Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Центр післядипломної освіти

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для управління та обліку роботи сервісного центру з ремонту побутової техніки
(тема)

Виконав:
студент 4 курсу, групи ПЗПІп-22-2

Фісун Є.О.
(прізвище, ініціали)

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

Керівник проф. кафедри ПІ Єрохін А.Л.
(посада, прізвище, ініціали)

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

2024 р.

Харківський національний університет радіоелектроніки

Центр післядипломної освіти
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Фісуну Євгену Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для управління та обліку роботи сервісного центру з ремонту побутової техніки
Затверджена наказом по університету від 17.06.2024р. № 588 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 24.07.2024
3. Вихідні дані до роботи Розробити веб-застосунок для управління та обліку роботи сервісного центру, з використанням мов програмування Java та JavaScript, реляційної бази даних та логіки об'єктно-орієнтованого програмування.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	16.05.2024	*виконано*
2	Створення специфікації ПЗ	20.05.2024	*виконано*
3	Проектування ПЗ	23.05.2024	*виконано*
4	Розробка ПЗ	07.06.2024	*виконано*
5	Тестування ПЗ	15.06.2024	*виконано*
6	Оформлення пояснювальної записки	11.07.2024	*виконано*
7	Підготовка презентації та доповіді	19.07.2024	*виконано*
8	Попередній захист	21.07.2024	*виконано*
9	Нормоконтроль, рецензування	19.07.2024	*виконано*
10	Здача роботи у електронний архів	19.07.2024	*виконано*
11	Допуск до захисту у зав. кафедри	21.07.2024	*виконано*

Дата видачі завдання 16 травня 2024р.

Студент (ка)_____ Фісун Є.О.
(підпис)

Керівник роботи _____ проф. кафедри ПІ Єрохін А.Л.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 78 стор., 34 рис., 2 табл., 13 джерел.

МОВА ПРОГРАМУВАННЯ JAVA, РЕЛЯЦІЙНА БАЗА ДАНИХ, СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ, HIBERNATE, JWT, MYSQL, REACT, SPRING JPA

Об’єкт розробки – веб-застосунок «Програмна система для управління та обліку роботи сервісного центру з ремонту побутової техніки».

Мета розробки – створення веб-застосунку, для управління та обліку роботи сервісного центру з ремонту побутової техніки.

Метод рішення – середовище розробки JetBrains IntelliJ IDEA та Microsoft Visual Studio Code, мови програмування Java та JavaScript, фреймворк Spring Boot з модулем Spring Data JPA для розробки серверної частини, фреймворк React для розробки користувацького інтерфейса та система управління базами даних MySQL.

У результаті розробки створено веб-застосунок, який дозволяє взаємодіяти та зберігати дані про клієнтів та їх адреси, замовлення, майстрів, запчастини та ремонтні роботи. Також програмна система дозволяє шукати, фільтрувати інформацію, отримувати статистики, реалізовано задачу автоматизації з розсилки листів на пошту замовників.

JAVA PROGRAMMING LANGUAGE, RELATIONAL DATABASE, DATABASE MANAGEMENT SYSTEM, HIBERNATE, JWT, MYSQL, REACT, SPRING JPA

The object of development is the web application "Software System for Managing and Accounting the Operations of a Household Appliance Repair Service Center"

The goal of development is to create a web application for managing and accounting the work of a household appliance repair service center.

The solution method includes the development environments JetBrains IntelliJ IDEA and Microsoft Visual Studio Code, programming languages Java and JavaScript, the Spring Boot framework with the Spring Data JPA module for backend development, the React framework for user interface development, and the MySQL database management system.

As a result of the development, a web application was created that allows interaction and storage of data about customers and their addresses, orders, technicians, spare parts, and repair work. The software system also enables searching, filtering information, generating statistics, and automating the task of sending emails to customers.

Я, Фісун Євген Олександрович, студент гр. ПЗПІп-22-2, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система для управління та обліку роботи сервісного центру з ремонту побутової техніки», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

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

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – Програмне забезпечення

ПІБ – Прізвище, ім’я, по-батькові

ПС – Програмна система

СУБД – Система управління базами даних

СЦ – Сервісний центр

API – Application Programming Interface

CSS – Cascading Style Sheets

DOM – Document Object Model

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

JPA – Java Persistence API

JWT – JSON Web Token

ORM – Object-relational mapping

REST – Representational State Transfer

SQL – Structured Query Language

UI – User Interface

UML – Unified Modeling Language

UX – User Experience

ВСТУП

Темою кваліфікаційної роботи є розробка веб-застосунка для управління та обліку роботи сервісного центру.

У сучасному світі сервісні центри з ремонту побутової техніки є невід'ємною складовою сфери обслуговування технічних пристроїв та апаратури. Однак, ручне ведення обліку замовлень та взаємодія з клієнтами може бути складним і ресурсозатратним процесом. Тому, для оптимізації роботи сервісних центрів, виникає необхідність у впровадженні інформаційної системи, яка дозволить автоматизувати процес обліку замовлень та поліпшити комунікацію з клієнтами.

Актуальність даної роботи полягає у тому, що автоматизація обліку замовлень та оптимізація процесів роботи сервісних центрів з ремонту побутової техніки сприяє підвищенню ефективності та якості надання послуг, забезпечує зручну взаємодію з клієнтами та знижує час, необхідний для обробки замовлень. Така система може бути корисною для різних організацій, що надають послуги з ремонту побутової техніки та мають потребу в ефективному обліку та керуванні замовленнями та наявною базою клієнтів.

Таким чином, робота спрямована на створення інформаційної системи (веб-застосунку), яка допоможе оптимізувати процес ведення обліку замовлень сервісного центру побутової техніки, поліпшити комунікацію з клієнтами та підвищити ефективність роботи.

Веб-застосунок є досить легким для розуміння та швидкого пристосування, отже людина з будь-яким досвідом зможе з легкістю його використовувати.

В ході створення веб-застосунку були використані середа розробки IntelliJ IDEA для розробки бек-енд частини застосунку, середа розробки Microsoft Visual Studio Code для розробки фронт-енд частини, мови програмування Java та JavaScript відповідно, фреймворки Spring та React, та система управління базами даних MySQL.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Сервісний центр – це організація, що займається наданням послуг з обслуговування та ремонту побутової техніки. Діяльність сервісних центрів включає гарантійний та після гарантійний ремонт. Організаційно структура сервісних центрів може відрізнятися в залежності від конкретної компанії, її масштабів та специфіки бізнесу. Проте, загалом вона складається з таких підрозділів:

- відділ прийому замовлень та обслуговування клієнтів. Цей підрозділ відповідає за прийом заявок від клієнтів та реєстрацію їх у системі. Тут клієнти можуть надати інформацію про проблему зі своєю побутовою технікою. Також цей відділ забезпечує взаємодію з клієнтами та вирішує їхні запити та скарги;
- відділ технічної підтримки. Цей підрозділ займається технічним обслуговуванням та ремонтом побутової техніки. Він може бути розділений на декілька підрозділів залежно від спеціалізації робіт, наприклад, на електроніку, механіку, холодильну техніку тощо;
- відділ логістики та складу. Цей підрозділ відповідає за складський облік запасних частин, придбання необхідного обладнання та матеріалів, а також за доставку запасних частин на місце ремонту;
- відділ адміністрації та фінансів. Цей підрозділ відповідає за управління фінансами та документообігом в сервісному центрі. Він може включати в себе відділ кадрів, відділ бухгалтерії та відділ адміністративної підтримки;
- технічний відділ – це відділ, відповідальний за проведення технічного обслуговування та ремонту побутової техніки. Він забезпечує проведення ремонтних робіт відповідно до стандартів виробників, контролює якість виконаних робіт, відповідає за збереження запасних частин та інструментів для виконання ремонтів. У технічному відділі працюють майстри, які володіють необхідними знаннями та навичками для проведення ремонтів.

Мета роботи сервісного центру – надання якісних та своєчасних послуг з ремонту та обслуговування побутової техніки, а також задоволення потреб клієнтів у цих послугах. Для досягнення цієї мети сервісний центр повинен мати належну організаційну структуру, висококваліфікований персонал, необхідні матеріально-технічні засоби та інструменти для проведення ремонтних робіт та обслуговування техніки. Окрім цього, сервісний центр повинен мати ефективну систему контролю якості виконаної роботи та забезпечення безпеки праці для свого персоналу.

Робота сервісного центру пов’язана:

- з забезпеченням надійного та якісного обслуговування техніки, а також задоволення потреб та очікувань клієнтів.;
- з наданням послуг з заміни запчастин, профілактики техніки, консультації клієнтів та інші супутні послуги.

Далі будемо розглядати процес з організації та обліку робочого процесу сервісного центру.

Цей процес включає в себе наступні етапи:

- клієнт звертається до сервісного центру з проблемою своєї техніки;
- клієнт отримує консультацію від спеціаліста відділу прийому замовлень та обслуговування клієнтів щодо можливого ремонту та його вартості;
- клієнт подає заявку на ремонт техніки;
- спеціаліст відділу прийому замовлень та обслуговування клієнтів (або в нашій ПС - майстер) реєструє замовлення та вносить дані про клієнта, проблему з технікою та обрану ним послугу;
- майстер сервісного центру забирає техніку клієнта для ремонту;
- майстер діагностує техніку, встановлює несправність та необхідні запчастини для ремонту;
- спеціаліст відділу логістики та складу (або в нашій ПС - майстер) оформлює замовлення на запчастини та розпочинає їх пошук у постачальників;
- після отримання запчастин, майстер проводить ремонт техніки та перевіряє його якість;

– спеціаліст відділу прийому замовлень та обслуговування клієнтів (або в нашій ПС - майстер) повідомляє клієнта про готовність його техніки та вартість ремонту;
– клієнт отримує свою техніку та сплачує за ремонт.

Ведення журналу виконаних робіт та сповіщення клієнтів достатньо часозатратна операція, тому спеціалісти відділу прийому замовлень та обслуговування клієнтів потребують тут певної підтримки (майбутня задача автоматизації).

1.2 Виявлення та вирішення проблем

При аналізі роботи сервісних центрів з ремонту побутової техніки було виявлено низку проблем, що значно ускладнюють ефективне управління та облік робіт.

Основні проблеми включають:

– розрізненість інформації – у багатьох сервісних центрах інформація про клієнтів, замовлення та виконані роботи зберігається у різних форматах (паперові записи, електронні таблиці тощо). Це ускладнює доступ до даних, їх оновлення та аналіз;
– відсутність автоматизації – відсутність автоматизованої системи управління призводить до того, що процеси прийому замовлень, обліку виконаних робіт, управління запасами запчастин і матеріалів виконуються вручну. Це збільшує ймовірність людських помилок і затримок;
– обмежені можливості обробки даних – більшість існуючих рішень не забезпечують повноцінної обробки даних. Наприклад, відсутність можливостей для сортування, пошуку, фільтрації та аналітики ускладнює управління замовленнями і запасами;
– недостатній контроль за запасами – відсутність єдиної системи обліку запасів призводить до ситуацій, коли необхідні запчастини можуть бути відсутні на складі в момент потреби. Це затримує ремонтні роботи та знижує рівень обслуговування клієнтів;

- відсутність аналітики та звітності – недостатня аналітика і звітність обмежують можливості керівництва у прийнятті обґрунтованих рішень щодо оптимізації роботи сервісного центру. Немає можливості швидко отримати інформацію про найчастіше використовувані запчастини, найбільш проблемні моделі техніки тощо;
- відсутність централізованої бази даних – багато сервісних центрів не мають централізованої бази даних для зберігання інформації про клієнтів, замовлення та виконані роботи, що ускладнює доступ до актуальних даних та їх обробку.

Для вирішення виявлених проблем пропонується розробка веб-застосунку, який автоматизує основні процеси управління та обліку роботи сервісного центру з ремонту побутової техніки.

Основні рішення включають:

- запровадження централізованої бази даних для зберігання інформації про клієнтів, замовлення, запасні частини та виконані роботи. Це дозволить забезпечити доступ до актуальних даних у режимі реального часу;
- автоматизація процесів прийому замовлень, обліку виконаних робіт та управління запасами. Це знизить ймовірність людських помилок, прискорить обробку замовлень та покращить ефективність роботи сервісного центру;
- обробка даних – реалізація функціоналу для сортування, пошуку та фільтрації даних. Це дозволить швидко знаходити необхідну інформацію про клієнтів, запчастини, моделі техніки та виконані роботи;
- управління запасами – впровадження модуля для управління запасами, який буде відстежувати наявність запчастин на складі, їх витрати та поповнення. Система буде автоматично повідомляти про необхідність замовлення запчастин, кількість яких знизилась до критичного рівня;
- аналітика та звітність – розробка аналітичних інструментів для керівництва сервісного центру. Система буде генерувати звіти про найчастіше використовувані запчастини, найбільш проблемні моделі

техніки, кількість звернень клієнтів тощо. Це дозволить приймати обґрунтовані управлінські рішення;

– забезпечення захисту даних клієнтів та замовлень шляхом впровадження сучасних методів шифрування та контролю доступу. Це забезпечить конфіденційність та цілісність інформації.

Запропоновані рішення дозволять значно підвищити ефективність роботи сервісного центру, знизити кількість помилок в обліку, покращити рівень обслуговування клієнтів та забезпечити керівництво необхідною аналітикою для прийняття обґрунтованих рішень.

1.3 Аналіз існуючих аналогів

Наразі існує багато застосунків для ведення управлінського обліку на підприємстві. Найбільш популярними серед них можна визначити «Orderry»[2], «Zendesk»[3], «ServiceMax»[4] та інші, але програмою, що найбільше відповідає тематиці даного курсового проекту є програма «RemOnline»[5]. За допомогою даного застосунку працівники сервісного центру можуть управляти замовленнями, клієнтською базою, вести складський облік товарів та запчастин, вести бухгалтерію сервісного центру. Програма дозволяє налаштовувати інтерфейс та рівні доступу до даних в залежності від ролі користувача: Інженер, Виконавець, Менеджер, Приймальник, Повний доступ та інших (див. рис. 1.1).

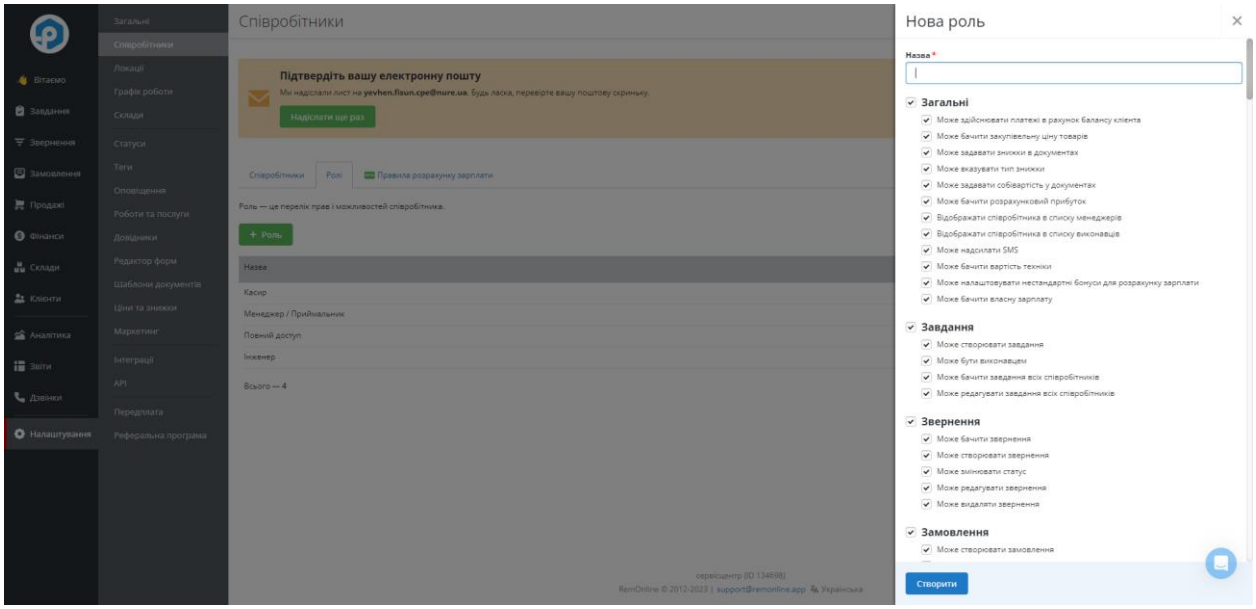


Рисунок 1.1 – Інтерфейс налаштування рівнів доступу програми «RemOnline» (за даними [5])

Програма дозволяє гнучко створювати і налаштовувати категорії побутової техніки під час створення нового замовлення (див. рис. 1.2). Для швидкої роботи з замовленням застосовуються пошук за кодом, найменуванням та артикулом. Також програма дозволяє працювати з декількома складами запчастин, техніки, що знаходиться у ремонті та ін..

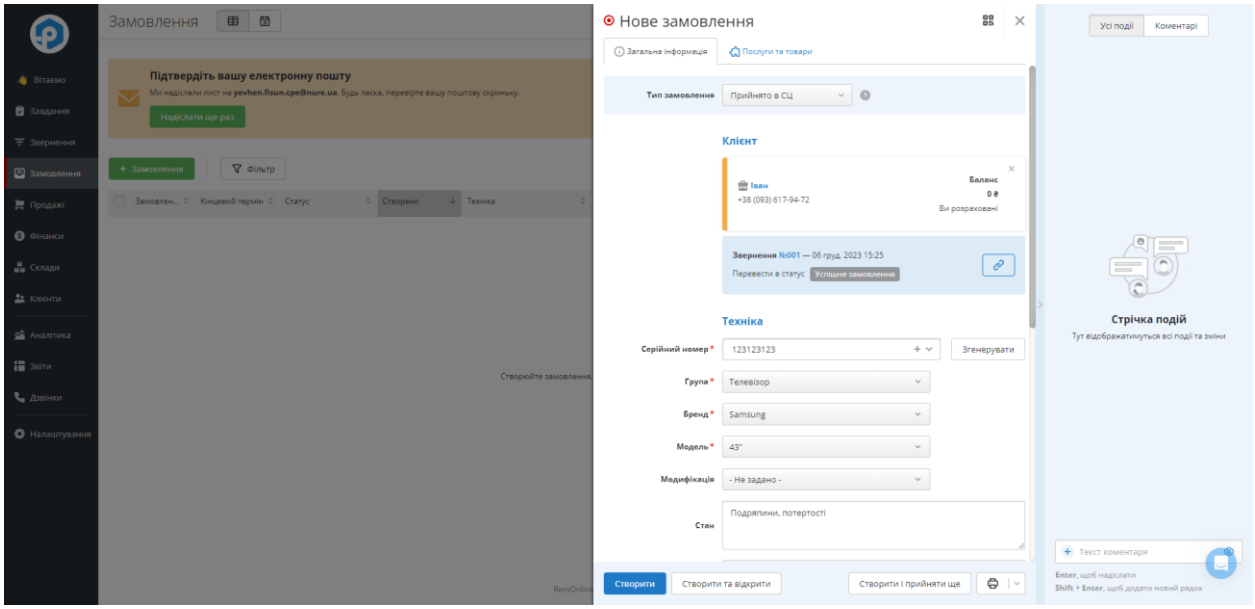


Рисунок 1.2 – Інтерфейс створення нового замовлення (за даними [5])

За допомогою додатку менеджери підприємства здійснюють взаєморозрахунки з замовниками та постачальниками запасних частин, облік складу та продажів (див. рис. 1.3).

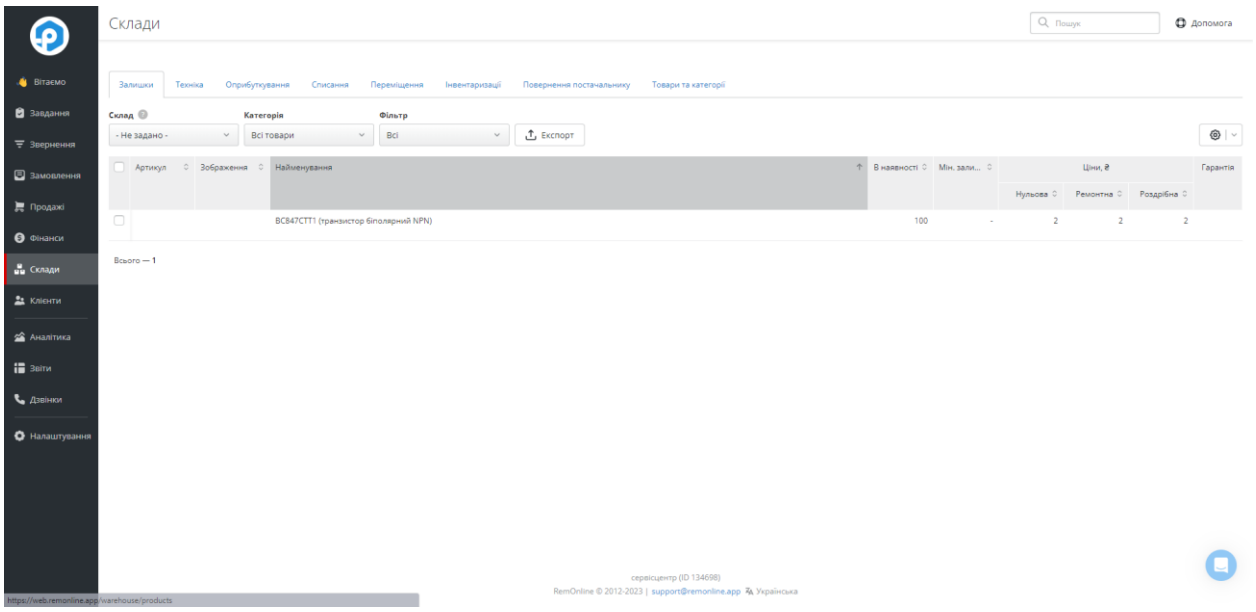


Рисунок 1.3 – Облік складу в програмі «RemOnline» (за даними [5])

За допомогою додатку менеджер має змогу формувати складські звіти, що запобігає проблемам з постачанням запасних частин, також на вкладці «Аналітика» можна побачити показники компанії – статистику замовлень та платежів за сьогодні, кількість замовлень в роботі, отриманих та поточну касу СЦ, що допомагає аналізувати якість обслуговування клієнтів.

Таким чином, перевагами програми «RemOnline» є її зручність у використанні, обслуговування всієї діяльності підприємства, оперативний супровід розробниками, а з недоліків – складність первинного налаштування додатку, доступ тільки на платній основі.

1.4 Постановка задачі

Основними задачами, які необхідно вирішити під час розробки програмної системи, є:

- створення зручного інтерфейсу для прийому та обробки замовлень;
- розробка модуля для управління запасами запчастин;
- впровадження системи обліку виконаних робіт і ведення бази даних клієнтів;
- створення аналітичних інструментів для керівництва.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Розробка програмної системи для управління та обліку роботи сервісного центру з ремонту побутової техніки вимагає чіткого визначення функціональних і нефункціональних вимог. У цьому розділі розглядаються основні вимоги до системи, що забезпечують її ефективне функціонування та задоволення потреб користувачів.

2.1 Функціональні вимоги

Функціональні вимоги визначають конкретні функції, які повинна виконувати система. Вони включають:

а) система повинна забезпечувати управління клієнтами:

1) збереження інформації про клієнтів, включаючи ПІБ, контактні дані, адреси та історію замовлень;
2) додавання нових клієнтів до бази даних;
3) редагування інформації про існуючих клієнтів;
4) видалення клієнтів з бази даних із підтвердженням користувачем;

б) система повинна надавати можливість управління майстрами:

1) збереження інформації про майстрів, включаючи ПІБ, спеціалізацію та контактні дані;
2) додавання нових майстрів до бази даних;
3) редагування інформації про існуючих майстрів;
4) видалення майстрів з бази даних із підтвердженням користувачем;

в) система повинна забезпечувати управління запасними частинами:

1) збереження інформації про запасні частини, включаючи назву, кількість, дату закупівлі та постачальника;
2) додавання нових запасних частин до бази даних;
3) редагування інформації про існуючі запасні частини;
4) видалення запасних частин з бази даних із підтвердженням користувачем;

5) автоматичне відстеження кількості запасних частин на складі та повідомлення про необхідність їх поповнення;

г) система повинна забезпечувати управління замовленнями:

1) збереження інформації про замовлення, включаючи дату, клієнта, модель техніки, опис поломки, майстра та статус виконання;
2) додавання нових замовлень;
3) редагування інформації про існуючі замовлення;
4) видалення замовлень із підтвердженням користувачем;
5) зміна статусу замовлення (нове, в роботі, виконане);
6) автоматичне формування гарантійного чеку після завершення ремонту;

д) система повинна надавати можливість обробляти дані:

1) сортувати клієнтів за ПІБ, майстрів за ПІБ, список запасних частин за назвою та кількістю, виконані ремонтні роботи за датою виконання;
2) здійснювати пошук інформації про клієнта за його повним або частковим ПІБ, за назвою моделі побутової техніки, про запасну частину за її повною або частковою назвою;
3) фільтрувати інформацію про виконані роботи за датою, вартістю виконаних робіт, замовлення за статусом виконання;

е) система повинна забезпечувати формування статистики та звітів:

1) отримання статистики про запчастини, кількість яких на складі менше 3 шт. (назва запчастини, кількість на складі, остання дата закупівлі, постачальник, кількість використаних запчастин за останній місяць).
2) отримання списку клієнтів, що найчастіше звертались (ПІБ клієнта, кількість звернень, вартість наданих послуг для клієнта);
3) отримання топ-5 моделей техніки, найчастіше надходивших на ремонт (назва моделі, кількість звернень);

4) отримання кількості звернень клієнтів за кожен місяць поточного року (місяць, кількість звернень за місяць);
5) формування та друк гарантійного чеку на виконану роботу (номер чеку, дата виконаної роботи, ПІБ клієнта, адреса клієнта, модель техніки, деталі поломки, вартість ремонту);
6) формування та друк звіту про наявні запасні частини на складі (назва, кількість, дата закупівлі, постачальник).

2.2 Нефункціональні вимоги

Нефункціональні вимоги визначають загальні характеристики системи, такі як продуктивність, безпека, масштабованість тощо. Вони включають:

- продуктивність – система повинна забезпечувати високу швидкість обробки запитів та швидкий доступ до даних. Всі операції додавання, редагування та видалення повинні виконуватися без затримок;
- безпека – система повинна забезпечувати захист даних клієнтів та замовлень шляхом впровадження сучасних методів шифрування та контролю доступу. Тільки авторизовані користувачі повинні мати доступ до конфіденційної інформації;
- масштабованість – система повинна бути розроблена з урахуванням можливості масштабування для обробки збільшеного обсягу даних та користувачів. Це включає використання сучасних архітектурних підходів, таких як мікросервіси;
- зручність використання – інтерфейс користувача повинен бути інтуїтивно зрозумілим і зручним для використання. Для цього використовуються сучасні технології, такі як React та Bootstrap, що дозволяють створити адаптивний та привабливий інтерфейс;
- надійність – система повинна забезпечувати надійне зберігання та обробку даних. Усі операції з даними повинні бути транзакційними, що гарантує цілісність даних навіть у разі збоїв;

– сумісність – система повинна бути сумісною з різними веб-браузерами та пристроями. Веб-застосунок повинен коректно працювати на різних операційних системах і мобільних пристроях.

2.3 Технологічні вимоги

До технологічних вимог відносяться наступні:

– серверна частина – для реалізації серверної частини використовується Spring Boot, що забезпечує високу продуктивність і надійність. Spring Boot дозволяє швидко розробляти і розгортати веб-застосунки з мінімальною конфігурацією;
– база даних – для зберігання даних використовується реляційна база даних MySQL, яка забезпечує високу продуктивність та надійність. Інтеграція з базою даних реалізована за допомогою Spring JPA, що забезпечує ORM і спрощує роботу з базою даних;
– клієнтська частина – розроблена з використанням React, що дозволяє створювати динамічні та інтерактивні інтерфейси користувача. Використання React Router DOM забезпечує зручну маршрутизацію на клієнтській стороні;
– сторонні бібліотеки – для створення адаптивного та сучасного дизайну інтерфейсу використовується Bootstrap. Для виконання HTTP-запитів до серверної частини застосунку використовується Axios, що забезпечує зручну і ефективну роботу з API.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі детально розглянемо архітектуру, проєктування системи зберігання даних, розробку дизайну системи. Розділ містить підрозділи, які відображають сутність роботи, зокрема UML проєктування ПЗ, проєктування архітектури ПЗ, структури зберігання даних, опис методів, створення UI/UX.

3.1 UML проєктування ПЗ

UML проєктування є важливою частиною розробки програмного забезпечення, оскільки допомагає візуалізувати структуру та поведінку системи. Для нашого веб-застосунку було використано кілька типів UML діаграм: діаграми варіантів використання, класів, послідовностей та активностей.

3.1.1 Діаграма варіантів використання

Діаграма варіантів використання описує основні сценарії взаємодії користувачів із системою. Основними акторами нашої системи є:

- адміністратор – додає, редагує, видаляє та переглядає інформацію про клієнтів, майстрів та запасні частини;
- майстер – виконує ремонтні роботи, оновлює статус замовлень;
- клієнт – отримує інформацію про статус своїх замовлень.

Основні варіанти використання включають управління клієнтами, майстрами, запчастинами, замовленнями, обробку даних та автоматизацію задач.

Після детального аналізу була створена Use case діаграма (див. рис. 3.1).

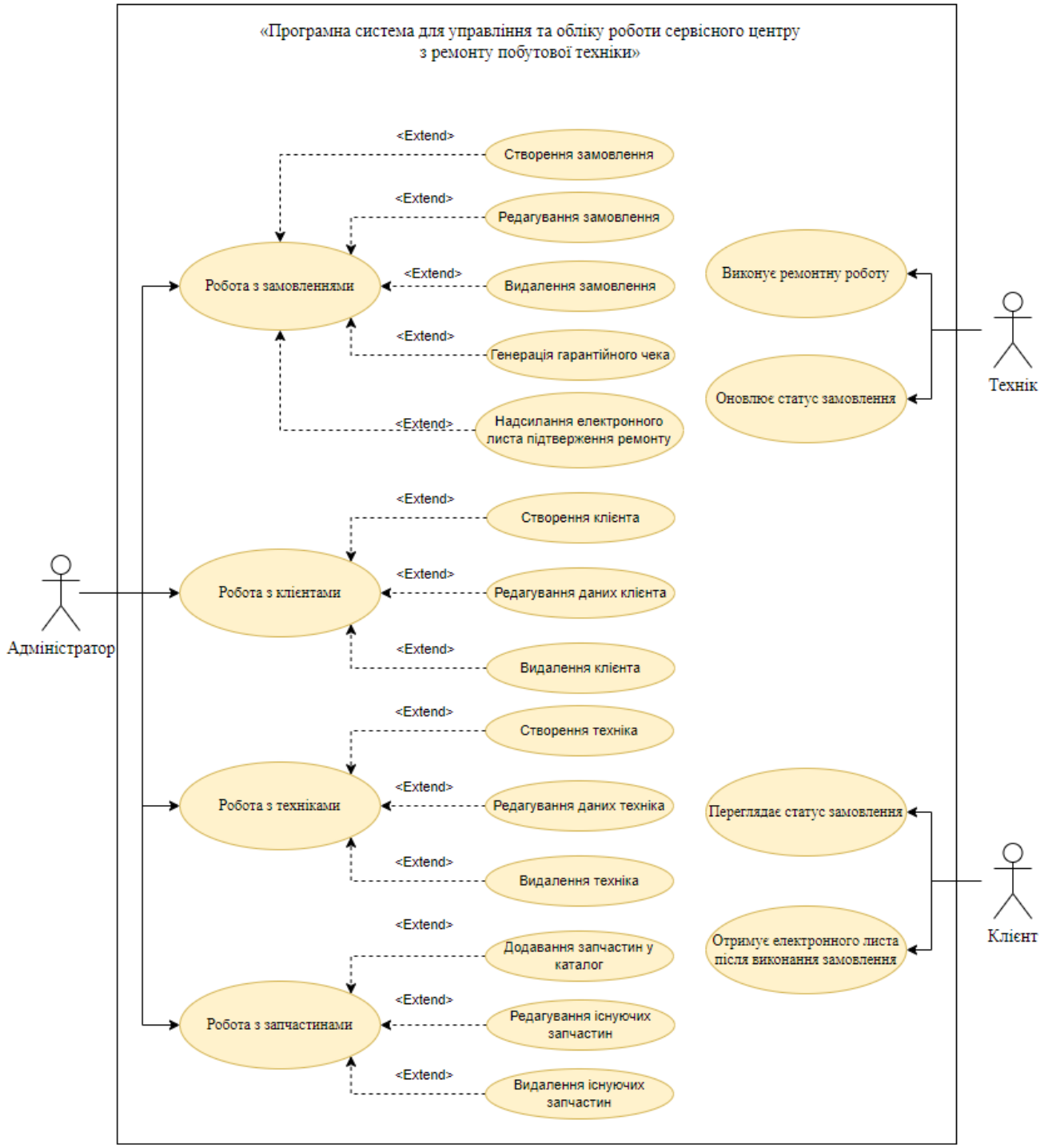


Рисунок 3.1 – Use case діаграма веб-застосунку (рисунок виконано самостійно)

3.1.2 Діаграма класів

Діаграма класів відображає структуру системи з ключовими класами, їх атрибутами та методами. Основні класи включають:

- client – включає атрибути для зберігання інформації про клієнтів та методи для додавання, редагування, видалення і перегляду даних клієнтів;

- address – містить атрибути для зберігання інформації про адресу клієнтів та методи для додавання, редагування, видалення і перегляду даних клієнтів;
- employee – включає атрибути для зберігання інформації про майстрів та методи для управління даними про майстрів;
- model – включає атрибути для зберігання інформації про моделі побутової техніки та методи для управління даними про моделі;
- order – містить атрибути для інформації про замовлення та методи для управління замовленнями;
- repairWork – включає атрибути для зберігання інформації про виконану ремонтну роботу та методи для управління цією сутністю;
- sparePart – містить атрибути для інформації про запасні частини та методи для управління ними.

Після аналізу була створена наступна діаграма класів (див. рис. 3.2).

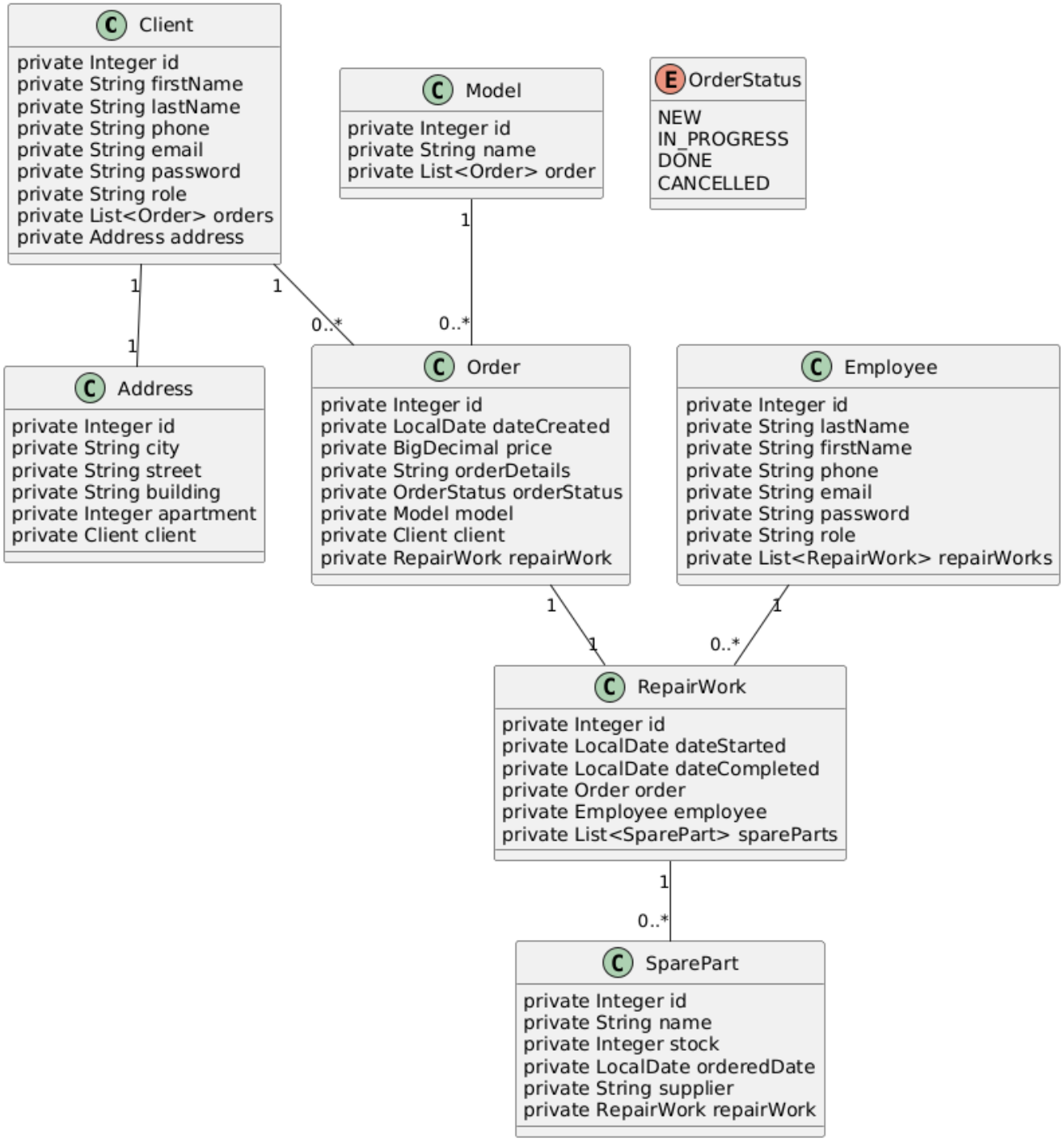


Рисунок 3.2 – Діаграма класів веб-застосунку (рисунок виконано самостійно)

3.1.3 Діаграма послідовності

Діаграма послідовності демонструє взаємодію між об'єктами в системі для певного сценарію. Наприклад, сценарій додавання нового замовлення:

– клієнт звертається до адміністратора із запитом на ремонт;

– адміністратор додає нове замовлення в систему, заповнюючи необхідні дані;

– система зберігає нове замовлення у базі даних;
– система повертає підтвердження про успішне додавання замовлення адміністратору;
– адміністратор інформує клієнта про прийняття замовлення.

Як результат дослідження була побудована наступна діаграма послідовності (див. рис. 3.3).

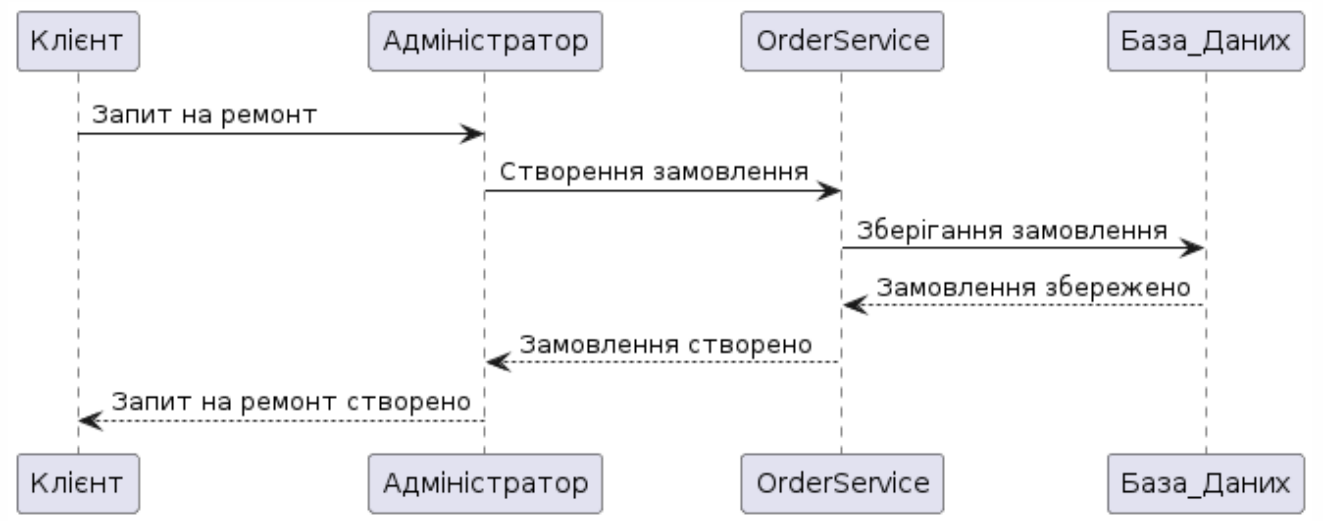


Рисунок 3.3 – Діаграма послідовності веб-застосунку (рисунок виконано самостійно)

3.1.4 Діаграма діяльності

Діаграма діяльності (Activity diagram) ілюструє процес обробки замовлення від прийому до завершення:

– прийом замовлення;
– призначення майстра;
– виконання ремонту;
– зміна статусу замовлення на "Виконане";
– формування гарантійного чеку;
– надсилання електронного листа клієнту;
– завершення обробки замовлення.

В результаті було побудовано наступну діаграму діяльності (див. рис. 3.4)

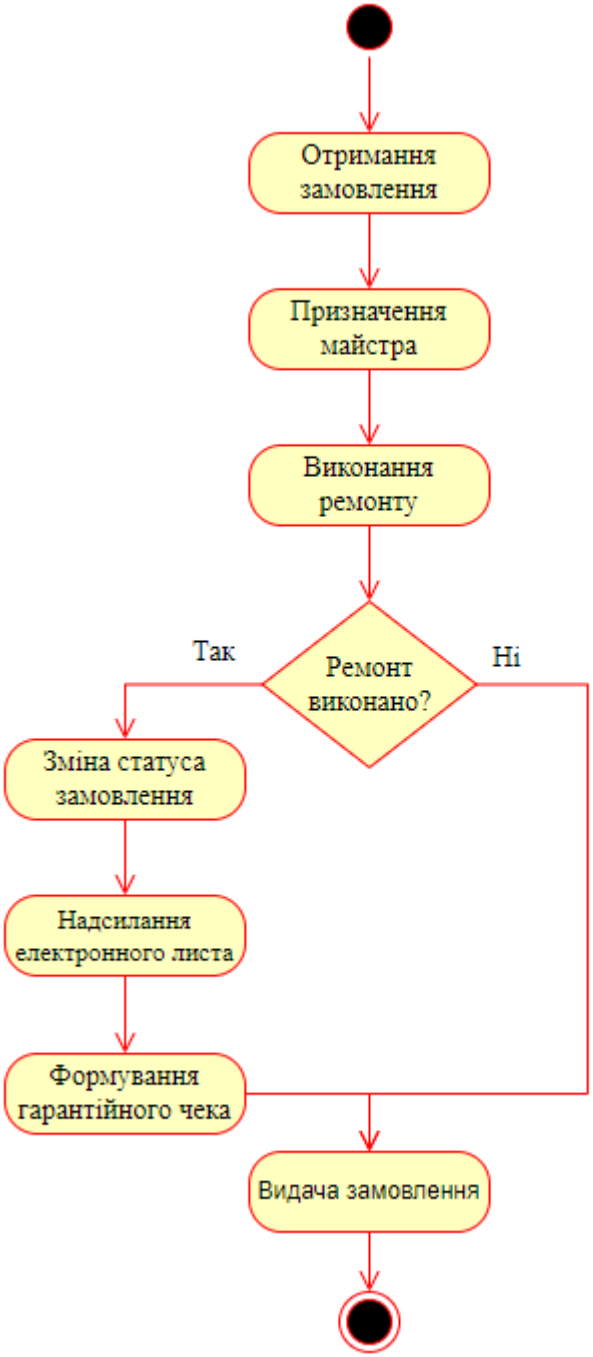


Рисунок 3.4 – Діаграма діяльності веб-застосунку (рисунок виконано самостійно)

3.2 Проєктування архітектури ПЗ

Проєктування архітектури включає визначення основних компонентів системи, їх взаємодію та технології, які будуть використовуватися. Веб-застосунок побудований на архітектурі "клієнт-сервер", де:

- клієнтська частина реалізована з використанням React, Bootstrap, React Router DOM та Axios;

- серверна частина реалізована з використанням Spring Boot та Spring JPA;
- база даних – реляційна база даних (наприклад, MySQL), яка зберігає інформацію про клієнтів, майстрів, запасні частини та замовлення.

Компоненти системи взаємодіють між собою за допомогою REST API.

Основні компоненти системи:

а) клієнтська частина (frontend):

1) інтерфейс користувача, створений за допомогою React та Bootstrap;
2) компоненти для відображення даних та взаємодії з користувачем;
3) маршрутизація для навігації між сторінками виконана за допомогою React Router DOM;
4) для виконання HTTP-запитів до серверної частини використовується Axios;

б) серверна частина (backend):

1) контролери для обробки HTTP-запитів;
2) сервіси для реалізації бізнес-логіки;
3) репозиторії для доступу до бази даних за допомогою Spring JPA;
4) моделі для представлення таблиць бази даних;

в) база даних:

1) таблиці для зберігання інформації про клієнтів, адреси, майстрів, запасні частини та замовлення;
2) запити SQL для взаємодії з базою даних.

3.3 Проєктування структури зберігання даних

Проєктування структури зберігання даних передбачає створення моделей даних та визначення їх зв'язків. У цій програмній системі використовуються наступні моделі:

а) clients – таблиця для зберігання інформації про клієнтів:

1) client_id – первинний ключ;
2) last_name – прізвище клієнта;
3) first_name – прізвище клієнта;

4) phone – телефон клієнта;
5) email – електронна пошта клієнта;
6) password – пароль клієнта;
7) role – роль клієнта в системі;

б) addresses – таблиця для зберігання інформації про адреси клієнтів:

1) address_id – первинний ключ;
2) city – місто;
3) street – вулиця;
4) building – номер будинку;
5) apartment – номер квартири;
6) client_id – зовнішній ключ, посилання на таблицю clients;

в) employees – таблиця для зберігання інформації про майстрів:

1) employee_id – первинний ключ;
2) last_name – прізвище майстра;
3) first_name – ім’я майстра;
4) phone – телефон майстра;
5) email – електронна пошта майстра;
6) password – пароль майстра;
7) role – роль майстра в системі;

г) orders –таблиця для зберігання інформації про замовлення:

1) order_id – первинний ключ;
2) date_created – дата створення замовлення;
3) price – вартість ремонту;
4) order_details – опис поломки;
5) status – статус замовлення;
6) model_id – зовнішній ключ, посилання на таблицю models;
7) client_id – зовнішній ключ, посилання на таблицю clients;

д) repair_works – таблиця для зберігання інформації про проведені ремонтні роботи:

1) repair_work_id – первинний ключ;

2) date_completed – дата завершення виконання ремонтної роботи;
3) date_started – дата початку виконання ремонтної роботи;
4) order_id – зовнішній ключ, посилання на таблицю orders;
5) employee_id – зовнішній ключ, посилання на таблицю employees;

е) spare_parts – таблиця для зберігання інформації про запасні частини:

1) spare_part_id – первинний ключ;
2) name – назва запасної частини;
3) stock – кількість на складі;
4) ordered_date – дата закупівлі;
5) supplier – постачальник;
6) repair_work_id – зовнішній ключ, посилання на таблицю repair_works;

ж) models – таблиця для зберігання інформації про моделі побутової техніки:

1) model_id – первинний ключ;
2) name – назва моделі.

Зв'язки між таблицями визначаються зовнішніми ключами. Наприклад, таблиця orders містить зовнішні ключі model_id та client_id, які посилаються на відповідні таблиці clients та models.

На основі отриманих даних була розроблена ER-діаграма за нотацією Баркера (див. рис. 3.5).

Рисунок 3.5 – ER-діаграма за нотацією Баркера (рисунок виконано самостійно)

Реляційна база даних – це база даних, яка використовується для зберігання та організації доступу до взаємопов’язаних елементів інформації. Реляційні бази даних ґрунтуються на реляційній моделі – інтуїтивно зрозумілому, наочному поданні інформації у вигляді таблиць. Кожен рядок у таблиці такої бази даних являє собою запис унікальним ідентифікатором – ключем. Стовпці таблиць мають атрибути даних, що дає змогу встановлювати зв’язки між елементами даних. Саме цьому, реляційна модель бази даних є найбільш ефективною для вирішення задачі даної кваліфікаційної роботи (див. рис. 3.6).

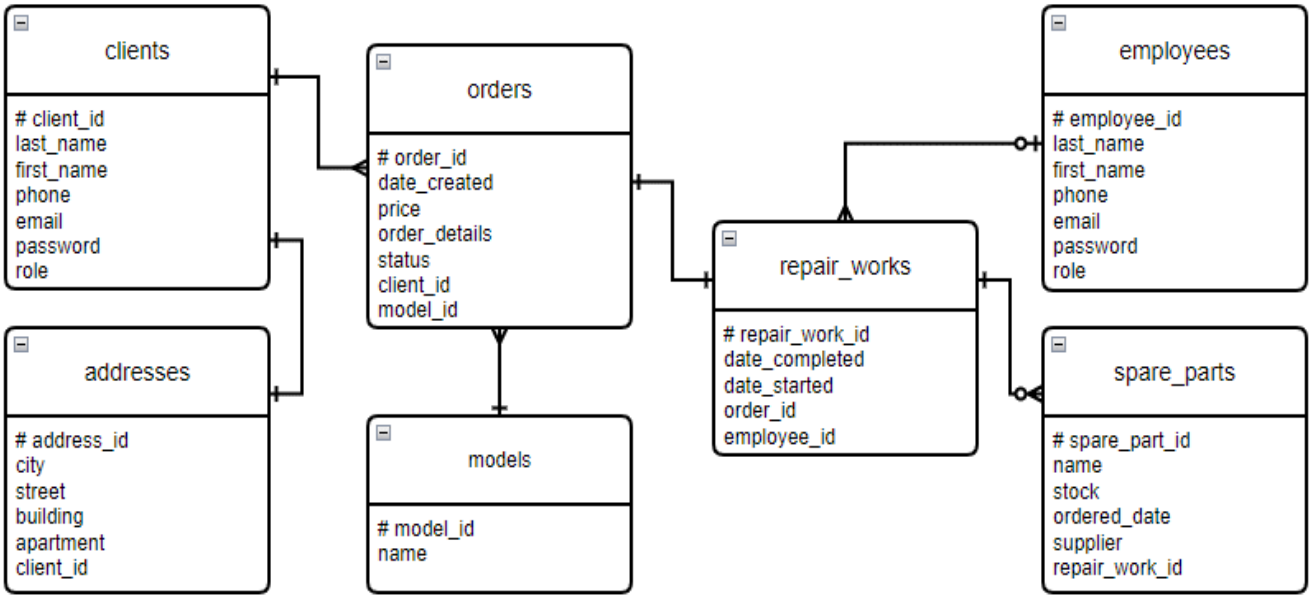


Рисунок 3.6 – Схема реляційної бази даних (рисунок виконано самостійно)

3.4 Створення UI / UX або іншого дизайну системи

Дизайн інтерфейсу користувача та взаємодії з користувачем є ключовими елементами успішного програмного забезпечення. У даній програмній системі дизайн реалізовано з використанням React та Bootstrap, що забезпечує сучасний та зручний інтерфейс.

Система складається з наступних сторінок:

а) головна сторінка:

1) відображає основну інформацію про замовлення, які знаходяться в роботі та надає навігацію до різних розділів системи;
2) використовує компоненти Bootstrap для створення адаптивного дизайну;

б) сторінка входу в систему:

1) містить форму для вводу електронної пошти та паролю користувача та кнопку входу;
2) надає можливості входу в систему для зареєстрованих користувачів

в) сторінка клієнтів:

1) містить таблицю з інформацією про клієнтів;
2) надає можливості сортування та пошуку клієнтів;

3) містить кнопки, що дозволяють перейти до сторінки редагування інформації про клієнта, створити замовлення для існуючого клієнта, видалити клієнта;

г) сторінка майстрів:

1) містить таблицю з інформацією про майстрів;
2) надає можливості сортування та пошуку майстрів;
3) містить кнопки, що дозволяють перейти до сторінки редагування інформації про майстра, видалити майстра;

д) сторінка запасних частин:

1) містить таблицю з інформацією про запасні частини;
2) надає можливості сортування та пошуку запасних частин;
3) містить кнопки, що дозволяють перейти до сторінки редагування інформації про запасну частину, видалити запасну частину;

е) сторінка замовлень:

1) містить таблицю з інформацією про замовлення;
2) надає можливості сортування, пошуку та фільтрації замовлень;
3) містить кнопку генерації гарантійного чеку для замовлення;
4) містить кнопки, що дозволяють перейти до сторінки редагування інформації про замовлення, видалення замовлення;

ж) сторінка додавання/редагування нових клієнтів:

1) включає форми для додавання та редагування клієнтів та їх адрес;

з) сторінка додавання/редагування нових майстрів:

1) включає форми для додавання та редагування майстрів;

и) сторінка додавання/редагування нових запчастин:

1) включає форми для додавання та редагування запасних частин

к) сторінка додавання/редагування нових замовлень:

1) включає форми для додавання та редагування замовлення, оновлення статусу замовлення;

л) сторінка статистики та звітів:

1) відображає різні статистичні дані.

Цей дизайн забезпечує зручність використання системи для всіх категорій користувачів, включаючи адміністраторів, майстрів та клієнтів. Застосування адаптивного дизайну гарантує коректне відображення інтерфейсу на різних пристроях, включаючи комп'ютери, планшети та смартфони.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Архітектурні рішення

Основою для розробки веб-застосунку було обрано архітектуру типу "клієнт-сервер". Серверна частина реалізована на базі Spring Boot, що дозволяє створювати високопродуктивні, масштабовані та гнучкі додатки. Клієнтська частина реалізована за допомогою React, що забезпечує динамічний інтерфейс користувача та зручний користувацький досвід.

Серверна частина відповідає за бізнес-логіку, управління базою даних та обробку запитів від клієнтської частини. Клієнтська частина взаємодіє із серверною через RESTful API, що забезпечує чітке розмежування функціональності та дозволяє розробляти клієнтську частину незалежно від серверної.

Серверна частина розроблена за допомогою мови програмування Java та фреймворку Spring Boot у середовищі IntelliJ IDEA. Вона відповідає за обробку запитів з клієнта, взаємодію з базою даних та надання необхідних даних клієнту. Бекенд буде містити API, який надасть доступ до функціональності додатку, такої як отримання списку замовлень, збереження даних клієнта або замовлення в базі даних, видалення клієнта тощо.

Клієнтська частина відповідає за представлення даних та взаємодію з користувачем. Фронтенд буде реалізований з використанням сучасних веб-технологій, таких як HTML, CSS, JavaScript, React, Bootstrap та розроблявся у середовищі Microsoft Visual Studio Code.

Для зберігання даних про клієнтів, замовлення та іншу відповідну інформацію була використана реляційна база даних MySQL. Структура бази даних буде відображати моделі клієнтів, їх адрес, замовлень та зв’язки між ними.

Високорівневу архітектурну діаграму можна побачити на рисунку 4.1

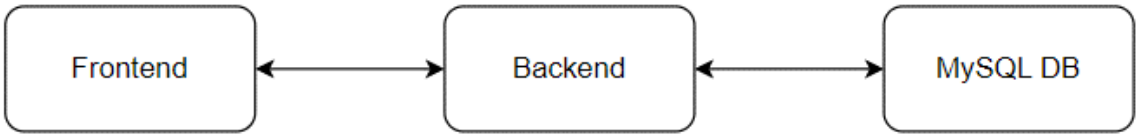


Рисунок 4.1 – Високорівнева архітектурна діаграма ПС (рисунок виконано самостійно)

Ця архітектура забезпечує розділення відповідальностей між клієнтською та серверною частинами додатку, дозволяє забезпечити більшу швидкість та ефективність роботи, а також покращену користувацьку взаємодію.

4.2 Використані технології

Компоненти системи реалізовано з використанням наступних технологій та бібліотек:

– Java – об’єктно-орієнтована мова програмування, що має велику популярність та використовується для різноманітних додатків, включаючи веб-застосунки. Java надає потужні засоби для роботи з рядками, масивами, колекціями, виключеннями та багатьма іншими структурами даних[6]. В цій роботі вона використовувалась для розробки серверної логіки додатку, такої як обробка запитів, взаємодія з базою даних та реалізація бізнес-логіки;

– Spring Boot – фреймворк для розробки Java-додатків, який спрощує процес налаштування та розгортання додатків. Він надає уніфікований підхід до конфігурації, який дозволяє швидко створювати готові до використання додатки з мінімальними зусиллями. Також він забезпечує вбудований контейнер сервлетів, що дозволяє легко розгортати додаток на сервері[7];

– MySQL – це СУБД, яка використовує мову SQL для зберігання та управління даними. Вона пропонує широкі можливості для створення, зчитування, оновлення та видалення даних з бази даних. MySQL є популярним вибором для зберігання даних в багатьох веб-застосунках завдяки своїй надійності, швидкодії та легкості використання[8];

– Spring Data JPA – це модуль Spring, який надає спрощений спосіб взаємодії з базами даних за допомогою JPA. Він дозволяє використовувати анотації для опису моделей даних і автоматично генерує SQL-запити на основі цих анотацій. Цей модуль спрощує роботу з базами даних, таку як збереження, оновлення, видалення та запити до бази даних;

– JavaScript - мова програмування, яка використовується для розробки клієнтської частини веб-застосунків. Вона надає можливість динамічно змінювати та взаємодіяти з елементами сторінки, обробляти події, виконувати запити до сервера та багато іншого. JavaScript є основною мовою програмування для розробки інтерактивних функцій на стороні клієнта;

– React – це бібліотека JavaScript для розробки користувацького інтерфейсу. Вона дозволяє створювати ефективні та перевикористовувані компоненти, які реагують на зміни даних та автоматично оновлюються на сторінці. React використовує віртуальний DOM для оптимізації процесу оновлення сторінки, що забезпечує високу продуктивність додатку[9];

– Bootstrap – це фреймворк CSS, який містить набір стилів та компонентів для швидкого та простого оформлення веб-сторінок. Він надає різні готові компоненти, такі як кнопки, форми, таблиці, навігаційні панелі тощо, що допомагає забезпечити єдність в дизайні вашого додатку[10];

– React Router DOM – це набір розширень для маршрутизації в React-додатках. Він дозволяє визначати шляхи (роути) та пов'язані з ними компоненти, щоб забезпечити навігацію між різними сторінками вашого додатку[11];

– Axios – це бібліотека JavaScript для здійснення HTTP-запитів з клієнтської сторони. Вона надає простий та зручний інтерфейс для виконання різних видів запитів, таких як GET, POST, PUT, DELETE тощо. Axios дозволяє взаємодіяти з сервером, обмінюватися даними та отримувати відповіді[12];

– JWT (JSON Web Token) — це стандарт відкритого формату для створення токенів доступу, що можуть бути використані для обміну інформацією між сторонами у безпечний спосіб. JWT широко використовується для аутентифікації та авторизації користувачів у веб-застосунках[13]. Використання JWT забезпечує

безпечну та ефективну аутентифікацію та авторизацію користувачів у нашому веб-застосунку. Це дозволяє розмежувати доступ до різних частин системи для клієнтів, майстрів та адміністраторів, підвищуючи рівень безпеки та контролю за даними. JWT токени надають можливість зберігати стан авторизації на клієнтській стороні, що спрощує взаємодію між клієнтом та сервером і забезпечує більш масштабовану та продуктивну систему.

Комбінування цих інструментів дозволяє створити потужну, ефективну та масштабовану систему зі зручним користувацьким інтерфейсом.

4.3 Опис програмної реалізації

Для початку роботи з системою користувач має увійти з використанням комбінації електронної пошти та паролю. Робота з системою для неавторизованих користувачів заборонена, при спробі відкриття будь-якої сторінки користувач буде перенаправлений на сторінку входу. Сторінка входу має валідацію введених даних і перевірку використання неіснуючих даних користувача. Сторінку входу зображено на рисунку 4.2.

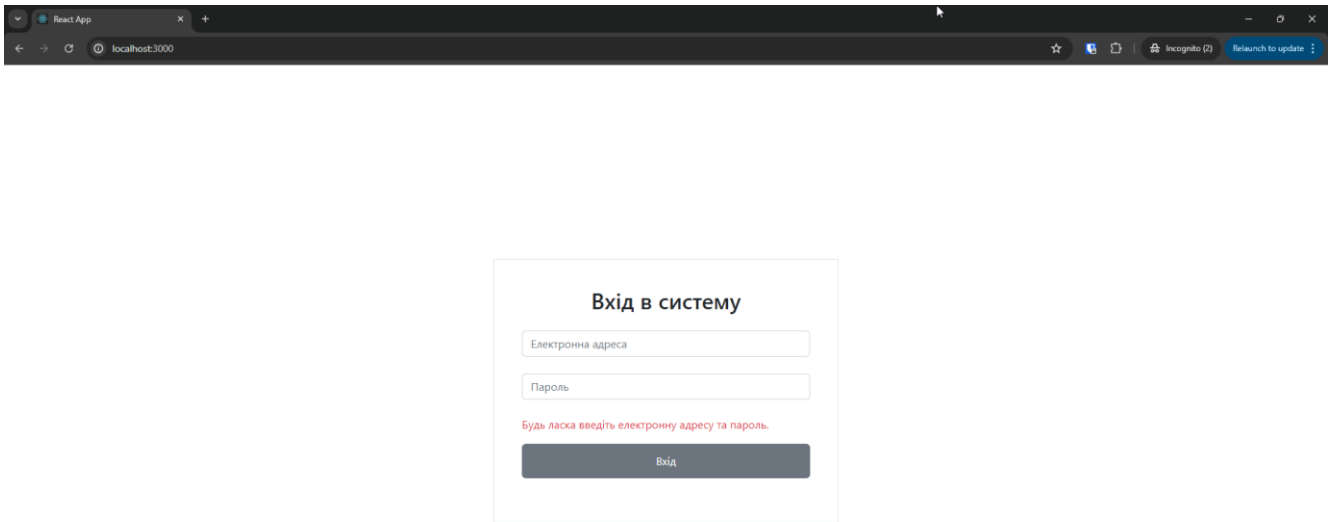


Рисунок 4.2 – Сторінка входу в систему з валідацією введених даних (рисунок виконано самостійно)

Додавання нових клієнтів, їх адреси, замовлень, моделей техніки, майстрів, запасних частин, ремонтних робіт відбувається на відповідних сторінках, потрапити на які можна з навігаційного меню у хедері веб-застосунку. Приклад посилання на сторінку створення клієнта у навігаційному меню можна побачити на рисунку 4.3.

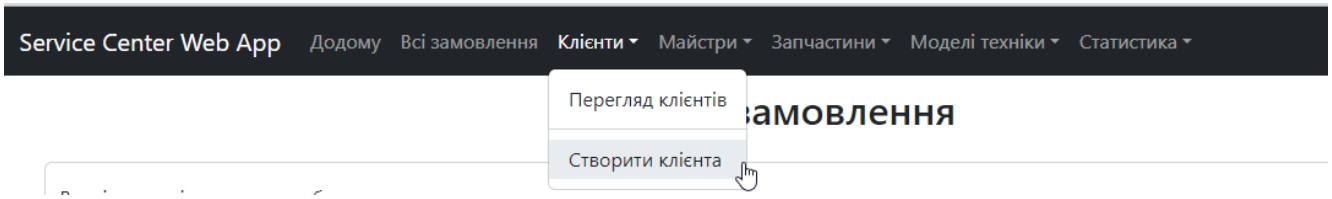


Рисунок 4.3 – Навігаційне меню, з посиланням на створення клієнта (рисунок виконано самостійно)

Після натискання на посилання відкривається сторінка створення нового клієнта та адреси клієнта. Форма створення клієнта/адреси має фронтенд валідацію для запобігання створення сутностей з порожніми полями. Валідацію представлено на рисунку 4.4.

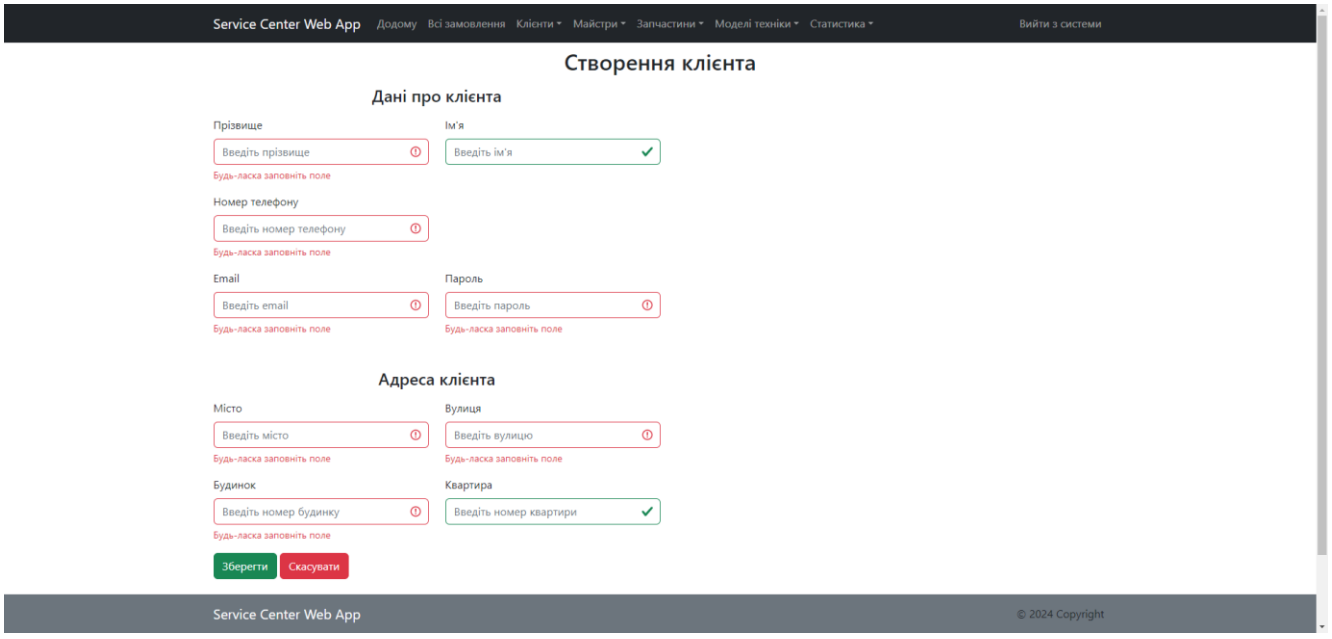


Рисунок 4.4 – Сторінка створення клієнта/адреси після спроби підтвердити форму з порожніми полями (рисунок виконано самостійно)

Після заповнення форми та натискання кнопки «Зберегти» відправляється запит на серверну частину, яка відповідає за комунікацію з базою даних і створює

новий запис у БД. При натисканні кнопки «Скасувати» всі введені дані ігноруються, новий запис не створюється та користувач перенаправляється на сторінку перегляду всіх клієнтів.

Функція перегляду інформації виконана у вигляді сторінок з таблицями, що містять інформацію про сутності з БД. Приклад сторінки перегляду клієнтів наведено на рисунку 4.5.

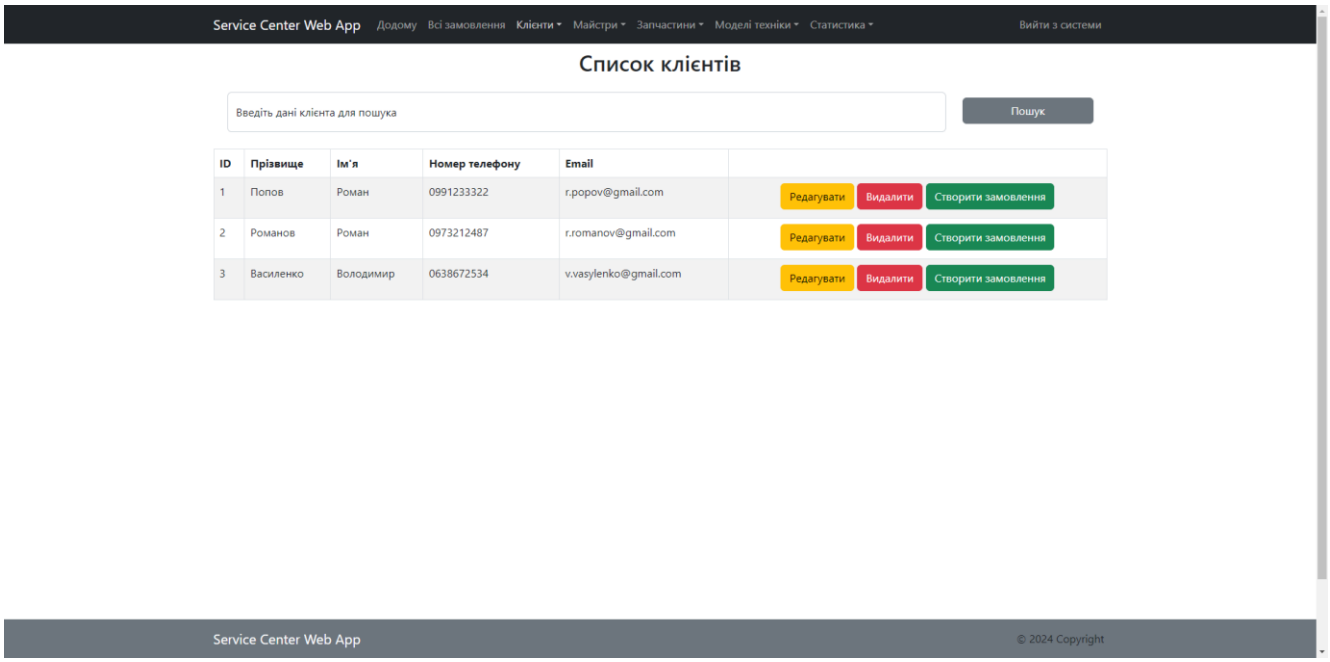


Рисунок 4.5 – Сторінка перегляду клієнтів (рисунок виконано самостійно)

На цій сторінці користувач може переглядати інформацію про всіх збережених клієнтів, для кожного клієнта існує можливість перейти до сторінки редагування їх інформації, видалити запис з БД або створити замовлення для цього клієнта. Також на сторінках перегляду інформації можуть бути присутні поля пошуку, фільтри та можливість сортування.

Функція редагування підтримується для всіх сутностей даної інформаційної системи. При відкритті сторінки редагування поля форми автоматично заповнюються даними про сутність, які зараз збережені у БД. Форма має фронтенд валідацію для уникнення помилок при оновленні сутності з порожніми полями. Приклад коду для оновлення замовлення наведено на рисунках 4.6 та 4.7.

```
@PutMapping("/orders/{id}")
public ResponseEntity<Order> updateOrder(@PathVariable int id, @RequestBody Order orderRequest) {
    Order order = service.updateOrder(id, orderRequest);
    if (order.getOrderStatus().equals(OrderStatus.DONE)) {
        emailService.sendOrderConfirmationEmail(order);
    }
    return new ResponseEntity<>(order, HttpStatus.OK);
}
```

Рисунок 4.6 – Код контролеру оновлення замовлень (рисунок виконано самостійно)

```
@Override
public Order updateOrder(int id, Order order) {
    Order existingOrder = repository.findById(id).orElseThrow(() -> new ResourceNotFoundException(format("Order with id %d not found", id)));
    existingOrder.setDateCreated(order.getDateCreated());
    existingOrder.setPrice(order.getPrice());
    existingOrder.setOrderDetails(order.getOrderDetails());
    existingOrder.setOrderStatus(order.getOrderStatus());
    existingOrder.setModel(order.getModel());
    return repository.save(existingOrder);
}
```

Рисунок 4.7 – Код, для оновлення замовлення на серверній частині (рисунок виконано самостійно)

Після натискання кнопки «Скасувати» користувач повертається на сторінку «Всі замовлення». Після натискання кнопки «Зберегти» оновлені дані відправляються до серверної частини, для подальшого оновлення даних у БД. Також на цій сторінці майстер може розпочати та завершити роботу над замовленням. Приклад сторінки редагування нового замовлення наведено на рисунку 4.8.

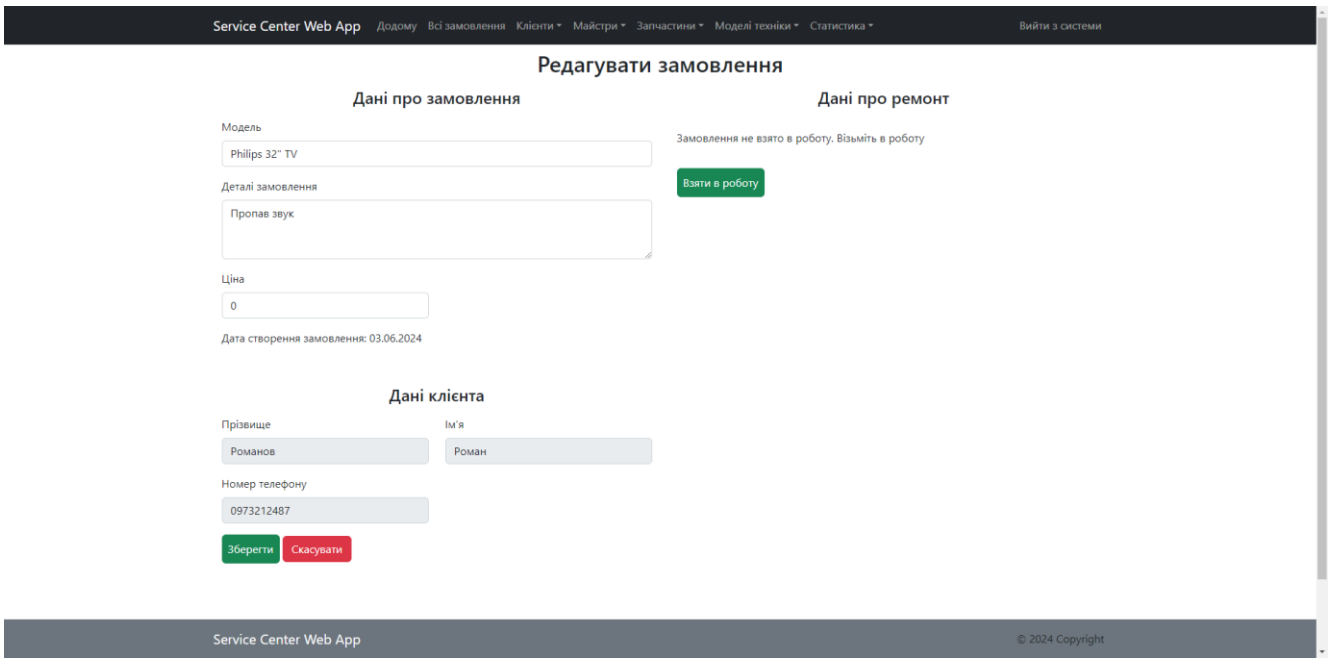


Рисунок 4.8 – Сторінка редагування нового замовлення (рисунок виконано самостійно)

На рисунку 4.9 наведено приклад сторінки редагування замовлення зі статусом «В роботі». Для таких замовлень майстер може додати використані запчастини і вказати їх кількість.

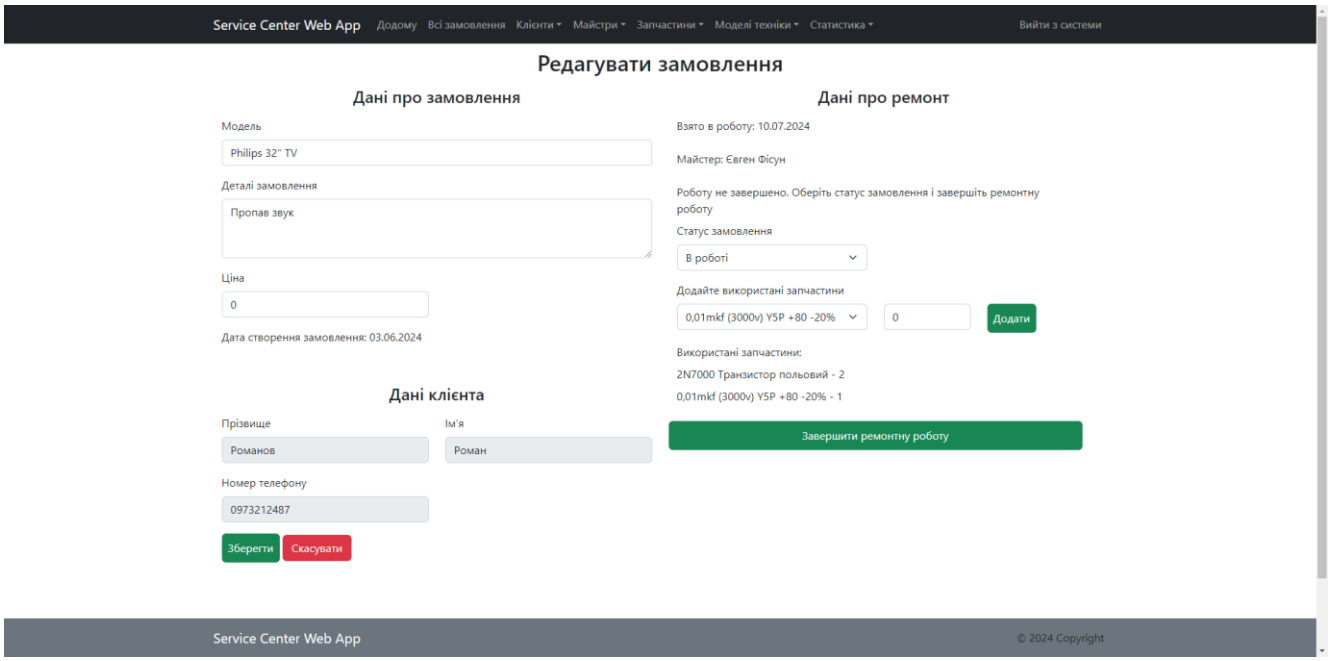


Рисунок 4.9 – Сторінка редагування замовлення зі статусом «В роботі» (рисунок виконано самостійно)

Видалення сутностей з БД відбувається після натискання відповідної кнопки на сторінці перегляду списку обраних сутностей. Після натискання кнопки «Видалити» з’являється підтвердження даної операції, для уникнення помилкових видалень сутностей. Приклад сторінки перегляду запасних частин з кнопками видалення наведено на рисунку 4.10.

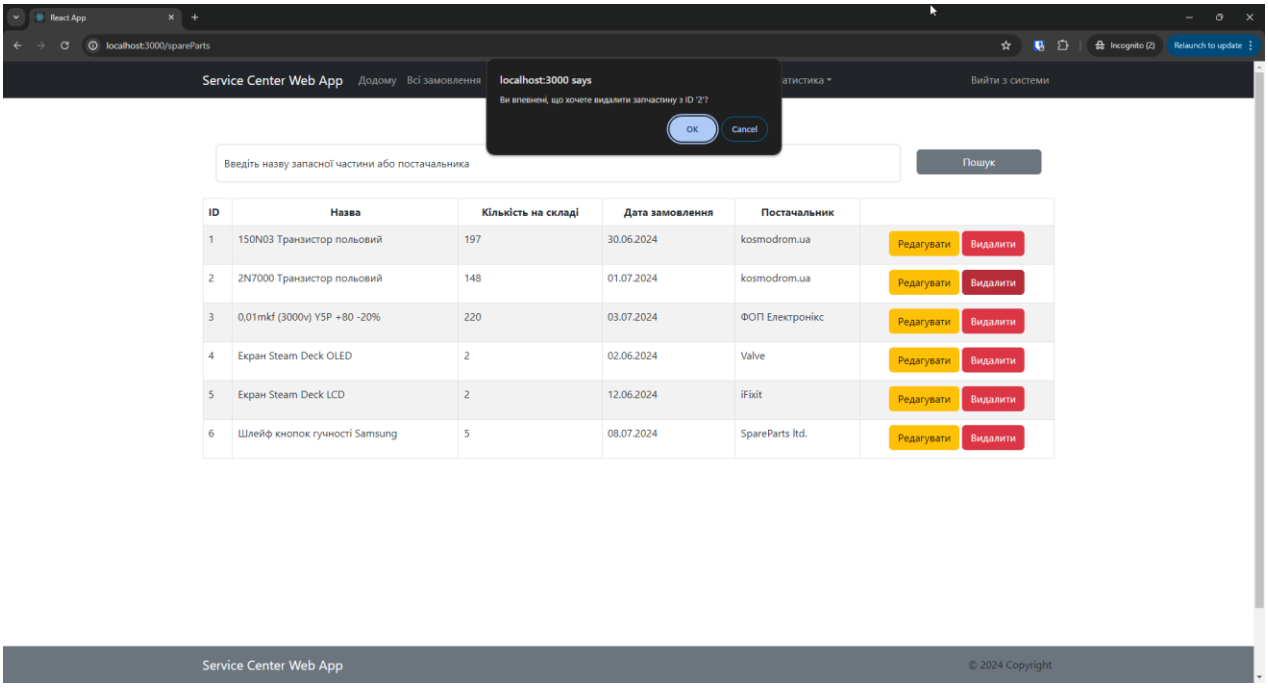


Рисунок 4.10 – Сторінка перегляду всіх запасних частин з кнопкою «Видалити» (рисунок виконано самостійно)

Функції пошуку інформації, сортування та фільтрації реалізовані на сторінках перегляду інформації про всі сутності таблиці. Для сторінки «Всі замовлення» користувач може виконувати пошук по деталям замовлення або назві моделі. Відображення інформації відбувається одразу після введення пошукового запиту. Результати пошуку відображено на рисунку 4.11

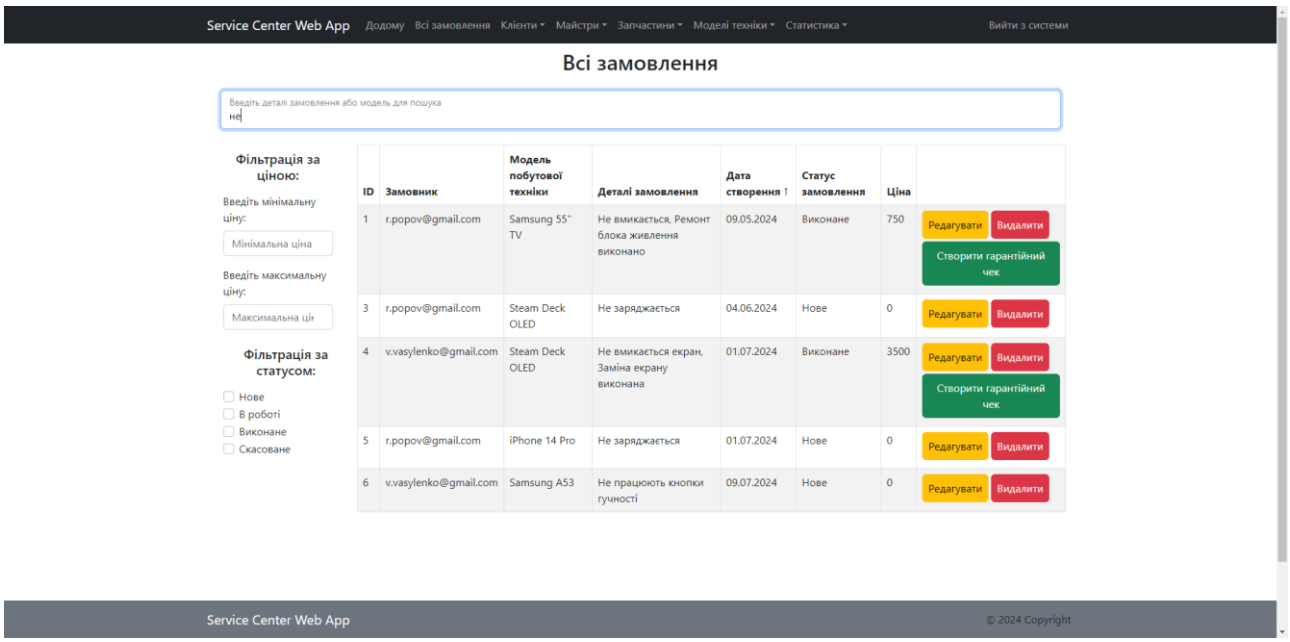


Рисунок 4.11 – Пошук за назвою моделі або деталями замовлення (рисунок виконано самостійно)

Фільтрація замовлень відбувається за ціною та/або статусом замовлення, враховуючи пошуковий запит. Приклад фільтрації за ціною наведено на рисунку 4.12.

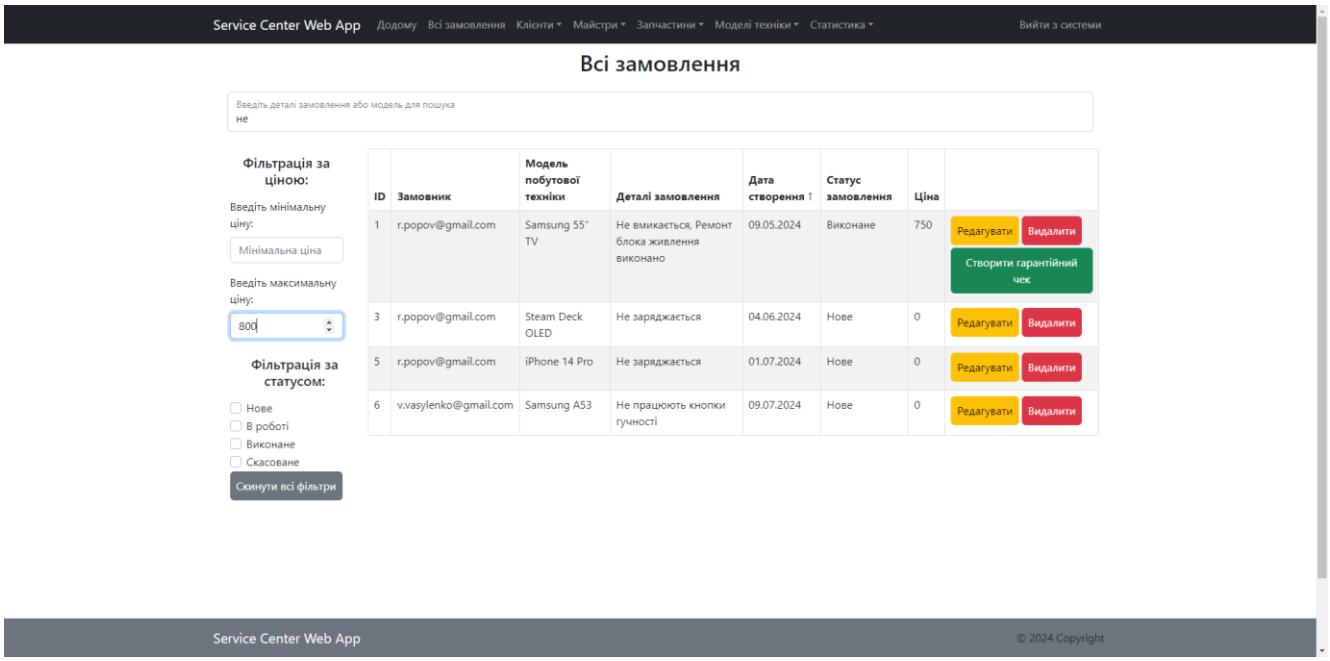


Рисунок 4.12 – Фільтрація за ціною (рисунок виконано самостійно)

Після застосування будь-якого фільтру з’являється кнопка, яка дозволяє в одну дію скинути всі застосовані фільтри. Фільтрація за статусом замовлення

також працює у поєднанні з іншим фільтром та/або пошуковим запитом. Сортування результатів цієї таблиці може бути виконане за допомогою натискання на голову колонки «Дата створення» або «Ціна». Сортування працює у поєднанні з фільтрацією та пошуком. Приклад наведено на рисунку 4.13

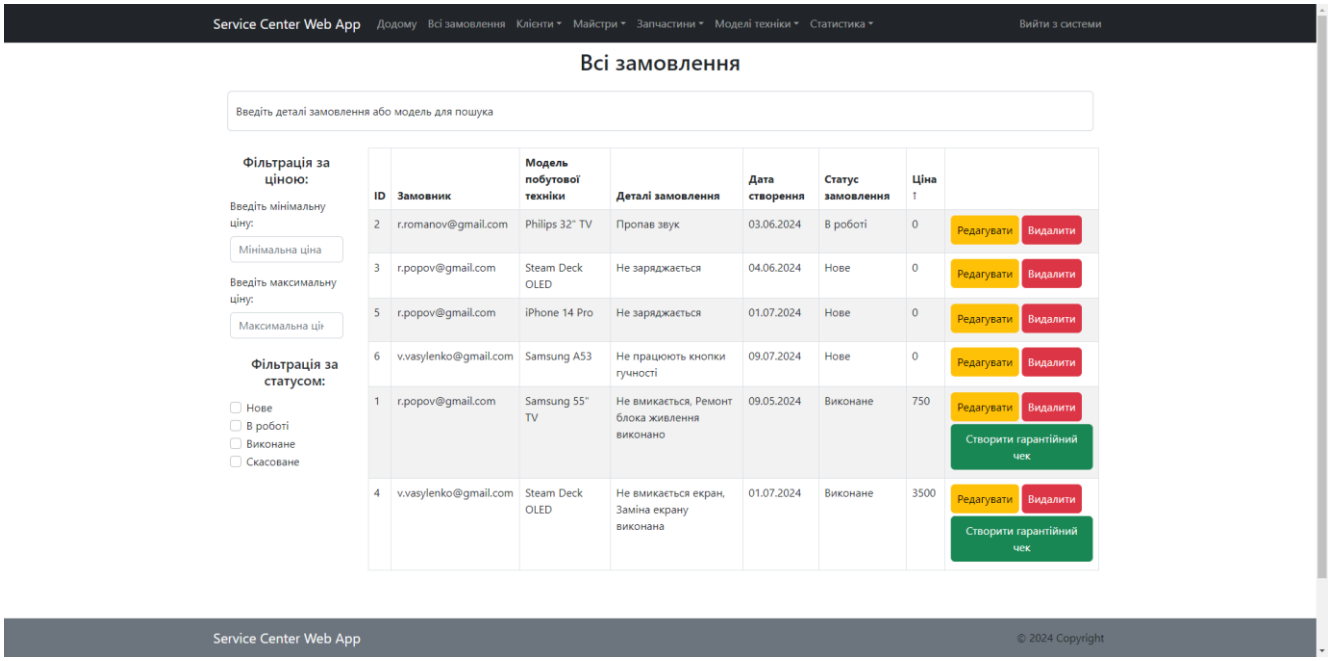


Рисунок 4.13 – Приклад фільтрації за статусом, пошуком та сортуванням за зростанням ціни (рисунок виконано самостійно)

Статистики представлені у окремому меню навігаційної панелі. Статистику «Список клієнтів, що найчастіше зверталися» наведено на рисунку 4.14.

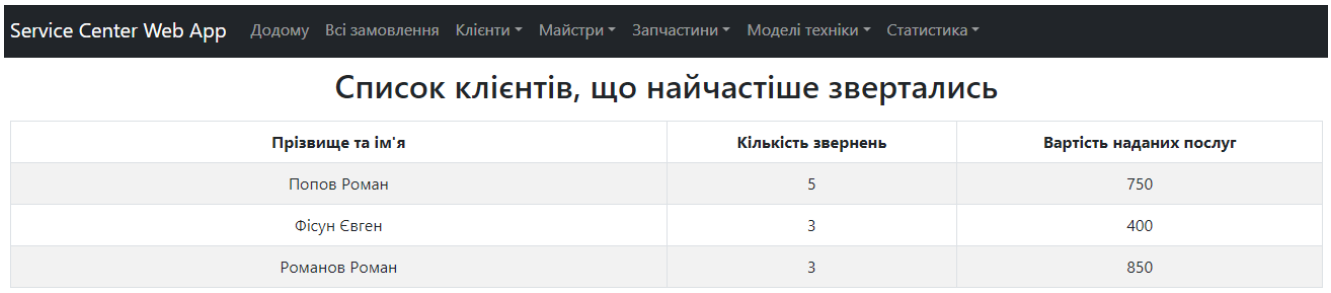


Рисунок 4.14 – Топ-3 клієнтів за кількістю звернень (рисунок виконано самостійно)

Приклад SQL-запиту для отримання цієї статистики наведено на рисунку 4.15.

```
@Query(value = """
        SELECT
        c.client_id AS clientId,
        CONCAT(c.last_name, ' ', c.first_name) AS clientName,
        COUNT(o.order_id) AS numberOfOrders,
        COALESCE(SUM(o.price), 0) AS totalServicesCost
        FROM clients c
        LEFT JOIN orders o ON c.client_id = o.client_id
        GROUP BY c.client_id, clientName
        ORDER BY numberOfOrders DESC
        LIMIT 3""", nativeQuery = true)
List<Object[]> getClientStatisticsRaw();
```

Рисунок 4.15 – SQL-запит для отримання статистики «Топ-3 клієнтів за кількістю звернень» (рисунок виконано самостійно)

Приклад сторінки статистики «Топ-5 моделей з якими реєструються звернення» наведено на рисунку 4.16.

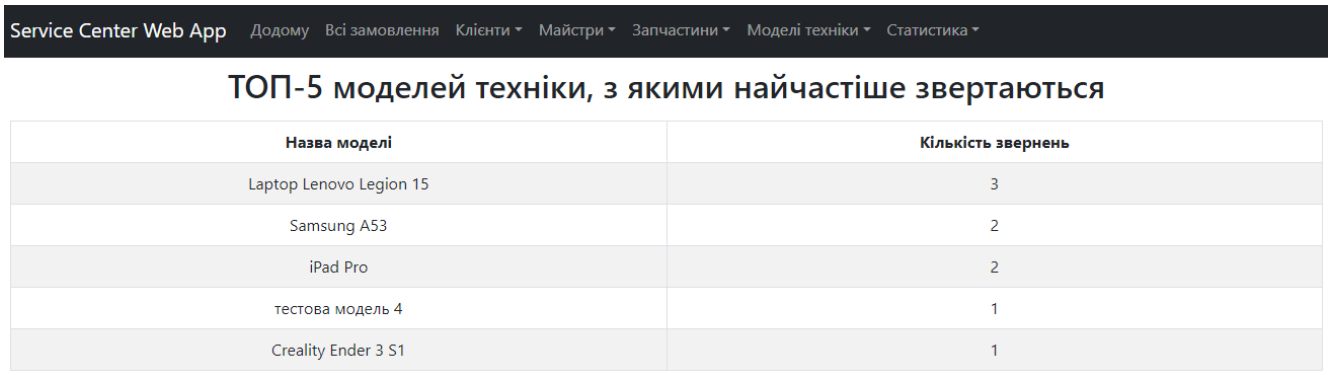


Рисунок 4.16 - Сторінка статистики «Топ-5 моделей з якими реєструються звернення» (рисунок виконано самостійно)

Приклад SQL-запиту для отримання цієї статистики наведено на рисунку 4.17.

```
@Query(value = """
        SELECT
        m.model_id as modelId,
        m.name AS modelName,
        COUNT(o.order_id) AS numberOfOrders
        FROM models m
        JOIN orders o ON m.model_id = o.model_id
        GROUP BY m.model_id, modelName
        ORDER BY numberOfOrders DESC
        LIMIT 5""", nativeQuery = true)
List<Object[]> getModelStatisticsRaw();
```

Рисунок 4.17 - SQL-запит для отримання статистики «Топ-5 моделей з якими реєструються звернення» (рисунок виконано самостійно)

Приклад сторінки статистики «Кількість звернень клієнтів за кожен місяць поточного року» наведено на рисунку 4.18.

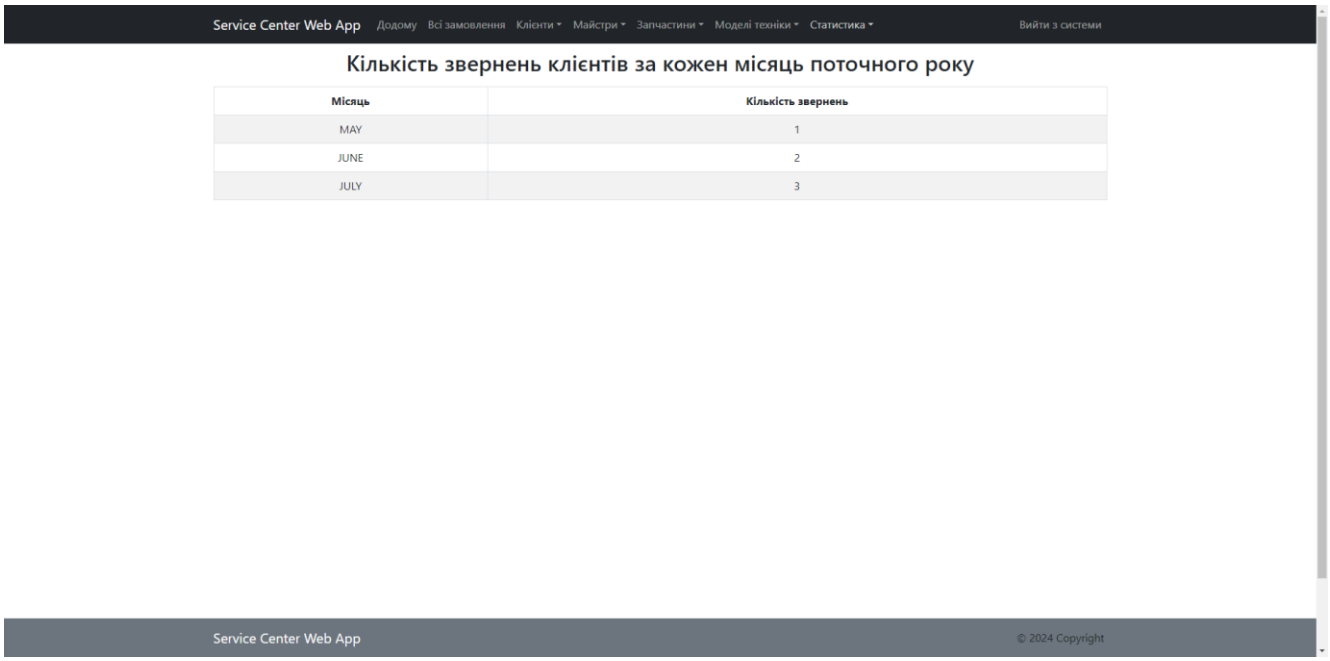


Рисунок 4.18 - Сторінка статистики «Кількість звернень клієнтів за кожен місяць поточного року» (рисунок виконано самостійно)

Приклад SQL-запиту для отримання цієї статистики наведено на рисунку 4.19.

```
@Query(value = """
        SELECT
        MONTH(date_created) AS month,
        COUNT(*) AS requests_count
        FROM orders
        WHERE YEAR(date_created) = YEAR(CURDATE())
        GROUP BY MONTH(date_created)
        """, nativeQuery = true)
List<Object[]> getClientRequestsCountByMonth();
```

Рисунок 4.19 - SQL-запит для отримання статистики «Кількість звернень клієнтів за кожен місяць поточного року» (рисунок виконано самостійно)

Після зміни статусу замовлення на «Виконане» для даного замовлення з’являється кнопка «Створити гарантійний чек» після натискання на яку користувачу на ПК завантажується згенерований PDF файл звіту, зі всіма даними замовлення. Приклад згенерованого чеку наведено на рисунку 4.20.

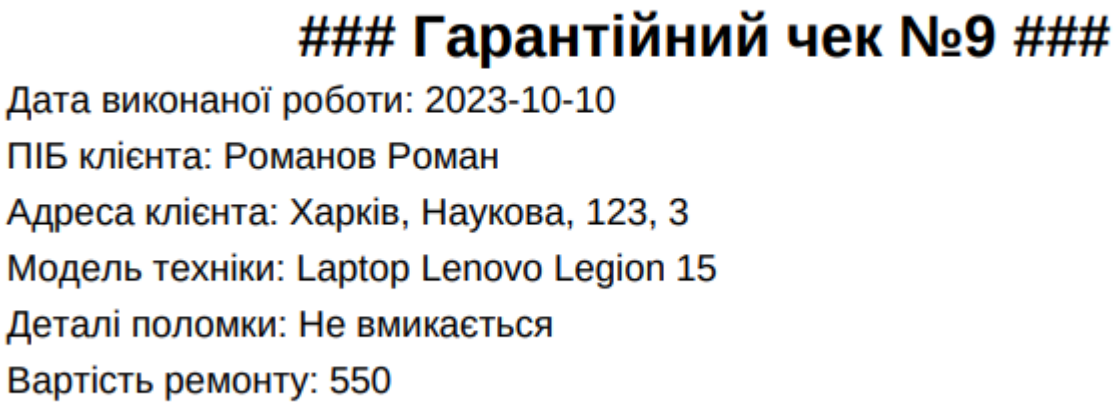


Рисунок 4.20 – Згенерований звіт «Гарантійний чек» (рисунок виконано самостійно)

4.4 Опис задачі автоматизації

Після того як користувач змінює статус замовлення на «Виконане» на серверній частині запускається процес генерації та надсилання електронного листа

з підтвердженням виконання замовлення клієнту. Приклад електронного листа згенерованого після виконання ремонтної роботи наведено на рисунку 4.21.

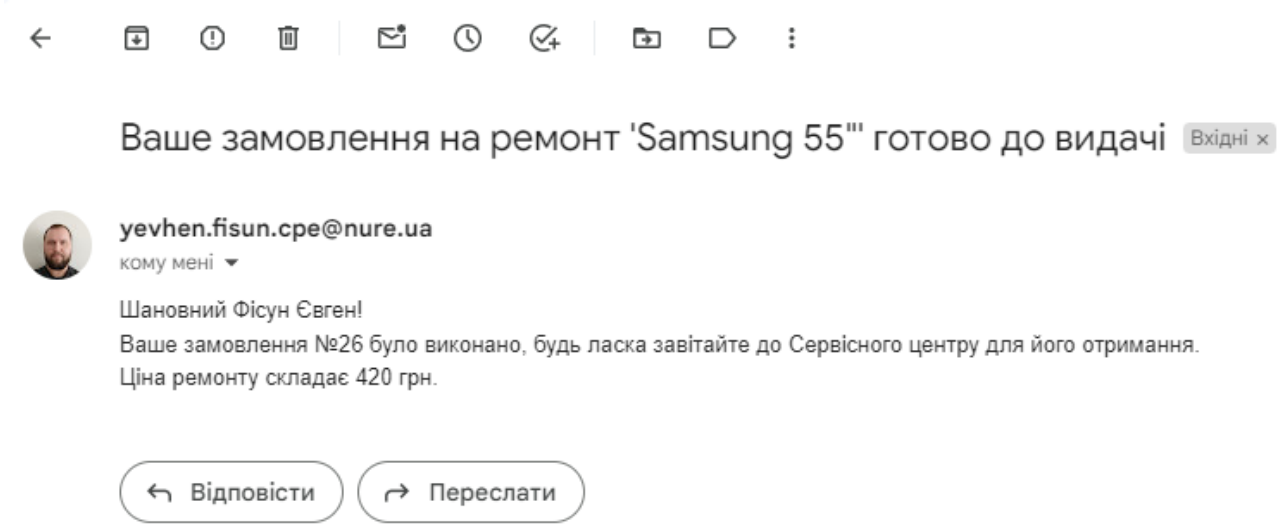


Рисунок 4.21 – Приклад електронного листа про завершення ремонту (рисунок виконано самостійно)

Код сервісу генерації та надсилання електронних листів наведено на рисунку 4.22.

```java
@Service
public class EmailServiceImpl implements EmailService {

    private static final String ORDER_READY_SUBJECT = "Ваше замовлення на ремонт '%s' готово до видачі";

    @Autowired
    private JavaMailSender mailSender;

    @Override
    public void sendEmail(String to, String subject, String body) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setTo(to);
        message.setSubject(subject);
        message.setText(body);

        mailSender.send(message);
    }

    @Override
    public void sendOrderConfirmationEmail(Order order) {
        sendEmail(order.getClient().getEmail(), getEmailSubject(order), getEmailBody(order));
    }

    private String getEmailSubject(Order order) {
        return String.format(ORDER_READY_SUBJECT, order.getModel().getName());
    }

    private String getEmailBody(Order order) {
        StringBuilder sb = new StringBuilder();
        sb.append("Шановний ")
                .append(order.getClient().getLastName())
                .append(StringUtils.SPACE)
                .append(order.getClient().getFirstName())
                .append("!\n")
                .append("Ваше замовлення №")
                .append(order.getId())
                .append(" було виконано, будь ласка завітайте до Сервісного центру для його отримання.")
                .append("\n")
                .append("Ціна ремонту складає ")
                .append(order.getPrice())
                .append(" грн.");
        return sb.toString();
    }
}
```

Рисунок 4.22 - Код сервісу генерації та надсилання електронних листів (рисунок виконано самостійно)

4.5 Виклик і завантаження програмної системи

Для початку роботи з веб-застосунком потрібно встановити середовище розробки. Для цього необхідно встановити наступне програмне забезпечення:

– Java Development Kit – завантажується з офіційного сайту;

– Node.js – завантажується з офіційного сайту.

Після копіювання вихідного коду додатку на сервер потрібно встановити залежності клієнтської частини за допомогою команди

npm install або *npm i*

Після цього необхідно перейти у теку з серверною частиною, та у файлі *src/main/resources/application.properties* вказати посилання на базу даних MySQL у властивості *spring.datasource.url,* логін та пароль для підключення до бази даних у відповідних властивостях, вказаних на рисунку 4.23.

```
spring.datasource.url=jdbc:mysql://localhost:3306/service_center?useSSL=false
spring.datasource.username=root
spring.datasource.password=
```

Рисунок 4.23 – Рядки конфігурації для підключення до бази даних (рисунок виконано самостійно)

Після цього можна запустити серверну частину за допомогою команди з консолі.

mvnw spring-boot:run

У випадку, якщо все налаштовано вірно ви побачите повідомлення про успішний старт серверної частини додатку, відображене на рисунку 4.24.

```
LiveReload server is running on port 35729
Tomcat started on port(s): 8080 (http) with context path '/api'
Started SpringbootBackendApplication in 4.805 seconds (process running for 5.259)
```

Рисунок 4.24 – Повідомлення про успішний старт серверної частини веб-застосунку (рисунок виконано самостійно)

Для запуску клієнтської частини необхідно виконати команду

npm start

У випадку, якщо все налаштовано вірно система стане доступною через браузер за посиланням *http://localhost:3000*

Консоль має містити повідомлення представлене на рисунку 4.25.

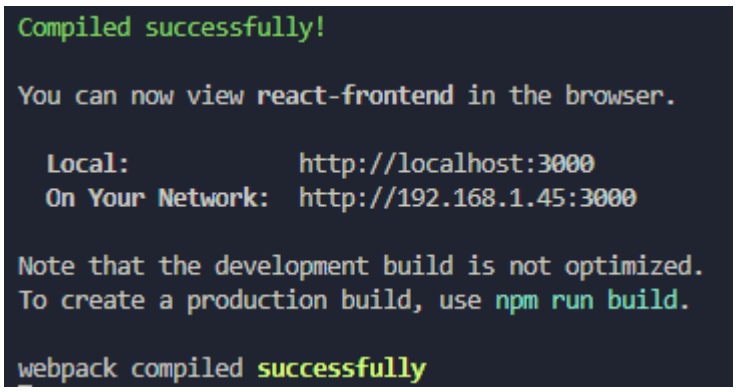


Рисунок 4.25 – Повідомлення про успішний старт клієнтської частини веб-застосунку (рисунок виконано самостійно)

Після того як обидві частини було успішно запущено, користувач може приступати до роботи з веб-застосунком використовуючи будь-який браузер для відкриття вказаної в консолі адреси клієнтської частини.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення (ПЗ) є важливим етапом розробки, який забезпечує виявлення та усунення дефектів, перевірку відповідності системи вимогам, а також забезпечення якості та надійності продукту. Тестування веб-застосунку для управління та обліку роботи сервісного центру з ремонту побутової техніки включало різні підходи та методи.

5.1 Підходи до тестування

У процесі тестування даного веб-застосунку використовувалися наступні підходи:

– модульне тестування - цей підхід передбачає тестування окремих модулів або компонентів системи. Для цього використовувалися фреймворки JUnit для Java;

– інтеграційне тестування - перевірка взаємодії між різними модулями та компонентами. Використовувалися Spring Boot Test для серверної частини та React Testing Library для клієнтської частини;

– функціональне тестування - перевірка функціональності системи відповідно до вимог. Функціональні тести включали тестування основних бізнес-процесів, таких як реєстрація користувачів, вхід в систему, робота з різними ролями користувачів, створення та обробка замовлень, управління запасними частинами тощо;

– регресійне тестування – перевірка системи після внесення змін для виявлення можливих дефектів, що могли виникнути внаслідок змін у коді;

– користувацьке тестування (User Acceptance Testing) – перевірка системи реальними користувачами з метою оцінки її зручності та відповідності вимогам.

5.2 Модульне тестування

Модульне тестування зосереджувалося на перевірці окремих класів та функцій. Веб-застосунок включає різні модулі, такі як аутентифікація, управління

користувачами, управління замовленнями та запасними частинами. Наприклад, для перевірки функції аутентифікації використовувалися тести, що імітують логін користувача та перевіряють правильність створення JWT токена.

5.3 Інтеграційне тестування

Інтеграційне тестування проводилося для перевірки взаємодії між різними компонентами системи, такими як взаємодія між контролерами, сервісами та репозиторіями. Це забезпечувало перевірку того, що всі частини системи коректно працюють разом.

5.4 Функціональне тестування

Функціональне тестування забезпечує перевірку того, що всі вимоги до системи виконуються. Тести включали сценарії для перевірки реєстрації користувачів, входу в систему, створення та редагування замовлень, управління запасними частинами тощо. Наприклад, тест на функціональність реєстрації користувача перевіряв, що всі необхідні поля заповнені правильно, і що користувач успішно створюється в системі. Приклади тест кейса наведено у таблиці 5.1.

Таблиця 5.1 – Тест-кейс №1 (таблиця виконана самостійно)

Інформація про тест-кейс		
Ідентифікатор тест-кейсу	TC01 ver1.0	
Власник тест-кейсу	Фісун Євген Олександрович	
Дата створення	31.05.2024	
Мета тест-кейсу	Перевірити, що новий клієнт може бути створений при введенні коректних даних клієнта.	
Методика тестування		
Налаштування прогону	Користувач знаходиться на сторінці створення нового клієнту	(V)

Кінець таблиці 5.1

Крок	Дія	Очікуваний результат	Відмітка (V)*
1	Ввести прізвище у поле Прізвище	Введені дані відображаються у полі Прізвище	(V)
2	Ввести ім’я у поле Ім’я	Введені дані відображаються у полі Ім’я	(V)
3	Ввести номер телефону у поле Телефон	Введені дані відображаються у полі Телефон	(V)
4	Ввести валідну електронну пошту у полі Email	Введені дані відображаються у полі Email	(V)
5	Ввести пароль у полі Пароль	Введений пароль відображається у прихованому вигляді	(V)
6	Ввести місто у поле Місто	Введені дані відображаються у полі Місто	(V)
7	Ввести вулицю у поле Вулиця	Введені дані відображаються у полі Вулиця	(V)
8	Ввести номер будинку у поле Будинок	Введені дані відображаються у полі Будинок	(V)
9	Залишити поле Квартира порожнім	Поле порожнє	(V)
10	Натиснути кнопку Зберегти	Користувач перенаправляється на сторінку Всі клієнти, новий клієнт з введеними даними відображається у списку клієнтів	(V)
Результати тестування			
Тестувальник: Фісун Є.О.		Статус прогону тесту (Passed/Failed/Blocked)	P

Тест на перевірку, що новий клієнт не створюється після натискання кнопки «Скасувати» на сторінці створення клієнта наведено у таблиці 5.2.

Таблиця 5.2 – Тест-кейс №2 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тест-кейсу	TC02 ver1.0		
Власник тест кейсу	Фісун Євген Олександрович		
Дата створення	31.05.2024		
Мета тест-кейсу	Перевірити, що новий клієнт не створюється після скасування створення.		
Методика тестування			
Налаштування прогону тесту	Користувач знаходиться на сторінці створення нового клієнту		(V)
Крок	Дія	Очікуваний результат	Відмітка (V)*
1	Ввести прізвище у поле Прізвище	Введені дані відображаються у полі Прізвище	(V)
2	Ввести ім’я у поле Ім’я	Введені дані відображаються у полі Ім’я	(V)
3	Ввести номер телефону у поле Телефон	Введені дані відображаються у полі Телефон	(V)
4	Натиснути кнопку Скасувати	Користувач перенаправляється на сторінку Всі клієнти, новий клієнт з введеними даними не відображається у списку клієнтів	(V)
Результати тестування:			
Тестувальник: Фісун Є.О.		Статус прогону тесту (Passed/Failed/Blocked)	P

5.5 Регресійне тестування

Регресійне тестування проводилося після кожного внесення змін до коду для перевірки того, що нові зміни не призвели до появи нових дефектів у раніше працюючих функціях. Це допомагало забезпечити стабільність системи та уникнути повторних помилок.

5.6 Користувацьке тестування

Користувацьке тестування включало залучення реальних користувачів для перевірки зручності використання системи та її відповідності вимогам. Було проведено кілька сесій з користувачами, які виконували типові завдання, такі як реєстрація користувачів, створення замовлень, управління запасними частинами та моделями техніки, перегляд звітів. На основі їхнього зворотного зв'язку було внесено зміни до інтерфейсу користувача та функціональності системи.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи на тему " Програмна система для управління та обліку роботи сервісного центру з ремонту побутової техніки " було розроблено веб-застосунок, що відповідає поставленим вимогам та забезпечує ефективне управління процесами сервісного центру. Робота охоплювала всі етапи розробки програмного забезпечення, включаючи аналіз предметної галузі, формування вимог, проєктування архітектури та інтерфейсу користувача, реалізацію функціонал та тестування веб-застосунку.

Під час виконання роботи проведено глибокий аналіз предметної галузі, визначено основні проблеми, з якими стикаються сервісні центри з ремонту побутової техніки. Сформовано вимоги до програмної системи, що включають управління клієнтами, майстрами, замовленнями, запасними частинами, обчислення вартості ремонтних робіт та генерацію звітів.

Згідно з результатами аналізу було розроблено архітектуру системи з використанням мови Java та на основі фреймворку Spring Boot для серверної частини та мови JavaScript і бібліотеки React для клієнтської частини. Використання цих технологій забезпечило високу продуктивність, масштабованість та гнучкість системи. Проєктування бази даних здійснено з використанням реляційної моделі, що дозволило ефективно зберігати та обробляти дані про клієнтів, майстрів, замовлення та запасні частини.

Розроблено інтуїтивно зрозумілий та зручний інтерфейс користувача з використанням React та Bootstrap. Інтерфейс забезпечує простоту навігації та доступність основних функцій системи для різних типів користувачів (клієнти, майстри, адміністратори).

Розроблена система підтримує реєстрацію та аутентифікацію користувачів з використанням JWT, що забезпечує безпеку доступу до різних функцій. Реалізовано функції управління клієнтами, майстрами, замовленнями та запасними частинами, що дозволяють додавати, редагувати, видаляти та переглядати інформацію. Впроваджено механізми сортування, пошуку та фільтрації даних, що значно спрощують роботу з великою кількістю записів. Система автоматично

генерує статистику та звіти, що дозволяє керівництву сервісного центру отримувати важливу інформацію для прийняття рішень.

Проведено комплексне тестування системи, включаючи модульне, інтеграційне, функціональне, регресійне та користувацьке тестування. Використання сучасних інструментів та фреймворків дозволило забезпечити високу якість та надійність системи. За результатами тестування були виявлені та усунені дефекти, що забезпечило стабільну та продуктивну роботу системи.

Розроблена програмна система повністю відповідає поставленим завданням та вимогам. Вона забезпечує ефективне управління та облік роботи сервісного центру з ремонту побутової техніки, спрощує процеси обробки замовлень, управління запасними частинами та взаємодії з клієнтами. Завдяки використанню сучасних технологій, таких як Spring Boot, React, JWT та інші, система має високу продуктивність, безпеку та гнучкість. Результати тестування підтвердили стабільність та надійність системи.

Таким чином, можна зробити висновок, що розроблена програмна система має великий потенціал для впровадження у різних сервісних центрах з ремонту побутової техніки, сприяючи покращенню якості обслуговування клієнтів та оптимізації внутрішніх процесів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методичні вказівки до кваліфікаційної роботи бакалавра за спеціальністю 121 – Інженерія програмного забезпечення, освітньо-професійна програма «Програмна інженерія» для здобувачів усіх форм навчання / Упоряд.: З.В. Дудар, О. В. Вечур, В.І Каук, Р. В Мельнікова., В. Ю Нечволод, О.В. Олійник, Ю. С. Новіков, І.А. Ревенчук, І. П. Сокорчук, І. Ю. Шубін – Харків: ХНУРЕ, 2023. − 53 с.

2. Software for service businesses. orderry.com. URL: https://orderry.com/ (дата звернення: 14.06.2024).

3. Customer service software for the best customer experiences | Zendesk. Zendesk.com. URL: https://www.zendesk.com/service/ (дата звернення: 14.06.2024).

4. Field Service Management Software | ServiceMax Field Service Software. servicemax.com. URL: https://www.servicemax.com/ (дата звернення: 14.06.2024).

5. Програма для сервісного центру з ремонту побутової техніки. remonline.ua. URL: https://remonline.ua/appliance-repair-shop/ (дата звернення: 14.06.2024).

6. Java | Oracle. java.com. URL: https://www.java.com/en/ (дата звернення: 15.06.2024).

7. Spring Boot. spring.io. URL: https://spring.io/projects/spring-boot (дата звернення: 15.06.2024).

8. MySQL. mysql.com. URL: https://www.mysql.com/ (дата звернення: 15.06.2024).

9. React. react.dev. URL: https://react.dev/ (дата звернення: 15.06.2024).

10. Bootstrap. getbootstrap.com. URL: https://getbootstrap.com/(дата звернення: 15.06.2024).

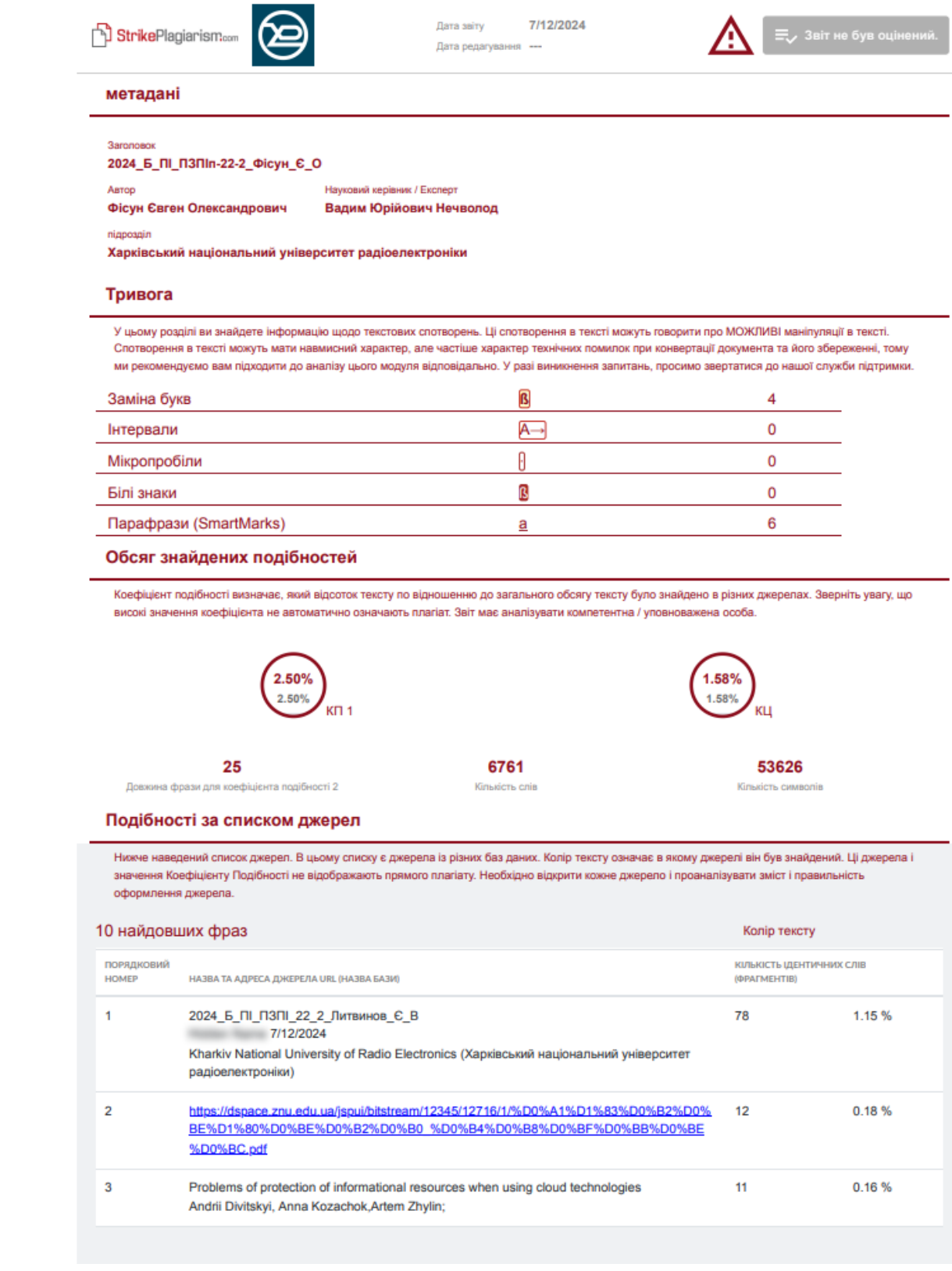
11. Home v6.20.1 | React Router. reactrouter.com. URL: https://reactrouter.com/en/main (дата звернення: 15.06.2024).

12. Axios. axios-http.com. URL: https://axios-http.com/uk/ (дата звернення: 15.06.2024).

13. JWT.IO. JSON Web Tokens - jwt.io. URL: https://jwt.io/ (дата звернення: 15.06.2024).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



StrikePlagiarism.com

Дата звіту 7/12/2024
Дата редагування ---

Звіт не був оцінений.

метадані

Заголовок
2024_Б_ПІ_ПЗПІп-22-2_Фісун_Є_О

Автор
Фісун Євген Олександрович

Науковий керівник / Експерт
Вадим Юрійович Нечволод

підрозділ
Харківський національний університет радіоелектроніки

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	4
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	6

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

2.50% 2.50% КП 1

1.58% 1.58% КЦ

25
Довжина фрази для коефіцієнта подібності 2

6761
Кількість слів

53626
Кількість символів

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	2024_Б_ПІ_ПЗПІ_22_2_Литвинов_Є_В [redacted] 7/12/2024 Kharkiv National University of Radio Electronics (Харківський національний університет радіоелектроніки)	78	1.15 %
2	https://dspace.znu.edu.ua/jspui/bitstream/12345/12716/1/%D0%A1%D1%83%D0%B2%D0%BE%D1%80%D0%BE%D0%B2%D0%B0_%D0%B4%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC.pdf	12	0.18 %
3	Problems of protection of informational resources when using cloud technologies Andrii Divitskyi, Anna Kozachok,Artem Zhylin;	11	0.16 %

ДОДАТОК Б

Слайди презентації

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кваліфікаційна робота бакалавра

Програмна система для управління та обліку роботи сервісного центру з ремонту побутової техніки

Виконав:
студент гр. ПЗПІп-22-2
Фісун Є.О.

Науковий керівник:
доктор технічних наук, професор каф. ПІ
Єрохін А.Л.

1

Вступ

Актуальність роботи:
- Зростаюча кількість побутової техніки вимагає ефективного сервісного обслуговування.
- Необхідність автоматизації процесів управління сервісними центрами.

Мета роботи: Розробка веб-застосунку для автоматизації роботи сервісного центру.

Завдання роботи:
- Аналіз предметної області.
- Формування вимог до системи.
- Проєктування та розробка системи.
- Тестування системи.

2

Аналіз предметної галузі

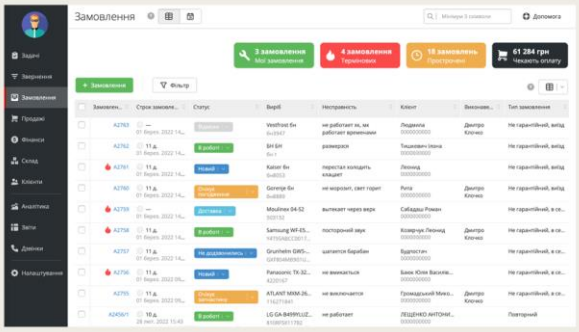
Аналіз предметної галузі:

- Основні проблеми сервісних центрів.
- Вимоги до ефективного управління та обліку.

Виявлення та вирішення проблем: Необхідність автоматизації обліку замовлень, клієнтів та запасних частин.

Постановка задачі: Створення веб-застосунку, що дозволяє ефективно керувати всіма аспектами роботи сервісного центру.

Аналіз аналогів



RemOnline – це програма для сервісного центру з ремонту побутової техніки, орієнтована на комплексну автоматизацію та облік роботи.

\+ Можливість створення та обліку всіх необхідних для роботи сервісного центру сутностей

\+ Генерація гарантійних талонів та ведення фінансового обліку

\+ Мобільний доступ

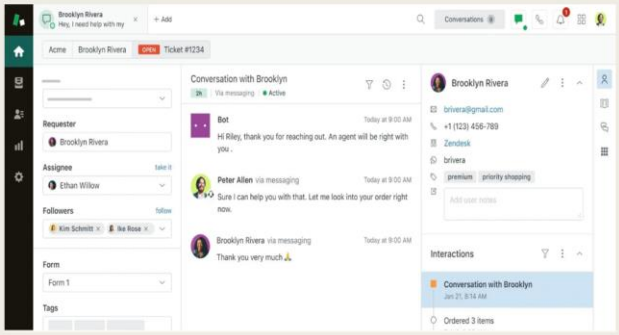
\+ Локалізація

\- Не має інтеграції зі ШІ

Аналіз аналогів

Zendesk - це платформа для обслуговування клієнтів та управління тікетами, яка може бути корисною для сервісних центрів.

+ Багато каналів комунікації з клієнтами
+ Широкий набір статистик
- Не має локалізації
- Не має інтеграції зі ШІ
- Висока ціна
- Застосунок більшою мірою орієнтований на роботу з клієнтами, тому в ньому неможливо вести облік складу



5

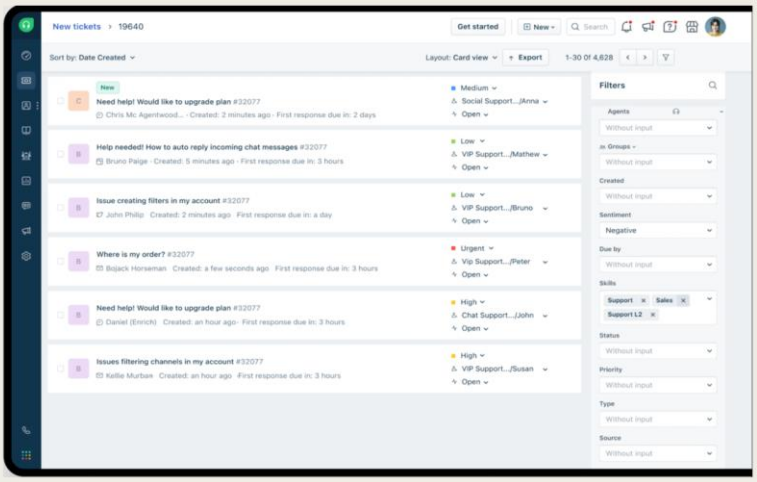
Аналіз аналогів

ServiceMax - це платформа для управління обслуговуванням клієнтів і технічним обслуговуванням, що надає ряд функцій для цього.

+ Генерація гарантійних талонів та ведення фінансового обліку
+ Мобільний доступ
+ Інтеграція зі ШІ
- Не має локалізації
- Дуже висока ціна

6

Аналіз аналогів



Freshdesk - це платформа для обслуговування клієнтів, яка може бути використана для управління замовленнями та сервісним центром.

\+ Багато каналів комунікації з клієнтами

\+ Широкий набір статистик

\- Не має локалізації

\- Не має інтеграції зі ШІ

\- Висока ціна

\- Застосунок більшою мірою орієнтований на роботу з клієнтами, тому в ньому неможливо вести облік складу та фінансів

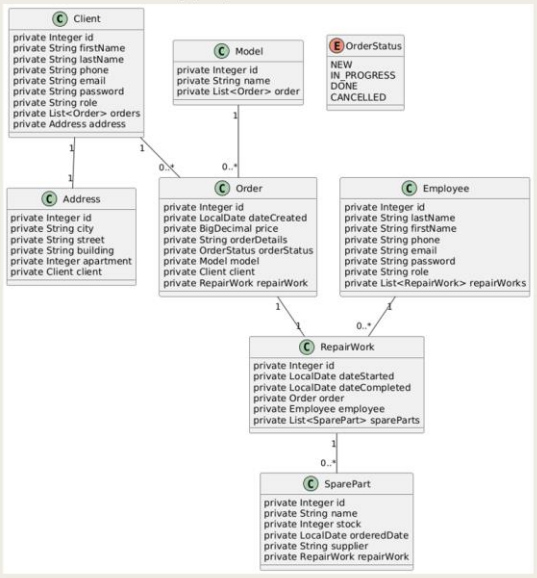
Формування вимог до програмної системи

Основні вимоги:

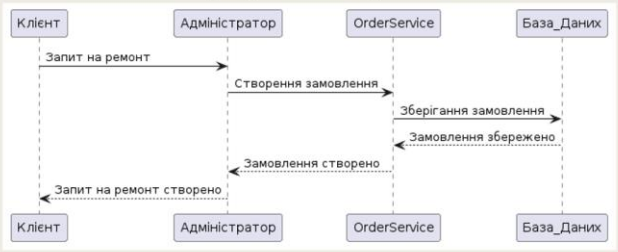
- Управління клієнтами, майстрами, замовленнями, запасними частинами
- Обчислення вартості робіт та генерація звітів
- Сортування, пошук та фільтрація даних
- Реєстрація та аутентифікація користувачів з різними ролями
- Автоматичне надсилання повідомлень клієнтам

UML проєктування ПЗ

Діаграма класів

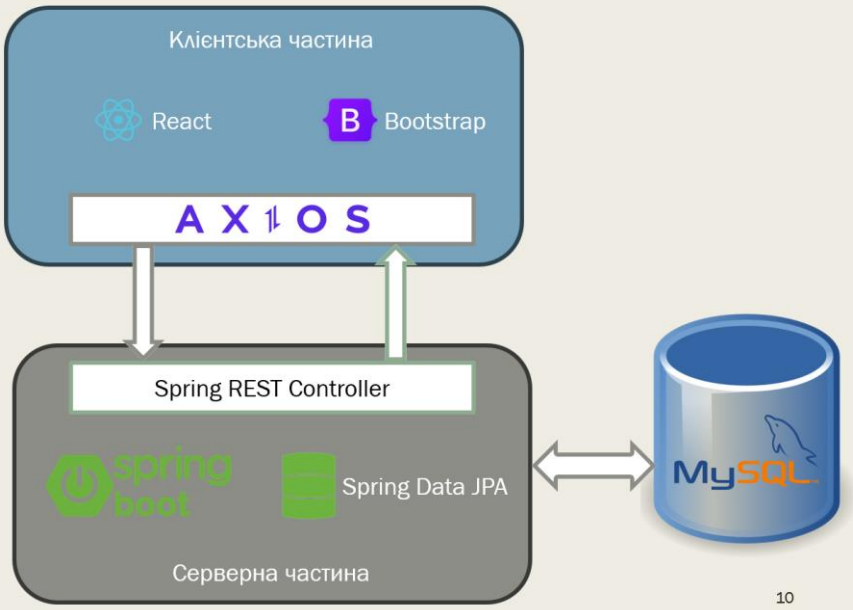


Діаграма послідовностей



9

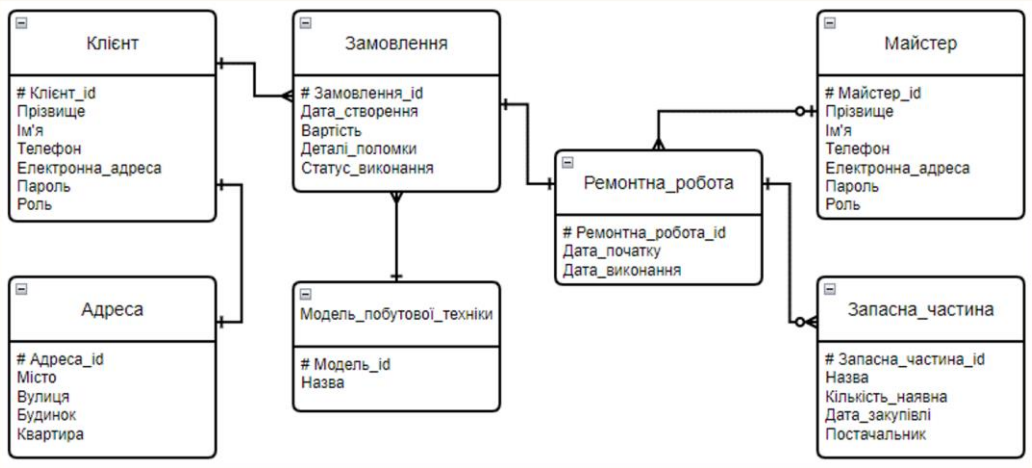
Проєктування архітектури ПЗ



10

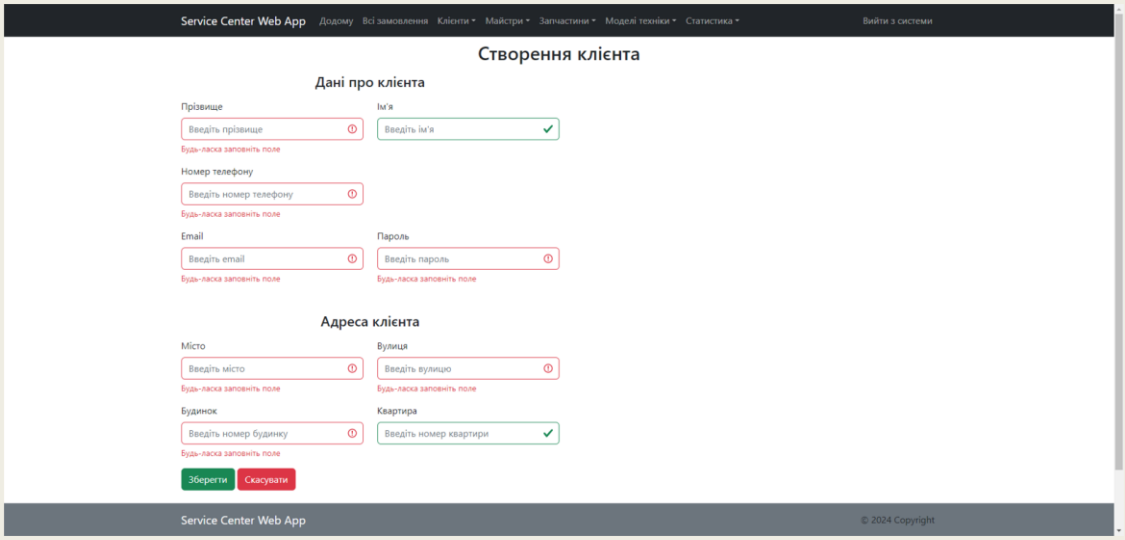
Проєктування структури зберігання даних

ER-діаграма



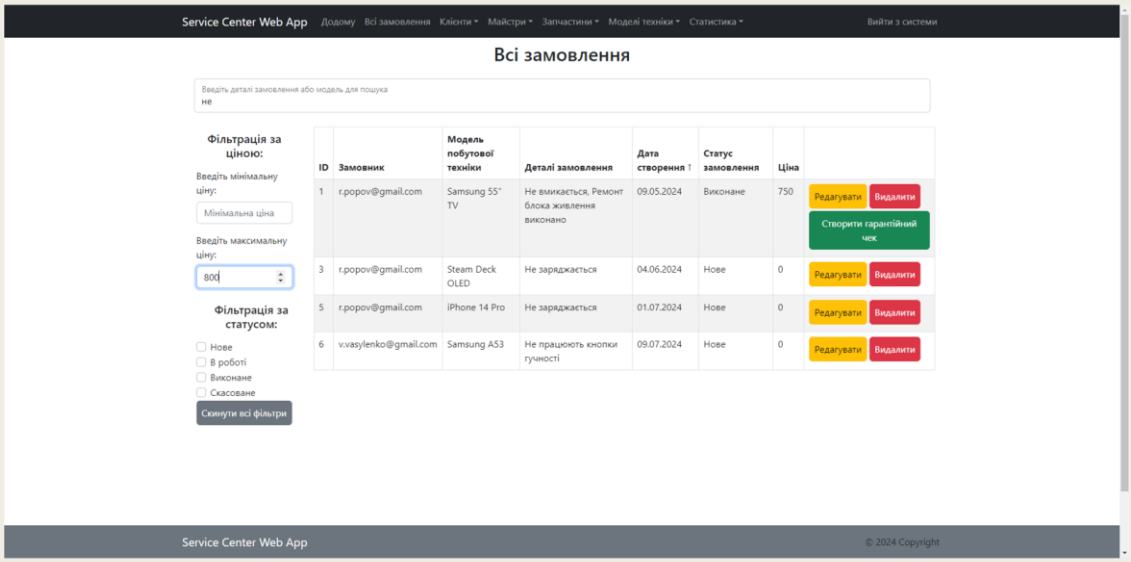
11

Інтерфейс користувача



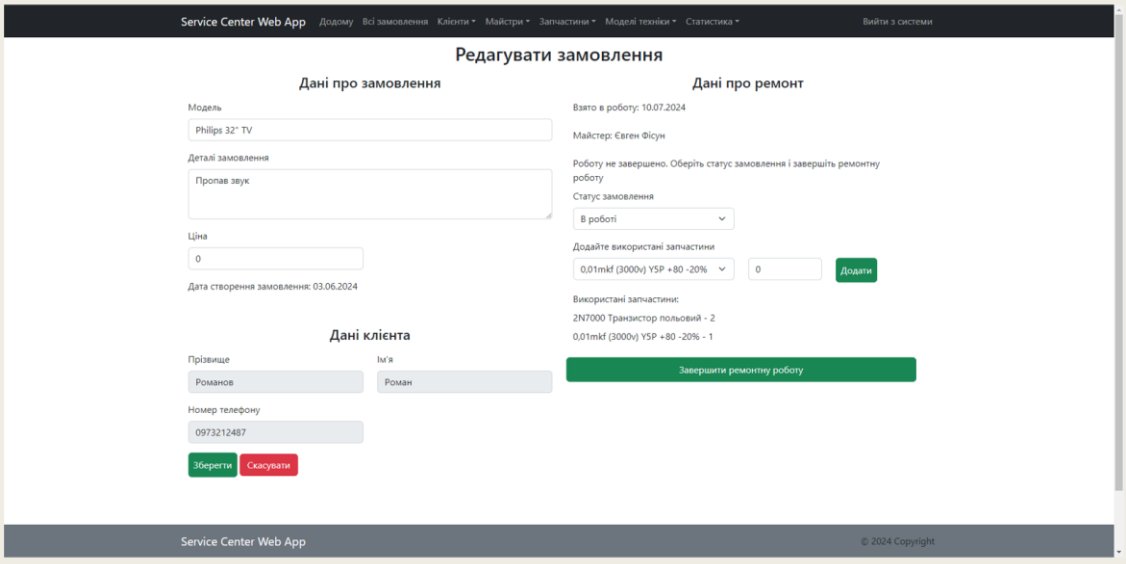
12

Інтерфейс користувача



13

Інтерфейс користувача



14

Тестування розробленого ПЗ

Методи тестування:

- Модульне тестування;
- Інтеграційне тестування;
- Функціональне тестування;
- Регресійне тестування;
- Користувацьке тестування.

Інструменти тестування: JUnit, Spring Boot Test.

Результати тестування: Висока стабільність та надійність системи.

15

Висновки

Основні результати роботи:

- Створено ефективний інструмент для управління сервісним центром.
- Забезпечено автоматизацію основних процесів.

Оцінка роботи: Система відповідає вимогам, є надійною та зручною у використанні.

Перспективи розвитку:

- Можливість розширення функціоналу.
- Інтеграція з іншими системами.

16

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК В

Лістинг класу ClientServiceImpl.java

```
package com.servicecenter.backend.service.impl;

import com.servicecenter.backend.dto.ClientDto;
import com.servicecenter.backend.dto.Response;
import com.servicecenter.backend.exception.CustomException;
import com.servicecenter.backend.exception.ResourceNotFoundException;
import com.servicecenter.backend.model.Client;
import com.servicecenter.backend.repository.ClientRepository;
import com.servicecenter.backend.service.ClientService;
import com.servicecenter.backend.utils.Utils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.List;

import static java.lang.String.format;

@Service
public class ClientServiceImpl implements ClientService {

    private  static final String CLIENT_ROLE_NAME = "CLIENT";

    @Autowired
    private ClientRepository repository;
    @Autowired
    private PasswordEncoder passwordEncoder;

    @Override
    public Response saveClient(Client client) {
        Response response = new Response();
        try {
            if (client.getRole() == null || client.getRole().isEmpty()) {
                client.setRole(CLIENT_ROLE_NAME);
            }
            if (repository.existsByEmail(client.getEmail())) {
                throw new CustomException(String.format("Employee with '%s' email
already exists", client.getEmail()));
            }
            client.setPassword(passwordEncoder.encode(client.getPassword()));
            Client savedClient = repository.save(client);
            ClientDto clientDto = Utils.mapClientEntityToClientDto(savedClient);
            response.setStatusCode(200);
            response.setClient(clientDto);
        } catch (CustomException e) {
            response.setStatusCode(400);
            response.setMessage(e.getMessage());
        } catch (Exception e) {
            response.setStatusCode(500);
            response.setMessage("Error occured during employee registration " +
e.getMessage());
        }
        return response;
    }
```

```
    @Override
    public List<Client> getAllClients() {
        return repository.findAll();
    }

    @Override
    public Client getClientById(int id) {
        return repository.findById(id).orElseThrow(() -> new
ResourceNotFoundException(getClientNotFoundErrorMessage(id)));
    }

    @Override
    public Client updateClient(int id, Client client) {
        Client existingClient = repository.findById(id).orElseThrow(() -> new
ResourceNotFoundException(getClientNotFoundErrorMessage(client.getId())));
        existingClient.setLastName(client.getLastName());
        existingClient.setFirstName(client.getFirstName());
        existingClient.setPhone(client.getPhone());
        existingClient.setEmail(client.getEmail());
        existingClient.setAddress(client.getAddress());
        return repository.save(existingClient);
    }

    @Override
    public String deleteClient(int id) {
        Client client = repository.findById(id).orElseThrow(() -> new
ResourceNotFoundException(getClientNotFoundErrorMessage(id)));
        if (client.getOrders() != null && !client.getOrders().isEmpty()) {
            throw new CustomException("Client has orders, cannot be deleted");
        }
        repository.deleteById(id);
        return format("Client with id %d removed", id);
    }

    @Override
    public List<Client> searchClientsAny(String keyword) {
        return
repository.findByFirstNameContainingOrLastNameContainingOrPhoneContainingOrEmailConta
ining(keyword, keyword, keyword, keyword);
    }

    @Override
    public Response getMyInfo(String email) {
        Response response = new Response();
        try {
            Client client = repository.findByEmail(email).orElseThrow(() -> new
ResourceNotFoundException(format("Client with email %s not found", email)));
            ClientDto clientDto = Utils.mapClientEntityToClientDto(client);
            response.setStatusCode(200);
            response.setMessage("successful");
            response.setClient(clientDto);
        } catch (CustomException e) {
            response.setStatusCode(404);
            response.setMessage(e.getMessage());
        } catch (Exception e) {
            response.setStatusCode(500);
            response.setMessage("Error getting client " + e.getMessage());
        }
```

```
        return response;
    }

    private String getClientNotFoundErrorMessage(int id) {
        return format("Client with id %d not found", id);
    }
}
```

ДОДАТОК Г

Лістинг класу EmployeeServiceImpl.java

```
package com.servicecenter.backend.service.impl;

import com.servicecenter.backend.dto.EmployeeDto;
import com.servicecenter.backend.dto.Response;
import com.servicecenter.backend.exception.CustomException;
import com.servicecenter.backend.exception.ResourceNotFoundException;
import com.servicecenter.backend.model.Employee;
import com.servicecenter.backend.repository.EmployeeRepository;
import com.servicecenter.backend.service.EmployeeService;
import com.servicecenter.backend.utils.Utils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.List;

import static java.lang.String.format;

@Service
public class EmployeeServiceImpl implements EmployeeService {

    @Autowired
    private EmployeeRepository repository;
    @Autowired
    private PasswordEncoder passwordEncoder;

    @Override
    public Response registerEmployee(Employee employee) {
        Response response = new Response();
        try {
            if (employee.getRole() == null || employee.getRole().isEmpty()) {
                employee.setRole("EMPLOYEE");
            }
            if (repository.existsByEmail(employee.getEmail())) {
                throw new CustomException(String.format("Employee with '%s' email
already exists", employee.getEmail()));
            }
            employee.setPassword(passwordEncoder.encode(employee.getPassword()));
            Employee savedEmployee = repository.save(employee);
            EmployeeDto employeeDto =
Utils.mapEmployeeEntityToEmployeeDto(savedEmployee);
            response.setStatusCode(200);
            response.setEmployee(employeeDto);
        } catch (CustomException e) {
            response.setStatusCode(400);
            response.setMessage(e.getMessage());
        } catch (Exception e) {
            response.setStatusCode(500);
            response.setMessage("Error occured during employee registration " +
e.getMessage());
        }
        return response;
    }
```

```
    @Override
    public Response getAllEmployees() {
        Response response = new Response();
        try {
            List<Employee> employees = repository.findAll();
            List<EmployeeDto> employeeDtos =
Utils.mapEmployeeListEntityToEmployeeListDto(employees);
            response.setStatusCode(200);
            response.setMessage("successful");
            response.setEmployees(employeeDtos);
        } catch (Exception e) {
            response.setStatusCode(500);
            response.setMessage("Error occured during fetching employees " +
e.getMessage());
        }
        return response;
    }

    @Override
    public Response getEmployeeById(int id) {
        Response response = new Response();
        try {
            Employee employee = repository.findById(id).orElseThrow(() -> new
CustomException(getEmployeeNotFoundErrorMessage(id)));
            EmployeeDto employeeDto = Utils.mapEmployeeEntityToEmployeeDto(employee);
            response.setStatusCode(200);
            response.setMessage("successful");
            response.setEmployee(employeeDto);
        } catch (CustomException e) {
            response.setStatusCode(404);
            response.setMessage(e.getMessage());
        } catch (Exception e) {
            response.setStatusCode(500);
            response.setMessage("Error occured during fetching employee " +
e.getMessage());
        }
        return response;
    }

    @Override
    public Employee updateEmployee(int id, Employee employee) {
        Employee existingEmployee = repository.findById(id).orElseThrow(() -> new
ResourceNotFoundException(getEmployeeNotFoundErrorMessage(employee.getId())));
        existingEmployee.setLastName(employee.getLastName());
        existingEmployee.setFirstName(employee.getFirstName());
        existingEmployee.setPhone(employee.getPhone());
        existingEmployee.setEmail(employee.getEmail());
        existingEmployee.setRole(employee.getRole());
        return repository.save(existingEmployee);
    }

    @Override
    public Response deleteEmployee(int id) {
        Response response = new Response();
        try {
            repository.findById(id).orElseThrow(() -> new
CustomException(getEmployeeNotFoundErrorMessage(id)));
            repository.deleteById(id);
            response.setStatusCode(200);
```

```
            response.setMessage("successful");
        } catch (CustomException e) {
            response.setStatusCode(404);
            response.setMessage(e.getMessage());
        } catch (Exception e) {
            response.setStatusCode(500);
            response.setMessage("Error occured during deleting employee " +
e.getMessage());
        }
        return response;
    }

    @Override
    public List<Employee> searchEmployeesAny(String keyword) {
        return
repository.findByFirstNameContainingOrLastNameContainingOrPhoneContaining(keyword,
keyword, keyword);
    }

    @Override
    public Response getMyInfo(String email) {
        Response response = new Response();
        try {
            Employee employee = repository.findByEmail(email).orElseThrow(() -> new
CustomException("Employee Not Found"));
            EmployeeDto employeeDto = Utils.mapEmployeeEntityToEmployeeDto(employee);
            response.setStatusCode(200);
            response.setMessage("successful");
            response.setEmployee(employeeDto);
        } catch (CustomException e) {
            response.setStatusCode(404);
            response.setMessage(e.getMessage());
        } catch (Exception e) {
            response.setStatusCode(500);
            response.setMessage("Error getting employee " + e.getMessage());
        }
        return response;
    }

    private String getEmployeeNotFoundErrorMessage(int id) {
        return format("Employee with id %d not found", id);
    }
}
```

ДОДАТОК Д

Лістинг класу AuthServiceImpl.java

```
package com.servicecenter.backend.service.impl;

import com.servicecenter.backend.dto.LoginRequest;
import com.servicecenter.backend.dto.Response;
import com.servicecenter.backend.exception.CustomException;
import com.servicecenter.backend.model.Client;
import com.servicecenter.backend.model.Employee;
import com.servicecenter.backend.repository.ClientRepository;
import com.servicecenter.backend.repository.EmployeeRepository;
import com.servicecenter.backend.service.AuthService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Service;

@Service
public class AuthServiceImpl implements AuthService {

    @Autowired
    private ClientRepository clientRepository;
    @Autowired
    private EmployeeRepository employeeRepository;
    @Autowired
    private JwtService jwtService;
    @Autowired
    private AuthenticationManager authenticationManager;

    @Override
    public Response login(LoginRequest loginRequest) {
        Response response = new Response();
        try {
            Authentication authenticate = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(loginRequest.getEmail(),
loginRequest.getPassword()));
            String token;
            if (authenticate.getPrincipal() instanceof Employee) {
                Employee employee =
employeeRepository.findByEmail(loginRequest.getEmail()).orElseThrow(() -> new
CustomException("Employee not found"));
                token = jwtService.generateToken(employee);
                response.setRole(employee.getRole());
            } else {
                Client client =
clientRepository.findByEmail(loginRequest.getEmail()).orElseThrow(() -> new
CustomException("Client not found"));
                token = jwtService.generateToken(client);
                response.setRole(client.getRole());
            }
            response.setStatusCode(200);
            response.setToken(token);
            response.setExpirationTime("7 Days");
            response.setMessage("successful");
```

```
        } catch (CustomException e) {
            response.setStatusCode(400);
            response.setMessage(e.getMessage());
        } catch (Exception e) {
            response.setStatusCode(500);
            response.setMessage("Error occured during login: " + e.getMessage());
        }
        return response;
    }
}
```

ДОДАТОК Е

Лістинг класу AuthController.java

```
package com.servicecenter.backend.controller;

import com.servicecenter.backend.dto.LoginRequest;
import com.servicecenter.backend.dto.Response;
import com.servicecenter.backend.service.AuthService;
import com.servicecenter.backend.service.ClientService;
import com.servicecenter.backend.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/auth")
public class AuthController {

    private static final String CLIENT_ROLE = "CLIENT";

    @Autowired
    private EmployeeService employeeService;
    @Autowired
    private ClientService clientService;
    @Autowired
    private AuthService authService;

    @PostMapping("/login")
    public ResponseEntity<Response> login(@RequestBody LoginRequest loginRequest) {
        Response response = authService.login(loginRequest);
        return ResponseEntity.status(response.getStatusCode()).body(response);
    }

    @GetMapping("/get-logged-in-profile-info")
    public ResponseEntity<Response> getLoggedInUserProfile() {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        String role = authentication.getAuthorities().toArray()[0].toString();
        String email = authentication.getName();
        Response response;
        if (role.equals(CLIENT_ROLE)) {
            response = clientService.getMyInfo(email);
        } else {
            response = employeeService.getMyInfo(email);
        }
        return ResponseEntity.status(response.getStatusCode()).body(response);
    }
}
```