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". Devops, Database
(тема)

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

Жеребний В.В.
(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 Жеребному Віктору Валерійовичу
(прізвище, ім'я, по батькові)

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

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 DevOps інструменти для веб-сервісу житлових комплексів для побудови та розгортання на ріних середовищах, розробити базу даних для веб-сервісу

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	09.04.2024	*виконано*
3	Проєктування ПЗ	11.04.2024	*виконано*
4	Розробка ПЗ	15.04.2024	*виконано*
5	Тестування ПЗ	03.05.2024	*виконано*
6	Оформлення пояснювальної записки	06.05.2024	*виконано*
7	Підготовка презентації та доповіді	20.05.2024	*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) _____ Жеребний В.В.
(підпис)

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

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра: 73 с., 20 рис., 12 джерел.

АДМІНІСТРУВАННЯ, ВЕБ СЕРВІС, ЖИТЛОВІ КОМПЛЕКСИ, КОМУНАЛЬНІ ПОСЛУГИ, ТЕХНОЛОГІЯ DOCKER, ПЛАТФОРМА GITHUB ACTIONS, БАЗА ДАНИХ AZURE SQL

Об’єкт розробки – DevOps та Database частина веб-сервісу для автоматизації інформаційних процесів спільноти мешканців одного чи кількох територіально поєднаних будинків.

Мета розробки – створення бази даних з необхідними таблицями, контейнеризація BackEnd та FrontEnd сервісів, створення пайплайнів для побудови та розгортання веб-сервісу для житлових комплексів.

Метод рішення – середовище розробки VS Code, платформа Azure.

У результаті розробки створено базу даних Azure SQL, Docker контейнери для веб-сервісу та пайплайн для його розгортання на платформі Azure.

ADMINISTRATION, WEB SERVICE, RESIDENTIAL COPLEXES, COMMUNAL SERVICES, DOCKER TECHNOLOGY, GITHUB ACTIONS PLATFORM, AZURE SQL DATABASE

The object of Development – the design and development of the database and DevOps services for the residential complex web service.

The purpose of Development – create a database with the necessary tables, containerize the BackEnd and FrontEnd services, and create pipelines for building and deploying the web service.

Solution Method – VS Code Development environment, Azure platform.

As a result of development an Azure SQL database, Docker containers for the web service, and a deployment pipeline on the Azure platform have been created.

Я, Жеребний Віктор Валерійович, 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". DevOps, Database», 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

Сучасні житлові комплекси потребують ефективного управління, що включає в себе облік і контроль ресурсів, комунікацію з мешканцями, технічне обслуговування та інші адміністративні функції. Веб-системи можуть значно покращити ці процеси, забезпечуючи централізований доступ до інформації та автоматизацію рутинних завдань.

З точки зору DevOps, розробка такої системи вимагає інтеграції різних інструментів і процесів для забезпечення надійного та ефективного розгортання і підтримки веб-додатка. DevOps підхід включає в себе безперервну інтеграцію та розгортання (CI/CD), що дозволяє швидко впроваджувати нові функції та оновлення, мінімізуючи ризики виникнення збоїв. Використання контейнеризації, наприклад Docker, забезпечує ізоляцію середовищ розробки і продуктивного середовища, що сприяє стабільній роботі додатка. Крім того, застосування інструментів оркестрації, дозволяє масштабувати систему в залежності від навантаження та забезпечувати високу доступність сервісів.

Важливим пунктом також є бази даних, оскільки ефективна веб-система для житлових комплексів повинна мати надійну і масштабовану архітектуру баз даних. Це включає в себе вибір відповідного типу бази даних, наприклад реляційної бази даних для структурованих даних. Управління базами даних передбачає налаштування резервного копіювання, відновлення даних та забезпечення безпеки. Крім того, важливим аспектом є оптимізація запитів і індексація, що забезпечує швидкий доступ до необхідної інформації.

Успішна реалізація проєкту потребує тісної взаємодії між різними складовими веб-сервісу. DevOps методологія сприяє цій співпраці, забезпечуючи прозорість процесів та автоматизацію задач, що, у свою чергу, дозволяє фокусуватися на покращенні якості продукту та

задоволенні потреб користувачів. Інтеграція баз даних з іншими компонентами системи через API забезпечує гнучкість та модульність архітектури, що полегшує впровадження нових функціональних можливостей та адаптацію до змінних вимог бізнесу.

Таким чином метою роботи є розробка DevOps сервісів для системи житлових комплексів з урахуванням практик та ефективного управління базами даних, що є ключовим етапом у створенні сучасного, надійного та масштабованого рішення для управління житловою інфраструктурою. Це забезпечить не тільки покращення обслуговування мешканців, але й оптимізацію операційних процесів для управляючих компаній.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Аналіз предметної галузі веб-сервісів для житлових комплексів потребує детального аналізу з точки зору DevOps та Database розробки і включає детальне вивчення сучасних підходів до управління житловими комплексами та існуючих технологічних рішень. Основними аспектами, які потребують розгляду, є вимоги до функціональності системи, існуючі проблеми в управлінні житловими комплексами та можливості покращення цих процесів за допомогою сучасних технологій.

Управління житловими комплексами включає різні завдання, такі як облік мешканців, управління фінансовими транзакціями, планування технічного обслуговування, комунікація з мешканцями та інші адміністративні функції. Традиційні методи управління часто вимагають значних ресурсів та є менш ефективними, що призводить до затримок і помилок. Веб-система, яка автоматизує ці процеси, може значно підвищити ефективність та точність роботи.

З точки зору DevOps, сучасні методи розробки програмного забезпечення націлені на автоматизацію та оптимізацію процесів розгортання, тестування та моніторингу [1, с. 98]. DevOps підхід дозволяє забезпечити безперервну інтеграцію та доставку, а також мінімізує ризики збоїв. Автоматизація тестування та розгортання дозволяє виявляти та виправляти помилки на ранніх етапах, забезпечуючи високу якість кінцевого продукту. Крім того, використання інфраструктури як коду (IaC) сприяє стандартизації середовищ розробки та продуктивного середовища, що полегшує управління конфігураціями та забезпечує повторюваність процесів.

З точки зору баз даних, ефективне управління даними є критично важливим для веб-системи житлових комплексів. Необхідно забезпечити

надійність, масштабованість та безпеку даних. Для зберігання структурованих даних, таких як інформація про мешканців, фінансові транзакції та розклади технічного обслуговування можут використовуватися реляційні бази даних, наприклад Azure SQL. Для зберігання нереляційних даних, таких як журнали подій або дані з лічильників, в подальшому, можуть використовуватися NoSQL бази даних, такі як MongoDB або Cassandra. Забезпечення високої доступності та відмовостійкості баз даних досягається за допомогою реплікації та налаштування кластерів. Також важливим є налаштування регулярного резервного копіювання та відновлення даних для запобігання втратам інформації.

Важливо врахувати існуючі рішення на ринку та їхні можливості. Це дозволить виявити їхні сильні та слабкі сторони, а також визначити унікальні характеристики та переваги розроблюваної веб-системи. Аналіз конкурентів може включати вивчення функціональності, інтерфейсів користувача, технологічної архітектури та відгуків користувачів, що дозволить сформулювати вимоги до системи, що забезпечують її конкурентоспроможність та відповідність потребам ринку.

Важливо також розглянути, як веб-система може допомогти вирішити сучасні проблеми та виклики, такі як екологічні питання, активна урбанізація та ефективне управління комунальними послугами:

— система може сприяти екологічності, дозволяючи відслідковувати та оптимізувати споживання енергії та води, впроваджувати системи роздільного збору сміття та інші екологічні ініціативи;

— в умовах активної урбанізації веб-система може забезпечити ефективну комунікацію між мешканцями та управителями, полегшуючи вирішення питань, що виникають у густонаселених районах;

— управління комунальними послугами може бути значно спрощене завдяки автоматизації обліку споживання, плануванню та моніторингу технічного обслуговування, а також інтеграції з різними постачальниками послуг.

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 функціонал, наприклад замовлення послуг, оплата комунальних послуг та можливість голосування (див. рис. 1.1), та активно використовується в сполучених штатах.

З точки зору DevOps реалізації веб-сервісу варто зазначити, що вони, ймовірно, використовують популярні інструменти для автоматизації процесів безперервної інтеграції та розгортання, такі як Jenkins або GitLab CI/CD. Це забезпечує стабільність, швидке впровадження нових функцій та виправлення помилок. Більше того, Condo Control Central забезпечує високу ефективність та швидкість обробки запитів користувача та завантаження усіх статичних матеріалів, що ймовірно реалізовано за допомогою кешування та завдяки оптимізованій архітектурі та використанню передових технологій для масштабування та розподілу навантаження.

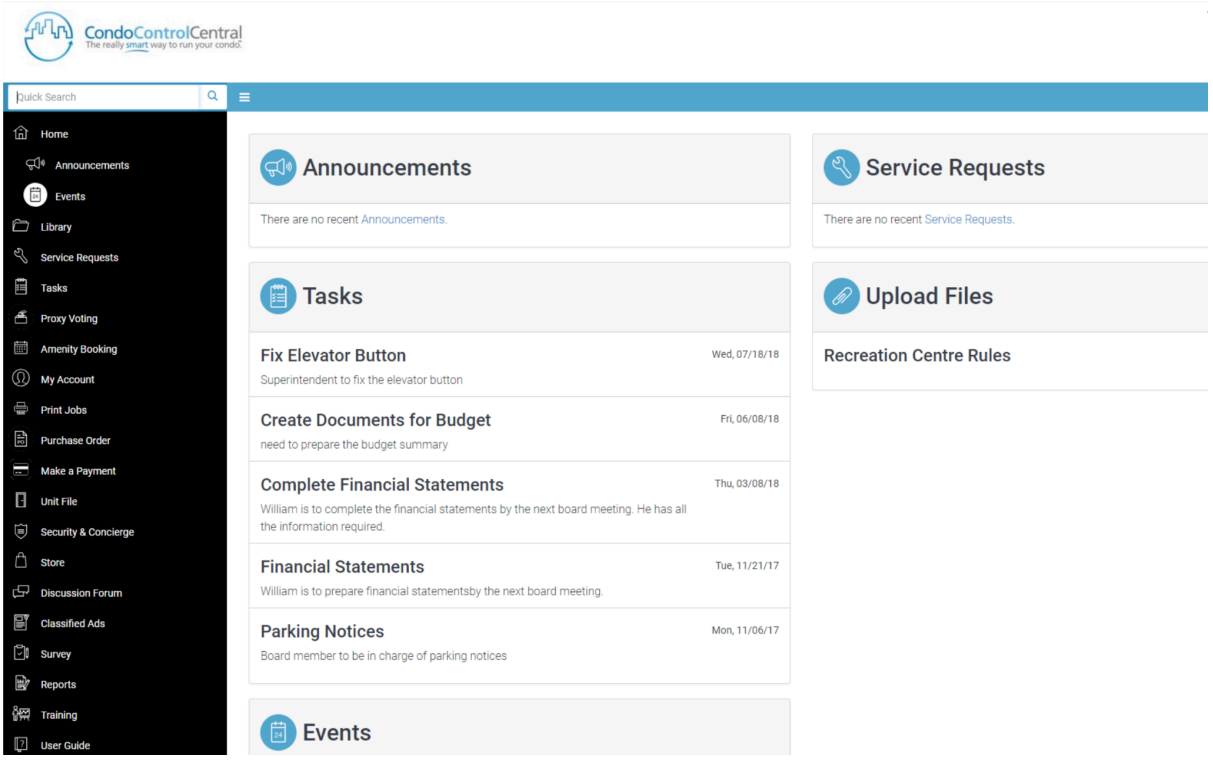


Рисунок 1.1 – Інтерфейс сторінки адміністратора Condo Control Central (за даними [2])

BuildingLink також пропонує широкий функціонал (див. рис. 1.2), але вона може бути менш інтегрованою та менш гнучкою, ніж розроблювана система. Вона спеціалізується на об’єктах, що будуються або були введені в експлуатацію впродовж останніх декількох років.

Працює BuildingLink досить повільно, з чого можна зробити висновок про те, що вони в своїй реалізації використовують застарілі реляційні бази даних без можливості легкої масштабованості. Відсутність підтримки сучасних NoSQL баз даних обмежує їх здатність ефективно працювати з великими обсягами даних в реальному часі. Відсутність реплікації та кластеризації баз даних призводить до низької доступності та підвищеного ризику втрат даних у випадку збоїв, що в багатьох випадках може призвести до критичних наслідків.

Застаріла архітектура BuildingLink також не дозволяє ефективно масштабувати систему для обробки високих навантажень, що призводить

до затримок і зниження продуктивності. Відсутність оптимізації запитів та індексації знижує швидкість доступу до даних, що негативно впливає на користувацький досвід. Ще більше проблема полягає в тому, що BuildingLink не забезпечує належного захисту даних. Відсутність сучасних методів шифрування та контролю доступу робить систему вразливою до атак і несанкціонованого доступу, що підвищує ризик компрометації даних.

Рисунок 1.2 – Інтерфейс сторінки зі списком мешканців BuildingLink (за даними [3])

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.

Yardi Voyager, як велика система управління нерухомістю, використовує складну архітектуру баз даних, включаючи реляційні бази

даних, що завдяки своїм розширеним можливостям дозволяє веб-сервісу ефективно обробляти великий обсяг запитів, забезпечуючи при цьому високу продуктивність та швидкість доступу до даних. Також важливо зазначити те, що Yardi Voyager приймає усі необхідні заходи безпеки для захисту даних, включаючи шифрування, ретельний контроль доступу та комплексні плани відновлення після збоїв, що гарантує високий рівень захисту даних клієнтів

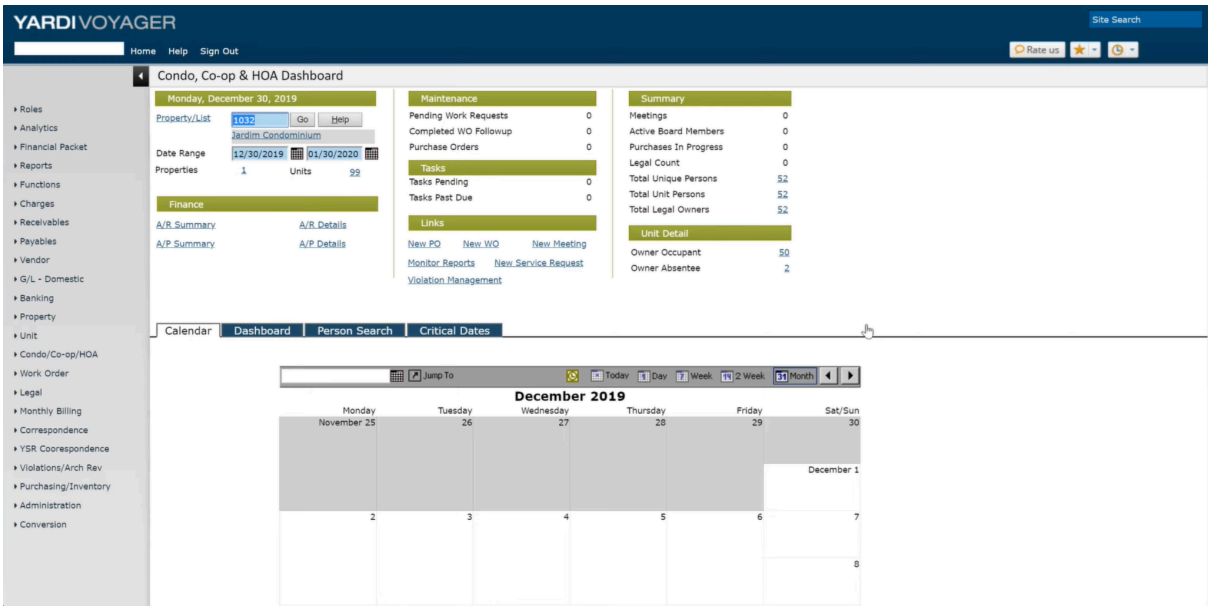


Рисунок 1.3 – Сторінка перегляду запланованих заходів в веб-сервісі Yardi Voyager (за даними [4])

Розроблювана система вирізняється широким функціоналом, гнучкістю, персоналізацією та можливістю інтеграції з великою кількістю житлових комплексів, що робить її привабливим вибором для управління нерухомості. Також у порівнянні з конкурентами, деякі з яких можна вважати технічно застарілими, наш веб-сервіс для житлових комплексів більше орієнтований на використання сучасних DevOps практик та ефективне управління базами даних, що дозволить запропонувати потенційним клієнтам ще кращі рішення. Розроблювана система буде

забезпечувати високу продуктивність, надійність, безпеку та ефективність, що зробить її конкурентоспроможною на ринку та відповідатиме потребам сучасного управління житловими комплексами.

1.3 Виявлення проблем та актуалізація рішень

Треба розробити веб-сервіс, що не буде стикатися з проблемами повільного циклу розгортання та оновлення програмного забезпечення, що може знижувати можливість швидкого впровадження нових функцій та виправлення помилок. Відсутність автоматизованих процесів тестування і розгортання підвищує ризик людських помилок та знижує стабільність системи. Щоб вирішити ці проблеми, необхідно впровадити безперервну інтеграцію та доставку (CI/CD), що дозволяє автоматизувати процеси тестування, збірки та розгортання. Використання контейнеризації та оркестрації контейнерів сприяє більшій гнучкості та масштабованості системи, дозволяючи швидко розгортати нові версії програмного забезпечення та забезпечувати високу доступність.

З точки зору розробки баз даних, ключовою проблемою є забезпечення ефективного управління великим обсягом даних, при цьому в майбутньому традиційні реляційні бази даних можуть стати вузьким місцем при обробці необхідного обсягу даних або при масштабуванні [5, с. 64]. Для вирішення цієї проблеми необхідно використовувати гібридний підхід, поєднуючи реляційні бази даних для структурованих даних та NoSQL бази даних для гнучкого зберігання інших даних.

Іншою актуальною проблемою є забезпечення безпеки даних, оскільки веб-система для житлових комплексів обробляє велику кількість конфіденційної інформації. Впровадження сучасних методів шифрування даних, багатофакторної аутентифікації та контролю доступу є необхідним для захисту даних від несанкціонованого доступу та кібератак. Регулярні

аудити безпеки та впровадження рекомендацій з безпеки також є важливими для підтримки високого рівня захисту.

Загалом, виявлення проблем та актуалізація рішень у розробці веб-системи для житлових комплексів з точки зору DevOps та управління базами даних дозволяє створити ефективну, надійну та безпечну систему, яка відповідає сучасним вимогам і допомагає вирішити ключові проблеми управління житловими комплексами.

1.4 Постановка задачі

В розроблюваному веб-сервісі для житлових комплексів передбачаються дві ролі для користувачів: адміністратор, що має повний доступ для керування сервісом, та мешканець, який може використовувати веб-сервіс для вирішення нагальних проблем та спілкування з іншими мешканцями.

Постановка задачі розробки веб-сервісу для житлових комплексів з точки зору DevOps та Database розробки включає наступні пункти:

— визначення функціональних та нефункціональних вимог до системи з урахуванням потреб управителів житлових комплексів, мешканців та інших зацікавлених сторін;

— розробка архітектури системи, включаючи вибір технологій, розробку мікросервісної або монолітної архітектури, та визначення взаємодії між компонентами;

— налаштування системи безперервної інтеграції (CI) та безперервної доставки (CD), до якої буде мати доступ адміністратор веб-сервісу, вибір та налаштування інструментів автоматизації розгортання, моніторингу та логування для забезпечення швидкого впровадження оновлень та виявлення помилок з використанням GitHub Actions [6];

— створення та оптимізація реляційної Azure SQL бази даних для ефективного зберігання, обробки та доступу до даних щодо користувачів, включно з мешканцями та адміністраторами, їх фінансові дані, технічне обслуговування житлових комплексів та іншу важливу інформацію, адміністратор має при цьому мати повний доступ до даної бази даних;

— впровадження заходів безпеки даних, включаючи шифрування, контроль доступу, аудит безпеки та впровадження найкращих практик для захисту конфіденційної інформації;

— проведення регулярного тестування для перевірки функціональності, продуктивності та безпеки системи, а також організація проведення збору зворотного зв’язку від користувачів сервісу;

Для розгортання веб-сервісу планується використовувати Docker, оскільки його використання для розгортання проєкту має низку переваг, наприклад Docker може забезпечити ізольоване середовище для системи, що дозволяє уникнути конфліктів залежностей та інших проблем, пов'язаних зі змінними конфігураціями середовища, а також контейнери можуть бути легко перенесені між різними середовищами, що дозволяє вам розгортати свій проєкт на будь-якому сервері або в хмарному середовищі без проблем із сумісністю та меншій потребі в часі для розгортання [7].

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Функціональні вимоги

Функціональні вимоги в програмній системі для управління житловими комплексами з точки зору Devops включає у себе наступні пункти:

а) аутентифікація та авторизація;

1) налаштування безперервної інтеграції для автоматизованого тестування модулів аутентифікації та авторизації, за допомогою GitHub Actions;

2) використання контейнеризації для ізоляції сервісів, що відповідають за аутентифікацію та авторизацію, використовуючи Docker;

3) забезпечення шифрування та захисту маркерів доступу;

4) використання централізованої системи керування доступом для ефективного управління правами користувачів;

б) модуль замовлення та відстеження послуг;

1) Налаштування системи логування та моніторингу для відстеження дій користувачів та змін статусу заявок;

2) використання сучасних методів підписки на події для нотифікації користувачів про зміни статусу їхніх заявок;

в) внутрішні чати для мешканців;

1) розробка архітектури для чат-системи з можливістю горизонтального масштабування;

2) використання бази даних з вбудованим підтримкою транзакцій для забезпечення консистентності даних у чатах;

г) модуль оголошень;

1) розробка системи кешування для швидкого відображення оголошень;

2) використання реляційних баз даних для зберігання оголошень та коментарів;

д) модуль голосування;

1) створення бази даних Azure SQL для зберігання результатів голосувань та швидкого доступу до них.

2.2 Нефункціональні вимоги

До нефункціональних вимог веб-сервісу можна віднести:

а) продуктивність;

1) налаштування та оптимізація серверної та клієнтської частини для мінімізації часу відповіді на запити користувачів. Використання кешування та CDN для завантаження статичного контенту швидше 50 мс;

2) Використання асинхронних запитів та оптимізація запитів до баз даних для забезпечення мінімальних затримок, підтримка базою даних ACID транзакцій;

б) безпека;

1) використання сучасних методів шифрування для захисту особистих даних користувачів та конфіденційної інформації, виконання вимог GDPR;

2) використання механізмів захисту від SQL-ін'єкцій, кросс-сайт скриптінгу та інших відомих атак, а також встановлення брандмауерів та систем виявлення вторгнень;

в) надійність;

1) використання цілодобового моніторингу та систем реагування на відмови для забезпечення неперервної роботи системи та швидкого виявлення та усунення проблем;

г) підтримка;

1) регулярні оновлення системи та надання підтримки з інтервалом часу для забезпечення актуальності та безпеки системи;

2.3 Вимоги до середовища розгортання

В якості середовища розгортання для веб-сервісу буде використовуватися Azure, воно повиннобути налаштоване для забезпечення швидкого та ефективного розгортання, масштабованості, безпеки та стабільності системи [8]. З точки зору DevOps, це означає налаштування автоматизованих процесів розгортання (CI/CD), використання інструментів для контролю версій коду та управління конфігураціями, а також моніторингу та логування для виявлення та вирішення проблем. Стосовно баз даних, вимоги включають створення та налаштування баз даних у середовищі з урахуванням вимог до продуктивності, масштабованості та безпеки. Важливо також забезпечити регулярне резервне копіювання та відновлення даних для запобігання втрати інформації та забезпечення більшої надійності системи.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ. 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.

Більш детально з DevOps точки зору функціональне моделювання веб-сервісу для житлових комплексів охоплює кілька важливих аспектів, що забезпечують його ефективну роботу і масштабованість., основною метою є автоматизація процесів розгортання, моніторингу та підтримки системи, що забезпечує надійність і безперервність сервісу. Це включає в себе налаштування CI/CD для автоматизації розгортання нових версій додатка. Використання контейнеризації дозволяє забезпечити консистентність середовищ розробки, тестування і продуктивного

середовища, а подальша оркестрація контейнерів за допомогою дозволяє легко масштабувати сервіс залежно від навантаження.

З боку роботи над базою даних, функціональне моделювання включає проєктування структури даних, що забезпечить ефективне зберігання та доступ до інформації. Для веб-сервісу житлових комплексів важливо розробити реляційну базу даних, яка зможе обробляти велику кількість запитів від різних користувачів. Це включає створення бази даних для зберігання даних про мешканців, історію чатів, запити, оголошення, тощо. Використання нормалізації дозволяє зменшити дублювання даних і покращити цілісність бази даних. Водночас, для забезпечення високої продуктивності, необхідно розробити індекси для часто використовуваних полів та реалізувати механізми кешування.

Таким чином, функціональне моделювання веб-сервісу для житлових комплексів з точки зору DevOps та Database розробки охоплює цілу низку заходів з автоматизації, моніторингу та забезпечення надійності, а також детальне проєктування структури бази даних для ефективного зберігання і обробки інформації.

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.

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

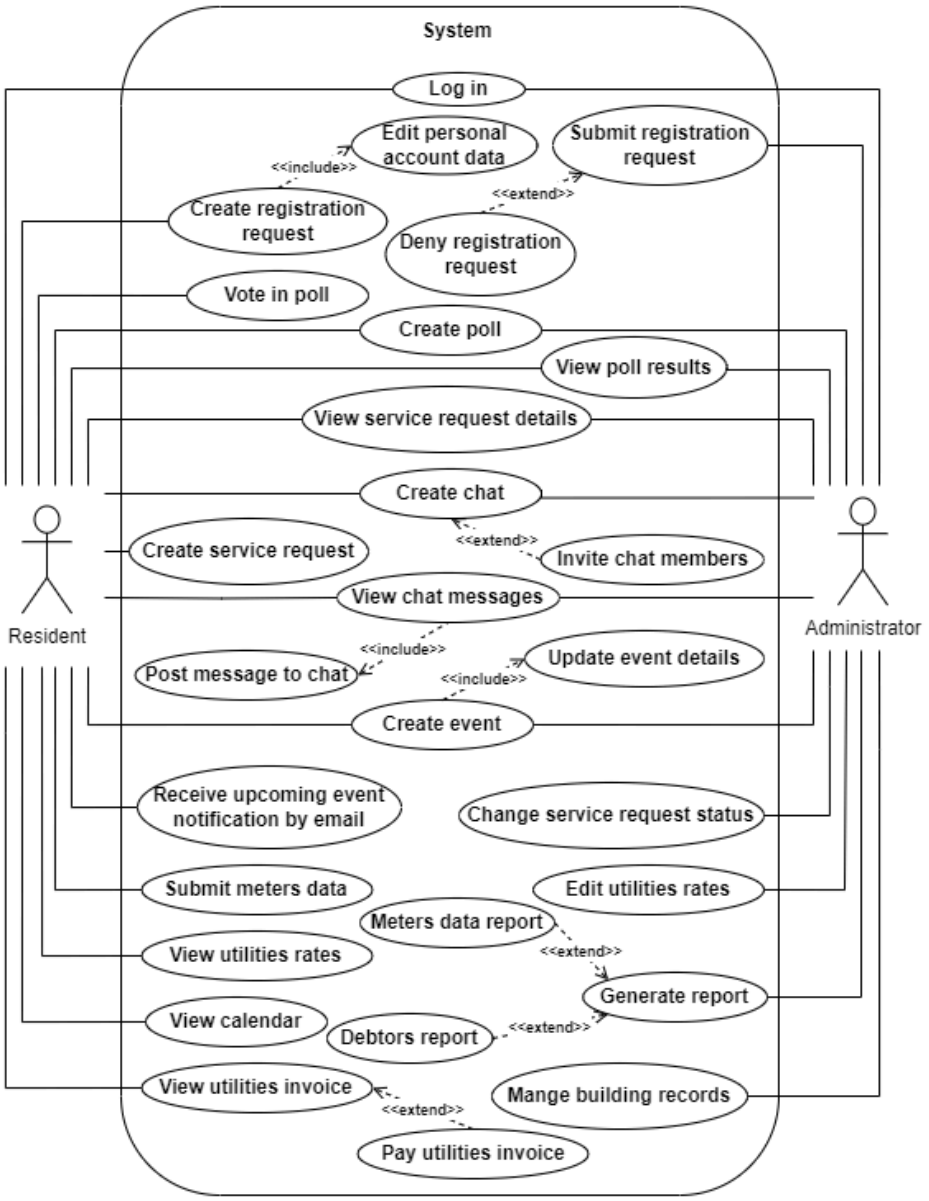


Рисунок 3.1 – Use Case діаграма (виконаний самостійно)

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

3.2 Проєктування архітектури ПЗ

Проєктування архітектури програмного забезпечення з використанням сервісів Azure та .NET з точки зору DevOps і Database

розробки включає створення чистої архітектури, де кожен сервіс незалежно масштабується та розгортається. Використання Azure DevOps забезпечує автоматизацію процесів CI/CD для швидкого та безперервного розгортання та інтеграції. В якості основної реляційної бази даних планується використання Azure SQL. Архітектура повинна передбачати використання контейнеризації та оркестрації для управління та розгортання сервісів. Забезпечення безпеки даних та додатків досягається за рахунок використання Azure Key Vault для управління секретами та Azure Active Directory для аутентифікації та авторизації. Моніторинг і логування реалізуються за допомогою Azure Monitor та Azure Log Analytics. При розробці важливо використати наступні шаблони проєктування:

— Task Queue, яка в контексті DevOps дозволяє автоматизувати та оптимізувати обробку складних або ресурсомістких операцій, що є критично важливим для безперервного розгортання та доставки оновлень за допомогою GitHub Actions. Черга завдань також полегшує моніторинг і відстеження стану завдань, дозволяючи швидко реагувати на помилки або затримки.

— Replicated Service, що використовується в Azure App Service, забезпечує високу доступність і надійність додатка шляхом створення декількох копій сервісів, які працюють одночасно. Це дозволяє зберегти працездатність системи навіть у разі збою окремих компонентів. Реплікація сервісів також сприяє балансуванню навантаження, що покращує масштабованість і продуктивність додатка.

На діаграмі пакетів схематично зображені компоненти архітектури системи, що розроблюється, та зв’язки між ними (див. рис. 3.2).

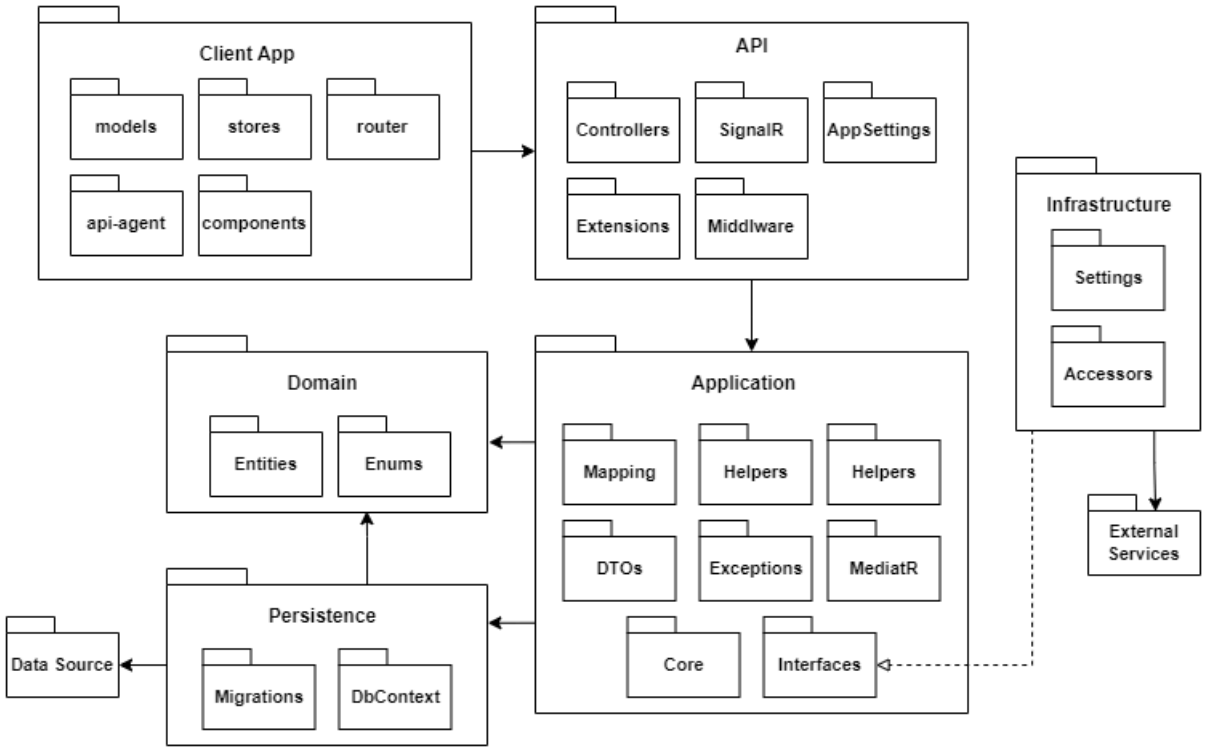


Рисунок 3.2 – Діаграма пакетів (виконаний самостійно)

Іншою значною перевагою чистої архітектури є те, що її використання сприяє спрощенню вертикальної та горизонтальної маштабованості. Частини системи максимально незалежні одна від одної, що спрощує масштабування окремих компонентів.

У якості сховища великих файлів, таких як зображення та відео, планується використовувати хмарний сервіс «Cloudinary», за комунікацію з яким безпосередньо відповідає шар інфраструктури.

З точки зору DevOps, розгортання інфраструктури додатку буде відбуватись у хмарному рішенні Microsoft Azure та задіятиме наступні сервіси:

— Azure App Service: повністю керована платформа для розгортання, масштабування та управління веб-додатками та службами. Такі частини нашої системи як Client App та API будуть розгорнуті за допомогою цього сервісу. Перевагами використання Azure App Service є

швидке розгортання, автоматичне масштабування та зручність інтеграції з іншими сервісами Azure.

— Azure SQL Database: хмарна реляційна база даних, що зберігатиме всі дані нашої системи крім великих файлів. Головною перевагою цього хмарного рішення є висока доступність та масштабованість;

— Azure Monitor: система для моніторингу роботи нашого веб-сервісу у реальному часу для завчасного виявлення та усунення проблем у роботі наших компонентів системи.

В рамках проєктування архітектури програмного забезпечення для веб-сервісу житлових комплексів ще одним важливим аспектом є необхідність використання Docker, що забезпечує консистентність, ізоляцію і масштабованість середовищ розробки, тестування та розгортання. Docker дозволяє контейнеризувати додаток з усіма його залежностями у контейнери, що гарантує стабільну роботу додатка незалежно від середовища, в якому він запускається. Це особливо важливо для розподілених систем, які складаються з різних компонентів, оскільки забезпечує ізольовану роботу кожного з компонентів і мінімізує конфлікти між ними.

Розроблюваний веб-сервіс поділений на FrontEnd та BackEnd компоненти, кожен з яких виконує свої специфічні функції і має свої власні вимоги до середовища виконання. FrontEnd відповідає за інтерфейс користувача, забезпечуючи взаємодію користувачів з системою через веб-браузер. В рамках веб-сервісу він буде побудований з використанням таких технологій, як TypeScript, React та MobX. BackEnd, у свою чергу, відповідає за обробку запитів, бізнес-логіку і взаємодію з базою даних. Він буде побудований з використанням таких технологій як .NET та SignalR.

Для забезпечення ізоляції і управління залежностями кожного з цих компонентів використовується Docker. Контейнеризація FrontEnd та

BackEnd дозволяє створити стандартизоване середовище для кожного з них, що включає всі необхідні бібліотеки, пакети і конфігурації. Це гарантує, що додаток буде працювати однаково на всіх етапах розробки та експлуатації – від локального середовища розробників до тестових серверів і продуктивного середовища.

Створення образів Docker для FrontEnd та BackEnd – це один з ключових етапів у пайплайні CI/CD. Після того як код перевірений і затверджений за допомогою GitHub Actions запускається процес побудови образів. Спеціальні Docker-файли для FrontEnd і BackEnd визначають, як саме повинні бути побудовані ці образи. Наприклад, для FrontEnd може використовуватися базове зображення `node `, а для BackEnd - ` dotnet:sdk`. Процес побудови включає в себе встановлення всіх необхідних залежностей і налаштування додатків, що гарантує, що готовий образ міститиме все необхідне для запуску додатка.

Після створення образів вони завантажуються до реєстру контейнерів, де вони можуть бути доступні для розгортання. На наступному етапі пайплайн завантажує їх до реєстру контейнерів, який далі буде використовуватись для налаштування хмарних сервісів Azure App Service для FrontEnd та BackEnd.

Для додаткового забезпечення надійності і масштабованості, Azure App Service дозволяє визначати політики автоматичного перезапуску контейнерів у разі їх падіння, а також налаштовувати обмеження на ресурси, які можуть використовувати контейнери. Це забезпечує стабільну роботу системи навіть у разі виникнення непередбачених ситуацій.

Важливою перевагою використання Docker в пайплайні є можливість легко відтворювати і тестувати середовища на різних етапах розробки. Тестові середовища можуть бути створені і знищені за запитом, що дозволяє розробникам швидко перевіряти зміни і забезпечувати високу якість коду перед його випуском у продуктивне середовище. Крім того,

конфігурації середовищ можуть бути версіоновані разом із кодом, що забезпечує точність і повторюваність процесів розгортання.

Таким чином, використання Docker у проєктуванні архітектури веб-сервісу для житлових комплексів дозволяє створити ефективний, надійний і гнучкий процес розгортання, який автоматизується через CI/CD пайплайн. Це забезпечує швидке впровадження змін, мінімізацію ризиків, пов'язаних з конфігурацією середовищ, і високу доступність системи для кінцевих користувачів. Docker дозволяє ізолювати компоненти системи, спрощувати управління залежностями і забезпечувати безперебійну інтеграцію і доставку програмного забезпечення, що є критичним для успішної реалізації веб-сервісу з високими вимогами до надійності і продуктивності. На основі прийнятих рішень було сплановано та візуалізовано процеси сборки та розгортання компонентів веб-сервісу (див. рис. 3.3 та рис. 3.4).

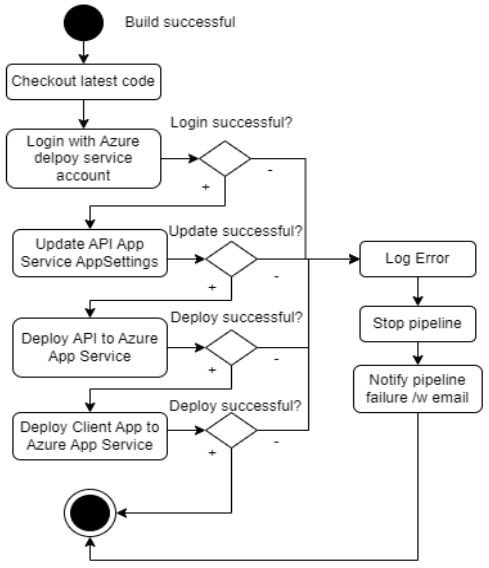


Рисунок 3.3 - Діаграма діяльності процесу розгортання веб сервісу (виконаний самостійно)

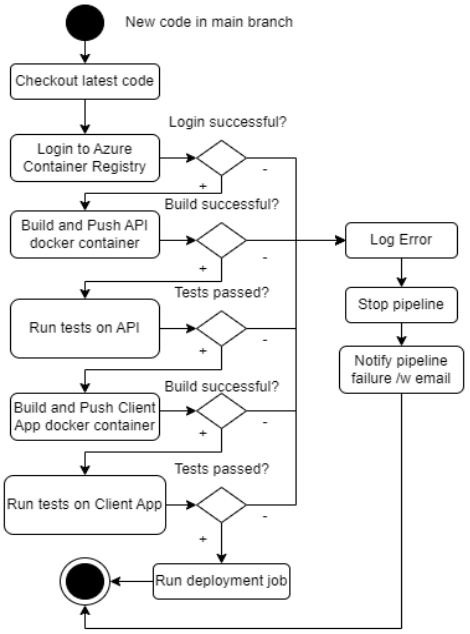


Рисунок 3.4 - Діаграма діяльності процесу зборки веб сервісу (виконаний самостійно)

Таким чином було обрано та детально сплановано архітектуру веб-сервісу «Житловий комплекс» та її розгортання у хмарі.

3.3 Проєктування структури зберігання даних

Для проєктування бази даних веб-сервісу було обрано реляційну модель бази даних за наступних причин:

— Структурований підхід до організації даних. Дані зберігаються у вигляді таблиць з рядками і стовпцями, що спрощує організацію та управління великими обсягами інформації;

— Можливість використовувати різні типи обмежень цілісності, такі як унікальність, первинний ключ, зовнішній ключ тощо. Це дозволяє забезпечити високу якість даних та запобігти появі некоректних або несумісних даних;

— Широкий спектр операцій, включаючи пошук, сортування, фільтрацію, з'єднання таблиць та агрегацію даних. Це надає розробникам

гнучкість та можливість ефективно взаємодіяти з даними в різних сценаріях;

— Реляційна модель дозволяє розділити фізичне збереження даних від логічної структури бази даних. Це означає, що можна змінювати фізичну структуру без зміни логічної структури даних, що дозволяє забезпечити більшу гнучкість та ефективність в управлінні даними;

— Реляційна модель даних є масштабованою та може обробляти великі обсяги даних. За допомогою оптимізаційних технік та індексів можна забезпечити швидкий доступ до даних, навіть у великих базах даних з мільйонами рядків;

Таблиці бази даних, що використовуються сервером:

— “Activity” таблиця, що зберігає інформацію про різноманітні діяльності, такі як анонси, чати, опитування тощо;

— “AppUser” таблиця для зберігання інформації про користувачів додатку;

— “Announcement” підтип таблиці Activity, яка містить додаткові поля для оголошень;

— ”Chat” Підтип таблиці Activity, яка представляє чати.

— ”Comment” таблиця для зберігання коментарів до діяльностей.

— ”Invoice” таблиця для зберігання рахунків, пов'язаних із певними квартирами.

— “MeterData” таблиця для зберігання даних з лічильників, пов'язаних із певними квартирами.

— ”Poll” підтип таблиці Activity, що представляє опитування.

— ”PollOption” таблиця для зберігання варіантів відповідей в опитуваннях.

— ”RegistrationRequest” таблиця для зберігання запитів на реєстрацію, пов'язаних із певними квартирами.

— ”ResidentBuilding” таблиця для зберігання інформації про будівлі, в яких розташовані квартири..

— ”Photo” таблиця для зберігання фотографій, що можуть бути пов'язані з різними сутностями.

— ”ActivityAttendee” таблиця для зберігання відносин між діяльностями та учасниками.

— ”Votes” таблиця для зберігання голосів за варіанти в опитуваннях.

— ”Apartment” таблиця для зберігання інформації про квартири.

На основі описаних вище таблиць була побудована схема бази даних веб-сервісу «Житловий комплекс» (див. рис. 3.3).

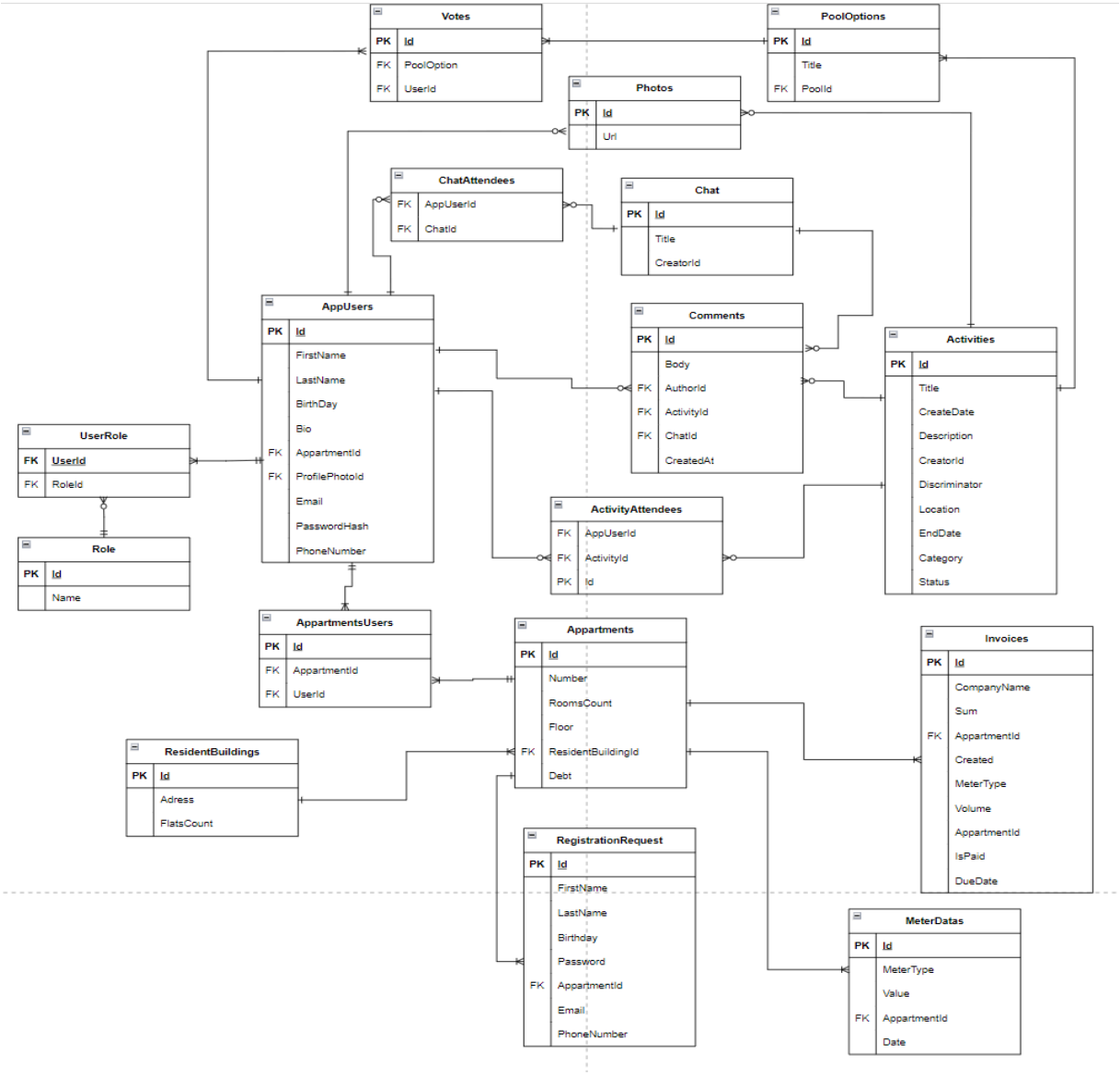


Рисунок 3.5 – ER-діаграма бази даних (виконаний самостійно)

База даних знаходиться в третій нормальній формі, що підтверджує її нормалізацію [9, с. 104]. 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.

Використання Azure SQL в якості бази даних для веб-сервісу для житлових комплексів надає безліч переваг, оскільки вона є повністю керованою реляційною базою даних як послуга (DBaaS), що забезпечує

високу продуктивність, масштабованість і надійність, необхідні для сучасних веб-додатків.

Ще однією з основних переваг Azure SQL є її інтеграція з іншими сервісами Azure, що дозволяє створити єдину екосистему для розробки та розгортання додатків. Для проєктів, щовикористовують .NET, це означає можливість використовувати найбільш опитимізовані інструменти та середовища для безперебійної інтеграції та безперервної доставки. Azure SQL підтримує всі сучасні можливості SQL, що забезпечує розробникам потужний набір інструментів для управління даними та виконання складних запитів.

Azure SQL також забезпечує високу доступність і надійність завдяки вбудованим функціям, таким як автоматичне резервне копіювання, відновлення після збоїв та гео-реплікація. Ці функції дозволяють зберігати дані в різних географічних регіонах, що забезпечує безперервну роботу додатка навіть у разі регіональних збоїв. Автоматичне резервне копіювання дозволяє зберігати резервні копії на різні періоди часу, що забезпечує можливість відновлення даних на будь-який момент. Гео-реплікація, в свою чергу, дозволяє створювати читальні репліки бази даних у різних регіонах, що забезпечує швидкий доступ до даних для користувачів з різних географічних локацій і зменшує затримки.

Особливістю Azure SQL є її здатність автоматично масштабувати ресурси відповідно до змінних навантажень. Це дозволяє базі даних адаптуватися до пікових навантажень, забезпечуючи необхідну продуктивність без необхідності втручання. Динамічне масштабування забезпечує оптимальне використання ресурсів і зниження витрат, оскільки платити необхідно тільки за ті ресурси, які фактично використовуються. Це особливо важливо для веб-сервісу, де навантаження може суттєво варіюватися залежно від часу доби, сезону, тощо.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Опис процесу розгортання та моніторингу сервісу

В ході опису прийнятих програмних рішень для веб-сервісу житлових комплексів з точки зору DevOps та розробки баз даних необхідно приділити увагу детальному розгляду вибраних технологій і підходів. У якості CI/CD платформи було обрано GitHub Actions, а в якості бази даних використовується Azure SQL. Ці рішення забезпечують високу надійність, продуктивність і безпеку системи.

З точки зору DevOps, вибір GitHub Actions для CI/CD забезпечує інтеграцію процесів розгортання безпосередньо з репозиторієм коду, що дозволяє значно спростити і автоматизувати робочі процеси. GitHub Actions надає можливість створювати робочі процеси, що автоматично запускаються при певних подіях, таких як пуш коду, створення пул-реквесту або реліз нової версії. Цей інструмент дозволяє налаштовувати послідовності завдань для перевірки, збірки та розгортання додатка. На основі GitHub Actions було розроблено два пайплайни для зборки проєкту (див. рис. 4.1) та його подальшого деплою (див. рис. 4.2).

Виконання пайплайнів запускається при кожному коміті до основної гілки, що забезпечує високу стабільність сервісу, можливість швидкого внесення змін у роботу сервісу в продакшн середовищі.

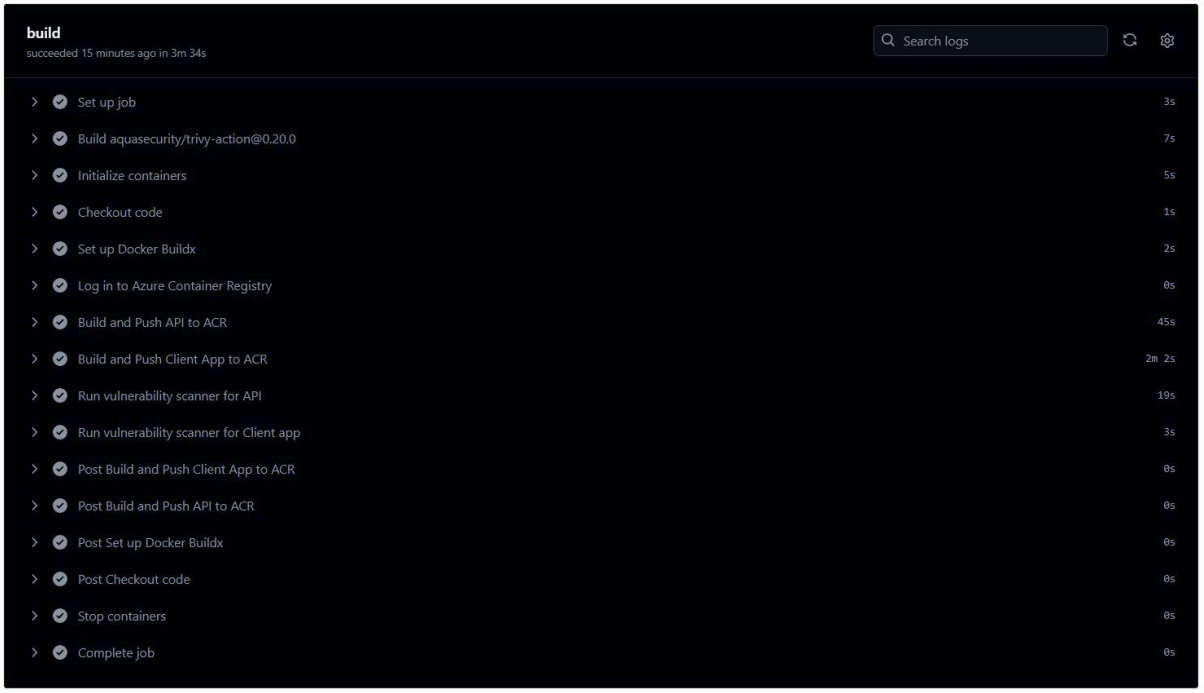


Рисунок 4.1 – Пайплайн для зборки сервісів GitHub Actions (виконаний самостійно)

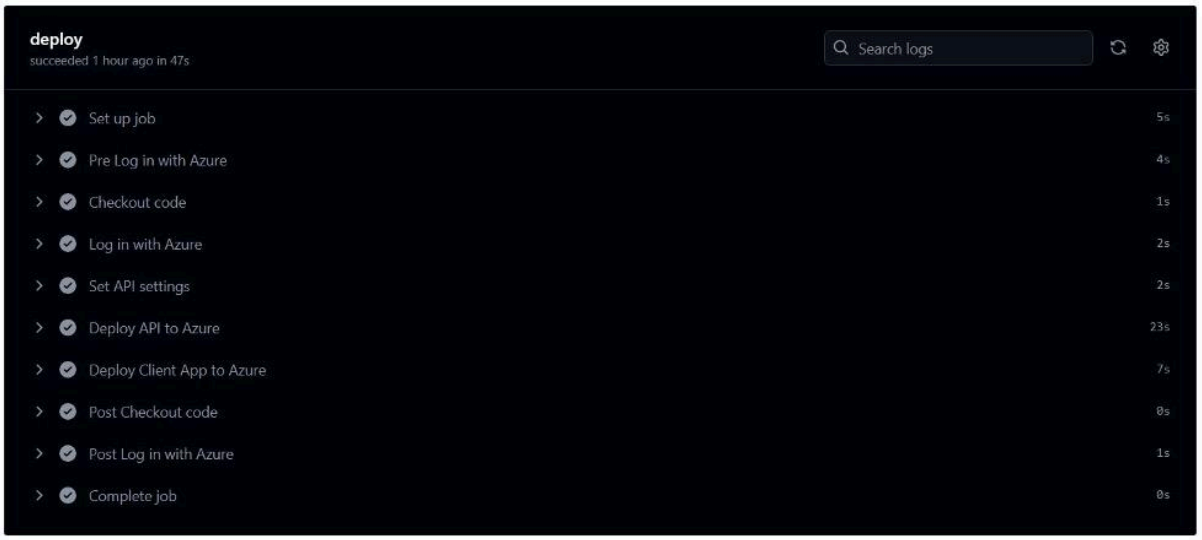


Рисунок 4.2 – Пайплайн для деплою сервісів GitHub Actions (виконаний самостійно)

GitHub Actions також підтримує інтеграцію з іншими сервісами та інструментами, такими як Docker для контейнеризації додатків, що

дозволяє створювати ізольовані середовища для тестування та забезпечує консистентність на різних середовищах. В ході розробки були контейнерізовані обидва FrontEnd сервіс (див. рис. 4.3) та BackEnd сервіс (див. рис. 4.4). Використання контейнеризації з Docker дозволяє створювати контейнери, що містять усі необхідні залежності для запуску додатка, що забезпечує незалежність від конфігурацій середовищ. Це дозволяє працювати у локальних середовищах, ідентичних до тестових та продуктивних, що значно зменшує кількість проблем, пов'язаних з різницею в налаштуваннях середовищ. Також для запуску багатоконтейнерних Docker додатків, що складаються з кількох сервісів, можна використовувати Docker Compose [10]. Це дозволяє розгортати весь стек додатків разом, включаючи веб-сервер, FrontEnd додаток і базу даних.

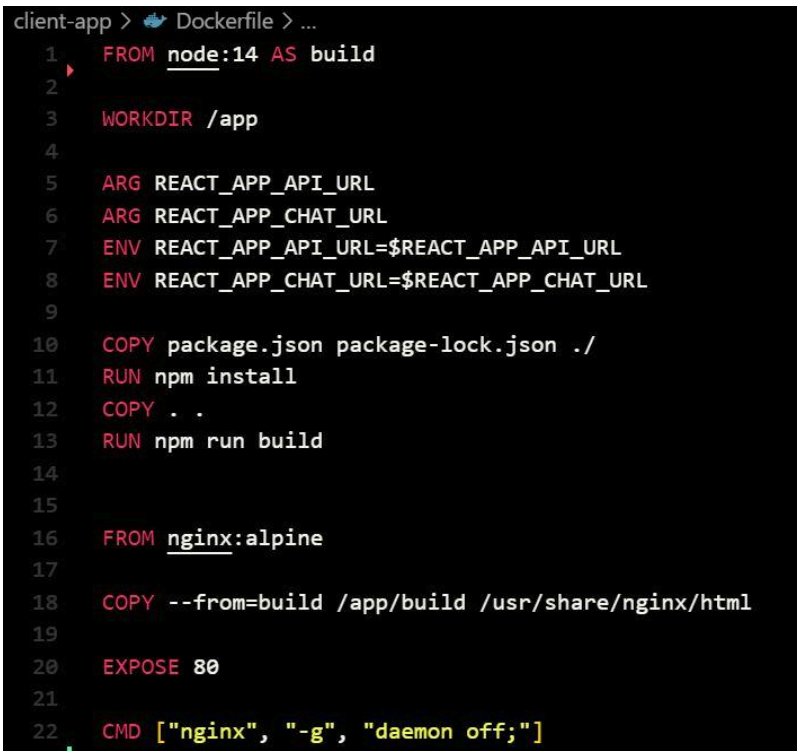


Рисунок 4.3 – Dockerfile FrontEnd сервісу (виконаний самостійно)

Оркестрація контейнерів за допомогою забезпечує автоматичне масштабування і управління додатками в контейнерах, що підвищує

надійність і доступність сервісу. Azure надає можливість автоматичного розподілу навантаження, самовідновлення та управління конфігураціями, що дозволяє ефективно використовувати ресурси і швидко реагувати на змінні вимоги. Наприклад, можливе автоматичне масштабування додатку в залежності від навантаження, забезпечуючи необхідну продуктивність навіть під час пікових навантажень. Це дозволяє забезпечити безперервну доступність веб-сервісу для користувачів житлових комплексів.

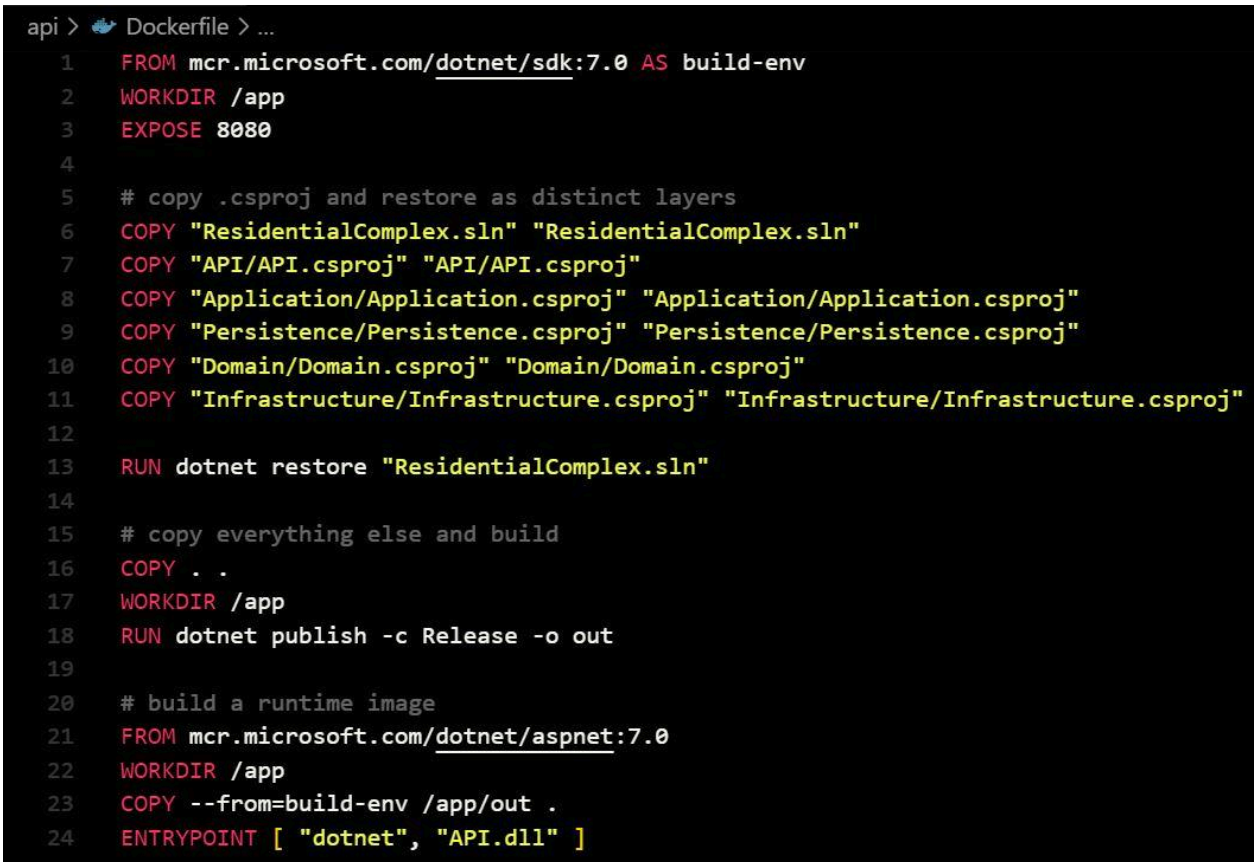


Рисунок 4.4 – Dockerfile BackEnd сервісу (виконаний самостійно)

Моніторинг і логування відіграють ключову роль у забезпеченні стабільної роботи системи. Використання інструментів для збору метрик і для їх візуалізації дозволяє відстежувати стан системи в реальному часі, виявляти та реагувати на проблеми на ранніх етапах. В результаті розробки були налаштовані відстеження метрик використання ресурсів, таких як

процесор, пам’ять, диск і мережа, а також метрик продуктивності додатка, таких як час відгуку і кількість запитів. Це буде дозволяти розробникам та адміністраторам оперативно реагувати на зміни у стані системи і виявляти потенційні проблеми до того, як вони вплинуть на кінцевих користувачів. Розроблений дашборд з метриками зображено на рисунку 4.5.

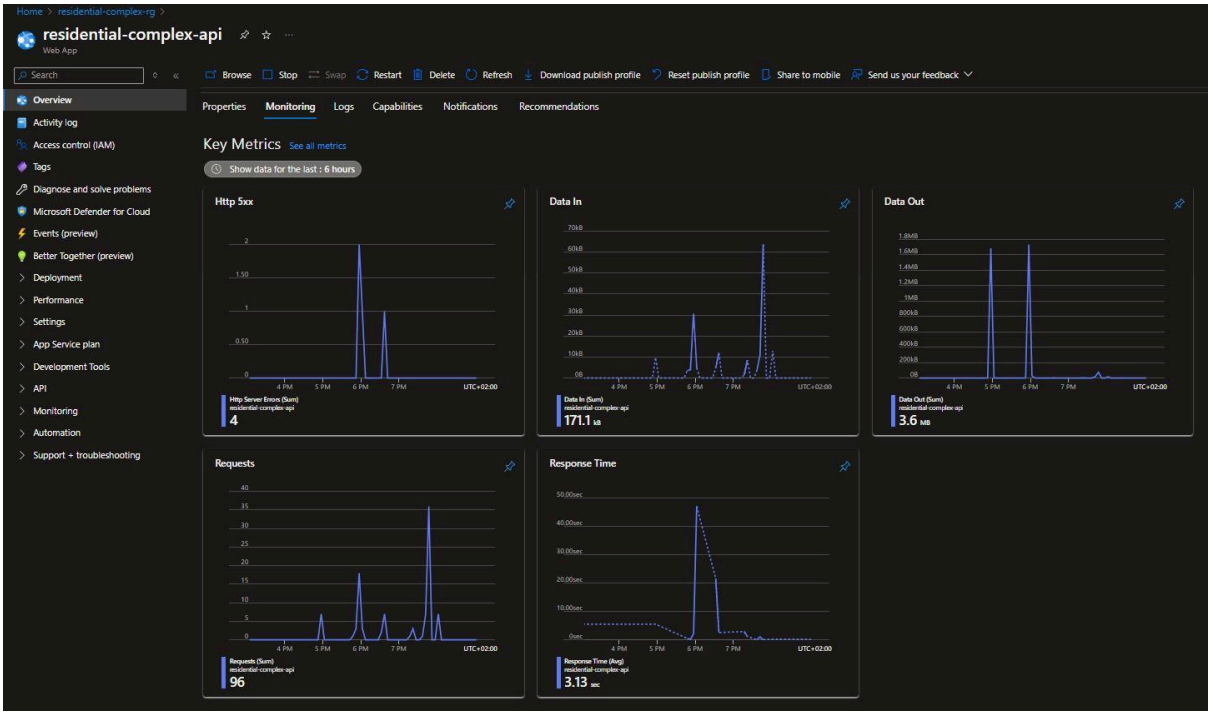


Рисунок 4.5 – Дашборд з метриками (виконаний самостійно)

Azure також забезпечує ефективне управління логами, дозволяючи збирати, аналізувати та візуалізувати журнали подій з різних компонентів системи, а також зберігати та індексувати їх для швидкого доступу. Це спрощує процес діагностики проблем і відстеження їх причин, а також обробляє та перетворює вхідні дані журналів.

4.2 Опис використаних рішень з розробки бази даних

З точки зору баз даних, вибір Azure SQL як основної СУБД обумовлений її високою надійністю, масштабованістю та інтеграцією з

іншими сервісами Azure. Azure SQL надає можливість створювати реляційні бази даних з високою продуктивністю і доступністю. Однією з головних переваг Azure SQL є її здатність автоматично масштабуватися у відповідь на змінні навантаження, що дозволяє підтримувати стабільну продуктивність навіть під час пікових навантажень. Azure SQL може автоматично збільшувати або зменшувати ресурси, виділені для бази даних, залежно від поточних вимог, що забезпечує оптимальне використання ресурсів і знижує витрати. Azure SQL також забезпечує високу безпеку даних завдяки вбудованим механізмам шифрування даних у стані спокою і під час передачі, а також підтримці керування доступом на основі ролей. База даних була розроблена відповідно до поставлених вимог (див. рис. 4.6) та з підтримкою усього необхідного функціоналу, що пропонує Azure.

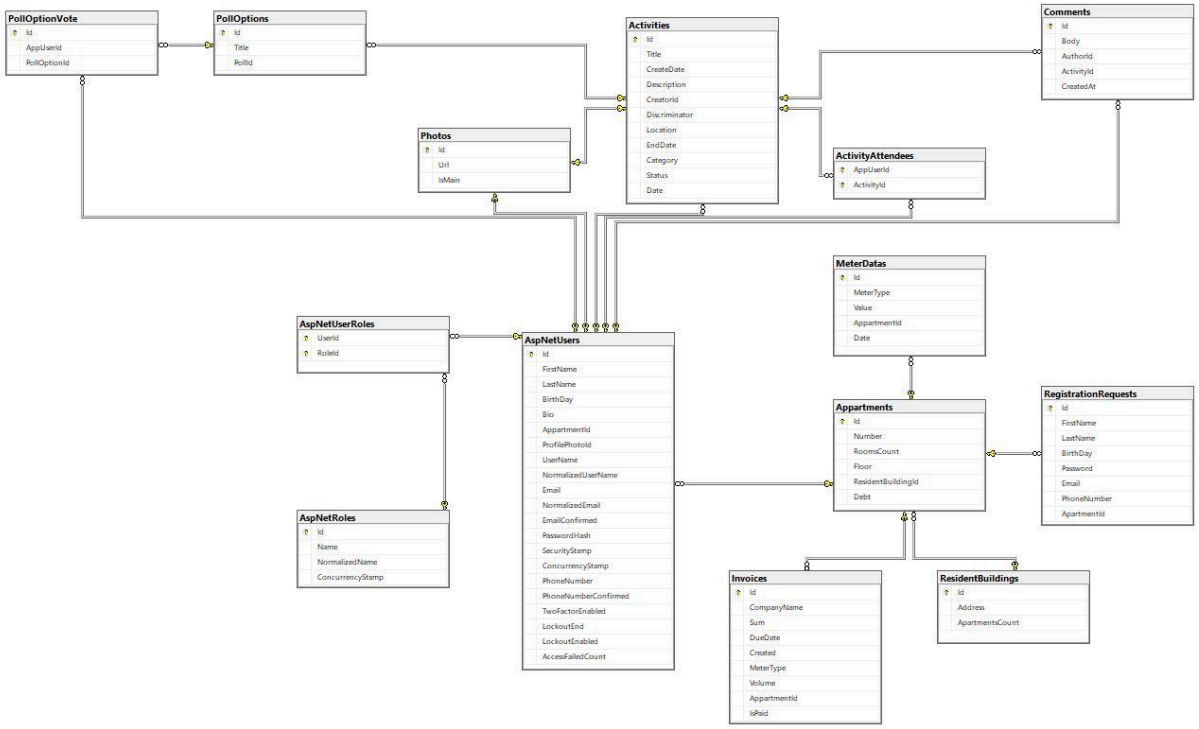


Рисунок 4.6 – Даталогічна модель розробленої бази даних (виконаний самостійно)

Для забезпечення цілісності та консистентності даних у базі використовується модель ACID. Це означає, що всі транзакції в базі даних виконуються як єдині цілісні операції, що забезпечує їх коректне виконання навіть у випадку збоїв. Використання індексів для часто використовуваних полів значно підвищує швидкість виконання запитів, що особливо важливо для забезпечення високої продуктивності сервісу під час обробки великих обсягів даних. Крім того, Azure SQL підтримує складні запити і аналітику в реальному часі, що дозволяє отримувати актуальні дані і швидко приймати рішення на основі цих даних.

Забезпечення безпеки бази даних включає налаштування прав доступу, шифрування даних, а також захист від SQL-ін'єкцій та інших видів атак [11, с. 62]. Використання GitHub Actions Secrets (див. рис. 4.7) для управління доступом дозволяє забезпечити централізоване управління користувачами і ролями, спрощуючи адміністрування і підвищуючи рівень безпеки. GitHub забезпечує багатофакторну аутентифікацію і підтримує політики доступу на основі умов, що дозволяє додатково захистити дані від несанкціонованого доступу. Крім того, впровадження практик безперервного моніторингу безпеки і регулярного аудиту дозволяє вчасно виявляти і усувати потенційні загрози. Наприклад, можна використовувати інструменти для виявлення аномальної активності та автоматичного реагування на підозрілі дії, що забезпечує проактивний підхід до захисту даних.

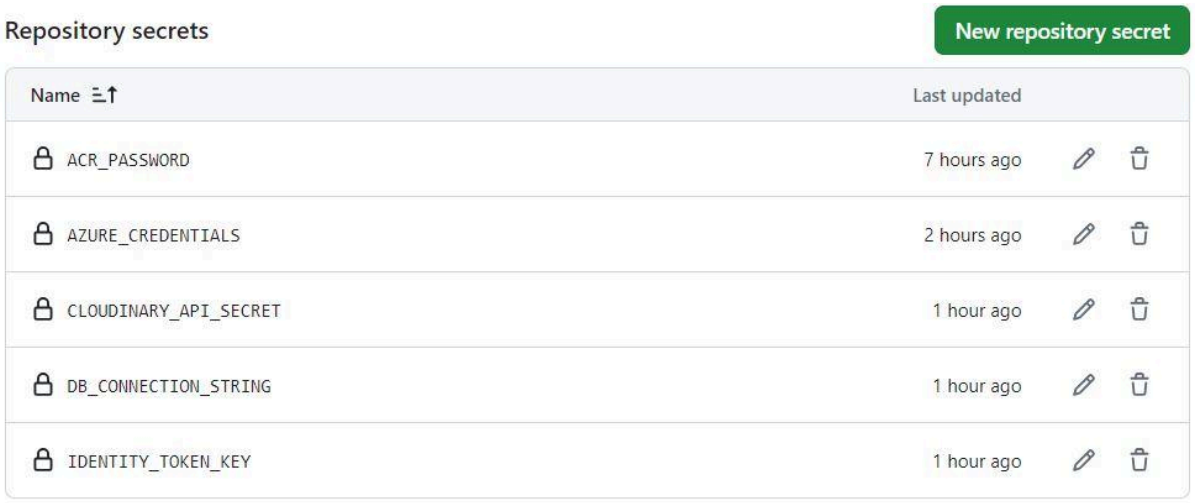


Рисунок 4.7 – Список токенів використаних у GitHub Actions Secrets (виконаний самостійно)

Таким чином, прийняті програмні рішення для веб-сервісу житлових комплексів включають використання GitHub Actions для автоматизації CI/CD процесів, контейнеризацію додатків з використанням Docker, моніторинг та логування за допомогою сервісів Azure, а також використання Azure SQL як основної бази даних з високою продуктивністю, надійністю та безпекою. Такий підхід забезпечує високу якість, надійність і масштабованість сервісу, що дозволяє ефективно обслуговувати житлові комплекси і надавати користувачам стабільний та безпечний доступ до необхідних функцій і даних.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Тестування розробленого програмного забезпечення

Тестування системи веб-сервісу для житлових комплексів з точки зору DevOps та Database розробки є критично важливим етапом, що охоплює широкий спектр завдань, спрямованих на забезпечення надійності, продуктивності та безпеки системи.

В рамках DevOps автоматизація тестування процесів є ключовим аспектом, що дозволяє швидко і ефективно виявляти помилки та недоліки на ранніх етапах розробки. Впровадження CI/CD пайплайну забезпечує автоматичне виконання тестів при кожному мерджі коду, що включає юніт-тести для перевірки окремих компонентів системи, інтеграційні тести для забезпечення коректної взаємодії між компонентами, та енд-то-енд тести, що моделюють дії кінцевого користувача і перевіряють роботу всієї системи в цілому. Такий підхід дозволяє мінімізувати ризик появи критичних помилок у продуктивному середовищі та забезпечує стабільність коду.

Контейнеризація з використанням Docker є ще одним важливим аспектом тестування в рамках DevOps. Вона дозволяє створювати ізольовані середовища для тестування, що допомагає уникнути проблем, пов’язаних з різними конфігураціями середовищ розробки, тестування та продуктивного середовища. Це особливо важливо в мікросервісній архітектурі, де кожен сервіс може бути протестований окремо або у взаємодії з іншими сервісами за допомогою симуляторів або моків. Такий підхід забезпечує гнучкість і масштабованість процесу тестування, дозволяючи легко адаптувати його до змін у системі.

Моніторинг і логування також є невід'ємною частиною процесу тестування. Важливо забезпечити безперервний моніторинг системи за

допомогою спеціалізованих інструментів, що дозволяють збирати і аналізувати метрики продуктивності та наявність помилок. Це допомагає виявляти проблеми, які можуть не проявитися в тестовому середовищі, але виникають під час реальної експлуатації. Аналіз логів дозволяє швидко знаходити і аналізувати помилки, що виникають у системі, забезпечуючи оперативне реагування на проблеми та їх швидке усунення. Це сприяє підвищенню надійності та стабільності сервісу.

З точки зору баз даних, тестування охоплює кілька важливих напрямків. Перевірка цілісності та консистентності даних є ключовою. Це включає тестування транзакцій для забезпечення їх атомарності, консистентності, ізольованості та довговічності. Використання тестових даних, які максимально наближені до реальних, допомагає виявити потенційні проблеми у структурі даних та запитах. Це включає перевірку правильності відносин між таблицями, коректності індексів, а також адекватності та ефективності виконання SQL-запитів. Також важливо тестувати механізми реплікації та кластеризації баз даних для забезпечення безперервної доступності даних і високої продуктивності системи.

Іншим критичним аспектом перевірки продуктивності бази даних є навантажувальне тестування [12]. Воно допомагає оцінити, як база даних буде працювати під високим навантаженням, та виявити "вузькі місця" у продуктивності запитів. Інструменти на зразок Apache JMeter або Gatling дозволяють створювати сценарії навантажувального тестування, що імітують різні умови експлуатації системи, такі як пікові навантаження або тривале високоефективне навантаження. Це допомагає зрозуміти, як система реагуватиме на різні рівні навантаження і які аспекти потрібно оптимізувати для покращення продуктивності.

В межах DevOps тестування веб-сервісу «Житловий комплекс» було сплановано та проведено тестування під навантаженням хмарних сервісів

бази даних та API. Для проведення тестування було використано сервіс Azure Load Testing та сконфігуровано сценарій високого навантаження на HTTP ендпоінти API та налаштовано моніторінг параметрів API сервісу та серверу бази даних.

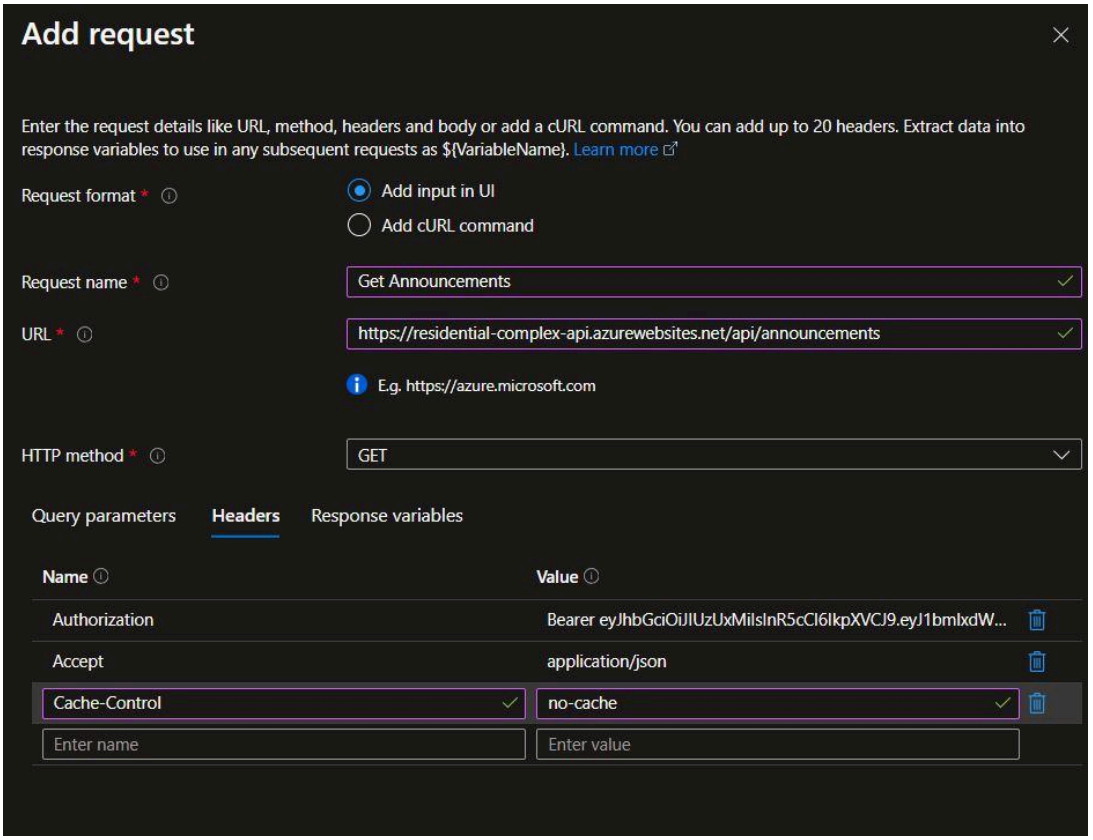


Рисунок 5.1 - Налаштування HTTP ендпоінта для тестування на навантаження (виконаний самостійно)

Тест вважається невдалим якщо середній час виконання запиту перевищує 3 секунди або більше ніж 50 відсотків запитів в певний проміжок часу повертають помилку. Для задоволення цих умов були створені тестові критерії (див. рис. 5.1).

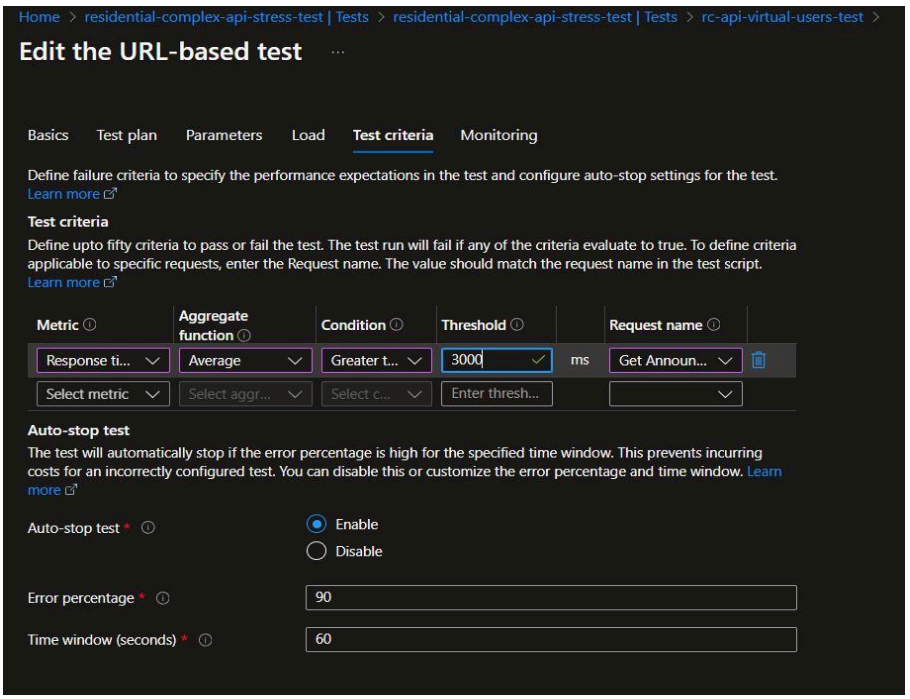
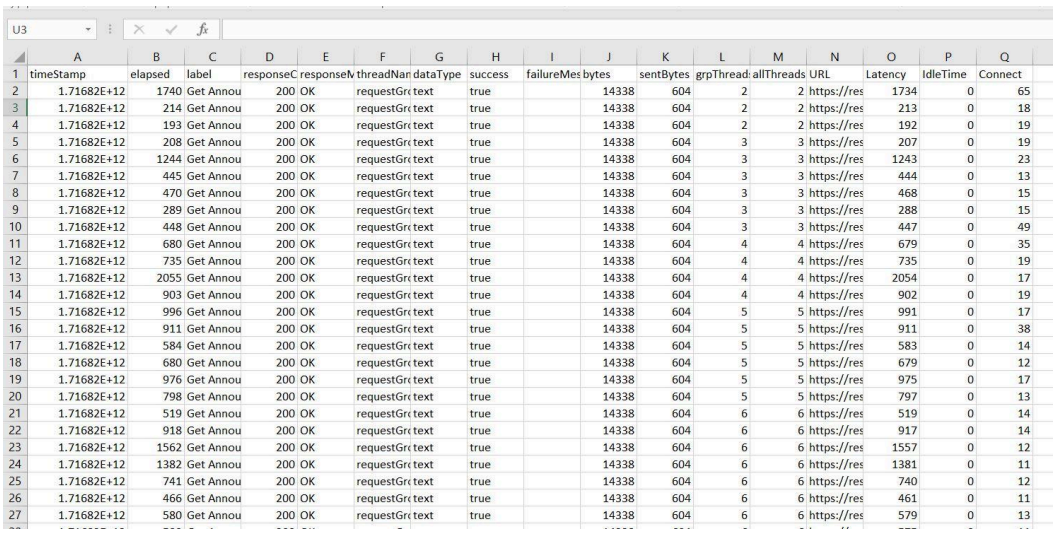


Рисунок 5.2 - Налаштування критеріїв для тестування на навантаження (виконаний самостійно)

Виконаний тест завершився вдало та згенерував подробні звіти по результатам запуску, на основі яких можна прийняти рішення щодо покращення продуктивності (див. рис. 5.3 та рис. 5.4)

Рисунок 5.3 - Результати тестування на навантаження. Метрики (виконаний самостійно)



U3

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	timeStamp	elapsed	label	responseC	responseM	threadNar	dataType	success	failureMes	bytes	sentBytes	grpThread	allThreads	URL	Latency	IdleTime	Connect
2	1.71682E+12	1740	Get Annou	200	OK	requestGr	text	true		14338	604	2	2	https://res	1734	0	65
3	1.71682E+12	214	Get Annou	200	OK	requestGr	text	true		14338	604	2	2	https://res	213	0	18
4	1.71682E+12	193	Get Annou	200	OK	requestGr	text	true		14338	604	2	2	https://res	192	0	19
5	1.71682E+12	208	Get Annou	200	OK	requestGr	text	true		14338	604	3	3	https://res	207	0	19
6	1.71682E+12	1244	Get Annou	200	OK	requestGr	text	true		14338	604	3	3	https://res	1243	0	23
7	1.71682E+12	445	Get Annou	200	OK	requestGr	text	true		14338	604	3	3	https://res	444	0	13
8	1.71682E+12	470	Get Annou	200	OK	requestGr	text	true		14338	604	3	3	https://res	468	0	15
9	1.71682E+12	289	Get Annou	200	OK	requestGr	text	true		14338	604	3	3	https://res	288	0	15
10	1.71682E+12	448	Get Annou	200	OK	requestGr	text	true		14338	604	3	3	https://res	447	0	49
11	1.71682E+12	680	Get Annou	200	OK	requestGr	text	true		14338	604	4	4	https://res	679	0	35
12	1.71682E+12	735	Get Annou	200	OK	requestGr	text	true		14338	604	4	4	https://res	735	0	19
13	1.71682E+12	2055	Get Annou	200	OK	requestGr	text	true		14338	604	4	4	https://res	2054	0	17
14	1.71682E+12	903	Get Annou	200	OK	requestGr	text	true		14338	604	4	4	https://res	902	0	19
15	1.71682E+12	996	Get Annou	200	OK	requestGr	text	true		14338	604	5	5	https://res	991	0	17
16	1.71682E+12	911	Get Annou	200	OK	requestGr	text	true		14338	604	5	5	https://res	911	0	38
17	1.71682E+12	584	Get Annou	200	OK	requestGr	text	true		14338	604	5	5	https://res	583	0	14
18	1.71682E+12	680	Get Annou	200	OK	requestGr	text	true		14338	604	5	5	https://res	679	0	12
19	1.71682E+12	976	Get Annou	200	OK	requestGr	text	true		14338	604	5	5	https://res	975	0	17
20	1.71682E+12	798	Get Annou	200	OK	requestGr	text	true		14338	604	5	5	https://res	797	0	13
21	1.71682E+12	519	Get Annou	200	OK	requestGr	text	true		14338	604	6	6	https://res	519	0	14
22	1.71682E+12	918	Get Annou	200	OK	requestGr	text	true		14338	604	6	6	https://res	917	0	14
23	1.71682E+12	1562	Get Annou	200	OK	requestGr	text	true		14338	604	6	6	https://res	1557	0	12
24	1.71682E+12	1382	Get Annou	200	OK	requestGr	text	true		14338	604	6	6	https://res	1381	0	11
25	1.71682E+12	741	Get Annou	200	OK	requestGr	text	true		14338	604	6	6	https://res	740	0	12
26	1.71682E+12	466	Get Annou	200	OK	requestGr	text	true		14338	604	6	6	https://res	461	0	11
27	1.71682E+12	580	Get Annou	200	OK	requestGr	text	true		14338	604	6	6	https://res	579	0	13

Рисунок 5.4 - Результати тестування на навантаження. Таблиця запитів (виконаний самостійно)

Також ретельну уваги необхідно приділити тестуванню резервного копіювання та відновлення даних, що має включати в себе регулярне створення резервних копій даних та перевірку процесу відновлення, щоб переконатися, що у разі збою є можливість відновлення даних швидко і повному обсязі. Така перевірка допомагає забезпечити, що механізми резервного копіювання функціонують належним чином і можуть бути використані для мінімізації втрат даних у разі непередбачених обставин. Важливо також перевірити стратегії зберігання резервних копій та їх безпеку, щоб захистити дані від несанкціонованого доступу.

Іншою невід'ємною частиною тестування бази даних є безпекове тестування, що включає в себе перевірку прав доступу, забезпечення захисту від SQL-ін'єкцій та інших видів атак. Важливо впровадити механізми шифрування даних як у стані спокою, так і під час передачі, а також налаштувати політику безпеки таким чином, щоб регулювати доступ до даних на основі ролей користувачів. Тестування на вразливості, проведення пенетраційних тестів та регулярний аудит безпеки

допомагають виявити та усунути потенційні загрози, забезпечуючи високий рівень захисту даних користувачів.

В циклі розробки веб-сервісу для житлових комплексів тестування на вразливості виконується за допомогою сканера вразливості Trivy (див. рис. 5.5)х в окремих кроках CI/CD пайплайну.

Рисунок 5.5 - Результати виконання сканування на вразливості API (виконаний самостійно)

Таким чином, тестування системи веб-сервісу для житлових комплексів з точки зору DevOps та баз даних є комплексним процесом, що включає автоматизацію тестування, моніторинг і логування, перевірку продуктивності та безпеки, а також забезпечення цілісності та надійності даних. Це дозволяє створити надійний, безпечний і високопродуктивний сервіс, здатний ефективно працювати в умовах реальної експлуатації. Виконання всіх цих заходів забезпечує, що система не лише відповідає поточним вимогам, але й здатна адаптуватися до змін та нових викликів, що можуть виникнути у майбутньому.

ВИСНОВКИ

В ході проєктування, розробки та тестування веб-сервісу для житлових комплексів було використано актуальні принципи DevOps, які включають в себе застосування сучасних інструментів і підходів дозволили досягти високої продуктивності, надійності і безпеки системи. GitHub Actions для автоматизації процесів CI/CD значно спрощує і прискорює розробку, тестування та розгортання нових версій додатка. Це рішення забезпечує безперервну інтеграцію та доставку, дозволяючи оперативно впроваджувати зміни і нові функції без ризику для стабільності системи. Автоматизація тестування різних рівнів дозволяє забезпечити високу якість коду і знизити ймовірність виникнення помилок у продуктивному середовищі.

Контейнеризація додатків з використанням Docker забезпечує ізольоване і консистентне середовище для розробки, тестування та розгортання. Це дозволяє усунути проблеми, пов’язані з різницею конфігурацій середовищ, і спрощує управління залежностями.

Моніторинг та логування за допомогою інструментів платформи Azure дозволяють відстежувати стан системи в реальному часі, виявляти і реагувати на проблеми на ранніх етапах. Це забезпечує кращий підхід до управління продуктивністю і безпекою системи, дозволяючи оперативно вирішувати проблеми і забезпечувати стабільну роботу сервісу. Візуалізація метрик і логів надає розробникам отримувати детальну інформацію про роботу системи, що спрощує діагностику і усунення проблем.

Використання Azure SQL як основної бази даних забезпечує високу продуктивність, масштабованість і безпеку даних. Автоматичне масштабування у відповідь на змінні навантаження дозволяє підтримувати стабільну продуктивність навіть під час пікових навантажень. Вбудовані

механізми шифрування даних і керування доступом на основі ролей забезпечують високий рівень безпеки даних. Резервне копіювання та можливість відновлення даних мінімізують ризик втрати інформації і забезпечують швидке відновлення у разі збоїв. Підтримка ACID-транзакцій гарантує цілісність і консистентність даних, що є критично важливим для надійної роботи системи.

Загалом, прийняті програмні рішення дозволяють забезпечити високу якість, надійність і безпеку веб-сервісу для житлових комплексів. Інтеграція сучасних DevOps практик і технологій управління базами даних створює міцну основу для ефективного управління і розвитку системи. Цей підхід дозволяє швидко адаптуватися до змінних вимог користувачів і ринку, забезпечуючи стабільну і безперебійну роботу сервісу. Впровадження цих рішень також сприяє зниженню операційних витрат і підвищенню ефективності роботи команди розробників, дозволяючи їм зосередитися на створенні нових функцій і покращенні користувацького досвіду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. DevOps / C. Ebert та ін. *IEEE Software*. 2016. Т. 33, № 3. С. 94–100. URL: https://doi.org/10.1109/ms.2016.68(дата звернення: 26.05.2024)

2. Condo Control Central Reviews. URL: https://www.getapp.com/real-estate-property-software/a/condo-control-central/reviews/ (дата звернення: 09.05.2024)

3. BuildingLink Reviews. URL: https://www.capterra.com/p/114529/BuildingLink/reviews/ (дата звернення: 09.05.2024)

4. Yardi Voyager Reviews. URL: https://www.getapp.com/real-estate-property-software/a/yardi-voyager/reviews/ (дата звернення: 09.05.2024)

5. Roy M. SQL: Learn Basics of Queries and Implement Easily(sql Database Programming, SQL 2016, SQL Beginners Guide, SQL Design Patterns, SQL Workbook, SQL Database, SQL Analytics, SQL Certification, SQL Data Analysis. Independently Published, 2019

6. How to build a CI/CD pipeline with GitHub Actions?. URL: https://github.blog/2022-02-02-build-ci-cd-pipeline-github-actions-four-steps/ (дата звернення: 09.05.2024)

7. What is Docker used for?. URL: https://www.freecodecamp.org/news/what-is-docker-used-for-a-docker-container-tutorial-for-beginners/ (дата звернення: 09.05.2024)

8. Azure Services: The Complete Guige. URL: https://www.splunk.com/en_us/blog/learn/microsoft-azure-services.html (дата звернення: 09.05.2024)

9. Köhler H., Link S. SQL schema design: foundations, normal forms, and normalization. *Information Systems*. 2018. Т. 76. С. 88–113

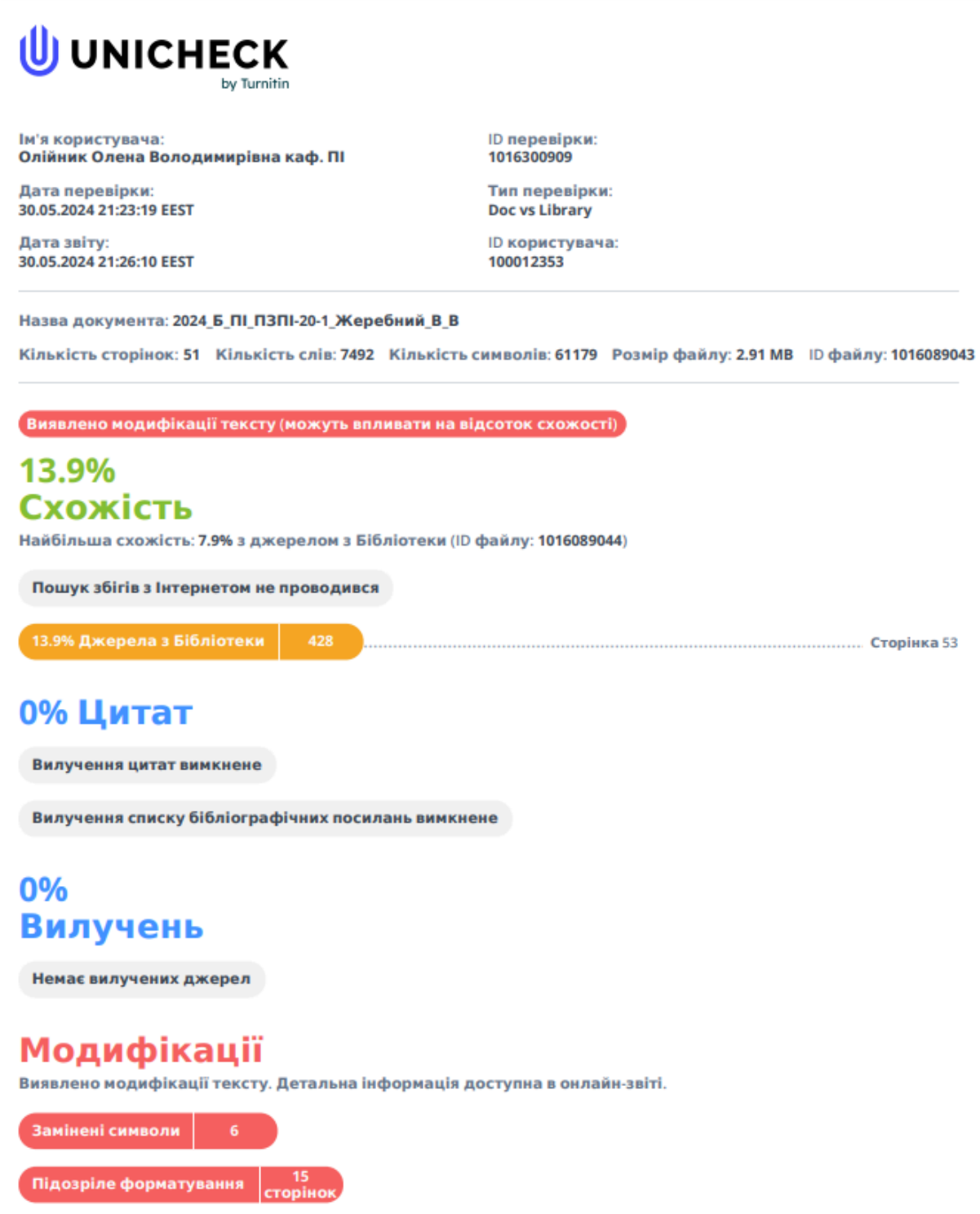
10. Ibrahim M. H., Sayagh M., Hassan A. E. A study of how Docker Compose is used to compose multi-component systems. *Empirical Software Engineering*. 2021. T. 26, № 6. URL: https://doi.org/10.1007/s10664-021-10025-1(дата звернення: 09.05.2024)

11. SQL Injection and Areas of Security Concern / S. Singh та ін. *International Journal of Computer Science and Mobile Computing*. 2021. T. 10, № 5. С. 60–66. URL: https://doi.org/10.47760/ijcsmc.2021.v10i05.006 (дата звернення: 09.05.2024)

12. Stress testing: A beginner's guide. URL: https://grafana.com/blog/2024/01/30/stress-testing/ (дата звернення: 09.05.2024)

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016300909

Дата перевірки:
30.05.2024 21:23:19 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
30.05.2024 21:26:10 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІ-20-1_Жеребний_В_В

Кількість сторінок: 51 Кількість слів: 7492 Кількість символів: 61179 Розмір файлу: 2.91 MB ID файлу: 1016089043

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

13.9%
Схожість

Найбільша схожість: 7.9% з джерелом з Бібліотеки (ID файлу: 1016089044)

Пошук збігів з Інтернетом не проводився

13.9% Джерела з Бібліотеки 428 ..... Сторінка 53

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.

Замінені символи 6

Підозріле форматування 15 сторінок

ДОДАТОК Б

Слайди презентації

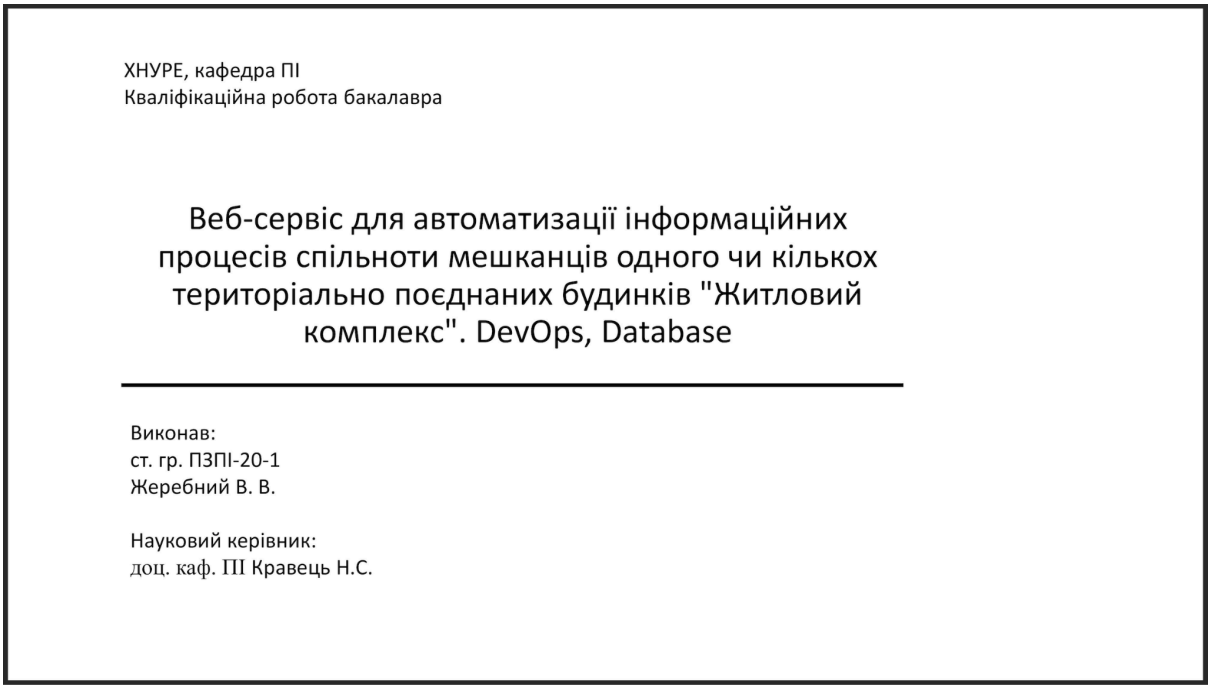


Рисунок Б.1 – Слайд 1 (рисунок виконаний самостійно)

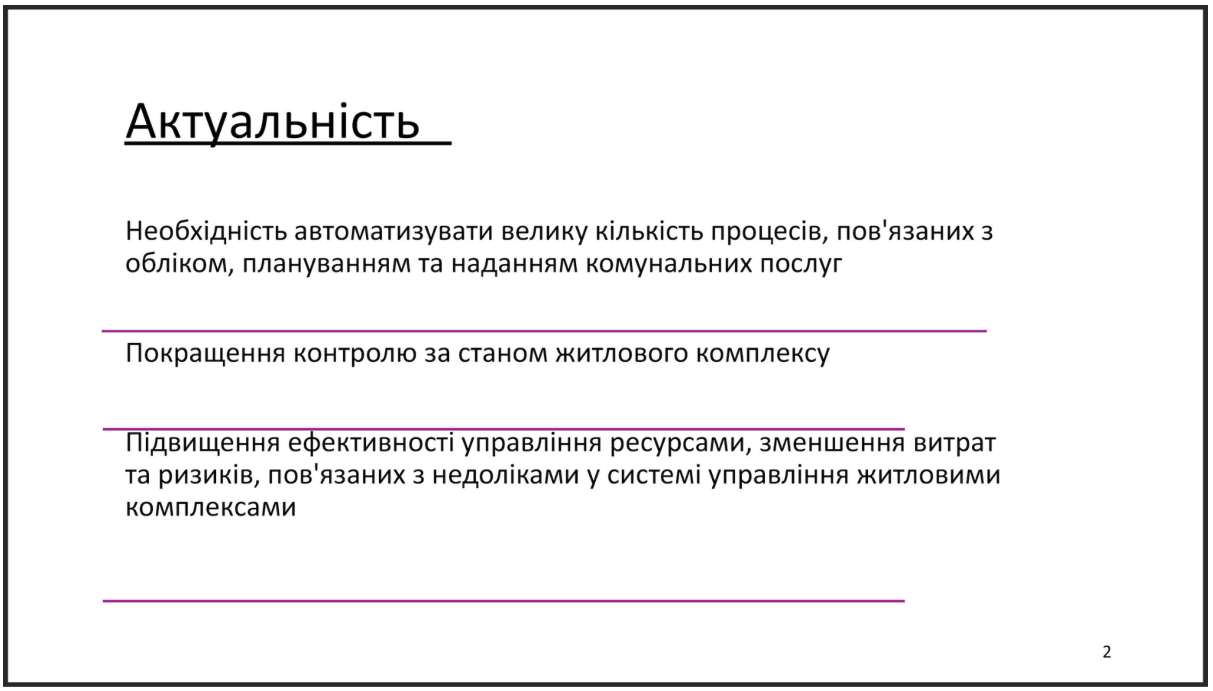


Рисунок Б.2 – Слайд 2 (рисунок виконаний самостійно)

Об’єкт роботи та Мета розробки

Об’єкт розробки – DevOps та Database частина веб-сервісу для автоматизації інформаційних процесів спільноти мешканців одного чи кількох територіально поєднаних будинків.

Мета розробки – створення бази даних з необхідними таблицями, контейнеризація BackEnd та FrontEnd сервісів, створення пайплайнів для побудови та розгортання веб-сервісу для житлових комплексів.

3

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

Постановка задачі

- Визначення вимог;
- Розробка архітектури системи;
- Налаштування системи безперервної інтеграції (CI) та безперервної доставки (CD);
- Створення та оптимізація реляційної бази даних;
- Впровадження заходів безпеки даних;
- Проведення регулярного тестування.

4

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

Чому саме Azure

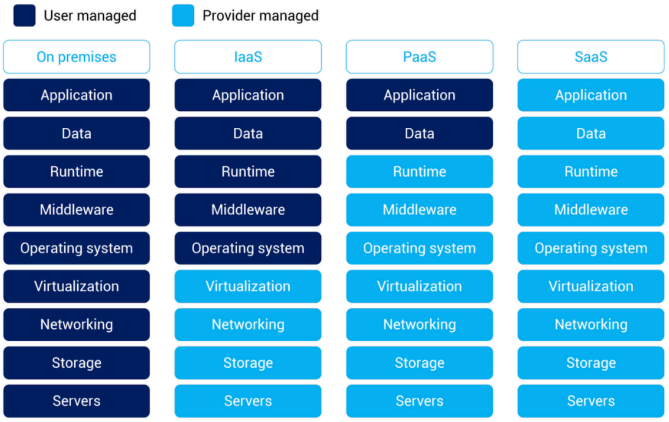
- Масштабованість
- Висока доступність і надійність
- Безпека
- Інтеграція з іншими сервісами
- Гнучкість у виборі технологій
- Керованість та автоматизація
- Економічна ефективність
- Підтримка та документація

5

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

PaaS

Переваги:
- ✓ Швидке розгортання
- ✓ Масштабованість
- ✓ Зниження витрат
- ✓ Автоматичне оновлення
- ✓ Безпека
- ✓ Інтеграція



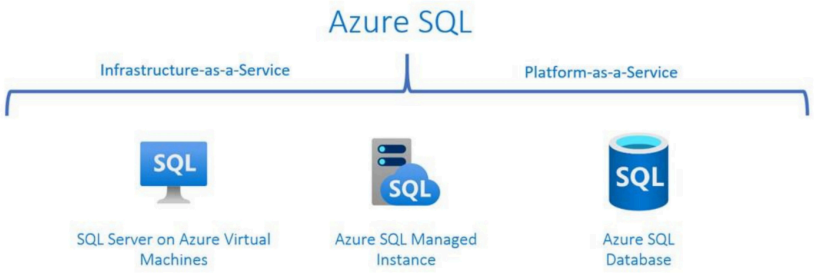
6

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

Azure SQL

Azure SQL Database - це керована реляційна база даних як послуга (PaaS) від Microsoft Azure

- Автоматичне оновлення та виправлення
- Резервне копіювання та відновлення
- Моніторинг продуктивності
- Масштабування ресурсів



7

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

Docker

Контейнеризація — це технологія, що дозволяє пакувати додаток і всі його залежності в ізольовані контейнери. Контейнери незалежні від базової інфраструктури.

8

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

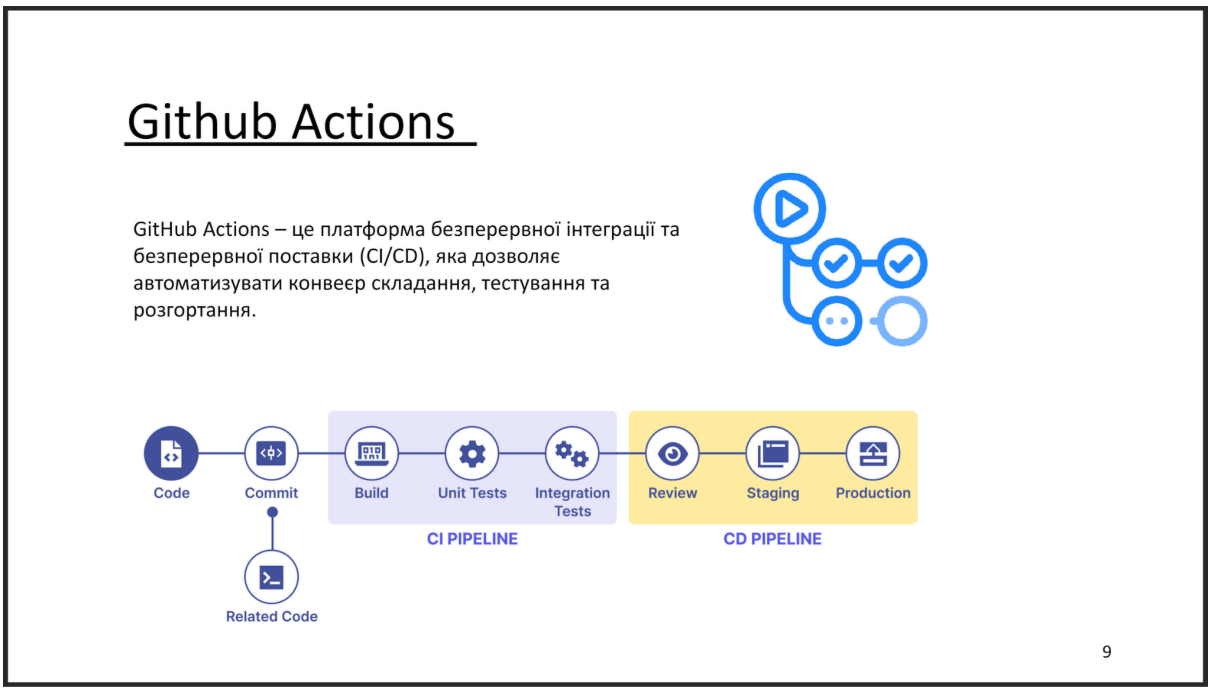


Рисунок Б.9 – Слайд 9 (рисунок виконаний самостійно)

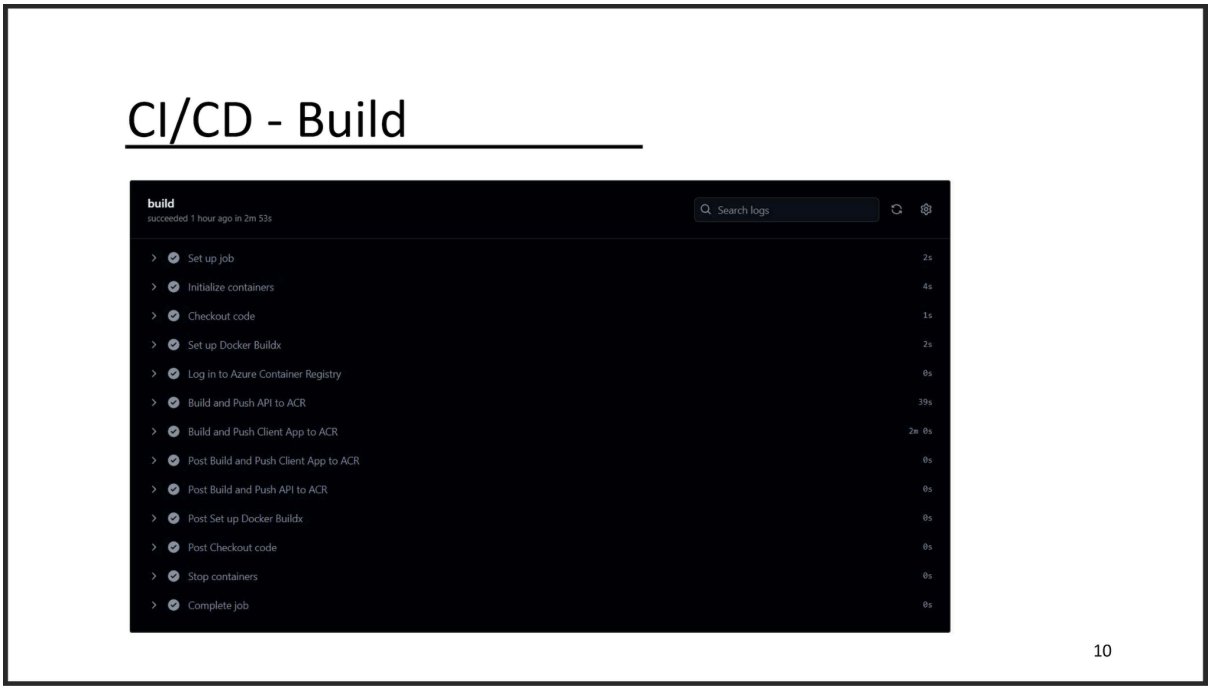
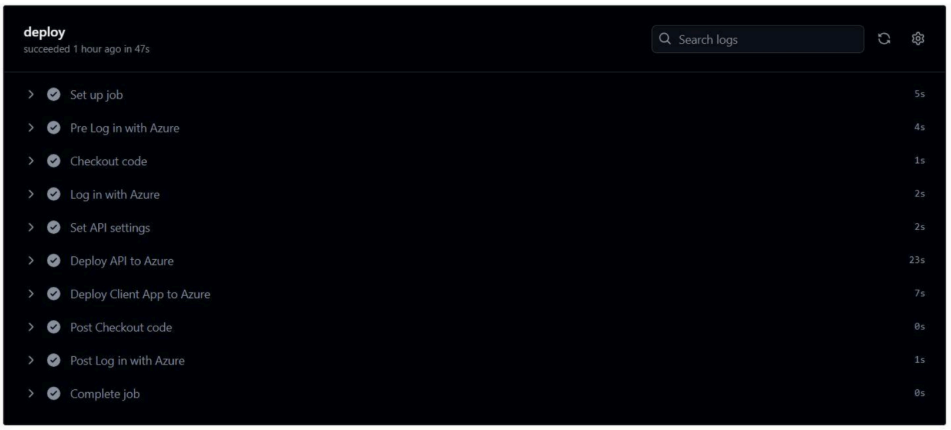


Рисунок Б.10 – Слайд 10 (рисунок виконаний самостійно)

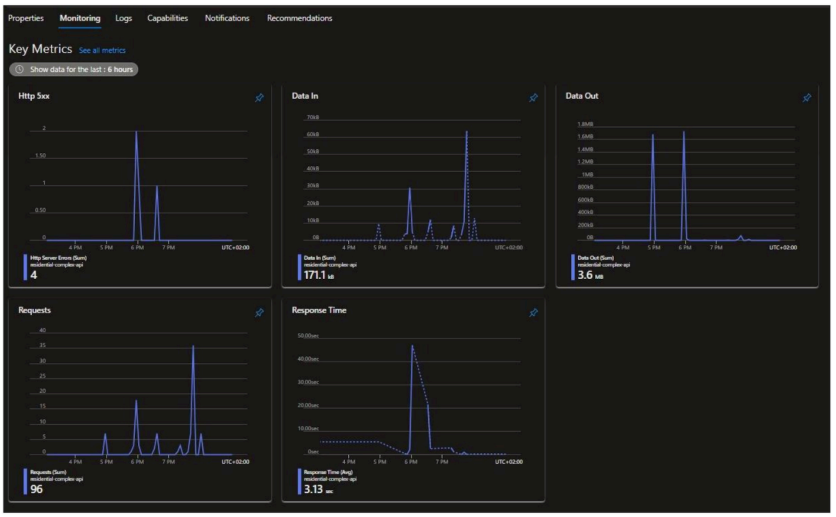
CI/CD – Deploy



11

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

Моніторінг



12

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

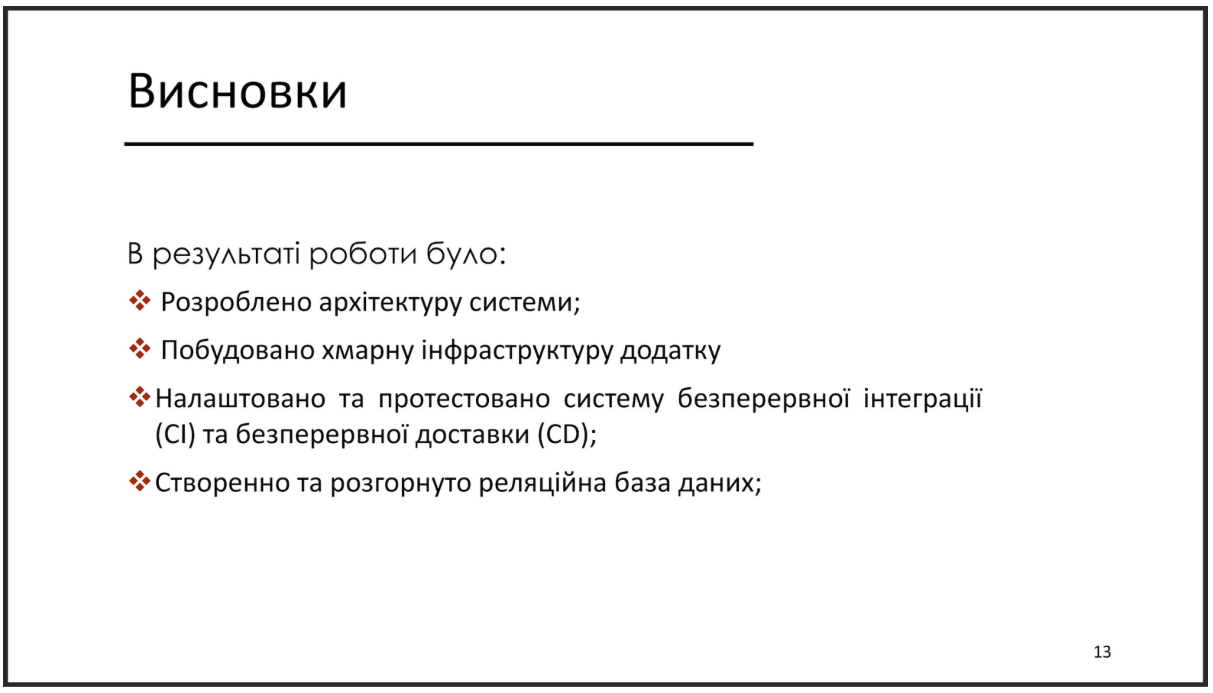


Рисунок Б.13 – Слайд 13 (рисунок виконаний самостійно)

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". DevOps/Database

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.