перед собою завдання забезпечення економічної ефективності, соціальної справедливості та екологічної стійкості. Відповідно, управління житлово-комунальним господарством повинне бути спрямоване на досягнення цих цілей, враховуючи інтереси

різних соціальних груп, включно з людьми обмеженими можливостями, та забезпечуючи збалансований розвиток міст.

Крім того, до викликів галузі варто віднести нестабільність в управлінні, високу вартість та недостатню доступність житлового фонду для ряду соціальних груп, а також потребу вдосконалення законодавства та регулювання у цій сфері.

Усі ці тенденції та виклики вимагають від галузі житлово-комунального господарства постійного удосконалення та адаптації до змін, що активно відбуваються у суспільстві та економіці.

Використання цифрових технологій в сфері ЖКГ також може допомогти автоматизувати багато рутинних операцій управління, таких як облік наявних ресурсів, моніторинг стану об'єктів інфраструктури, планування обслуговування, тощо. Це дозволяє зменшити витрати на адміністрування та оптимізувати робочі процеси. Більше того новітні технології здатні значно покращити комунікацію між мешканцями та управлінням житлових комплексів, спростити процедури заявок та скарг, а також забезпечити швидку реакцію на них. Це дозволяє підвищити задоволеність клієнтів та покращити їхнє життя.

В подальшому впровадження таких цифрових рішень дозволить реалізувати концепцію "розумного" міста, де системи управління житлово-комунальною інфраструктурою інтегровані в єдину мережу. Це дає змогу ефективно координувати роботу різних служб, реагувати на надзвичайні ситуації та забезпечувати безпеку та комфорт для мешканців.

У цілому, цифрова трансформація в сфері ЖКГ не лише поліпшує управління та якість життя мешканців, але й сприяє сталому розвитку міст та збереженню навколишнього середовища.

1.2 Огляд конкурентів

На ринку систем для управління житловими комплексами існує кілька конкурентів, які пропонують схожі послуги. Розглянемо деякі з них.

Condo Control Central також має широкий (див. рис. 1.1) та активно використовується в сполучених штатах. Вона спрямована на забезпечення основних потреб управління ЖК. З основних переваг BackEnd функціоналу в Condo Control Central можна виділити наступне:

— організація заходів;

— замовлення послуг

— оплата комунальних послуг;

— електроне голосування.

З недоліків варто зазначити неможливість комунікації між мешканцями всередині веб-сервісу.

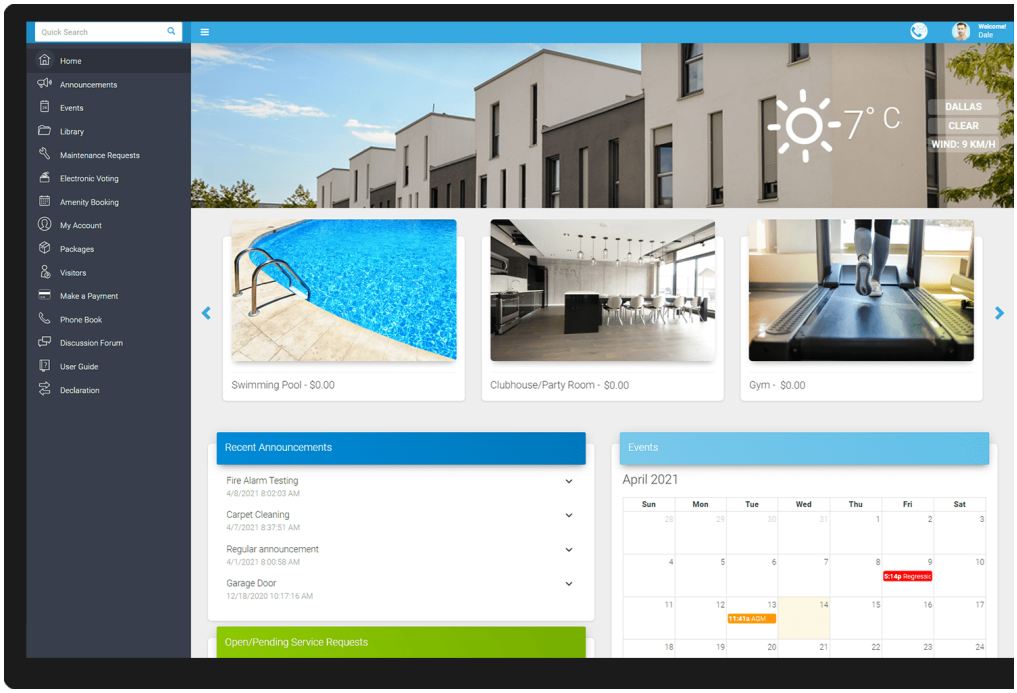


Рисунок 1.1 – Інтерфейс головної сторінки Condo Control Central (за даними [1])

Іншим потенційним конкурентом можна вважати BuildingLink (див. рис. 1.2). Вона спеціалізується на об’єктах, що будуються або були введені в експлуатацію впродовж останніх декількох років. З переваг BuildingLink важливо вказати простоту в використанні, а серед основних недоліків можна зауважити, що даний веб-сервіс є досить застарілим в технічному плані, не відповідає актуальним стандартам безпеки та надає відносно невелику кількість послуг, серед яких найважливішими є створення заявок та планування заходів.

Рисунок 1.2 – Інтерфейс сторінки адміністратора BuildingLink (за даними [2])

Інший потенційний конкурент на ринку систем для управління нерухомістю це Yardi Voyager (див. рис. 1.3), який пропонує більш загальну платформу для управління нерухомістю, яка охоплює більший спектр нерухомості, а не лише житлові комплекси. З переваг веб-сервісу Voyager варто зазначити велику кількість функціоналу, однак велика кількість

функцій не може бути використана для житлових комплексів, оскільки розрахована на інші види нерухомості, наприклад можливість розрахунку вартості оренди по договорам суборенди для великої комерційної нерухомості.

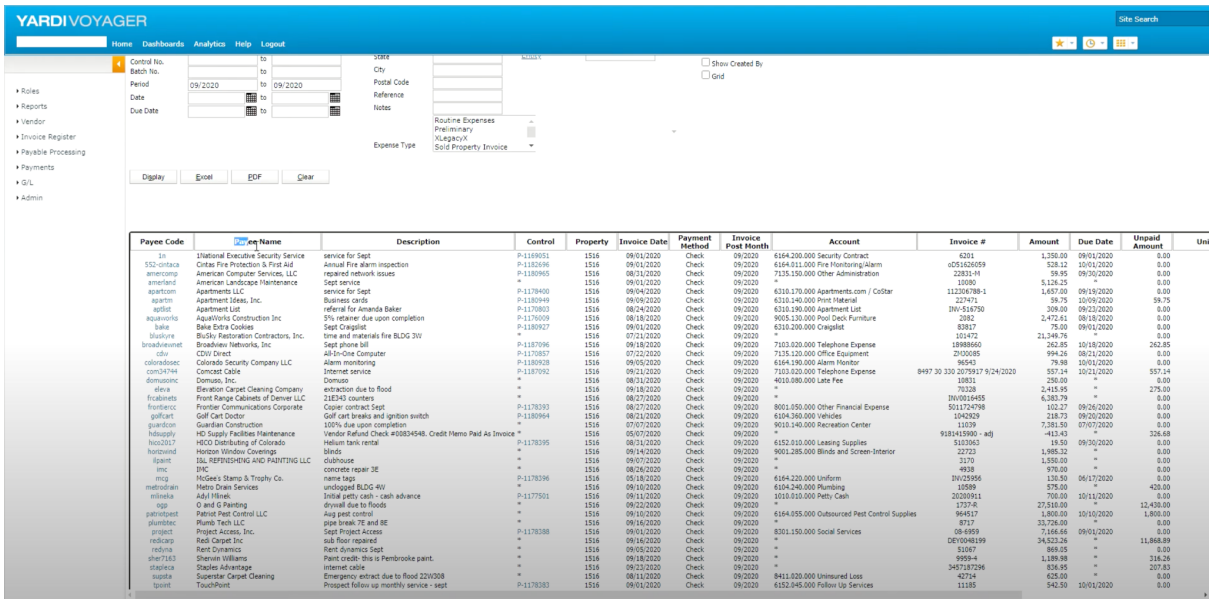


Рисунок 1.3 – Інтерфейс сторінки з виставленими рахунками Yardi Voyager (за даними [3])

Розроблюваний веб-сервіс вирізняється широким функціоналом, гнучкістю, персоналізацією та можливістю інтеграції з великою кількістю житлових комплексів, що робить його привабливим вибором для управління нерухомості. В порівнянні з іншими системами вона може надати більше можливостей та глибше розуміння потреб управління та мешканців. Веб-сервіс буде створюватися з урахуванням потреб сучасних житлових комплексів та їх мешканців. Він зможе запропонувати широкий спектр функцій, які охоплюють всі аспекти управління ЖК, від замовлення послуг до аналітики та звітності. Веб-сервіс також буде надавати можливість зробити взаємодію з надавачами комунальних послуг ще зручнішою та ефективнішою.

1.3 Виявлення проблем та актуалізація рішень

Виявлення проблем та актуалізація рішень в сфері модернізації роботи житлових комплексів є ключовим етапом для покращення ефективності та якості життя, а також для створення веб-системи кращої від конкурентів. Ось декілька кроків, які можуть бути вжиті для виявлення проблем та їхнього вирішення:

— проведення комплексного аналізу стану інфраструктури, щоб виявити проблемні ділянки, які потребують негайного втручання. Це може бути звірено з потребами мешканців та сучасними вимогами до інфраструктури. На відміну від деяких конкурентів наш веб-сервіс буде фокусуватися саме на житлових комплексах, а не на великій кількості нерухомості, що дозволить провести більш детальний аналіз;

— впровадження системи збору та аналізу даних для оцінки рівня задоволеності мешканців, ідентифікації проблемних ситуацій та моніторингу результатів заходів їх вирішення. Це дозволить здійснювати об'єктивну оцінку ситуації та приймати дієві рішення;

— виявлення найбільш критичних проблем, а також проведення їх приоритизації з урахуванням їхнього впливу на користувачів та бізнес-процеси;

— запроваджувати нові рішення та проводити постійній моніторинг їхньої ефективності, що допоможе забезпечити постійне вдосконалення та коригування процесів в залежності від результатів. Важливо, щоб розроблювана система використовувала нові технології та виконувала усі необхідні вимоги щодо безпеки даних, що є великою проблемою для деяких конкурентів;

— застосування сучасних цифрових технології, моніторинг та керування витратами енергії та води, а також впровадження електронних

сервісів для зручного подання заяв та подальше отримання необхідних мешканцям послу, а також можливість контролю за виконанням цих заяв;

— активна комунікація та співпраця з мешканцями, управляючими компаніями, місцевими органами влади та іншими зацікавленими сторонами для спільного виявлення та вирішення нагальних потреб;

— забезпечення постійного контролю за впровадженням рішень та їхнім впливом на якість комунальних послуг. Це допомагає своєчасно виявляти неполадки та коригувати стратегію дій, як в розрізі окремого житлового комплексу, так і в розрізі міста в цілому;

— встановлення механізмів зворотного зв'язку та постійний пошук нових шляхів для поліпшення якості та доступності комунальних послуг. Спільна робота над ініціативами та проєктами з розвитку ЖКГ допоможе досягти значних покращень у цій сфері.

Актуалізація рішень в сфері модернізації послуг в житлових комплексах має на меті забезпечити ефективний та сталий розвиток міського господарства, покращити якість життя мешканців та забезпечити екологічну стійкість.

1.4 Постановка задачі

Задача розробки веб-сервісу для житлових комплексів передбачає створення інтегрованої платформи, що сприятиме ефективному управлінню житловими об'єктами та полегшить взаємодію між мешканцями та управляючими компаніями. Детальна постановка задачі включає наступні аспекти:

а) Модуль замовлення та відстеження послуг;

1) реєстрація та відстеження заявок на побутові послуги (ремонт, прибирання, сантехнічні роботи тощо);

2) створення певної задачі, відстеження статусу виконання;

3) оцінка якості наданих послуг та збір відгуків мешканців;

б) внутрішні чати для мешканців;

1) можливість створення групових чатів для обговорення питань, пов'язаних з конкретним будинком або комплексом;

2) обмін інформацією між мешканцями, включаючи попередження про плановані ремонтні роботи або інші події;

в) оголошення від адміністрації та мешканців;

1) модуль оголошень для адміністрації, де можна публікувати інформацію про плановані роботи, графіки відключень, загальні збори тощо;

2) функція для мешканців, щоб розміщувати свої оголошення або сповіщення (наприклад, про збір коштів на ремонт чи організацію заходів);

г) оплата комунальних послуг;

1) можливість перегляду та оплати рахунків за комунальні послуги через онлайн-платформу з використанням даних з API третіх сервісів;

2) відображення історії споживання ресурсів (електроенергії, води, опалення тощо) для кожного мешканця відповідно до показників лічильників;

ґ) модуль голосування;

1) функція проведення голосувань серед мешканців щодо важливих питань будинку або житлового комплексу;

2) забезпечення анонімності голосів та відображення результатів голосування для всіх зацікавлених сторін;

д) система аналітики та звітності;

1) збір та аналіз даних про використання ресурсів, обсяги замовлень послуг, рівень задоволеності мешканців тощо;

2) підготовка звітів для управляючих компаній та адміністрації ЖК для прийняття обґрунтованих управлінських рішень.

Розробка веб-сервісу з таким функціоналом допоможе покращити якість життя мешканців, забезпечити ефективне управління житловими комплексами та підвищити рівень задоволеності клієнтів.

Розробка BackEnd для веб-системи для управління житловими комплексами буде виконуватися за допомогою фреймворку .NET Core і буде включати наступні кроки: проєктування архітектури системи, розробка, тестування, впровадження, налаштування, підтримка та після випускне обслуговування. Під час аналізу вимог здійснюється збір вимог від представників управління ЖК та мешканців, формулювання функціональних та нефункціональних вимог. Після цього проводиться проєктування архітектури системи та вибір оптимальних технічних рішень. Розробка з точки зору BackEnd включає створення API для обробки запитів від клієнтської частини системи. Процес тестування включатиме в себе юніт та інтеграційне тестування, а також тестування відповідності вимогам. Впровадження передбачає деплоймент системи на сервері та налаштування доступу користувачів ЖК. Після впровадження забезпечується підтримка системи та вирішення можливих проблем.

Також в подальшому важливим аспектом для розвитку системи, що розроблюється, може стати інтеграція з іншими сервісами, такими як платіжні шлюзи чи системи електронного документообігу, для розширення функціональності BackEnd частини веб-сервісу.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Функціональні вимоги

Функціональні вимоги в програмній системі для управління житловими комплексами є ключовим елементом, який визначає, як система буде взаємодіяти з користувачами та які функції вона має надавати [4]. В веб-системі для житлових комплексів передбачаються два види користувачів:

— адміністратор, який має доступ до усього функціоналу в веб-сервісу та даних.

— мешканець, що може використовувати веб-сервіс для замовлення послуг, комунікації з адміністратором та іншими мешканцями, тощо.

З точки зору BackEnd в ході роботи над веб-сервісом для житлових комплексів необхідно розробити API, яке буде підтримувати наступні дії:

а) аутентифікація та авторизація;

1) реєстрація та вхід в систему для користувачів з різними ролями, такими як мешканець або адміністратор;

2) контроль доступу до функціональності системи в залежності від ролей користувачів, адміністратор має мати доступ до більшого функціоналу, такого як створення чатів, оголошень, опитувань, тощо;

б) модуль замовлення та відстеження послуг;

1) мешканець має мати змогу створювати, переглядати та відстежувати статусу заявок на побутові послуги;

2) нотифікації користувачам-мешканцям про зміни статусу їхніх заявок;

в) внутрішні чати для мешканців;

1) можливість створення та управління чатами адміністратором для обговорення питань між мешканцями певного будинку або під'їзду;

г) модуль оголошень;

1) можливість публікації та перегляду мешканцями оголошень від адміністрації ЖК;

2) можливість коментувати та реагувати на оголошення;

д) система оплати та фінансова звітність;

1) онлайн-оплата рахунків за комунальні послуги;

2) відображення історії платежів та можливість завантаження рахунків у форматі PDF;

е) модуль голосування;

1) проведення голосувань серед мешканців за важливі питання, такі як ремонт, обслуговування будинку, вибір постачальників послуг тощо, створення таких голосувань при цьому покладається на адміністратора.

2.2 Нефункціональні вимоги

До нефункціональних вимог веб-сервісу можна віднести:

а) продуктивність;

1) швидкість реакції системи на запити користувачів та завантаження сторінок менше 200 мс;

2) мінімальні затримки при використанні функціональності;

б) безпека;

1) захист особистих даних користувачів та конфіденційної інформації відповідно до вимог GDPR;

2) запобігання атакам на систему та збереження цілісності даних за допомогою сервісів Azure;

в) масштабованість;

1) можливість масштабування системи з урахуванням кількості активних користувачів та об'ємом даних, що зберігаються;

г) надійність;

1) стабільна робота системи без перебоїв та відмов, доступність системи впродовж мінімум 99% часу;

д) підтримка;

1) доступність допомоги користувачам та швидке вирішення їхніх запитів;

2) регулярне оновлення системи та постійна підтримка;

Ці вимоги допоможуть побудувати систему, яка відповідає потребам користувачів та забезпечує їхній зручний та ефективний досвід користування.

2.3 Вимоги до середовища розгортання

Середовище розгортання для розроблюваної веб-системи управління житловими комплексами повинно відповідати ряду вимог для забезпечення ефективної та безпечної роботи системи. Для початку необхідно мати доступ до серверів або хмарних платформ, які підтримують .NET Core для розгортання backend частини та Node.js для запуску frontend додатку React JS, наприклад Microsoft Azure [5]. Також важливо мати базу даних, сумісну з .NET Core, до прикладу, Azure SQL, щоб забезпечити зберігання даних на backend стороні веб-сервісу. Для забезпечення безпеки мережевого зв'язку варто застосовувати шифрування та протокол HTTPS.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

3.1.1 Функціональне моделювання

Функціональне моделювання при розробці програмного забезпечення є ключовим інструментом для аналізу, проєктування та оптимізації складних систем. Використання функціональних моделей дозволяє уявити систему у вигляді набору функцій, які вона виконує, та взаємозв'язків між цими функціями. Це допомагає виявити ключові елементи системи та їх функціональні залежності.

Метою функціонального моделювання є розкрити складність системи, зокрема визначити важливість різних компонентів та їх взаємозв'язки. Це допомагає виявити можливі проблеми або вузькі місця у функціонуванні системи та спрямувати увагу розробника на ключові аспекти для подальшого вдосконалення.

Для даної кваліфікаційної роботи функціональне моделювання є основою для розробки та тестування гіпотез, а також для визначення стратегій оптимізації системи. Воно дозволяє отримати глибше розуміння функціональних характеристик системи та спрямувати їхні зусилля на досягнення конкретних цілей дослідження.

Під час проєктування веб-сервісу для житлових комплексів було виділено два основних види користувачів, а саме:

— мешканець ЖК, тобто людина, яка безпосередньо проживає в житловому комплексі;

— адміністратор ЖК, який відповідальний за управління житловим комплексом та задоволення потреб його мешканців.

Під час аналізу вимог та функціонального моделювання веб-сервісу для житлових комплексів було зазначено наступні варіанти використання ПЗ користувачами:

— увійти в систему: мешканець або адміністратор може увійти в систему, використовуючи свої електронну пошту та пароль;

— створити запит на реєстрацію: мешканець може створити запит на реєстрацію для нового мешканця;

— відредагувати дані особистого облікового запису: мешканець може редагувати свої дані особистого облікового запису, такі як ім'я, електронна пошта та номер телефону;

— надіслати запит на обслуговування: мешканець може створити запит на обслуговування, щоб повідомити про проблему в житловому комплексі;

— проголосувати в опитуванні: мешканець може проголосувати в опитуванні, яке створив адміністратор;

— створити опитування: адміністратор може створити опитування, щоб отримати відгуки від мешканців;

— переглянути результати опитування: мешканець або адміністратор може переглянути результати опитування;

— створити запит на обслуговування: мешканець може створювати запити на обслуговування, пов’язані з ремонтом, заміною лічильників, тощо;

— переглянути деталі запиту на обслуговування: мешканець або адміністратор може переглянути деталі запиту на обслуговування;

— створити чат: мешканець може створити чат, щоб спілкуватися з іншими мешканцями;

— запросити учасників чату: мешканець або адміністратор може запросити інших мешканців приєднатися до чату;

— переглянути повідомлення чату: мешканець може переглянути повідомлення чату;

— опублікувати повідомлення в чаті: мешканець може опублікувати повідомлення в чаті;

— отримати повідомлення про майбутні події електронною поштою: мешканець може завчасно отримувати повідомлення електронною поштою про майбутні події в житловому комплексі;

— надіслати дані лічильників: мешканець може надсилати дані зі своїх лічильників до системи;

— переглянути звіт про дані лічильників: адміністратор може переглянути звіт про дані лічильників, надані мешканцями;

— переглянути тарифи на комунальні послуги: мешканець може переглянути тарифи на комунальні послуги в житловому комплексі;

— сплатити рахунок за комунальні послуги: мешканець може сплатити рахунок за комунальні послуги;

— переглянути календар: мешканець може переглянути календар подій, які відбуваються в житловому комплексі;

— створити подію: адміністратор може створити подію, яка відбудеться в житловому комплексі;

— оновити деталі події: адміністратор може оновити деталі події, яка відбудеться в житловому комплексі;

— змінити статус запиту на обслуговування: адміністратор може змінити статус запиту на обслуговування;

— редагувати тарифи на комунальні послуги: адміністратор може редагувати тарифи на комунальні послуги в житловому комплексі;

— створити звіт про боржників: адміністратор може створити звіт про боржників, які не сплатили свої рахунки за комунальні послуги;

— керувати записами про будівлю: адміністратор може керувати записами про будівлю, такі як інформація про квартири та мешканців.

Зв’язки між користувачами системи та функціями, яка вона надає, детально зображені на діаграмі варіантів використання (див. рис. 3.1).

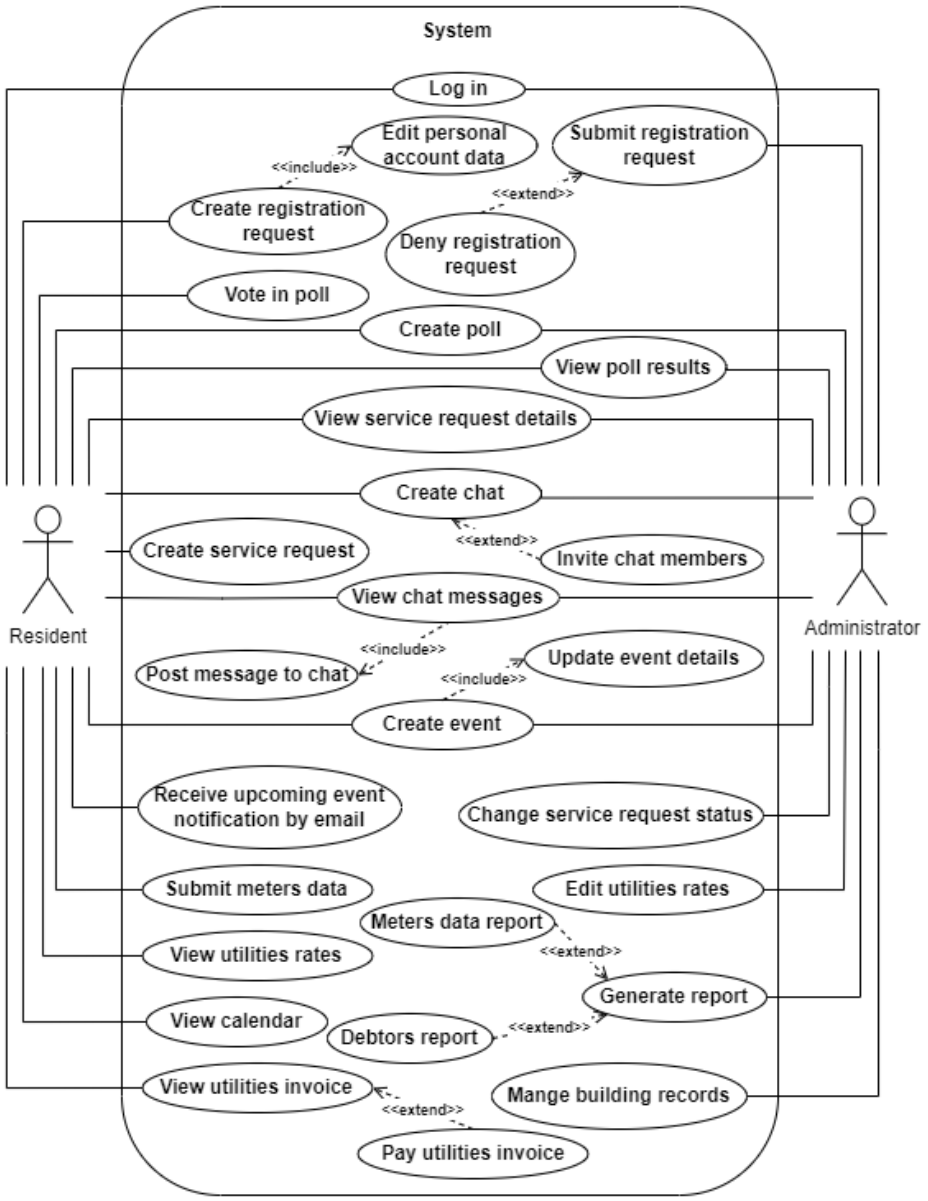


Рисунок 3.1 – Use Case діаграма (виконаний самостійно)

Таким чином, згідно із зазначеними раніше функціональними вимогами, були визначені та проілюстровані основні функції веб-сервісу для житлових комплексів.

3.2 Проєктування архітектури ПЗ

Розглядаючи вибір архітектури веб-сервісу «Житловий комплекс», можна виділити кілька підходів. Монолітна архітектура, де всі компоненти системи інтегровані в єдиний виконуваний файл, може спростити розробку та розгортання, але ускладнити масштабування та оновлення окремих частин системи. Сервіс-орієнтована архітектура (SOA) дозволяє інтегрувати різноманітні служби, але може збільшити затримки в комунікації та ускладнити управління залежностями. Мікросервісна архітектура надає високий рівень гнучкості та незалежності компонентів, але вимагає складного управління та може призвести до дублювання функціоналу [6].

З іншого боку, привабливим вибором є чиста, або Onion, архітектура. По-перше, вона забезпечує чітке розділення відповідальності між різними рівнями системи, що сприяє легкості тестування та підтримки [7, с. 247]. По-друге, централізація бізнес-логіки в ядрі системи дозволяє легко адаптуватися до змін у вимогах без необхідності зміни зовнішніх шарів, таких як інтерфейс користувача або інтеграція з іншими системами. По-третє, Onion архітектура сприяє використанню принципів інверсії залежностей, що зменшує залежність від конкретних технологій та фреймворків, забезпечуючи вищу ступінь гнучкості при виборі інструментів для реалізації системи.

Враховуючи потребу інтеграції з системами оплати та сховищами файлів, Onion архітектура дозволяє ізолювати ці інтеграції в окремих

модулях, що спрощує їх заміну або оновлення в майбутньому без впливу на основну бізнес-логіку. Також, ця архітектура підтримує принципи SOLID, що є важливим для підтримки та розширення великих систем, як «Житловий комплекс» [8]. Вибір Onion архітектури для системи, що розробляється, обґрунтований її здатністю адаптуватися до змінних вимог бізнесу, забезпечуючи при цьому стабільність та легкість управління складною системою. Тож чиста, або Onion архітектура, є оптимальним вибором для веб-сервісу «Житловий комплекс» з урахуванням усіх потреб до функціоналу.

Відповідно до обраної архітектури на BackEnd частині веб-сервісу буде використано наступні шаблони проєктування:

— Mediator, який у .NET допомагає забезпечити централізовану обробку запитів, зменшуючи прямі залежності між компонентами, а також спростити обробку запитів і команд, забезпечуючи чисту архітектуру;

— Chain of Responsibility, що є ключовим компонентом у .NET Core Middleware для обробки HTTP-запитів та відповідей. Він дозволяє додавати додаткову логіку на кожному етапі обробки запиту, включаючи автентифікацію, авторизацію, логування та обробку помилок;

— Unit of Work, який в рамках .NET Entity Framework дозволяє ефективно управляти змінами в базі даних, зменшуючи ризик неконсистентних даних та забезпечуючи атомарність операцій.

З урахуванням обраної архітектури, веб-сервіс «Житловий комплекс» складається з наступних компонентів:

а) Клієнтський додаток;

1) Models: моделі даних, з якими взаємодіє клієнтський додаток;

2) Stores: зберігання та управління даними клієнтського додатку;

3) Router: відповідає за маршрутизацію запитів користувачів до відповідних компонентів. Він використовує правила маршрутизації для визначення того, який контролер має обробити конкретний запит;

4) Components: ці компоненти є модульними блоками користувацького інтерфейсу клієнтського додатку;

5) SignalR: використовується для передачі даних у реальному часі між клієнтом та сервером [9];

6) App Settings: зберігає конфігураційні параметри клієнтського додатку;

б) WebAPI;

1) Controllers: кінцеві точки серверної частини. Вони отримують запити, валідують їх, та викликають функціонал шару додатків;

2) Extensions: додають нові функції до API. Вони можуть бути використані для додання автентифікації, авторизації або інших можливостей;

3) Middleware: використовується для перехоплення та обробки запитів до API. Він може використовуватися для таких завдань, як автентифікація, авторизація, валідація або реєстрування;

в) шар інфраструктури;

1) Settings: цей компонент зберігає конфігураційні параметри інфраструктури, такі як підключення до бази даних, URL-адреси служб та інші налаштування;

2) Accessors: ці компоненти забезпечують доступ до даних та ресурсів. Вони можуть використовуватися для доступу до бази даних, файлів або інших джерел даних;

г) шар додатку;

1) Helpers: ці компоненти надають допоміжні функції для загальних завдань, таких як валідація, форматування даних або обробка помилок;

2) Mapping: цей компонент використовується для відображення зв'язків між компонентами;

3) DTO: об'єкти, які використовуються для передачі даних між компонентами;

4) Exceptions: об'єкти для обробки помилок;

5) MediatR: бізнес-логіка системи;

ґ) шар збереження даних;

1) Data Source: джерело даних, яке використовується для зберігання даних додатку;

2) Migrations: ці компоненти використовуються для оновлення схеми бази даних. Вони необхідні для забезпечення сумісності з змінами в моделі даних;

3) DbContext: цей компонент використовується для взаємодії з базою даних. Він надає методи для додавання, оновлення, видалення та отримання даних;

д) шар домену;

1) Entities: об'єкти, які використовуються для зберігання даних. Вони відповідають сутностям, визначеним у доменному компоненті;

2) Enums: перерахування фіксованих значень. Вони використовуються для обмеження можливих значень певних атрибутів.

3.3 Проєктування структури зберігання даних

Проєктування структури зберігання даних з точки зору BackEnd розробки для веб-системи житлових комплексів є критичним етапом, який забезпечує ефективність, масштабованість, безпеку та зручність доступу до даних. Враховуючи використання .NET і сервісів Azure, структура зберігання даних повинна бути спроєктована таким чином, щоб максимально використовувати можливості цих технологій. Основними аспектами є реляційні бази даних, кешування, індексація, резервне копіювання та відновлення, а також забезпечення безпеки даних.

Для зберігання основних даних системи, таких як інформація про мешканців, заявки на послуги, фінансові транзакції та інші структуровані дані, доцільно використовувати реляційну базу даних, таку як Azure SQL. Ця база даних забезпечує підтримку складних запитів, транзакцій, індексації та референтної цілісності. Важливо враховувати нормалізацію даних для мінімізації дублювання та забезпечення цілісності, але також слід збалансувати її з продуктивністю, щоб уникнути надмірно складних запитів [10].

Іншим ключовим аспектом є індексація, що дозволяє забезпечити швидкий доступу до даних. Важливо правильно спроєктувати індекси для таблиць бази даних, щоб оптимізувати запити. Це включає створення індексів на стовпцях, які часто використовуються в умовах where, join та order by. Azure SQL надає інструменти для аналізу продуктивності запитів і рекомендації щодо створення індексів. Використання композитних

індексів може також значно покращити продуктивність запитів, що виконують складні фільтрації.

Резервне копіювання та відновлення даних є критичними для забезпечення надійності та безперервності роботи системи. Azure SQL забезпечує автоматичне резервне копіювання даних і можливість відновлення до певного моменту часу, що дозволяє мінімізувати втрати даних у разі збоїв.

Розроблена база даних (див. рис. 3.2) знаходиться в третій нормальній формі, що підтверджує її нормалізацію [11, с. 102]. Нормована структура бази даних має такі переваги, як відсутність дублювання, простота зберігання та обробки даних. Це сприяє ефективному використанню пам’яті, особливо під час обробки великих обсягів даних.

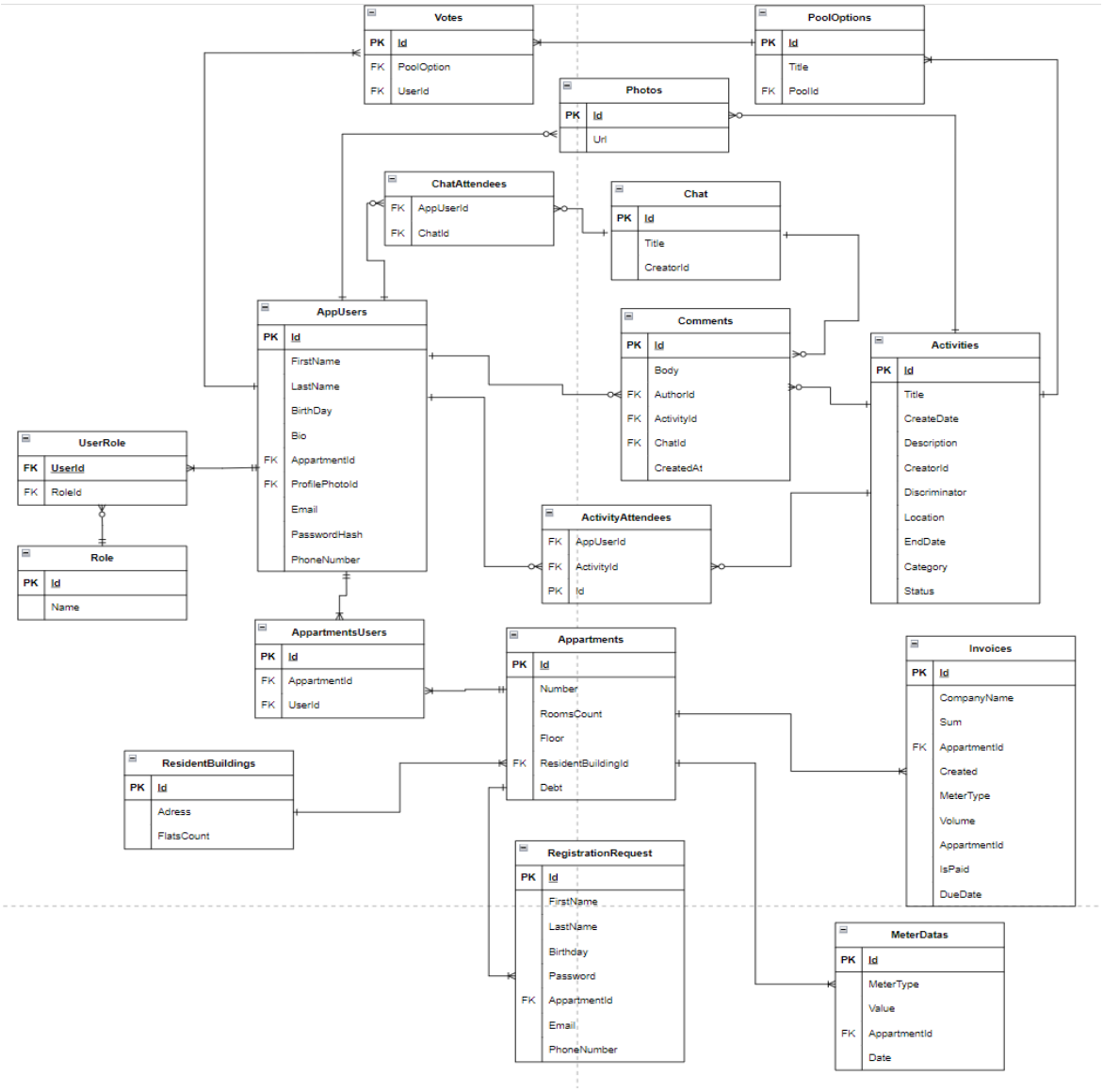


Рисунок 3.2 – Схема бази даних (виконаний самостійно)

3.4 Приклади найцікавіших алгоритмів та методів

Однією з найважливіших функцій веб-сервісу «Житловий комплекс» є отримання показників лічильників комунальних послуг з усіх квартир ЖК генерація звіту адміністратором, який пізніше надсилається до управління задля підтримки звітності та для повідомлення мешканців певних квартир про необхідність сплати заборгованості.

Задля реалізації даної функції системи було розроблено алгоритм, що збирає дані та робить необхідні підрахунки для формування звіту.

Основні кроки алгоритму:

а) отримати дані оплати комунальних послуг;

б) ініціалізувати змінні;

в) ітеративно пройти по отриманому списку квартир;

1) отримати дані лічильника за минулий та поточний місяць;

2) розрахувати суму оплати за комунальні послуги та загальну суму;

3) додати дані до звіту;

г) вивести звіт, загальну суму оплати за комунальні послуги та загальну суму оплати.

Схематично алгоритм формування щомісячного звіту з використання мешканцями ЖК комунальних послуг ЖЕК зображений на рисунку 3.3.

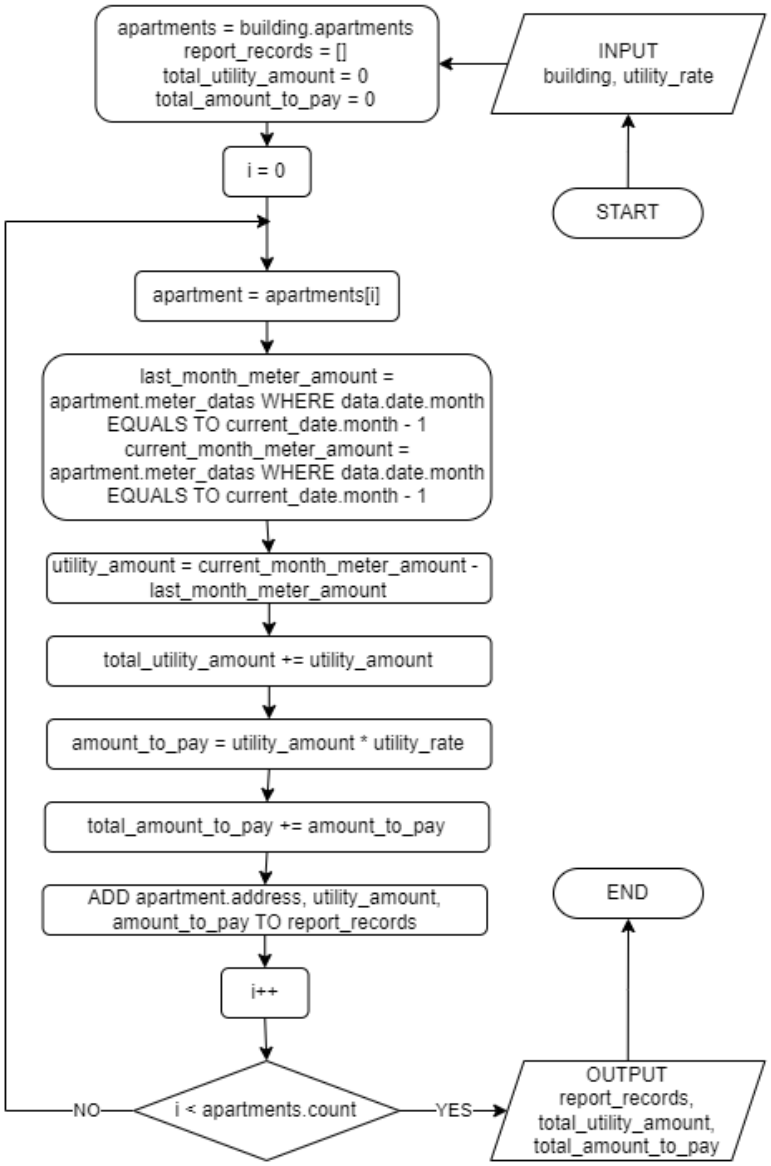


Рисунок 3.3 – Формування звіту з комунальних послуг (виконаний самостійно)

Додатково цей алгоритм дає адміністрації змогу генерувати звіти зі списком квартир з боргами з оплати комунальних послуг відповідно до даних показників лічильників.

Для втілення цієї функції був створений алгоритм, який збирає дані і проводить необхідні обчислення для створення звіту.

Детальні шаги виконання цього алгоритму зображені на блок-схемі нижче (див. рис. 3.4).

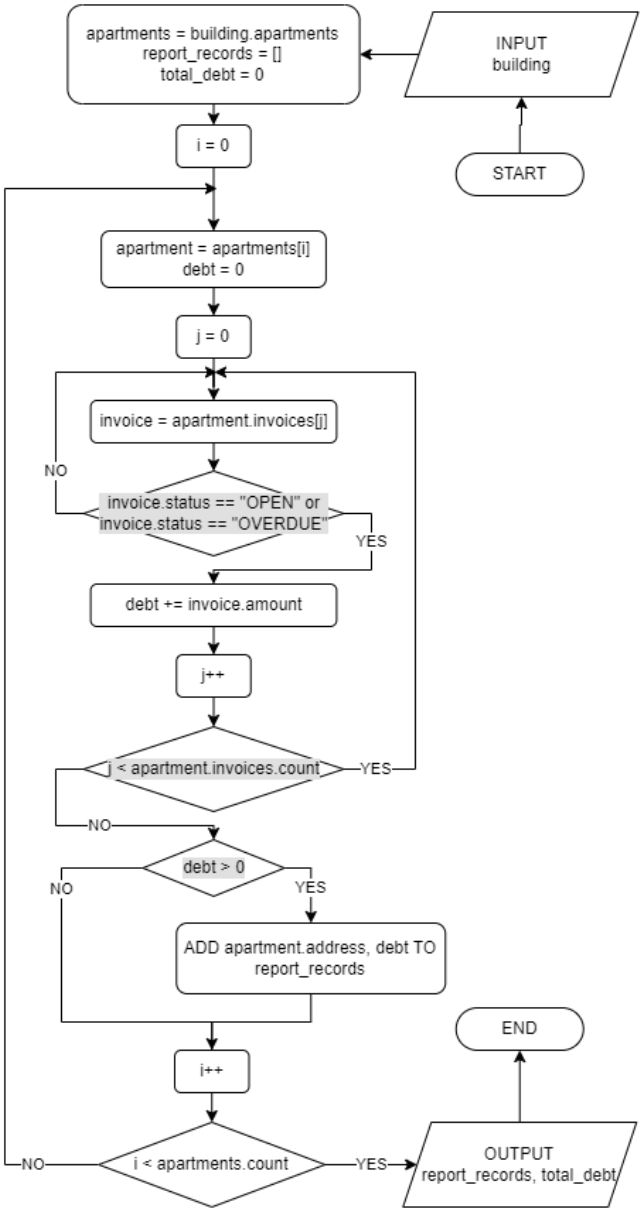


Рисунок 3.4 - Формування звіту з боржників (виконаний самостійно)

Алгоритми формування звітів є основою функції звітності веб-сервісу «Житловий комплекс» для адміністратора ЖК.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Опис процесу розгортання та моніторингу сервісу

Для розробки веб-сервісу для житлових комплексів були використані фреймворк .NET та база даних Azure SQL. Це рішення забезпечує високий рівень продуктивності, безпеки, масштабованості та надійності, необхідний для сучасних веб-додатків. Вибір .NET і Azure SQL обумовлений їх широкими можливостями для побудови комплексних та надійних систем.

.NET був обраний в якості основного фреймворку для BackEnd (див. рис. 4.1) розробки завдяки його багатофункціональності, надійності та широкому спектру інструментів для розробки. .NET забезпечує підтримку об'єктно-орієнтованого програмування, що дозволяє створювати добре структуровані та підтримувані програми. Основні компоненти нашої системи включають .NET Core для створення веб-API та служб, Entity Framework Core для взаємодії з базою даних та інші бібліотеки, що забезпечують необхідну функціональність. .NET Core є основою нашої BackEnd частини, оскільки з його допомогою можна створювати масштабовані, високопродуктивні веб-додатки. Використання контролерів та маршрутизації забезпечує чітку організацію обробки HTTP-запитів, дозволяючи легко додавати нові функції та розширювати існуючі. Middleware компоненти .NET Core використовуються для обробки запитів на різних етапах, таких як аутентифікація, авторизація, логування та обробка помилок, що забезпечує модульність та гнучкість системи.

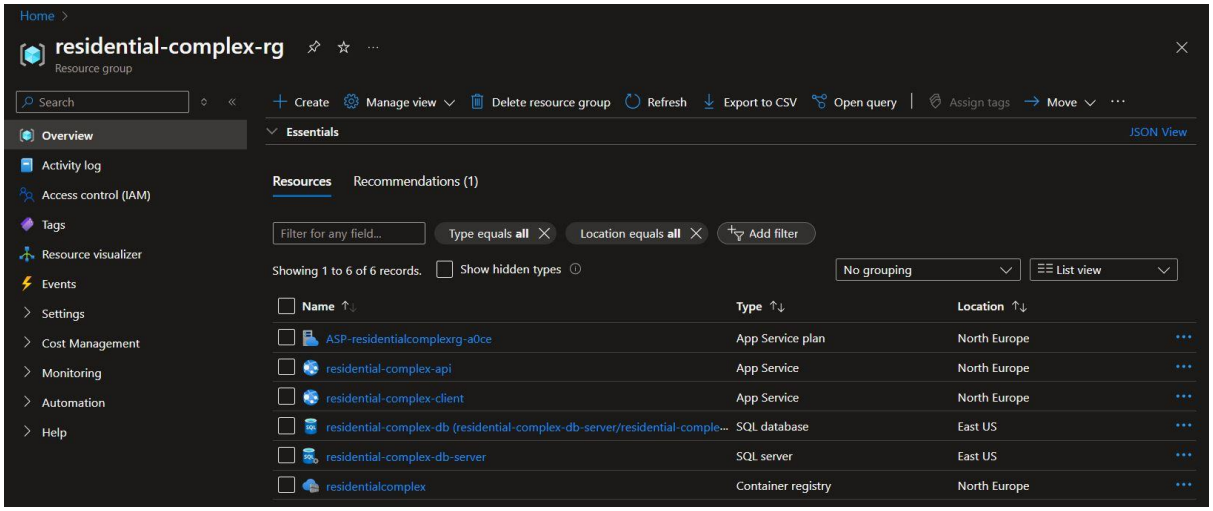


Рисунок 4.1 - BackEnd та інші ресурси веб-сервісу (виконаний самостійно)

Для взаємодії з базою даних (див. рис. 4.2) ми використовуємо Entity Framework Core, який являє собою ORM, що забезпечує зручний та ефективний доступ до даних. Цей фреймворк дозволяє працювати з базою даних на рівні об'єктів, що спрощує розробку та підтримку коду. Ми використовуємо модель першого підходу, що дозволяє визначати моделі даних як класи C# і автоматично створювати відповідні таблиці в базі даних [12]. Це забезпечує високу гнучкість і спрощує міграції бази даних при внесенні змін у моделі. Крім того, EF Core підтримує технологію LINQ, що дозволяє писати запити до бази даних у зручному синтаксисі C#, зменшуючи ймовірність помилок і підвищуючи рівень безпеки системи.

Azure SQL Database обрана як основне сховище даних завдяки її надійності, масштабованості та широким можливостям інтеграції з іншими сервісами Azure. Azure SQL забезпечує високий рівень доступності та відмово стійкості, автоматичне резервне копіювання, шифрування даних та можливість відновлення до певного моменту часу. Ми проєктували нашу базу даних з урахуванням нормалізації даних для забезпечення цілісності та уникнення дублювання. Водночас, для оптимізації продуктивності, була

враховувано необхідність створення індексів для часто використовуваних запитів. Індексація є ключовим аспектом для забезпечення швидкого доступу до даних. Також Azure SQL Database надає інструменти для аналізу продуктивності запитів і рекомендації щодо створення індексів. Використання композитних індексів може також значно покращити продуктивність запитів, в яких виконується багато фільтрацій.

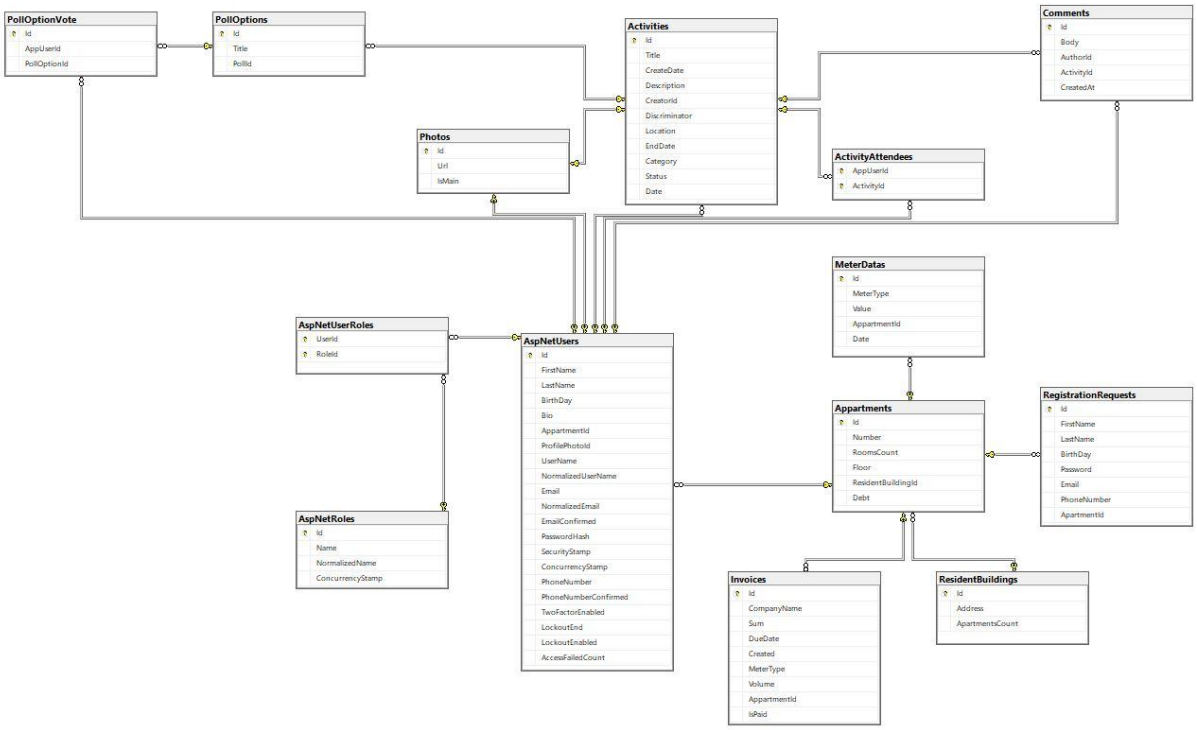


Рисунок 4.2 - Даталогічна модель БД (виконаний самостійно)

Для забезпечення високого рівня безпеки ми реалізували аутентифікацію та авторизацію за допомогою .NET Core Identity, який дозволяє централізовано керувати користувачами (див. рис. 4.3) та доступом до ресурсів, забезпечуючи надійну аутентифікацію, а також забезпечує управління користувачами, ролями та політиками доступу в рамках додатка. Це дозволяє гнучко налаштовувати права доступу до різних частин системи залежно від ролей користувачів. Крім того, для підвищення

безпеки облікових записів та персональних даних користувачів було впроваджено шифрування даних на рівні зберігання і передачі за допомогою TLS/SSL для захисту конфіденційної інформації.

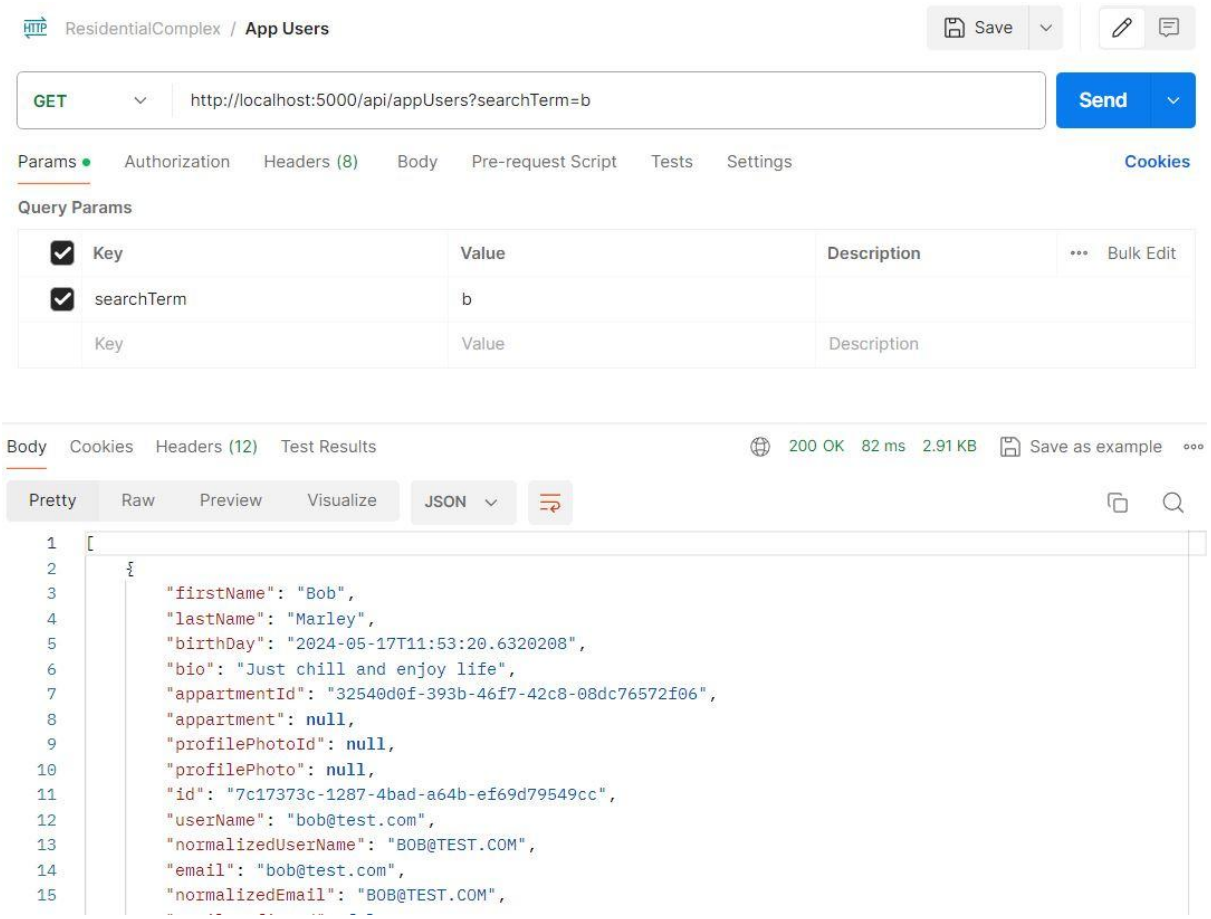


Рисунок 4.3 - Ендпойнт для отримання інформації щодо користувачів (виконаний самостійно)

Для обробки та зберігання великих обсягів логів та метрик в сервіси використовується Azure (див. рис. 4.4). Це дозволяє відстежувати продуктивність додатка, виявляти та усувати проблеми, аналізувати поведінку користувачів і отримувати інші корисні аналітичні дані. Використання цих інструментів допомагає забезпечити високу стабільність і надійність системи, своєчасно реагуючи на потенційні проблеми. Azure Monitor збирає дані з різних джерел, таких як журнали, метрики, події, і

забезпечує централізоване зберігання та аналіз цих даних. Application Insights забезпечує моніторинг продуктивності додатка, виявлення аномалій і діагностику помилок, що дозволяє швидко реагувати на будь-які проблеми і підвищувати загальну якість обслуговування.

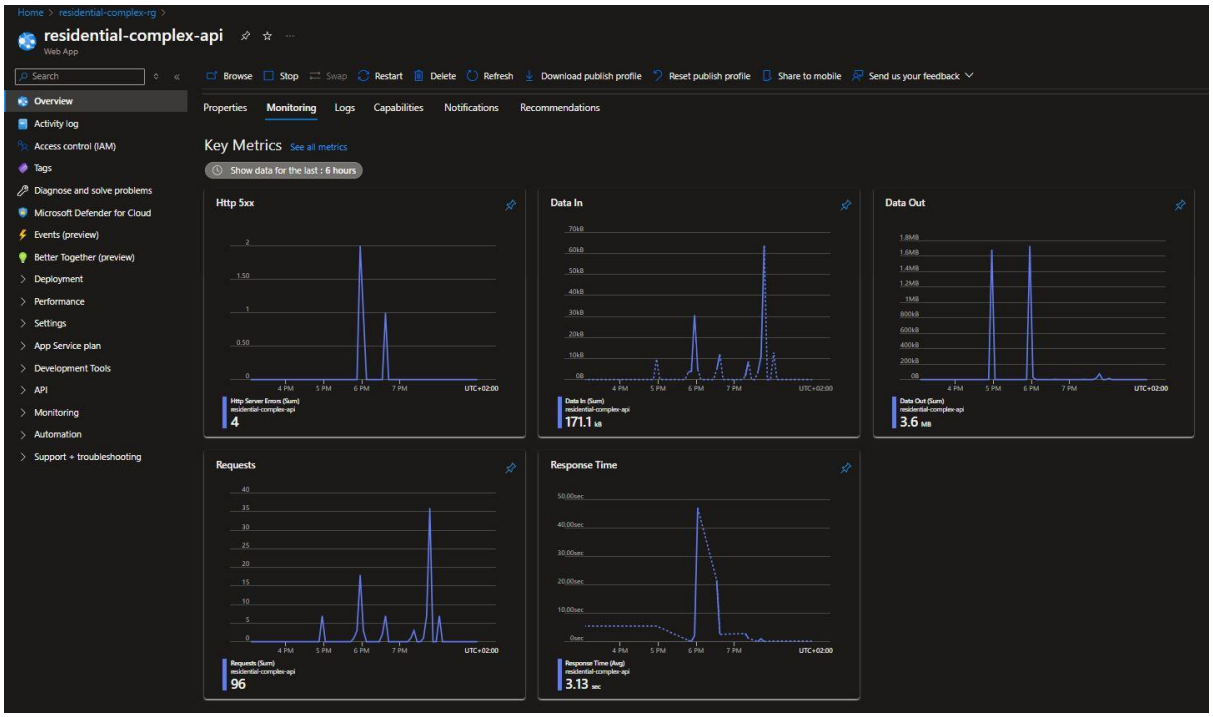


Рисунок 4.4 - Дашборд з метриками веб-сервісу (виконаний самостійно)

З точки зору розробленого функціоналу можна виділити модуль оголошень дозволяє адміністрації житлових комплексів та мешканцям публікувати та переглядати оголошення. Оголошення можуть бути коментовані та позначені як важливі. Цей модуль також використовує API інтерфейс (див. рис. 4.5) для обробки запитів та забезпечення взаємодії між FrontEnd та BackEnd частинами системи. Оголошення зберігаються в базі даних Azure SQL, що забезпечує їх доступність та надійність зберігання.

Модуль огошень також напряму пов’язаний з функціоналом голосування, який дозволяє проводити опитування щодо важливих питань серед мешканців, таких як ремонт, обслуговування будинку або вибір постачальників послуг. Ця функціональність дозволяє адміністрації легко організовувати голосування, а мешканцям — брати участь у прийнятті важливих рішень.

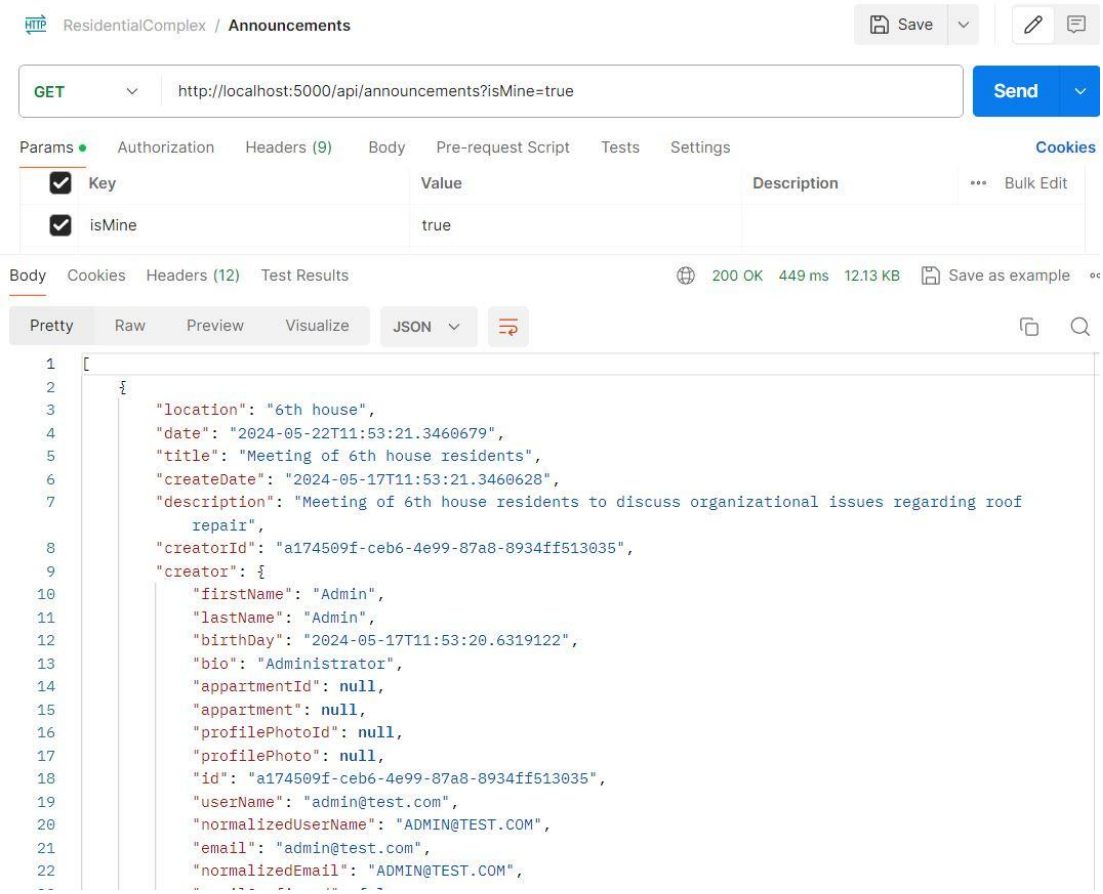


Рисунок 4.5 - Ендпойнт перегляду списку оголошень (виконаний самостійно)

В свою чергу внутрішні чати для мешканців (див. рис. 4.6), веб-сервіс дозволяю створювати та управляти чатами для обговорення питань між мешканцями певного будинку або під'їзду. Для реалізації цього

функціоналу використовується SignalR, який дозволяє забезпечити комунікацію між користувачами в режимі реального часу, дозволяючи їм швидко обмінюватися повідомленнями та обговорювати актуальні питання.

Коли користувач відкриває сторінку чату на клієнтській частині, відбувається підключення до SignalR хабу, що розгорнутий на сервері .NET. SignalR в даному випадку є центральним вузлом, який обробляє всі вхідні та вихідні повідомлення в чаті. Клієнтська частина, використовуючи бібліотеку SignalR JavaScript, ініціалізує з'єднання з хабом. Це з'єднання є постійним каналом зв'язку, який залишається відкритим протягом усього сеансу користувача на сторінці чату.

При надсиланні користувачем повідомлення, FrontEnd веб-сервісу використовує методи SignalR для відправлення цього повідомлення на сервер. Сервер, в свою чергу, отримує це повідомлення, обробляє його та зберігає в базі даних перед відправкою іншим користувачам. Після обробки це повідомлення транслюється всім підключеним клієнтам, включаючи відправника. Клієнтська частина на React, отримуючи повідомлення від хабу, оновлює інтерфейс користувача, додаючи нове повідомлення до списку чатів у реальному часі.

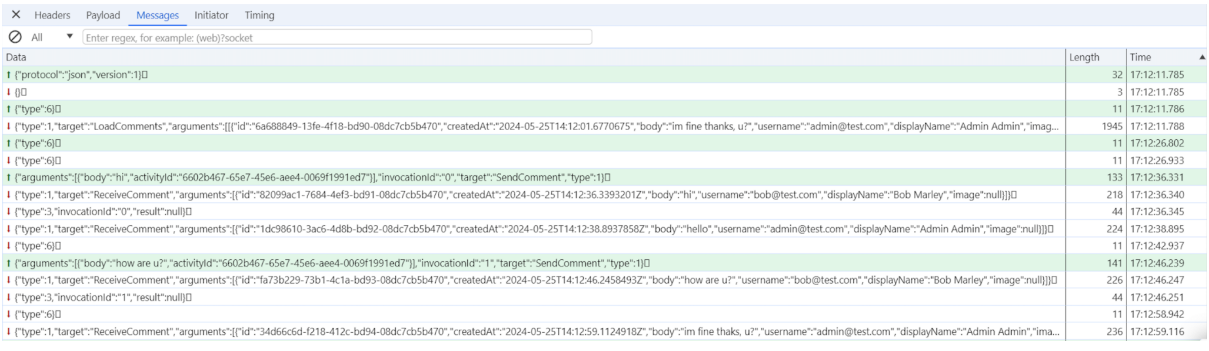


Рисунок 4.6 - Обмін повідомленнями для чатів між FrontEnd та BackEnd (виконаний самостійно)

Додатково, розроблений функціонал включає можливості для адміністрування системи, такі як управління користувачами та моніторинг активності. Адміністратори можуть переглядати статистику використання системи, управляти ролями користувачів та забезпечувати безпеку даних за допомогою різноманітних інструментів і сервісів Azure. Це дозволяє підтримувати високу стабільність і продуктивність системи, своєчасно реагувати на будь-які потенційні проблеми та забезпечувати безперебійне функціонування веб-системи для житлових комплексів.

Таким чином, впровадження цих функціональних модулів забезпечує комплексний підхід до управління житловими комплексами, дозволяючи користувачам взаємодіяти з системою зручно і безпечно. Прийняті програмні рішення з точки зору BackEnd розробки включають використання .NET для створення високопродуктивних веб-API та служб, Entity Framework Core для ефективної роботи з базою даних, Azure SQL для надійного та масштабованого зберігання даних, а також комплексний підхід до забезпечення безпеки, продуктивності та надійності системи за допомогою сервісів Azure.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Тестування розробленого програмного забезпечення

Тестування веб-сервісу для житлових комплексів з точки зору BackEnd розробки є багатогранним процесом, що забезпечує функціональність, безпеку, продуктивність, масштабованість та надійність системи. Використання .NET та сервісів Azure дозволяє реалізувати повний спектр тестувальних підходів, включаючи модульне тестування, інтеграційне тестування, тестування продуктивності, безпеки, баз даних та безперервну інтеграцію і розгортання.

Модульне тестування є основою для перевірки окремих компонентів коду. В контексті .NET, за допомогою фреймворку XUnit, можна ретельно тестувати функції, методи та класи. Наприклад, модульні тести можуть перевіряти правильність обробки користувацьких запитів (див. рис. 5.1), логіку бізнес-процесів та коректність обчислень. Використання модульні сприяє написанню тестів до реалізації функціоналу, що дозволяє виявляти помилки на ранніх стадіях розробки та забезпечувати високу якість коду. Цей підхід допомагає не лише у виявленні помилок, але й у підтримці чіткого та зрозумілого коду.

```
using Xunit;
using Moq;
using MediatR;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;
using API.Controllers;
using Application.Announcements;
using Application.Core;
using Domain;
using System;
using System.Collections.Generic;
using System.Threading;
public class AnnouncementsControllerTests
{
    private readonly Mock<IMediator> _mediatorMock;
    private readonly AnnouncementsController _controller;

    public AnnouncementsControllerTests()
    {
        _mediatorMock = new Mock<IMediator>();
        _controller = new AnnouncementsController { Mediator = _mediatorMock.Object };
    }

    [Fact]
    public async Task GetAnnouncements_ReturnsPagedResult()
    {
        var param = new AnnouncementParams();
        var announcements = new PagedList<Announcement>(new List<Announcement>(), 0, 1, 10);
        var result = Result<PagedList<Announcement>>.Success(announcements);

        _mediatorMock.Setup(m => m.Send(It.IsAny<List.Query>(), It.IsAny<CancellationToken>()))
            .ReturnsAsync(result);

        var response = await _controller.GetAnnouncements(param);

        var okResult = Assert.IsType<OkObjectResult>(response);
        Assert.Equal(200, okResult.StatusCode);
        Assert.Equal(result.Value, okResult.Value);
    }
}
```

Рисунок 5.1 - Приклад модульного тесту для ендпойнту оголошень (виконаний самостійно)

Інтеграційне тестування перевіряє взаємодію між різними компонентами системи. У нашому випадку такими компонентами є різні сервіси, база даних, зовнішні API та інші залежності. Використовуючи ті ж фреймворки, що і для модульного тестування, інтеграційні тести забезпечують перевірку взаємодії між різними частинами системи, включаючи перевірку відповідності даних між різними модулями.

Тестування продуктивності є не менш критичним для забезпечення швидкодії та стабільності веб-сервісу під навантаженням. Спеціальні

інструменти дозволяють моделювати різні сценарії навантаження, вимірювати час відповіді на запити, пропускну здатність системи та використання ресурсів. Тестування продуктивності включає перевірку часу завантаження сторінок, швидкості обробки запитів до бази даних, часу відповіді сервера та роботи під великим навантаженням. Наприклад, можна симулювати одночасне замовлення послуг великою кількістю користувачів та перевірити, як система справляється з таким навантаженням. Це допомагає виявити вузькі місця в архітектурі, такі як неефективні запити або обмеження ресурсів, та оптимізувати їх для забезпечення стабільної роботи навіть при пікових навантаженнях.

Тестування безпеки є важливим для захисту даних користувачів та запобігання атакам на систему, оскільки такий тип тестування допомагає виявити та усунути вразливості, такі як SQL-ін'єкції, XSS, CSRF та інші загрози [13]. Важливо також перевірити правильність налаштувань безпеки в Azure, включаючи мережеві політики, шифрування даних, управління доступом та використання Azure Key Vault для зберігання секретів і ключів шифрування. Наприклад, тести повинні перевіряти, чи всі API-запити належним чином аутентифіковані та авторизовані, чи дані передаються по захищених каналах та чи належним чином обробляються введені користувачами дані, щоб уникнути можливих вразливостей.

Тестування баз даних включає перевірку коректності роботи запитів, цілісності даних, продуктивності баз даних та процесів резервного копіювання та відновлення. Використовуючи Azure SQL, можна проводити тести, що перевіряють продуктивність запитів, індексацію, транзакції та ізоляцію. Наприклад, можна симулювати виконання складних запитів під навантаженням та перевіряти, як вони впливають на продуктивність системи. Важливо також тестувати процеси резервного копіювання та

відновлення, щоб переконатися, що дані можуть бути швидко та надійно відновлені у разі збоїв або втрат.

Документування результатів тестування є важливим аспектом процесу, що забезпечує прозорість та повторюваність. Використовуючи Azure, можна зберігати результати тестів, логування помилок та звіти про продуктивність у централізованому місці, що дозволяє легко відслідковувати та аналізувати історію тестувань. Це також допомагає виявити тенденції та проблеми, що повторюються, та приймати обґрунтовані рішення щодо оптимізації та покращення системи.

Загалом, тестування BackEnd частини веб-сервісу для житлових комплексів є складним і багатоетапним процесом, що включає різні види тестування для забезпечення високої якості, безпеки, продуктивності та надійності системи. Використання .NET та сервісів Azure дозволяє реалізувати комплексний підхід до тестування, автоматизуючи більшість процесів та забезпечуючи безперервну інтеграцію і розгортання, що дозволяє швидко виявляти та усувати проблеми, підтримуючи високу якість продукту на всіх етапах його життєвого циклу. Тестування повинно бути ретельним, систематичним і регулярним, щоб забезпечити надійну роботу веб-сервісу в реальних умовах, задовольняючи потреби користувачів та забезпечуючи їм безпечний та ефективний доступ до необхідної функціональності.

ВИСНОВКИ

Підсумовуючи розробку веб-сервісу для житлових комплексів з точки зору BackEnd, важливо підкреслити декілька ключових аспектів, що сприяли успішній реалізації проєкту. Перш за все, використання .NET як основної технології для BackEnd забезпечило високу продуктивність і надійність системи. Завдяки багатофункціональному середовищу розробки та потужному набору бібліотек і фреймворків, .NET дозволив ефективно реалізувати складну бізнес-логіку.

Були розроблені API ендпойнти, що дозволяють адміністрації створювати оголошення, опитування, замовлення послуг та чати, в яких мешканці можуть обговорювати важливі питання розвитку житлових комплексів. Застосування шаблонів проєктування відіграло важливу роль у структуризації та підтримуваності коду. Mediator спростив обробку запитів і взаємодію між компонентами, забезпечуючи модульність та зменшення залежностей. Middleware у середовищі .NET Core дозволив легко впровадити кросс-різні аспекти, такі як автентифікація, авторизація та обробка помилок, що підвищило безпеку та стабільність додатка. Unit of Work у контексті Entity Framework забезпечив атомарність і узгодженість транзакцій, мінімізуючи ризик виникнення неконсистентних даних у базі даних.

Загалом, розробка BackEnd для веб-сервісу для житлових комплексів показала важливість використання сучасних технологій та підходів у розробці. Поєднання .NET та Azure SQL разом забезпечило створення надійної, продуктивної та масштабованої системи. Проєктування системи з урахуванням принципів чистої архітектури та використання перевірених шаблонів проєктування сприяло досягненню високого рівня підтримуваності та гнучкості, що є ключовим фактором для успішної експлуатації та подальшого розвитку програмного продукту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Condo Control Central 2023 Review, Pros & Cons. URL: https://softwareconnect.com/property-management/condo-control-central/ (дата звернення: 09.05.2024)

2. BuildingLink Software review. URL: https://www.softwareadvice.com/property/buildinglink-profile/ (дата звернення: 09.05.2024)

3. Yardi Voyager Property Management. URL: https://softwareconnect.com/property-management/yardi-voyager/ (дата звернення: 09.05.2024)

4. Functional vs Non Functional Requirements. URL: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/ (дата звернення: 09.05.2024)

5. Azure documentation. URL: https://learn.microsoft.com/en-gb/azure/ (дата звернення: 09.05.2024)

6. Common web applications architectures. URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (дата звернення: 09.05.2024)

7. Thilmany C. .NET Patterns: Architecture, Design, and Process. Addison-Wesley Professional, 2003. 448 с.

8. SOLID: The First 5 Principles of Object Oriented Design. URL: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design (дата звернення: 09.05.2024)

9. Choudhry A., Premchand A. Real Time Apps Using SignalR. *International Journal of Computer Trends and Technology*. 2014. Т. 15, № 3. С. 92–96. URL: https://doi.org/10.14445/22312803/ijctt-v15p121(дата звернення: 26.05.2024)

10. Ward B. What Is Azure SQL?. *Azure SQL Revealed*. Berkeley, CA, 2020. С. 39–80. URL: https://doi.org/10.1007/978-1-4842-5931-3_2(дата звернення: 09.05.2024)

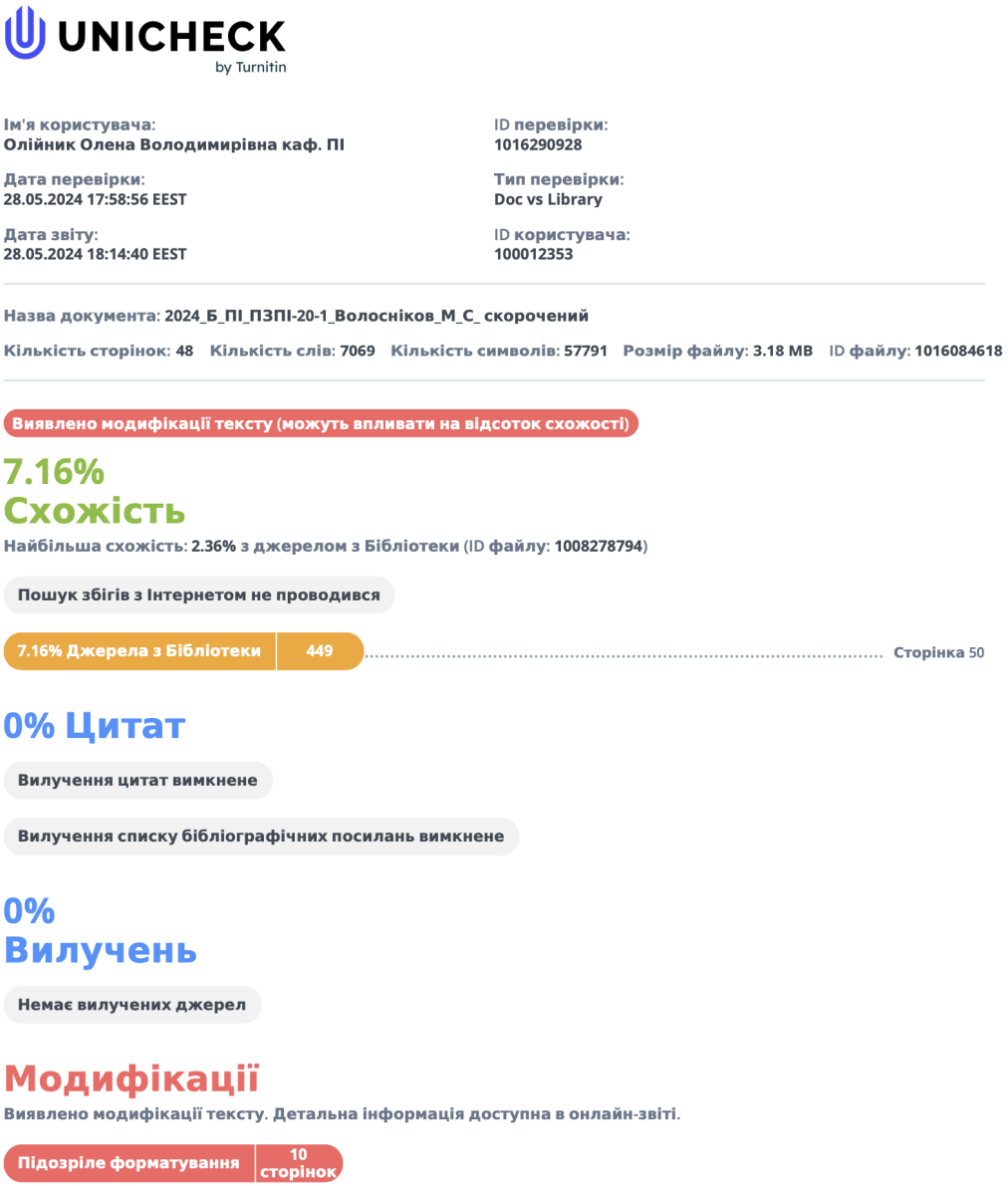
11. Köhler H., Link S. SQL schema design: foundations, normal forms, and normalization. *Information Systems*. 2018. Т. 76. С. 88–113. URL: https://doi.org/10.1016/j.is.2018.04.001 (дата звернення: 09.05.2024)

12. Gorman B. L. Introduction to Entity Framework. *Practical Entity Framework*. Berkeley, CA, 2020. С. 3–23. URL: https://doi.org/10.1007/978-1-4842-6044-9_1(дата звернення: 09.05.2024)

13. Wenz C. ASP. NET Core Security. Manning Publications Co. LLC, 2022.

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016290928

Дата перевірки:
28.05.2024 17:58:56 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
28.05.2024 18:14:40 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІ-20-1_Волосніков_М_С_ скорочений

Кількість сторінок: 48 Кількість слів: 7069 Кількість символів: 57791 Розмір файлу: 3.18 MB ID файлу: 1016084618

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

7.16%
Схожість

Найбільша схожість: 2.36% з джерелом з Бібліотеки (ID файлу: 1008278794)

Пошук збігів з Інтернетом не проводився

7.16% Джерела з Бібліотеки 449 ..... Сторінка 50

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування 10 сторінок

ДОДАТОК Б

Слайди презентації

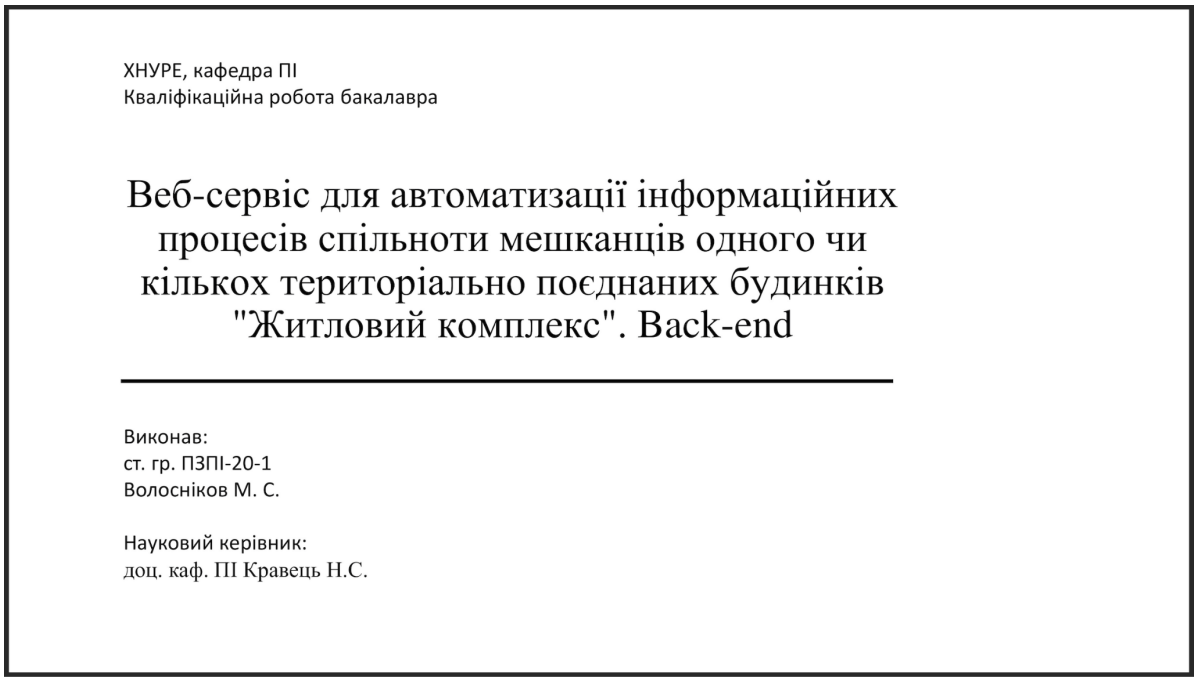


Рисунок Б.1 – Слайд 1 (рисунок виконаний самостійно)

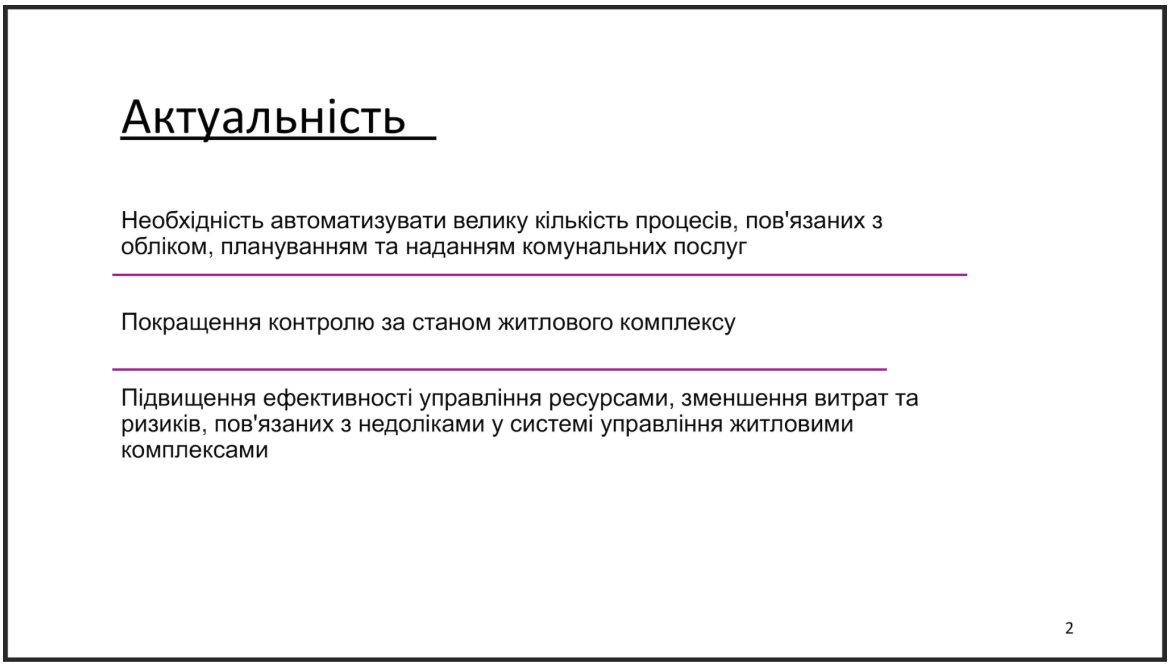


Рисунок Б.2 – Слайд 2 (рисунок виконаний самостійно)

Постановка задачі

Сервіс повинен відповідати таким вимогам:

- рєстрація та відстеження заявок на побутові послуги;
- модуль оголошень, де буде публікуватися інформація про плановані роботи, графіки відключень, загальні збори, тощо;
- функція проведення голосувань серед мешканців щодо важливих питань будинку або житлового комплексу;
- функціонал створення групових чатів для обговорення питань, пов'язаних з конкретним будинком або комплексом;;
- можливість перегляду та оплати рахунків за комунальні послуги через онлайн-платформу.

3

Рисунок Б.3 – Слайд 3 (рисунок виконаний самостійно)

Аналіз предметної галузі

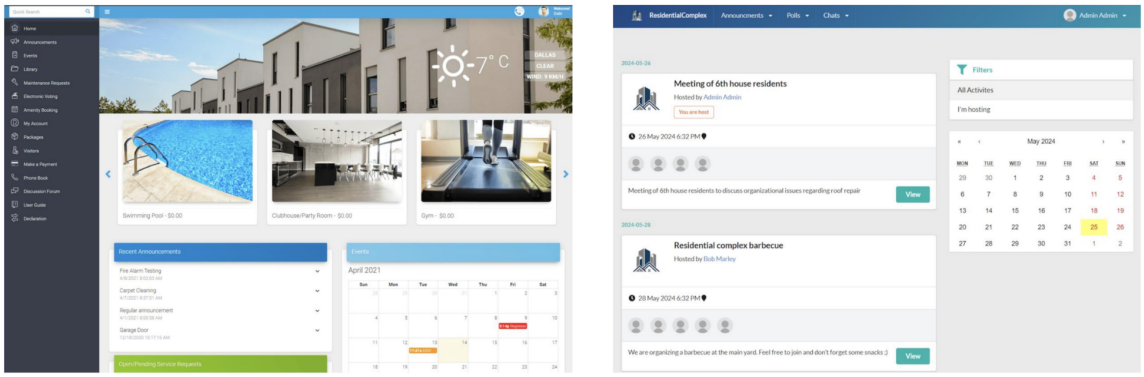
Основні тенденції в розвитку житлових комплексів:

- стрімкий розвиток технологій у галузі;
- зростання екологічних вимог до житлово-комунального господарства;
- збільшення кількості міського населення та концентрація людей у великих містах;
- питання сталого розвитку, в яке входить забезпечення економічної ефективності, соціальної справедливості та екологічної стійкості;
- нестабільність в управлінні, висока вартість та недостатня доступність житлового фонду для ряду соціальних груп.
- Необхідніть автоматизувати багато рутинних операцій управління, таких як облік наявних ресурсів, моніторинг стану об'єктів інфраструктури, планування обслуговування

4

Рисунок Б.4 – Слайд 4 (рисунок виконаний самостійно)

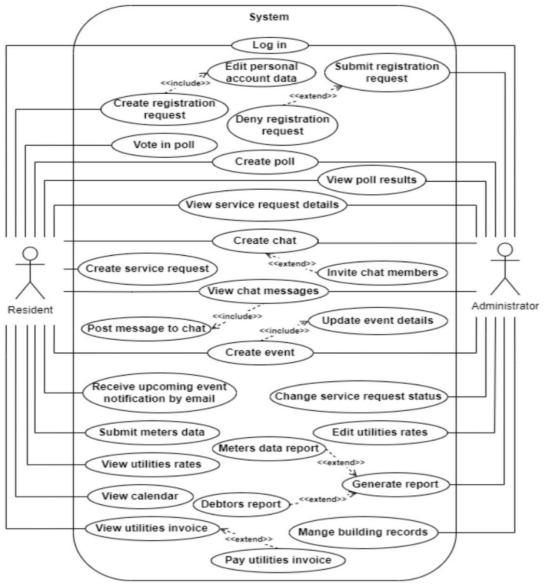
Огляд конкурентів



5

Рисунок Б.5 – Слайд 5 (рисунок виконаний самостійно)

Use Case Diagram



6

Рисунок Б.6 – Слайд 6 (рисунок виконаний самостійно)

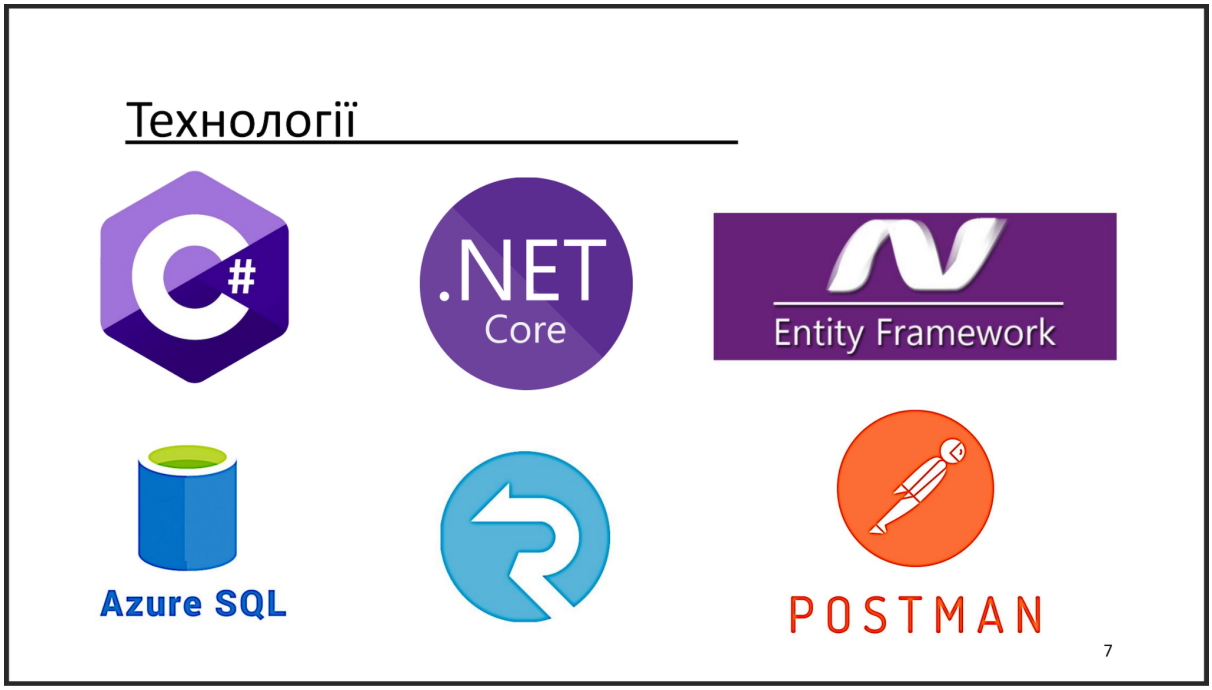
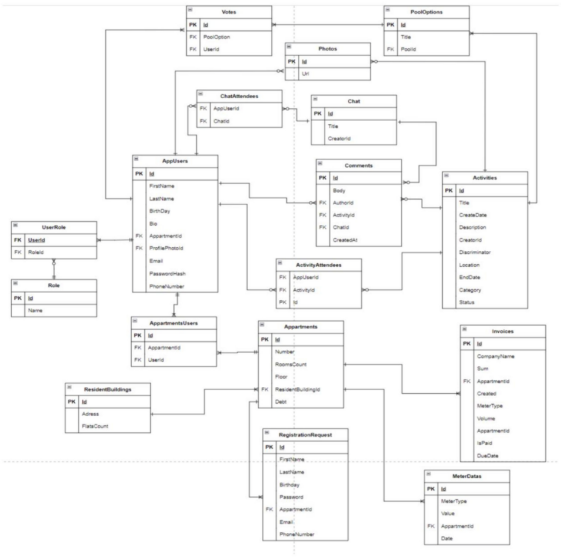


Рисунок Б.7 – Слайд 7 (рисунок виконаний самостійно)

Рисунок Б.8 – Слайд 8 (рисунок виконаний самостійно)

Структура зберігання даних

Для зберігання основних даних системи, таких як інформація про мешканців, заявки на послуги, фінансові транзакції та інші структуровані дані, доцільно використовувати реляційну базу даних, таку як Azure SQL. Ця база даних забезпечує підтримку складних запитів, транзакцій, індексації та референтної цілісності.

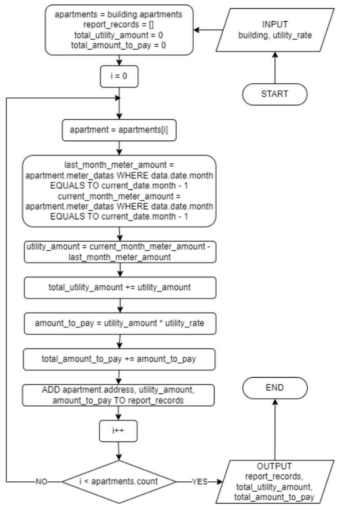


9

Рисунок Б.9 – Слайд 9 (рисунок виконаний самостійно)

Приклад найцікавішого алгоритму

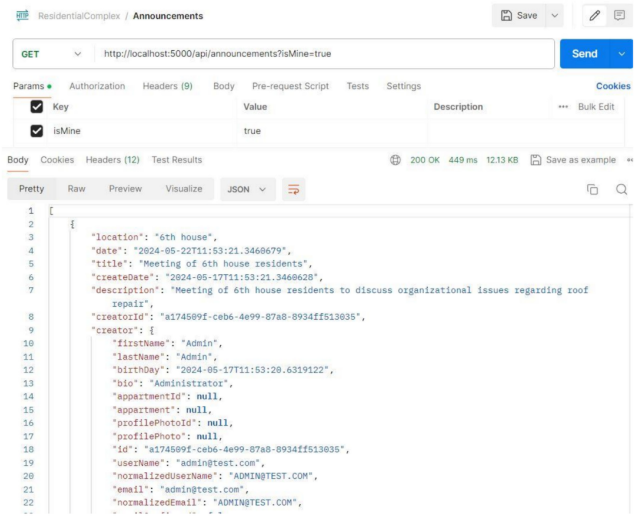
Основні кроки алгоритму:
а) Прочитати дані;
б) ініціалізувати змінні;
в) пройтися по квартирах;
1) отримати дані лічильника за минулий та поточний місяць;
2) розрахувати суму оплати за комунальні послуги та загальну суму;
3) додати дані до звіту;
г) вивести звіт, загальну суму оплати за комунальні послуги та загальну суму оплати.



10

Рисунок Б.10 – Слайд 10 (рисунок виконаний самостійно)

Приклад API ендпойнту

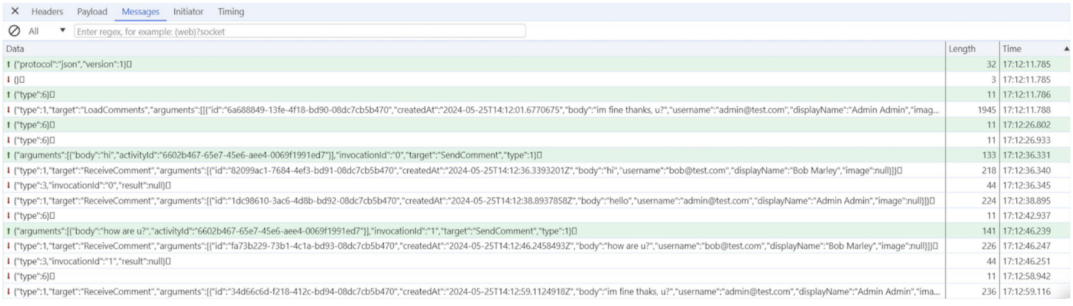


11

Рисунок Б.11 – Слайд 11 (рисунок виконаний самостійно)

Реалізація роботи чатів

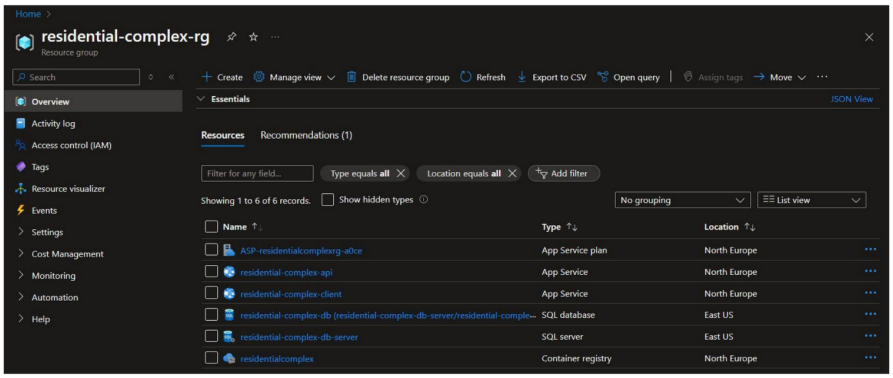
Для реалізації функціоналу чатів використовується SignalR, який дозволяє забезпечити комунікацію між користувачами в режимі реального часу, дозволяючи їм швидко обмінюватися повідомленнями та обговорювати актуальні питання.



12

Рисунок Б.12 – Слайд 12 (рисунок виконаний самостійно)

Розгортання BackEnd на Azure



13

Рисунок Б.13 – Слайд 13 (рисунок виконаний самостійно)

Висновки

В результаті роботи вдалося:

- знизити час і витрати на вирішення побутових питань та підвищило загальний рівень обслуговування;
- отримати зручний доступ до нових сервісів, таких як замовлення послуг, спілкування з адміністрацією та іншими мешканцями;
- полегшити обмін інформацією, забезпечити проведення опитуванням про важливі питання;
- підвищення рівня безпеки та захисту персональних даних;

14

Рисунок Б.14 – Слайд 14 (рисунок виконаний самостійно)

ДОДАТОК В

Специфікація вимог до програмного продукту

Специфікація ПЗ

Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук

Кафедра програмної інженерії

СПЕЦИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

веб-сервіс для автоматизації інформаційних процесів спільноти мшканців одного чи кількох територіально поєднаних будинків "Житловий комплекс". BackEnd

Студент гр. ПЗПІ-20-1 _____ Волосніков М.С.

Студент гр. ПЗПІ-20-1 _____ Жеребний В.В.

Студент гр. ПЗПІ-20-1 _____ Янов Д.І.

Харків

2024 р.

ЗМІСТ

1 ВСТУП

1.1 Огляд продукту

Програмний продукт є веб-сервісом для житлових комплексів, який забезпечує інтеграцію між мешканцями та адміністрацією житлових комплексів. Він надає можливості управління запитами на обслуговування, сповіщення мешканців про події та новини, а також здійснення онлайн-платежів. BackEnd частина реалізована на платформі .NET, FrontEnd – на React, база даних використовує Azure SQL, процеси CI/CD налаштовані за допомогою GitHub Actions, і все розгортається в Docker. Система створена з метою оптимізації взаємодії між мешканцями та адміністрацією, забезпечення прозорості та ефективності управління житловими комплексами, а також підвищення рівня задоволеності мешканців через покращену комунікацію та швидке вирішення проблем.

1.2 Мета

Метою розробки є створення надійного та безпечного веб-сервісу, який полегшить управління житловими комплексами, покращить комунікацію між мешканцями та адміністрацією, забезпечить швидке обслуговування запитів та підтримку різних видів онлайн-платежів. Програмний продукт повинен забезпечити ефективний процес управління запитами на обслуговування, знизити адміністративні витрати за рахунок автоматизації рутинних завдань, та забезпечити прозорість процесів для мешканців і адміністрації. Основна мета полягає у створенні інтуїтивно зрозумілого інтерфейсу для мешканців, що дозволить їм легко подавати запити, відстежувати їх статус та здійснювати платежі, а також надання адміністрації інструментів для ефективного управління житловим комплексом.

1.3 Межі

Продукт розрахований на використання в житлових комплексах і не призначений для управління комерційними або промисловими об'єктами. Він включає обмеження по кількості одночасних користувачів, розмірів бази даних і інтеграції з іншими системами. Продукт буде розроблений на основі веб-технологій і не включає розробку мобільних додатків. Межі також включають обмеження по функціоналу, зосереджуючись на ключових аспектах управління житловими комплексами, таких як управління запитами на обслуговування, сповіщення та онлайн-платежі. Не передбачається інтеграція з широким спектром зовнішніх систем, окрім необхідних платіжних шлюзів та сервісів сповіщень.

1.4 Посилання

Цей веб-сервіс для житлових комплексів призначений для спрощення та автоматизації управління, забезпечуючи мешканцям зручний доступ до необхідної інформації та сервісів. Впровадження даного продукту дозволить знизити кількість ручної роботи, оптимізувати комунікацію між адміністраторами та мешканцями, а також покращити загальну ефективність управління житловим комплексом. Система аутентифікації та авторизації забезпечить безпечний доступ та контроль над даними, що є критично важливим для забезпечення конфіденційності інформації.

Цей веб-сервіс допоможе управляючим компаніям раціоналізувати процеси, що дозволить більш ефективно використовувати ресурси та досягати поставлених цілей. Інтеграція з Azure SQL забезпечить високу надійність та продуктивність зберігання даних, а CI/CD процеси, налаштовані за допомогою GitHub Actions, дозволять швидко і безпечно впроваджувати нові функціональні можливості та оновлення.

Використання Docker дозволить легко масштабувати систему та забезпечити її стабільну роботу у різних середовищах.

Загалом, впровадження цього веб-сервісу з back-end на .NET, front-end на React, та базою даних Azure SQL дозволить житловим комплексам значно покращити управління, підвищити прозорість процесів та задоволеність мешканців. Веб-сервіс також забезпечить можливість для створення детальних звітів та аналізу даних, що допоможе приймати більш обґрунтовані управлінські рішення та розробляти ефективні стратегії для покращення обслуговування мешканців.

1.5 Означення та абревіатури

— CI/CD – безперервна інтеграція та доставка (Continuous Integration/Continuous Delivery);

— Azure SQL – керована реляційна база даних від Microsoft Azure;

— Docker – платформа для контейнеризації додатків;

— .NET – програмна платформа від Microsoft для розробки додатків;

— React – бібліотека JavaScript для створення користувацьких інтерфейсів;

— API – програмний інтерфейс додатків (Application Programming Interface);

— HTTPS – протокол захищеного перенесення гіпертексту (HyperText Transfer Protocol Secure);

— MFA – багатофакторна аутентифікація (Multi-Factor Authentication);

2 ЗАГАЛЬНИЙ ОПИС

2.1 Перспективи продукту

Веб-сервіс для житлових комплексів має на меті стати невід’ємною частиною сучасного управління житловими приміщеннями, полегшуючи взаємодію між адміністрацією та мешканцями. Очікується, що він буде постійно оновлюватися та розширюватися, інтегруючись з новими технологіями та пристроями, такими як IoT (Internet of Things). Майбутні версії можуть включати підтримку автоматизації управління енергоспоживанням, інтеграцію з розумними пристроями для підвищення комфорту мешканців та розширені функції аналітики для більш ефективного управління ресурсами. Перспективи продукту також включають можливість створення мобільних додатків для зручнішого доступу мешканців до послуг, розширення функціоналу за рахунок інтеграції з іншими платформами управління житловими комплексами та покращення користувацького досвіду через персоналізовані сповіщення та рекомендації.

2.2 Функції продукту

Продукт включає наступні функції: управління запитами на обслуговування (створення, перегляд, оновлення та закриття запитів), сповіщення мешканців про події та новини (створення та розсилка сповіщень через різні канали, такі як email та push-повідомлення), підтримка онлайн-платежів (можливість здійснення платежів через інтегровані платіжні системи), аналітика та звітність (генерація звітів про діяльність та взаємодію мешканців з адміністрацією). Крім основних функцій, продукт включає модулі для управління профілями користувачів, налаштування прав доступу для адміністрації, ведення історії взаємодій та інтеграції з зовнішніми сервісами для сповіщень та платежів. Функціонал

також включає можливість адміністрування та управління різними житловими комплексами з єдиного інтерфейсу, забезпечуючи зручний та ефективний контроль за всіма об’єктами в одному місці.

2.3 Характеристики користувачів

Користувачі продукту включають адміністрацію житлових комплексів, мешканців та технічний персонал. Адміністрація матиме доступ до функцій управління та звітності, мешканці – до функцій створення запитів та здійснення платежів, технічний персонал – до функцій обслуговування запитів. Адміністрація використовуватиме систему для відстеження запитів мешканців, управління обслуговуючим персоналом та генерації звітів для аналізу ефективності роботи. Мешканці використовуватимуть веб-сервіс для подання запитів на обслуговування, відстеження статусу своїх запитів, отримання сповіщень про новини та події, а також здійснення онлайн-платежів. Технічний персонал буде використовувати систему для отримання інформації про запити, призначення завдань та відстеження виконання робіт.

2.4 Загальні обмеження

Обмеження включають максимальну кількість одночасних користувачів, обсяг даних, що зберігається в базі даних, обмеження по пропускній здатності мережі, і обмеження по інтеграції з зовнішніми системами. Продукт має обмеження по обсягу даних, що зберігаються в Azure SQL, враховуючи необхідність збереження історичних даних для аналітики та звітності. Обмеження також стосуються можливості одночасної роботи великої кількості користувачів, що може вимагати додаткової оптимізації і масштабування для забезпечення стабільної роботи під час пікових навантажень. Крім того, система обмежена у можливостях

інтеграції з деякими зовнішніми сервісами через технічні або безпекові причини.

2.5 Припущення й залежності

Припускається, що користувачі мають доступ до сучасних веб-браузерів і стабільного інтернет-з'єднання. Залежність від хмарної платформи Azure передбачає необхідність наявності відповідного підписки та підтримки з боку Azure для забезпечення безперебійної роботи бази даних. Припускається також, що користувачі мають базові навички роботи з комп'ютером та інтернетом, що дозволить їм без проблем користуватися веб-сервісом. Залежність від Docker і GitHub Actions вимагає, щоб інфраструктура підтримувала контейнери та безперервну інтеграцію і доставку, що забезпечить швидке та надійне розгортання оновлень та виправлень.

3 КОНКРЕТНІ ВИМОГИ

3.1 Вимоги до зовнішніх інтерфейсів

3.1.1 Інтерфейс користувача

Інтерфейс користувача повинен бути розроблений на основі React, забезпечуючи зручний і інтуїтивно зрозумілий досвід користування. Користувацький інтерфейс повинен включати форми для створення запитів, інформаційні панелі для перегляду статусу запитів, функціонал для здійснення онлайн-платежів, та налаштування для керування профілем користувача. Інтерфейс має бути адаптивним, підтримувати різні розміри екранів та забезпечувати однаковий рівень зручності як на настільних комп’ютерах, так і на мобільних пристроях. Користувачам повинні бути доступні інтуїтивно зрозумілі меню та навігаційні елементи, що полегшують пошук потрібної інформації та виконання основних завдань. Інтерфейс також повинен включати функції доступності для користувачів з обмеженими можливостями, забезпечуючи доступність для широкого кола користувачів.

3.1.2 Апаратний інтерфейс

Веб-сервіс не вимагає спеціалізованого апаратного забезпечення і може працювати на будь-якому сучасному серверному обладнанні або віртуальній машині, що підтримує Docker. Система повинна бути оптимізована для роботи на хмарних платформах, таких як Azure, що забезпечить масштабованість та гнучкість у розгортанні. Необхідно забезпечити підтримку резервного копіювання та відновлення даних, що дозволить зберігати цілісність даних у випадку апаратних збоїв або інших технічних проблем. Апаратний інтерфейс також має забезпечувати ефективну взаємодію з мережею для забезпечення швидкої передачі даних між сервером та користувачами.

3.1.3 Програмний інтерфейс

Програмний інтерфейс включає RESTful API для взаємодії між FrontEnd та BackEnd, а також інтеграційні точки для взаємодії з зовнішніми сервісами, такими як платіжні системи та сервіси сповіщень. API повинен підтримувати основні операції CRUD (створення, читання, оновлення, видалення) для управління даними, а також забезпечувати безпечну аутентифікацію та авторизацію користувачів. Програмний інтерфейс має бути добре задокументований, що дозволить розробникам легко інтегруватися з системою та розширювати її функціонал. Інтерфейси мають бути розроблені з урахуванням принципів безпеки, забезпечуючи захист даних від несанкціонованого доступу та забезпечуючи шифрування конфіденційної інформації.

3.1.4 Комунікаційний протокол

Комунікація між компонентами системи повинна здійснюватися через протокол HTTPS для забезпечення безпеки переданих даних. Взаємодія між FrontEnd та BackEnd відбувається за допомогою REST API, що забезпечує стандартизовану та ефективну передачу даних. Використання HTTPS гарантує, що всі дані, передані між клієнтом і сервером, захищені від перехоплення та підміни. Комунікаційний протокол має бути налаштований для забезпечення високої швидкості передачі даних, мінімізуючи затримки та забезпечуючи швидкий відгук системи на запити користувачів. Додатково, протокол повинен підтримувати механізми автентифікації та авторизації для контролю доступу до ресурсів системи.

3.1.5 Обмеження пам’яті

Продукт має бути оптимізованим для роботи в середовищі з обмеженими ресурсами, включаючи обмеження по оперативній пам'яті і дисковому просторі для контейнерів Docker. Система повинна ефективно використовувати доступні ресурси, забезпечуючи високу продуктивність навіть при обмеженій кількості оперативної пам’яті та дискового простору. Необхідно забезпечити механізми для моніторингу та управління ресурсами, що дозволить вчасно виявляти та усувати проблеми, пов'язані з недостатністю ресурсів. Оптимізація коду та архітектури системи дозволить зменшити навантаження на пам’ять та забезпечити стабільну роботу навіть при значних навантаженнях.

3.1.6 Операції

Основні операції включають обробку запитів на обслуговування, генерацію сповіщень, обробку платежів, а також резервне копіювання та відновлення даних. Всі операції мають бути автоматизовані і здійснюватися через CI/CD пайплайн за допомогою GitHub Actions. Операції повинні виконуватися швидко та ефективно, забезпечуючи високу продуктивність системи та задоволення користувачів. Система має забезпечувати можливість масштабування операцій для обробки збільшеного обсягу запитів та даних без втрати продуктивності. Крім того, операції повинні бути задокументовані та забезпечувати можливість відстеження та моніторингу для виявлення та усунення проблем.

3.2 Атрибути програмного продукту

3.2.1 Надійність

Програмний продукт повинен забезпечувати високу надійність через впровадження механізмів автоматичного відновлення і дублювання даних. Azure SQL забезпечує автоматичне резервне копіювання та відновлення, що зменшує ризик втрати даних. Надійність системи досягається за рахунок використання стійкої архітектури, яка включає резервні копії даних, механізми відновлення після збоїв та автоматичне моніторинг стану системи. Додатково, система повинна підтримувати тестування на стресові та навантажувальні умови для виявлення потенційних слабких місць та забезпечення їх усунення до розгортання в реальне середовище.

3.2.2 Доступність

Продукт повинен бути доступний 24/7 з мінімальним часом простою. Використання Azure SQL і Docker забезпечує високу доступність та швидке відновлення сервісу у випадку збоїв. Доступність досягається за рахунок використання хмарних технологій, що забезпечують автоматичне масштабування та розподіл навантаження між серверами. Система повинна включати механізми резервного копіювання та відновлення, що забезпечить можливість швидкого відновлення роботи після збоїв або інших непередбачених ситуацій. Крім того, необхідно забезпечити надійність мережевих з'єднань та резервування ключових компонентів системи для забезпечення безперервної роботи.

3.2.3 Безпека

Безпека є критичним аспектом, включаючи захист даних, що зберігаються та передаються. Azure SQL забезпечує шифрування даних, а використання HTTPS гарантує безпечну передачу даних між компонентами

системи. Крім того, аутентифікація та авторизація користувачів повинні бути налаштовані з використанням сучасних методів захисту, таких як OAuth. Безпека системи також включає захист від атак типу SQL-ін'єкцій, XSS та CSRF, а також регулярне проведення аудитів безпеки та тестування на проникнення. Всі дані користувачів повинні Бути захищені відповідно до вимог GDPR або інших відповідних нормативних актів.

3.2.4 Супроводжуваність

Система повинна бути легко супроводжуваною завдяки добре структурованому коду та детальній документації. Використання .NET та React спрощує процес супроводу та оновлення компонентів. Супроводжуваність забезпечується за рахунок модульної архітектури, що дозволяє легко вносити зміни та додавати новий функціонал без впливу на інші частини системи. Документація повинна включати опис всіх компонентів, API та інструкції з розгортання та супроводу системи. Регулярні оновлення та виправлення помилок повинні здійснюватися через налаштований CI/CD пайплайн, що забезпечить швидке та безпечне впровадження змін.

3.2.5 Переносимість

Продукт повинен бути переносимим між різними середовищами завдяки використанню Docker. Контейнери забезпечують незалежність від апаратного забезпечення та операційної системи, що дозволяє легко переміщувати систему між локальними та хмарними середовищами. Переносимість досягається за рахунок стандартизованих контейнерів, що включають всі необхідні залежності та налаштування для запуску системи. Використання Docker Compose спрощує процес розгортання та конфігурації системи в різних середовищах, забезпечуючи можливість

швидкого та безпроблемного переміщення між середовищами розробки, тестування та продуктивного середовища.

3.2.6 Продуктивність

Система повинна забезпечувати високу продуктивність та швидкий відгук на запити користувачів. Оптимізація коду та використання ефективних алгоритмів дозволять зменшити затримки та забезпечити швидке виконання операцій. Azure SQL забезпечує високу продуктивність бази даних завдяки оптимізації запитів та ефективному управлінню ресурсами. Крім того, необхідно забезпечити моніторинг продуктивності та регулярне тестування системи на навантаження для виявлення та усунення потенційних вузьких місць. Використання масштабованих архітектурних рішень дозволить забезпечити високу продуктивність навіть при значному збільшенні навантаження на систему.

3.3 Вимоги бази даних

База даних повинна підтримувати зберігання великої кількості даних та швидкий доступ до них. Azure SQL забезпечує високу продуктивність та надійність завдяки вбудованим механізмам оптимізації запитів, автоматичному резервному копіюванню та відновленню. База даних повинна підтримувати складні запити та транзакції, забезпечуючи швидке виконання операцій та збереження цілісності даних. Необхідно забезпечити надійний захист даних, включаючи шифрування та контроль доступу на основі ролей (RBAC). Крім того, база даних повинна підтримувати механізми реплікації для забезпечення високої доступності та можливості відновлення даних у випадку збоїв.