Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва інституту)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістра
(рівень вищої освіти)

на тему

Аналіз методів автоматизації та розробка застосунку для покращення ефективності та якості технічної підтримки

Виконав студент 6 курсу, групи 601-ТН
спеціальності
122 Комп'ютерні науки
Загнойко В.В.
(прізвище та ініціали)
Керівник Руденко О.А.
(прізвище та ініціали)
Рецензент Бублій В.Е
(прізвище та ініціали)

Полтава – 2025 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

спеціальність 122 «Комп'ютерні науки»

на тему

«Аналіз методів автоматизації та розробка застосунку для покращення ефективності та якості технічної підтримки»

Студента групи 601-ТН Загнойка Віталія Валерійовича

Керівник роботи
кандидат технічнтх наук,
доцент Руденко О.А.

Завідувач кафедри
кандидат фізико-математичних наук,
Двірна О.А.

Полтава – 2025

РЕФЕРАТ

Кваліфікаційна робота магістра: 91 с ., 28 рис., 14 табл., 38 джерело.

Об’єкт дослідження: аналіз потреб клієнтів та вибір оптимальних технологій та платформ для розробки.

Мета роботи: покращення ефективності взаємодії між клієнтами та службою підтримки, модернізація внутрішніх процесів. Створення прототипу для перевірки ключових функцій.

Методи: методика ітеративної розробки та прототипування з постійним аналізом вимог користувачів і їхніх відгуків.

Ключові слова: веб-застосунок, автоматизація, java, quarkus, якість, технічна підтримка, оптимізація процесів, інтеграція.

ABSTRACT

Explanatory note to the DR: 81 p., 28 fig., 14 tabl., 38 references.

Object of research: analysis of customer needs and selection of optimal technologies and platforms for development.

Purpose of work: improvement of efficiency of interaction between customers and support service, modernization of internal processes. Creation of a prototype for testing of key functions.

Methods: iterative development and prototyping methodology with constant analysis of user requirements and their feedback.

Keywords: web application, automation, java, quarkus, quality, technical support, process optimization, integration.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПК – персональний комп’ютер

ШІ – штучний інтелект

ПЗ – програмне забезпечення

БД – база даних

ООП – об’єктно орієнтоване програмування

DAO – data access object

ВСТУП

Цифрові технології сучасного світу швидко прогресують і модернізують бізнес-середовище, роль оперативної та якісної технічної підтримки неможливо переоцінити. Зростання вимог до якості клієнтського обслуговування та необхідність оптимізації внутрішніх процесів спонукають організації до впровадження нових рішень. Одним із ефективних способів є автоматизація через спеціальні веб-застосунки.

Мета дипломної роботи – створення та аналіз веб-застосунку для автоматизації процесів технічної підтримки та обслуговування клієнтів. Актуальність дослідження зумовлена необхідністю забезпечити організації ефективним інструментом, який сприятиме підвищенню рівня задоволеності клієнтів, зменшенню витрат на обслуговування.

Сучасні технології, такі як Java (фреймворк Quarkus), Cassandra DB, AngularJS дозволили створити веб-застосунок, що забезпечує обширні можливості ефективного вирішення клієнтських проблем.

Дипломна робота присвячена практичній реалізації, що включає проєктування та розробку веб-застосунку для автоматизації процесів технічної підтримки та обслуговування. У ході роботи здійснюється не лише технічна реалізація ідей, але й глибокий аналіз вимог клієнтів із подальшою адаптацією функціоналу.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ

Односторінковий опис задачі

Аналіз методів автоматизації та розробка застосунку для покращення ефективності та якості технічної підтримки.

Вступ. Назва проєкту «Аналіз методів автоматизації та розробка застосунку для покращення ефективності та якості технічної підтримки».

Дата підготовки документа: 20 грудня 2024 р.

Основна мета – покращення ефективності взаємодії між клієнтами та службою підтримки, модернізація внутрішніх процесів.

Опис характеристик поставки. На ПК попередньо встановлено операційну систему Ubuntu 22.04 LTS, програмне забезпечення, компілятор та інтерпретатор для Java. Програмний продукт має бути розміщений на ПК, що забезпечує локальний доступ до нього за умови наявності необхідних файлів на жорсткому диску.

Цільова аудиторія додатку. Головними юзерами веб-додатку є працівники технічної підтримки, користувачі, що вимагають оперативного та ефективного розв’язання своїх запитів.

Порівняльний аналіз. Порівнюючи з наявними ринковими рішеннями, цей застосунок пропонує більш гнучкий підхід до управління запитами, що сприяє покращення якості та ефективності обслуговування.

Опис технічного процесу. Процес охоплює аналіз потреб клієнтів, вибір оптимальних технологій та платформ для розробки, а також створення прототипу для перевірки ключових функцій. Важливою складовою є тестування веб-застосунку, щоб гарантувати відповідність вимогам.

Список документів: Звіт дипломної роботи.

Основні дати: Початок розробки – 10 жовтня 2024р., завершення

розробки – 20 листопада 2024р.

Опис задачі

Автоматизація набуває все більшого значення в різних галузях промисловості, зокрема в технічній підтримці та обслуговуванні клієнтів. Впровадження автоматизованих процесів може суттєво покращити ефективність і результативність, допомагаючи організаціям забезпечувати високий рівень обслуговування та задоволення потреб клієнтів [1].

Для обслуговування клієнтів, сервісу та технічної підтримки автоматизація має низку переваг. Перше, вона дозволяє оптимізувати процеси, прискорюючи обробку запитів і скорочення часу відповіді. Друге, мінімізація помилок та невідповідності, властиві ручним процесам, що підвищує точність обслуговування і зменшує рівень незадоволеності клієнтів. Крім того, автоматизовані системи сприяють зростанню продуктивності, виконуючи рутинні завдання, та не відволікати команду від зосередження на стратегічно важливих питаннях. Нарешті, такі системи забезпечують доступ до аналітики в реальному часі, що дає змогу організаціям приймати обґрунтовані рішення основуючись на актуальних даних.

Концептуальні завдання дипломної роботи передбачають дослідження сучасних методів автоматизації у сфері технічної підтримки, створення концептуальної моделі застосунку, враховуючи структуру, ключові функції, формування основних вимог до застосунку, таких як функціональність, зручність інтерфейсу, безпека та швидкодія.

Технічні завдання передбачають вибір і налаштування технологічного стеку для розробки, проєктування та реалізації бекенду й фронтенду застосунку з подальшою їх інтеграцією. Вони також включають створення системи управління заявками, розробку інструментів для звітності та аналізу запитів, забезпечення високої продуктивності й стабільної роботи застосунку,

впровадження заходів для підвищення безпеки, а також тестування функціональних можливостей, інтерфейсу користувача та продуктивності веб-застосунку.

Мета дипломної роботи полягає в розробці веб-застосунку для автоматизації процесів технічної підтримки, обслуговуванні клієнтів та сервісу. Застосунок покликаний забезпечити ефективну та безперебійну роботу для клієнтів, та для команд підтримки, автоматизуючи ключові функції, зокрема, управління заявками, моніторинг проблем і організацію взаємодії.

Основні завдання дипломної роботи полягають в удосконаленні процесу обробки запитів до технічної підтримки, обслуговування клієнтів, сервісу, підвищенні ефективності команд підтримки і постійного удосконалення процесів. Створенні програмного рішення, здатного масштабуватися для покриття зростаючих потреб у технічній підтримці та обслуговуванні клієнтів.

У дипломній роботі застосовуються різні технології та методи, зокрема, розробка програмного забезпечення, аналіз даних, задля відповідності програми потребам і вимогам клієнтів та команд підтримки.

РОЗДІЛ 2
ЛІТЕРАТУРНИЙ ОГЛЯД

2.1 Аналіз поточних досліджень у сфері технічної підтримки та автоматизації обслуговування клієнтів

Останніми роками сфера автоматизації технічної підтримки та обслуговування клієнтів привернула значну увагу в дослідженнях, так як організації намагаються покращити ефективність процесів підтримки. Автоматизація має потенціал для вдосконалення цих процесів, зменшення часу відповіді та поліпшення взаємодії з клієнтами [2]

Поточні дослідження у сфері технічної підтримки та автоматизації обслуговування клієнтів показують, що організації все активніше використовують технології для підвищення ефективності своїх операцій. Основні напрямки досліджень включають.

- Автоматизація процесів підтримки. Дослідження підтверджують, що автоматизація значно скорочує час реагування та підвищує продуктивність. Це включає впровадження чат-ботів, автоматизованих систем маршрутизації запитів, а також інструментів для автоматичного аналізу та вирішення проблем.
- Системи управління знаннями (KMS). Створення автоматизованих баз знань для агентів підтримки дозволяє швидко знаходити необхідну інформацію, що знижує час на пошук і підвищує точність відповідей.
- Інтеграція з аналітикою в реальному часі. Використання аналітики для оцінки ефективності технічної підтримки та автоматизації процесів допомагає організаціям приймати обґрунтовані рішення щодо покращення обслуговування клієнтів.
- Машинне навчання та штучний інтелект. Використання цих технологій для аналізу великих обсягів даних допомагає передбачити потреби клієнтів, покращити взаємодію та автоматизувати рутинні завдання.

- Мобільні платформи та мультиканальність. Сучасні дослідження також зосереджуються на розробці та вдосконаленні мобільних застосунків та мультиканальних платформ для підтримки клієнтів, що дозволяє забезпечити безперервну комунікацію через різні канали (телефон, чат, соціальні мережі тощо).

Загалом, поточні дослідження підтверджують, що автоматизація є важливим інструментом для підвищення ефективності технічної підтримки та обслуговування клієнтів, а інтеграція новітніх технологій дозволяє покращити точність, швидкість і якість обслуговування.

Аналіз літератури виявляє потенційні переваги впровадження автоматизації у технічну підтримку та обслуговування клієнтів, можливі виклики, з якими можуть зіткнутися організації під час реалізації процесів. Аналіз підтверджує, що автоматизація здатна суттєво покращити ефективність підтримки, одночасно удосконалити якість взаємодії з клієнтами [3].

Росте зацікавленість до використання штучного інтелекту в сфері підтримки, що підкреслює його важливу роль та вплив. Унікальні можливості ШІ можуть відчутно покращити ефективність підтримки, забезпечуючи агентів актуальними рекомендаціями та інформацією в реальному часі. Однак разом із перспективами удосконалення ШІ виникають і певні виклики. Зокрема, необхідно враховувати ризик збереження поточних упереджень у роботі систем штучного інтелекту. Впровадження ШІ вимагає ретельного аналізу етичних моментів, аби гарантувати рівень відповідальності у процесах підтримки [4].

Аналіз літератури висвітлює розмаїття досліджень у сфері підтримки та обслуговуванні клієнтів, окреслює переваги та виклики, зв’язані з впровадженням. Це створює важливу базу розробки застосунку, спрямованого на автоматизацію процесів технічної підтримки.

2.2 Новітні тренди та інноваційні підходи в галузі

Сучасні тренди та інноваційні рішення в автоматизації технічної підтримки спрямовані на покращення ефективності, швидкості та персоналізації обслуговування клієнтів. Основні напрямки розвитку включають.

1. Використання штучного інтелекту (ШІ).
   - Чат-боти та віртуальні помічники на базі ШІ стають дедалі популярнішими завдяки здатності автоматично обробляти прості запити та навчатися на основі взаємодій з користувачами.
   - Системи на основі ШІ допомагають прогнозувати потреби клієнтів і пропонувати оптимальні рішення в реальному часі.
2. Роботизація процесів (RPA).
   - Інструменти RPA автоматизують повторювані завдання, такі як створення та маршрутизація заявок, що дозволяє звільнити час для складніших завдань.
3. Інтеграція мультиканальних платформ.
   - Рішення, які об’єднують різні канали комунікації (телефон, чат, електронна пошта, соціальні мережі), забезпечують клієнтам зручний доступ до підтримки та зберігають єдиний досвід обслуговування.
4. Системи самообслуговування.
   - Розробка інтуїтивних платформ самообслуговування, таких як інтерактивні бази знань і FAQ, дозволяє клієнтам швидко знаходити відповіді без втручання агентів підтримки.
5. Аналіз даних у реальному часі.
   - Інноваційні аналітичні інструменти відстежують і аналізують продуктивність процесів підтримки, допомагаючи приймати обґрунтовані рішення для покращення сервісу.
6. Гіперперсоналізація обслуговування.

- Використання великих даних та машинного навчання для адаптації підтримки під конкретні потреби клієнтів на основі їхньої історії взаємодій.

7. Етичні аспекти автоматизації:
   - Розробка рішень із врахуванням питань прозорості, відповідальності та зменшення ризику упереджень у системах ШІ.

Однією з актуальних тенденцій є розвиток систем управління знаннями, які забезпечують доступом до централізованого сховища інформації та найкращих практик. Це сприяє швидшому та ефективнішому розв'язанню запитів, а також покращує якість обслуговування [5].

Хмарні системи підтримки набувають значної популярності серед організацій завдяки їх високій гнучкості та масштабованості, які дозволяють задовольняти різноманітні потреби клієнтів. Вони здатні швидко адаптуватися до змін у попиті та забезпечують доступ з будь-якого місця. Це допомагає організаціям надавати узгоджену та ефективну підтримку незалежно від їхнього розташування [6].

2.3 Вплив автоматизації на робітничу силу

Одним із центральних питань, пов'язаних з автоматизацією технічної підтримки та обслуговування клієнтів, є її вплив на робочу силу. З одного боку, автоматизація підвищує ефективність, дозволяючи працівникам брати участь у вирішенні більш складніших завдань і покращуючи якість обслуговування клієнтів. З іншого боку, вона може призвести до скорочення робочих місць, оскільки машини беруть на себе виконання рутинних завдань.

Вплив автоматизації на робітничу силу є багатогранним і має як позитивні, так і негативні аспекти.

Позитивні аспекти.

- Підвищення ефективності. Автоматизація дозволяє оптимізувати рутинні процеси, звільняючи працівників для виконання складніших і творчих завдань.
- Розвиток навичок. Робітники мають можливість розвивати нові компетенції, спрямовані на управління та вдосконалення автоматизованих систем.
- Зростання продуктивності. Автоматизація сприяє прискоренню процесів, що підвищує загальну продуктивність і покращує результати роботи.

Негативні аспекти.

- Скорочення робочих місць. Виконання стандартних і повторюваних завдань перекладається на машини, що може призвести до втрати робочих місць.
- Вимоги до перекваліфікації. Зростає потреба в нових знаннях і вміннях, що може бути викликом для працівників, не готових до змін.
- Ризик соціальної нерівності. Зміни в структурі зайнятості можуть поглиблювати розрив між висококваліфікованими працівниками та тими, чия праця легко автоматизується.

Таким чином, автоматизація створює можливості для розвитку, але вимагає стратегічного підходу до перекваліфікації працівників та адаптації організацій до нових умов.

Слід підкреслити, що застосування автоматизаційних технологій у сфері технічної підтримки та обслуговування клієнтів невпинно зростає. Завдяки прогресу в галузях штучного інтелекту (AI) та машинного навчання (ML) створюються все більш досконалі та ефективні автоматизовані системи. Вони здатні обробляти численні запити клієнтів одночасно, надавати підтримку в режимі реального часу та використовувати прогнозну аналітику для виявлення та усунення потенційних проблем ще до їхнього виникнення [7].

Аналіз літератури свідчить про зростаючу популярність автоматизаційних технологій технічної підтримки та обслуговування клієнтів.

Хоча автоматизація приносить як значні переваги, так і певні виклики, організаціям важливо уважно оцінювати її вплив на робочу силу та забезпечувати ретельне планування для плавного переходу до автоматизованих систем підтримки [8].

2.4 Важливість вимірювання ефективності рішень автоматизації

Оцінка ефективності рішень автоматизації має ключове значення для забезпечення їхньої успішної реалізації та оптимізації. Основні аспекти, які підкреслюють важливість цього процесу.

1. Визначення досягнення цілей. Вимірювання дозволяє оцінити, чи автоматизація сприяє досягненню встановлених цілей, таких як підвищення продуктивності, скорочення витрат чи покращення якості обслуговування.
2. Підтримка прийняття рішень. Чіткі метрики ефективності надають дані, необхідні для ухвалення обґрунтованих рішень щодо продовження, розширення чи модифікації автоматизованих систем.
3. Виявлення проблемних зон. Регулярна оцінка допомагає визначити аспекти автоматизації, які працюють недостатньо ефективно, та впровадити відповідні коригування.
4. Оцінка окупності інвестицій (ROI). Вимірювання ефективності дозволяє оцінити економічну вигоду від впроваджених рішень, що особливо важливо для обґрунтування подальших інвестицій.
5. Покращення взаємодії з клієнтами. Аналіз результатів автоматизації допомагає зрозуміти, як зміни впливають на досвід клієнтів, і в разі потреби коригувати стратегії для його покращення.
6. Підтримка інновацій. Постійний моніторинг ефективності сприяє впровадженню нових технологій і вдосконалень, які відповідають змінюваним потребам організації та клієнтів.

Отже, систематичне вимірювання ефективності рішень автоматизації є важливим інструментом для їх успішного впровадження, оптимізації та подальшого розвитку.

Підсумовуючи огляд літератури, присвячений вимірюванню ефективності рішень автоматизації, слід закцентувати увагу на необхідності регулярно оцінювати результати та вплив своїх ініціатив у цій сфері. Це включає аналіз впливу на робочу силу, фінансові показники та якість обслуговування й підтримки клієнтів. Використовуючи належні підходи та інструменти, організації можуть використати позитивні сторони автоматизації, забезпечуючи клієнтам високий рівень сервісу та підтримки [9].

2.5 Java

Java займає провідну позицію серед сучасних технологічних рішень у сфері розробки програмного забезпечення. Завдяки своїй універсальності, надійності та масштабованості, ця мова активно використовується для створення різноманітних програмних продуктів, включаючи веб-додатки, мобільні застосунки та корпоративні системи. Її популярність пояснюється також широкою підтримкою спільноти розробників та постійним удосконаленням екосистеми. Java не лише є потужним інструментом для розробки, але й виступає важливим об’єктом для досліджень у сфері програмування.

Корпоративні застосунки.

Java є однією з провідних мов програмування для розробки корпоративних застосунків завдяки своїй надійності, масштабованості та багатій екосистемі. Вона забезпечує можливість створення складних багаторівневих систем, що можуть обробляти великі обсяги даних і підтримувати високі навантаження.

Основні причини популярності Java в корпоративному середовищі.

- Платформна незалежність. Можливість запуску застосунків на різних операційних системах завдяки JVM (Java Virtual Machine).
- Широкий вибір фреймворків. Інструменти, такі як Spring і Hibernate, спрощують розробку, тестування та розгортання програм.
- Безпека. Вбудовані механізми захисту даних та підтримка сучасних стандартів безпеки роблять її ідеальною для корпоративних потреб.
- Масштабованість. Java добре підходить для систем, які повинні адаптуватися до зростання бізнесу та збільшення користувачів.

Ці переваги роблять Java незамінним вибором для створення ERP-систем, CRM-рішень, фінансових платформ і хмарних сервісів.

Серверні застосунки і веб-розробка.

Java виступає однією з основних технологій у веб-розробці. Завдяки фреймворкам, таким як Spring, Micronaut і Quarkus, розробники можуть швидко й ефективно створювати гнучкі та масштабовані веб-застосунки. Зараз активно впроваджуються REST-контролери, які служать основою для серверних застосунків та відповідають за обробку клієнтських запитів.

Мобільна розробка

Java є однією з основних мов програмування для мобільної розробки, зокрема для створення додатків для платформи Android. Завдяки своїй стабільності, безпеці та високій продуктивності, Java залишається популярним вибором серед розробників мобільних додатків.

Наукові та дослідницькі проєкти

Java активно використовується в наукових та дослідницьких проєктах завдяки своїй стабільності, високій продуктивності та багатій екосистемі. Вона є популярною мовою програмування у таких сферах, як обробка великих обсягів даних, машинне навчання, штучний інтелект та обчислювальні науки.

Завдяки своїй гнучкості, широким можливостям для паралельних обчислень, великій кількості доступних бібліотек і інструментів, Java є

важливим інструментом для наукових і дослідницьких проєктів у багатьох галузях науки та техніки.

Наразі Java використовує модель випуску, що включає версії з довгостроковою підтримкою (LTS) та короткострокові релізи. LTS-версії підтримуються протягом тривалішого періоду, гарантуючи бізнес-середовищам стабільність і сумісність. Зазвичай нові LTS-версії виходять кожні два-три роки.

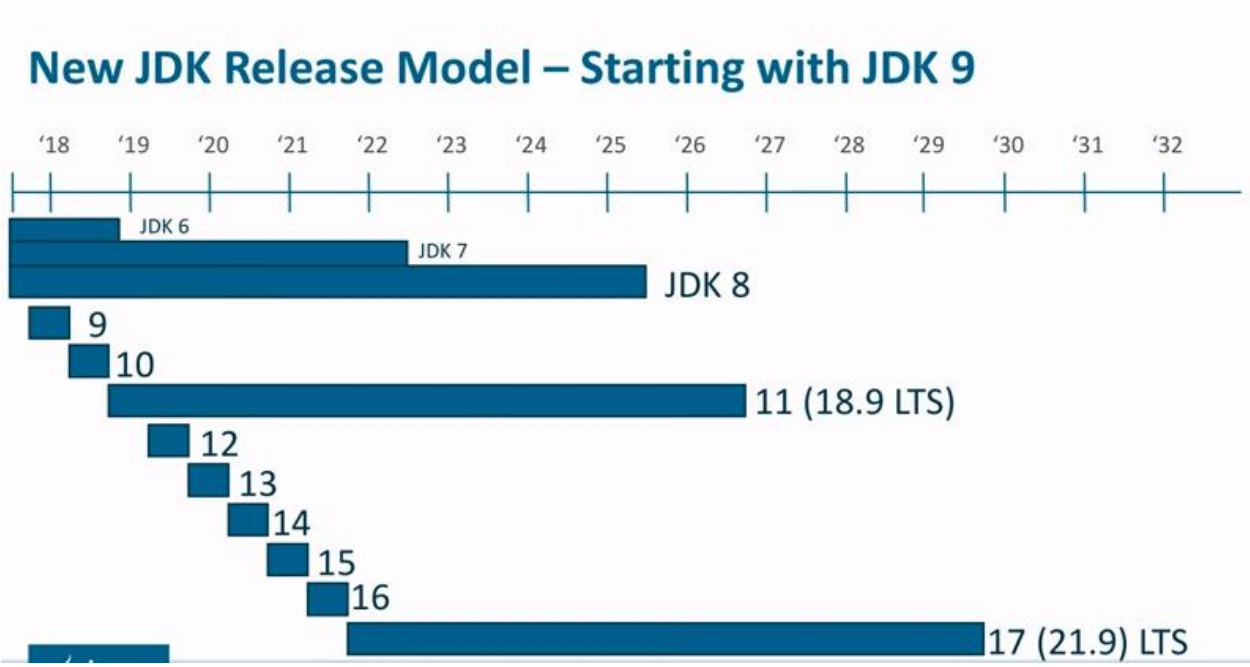


Рисунок 2.1 – План по випуску нових версій Java

На рис. 2.1 представлено графік виходу нових версій Java та тривалість їхньої підтримки. З нього видно, що LTS-версії мають значно триваліший період підтримки. Також помітно, що після випуску Java 8 змінився підхід до релізів. До Java 8 нові версії з'являлися кожні 2-3 роки та мали статус LTS. Починаючи з Java 9, стратегія змінилася: тепер виходить кілька проміжних версій, що дозволяє швидше впроваджувати нові функції мови. Водночас LTS-версії, як і раніше, з'являються з інтервалом у 2-3 роки.

Крім того, регулярно виходять короткострокові версії, які включають нові функції та покращення. Це дозволяє швидше впроваджувати інновації та реагувати на зміни ринку. Підтримка таких версій є обмеженою, тому розробникам рекомендується переходити на LTS-версії для забезпечення тривалої стабільності.

Важливим аспектом є активна участь спільноти та залучення розробників у процес розробки, що дозволяє Java швидко реагувати на зміни технологічного ландшафту та вдосконалювати якість реалізації мови.

Java Virtual Machine (JVM) є основою платформи Java, яка забезпечує виконання програмного коду незалежно від операційної системи та апаратного забезпечення. Хоча JVM створена для запуску програм, написаних мовою Java, вона також підтримує інші мови програмування. Завдяки цій можливості JVM стає універсальним середовищем для розробки різноманітних програм, даючи розробникам свободу вибору мови залежно від потреб проєкту.

Основні мови, що працюють на JVM.

1. Java. Базова мова для JVM, яка забезпечує високу продуктивність, стабільність та багату екосистему бібліотек.

2. Scala. Поєднує об’єктно-орієнтовану та функціональну парадигми, дозволяючи створювати компактний і виразний код. Часто використовується для розробки систем обробки даних та великих розподілених систем.

3. Kotlin. Сучасна мова з акцентом на продуктивність і безпеку. Вона стала офіційною мовою для розробки Android-додатків і популярною в розробці серверних застосунків.

4. Groovy. Динамічна мова з простою синтаксичною структурою, яка інтегрується з Java та часто використовується для створення скриптів, автоматизації процесів і тестування.

5. Clojure. Функціональна мова програмування, яка базується на Lisp. Вона забезпечує високу підтримку багатопоточності та часто використовується для створення конкурентних систем.

6. Jruby. Реалізація Ruby для JVM, яка дозволяє запускати Ruby-програми у середовищі JVM.

7. Jython. Реалізація Python для JVM, що забезпечує інтеграцію програм Python із бібліотеками Java.

8. Apache Groovy. Легка мова програмування, що доповнює Java,

дозволяючи швидко створювати сценарії та прототипи.

Переваги використання мов на JVM.

- Платформонезалежність. Програми, написані на будь-якій з мов для JVM, можуть виконуватись на будь-якому пристрої, що підтримує JVM.
- Взаємодія. Мови на JVM можуть легко взаємодіяти між собою та з бібліотеками Java.
- Гнучкість. Широкий вибір мов дозволяє вибрати ту, яка найкраще відповідає потребам конкретного проєкту.
- Масштабованість. JVM забезпечує високу продуктивність і масштабованість для великих проєктів.

JVM залишається потужною платформою для багатопрофільної розробки, завдяки чому екосистема мов, що працюють на ній, постійно розширюється.

Кросплатформенність є однією з визначальних характеристик мови програмування Java, яка забезпечує можливість виконання програмного коду на будь-якому пристрої або операційній системі, де встановлена Java Virtual Machine (JVM). Ця властивість дозволяє створювати програми, що незалежно від апаратної чи програмної платформи залишаються стабільними та ефективними.

Основою кросплатформенності є JVM, яка виконує байт-код, згенерований під час компіляції програми. Такий підхід дає змогу абстрагуватись від особливостей конкретних платформ, забезпечуючи високу переносимість коду.

Особливо важливою кросплатформенність є для сучасних технологій, таких як мобільні пристрої та Інтернет речей (IoT), де різноманітність апаратного забезпечення та операційних систем значно ускладнює розробку. Можливість використання єдиного коду на різних платформах заощаджує ресурси, скорочує час розробки та полегшує подальшу підтримку програмного забезпечення.

Java також поєднує кросплатформенність із об’єктно-орієнтованою архітектурою, що робить її не лише гнучким, але й зручним інструментом для створення сучасних програмних рішень.

ООП в Java передбачає використання об’єктів, які є екземплярами класів, і забезпечує високий рівень перевикористання коду та легкість його розширення. Це особливо важливо для проєктів з великою кодовою базою, де чітка структура та розділення функціональності допомагають зберігати код організованим і легко зрозумілим.

Гнучкість Java виявляється в здатності адаптуватися до змінних вимог індустрії. Багато років активного розвитку та підтримки спільноти розробників дозволяють швидко впроваджувати нові технології та моделі програмування без суттєвих змін у вже наявному коді.

Використання гнучкості та об’єктно-орієнтованого підходу в Java сприяє ефективній розробці застосунків, спрощує процес тестування та підтримує програмний продукт актуальним навіть в умовах постійних змін ринку та технологій.

Постійне оновлення мови та розширення стандартної бібліотеки демонструють безперервний розвиток екосистеми Java. Це надає розробникам доступ до сучасних технологій і функціональності, дозволяючи їм зберігати конкурентоспроможність у динамічному світі програмної розробки.

Java використовує синтаксис, подібний до C-подібного, що значно сприяє її популярності та легкості засвоєння серед розробників. Така схожість має низку переваг: програмісти з досвідом у C, C++ чи C# швидше освоюють Java завдяки знайомій структурі. Це робить процес навчання простішим і полегшує адаптацію новачків до роботи з мовою.

Для новачків у програмуванні, зокрема тих, хто починав з мови C, вивчення Java може бути більш природнім та ефективним процесом через подібний синтаксис. Узагальнюючи, схожий до C синтаксис в Java створює

комфортні умови для широкого кола розробників, забезпечуючи високий рівень доступності та придатності для великої аудиторії програмістів.

2.6 Quarkus

Quarkus – це сучасний фреймворк для Java, який розроблений для побудови високопродуктивних, масштабованих та хмарних додатків. Він оптимізований для роботи в контейнерах, таких як Docker та Kubernetes, і надає значні переваги в порівнянні з традиційними Java-фреймворками завдяки швидкому старту, низькому споживанню пам’яті та підтримці функціональних можливостей для мікросервісної архітектури.

Quarkus був представлений компанією Red Hat у 2019 році як відповідь на запити, які виникають через зростання популярності мікросервісної архітектури та запити хмарних застосунків. Назва "Quarkus" поєднує слова "Quark" (елементарна частка) та "Us" (ми), підкреслюючи легкість і колективний підхід до розробки.

Одна з основних цілей Quarkus – відмовитися від концепції "важких" серверів, орієнтуючись на забезпечення легкості, швидкості та масштабованості для розробників у програмному середовищі. Цей фреймворк став революційним інструментом для веб-розробників, котрі шукають гнучкі рішення з хорошою ефективністю для створення мікросервісів і хмарних застосунків.

Філософія Quarkus зосереджена на наданні розробникам інструментів для створення високопродуктивних, масштабованих і ефективних мікросервісів і хмарних застосунків. Основні принципи фреймворку.

1. Швидкість запуску та малий розмір: Quarkus оптимізований для швидкого старту і малих розмірів додатків, що є важливими характеристиками для розгортання в контейнерах і хмарних середовищах. Це забезпечує ефективне використання ресурсів і зменшує час, необхідний для запуску.

2. Інтеграція з контейнерами та Kubernetes: Quarkus побудований з урахуванням сучасних хмарних архітектур, зокрема орієнтуючись на роботу з контейнерами та оркестрацією через Kubernetes. Це дає змогу розробникам швидко створювати додатки, які легко масштабуються і інтегруються з хмарними платформами.

3. Нативна компіляція: Один з основних принципів фреймворку – це підтримка компіляції до нативного коду, що дозволяє створювати додатки з мінімальними вимогами до пам’яті та швидким запуском. Завдяки цьому можна досягти суттєвого прискорення роботи програми.

4. Мінімалізм та продуктивність: Quarkus сприяє створенню легких, швидких додатків без зайвих накладних витрат, оптимізуючи використання ресурсів на всіх етапах – від розробки до виконання.

5. Розширюваність через доповнення: Quarkus підтримує розширення через доповнення (extensions), які дозволяють інтегрувати різноманітні технології та бібліотеки без необхідності додавати великі об’єми коду. Це забезпечує гнучкість і дозволяє підлаштовувати фреймворк під конкретні потреби проєкту.

Філософія Quarkus є адаптацією до нових вимог сучасної розробки, орієнтуючись на швидкість, масштабованість і ефективність, що робить його ідеальним вибором для створення хмарних і мікросервісних рішень.

Quarkus здобув популярність серед великих корпорацій завдяки своїй легкості та високій продуктивності. Успішні впровадження цього фреймворку в корпоративних системах підтверджують його ефективність у великих проектах. Компанія Red Hat активно використовує Quarkus у своїх продуктах і сервісах, що забезпечує високу продуктивність та прискорену розробку.

Quarkus отримує високу оцінку від великих корпоративних систем за здатність без проблем інтегруватися з уже існуючими технологіями, забезпечуючи високу продуктивність і надійність навіть у середовищах з

високими навантаженнями. Зростаюче використання Quarkus у таких проектах підтверджує його значний вплив на корпоративні технологічні рішення.

Загалом користувачі Quarkus відзначають, що цей фреймворк значно прискорює процес розробки, забезпечуючи високу продуктивність і легкість інтеграції з іншими інструментами. Відгуки часто акцентують на вражаючій швидкості запуску та виконання застосунків, що робить Quarkus привабливим вибором для великих проектів і навантажених систем. Такі відгуки допомагають вдосконалювати Quarkus, враховуючи потреби та побажання користувачів у майбутніх версіях.

Майбутнє Quarkus передбачає подальший активний розвиток і розширення його функціональних можливостей, спрямованих на підвищення ефективності та гнучкості. Експерти прогнозують, що фреймворк активно інтегруватиме технології, таких як обробка подій (event-driven programming) і розширення підтримки безсерверних (serverless) архітектур.

Прогнозується покращення підтримки Java-екосистеми, яке підтримуватиме нові версії Java, глибшу інтеграцію, покращення інструментів для налагодження та аналізу, що спростить роботу розробників.

В цілому, розширення можливостей та підвищення продуктивності, орієнтовані на вимоги користувачів і тенденції, створюють перспективи для Quarkus у сфері розробки програмного забезпечення.

РОЗДІЛ 3
МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ

Методологія – це система принципів, підходів та методів, яка застосовується для організації та здійснення досліджень, проектів чи процесів. Вона включає в себе вибір інструментів, методів збору даних, аналізу, а також структурні етапи для досягнення бажаного результату.

У цьому розділі надано огляд методології дослідження, застосованої в проєктуванню та аналізі веб-застосунку, що має на меті автоматизацію технічної підтримки, сервісного обслуговування та підтримки клієнтів. Методологія включає в себе дослідження, збір даних та підходи до їх аналізу, які були використані.

Розробка включає застосування фреймворку Quarkus для бекенду, бази даних Cassandra та використання AngularJS в фронтенді. Цей набір технологій забезпечує надійність та ефективність для створення рішення, що підвищує взаємодію з користувачем і оптимізує процес сервісу.

Застосовуючи підхід, який поєднує аналітичні та технічні методи, методологія дає повне розуміння сфери автоматизації технічної підтримки. Також підтримує ітеративну модернізацію програми через постійний фідбек з користувачами, що допомагає розробці користувацько орієнтованого рішення.

3.1 Вивчення дизайну та методів розробки.

У цьому підрозділі представлено підхід до розробки та аналіз, що були обрані для створення та оцінки застосунку, орієнтованого на покращення ефективності та якості технічної підтримки. План дослідження поєднує як аналітичні, так і технічні методи, що дозволяє забезпечити комплексне розуміння підходу.

Використання аналітичного підходу прогнозує детальне вивчення поточних тенденцій та викликів автоматизації підтримки. Цей етап забезпечує

відомості для створення ефективного рішення, яке відповідає вимогам реальних умов.

Прикладний підхід доповнює дослідження, орієнтуючи ітераційне проєктування, розробку та тестування застосунку. Цей процес постійного вдосконалення на основі відгуків користувачів забезпечує, щоб кінцеве рішення задовольнило вимоги.

Аналітичний етап цього дослідження охоплює детальний аналіз поточних тенденцій та проблем у сфері технічної підтримки та сервісу. Визначення основні характеристики, можливості та потреби, важливі для створення застосунку.

Вивчаючи літературу, звіти та практики, попередній аналіз висвітлює найважливіші аспекти автоматизації підтримки та сервісу. Ця інформація стане основою для проєктування та розробки веб-застосунку [10].

Орієнтуючись на результати аналізу, прикладний підхід спрямовує ітеративний процес дизайну, розробки та тестування застосунку. Це дозволяє постійно вдосконалювати продукт на основі зворотного зв’язку від користувачів, що забезпечує ефективне вирішення реальних проблем та відповідність вимогам.

Ітеративний процес базується на принципах адаптивної розробки, що підкреслює ітеративність і швидке реагування на змінювані вимоги. Такий підхід дозволяє вдосконалювати дизайн, функціональні можливості та взаємодію з користувачем, спираючись на відгуки, отримані від усіх команд учасників процесу.

Комбінація пошукових і прикладних дослідницьких методів сприяє створенню рішення, орієнтованого на користувача, яке об’єднує висновки, отримані під час пошукового етапу, з адаптивним процесом розробки. Такий підхід забезпечує, що фінальний веб-застосунок не лише базується на глибокому аналізі, а й постійно вдосконалюється, аби відповідати потребам.

Інтеграція специфічних технологій відіграє ключову роль у дослідницькому дизайні та підході до розробки веб-застосунку. Застосування Quarkus для бекенду, Cassandra DB для управління даними та AngularJS для фронтенду демонструє прагнення використовувати сучасні технології для забезпечення автоматизації.

Quarkus було обрано за його високу продуктивність, ефективність і зручність у розробці, що ідеально відповідає завданням створення гнучкого та ефективного застосунку. Cassandra DB зі своєю масштабованістю та функціоналом NoSQL забезпечує оптимальне рішення для управління та зберігання даних, пов’язаних з технічною підтримкою. AngularJS пропонує динамічний та інтуїтивно зрозумілий фронтенд, що сприяє плавній взаємодії між користувачами та додатком.

Аналіз підходу та дизайну до розробки виступає картою, що спрямовує систематичне створення та оцінювання застосунку. Етап пошукового дослідження надає цінне розуміння аспектів автоматизації підтримки та сервісу, забезпечуючи основу для прийняття обґрунтованих рішень у процесі розробки [11].

3.2 Аналіз та збір даних

В розділі розглянуто підходи до збору та аналізу даних, застосовані для отримання та обробки інформації під час розробки й оцінювання додатку, призначеного для автоматизації процесів сервісу та обслуговування.

Збір даних включав кілька методів. Виконано огляд літератури, що охоплював наукові статті, практики та галузеві звіти [12]. Проведено опитування серед користувачів, працівників технічної підтримки та експертів у цій галузі. Це дозволило зібрати кількісні дані про їхні вподобання, труднощі й очікування. Отримані дані аналізувалися з метою виявлення ключових закономірностей, ідей і тенденцій. Якісну інформацію з інтерв’ю та відкритих запитань піддано контент-аналізу, щоб визначити основні теми, настрої та

потреби користувачів. Дані з опитувань і метрик використання оброблено статистичними методами для оцінки продуктивності додатку, вподобань та рівня задоволеності користувачів. Поєднання якісного й кількісного аналізу забезпечило всебічне розуміння потреб користувачів і сприяло формуванню інформованого підходу до розробки та оцінки веб-додатку [13].

3.3 Розробка та тестування прототипу

Розглянемо процес створення прототипу та застосований підхід до тестування під час розробки веб-додатку для автоматизації тех. підтримки та клієнтського обслуговування. Розробка додатку здійснювалася з особливою увагою до вибору технологій: для бекенду використовувався фреймворк Quarkus, для зберігання даних – Cassandra DB, а для фронтенду – AngularJS. Реалізація базувалася на гнучкій методології, що забезпечувала можливість ітеративного вдосконалення за рахунок врахування фідбеку від користувачів, технічних експертів.

Завдяки Quarkus і DataStax Java Driver створення бекенду стала простішою. Вони забезпечують надійну роботу коду, швидкий запуск та низьке використання пам’яті. Cassandra DB з її розподіленою архітектурою дозволила безперебійно зберігати й отримувати дані. Можливості AngularJS для побудови фронтенду покращили взаємодію з користувачем.

Під час створення прототипу регулярно проводилися юзабіліті-тести, для переконання, що застосунок покриває потреби користувачів і ефективно виконує свої функції. Тести покриваля кейси, які відображали реальні ситуації взаємодії з обслуговуванням клієнтів.

Прототип також пройшов детальне тестування продуктивності для оцінки швидкодії, продуктивність та гнучкості. Ці перевірки включали навантаження на систему, моделювали різні рівні користувацької активності, що дозволило визначити здатність застосунку підтримувати стабільну продуктивність у

різних умовах. Це забезпечило його надійну роботу без компромісів щодо функціональності [14].

3.4 Математична модель M/M/1 черги

Математична модель M/M/1 черги описує систему обслуговування з одним сервером, де надходження заявок (клієнтів) і обслуговування описуються пуассонівськими процесами. Ця модель є базовою в теорії масового обслуговування і позначається як M/M/1, де:

1. M (Markovian arrival process) – час між надходженнями заявок є експоненційно розподіленим.
2. M (Markovian service process) – час обслуговування заявок також є експоненційно розподіленим.
3. 1 – єдиний сервер для обслуговування заявок.

Основні припущення моделі.

1. Вхідний потік заявок є стаціонарним і слідує пуассонівському процесу з параметром λ (середня кількість заявок за одиницю часу).
2. Час обслуговування кожної заявки має експоненційний розподіл з параметром μ (середня кількість заявок, що обслуговуються за одиницю часу).
3. Обслуговування здійснюється за принципом "перший прийшов – перший обслуговується" (FIFO).
4. Черга може бути нескінченною.

Основні характеристики системи M/M/1:

1. Коефіцієнт завантаження:

$$\rho = \frac{\lambda}{\mu} \tag{3.1}$$

Це відношення інтенсивності надходження заявок (λ) до інтенсивності обслуговування (μ). Для стабільної роботи системи має виконуватися умова: ρ<1.

2. Середня кількість заявок у системі (черга + обслуговування)

$$L = \frac{\rho}{1-\rho} \tag{3.2}$$

3. Середня кількість заявок у черзі:

$$L_q = \frac{\rho^2}{1-\rho} \tag{3.3}$$

4. Середній час перебування заявки в системі:

$$W = \frac{1}{\mu-\lambda} \tag{3.4}$$

5. Середній час очікування в черзі:

$$W_q = \frac{\rho}{\mu-\lambda} \tag{3.5}$$

6. Ймовірність того, що в системі знаходиться n заявок:

$$P_n = (1-\rho)\rho^n, n \geq 0 \tag{3.6}$$

7. Ймовірність того, що система порожня (немає заявок):

$$P_0 = 1-\rho \tag{3.7}$$

Ця модель є корисною для аналізу систем обслуговування, таких як телефонні центри, веб-сервери, виробничі лінії тощо. Вона дозволяє оцінити продуктивність системи, виявити вузькі місця і приймати рішення для оптимізації.

Клас ProjectStatisticService служить інструментом збирання, обробки та аналізу даних, пов’язаних із проєктами, у системах керування проєктами або відстеження помилок. Його основне призначення – використовувати дані для обчислення основних параметрів моделі черг M/M/1, яка є базовою моделлю в теорії масового обслуговування [16].

На рис. 3.1 представлено клас ProjectStatisticService, що включає набір методів, вони виконують специфічні ролі у зборі та аналізі статистики. До складу класу входять публічні методи, які забезпечують API для взаємодії з іншими класами, та приватні методи [17].

```
    public void calculateStatisticsForCurrentWeek(String projectId, boolean currentTime) { ... }

    public void calculateStatisticsForCurrentWeek(String projectId, int week, boolean currentTime) { ... }

    private int countWorkingHoursForCurrentTime() { ... }

    private float calculateArrivalRate(List<Ticket> tickets, int workHours) { ... }

    private float calculateServiceRate(List<Ticket> tickets, LocalDateTime startOfWeek, int workHours) { ... }

    private StartEndDate getStartAndEndOfWeek(int year, int week) { ... }

}
```

Рисунок 3.1 – ProjectStatisticService клас

Метод calculateStatisticsForCurrentWeek(String projectId, boolean currentTime) веде обчислення статистики тижня для заданого проєкту. Він визначає часові межі поточного тижня і стягує інформацію про тікети за цей проміжок часу за допомогою ticketDao. В залежності від значення параметра currentTime, метод може обчислювати кількість пройдених робочих годин або

приймати за повний робочий тиждень. Після цього розраховуються значення часу прибуття та обслуговування, після чого отримані значення записуються в базу даних.

Метод calculateStatisticsForCurrentWeek(String projectId, int week, boolean currentTime) є перевантаженою версією методу вище, обчислює статистику для любого тижня року. Аналогічний попередньому методу, але визначення тижня здійснюється через параметр week.

Метод countWorkingHoursForCurrentTime() розраховує робочі годин до даного моменту з дня виклику. Вираховує робочий час, обідні перерви та робочі дні з початку тижня.

Метод calculateServiceRate(List<Ticket> tickets, LocalDateTime startOfWeek, int workHours) рахує швидкість обслуговування (μ) – середню кількість тікетів, опрацьованих за одиницю часу.

Метод calculateArrivalRate(List<Ticket> tickets, int workHours) вираховує прибуття тікетів (λ) – середня кількість тікетів за одиницю часу. Він ділить кількість тікетів (загальну) на робочі години (загальні).

А метод calculateServiceRate(List<Ticket> tickets, LocalDateTime startOfWeek, int workHours) обчислює швидкість обслуговування (μ), яка є середньою кількістю тікетів, оброблених за одиницю часу. Використовує дані закритих тікетів для розрахунку цього показника.

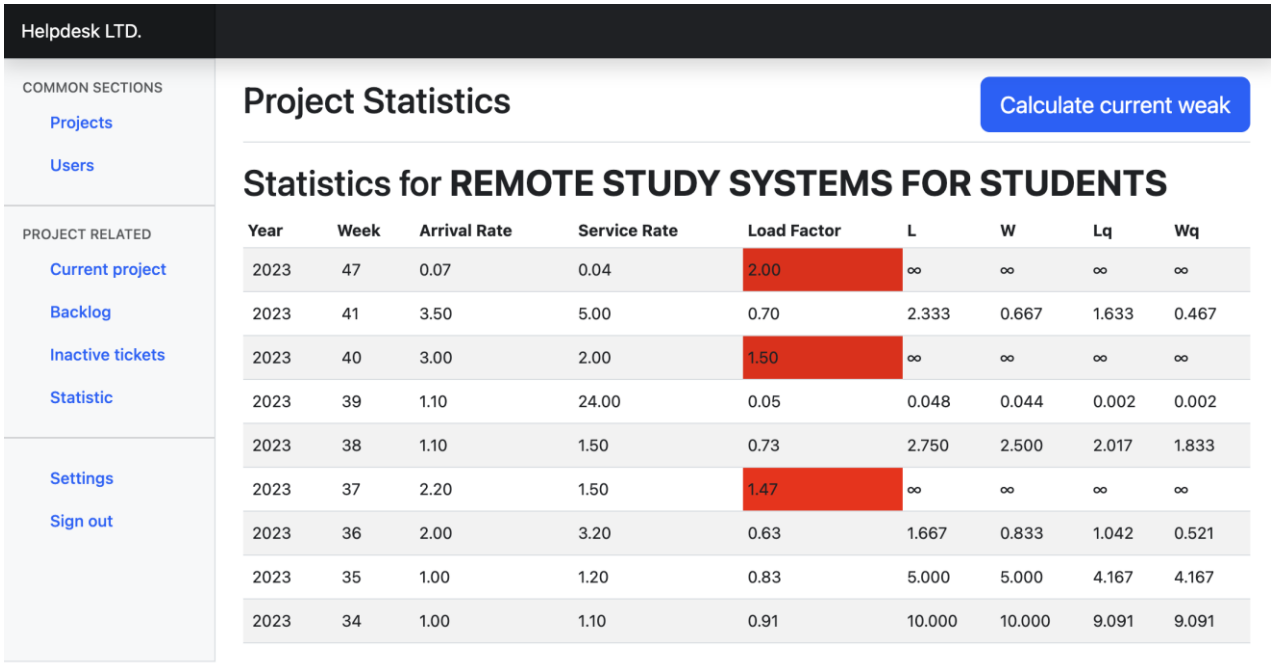
getStartAndEndOfWeek(int year, int week) – допоміжний метод для визначення тижня року. Встановлює часовий діапазон для збирання тікетів.

Кожен з цих методів є частиною загальної і відіграють ключову роль у зборі та аналізі даних проєкту, дозволяючи розраховувати важливі показники, які можуть бути використані для оцінки ефективності управління проектом та оптимізації процесів обслуговування.

На рис. 3.2 зображена веб-сторінка, що демонструє статистику для проекту під назвою "REMOTE STUDY SYSTEMS FOR STUDENTS". Зліва розташоване вертикальне меню, яке включає загальні секції, такі як "Projects" і

"Users", а також розділи, пов’язані з проектом: "Current project", "Backlog", "Inactive tickets", "Statistic" і "Settings". контент сторінки представлений у вигляді таблиці статистики, яка містить такі колонки: "Year", "Week", "Arrival Rate", "Service Rate", "Load Factor", "L", "W", "Lq" і "Wq". Дані в таблиці подаються по тижнях за 2024 рік.

- "Year" і "Week" позначають рік і номер тижня.
- "Arrival Rate (λ)" вказує на швидкість прибуття тікетів.
- "Service Rate (μ)" показує швидкість їх обслуговування.
- "Load Factor" є співвідношенням швидкості прибуття до швидкості обслуговування.
- "L" – це середня кількість тікетів у системі.
- "W" – середній час перебування тікета в системі.
- "Lq" – середня кількість тікетів у черзі.
- "Wq" – середній час очікування в черзі.



Year	Week	Arrival Rate	Service Rate	Load Factor	L	W	Lq	Wq
2023	47	0.07	0.04	2.00	∞	∞	∞	∞
2023	41	3.50	5.00	0.70	2.333	0.667	1.633	0.467
2023	40	3.00	2.00	1.50	∞	∞	∞	∞
2023	39	1.10	24.00	0.05	0.048	0.044	0.002	0.002
2023	38	1.10	1.50	0.73	2.750	2.500	2.017	1.833
2023	37	2.20	1.50	1.47	∞	∞	∞	∞
2023	36	2.00	3.20	0.63	1.667	0.833	1.042	0.521
2023	35	1.00	1.20	0.83	5.000	5.000	4.167	4.167
2023	34	1.00	1.10	0.91	10.000	10.000	9.091	9.091

Рисунок 3.2 – Сторінка зі статистикою для проекту

У таблиці деякі рядки виділені червоним кольором, це показує, що показники близькі до критичних. Надвисокі значення "Load Factor", виділені

червоним, вказують на те, що прибуття тікетів перевищує їх обслуговування, що може спричинити підвищення черги.

Кнопка "Calculate current week", яка знаходиться у правому верхньому куті сторінки, дозволяє виконати розрахунки статистики для тижня.

Дані таблиці охоплюють тижні з 34-го по 47-й. У деяких тижнях значення "Arrival Rate" і "Service Rate" значно коливаються, а "Load Factor" демонструє надмірне навантаження системи. Символ "∞" у колонках "L" і "Wq" указує на неможливість моделі розрахувати ці значення через перевантаженість системи у відповідні періоди.

РОЗДІЛ 4
ПРОЕКТУВАННЯ І РОЗРОБКА ЗАСТОСУНКУ

4.1 Порівняння багаторівневої архітектури з іншими типами

Багаторівнева архітектура (layered architecture) – це підхід до проєктування програмних систем, де система розділена на окремі рівні (шари), кожен з яких має чітко визначені обов’язки. Цей підхід є одним із найпоширеніших і широко застосовується у розробці програмного забезпечення, особливо у корпоративних і веб-додатках.

Основні характеристики багаторівневої архітектури.

1. Розділення відповідальності.
   - Рівень презентації (Presentation Layer): відповідає за інтерфейс користувача.
   - Рівень логіки (Business Logic Layer): реалізує бізнес-правила та логіку. Ще називають рівнем сервісів або додатків.
   - Рівень даних (Data Access Layer): відповідає за роботу з базами даних.
2. Інкапсуляція.
   - Кожен рівень приховує свою внутрішню реалізацію, відкриваючи інтерфейси для взаємодії.
3. Послідовна взаємодія.
   - Рівні взаємодіють між собою лише у визначеному порядку (зазвичай зверху вниз або через проміжний рівень).

На рис. 4.1 відображається взаємодія між рівнями застосунка, користувачем і рівнем презентації, рівнем зберігання і базою даних.

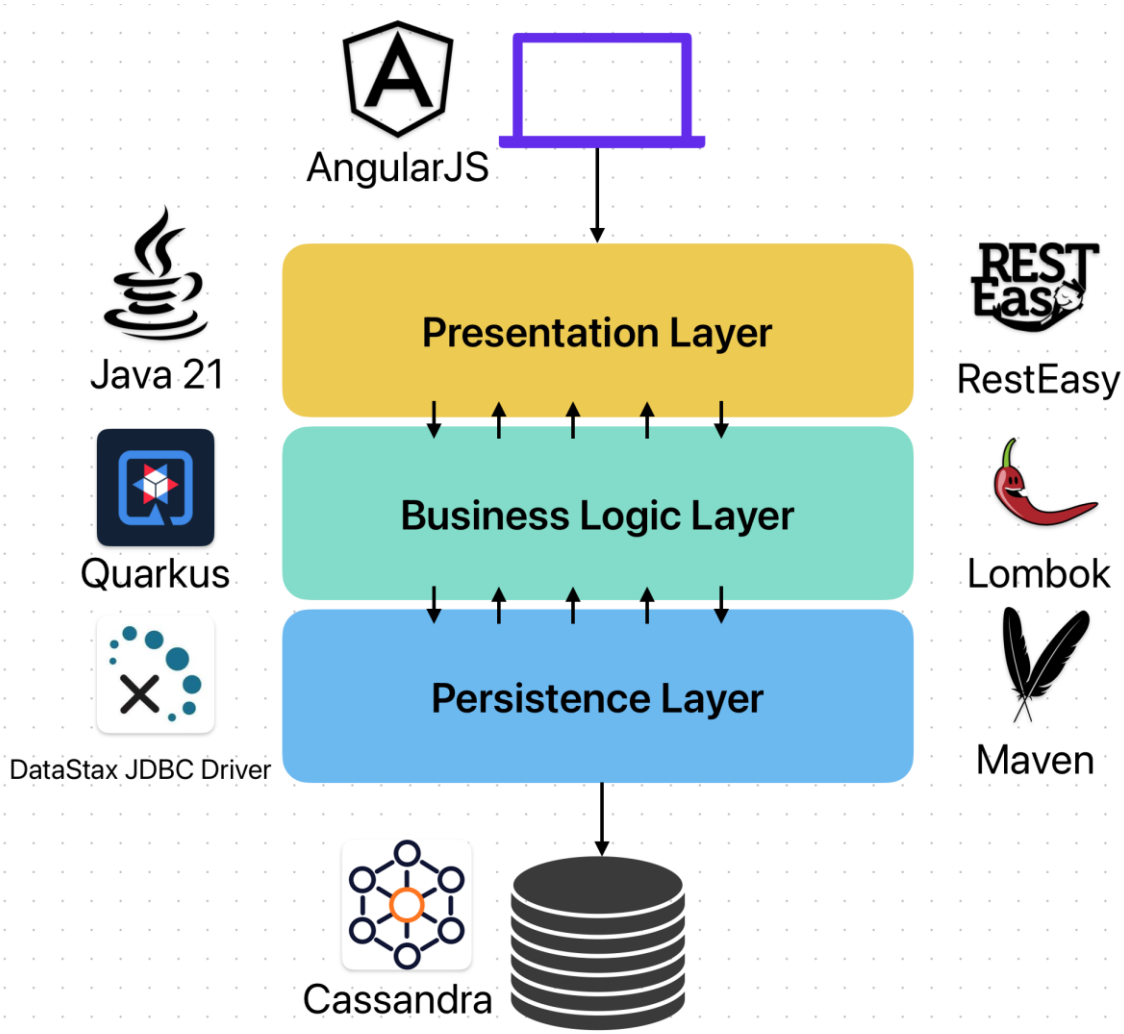


Рисунок 4.1 – Архітектура застосунка (трьохрівнева) та технології використані при розробці

Переваги багаторівневої архітектури.

- Модульність. Полегшує розробку, тестування та обслуговування, оскільки можна працювати над кожним рівнем окремо.
- Повторне використання. Рівні можуть бути використані в різних проєктах, якщо мають чітко визначені інтерфейси.
- Масштабованість. Легко масштабувати окремі рівні, наприклад, додати сервери для рівня даних.
- Простота заміни. Окремий рівень можна оновлювати або замінювати, не впливаючи на інші.

- Зрозуміла структура. Легко зрозуміти та документувати, особливо для великих команд.

Мікросервісна архітектура ділить додаток на маленькі, незалежні сервіси, які функціонують окремо і взаємодіють за допомогою HTTP/REST або асинхронної передачі повідомлень, причому кожен відповідає за конкретну бізнес-функцію. Це дозволяє досягти кращої масштабованості та відмовостійкості, прискорює випуск нових релізів. Проте така архітектура може ускладнити процес розгортання, комунікацію між сервісами та викликати можливі проблеми з узгодженістю даних.

Шестикутна архітектура (Hexagonal Architecture), або архітектура порти та адаптери, є стилем проєктування програмного забезпечення, що розділяє основну бізнес-логіку програми від зовнішніх компонентів, таких як бази даних, веб-сервери або інші зовнішні системи. Ідея полягає в тому, щоб організувати код таким чином, щоб внутрішня логіка могла бути незалежною від того, як вона взаємодіє з зовнішнім світом.

Багаторівнева архітектура створює надійну основу для розробки масштабованих і добре організованих додатків. Її переваги полягають у простоті та ефективному розподілі завдань. Але жодна архітектура не є універсальною. Вибір архітектурного стилю має залежати від конкретних вимог. Хоча багаторівнева архітектура є ідеальним варіантом для багатьох додатків, інші підходи, такі як мікросервіси, гексагональна архітектура або архітектура подій, інколи краще відповідають певним сценаріям, що показує важливість розуміння компромісів кожного з них [19].

4.2 Опис структури та складових частин застосунку.

У цьому розділі надається детальний огляд архітектури та основних компонентів застосунку, створеного для автоматизації обслуговування, технічної підтримки та сервісу клієнтів. Для створення додатку була використана комбінація технологій, таких як Java 11(21), Quarkus, AngularJS, REST, Docker, Cassandra та DataStax Java Driver. Для розробки застосунку використовувалося інтегроване середовище розробки (IDE) IntelliJ IDEA.

На рисунку 4.2 представлено UML-діаграму процесу відкриття сторінки projects.html. Користувач ініціює запит на відкриття веб-сторінки projects.html через інтерфейс користувача (UI) на базі AngularJS. Запускається порядок взаємодій, який веде до запиту списку проєктів у Java (Quarkus). Потім застосунок звертається до бази даних Cassandra для отримання цієї інформації. Отримавши відповідь від бази даних, застосунок передає список проєктів на початок через UI, де можемо їх переглядати.

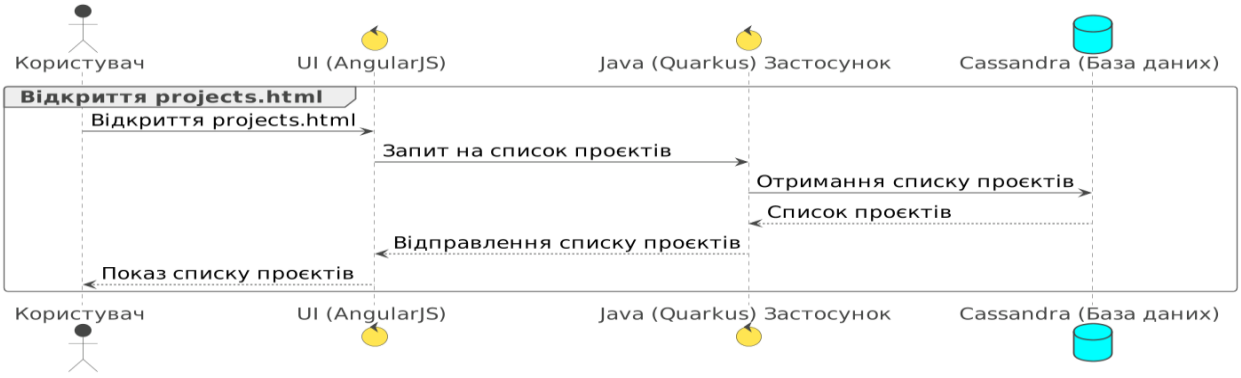


Рисунок 4.2 – UML-діаграма, що ілюструє процес відкриття користувачем сторінки projects.html

На рис. 4.3 наведено UML-діаграму, яка відображає взаємодію між користувачем, інтерфейсом користувача на AngularJS, застосунком на Java (Quarkus) та базою даних Cassandra, а саме відкриття користувачем сторінки users.html.

Користувач починає процес з відкриття сторінки users.html за допомогою інтерфейсу. Потім інтерфейс надсилає запит до Java-застосунку для повернення списку користувачів. Застосунок обробляє запит, звертається до бази даних, щоб отримати необхідний список.

База даних після обробки запиту надсилає список користувачів до застосунку. Отримавши цю інформацію, застосунок передає її інтерфейсу. У фіналі інтерфейс відображає отриманий список.

У цілому, ця діаграма демонструє типовий потік даних та взаємодію між різними компонентами системи під час відкриття та відображення списку користувачів на веб-сторінці.

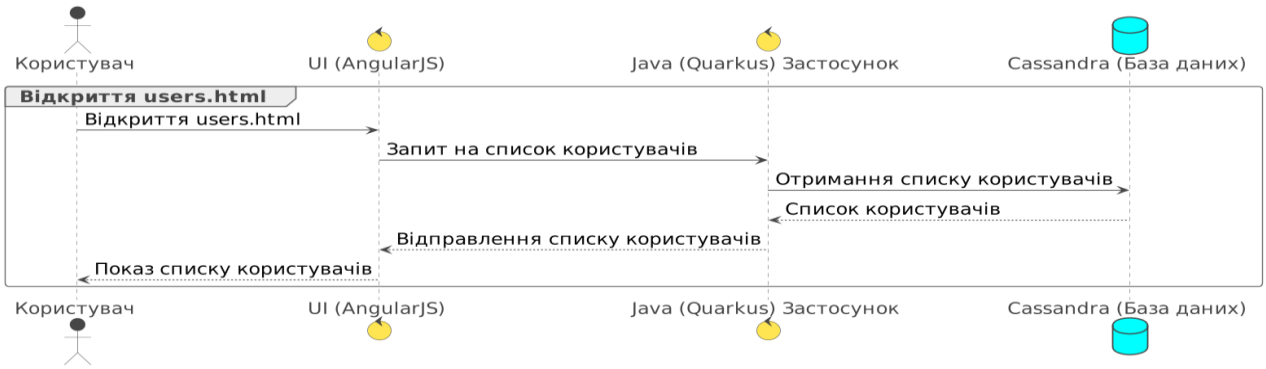


Рисунок 4.3 – UML-діаграма, що відображає процес відкриття сторінки users.html користувачем

На UML-діаграмі, представленій на рис. 4.4, показана взаємодія користувача та інтерфейсу, застосунку та базою даних. Діаграма зосереджена на процесі відкриття project.html і відображає три ключові етапи взаємодії.

Користувач відкриває project.html через UI, який надсилає запит до застосунку для повернення списку активних тікетів. Java-застосунок звертається до бази даних за необхідною інформацією, і віддає її UI.

Отримані статуси застосунок передає їх збази даних назад UI. Цю інформацію можна використовувати при створенні тікетів, оскільки необхідні

дані були завантажені заздалегідь, і користувач не змушений чекати в подальшому.

Java застосунок отримує цю інформацію з бази даних і передає список назад UI. Аналогічно попередньому випадку, дані завантажуються на UI завчасно.

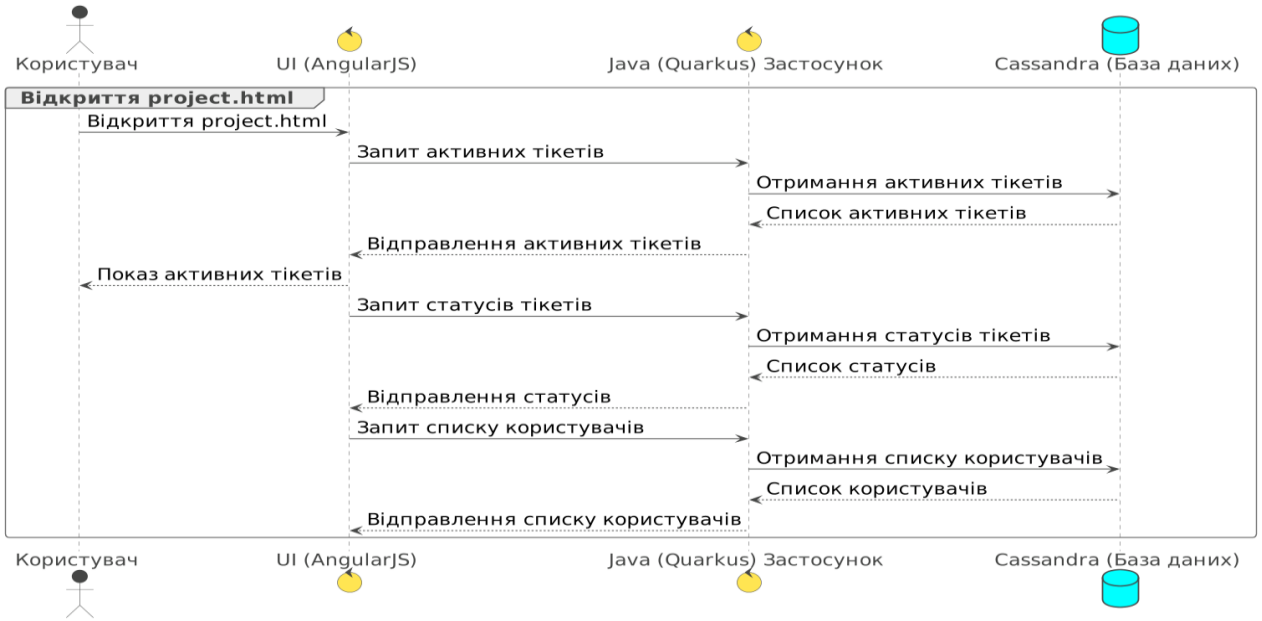


Рисунок 4.4 – UML діаграма, що показує процес, коли користувач відкриває файл project.html

UML діаграма, показана на рис. 4.5, ілюструє взаємодію користувача, UI, Java Quarkus і Cassandra при відкритті сторінки ticket.html, акцентуючи увагу на асинхронному обміні даними.

Відкриваючи сторінку ticket.html ініціюється взаємодія. UI надсилає запит до застосунку для отримання інформації тікету. Потім йде звернення застосунку до бази даних, одночасно асинхронно запитуючи дані про проєкт, пов’язаний з тікетом, а також інформацію про користувачів, які з ним пов’язані.

Застосунок, отримавши відповіді від бази даних передає інформацію про тікет UI, який її відображає. Також користувачу відображається інформація про користувача, що створив тікет, та того, хто займається його опрацюванням.

Потім надсилаються запити списків коментарів до тікету, інформації про того, хто створив тікет, і того, хто займається його опрацюванням. Всі ці запити опрацьовуються застосунком, який передає отриману інформацію з бази даних до UI.

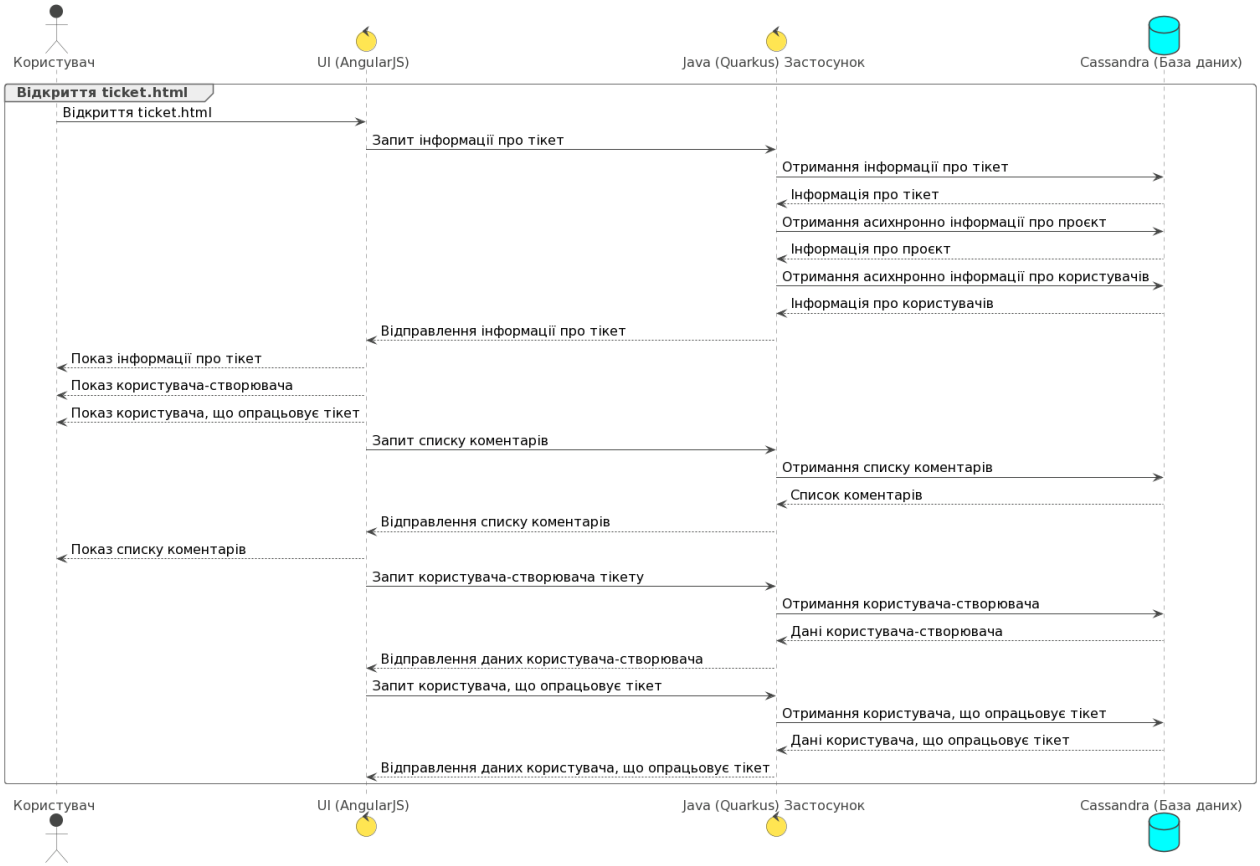


Рисунок 4.5 – UML діаграма, що зображає процес, коли користувач відкриває файл project.html

За допомогою Java 11 (21) та Quarkus була створена Внутрішня архітектура додатку. Quarkus дозволив швидко створити застосунок і легко інтегрувати необхідні модулі (extensions), що спростили впровадження інтеграції з базою даних і REST. Maven використовувався як інструмент для збірки і дав можливість налаштовувати процес збірки та автоматизувати рутинні завдання [20].

Бекенд включає Entity Layer, що є основою програми і забезпечує зв’язок між об’єктно-орієнтованою Java та реляційною базою даних. Цей рівень складається з класів, що відображають таблиці бази даних і містять анотації. Сутності зазвичай знаходаться на рівнях DAO або сервісів для виконання операцій CRUD.

Domain Layer є основою бізнес-логіки, приховуючи робочі процеси. Цей рівень складається з об’єктів домену, агрегатів, фабрик та об’єктів значень, які взаємодіють із сервісним і ресурсним рівнями для виконання операцій, що стосуються домену.

Data Access Layer (DAL) – це шар у багатошаровій архітектурі програмного забезпечення, відповідальний за взаємодію з базою даних або іншими джерелами зберігання даних. Його основна мета – інкапсулювати доступ до даних, забезпечуючи чітке розділення між бізнес-логікою програми і специфічними деталями роботи з базою даних. Структура DAL зазвичай включає.

- DAO (Data Access Objects). Об’єкти, що безпосередньо виконують операції CRUD (Create, Read, Update, Delete).
- Репозиторії. Більш високорівневі об’єкти, які інкапсулюють складні запити та забезпечують бізнес-логіку роботи з даними.
- ORM-фреймворки. Наприклад, Hibernate або JPA у Java для спрощення взаємодії з реляційними базами даних.

Service Layer (Сервісний шар) – це рівень у багатошаровій архітектурі програмного забезпечення, який відповідає за реалізацію бізнес-логіки та управління виконанням основних операцій програми. Він слугує посередником між користувацьким інтерфейсом (або API) та іншими внутрішніми компонентами, такими як Data Access Layer (DAL) або Domain Layer.

Transformer Layer – це шар у програмній архітектурі або обчислювальному процесі, який відповідає за трансформацію даних між різними форматами, моделями або рівнями абстракції. Його основне завдання –

забезпечити сумісність даних між різними компонентами системи або етапами обробки.

Configuration Layer – це шар в архітектурі програмного забезпечення, який відповідає за управління налаштуваннями та параметрами програми. Його завдання – забезпечити зручне й централізоване зберігання, доступ і модифікацію конфігураційних даних, необхідних для роботи додатку.

Resource Layer – це рівень у багатошаровій архітектурі програмного забезпечення, який забезпечує зовнішній інтерфейс для доступу до функціоналу програми. Він використовується для прийому запитів від клієнтів (користувачів або інших систем) і передачі відповідей. У веб-додатках Resource Layer зазвичай відповідає за роботу REST API, GraphQL, чи інших протоколів інтеграції.

Архітектура інтерфейсу застосунку побудована на базі AngularJS – динамічного фреймворку, орієнтованого на створення зручного та інтерактивного користувацького досвіду. Завдяки AngularJS вдалося розробити застосунок із багатим функціоналом і високим рівнем взаємодії з користувачем.

Для ефективної інтеграції фронтенду з бекендом використовувалися RESTful API. Визначені REST-ендпойнти забезпечували безперебійну передачу даних між компонентами, дозволяючи користувачам легко взаємодіяти з функціональністю застосунку [21].

Для управління базами даних вибір пав на Cassandra, оскільки її ефективність та розподілена архітектура дозволяє обробляти великі обсяги даних ідеально відповідали вимогам проєкту. У моделі даних застосунку використовувалися таблиці, такі як "project", "comment", "ticket", "ticket_status" і "user".

У проєкті було використано бібліотеку Lombok, яка спростила створення, зменшивши обсяг стандартного коду. Для взаємодії з базою даних Cassandra застосовувався DataStax Java Driver, який забезпечив ефективність операцій пошуку та збереження даних. Він перетворював результати запитів у об’єкти сутностей (entity) та спрощував роботу з CQL-запитами, генеруючи їх на основі

назв методів, анотацій і змінних, що використовувалися у методах, без необхідності вручну писати код запитів.

IntelliJ IDEA використовувалась як інтегроване середовище розробки (IDE) для розробки, тестування та модернізації коду застосунку. Це середовище забезпечило зрозумілий інтерфейс, що значно спростило процес створення. Крім того, доступні в IDE плагіни надали можливості для легкого доступу до бази даних, валідації та форматування коду, а також забезпечили просту й інтуїтивно зрозумілу навігацію по кодовій базі.

Модульний підхід, чітко організовані пакети та застосування інструментів забезпечили розробку додатку, який має потенціал змінити процеси сервісу та обслуговування клієнтів.

4.3 Перехід між java 11 та java 21

Java 11 була високо оцінена, оскільки стала першим випуском з підтримкою (LTS) після зміни політики релізів компанією Oracle. Вона принесла безліч нових функцій, покращень у JVM та вдосконалень API. Вибір Java 11 як бази для веб-застосунку був логічним, вона дає стабільну платформу з тривалою підтримкою.

Java як мова та платформа не стояла на місці. З впровадженням нового циклу випусків кожні півроку, з’являлися нові версії, всі містили оптимізації, нові функції. Така постійна еволюція зробила Java більш динамічною, гнучкоюі та підвищила як продуктивність розробників, так і ефективність додатків [22].

Java 21, як нова версія з довготривалою підтримкою (LTS), приносить ряд значних переваг, які покращують продуктивність, безпеку та зручність для розробників. Ключові переваги Java 21.

1. Покращення продуктивності.
    - Оптимізація виконання коду. Покращення в компіляції та виконанні коду, що забезпечує швидший старт та виконання програм.

- Покращення в роботі з пам’яттю. Оптимізації для зменшення використання пам’яті та більш ефективного управління ресурсами, що підвищує швидкість роботи додатків.

2. Нова система збору сміття.

- Java 21 включає нові алгоритми збору сміття, які знижують час затримки, покращують роботу з великими обсягами даних і допомагають знизити навантаження на систему.

3. Модульність.

- Покращення в системі модулів, що дозволяє ще більше зменшити розмір додатків і покращити управління залежностями. Це робить код більш чистим, організованим та зручним для масштабування.

4. Поліпшення підтримки безпеки.

- Введення нових функцій для посилення безпеки, зокрема підтримка нових алгоритмів шифрування і більш захищеного виконання коду.
- Покращена підтримка TLS та нові функції для захисту даних і зменшення вразливостей.

5. Нове API та можливості для розробників.

- В Java 21 з’явилися нові API для зручнішої роботи з потоками даних, покращеннями в синхронізації та інтеграцією з іншими технологіями.
- Підвищена підтримка лямбда-виразів і нові можливості для кращої роботи з анотаціями та обробкою метаданих.

6. Покращення підтримки зворотної сумісності.

- Java 21 підтримує високу зворотну сумісність, що дозволяє без проблем мігрувати старі проекти на нову версію з мінімальними змінами.

7. Покращена документація та інструменти для розробників.

- Нові інструменти для моніторингу і діагностики додатків, що полегшують процес налагодження і оптимізації.
- Поліпшення в документації та зручності її використання, що допомагає розробникам швидше розуміти і використовувати нові функції.

8. Зміни в мовному рівні.
   - Впровадження нових функцій для pattern matching, поліпшення конструкцій switch та інших можливостей мови, які зменшують складність коду і роблять його більш виразним.
9. Екосистема та підтримка.
   - Оскільки Java 21 є LTS-версією, вона отримує тривалу підтримку від Oracle, що гарантує стабільність і безпеку на довгий період часу.

Ці переваги роблять Java 21 потужним інструментом для розробників, дозволяючи створювати більш ефективні, масштабовані і безпечні додатки з використанням новітніх технологій.

Розробка на Java 21 – це було стратегічне рішення, орієнтоване на майбутнє, яке охоплює сучасні парадигми розробки, забезпечує високу продуктивність і гарантує безпечне та ефективне середовище [23].

4.4 Основні особливості та функціональність застосунку

Досліджуючи можливості та функціонал застосунку, розділ зосереджений на різних сторінках, які сприяють покращенню автоматизації [24].

Сторінка "projects.html" є важливим елементом програми, функціонуючи як центр для керування проєктами. На ній користувачі мають змогу не тільки переглядати перелік проєктів, а й робити різноманітні дії.

На рис. 4.6 представлений спливаючий діалог де можна створити новий проєкт, де необхідно заповнити основні дані опису.

Можна оновлювати проєкти, оскільки існуючі дані можна просто коригувати, щоб адаптувати їх до змін або нових вимог. Коли проєкти стають

застарівшими, їх можна швидко видаляти із системи, як на рисунку 4.7. Веб-сторінка "users.html" робить простішим управління користувачами, надаючи повний огляд профілів і дає змогу робити різні дії. Нових учасників команди можна додати до системи. Приклад показано на рис. 4.8.

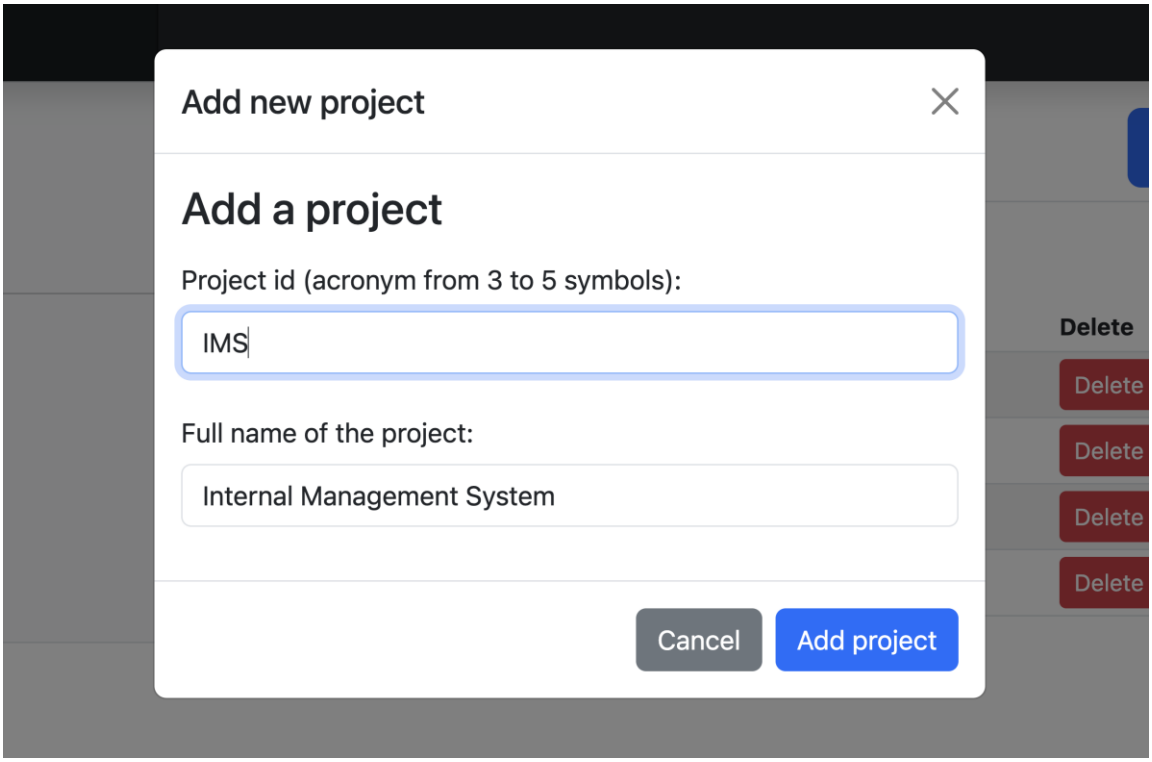


Рисунок 4.6 – Спливаюче вікно для додавання проєкту.

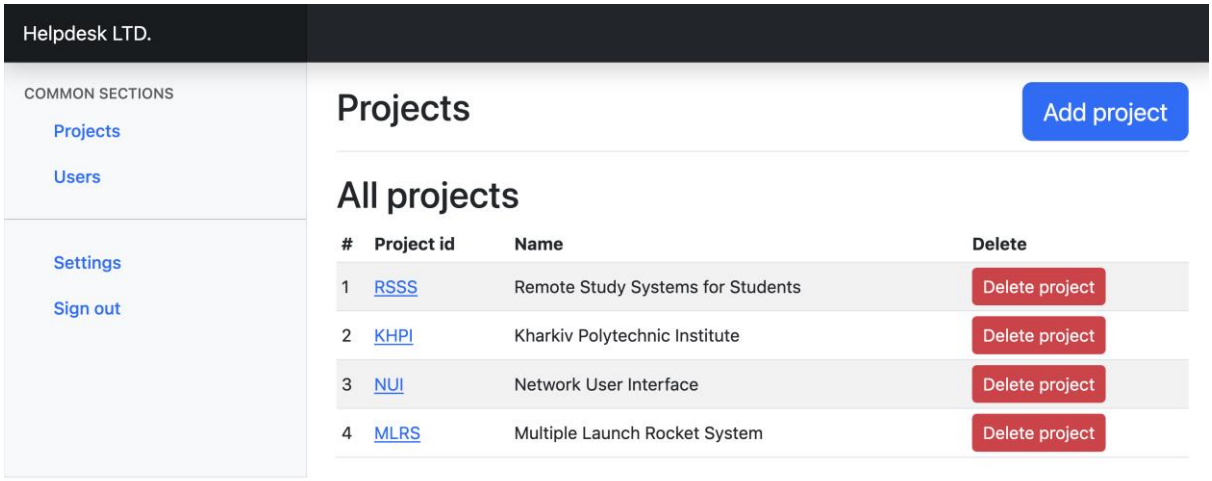


Рисунок 4.7 – Сторінка керування проєктами

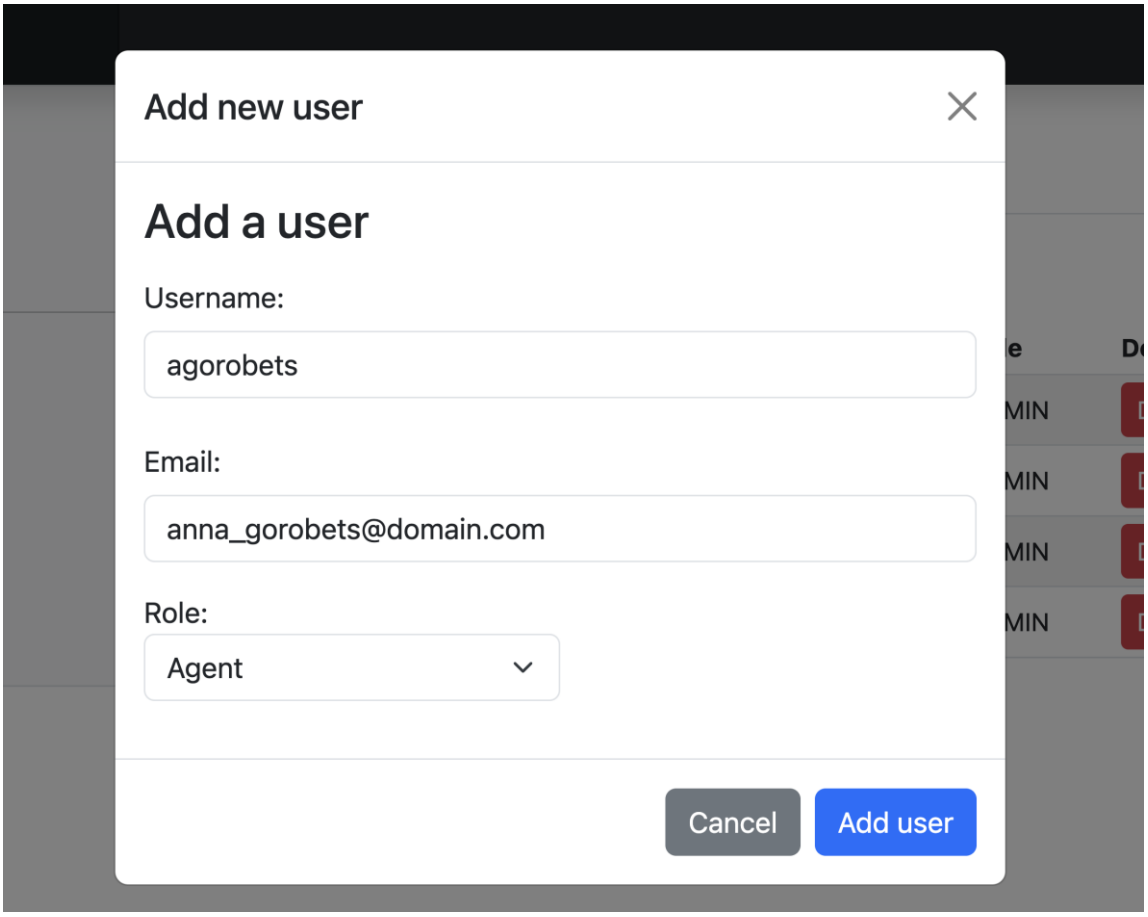


Рисунок 4.8 – Спливаюче вікно для додавання нового користувача в систему

Розглянемо ситуацію, коли адміністратор повинен коректувати дані або додати нових учасників команди підтримки. На рисунку 4.9 показана сторінка "users.html", з тестовими даними. Вона є важливою складовою для підтримки актуальної та точної бази даних.

На сторінці "project.html" зберігаються активні завдання та проєктна інформація, котра дає повний огляд окремих проєктів, з детальним описом і списком активних запитів.

Користувачі можуть отримувати інформацію про проєкт, дедлайни цілі.

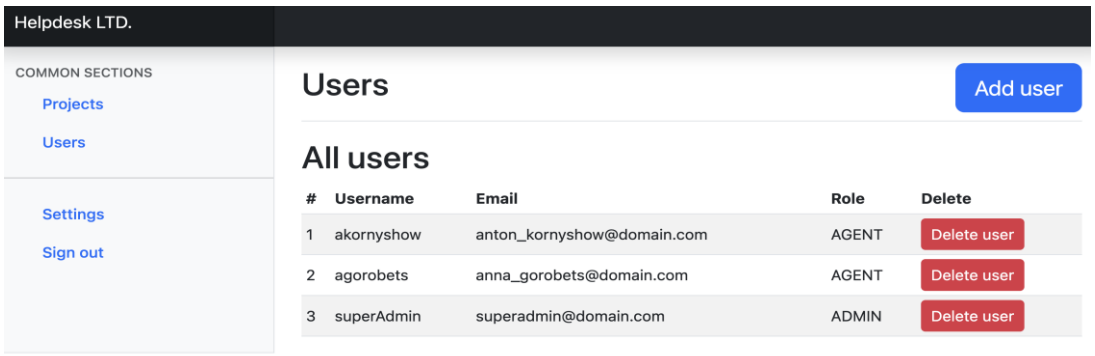


Рисунок 4.9 – Сторінка керування користувачами

На рис. 4.10 показано інтерфейс, який дозволяє легко додавати нові тікети, зазначаючи всі деталі для вирішення запиту.

Інформація про користувача, який надіслав тікет, може бути коригована. Це зручно, коли користувач має проблеми з доступом, і інший користувач може заповнити ці дані. Згодом це поле, як і поле з інформацією про відповідального за тікет, може бути оновлене. Для того щоб менеджер або команда мали змогу правильно визначати пріоритетність завдань, поле "Priority" має бути заповнене коректно. У разі потреби його також можна змінити пізніше.

Користувачі також мають змогу бачити активні завдання. Як показано на рис. 4.11, відображається компактний список тікетів, які перебувають у роботі.

Наприклад, менеджер відкриває сторінку "project.html", для оцінки ходу виконання проєкту, проаналізувати головні кроки та контролювати розподіл активних тікетів для їх своєчасного вирішення.

Сторінка " backlog.html" працює для проєктів в яких завдання ще не призначені, але вже активні. Головна її мета - це ефективне управління та розподілі завдань.

На сторінці представлений список невизначених тікетів. Підтримка має можливість розглядати ці завдання та призначати їх відповідальним, враховуючи пріоритетність.

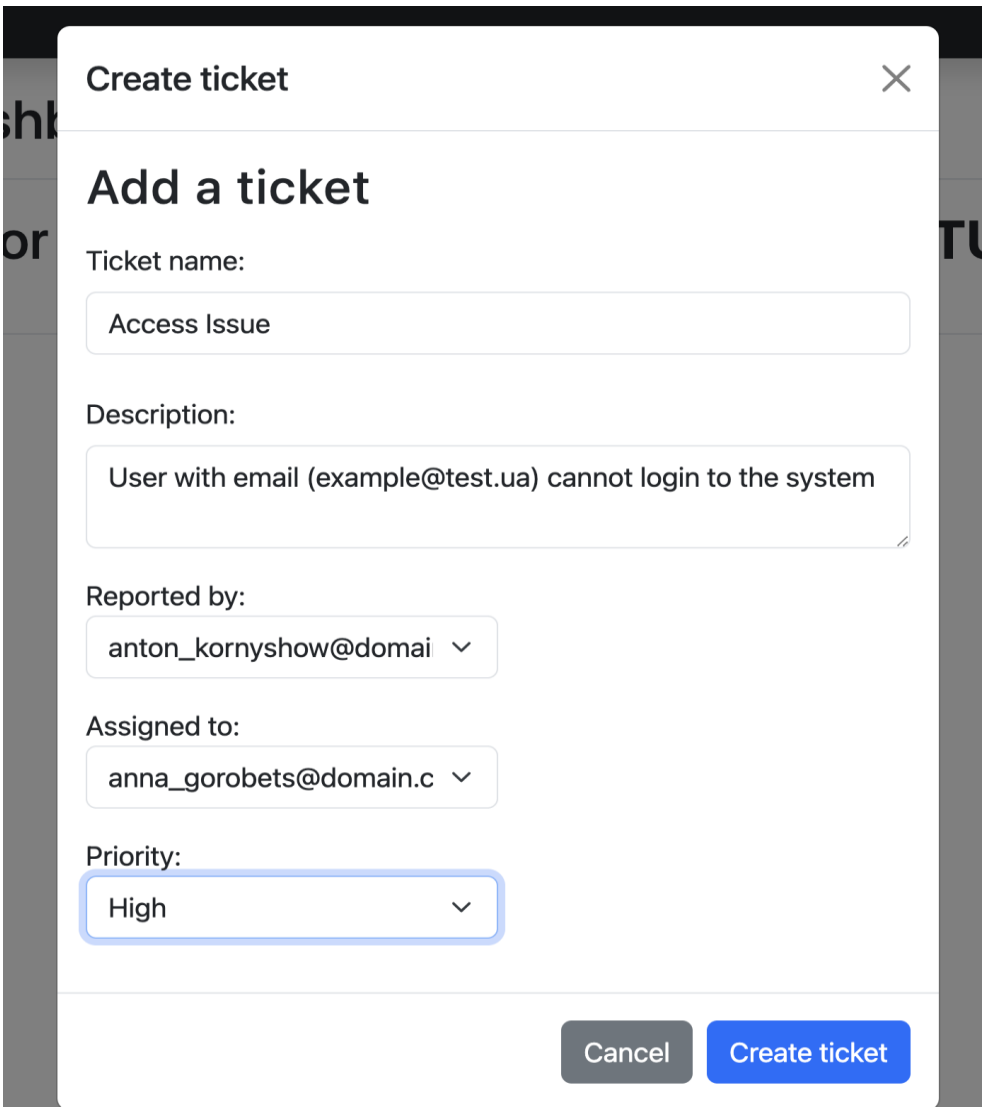


Рисунок 4.10 – Вікно для створення нового тікета

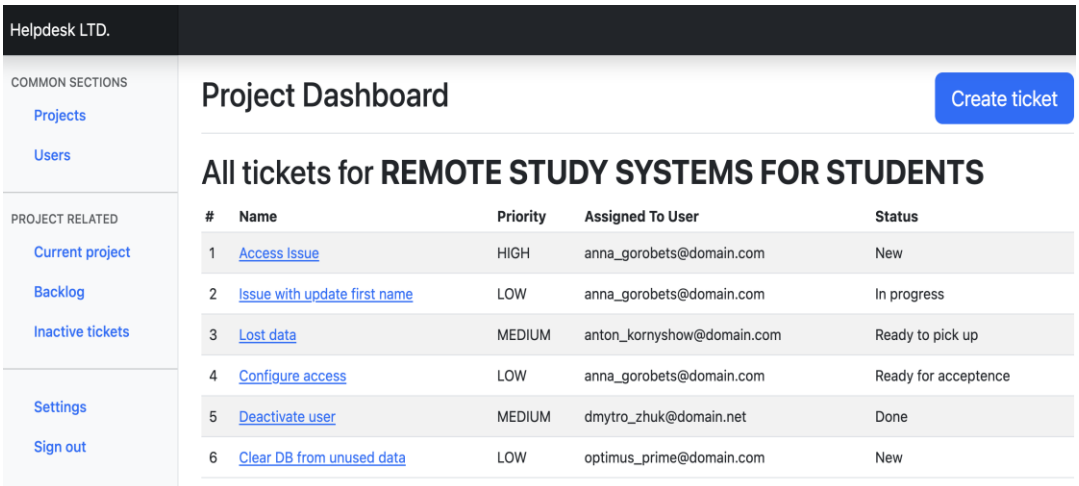


Рисунок 4.11 – Сторінка для перегляду активних тікетів, які перебувають у роботі або плануються до виконання найближчим часом

Оптимізація обсягу роботи є однією із цілей сторінки ‘backlog.html’, показаної на рисунку 4.12. У ситуаціях, коли службі підтримки треба регулярно опрацьовувати вільні тікети, ‘backlog.html’ робить процес простіше.

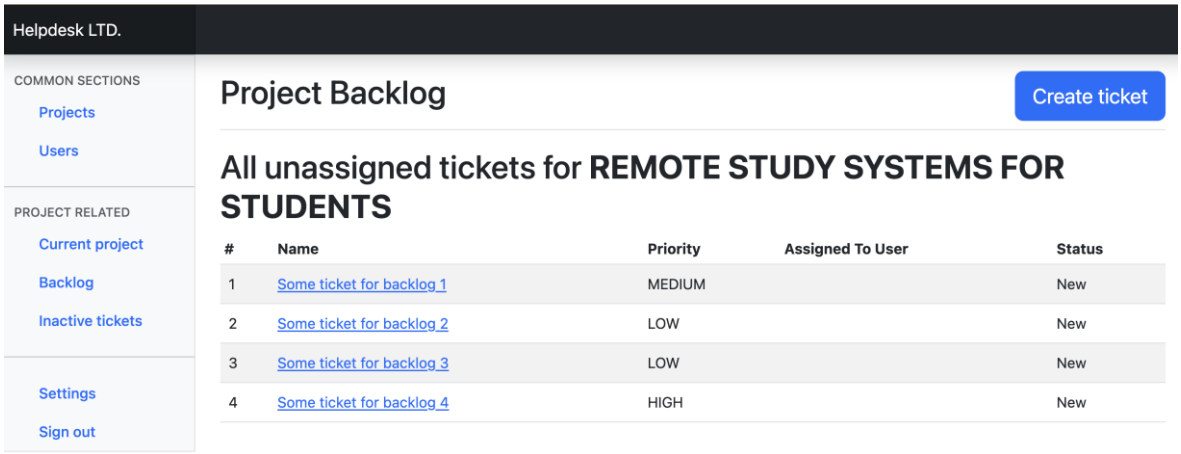


Рисунок 4.12 – Сторінка перегляду нерозподілених завдань

Сторінка inactiveTickets.html, на рисунку 4.13, потрібна для відображення ретроспективних даних. Організований список котрий містинь неактивні завдання, що містить ця сторінка, дає змогу побачити завершену роботу. Це зроблено задля зручності аналізу, виявлення тенденцій і рутинних проблем.

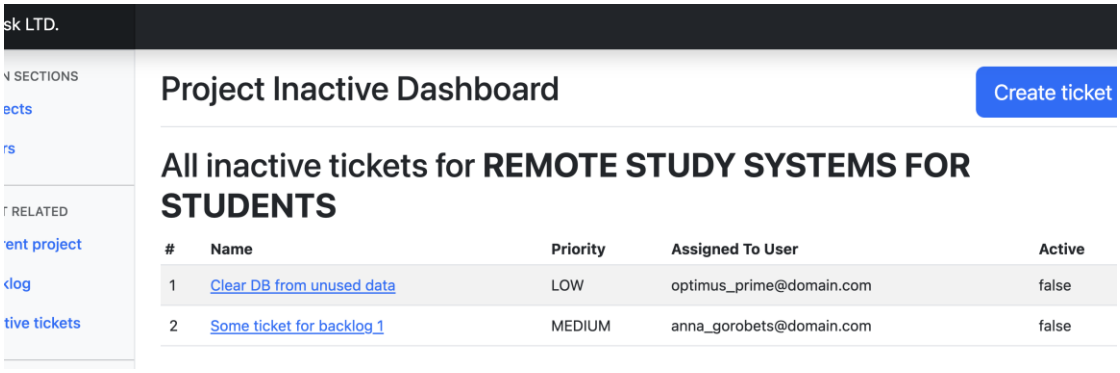


Рисунок 4.13 – Сторінка виконаних завдань завдань

Сторінка "ticket.html" виступає як зручний інтерфейс керування тікетами, котрий показано на рис. 4.14.

Тут знаходиться інформація про тікет. Така як статус, коментарі. Також

можна відслідковувати прогрес виконання.

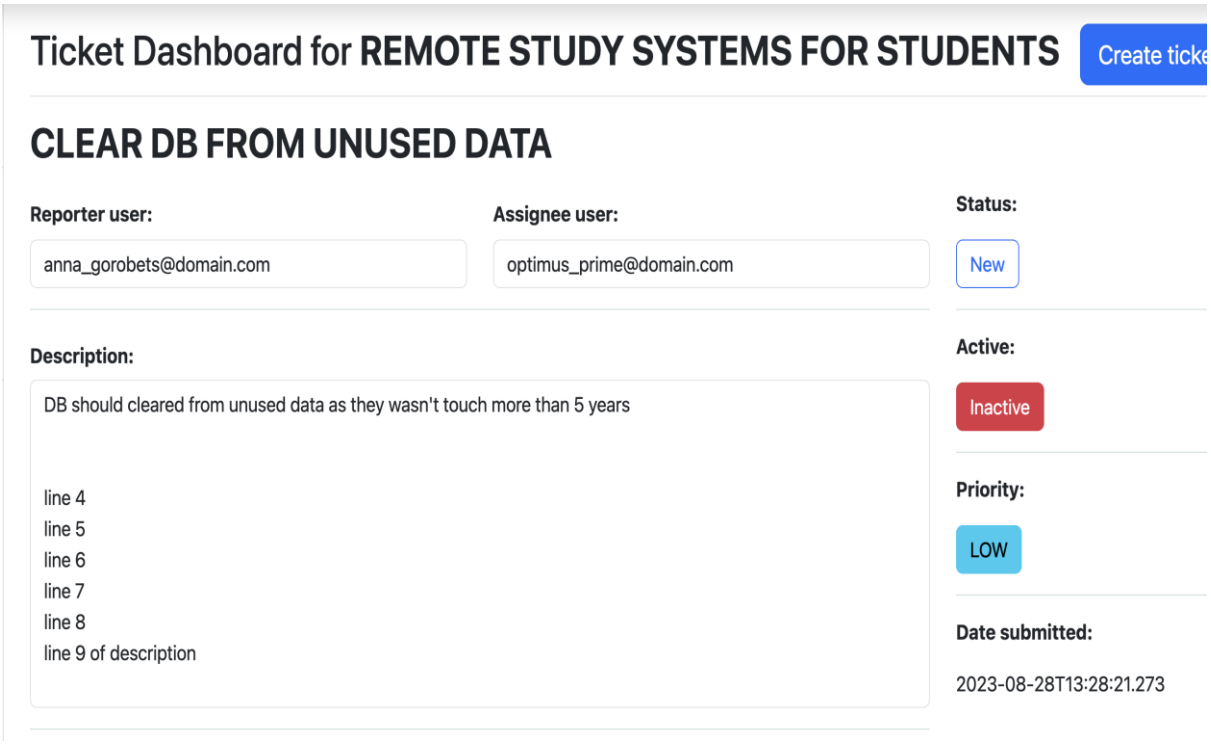


Рисунок 4.14 – Детальний огляд тікета

Оновлення інформації в тікеті здійснюється інтуїтивно просто. Для зміни репортера треба нажати на поле email, після чого побачите список усіх користувачів. Як відображено на рис. 4.15. Або почати вводити нік, після чого пройде фільтрація списку. Потім відправиться REST-запит до сервера, і оновлення відбувається автоматично.

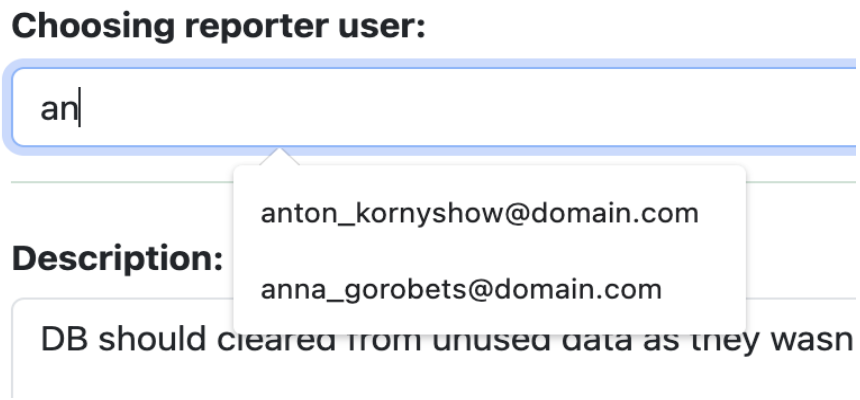


Рисунок 4.15 – Приклад фільтрації користувачів та оновлення поля репортера

Оновлення статусу зображено на рис. 4.16.

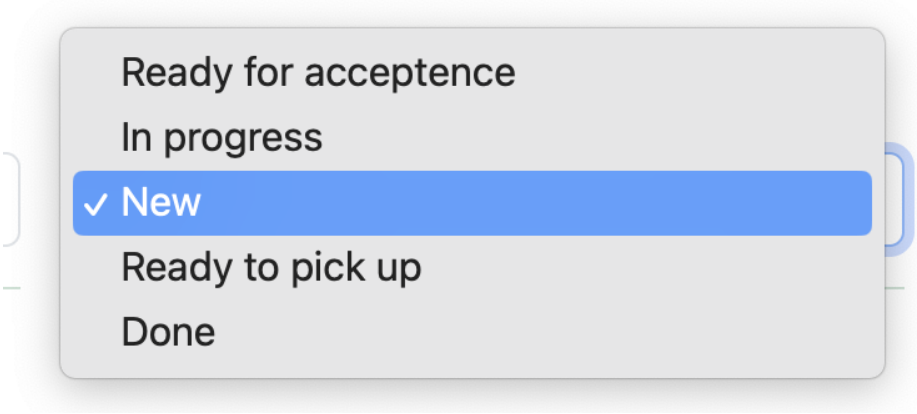


Рисунок 4.16 – Процес оновлення статусу

На рис. 4.17 показано процес оновлення тікета. Треба навести курсор на кнопку, після чого її текст зміниться на запит щодо дії. Після кліку стан тікета буде оновлено. Аналогічно можна активувати тікет.

Рисунок 4.17 – Оновлення стану з Active в Inactive

Рис. 4.18 показує процес оновлення пріоритету, процес аналогічний зміні статусу.

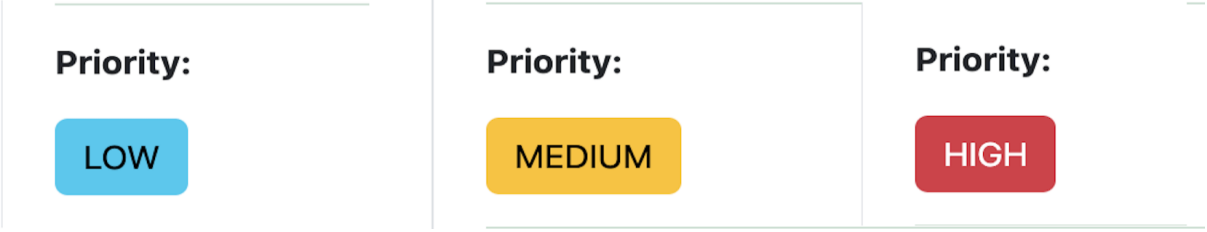


Рисунок 4.18 – Значення пріоритетів

Веб-сторінка "ticket.html" включає ключову функцію під назвою

"Коментарі", яка сприяє ефективній взаємодії між командою та клієнтами. Розділ зосереджується на тому, як інтерфейс (UI) для коментарів розроблено, щоб забезпечити зручну взаємодію.

Дизайн та макет. На сторінці "ticket.html" розділ коментарів зазвичай розташований у нижній частині, під деталями тікета. Його мінімалістичний дизайн гармонійно вписується в загальну естетику веб-додатку, не відволікаючи уваги від основного змісту тікета.

Новий коментар. Для додавання коментаря користувачі можуть скористатися полем введення та кнопкою "Add Comment", розташованими у верхній частині розділу. Це розташування дає зручний доступ і стимулює активність. Після введення тексту коментаря натискання кнопки "Add Comment" публікує його. Нові коментарі з’являються на початку списку, дотримуючись порядку "останнє-перше", що дозволяє швидко переглядати свіжі взаємодії без необхідності прокрутки.

Коментарі представлені у виді карток, кожна з яких містить кілька важливих елементів. Верхня частині картки має ідентифікатор проєкту та тікета, для зрозуміння контексту. Вказана дата подання, яка дозволяє оцінити актуальність обговорення. Ім’я користувача, що написав коментар, також відображається для його ідентифікації. Текст самого коментаря для зручного читання розташований на видному місці.

Веб-сторінки застосунку, такі як "projects.html", "users.html", "project.html", "backlog.html", "inactiveTickets.html" і "ticket.html", спільно сприяють автоматизації процесів сервісу. Дизайн та надійна функціональність, дозволяють командам ефективно управляти проєктами [25].

4.5 Особливості коду застосунку

Цей розділ описує окремі класи з кожного пакету: їхній вигляд, функціональність, опис та інші можливості [26].

```
public record ProjectDto (
    @Size(message = "Project id should be greater 2 symbols and less 6", min = 3, max = 5)
    String id,

    @Size(message = "Name should have at lease 5 symbols", min = 5)
    @NotBlank(message="Name may not be blank")
    String name
) {}
```

Рисунок 4.19 – ProjectDto class

Рисунок 4.19 Демонструє об’єкт передачі даних (DTO) для домену Project, який застосовується для обміну даними між різними рівнями програми.

Використані такі анотації як @Size і @NotBlank.

@Size застосовується для перевірки, що розмір анотованого елемента (у цьому випадку довжина рядка) знаходиться в межах заданих мінімального та максимального значень [27].

Анотація @Size у класі ProjectDto контролює довжину полів "id" та "name". Якщо довжина цих полів виходить за межі вказаного діапазону, тоді з’являється помилка валідації. Вона має атрибути для встановлення мінімальної та максимальної довжини, а також дозволяє налаштувати власне повідомлення про помилку в разі порушення правил.

@NotBlank забезпечує, що елемент не є null і має ненульову довжину після обрізання пробілів. Іншими словами, вона робить перевірку на порожність рядку або чи не складається від тільки з пробілів.

Під час перевірки об’єкта ProjectDto анотація @NotBlank застосовується до поля "name". Вона викликає помилку валідації, якщо поле є null, порожнім або містить лише пробіли. За допомогою атрибуту ‘message’ можна задати кастомне повідомлення про помилку, яке буде повернуто у разі невдалої перевірки.

У фреймворку Quarkus це реалізується завдяки інтеграції з Hibernate Validator, який є реалізацією специфікації Bean Validation. Quarkus забезпечує просту інтеграцію з Hibernate Validator, що дозволяє використовувати анотації

@Size і @NotBlank.

Під час обробки запиту в програмі Quarkus, коли REST-кінцева точка отримала запит, а тіло запиту відображається у DTO (як у ProjectDto), фреймворк автоматично виконує перевірку DTO перед його подальшою обробкою. У разі порушення будь-яких обмежень валідації Quarkus формує відповідь HTTP (зазвичай 400 Bad Request) з детальною інформацією про помилки валідації та надсилає її клієнту.

У разі невдалої валідації до відповіді додаються кастомні повідомлення про помилки.

```
@Dao
public interface ProjectDao {

    @Update
    void update(Project project);
    @Select
    PagingIterable<Project> findAll();
    @Select
    Project findById(String projectId);
    @Select
    CompletionStage<Project> findByIdAsync(String projectId);
    @Insert
    void save(Project project);
    @Delete(ifExists = true)
    boolean delete(Project project);

}
```

Рисунок 4.20 – class ProjectDao

На рис. 4.20 зображено інтерфейс який визначає методи доступу до даних Project в базі даних. Він використовує анотації з драйвера Cassandra для визначення таких операцій, як вибірка, вставка і видалення [28].

Рис. 4.20 показує інтерфейс, в якому методи для роботи з даними Project.

Анотації використовуються з драйвера Cassandra.

@Dao помічає інтерфейс як об’єкт доступу до даних (DAO) для роботи з Cassandra, вказуючи, що він містить методи взаємодії з базою даних. Завдяки інтеграції Quarkus з драйвером DataStax Cassandra, ця анотація розпізнається, і під час збірки застосунка Quarkus автоматично генерує реалізацію інтерфейсу DAO.

@Update позначає метод, який виконує операцію оновлення. Викликаючи його використовується метод який реалізований DataStax

Анотація @Select дає команду методу виконати вибірку.

Анотація @Insert вказує на те, що метод повинен вставити надану сутність в базу даних. Процес аналогічний виконанню анотації @Update.

Анотація @Delete позначає метод, який видаляє вказану сутність з бази даних. Якщо атрибут "ifExists" встановлено в true, видалення виконується лише за умови існування сутності в базі. У разі відсутності даних у базі буде згенеровано помилку виконання.

```
@Entity
public record Comment(

        @PartitionKey
        String projectId, // The ID of the project to which the ticket belongs.
        @ClusteringColumn(1)
        int ticketId,
        @ClusteringColumn(2)
        LocalDateTime dateSubmitted,

        String content,
        UUID userId
) {}
```

Рисунок 4.21 – class Comment

Рисунок 4.21 представляє сутність коментаря.

@Entity позначає клас для Cassandra як сутність, вказуючи, що його

екземпляри можуть бути відображені на рядки в таблиці. Quarkus і DataStax Cassandra визначає цю анотацію та оперує нею, щоб згенерувати код, необхідного для співставлення екземплярів класу Comment з відповідними рядками в таблиці Cassandra.

@PartitionKey помічає поле ключем розділу для таблиці Cassandra, що визначає, як дані розподіляються між вузлами. Коли поле "projectId" позначене @PartitionKey, тоді інформація коментаря буде групуватися за значенням "projectId". Таким чином, коментарі, що пов’язані з конкретним проєктом, зберігатимутья на одному вузлі, що дозволяє оптимізувати процес пошуку даних.

Анотація @ClusteringColumn потрібна щоб позначити стовпці кластеризації в Cassandra, які визначають порядок зберігання рядків (з однаковим ключем розділу) у межах розділу. Атрибути (позиція кластеризаційного стовпця), такі як 1 або 2, визначають черговість упорядкування. Вищий пріоритет мають мають у впорядкуванні. В межах одного розділу, визначеного "projectId", коментарі спочатку впорядковуються за "ticketId", а потім за "dateSubmitted".

На рис. 4.22 відображено клас що відчиняє RESTful фінальні точки для CRUD над Comment.

```
@Path("/comments")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class CommentResource {

    @Inject
    CommentService commentService;

    @GET
    @Path("/projectId/{projectId}/ticketId/{ticketId}")
    public List<CommentDto> getAllByIds(String projectId, int ticketId) {
        return commentService.findAllByIds(projectId, ticketId);
    }

    @POST
    public void add(@Valid CommentDto comment) {
        commentService.save(comment);
    }

    @DELETE
    public void delete(CommentDto comment) {
        commentService.delete(comment);
    }
```

Рисунок 4.22 – class CommentResource

Анотації використані в класі: @Path, @Inject, @GET і @POST [29].

@Path задає основний шлях URI для класу ресурсу. Методи цього класу використовуватимуть вказаний основний шлях. Quarkus, спільно з JAX-RS (Jakarta API для RESTful), застосовує анотацію @Path для переадресації вхідних HTTP-запитів до відповідних методів ресурсу базуючись на URI [30].

@Produces визначає типи медіа, котрі створюються методами ресурсу. В цьому кейсі вказано, що методи будуть створювати JSON. Коли повертається відповідь, Quarkus забезпечує серіалізацію цієї відповіді у зазначений тип даних (в даному випадку JSON) перед тим, як відправити клієнту.

@Consumes встановлює типи даних, котрі методи можуть отримувати від клієнта. У цьому випадку вказується, що методи будуть приймати JSON. Quarkus десеріалізує це тіло надіслане клієнтом з вказаного типу даних (у даному випадку JSON) в об’єкт Java.

@Inject застосовується для впровадженні залежностей. Анотація говорить Quarkus установити екземпляр bean-а (в даному випадку CommentService) в анотоване поле. Для ін’єкції Quarkus користується специфікацію CDI (Contexts and Dependency Injection). Quarkus забезпечує надання та встановлення екземпляра CommentService у відповідне поле після створення його екземпляру.

Анотації @GET, @POST, @DELETE вказують HTTP-метод, і Quarkus на цей метод має відповісти. Він використовує анотації щоб перенаправити вхідні HTTP-запити методу ресурсу в залежності від HTTP-методу запиту [31].

Інтеграція з JAX-RS у Quarkus забезпечує підтримку створення RESTful веб-сервісів на Java, в якому анотації класу CommentResource являються часткою JAX-RS. За допомогою бібліотек як Jackson, серіалізація та десеріалізація об’єктів Java в JSON реалізується автоматично.

Рис. 4.23 показує ProjectService, що виконує функції сервісного рівня в архітектурі застосунку, надаючи високорівневий API, щоб робити з проєктами. Цей сервіс відділяє інші компоненти програми від деталей доступу до даних та їх перетворення, позволяючи взаємодіяти з проєктами без необхідності знати

внутрішні аспекти бази даних чи деталі трансформації даних [32].

```
@ApplicationScoped
public class ProjectService {

    @Inject
    ProjectDao dao;
    @Inject
    ProjectTransformer transformer;

    public void save(ProjectDto projectDto) {
        dao.save(transformer.transform(projectDto));
    }
    public List<ProjectDto> getAll() {
        return dao.findAll().all().stream().map(transformer::transform).collect(Collectors.toList());
    }
    public void update(ProjectDto projectDto) {
        dao.update(transformer.transform(projectDto));
    }
    public boolean delete(ProjectDto projectDto) {
        return dao.delete(transformer.transform(projectDto));
    }
    public CompletionStage<ProjectDto> getById(String projectId) {
        return transformer.transform(dao.findByIdAsync(projectId));
    }
}
```

Рисунок 4.23 – class ProjectService

ProjectService містить різні методи для взаємодії з сутністю Project. Метод save отримує ProjectDto, перетворює його на сутність Project за допомогою ProjectTransformer і передає для зберігання в ProjectDao. Метод getAll через ProjectDao отримує всі проєкти, трансформує їх у ProjectDto і відправляє назад у вигляді списку. Метод update, аналогічно save, трансформує ProjectDto і передає для апдейту в ProjectDao. Метод delete видаляє проєкт, трансформуючи ProjectDto в сутність Project і передаючи операцію видалення в ProjectDao. Метод getById отримує проєкт за ідентифікатором асинхронно, користуючись ProjectDao та трансформуючи його в ProjectDto [33].

```
private final StampedLock stampedLock = new StampedLock();

@Inject
UserService userService;

@SneakyThrows
public CommentDto transform(Comment comment) { ...
}

public Comment transform(CommentDto dto) { ...
}

private LocalDateTime getNowDateTime() {
    long stamp = stampedLock.tryOptimisticRead();
    var now = LocalDateTime.now();
    if (!stampedLock.validate(stamp)) {
        stamp = stampedLock.writeLock();
        try {
            now = LocalDateTime.now();
        } finally {
            stampedLock.unlockWrite(stamp);
        }
    }
    return now;
}
```

Рисунок 4.24 – Частка основної функціональності CommentTransformer class

Рис. 4.24 показує застосування оптимістичної моделі читання за допомогою StampedLock в CommentTransformer.

StampedLock – це механізм синхронізації в Java, який надає більш гнучкий контроль за доступом до ресурсів у багатопотокових програмах. Він поєднує функціональність двох інших примітивів синхронізації: ReadWriteLock і ReentrantLock, але також вводить концепцію оптимістичного читання. У загальному, StampedLock – це потужний інструмент для зменшення блокувань і

підвищення ефективності при роботі з багатопотоковими програмами. [34].

Підхід модернізований для ситуацій, коли записи виконуються рідко, що дозволяє кільком потокам без блокувань читать одночасно, але при цьому забезпечує механізм для гарантованого узгодження даних у разі потреби.

```
public CompletionStage<ProjectDto> getById(String projectId) {
    return transformer.transform(dao.findByIdAsync(projectId));
}
```

Рисунок 4.25 –Асинхронна функціональність class ProjectService

Рисунок 4.25 показує застосування асинхронного трансформування з Project у ProjectDto. "#getById" – це метод який асинхронно отримує проєкт за його ідентифікатором. Повертає він CompletionStage, що демонструє майбутній результат обчислень. Метод "#findByIdAsync" в DAO асинхронно діє з базою даних, тому ця операція неблокуюча [35].

Далі результат трансформується з entity в DTO, це робить метод трансформатора "#transform". Ця трансформація здійснюється асинхронно, що забезпечує неблокуючий характер всієї операції [36].

РОЗДІЛ 5
ТЕСТУВАННЯ REST API

Ця частина роботи направлена на тестування REST API, що є важливим етапом у процесі розробки веб-застосунків, оскільки воно допомагає гарантувати якість, стабільність і безпеку взаємодії між різними програмними компонентами.

Тестування API включає в себе процес перевірки роботи API для забезпечення його коректності, безпеки та ефективності. Включає в себе функціональне тестування, тестування безпеки, сумісності, надійності, тестування відповіді на неправильні запити. Це дає змогу знаходити та правити помилки на ранніх етапах розробки, що значно зменшує витрати на їх виправлення та знижує ризики.

Табл. 5.1 містить опис тестових сценаріїв для REST-endpoint «Статистика проєкту». Мета тестових сценаріїв - це покриття всіх доступних методів у класі ProjectStatisticResource.

Таблиця 5.1 – Тестові сценарій контролера статистики проєкту (ProjectStatisticResource)

Сценарій	Метод	URL	Параметри	Очікуваний результат
Перевірка успішного отримання статистики проєкту	GET	/statistic/projectId/{projectId}	projectId (ID проєкту)	Успішний відгук (HTTP 200) з правильними даними статистики проекту.
Перевірка відгуку на недійсний ID проекту	GET	/statistic/projectId/{invalidId}	invalidId (недійсний ID проекту)	Помилковий відгук (наприклад, HTTP 404).

Табл. 5.2 містить ключові сценарії для REST-контролера «Коментарі», що відповідають сучасним вимогам.

Таблиця 5.2 – Тестові сценарії для контролера коментарів (CommentResource)

Сценарій	Метод	URL	Параметри	Очікуваний результат
Перевірка отримання списку коментарів	GET	/comments	відсутні	У відповіді маємо список коментарів.
Перевірка додавання нового коментаря	POST	/comments	Дані коментаря (формат JSON)	Створення коментаря (HTTP 201).
Перевірка видалення коментаря	DELETE	/comments/ {commentId}	commentId (ID коментаря)	Видалення коментаря (HTTP 200).

Табл. 5.3 містить CRUD-операції REST-контролеру «Статус», котрі дають змогу керувати сутністями статусу, зокрема: створювати, читати, оновлювати та видаляти.

Таблиця 5.3 – Тестові сценації контролеру статусів тікетів (TicketStatusResource)

Сценарій	Метод	URL	Параметри	Очікуваний результат
Перевірка отримання списку статусів	GET	/status	відсутні	У відповіді маємо список статусів.
Перевірка створення нового статусу	POST	/status	Дані статусу (в форматі JSON)	Створення статусу (HTTP 201).
Перевірка оновлення статусу	PUT	/status/ {statusId}	statusId (ID статусу), оновлені дані статусу (в форматі JSON)	Оновлення статусу (HTTP 200).
Перевірка видалення статусу	DELETE	/status/ {statusId}	statusId (ID статусу)	Видалення статусу (HTTP 200).

Табл. 5.4 містить CRUD-операції (створення, читання, оновлення та видалення) REST-контролеру «Користувач»

Таблиця 5.4 – Тестові сценації контролеру користувачів (UserResource)

Сценарій	Метод	URL	Параметри	Очікуваний результат
Перевірка отримання списку користувачів	GET	/users	відсутні	Список користувачів
Перевірка додавання нового користувача	POST	/users	Дані користувача (в форматі JSON)	Створення користувача (HTTP 201).
Перевірка оновлення даних користувача	PUT	/users/ {userId}	userId (ID користувача), оновлені дані користувача (в форматі JSON)	Оновлення даних користувача (HTTP 200).
Перевірка видалення користувача	DELETE	/users/ {userId}	userId (ID користувача)	Видалення користувача (HTTP 200).

Табл. 5.5 містить CRUD-операції REST-контролеру «Проєкт» та аналогічну функціональність керування сутністю проєкту.

Таблиця 5.5 – Тестові сценарії контролеру проектів (ProjectResource)

Сценарій	Метод	URL	Параметри	Очікуваний результат
Перевірка отримання списку проєктів	GET	/projects	відсутні	Список проєктів
Перевірка створення нового проєкту	POST	/projects	Дані проєкту (в форматі JSON)	Створення проєкту (HTTP 201).
Перевірка оновлення проєкту	PUT	/projects/ {projectId}	projectId (ID проєкту), оновлені дані проєкту (в форматі JSON)	Оновлення проєкту (HTTP 200).
Перевірка видалення проєкту	DELETE	/projects/ {projectId}	projectId (ID проєкту)	Видалення проєкту (HTTP 200).

Табл. 5.6 містить CRUD-операції REST-контролеру «Заявка» та аналогічну функціональність управління сутністю тікету.

Таблиця 5.6 – Тестові сценарії контролеру заявок (TicketResource)

Сценарій	Метод	URL	Параметри	Очікуваний результат
Перевірка отримання списку заявок	GET	/tickets	відсутні	Список заявок
Перевірка створення нової заявки	POST	/tickets	Дані заявки (в форматі JSON)	Створення заявки (HTTP 201).
Перевірка оновлення заявки	PUT	/tickets/ {ticketId}	ticketId (ID заявки), оновлені дані заявки (в форматі JSON)	Оновлення заявки (HTTP 200).
Перевірка видалення заявки	DELETE	/tickets/ {ticketId}	ticketId (ID заявки)	Видалення заявки (HTTP 200).

Тестування всіх точок інтеграції з бекендом виконані вручну, результати тестів пройшли порівняння з табличними значеннями та очікуваними результатами.

РОЗДІЛ 6
ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОБОТИ

6.1 Аналіз і підготовка вихідних даних

Оцінка потреби у покращенні якості та ефективності створення веб-застосунку для технічної підтримки базується на результатах досліджень і аналізу. Це включає інформацію, отриману під час переддипломної практики, а також матеріали з внутрішніх документів, систем документообігу, технічних звітів, дані про поточні методи обслуговування та підтримки, а також аналіз існуючих рішень на ринку.

Зібрані дані дозволяють визначити основні функціональні вимоги до веб-застосунку, порівняти його характеристики з наявними ринковими рішеннями та оцінити можливі переваги для кінцевих користувачів.

6.2 Опис характеристик продукту

Продукт, запропонований для виробництва та продажу, можна описати за такою структурою:

- характеристика продукту;
- призначення продукту, область використання;
- ключові особливості продукту;
- послуги, що надаються в рамках сервісної підтримки;
- позиціонування продукту на ринку;
- захист споживчих якостей;

Для створення цього веб-застосунку використовувалися сучасні технології та інструменти, зокрема мова програмування Java, REST для реалізації API, Cassandra, HTML і JavaScript для створення інтерфейсу користувача. Quarkus сприяв впровадженню багаторівневої та ефективної

архітектури, а Docker значно полегшив процес розгортання та управління контейнерами.

Веб-застосунок розроблений для підвищення ефективності та якості технічної підтримки, забезпечуючи користувачам зручні інструменти для вирішення технічних завдань. Основний функціонал включає обробку запитів, візуалізацію даних і контролю робочих процесів.

Цільова аудиторія застосунку включає сервісні центри, IT-компанії, та підприємства, які потребують ефективних інструментів для автоматизації своїх сервісів. Відмінною рисою продукту є його здатність до інтеграції з іншими системами, що забезпечує гнучкість застосунку.

Продукт буде представлено на ринку як всебічно вигідне рішення, яке сприяє високому рівню автоматизації та модернізації процесів технічної підтримки. Його ключовими перевагами є багатофункціональність, продуктивність, надійність, простий у користуванні інтерфейс, і здатність адаптуватися до індивідуальних потреб користувача.

Крім того, веб-застосунок оснащений детальною інструкцією з використання, яка допомагає користувачам швидко освоїти новий інструмент. Інструкція містить як текстові пояснення, так і графічні елементи, що наочно демонструють різноманітні функції застосунку.

Застосунок має шанси стати провідним продуктом у своєму сегменті, пропонуючи гнучке і надійне рішення для технічної підтримки, яке поєднує адміністрування запитами з аналізом ефективності робочих процесів.

6.3 Дослідження та аналіз ринків збуту

Ключовою ціллю спостереження є сегментування ринку та оцінка ємності його окремих сегментів.

Ємність товарного ринку – це максимальний обсяг товару або послуг, який може бути реалізований на певному ринку за певний період часу, за умови, що всі умови для його реалізації оптимальні, і попит задовольняє пропозицію.

Джерелами інформації про ємність ринку є статистичні дані, галузеві та корпоративні довідники, а також бюлетені іноземної комерційної інформації (БІКИ). Аналізуючи ємність ринку та динаміку її змін, можна оцінити потенціал ринку збуту.

Сегмент ринку – це частина ринку, яка об’єднує споживачів із подібними характеристиками, потребами чи поведінкою, що дозволяє компаніям адаптувати свої товари чи послуги до конкретної групи клієнтів.

Ринок можна сегментувати різними методами, враховуючи різні чинники.

Товари можна класифікувати за їхнім призначенням чи застосуванням. Сегментація є одним із ключових інструментів маркетингу, адже від правильного вибору ринкового сегмента значною мірою залежить успіх компанії у конкурентній боротьбі.

Найважливіші характеристики сегментів ринку.

- однорідність потреб;
- розмір сегмента;
- доступність;
- рентабельність;
- конкурентність;
- стабільність;
- лояльність споживачів.

Запропонована структура бізнес-плану спрямована на здійснення сегментації ринку за однорідністю потреб, розміром сегменту та конкурентністю.

6.3.1 Сегментація ринку по споживачах. Сегментацію ринку продукту за сферами використання і категоріями споживачів відображена в табл. 6.1.

Таблиця 6.1 – Сегментація ринку (I – інженери, технологи, конструктори, фахівці з експлуатації; II – викладачі університетів, технікумів, шкіл, лаборанти; III – наукові співробітники; IV – реалізатори; V – масові споживачі).

Області використання (сегменти)	Код сегмента	Споживачі				
		I	II	III	IV	V
Підприємства IT галузі	А	x	x	x		x
ВНЗ	Б		x	x		x
Коледжі	В		x	x		x
Малі підприємства	Г	x	x	x	x	x
Роздрібна торгівля	Д				x	x
Галузеві НДІ	Е	x	x	x		x

Аналіз ємності ринку показано в табл. 6.2.

Таблиця 6.2 – Аналіз ємності сегментів ринку

Області використання (сегменти)	Кількість об’єктів, які будуть використовувати виріб	Передбачуване число продажів одному об’єкту	Передбачувана місткість сегмента
Підприємства IT галузі	2500	2	5000
ВНЗ	2000	2	2000
Коледжі	740	1	740
Малі підприємства	2000	1	1150
Роздрібна торгівля	3500	1	3500
Галузеві НДІ	100	1	100
РАЗОМ ємність ринку	10840	8	12490

6.3.2 Параметрична сегментація ринку. Табл. 6.3 показує оцінки за параметричними сегментами. Аналізуючи підсумки даних таблиці, робимо висновок, що найбільш важливими є функціонал і безпека, а найбільші вимоги ставлять сегменти А, Б і В, оскільки сума балів за цими показниками є найвищою.

Таблиця 6.3 – Параметрична сегментація ринку

Параметри	Код сегментів						Підсумкова оцінка	Відсотки до (загального підсумку)
	А	Б	В	Г	Д	Е		
Ціна	4	3	3	3	2	3	18	28.57
Зручність	2	1	1	1	1	1	7	11.11
Безпека	3	4	4	3	3	3	20	31.75
Функціонал	2	4	4	3	3	2	18	28.57
РАЗОМ	11	12	12	10	9	9	63	100

6.3.3 Сегментація ринку за ключовими конкурентами. Конкурентна спроможність товару вираховується не його "абсолютною цінністю", а відносною користю характеристик порівняно з аналогами за економічними та технічними показниками. До технічних параметрів належать: функціональність, надійність, патентна захищеність, ергономіка, естетика, іншими словами характеристики, що впливають на якість продукту. Економічні параметри включають витрати на носії, налагодження, підтримку та амортизацію.

Результати дослідження конкурентної спроможності продукту за фактором ціни наведено в табл. 6.4.

Таблиця 6.4 – Розподілення конкурентності за фактором ціни(I – Spiceworks; II – Zoho Desk; III – Trello; IV – Freshdesk; V – Hesk.)

Сегменти	Передбачуваний конкурент				
	I	II	III	IV	V
Ціновий рівень, р .:	Об’єм продажу				
Формально безкоштовно	x	x	x	x	x
Низька			x		x
Середня		x		x	
Висока					

Загалом, мета аналізу конкурентної спроможності зводиться до порівняння показників продукту, що створюється, з показниками аналогічних виробів за визначеними параметрами.

Під час аналізу конкурентів помічаємо, що конкуренти I, II та V мають доступні ціни, що ставить їх на рівень зі створюваним продуктом. Але конкуренти, як і сам продукт, є досить унікальними і універсальним рішенням бути не можуть.

IV та II мають значно ширший функціонал, але і вищу ціну.

Всі конкуренти рентабельні та мають широке розповсюдження.

6.4 Відрахування на амортизацію та витрати на електричну енергію

Стаття зазначає амортизаційну вартість на основні обладнання, що необхідне для розробки. У випадку коли вартість одиниці більше 2500 грн. (якщо менше 2500 грн, в такому випадку обладнання підпадає до категорії малоцінних та швидкозношуваних (МШП) і повністю списується на собівартість).

Вартість представлена в табл. 6.5, амортизація розраховувалась завдяки формулі (6.1)

Таблиця 6.5 – Вартість прикладних програм та обладнання

Найменування	Вартість на початок року, грн.	Норма амортизації річна, %	Призначення
Пакет програм	1200	60	Для оформлення документації і ефективного написання кода
Ноутбук	80000	60	Для написання програми, оформлення документації
Монітор	9000	60	Щоб більше виводи одночасно інформації, що пришвидшує розробку ПП
Меблі (робочий стіл, стілець)	5000	25	Розміщення техніки і програміста

Амортизацію слід нараховувати лінійним методом і обчислювати за допомогою формули:

$$А_{нар} = \frac{Ц \cdot k_{на} \cdot К_{міс}}{12}, \quad (6.1)$$

де *Ц* – ціна програм і обладнання;

$k_{На}$ – коефіцієнт амортизаційної норми;

$К_{міс}$ – місяці експлуатації.

Згідно з формулою (6.1), з урахуванням 3-місячного терміну експлуатації, амортизаційні відрахування складають $A_{нар} = 13842.5$ грн

Витрати на електроенергію для експлуатації всіх засобів розраховуються за формулою:

$$Е_n = N_n \cdot Ц_{ел} \cdot (Т \cdot К_{в.n}) \quad (6.2)$$

де N_n – потужність n – го обладнання, кВт;

$Ц_{ел}$ – ціна 1 кВт/год електроенергії, грн.;

$(Т \cdot К_{в.n})$ – час роботи n обладнання, години;

$К_{в.п.}$ – коефіцієнт використання n обладнання на протязі розробки.

Розрахунки показано в таблиці 6.6.

Таблиця 6.6 – Обчислення витрат на електроенергію

№ з/п	Обладнання	N, кВт	Т, годин	$К_{в.п.}$	$Ц_{ел}$, грн.	$Е_n$, грн.
1	ПК	0.060	280	1	2.64	44.36
2	Монітор	0.030	280	1	2.64	22.18
	Сумма	-	-	-	-	66.54

Повні затрати на електроенергію складуть:

$$Е = \sum_1^n Е_n \quad (6.3)$$

6.5 Оцінка конкурентної спроможності

У цій роботі ми зосередимося на аналізі конкурентої спроможності враховуючи наявність або відсутність аналогів.

Якщо аналоги існують, застосовуємо таку схему:

1) обираємо найбільш схожі за параметрами продукти конкурентів у цих сегментах ринку;

2) визначаємо параметри продукту, які відрізняють його від конкурентних;

3) записуємо дані в таблиці.

На основі результатів ринкових досліджень робимо висновки. Вони повинні містити характеристики ринку, на якому планується реалізація продукту, обґрунтування доцільності та перспективи розробки товару.

Щоб оцінити технічний рівень продукту потрібно порівняти його параметри з параметрами виробів конкурентів і визначити загальний показник. Розрахунок здійснюємо за формою, показаною в табл. 6.7.

Таблиця 6.7 – Розрахунок загального показника якості продукту

Технічні параметри	Одиниці виміру	Абсолютне значення параметрів		Коефіцієнт вагомості (Відносний показник)	Відносний показник враховуючи коефіцієнт вагомості, Qi
		Виріб-конкурент	Проєктований виріб		
Кількість операторів;	Шт	1200	1350	1,0	1,125
Ціна;	USD	12	5	1,0	2,4
Витрати з навчання;	USD	20	10	1,0	2
потужність споживання;	ВТ	50	50	1,0	1
Розмір застосунку;	МБайт	500	400	1,0	1,25
РАЗОМ				1,0	-

Обчислення проводилися за наступною методикою:

1) Рядок (Кількість операторів): 1350/1200=1,125

2) Рядок (Ціна): 12/5=2,4

3) Рядок (Витрати з навчання): 20/10=2

4) Рядок (Потужність споживання): 50/50=1

5) Рядок (Розмір застосунку): 500/400=1,25

Відносні одиничні показники *Qi* потрібно обчислювати за формулами в залежності від характеристики параметра.

У випадку підвищення технічного параметра ПП свідчить про поліпшення, застосовуємо формулу для розрахунку:

$$Q_i = \frac{\mathrm{P}_i}{\mathrm{P}_{ia}} \tag{6.4}$$

У випадку пониження технічного параметра ПП свідчить про поліпшення, застосовуємо формулу для розрахунку:

$$Q_i = \frac{\mathrm{P}_{ia}}{\mathrm{P}_i} \tag{6.5}$$

де P_i і P_{ia} – абсолютні значення і-го показника та виробу конкурента.

ВИСНОВКИ

У дипломній роботі проведено ретельний аналіз та створено веб-застосунок для покращення якості технічної підтримки споживачів. У проєкті розроблено програмну та архітектурну частини, що включають новітні інструменти та технології, зокрема, Cassandra для створення бази даних, Java 21, REST для розробки API, JavaScript та HTML для створення фронтенду з AngularJS. Застосування фреймворку Quarkus та Docker дозволило побудувати багаторівневу архітектуру і дало можливість зручного розгортання та легкого управління контейнерами.

Створений застосунок спрямований на підвищення ефективності та якості підтримки споживачів, надаючи швидке вирішення технічних завдань. Основні функціональні можливості включають обробку запитів, візуалізацію даних, адміністрування робочих процесів та взаємодію з системами.

Проведено економічний аналіз, котрий підтвердив, що розробка та імплементація веб-застосунку є вигідними та в перспективі мають високий рівень окупності та ефективності. Були вивчені можливості для масштабування та інтеграції додатку для покриття потреб більш широкої аудиторії, забезпечуючи багатофункціональність, продуктивність та зручний інтерфейс.

Отже, створення та імплементація проєкту є обґрунтованими та гарантують економічні вигоди в майбутньому, оскільки відповідають вимогам ринку у сфері технічної підтримки та сервісу.

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Warren M. IT Help Desks Not Just For Large Enterprises [Електронний ресурс]. – 2011. – Режим доступу: https://www.informationweek.com/business-e-business/it-help-desks-not-just-for-largeenterprises/d/d-id/1098348.
2. Fisher C.A. Manage digital assets with ITIL: Improve product configurations and service management // Journal of Digital Asset Management. - 2006. - Vol. 2, No. 1. - P. 40-49.
3. Adams J. Top 10 Best Help Desk Software for 2020 [Електронний ресурс]. – 2020 Feb 15. – Режим доступу: https://search.proquest.com/docview/2355310264.
4. Laudon K. Management Information systems / K. Laudon, D. Laudon. - Publishing EBSCO, 2014.
5. McBride D. A Guide to Help Desk Technology, Tools & Techniques / D. McBride. - MI: Course Technology, 2000. – P.358.
6. Brynjolfsson E. The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies / E. Brynjolfsson, A. McAfee. - New York: W. W. Norton & Company, 2014.
7. West D. M. The Future of Work: Robots, AI, and Automation / D. M. West. - Washington, DC: Brookings Institution Press, 2018.
8. Kunstova R. Process Support of Business and IT Management in Czech Companies / R. Kunstova // In: C. Moller, S. Chaudhry (eds.) Re-conceptualizing Enterprise Information Systems. - Lecture Notes in Business Information Processing, vol. 105. - Berlin, Heidelberg: Springer, 2012. - P. 168-181.
9. Erone D. O. Customer Service and Customer Satisfaction / D. O. Erone. – OmniScriptum GmbH & Co. KG, 2015.
10. Self Service - Deside [Електронний ресурс]. – 2019. – Режим доступу: https://www.dezide.com/products/self-service. – Дата звернення: 17 Aug 2019.

11. Westland J. Project management life cycle: a complete step-by-step methodology for initiating, planning, executing and closing the project successfully / J. Westland. - GL.: Bell & Bain, 2006. - 229 p.

12. Price B. The Best Service is No Service: How to Liberate Your Customers from Customer Service, Keep Them Happy, and Control Costs / B. Price, D. Jaffe. - San Francisco: Jossey-Bass, 2008.

13. Kurhanov D.A. Software development to minimize time costs and increase productivity in the area of communication services / D.A. Kurhanov, A.A. Azaryan // In: A.E. Kiy, S.O. Semerikov, V.N. Soloviev, A.M. Striuk (eds.) Proceedings of the 1st Student Workshop on Computer Science & Software Engineering (CS&SE@SW 2018), Kryvyi Rih, Ukraine, November 30, 2018. - CEUR Workshop Proceedings 2292. - 2018. - P. 116-127. - Режим доступу: http://ceur-ws.org/Vol2292/paper13.pdf.

14. Корнієнко М. М. Інформатика. Бази даних. Системи управління базами даних / М. М. Корнієнко, І.Д. Іванова. – Х.: Видавництво «Ранок», 2009. – 48 с.

15. Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. – Boston, MA: Addison-Wesley, 2002. – 533 с.

16. Хорстманн К. С. Java SE 8. Вступний курс / К. С. Хорстманн. – М.: «Віл'ямс», 2014. – 208 с.

17. Рач В. А. Управління проєктами: практичні аспекти реалізації стратегій регіонального розвитку / В. А. Рач, О. В. Россошанська, О. М. Медведєва. – Видавництво «К.І.С.».

18. Bloch J. Effective Java / J. Bloch. - Upper Saddle River, NJ: Addison-Wesley Professional, 2018. - 3rd ed.

19. Lokesh Gupta. "Java 21 Features (LTS): Practical Examples and Insights." Published on October 14, 2023. [Електронний ресурс] – Режим доступу до ресурсу: https://howtodoinjava.com/java/java-21-new-features/

20. Duckett J. HTML and CSS: Design and Build Websites / J. Duckett. - Indianapolis, IN: Wiley, 2011.

21. Green D., Seshadri S. AngularJS Up and Running: Enhanced Productivity with Structured Web Apps / D. Green, S. Seshadri. – Sebastopol, CA: O'Reilly Media, 2014. – 302 с.

22. Koleoso T. Beginning Quarkus Framework: Build Cloud-Native Enterprise Java Applications and Microservices. – Silver Spring, MD, USA, 2020.

23. Hibernate Validator. The Bean Validation reference implementation [Електронний ресурс]. – Режим доступу: https://hibernate.org/validator/.

24. DataStax Java Driver for Apache Cassandra [Електронний ресурс]. – Режим доступу: https://docs.datastax.com/en/developer/java-driver/4.17/.

25. Soto Bueno A. Quarkus Cookbook: Kubernetes-Optimized Java Solutions / A. Soto Bueno, J. Porter. - Sebastopol, CA: O'Reilly Media, 2020.

26. Kalali M. Developing RESTful Services with JAX-RS 2.0, WebSockets, and JSON / M. Kalali, B. Mehta. – Packt Publishing, 2013. – 128 с.

27. Allamaraju S. RESTful Web Services Cookbook / S. Allamaraju. - Sebastopol, CA: O'Reilly Media, 2010.

28. Carpenter J. Cassandra: The Definitive Guide / J. Carpenter, E. Hewitt. - Sebastopol, CA: O'Reilly Media, 2020. - 3rd ed.

29. Oaks S. Java Performance: The Definitive Guide / S. Oaks. - Sebastopol, CA: O'Reilly Media, 2014.

30. Urma R.-G. Modern Java in Action: Lambdas, streams, functional and reactive programming / R.-G. Urma, M. Fusco, A. Mycroft. - Manning Publications, 2018.

31. Kleppmann M. Designing Data-Intensive Applications. – Sebastopol, CA: O’Reilly Media, Inc., 2017.

32. Brown M. Learning Apache Cassandra / M. Brown. - Birmingham: Packt Publishing, 2015.

33. MATLAB & Simulink. M/M/1 Queuing System [Електронний ресурс]. – Режим доступу: https://www.mathworks.com/help/simevents/ug/m-m-1-queuing-system.html.

34. Erl T. Cloud Computing: Concepts, Technology & Architecture / T. Erl, R. Puttini, Z. Mahmood. - Upper Saddle River, NJ: Prentice Hall, 2013.

35. Leon A. Database management System / A. Leon, M. Leon. - Wiley Inter-Science, 2014.

36. Mayer-Schonberger V. Big Data: A Revolution That Will Transform How We Live, Work, and Think / V. Mayer-Schonberger, K. Cukier. - Boston: Houghton Mifflin Harcourt, 2013.