Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система підтримки обліку робіт з ремонту та обслуговування комп'ютерної техніки у сервісному центрі
(тема)

Виконав:
студент 4 курсу, групи ПЗПІп-22-1

Козаченко Д. Ю.
(прізвище, ініціали)

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

Керівник доц. кафедри ПІ Кириченко І. В.
(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 Козаченку Дмитру Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система підтримки обліку робіт з ремонту та обслуговування комп'ютерної техніки у сервісному центрі
Затверджена наказом по університету від 17.06. 2024р. № 588 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 01.07.2024
3. Вихідні дані до роботи Розробити програмну систему підтримки обліку робіт з ремонту та обслуговування комп'ютерної техніки, що дозволяє автоматизувати діяльність сервісного центру
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис програми, тестування програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	15.05.2024	*виконано*
2	Створення специфікації ПЗ	22.05.2024	*виконано*
3	Проектування ПЗ	30.05.2024	*виконано*
4	Розробка ПЗ	10.06.2024	*виконано*
5	Тестування ПЗ	20.06.2024	*виконано*
6	Оформлення пояснювальної записки	22.06.2024	*виконано*
7	Підготовка презентації та доповіді	26.06.2024	*виконано*
8	Попередній захист	27.06.2024	*виконано*
9	Нормоконтроль, рецензування	27.06.2024	*виконано*
10	Здача роботи у електронний архів	27.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	29.06.2024	*виконано*

Дата видачі завдання 06 травня 2024р.

Студент (ка)_____ Козаченко Д. Ю.
(підпис)

Керівник роботи _____ доц. кафедри ПІ Кириченко І. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 71 с., 14 рисунків, 3 таблиці, 10 джерел.

МОВА ПРОГРАМУВАННЯ JAVASCRIPT, НЕРЕЛЯЦІЙНА БАЗА ДАНИХ, СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ, MONGO DB, REACT. JS, NODE. JS,

Метою кваліфікаційної роботи є розробка програмної системи підтримки обліку робіт з ремонту та обслуговування комп'ютерної техніки у сервісному центрі з використанням нереляційних баз даних та логіки об'єктно-орієнтованого програмування.

В ході створення додатку були використані середа розробки Microsoft Visual Studio Code, платформа Node.js, мова програмування JavaScript, фреймворк React.js для інтерфейсу користувача та система управління базами даних Mongo DB.

PROGRAMMING LANGUAGE JAVASCRIPT, REACT.JS, NODE.JS, NON-RELATIONAL DATABASE, DATABASE MANAGEMENT SYSTEM, MONGODB.

The objective of the qualification work is to develop a Support Software System for Recording Repair and Maintenance Work of Computer Equipment in a Service Center using non-relational databases and object-oriented programming logic.

During the development of the application, the following were used: Microsoft Visual Studio Code as the development environment, Node.js platform, JavaScript programming language, React.js framework for the user interface, and MongoDB database management system.

Я, Козаченко Дмитро Юрійович, студент гр. ПЗПІп-22-1, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система підтримки обліку робіт з ремонту та обслуговування комп'ютерної техніки у сервісному центрі», 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

CRM – Customer Relationship Management

CRUD – Create, Read, Update, Delete

ERP – Enterprise Resource Planning

ID – Identifier

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

JSON – JavaScript Object Notation

JWT – JSON Web Token

TCP – Transmission Control Protocol

WMS – Warehouse Management System

АТС – Автоматична телефонна станція

БД – База даних

ІПН – Індивідуальний податковий номер

ПІБ – Прізвище, ім’я, по-батькові

СЦ – Сервісний центр

УСО – Управління сервісним обслуговуванням

ВСТУП

На сьогоднішній день основна діяльність сервісних центрів з ремонту та обслуговування комп'ютерної техніки полягає в тому, що вони приймають від юридичних та фізичних осіб пристрої, які потребують ремонту, модернізації або будь-яких інших дій, не можливих без втручання фахівців. При цьому в ході ремонтних робіт у більшості випадків фахівці сервісних центрів спираються на свій особистий досвід з ремонту та обслуговування комп'ютерного обладнання.

Для підприємств, що надають послуги з ремонту та обслуговування, облік інформації про стан кожного виробу в конкретний момент є одним з найважливіших факторів якісної роботи. Гарантом успіху виробничо-технічного процесу організації є поінформованість всіх учасників у конкретний час.

На даний момент облік робіт з ремонту та обслуговування комп'ютерної техніки здійснюється вручну. Зі збільшенням числа клієнтів, і навіть зі збільшенням спектра послуг зростає кількість конфліктів, які виражаються, насамперед, у цьому, що менеджер, приймаючи устаткування, може втратити акт прийому з ремонту техніки, у результаті, термін виконання замовлення може бути втрачено. Крім того, така схема діяльності збільшує час обслуговування клієнтів, оскільки техніка надходить асинхронно, що призводить до плутанини під час пікових навантажень.

У ситуації, коли з компанії йде досвідчений інженер, разом з ним зникає і накопичений досвід з ремонту того чи іншого обладнання. Керівник сервісу змушений нести зайві фінансові та тимчасові витрати на навчання нового співробітника. Ці витрати можна скоротити, спираючись на досвід виконаних замовлень.

Отже, мета кваліфікаційної роботи полягає у створенні програмної системи підтримки обліку робіт з ремонту та обслуговування комп’ютерної техніки, що дозволяє автоматизувати діяльність сервісного центру.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

До сервісного центру з обслуговування комп'ютерної техніки надходять на ремонт вироби різних типів, виробників та моделей у несправному стані від фізичних і юридичних осіб. Прийом виробу здійснює співробітник СЦ, який заповнює паперовий бланк – акт прийому [1]. У цьому акті містяться наступні дані:

- номер акта;
- ім'я замовника, контактна інформація, платіжні реквізити для юридичних осіб;
- тип, виробник, модель пристрою, серійний номер;
- опис несправності (за словами замовника);
- дата прийому виробу в ремонт;
- ПІБ та підпис приймача;
- ПІБ та підпис особи, яка здала виріб у ремонт.

Заповнений акт прийому видається замовнику і підтверджує факт прийому виробу в СЦ для ремонту. Після цього на виріб наклеюється стікер із номером замовлення, і він передається до майстерні СЦ. Замовленню присвоюється статус 1 «прийнято до СЦ». Копія акта прийому залишається у СЦ.

Інженер СЦ приймає замовлення, проводить діагностику виробу, визначає несправності та доцільність подальшого ремонту. Якщо ремонт можливий і замовник згоден, інженер проводить ремонтні роботи.

Інженер записує до журналу виконаних робіт види виконаних робіт та поточний статус замовлення:

- статус 2 – проводиться діагностика;
- статус 3 – діагностика проведена, ремонт неможливий;
- статус 4 – діагностика проведена, ремонт можливий;
- статус 5 – ремонт проведено успішно.

Вартість кожного виду робіт встановлюється менеджером, прейскурант може змінюватися приймальниками або інженерами.

Після успішного завершення ремонту замовника інформують про закінчення робіт. Сума до оплати визначається підсумовуванням вартості робіт згідно з прейскурантом. Після оплати замовнику повертається виріб, і в замовленні фіксується дата видачі. Замовленню присвоюється статус 6 «виріб виданий замовнику».

Інформація про несправності моделей пристроїв та методи їх усунення, зібрана під час діагностики та ремонту, записується до журналу виконаних робіт. Це дозволяє прискорити та покращити технічне обслуговування подібних несправностей у майбутньому та служить довідковим посібником для нових співробітників сервісного центру.

1.2 Аналіз існуючих аналогів

1.2.1. Система «Alldevice»

Alldevice – це програмне забезпечення, розроблене для ефективного управління технічним обслуговуванням і ремонтом будь-якого обладнання, яке потребує регулярного обслуговування і ремонту [2].

Автоматизована система Alldevice призначена для управління будь-яким типом обладнання, яке потребує періодичного обслуговування. Основна мета системи – взяти на себе управління технічним обслуговуванням і ремонтними роботами для обраного обладнання, сукупності обладнання або об'єктів.

Alldevice складається з трьох основних функціональних компонентів:

- адміністрування робіт (управління як позаплановими ремонтами, так і плановими роботами з технічного обслуговування і ремонту обладнання. Система дозволяє створювати завдання і направляти їх відповідним виконавцям);
- база даних (зберігає всю інформацію, що стосується обладнання або об'єктів, включаючи специфікації, технічні креслення, інструкції, записи про ремонт і обслуговування. Це фактично є електронним щоденником обслуговування і ремонтів для кожного виду обладнання);
- аналіз несправностей (система проводить аналіз несправностей, виявля-

ючи обладнання або його компоненти, які створюють проблеми і потребують уваги обслуговуючого персоналу. Це допомагає ефективніше управляти технічним обслуговуванням).

Alldevice відзначається простотою у використанні, надаючи детальний огляд як виконаних, так і запланованих робіт з технічного обслуговування і ремонту. Це забезпечує впевненість у тому, що все необхідне знаходиться під контролем і жодне завдання не залишиться незавершеним.

Основні функції та можливості Alldevice:

- розмежування прав користувачів (система дозволяє налаштовувати різні рівні доступу для користувачів, забезпечуючи безпеку даних і контроль доступу);
- структурування об'єктів за принципом дерева каталогів (дозволяє організувати об'єкти і обладнання в зручну ієрархічну структуру для полегшення навігації та управління);
- можливість додавання необмеженої кількості об'єктів і підлеглих об'єктів (система не обмежує кількість доданих об'єктів, що дозволяє масштабувати управління на великі підприємства);
- швидкий пошук обладнання зі списку об'єктів (дерево об'єктів) (забезпечує швидкий доступ до необхідного обладнання або об'єкта, економлячи час користувачів);
- можливість копіювання обладнання або об'єктів для спрощення введення даних (ця функція дозволяє легко створювати нові записи на основі вже існуючих, що значно прискорює процес введення даних);
- додавання файлів на карту об'єкта (малюнки, інструкції тощо) (всі необхідні файли можна прикріпити до відповідного об'єкта, що забезпечує зручний доступ до всієї супровідної документації);
- зберігання всієї інформації про обладнання на картці обладнання (на картці обладнання знаходяться технічні дані, файли, додаткова інформація, а також записи про виконані і заплановані робочі замовлення);

- автоматичне генерування робочих замовлень для регулярного обслуговування (система автоматично створює робочі замовлення для регулярного обслуговування обладнання, що дозволяє не пропустити жодного планового завдання);
- можливість додавання робочих замовлень та інформації про ремонт (користувачі можуть додавати нові робочі замовлення та записи про ремонт, що полегшує управління ремонтними роботами);
- можливість додавання планового (нерегулярного) обслуговування (система дозволяє планувати та додавати завдання для нерегулярного обслуговування, забезпечуючи гнучкість в управлінні технічним обслуговуванням);
- перелік усіх робочих замовлень (користувачі можуть переглядати всі робочі замовлення в системі, що допомагає відслідковувати прогрес і статус виконання робіт);
- звіти про виконані робочі замовлення (система генерує звіти про виконані роботи, з можливістю фільтрації за об'єктом, обладнанням, працівником або періодом часу. Це забезпечує детальний аналіз ефективності роботи).

Таким чином, Alldevice є потужним інструментом для управління технічним обслуговуванням і ремонтом обладнання, забезпечуючи високу ефективність і контроль за всіма процесами.

Однією з ключових переваг Alldevice є її здатність адмініструвати як позапланові ремонтні роботи, так і планові технічні обслуговування. Це дозволяє сервісним центрам і майстерням ефективно планувати свою роботу, розподіляти завдання серед співробітників і контролювати їх виконання. Система автоматично генерує робочі замовлення для регулярного обслуговування, що значно знижує ймовірність пропуску важливих технічних заходів.

Додаткові можливості Alldevice включають розмежування прав користувачів, що дозволяє налаштувати різні рівні доступу до даних і функцій системи. Це забезпечує безпеку інформації і контроль над виконанням робіт. Структурування об'єктів за принципом дерева каталогів дозволяє організувати дані в зручну і зрозумілу ієрархію, що спрощує навігацію і управління.

Скріншот головної сторінки наведений на рисунку 1.1.

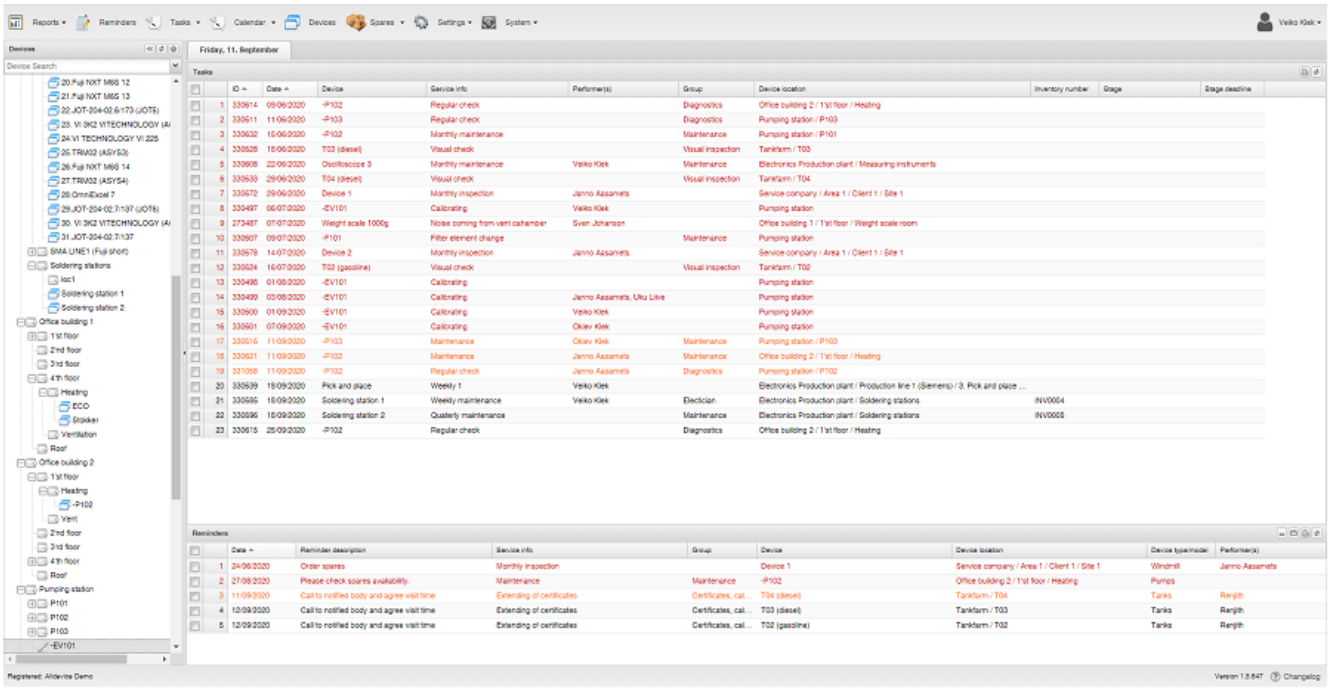


Рисунок 1.1 – Скріншот головної сторінки системи «Alldevice» [2]

1.2.2 Система «Gincore»

Gincore – це програмне забезпечення, яке спочатку розроблялося для обліку замовлень на ремонт електроніки, таких як мобільні телефони, ноутбуки і комп'ютери. На сьогоднішній день це потужна система для автоматизації бізнес-процесів у будь-якій ремонтній майстерні або сервісному центрі, незалежно від ніші та сфери діяльності [3].

Система Gincore включає в себе:

- онлайн журнал для обліку замовлень на ремонт (забезпечує зручне і швидке ведення обліку замовлень, дозволяючи відстежувати статуси і терміни виконання робіт);
- складську систему (містить адресну систему зберігання, унікальні номери для кожного товару, історію переміщень, оприбуткування та списання кожного виробу);
- бухгалтерію (дозволяє вести взаєморозрахунки з постачальниками, записувати в борг, списувати товари і грошові кошти, показувати операційну і чистий прибуток підприємства, автоматично розраховувати заробітну плату

співробітникам та інше);

- CRM для роботи з клієнтською базою (забезпечує гнучке управління клієнтами, ведення історії дзвінків, смс, кількість звернень, журнал прийнятої техніки клієнта з датами і історією ремонтів);
- аналітику та статистику (надає інструменти для детального аналізу і статистики, що допомагає оптимізувати бізнес-процеси);
- інструменти інтеграції з телефонією і смс-провайдерами (дозволяють ефективно комунікувати з клієнтами);
- інтеграцію з Google Analytics (забезпечує додаткову можливість відстеження і аналізу даних).

Проект Gincore постійно розвивається, враховуючи побажання користувачів. Щомісяця в програмі з'являються нові можливості, опції та функціональні рішення. Це робить Gincore єдиною системою обліку онлайн для сервісних центрів, яка об'єднує в собі серію професійних програмних рішень.

Переваги Gincore над іншими системами для сервісних центрів:

- повноцінна бухгалтерія (система дозволяє вести детальний облік взаєморозрахунків з постачальниками, записувати в борг, списувати з балансу товари і грошові кошти, відображає операційну і чисту прибуток підприємства, автоматично розраховує заробітну плату співробітникам та багато іншого);
- складська система з адресною системою зберігання (унікальна система зберігання, яка присвоює унікальні номери кожному товару, має унікальну історію переміщень, оприбуткування та списання кожного виробу);
- CRM (багатофункціональна система для управління клієнтською базою, що дозволяє створювати записи клієнтів, вести історію дзвінків, смс, кількість звернень, а також журнал прийнятої техніки клієнта з датами і історією ремонтів);
- автоматизація майстерні (система автоматично контролює терміни виконання ремонтних робіт, нагадуючи співробітникам про необхідність звернути увагу на певні замовлення);

- інтеграція з Google Analytics (дозволяє відстежувати і аналізувати дані для покращення ефективності бізнесу).

Завдяки унікальному поєднанню потужного функціоналу та зручного інтуїтивно зрозумілого інтерфейсу, Gincore не має аналогів на ринку СНД. Це робить її незамінною системою для сервісних центрів, які прагнуть автоматизувати свої бізнес-процеси і підвищити ефективність роботи. Скріншот сторінки наведений на рисунку 1.2.

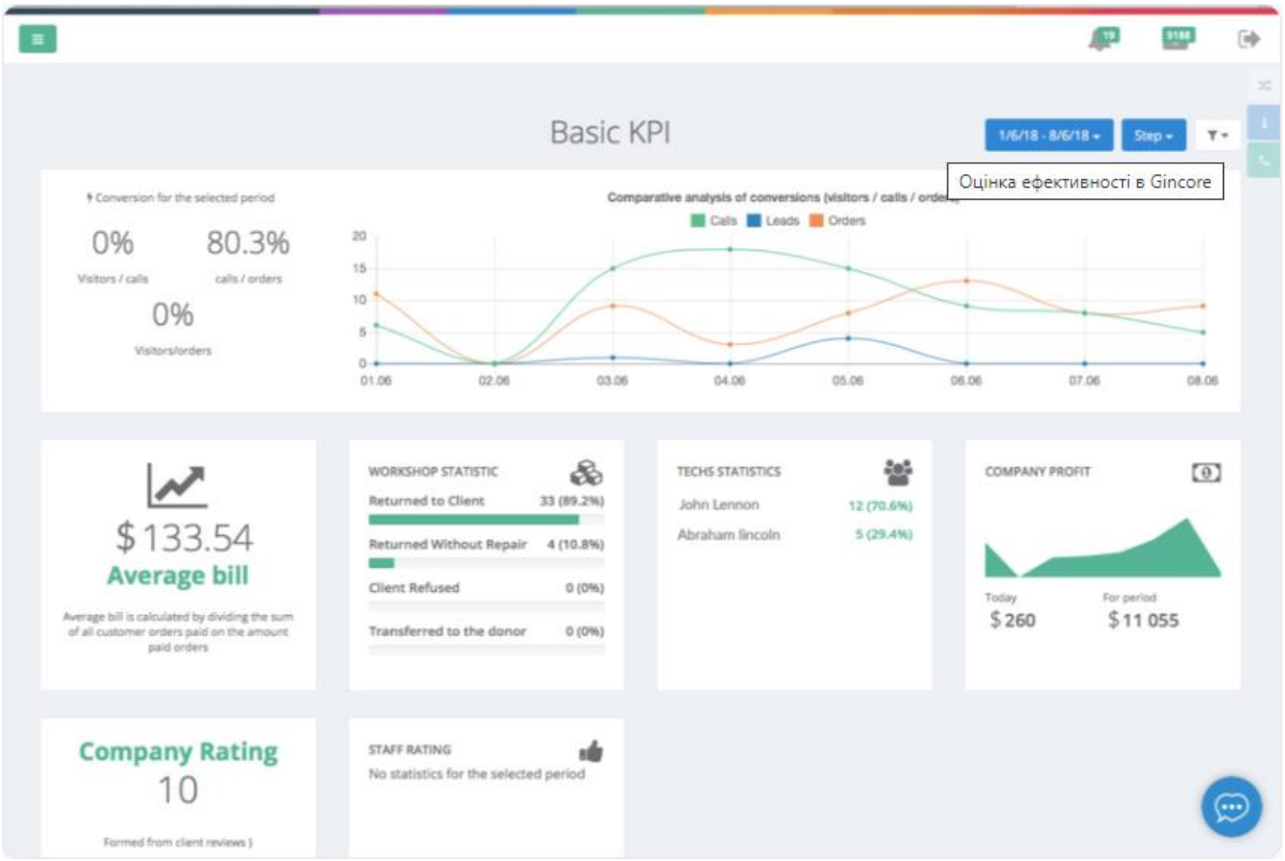


Рисунок 2.2 – Скріншот головної сторінки системи «Gincore» [3]

1.2.3 Система «USU Сервісний центр»

USU Сервісний центр – оптимізує виробничі процеси та комунікації як всередині компанії, так і з клієнтами, постачальниками та підрядниками. Завдяки цій системі значно скорочуються витрати часу співробітників на надання послуг і виконання робіт, забезпечується суворий порядок збереження інформації та товарно-матеріальних цінностей, що підвищує продуктивність персоналу та мінімізує витрати на матеріали, що використовуються, що в кінцевому результаті призводить до

зростання прибутків [4].

Облік сервісного центру містить кілька тематичних розділів для здійснення різноманітних операцій у єдиному контексті надання послуг з ремонту. Наприклад, у розділі «Наряди-замовлення» співробітники можуть створювати заявки, виконуючи нескладні процедури їх заповнення, які займають мінімум часу. Потрібно лише обрати відповідну опцію з випадаючого переліку, і заявка готова. Тут також буде вказана вартість робіт з урахуванням витратних матеріалів, вартість яких зазначена у прайс-листах, які діють на момент прийняття заявки. Система також контролює терміни виконання замовлення і повідомляє про можливі затримки.

Основні функції системи USU Сервісний центр:

- включає в себе загальну базу даних клієнтів, співробітників та постачальників, яка містить необхідну для роботи інформацію: найменування, контакти, адреси, реквізити тощо;
- система оптимізації автоматично контролює кожне замовлення на різних етапах виконання та оплати, призначаючи відповідальних осіб на кожному етапі робочого процесу;
- регулює затовареність складу, виявляючи неліквідні товари та рідко використовувані запчастини, і пропонує раціональну вартість для їх швидкого продажу;
- додає до замовлення окремі файли, що можуть бути важливими під час здачі замовлення, оскільки містять інформацію про його початковий стан;
- фотографує кожну річ, що здається в ремонт, і прикріплює фото до заявки, так що готовий бланк вже містить це зображення;
- обліковує не лише ремонтні роботи, але й реалізує супутні товари та необхідні для замовника витратні матеріали;
- розраховує відрядну заробітну плату співробітникам на основі трудового контракту, обсягів та якості виконаних робіт, а також робочого часу;
- веде повний фінансовий облік: контролює витрати, аналізує їх, розподіляє платежі та створює аналітичні звіти;

- надає керівництву широкий набір звітів, що дозволяють аналізувати діяльність компанії та оцінювати її стан за періодами;
- сумісна з АТС, що дозволяє миттєво визначити абонента і швидко надати необхідну інформацію, не змушуючи його чекати;
- інтегрується з корпоративним сайтом, завантажуючи туди інформацію про статус замовлення, кошторис вартості робіт тощо, для інформування клієнта;
- друкує фіскальний чек з діалогового вікна при оплаті замовлення або реалізації запчастин, має функцію повторного друку фіскального чека;
- забезпечує точність розрахунків, використовуючи затверджені методики, нормативно-правові положення та законодавчі акти;
- визначає запчастини, які найчастіше реалізуються, що дозволяє завчасно забезпечити їх наявність та придбати за вигідною ціною;
- налаштовує калькуляцію по всіх послугах та роботах, автоматично списуючи витратні матеріали з балансу та роблячи відповідні позначки у звітах.

Скріншот головної сторінки системи наведений на рисунку 1.3.

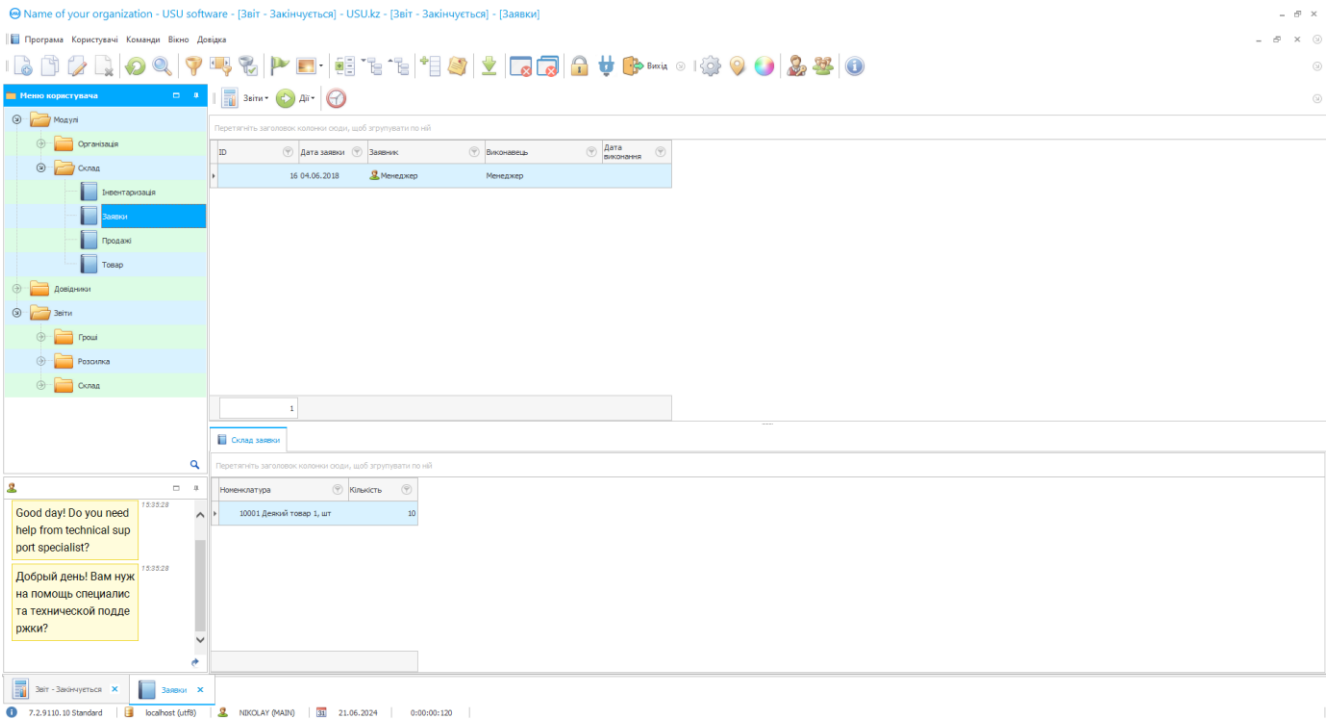


Рисунок 1.3 – Скріншот головної сторінки системи «USU Сервісний центр» [4]

1.2.4. Порівняння аналогів

Після аналізу систем, розглянутих у пунктах 1.2.1. – 1.2.3., була створена таблиця порівняння систем-аналогів (таблиця 1.1), де проводилася оцінка за такими критеріями:

- облік звернень;
- журнал ремонту;
- облік гарантійних талонів;
- клієнтська база;
- розмежування прав користувачів;
- додавання планового технічного обслуговування;
- доступ через хмару;
- перехід з іншої облікової системи;
- інтеграція з іншими сервісами;
- простота використання;
- вартість системи.

Таблиця 1.1 – Результати порівняння аналогів (таблиця виконана самостійно)

Система	Alldevice	Gincore	USU Сервісний центр
Облік звернень	так	так	так
Журнал ремонту	так	так	так
Облік гарантійних талонів	так	ні	ні
Клієнтська база	так	так	так
Розмежування прав користувачів	так	так	так
Додавання планового технічного обслуговування	так	ні	ні
Доступ через хмару	так	так	ні
Перехід з іншої облікової системи	ні	ні	так
Інтеграція з іншими сервісами	ні	так	ні
Простота використання	Складна	Складна	Складна
Вартість	$1200	$200/міс.	$40/міс.+сховані доплати

З наведеної таблиці видно, що кожна з наведених систем-аналогів є занадто

дорогою для впровадження у сервісному центрі.

Всі вищенаведені інформаційні системи мають широкий функціонал та можливості, які перевищують вимоги до розроблюваної системи. Через це вони вимагають витрати на реалізацію функцій, які не будуть використовуватись та підвищують вимоги до апаратного та програмного забезпечення. Також встановлення готової інформаційної системи робить підприємство залежним від конкретного виробника, що збільшує витрати на обслуговування.

1.3 Постановка задачі

В роботі необхідно спроектувати та реалізувати програмну систему для підтримки обліку робіт з ремонту та обслуговування комп’ютерної техніки у сервісному центрі з використанням таких технологій:

- мова програмування: JavaScript;
- нереляційна база даних: Mongo DB;
- система управління базами даних: Mongo DB;
- фреймворк для інтерфейсу користувача: React.js;
- платформа для серверної частини: Node.js;
- середовище розробки: Microsoft Visual Studio Code.

Основні функції системи:

- автоматизація обліку робіт з ремонту та обслуговування комп’ютерної техніки;
- збереження та управління інформацією про стан кожного виробу в конкретний момент;
- забезпечення зворотного зв'язку між співробітниками та клієнтами;
- мінімізація ручної роботи та зменшення ризику помилок;
- збереження та використання накопиченого досвіду для навчання нових співробітників.

Ця система має на меті надати зручний інструмент для ефективного управління ремонтними процесами та підвищити загальну ефективність роботи сервісного центру. Вона забезпечує можливість точного обліку звернень і ремонтів, що

сприяє швидкому реагуванню на клієнтські потреби. Ведення журналів ремонту і гарантійних талонів дозволяє стежити за історією обслуговування і забезпечує повноту інформації. Керування клієнтською базою сприяє покращенню обслуговування клієнтів і збереженню важливої інформації про них.

Розмежування прав користувачів і можливість додавання планового обслуговування дозволяють регулювати доступ до функціоналу системи і планувати періодичне обслуговування. Доступ через хмару забезпечує зручний і безпечний доступ до даних з будь-якого місця. Можливість переходу з іншої облікової системи і інтеграція з іншими сервісами спрощує впровадження системи і розширює її функціональні можливості.

Простота використання є ключовою для швидкого навчання персоналу та мінімізації часу на адаптацію до нової системи. Водночас, оптимізація вартості системи дозволяє досягти більшої ефективності за доступні кошти, що робить її вигідним інвестиційним рішенням для сервісних центрів.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Функції програмної системи

Програмна система підтримки обліку робіт з ремонту та обслуговування комп'ютерної техніки у сервісному центрі має широкий спектр функціональності, спрямований на оптимізацію та покращення робочих процесів. Основні функції продукту включають:

- облік клієнтів та їх техніки (збереження інформації про клієнтів та їх замовлення, ведення детальної інвентаризації техніки, що знаходиться на обслуговуванні);
- розподіл замовлень між інженерами (автоматизована система розподілу завдань між інженерами, враховуючи їхні навички, досвід та поточне навантаження);
- облік ремонтів та обслуговування (реєстрація та ведення журналу ремонтів, включаючи вартість послуг, використані запасні частини та час виконання);
- аналіз даних та статистика (генерація звітів та статистичних даних щодо роботи сервісного центру, включаючи кількість замовлень, обсяг робіт, ефективність інженерів тощо);
- управління запасними частинами (моніторинг та управління запасними частинами та інвентарем, включаючи замовлення нових деталей);
- інтеграція зовнішніми системами (можливість інтеграції з іншими внутрішніми та зовнішніми системами, такими як система управління складом чи онлайн-платформа для спіл-кування з клієнтами);
- забезпечення безпеки даних (захист конфіденційної інформації клієнтів та даних про їх техніку за допомогою встановлення прав доступу та шифрування).

Ці функції спрямовані на поліпшення продуктивності та якості обслуговування клієнтів, а також на оптимізацію управління ресурсами та процесами в сервісному центрі.

2.2 Загальні обмеження

Попри широкий функціонал та перспективи, інформаційна система обліку робіт з ремонту та обслуговування комп'ютерної техніки має певні обмеження, які варто врахувати при її розробці та впровадженні:

- технічні обмеження (використання обмежених ресурсів, таких як обсяг пам'яті та потужність обчислювальних пристроїв, може обмежувати об'єм та швидкість обробки даних);
- фінансові обмеження (бюджетні обмеження можуть вплинути на обсяг функціоналу та обмежити можливості розробки та підтримки системи);
- потреби користувачів (різні користувачі можуть мати різні вимоги до функціоналу системи, що може бути обмеженням при розробці універсального рішення);
- безпека (забезпечення безпеки даних та захисту від несанкціонованого доступу може бути обмеженням з точки зору технічних можливостей та фінансових ресурсів);
- сумісність (забезпечення сумісності із існуючими системами та технологічними платформами може обмежувати вибір технологій та архітектурних рішень);
- часові обмеження (обмежені терміни розробки та впровадження можуть вплинути на обсяг функціоналу та якість реалізації системи);
- масштабованість (недостатня масштабованість системи може обмежити її здатність працювати з великим обсягом даних або високим навантаженням);
- легкість використання (складність інтерфейсу та недостатня інтуїтивність користування можуть стати обмеженням для прийняття системою користувачами).

Урахування цих обмежень під час розробки та впровадження системи допоможе досягти оптимального балансу між функціональністю, ефективністю та вартістю проекту.

2.3 Вимоги до програмної системи

Необхідно спроектувати програмну систему підтримки обліку робіт з ремонту та обслуговування комп'ютерної техніки у сервісному центрі та створити інформаційну систему з автоматизації рівномірного розподілу замовлень між інженерами з урахуванням їх досвіду, специфіки техніки, завантаження, що буде забезпечувати наступні функції:

а) система повинна відображати дані:
   1) безпосередньо про основні поняття: замовників, інженерів, техніки;
   2) про пов’язані поняття: інформацію про техніку та їх власників, інформацію про техніку та виду несправності;

б) система повинна підтримувати арифметичну обробку даних у вигляді обчислювальних полів: стосовно загальної кількості замовників; загальної кількості відремонтованої техніки; вартості ремонту техніки;

в) система повинна підтримувати сортування, пошук та фільтрацію даних:
   1) сортувати замовників за ПІБ, модель техніки за назвою, ПІБ інженера;
   2) здійснювати пошук інформації про замовника за його повним або частковим ПІБ, про модель техніки за її повною або частковою назвою;
   3) здійснювати фільтрацію інформації по ремонтам техніки за датами, по інженерам;

г) система повинна підтримувати додавання нових даних замовників, інженерів, моделей техніки, вартості ремонту;

д) система повинна підтримувати можливості редагування інформації про замовників, інженерів, моделей техніки, вартості ремонту;

е) система повинна підтримувати можливості вилучення інформації про замовників, інженерів, моделей техніки, вартості ремонту з підтримкою режиму підтвердження користувачем видалення інформації про поточний об’єкт;

ж) система повинна підтримувати виконання наступних часто виникаючих запитів до БД:
   1) отримати статистику найбільш популярних типів послуг (назва послуги,

кількість послуг, загальна сума послуг);

2) отримати статистику гарантійного ремонту техніки (кількість, тип техніки, виробник);

3) отримати статистику ПІБ найефективнішого інженера (ПІБ інженера, кількість ремонтів, загальна сума ремонту);

4) отримати статистику ПІБ найчастішого клієнта (ПІБ клієнта, кількість звернень, загальна сума сплати за ремонт техніки);

з) система повинна підтримувати підготовку та друк наступних звітів:

1) квитанція приймання техніки у ремонт (ПІБ клієнта, ПІБ виконавця, дата приймання, номер квитанції, назва техніки, її кількість);

2) статус ремонту (ПІБ інженера, тип техніки, виробник, модель техніки, рік виробництва);

і) система повинна реалізовувати наступні задачі автоматизації:

1) збереження інформації стосовно циклу ремонту та обслуговування техніки, що забезпечує сервісний центр;

2) автоматичний рівномірний розподіл замовлень між інженерами з урахуванням їх досвіду, специфіки техніки, завантаження.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Концептуальне моделювання предметної області

Діаграма варіантів використання відіграє основну роль у моделюванні поведінки системи. Кожна така діаграма показує безліч варіантів використання, акторів та відносини між ними [5].

З предметної області, слід виділити трьох акторів (користувачів):

- керівник, який здійснює контроль та управління співробітниками, а також займається формуванням статистичної звітності;
- менеджер/адміністратор, який відповідає за взаємодію з клієнтами, реєстрацію та ведення заявок, роботу з постачальниками;
- інженер, відповідальний за діагностику та ремонт техніки.

Опис функціонального призначення програмної системи, що створюється наведено на рисунку 3.1.

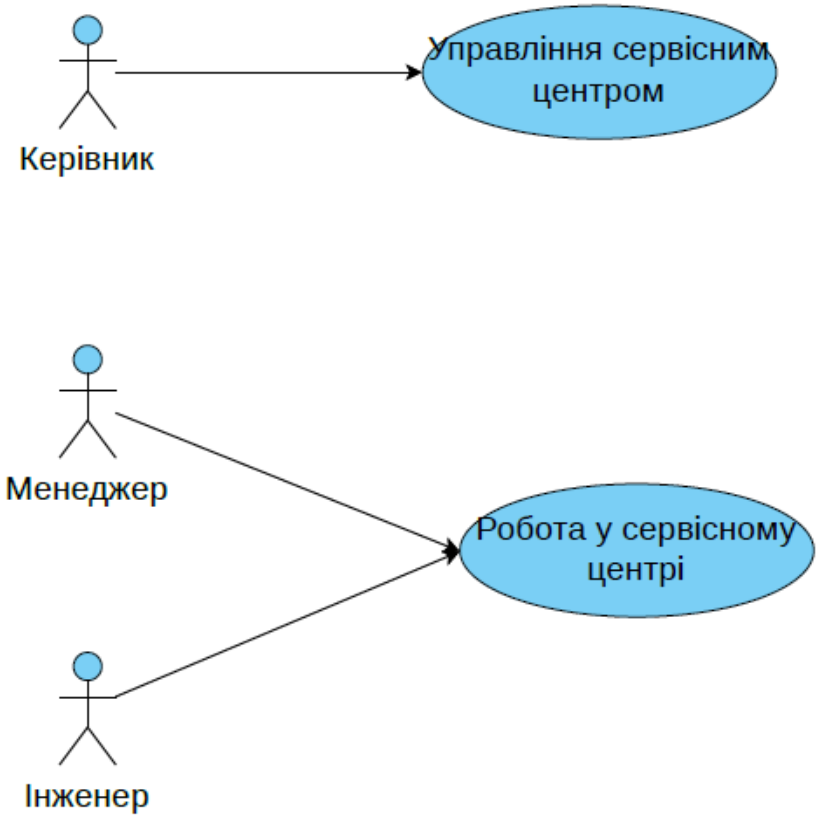


Рисунок 3.1 – Загальна USE-CASE діаграма

(рисунок виконаний самостійно)

З основних функцій користувачів виділяємо такі варіанти використання (рисунок 3.2):

- реєстрація замовлень;
- додавання нових клієнтів;
- додавання інформації про пристрій;
- замовлення комплектуючих;
- додавання інформації про комплектуючі;
- додавання інформації про постачальників;
- перегляд залишків комплектуючих на складах;
- оформлення видаткових/прибуткових накладних;
- зміна статусу заявки;
- додавання інформації про діагностику пристрою;
- складання заявки на комплектуючі;
- додавання інформації про ремонт пристрою;
- формування статистичної звітності;
- призначення працівників.

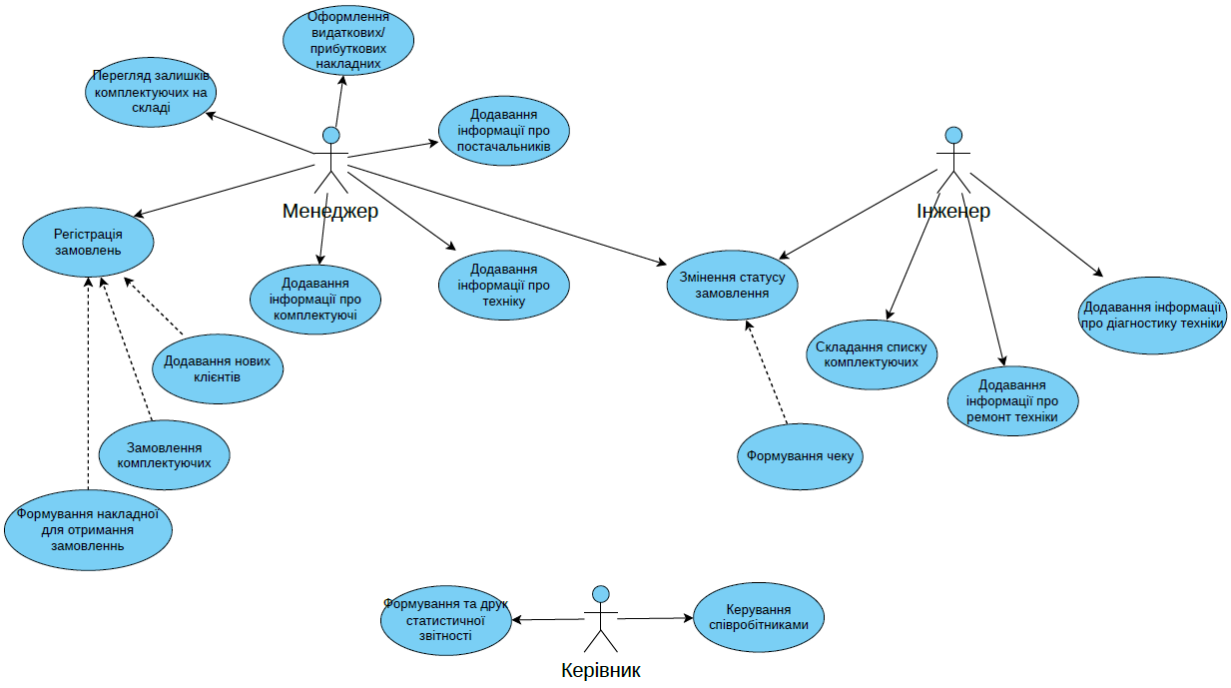


Рисунок 3.2 – Детальна USE-CASE діаграма для опису інформаційних потреб (рисунок виконаний самостійно)

На головній діаграмі прецедентів показано, за що відповідає кожен користувач (актор), якими функціями володіє.

Після визначення категорій користувачів та їх функцій можна перейти в описи інтерфейсу користувача.

Діаграма класів – діаграма, що показує безліч класів, інтерфейсів, кооперацій системи та відносин між ними [6].

Відносини між класами потрібні у тому, щоб зрозуміти як пов'язані класи між собою. Найчастіше застосовується, коли один клас використовує інший як аргумент.

Опис концептів програмної області, їх властивостей та зв'язків між ними зобразимо у вигляді загальної діаграми класів представлено на рисунку 3.3.

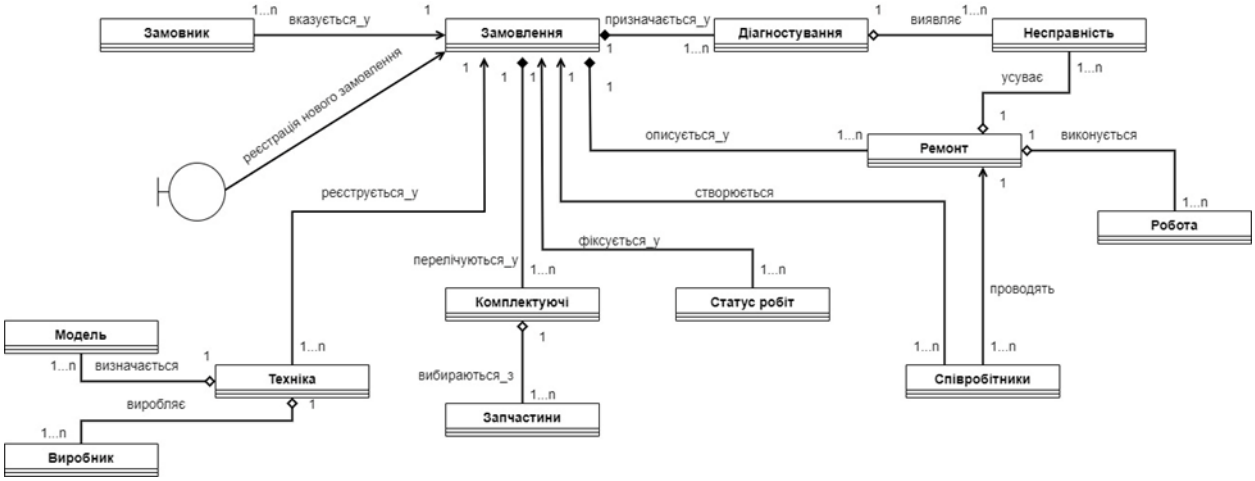


Рисунок 3.3 – Загальна діаграма класів (рисунок виконаний самостійно)

На структурній схемі з автоматизації сервісного центру видно, що довкілля інформаційної системи становлять (рисунок 3.4):

- співробітники (до цієї категорії включені всі співробітники сервісного центру: адміністратор/менеджер, інженер, керівник. Адміністратор/ менеджер приймає замовлення від клієнтів, реєструє заявку з усіма даними про клієнта та техніку, що приймається в ремонт, виставляє рахунок на оплату, веде переговори з клієнтами);

- замовники (замовники приносять у сервіс непрацюючу техніку, залишають заявку на ремонт техніку, оплачують ремонтні роботи та отримують вже відремонтовані пристрої);
- техніка (у категорії техніка знаходяться всі пристрої, які постачаються на діагностику та ремонт у сервісний центр);
- комплектуючі (до категорії комплектуючі відносяться всі запчастини, які зберігаються на складі, замовляються у постачальників і використовуються для ремонту техніки, що надійшло в сервісний центр);
- склад (на складі зберігаються комплектуючі, які можуть знадобитися для ремонту техніки. Усі комплектуючі надходять складу від постачальників);
- постачальники (постачальники поставляють на склад необхідні комплектуючі згідно з заявкою, що формується адміністратором/ менеджером сервісного центру).

Рисунок 3.4 – Структурна схема для задачі автоматизації
(рисунок виконаний самостійно)

Діяльність сервісного центру починається з реєстрації заявки клієнта, для чого співробітник сервісного центру (менеджер/адміністратор) фіксує всі дані про

клієнта та пристрій, який він приніс у ремонт. Після реєстрації заявки вона разом із технікою направляється до інженера з ремонту техніка. Він проводить діагностику пристрою, фіксуючи результати у діагностичну картку. У випадку, якщо техніка не підлягає ремонту, інженер повертає його назад у сервіс. Якщо ремонт техніки можливий, інженер складає список комплектуючих, необхідних для ремонту, а потім направляє цей список менеджеру.

Менеджер згідно зі списком, складеним інженером, робить заявку на склад для перевірки наявності запчастин на складі. Якщо необхідні запчастини є на складі, вони направляються до сервісного центру, якщо запчастини відсутні – менеджер оформляє замовлення на комплектуючі постачальникам, оплачує рахунок, і після цього всі комплектуючі надходять на склад, звідки направляються до сервісного центру для ремонту пристрою.

Після отримання всіх необхідних комплектуючих інженер проводить ремонт техніки, після чого передає його назад у сервіс. Адміністратор/менеджер отримує від інженера відремонтовану техніку та перелік усіх робіт. Після чого складається підсумкова квитанція з ремонту техніки та клієнту надається рахунок на ремонт.

Замовник оплачує рахунок на ремонт техніки та забирає відремонтований пристрій із сервісного центру.

3.2 Архітектура системи

Архітектура системи побудована на сучасних технологіях, що забезпечують високу продуктивність та масштабованість системи. Мова програмування JavaScript використовується для написання як серверної, так і клієнтської частини програми. На серверній частині застосовується платформа Node.js, яка дозволяє створювати швидкі та ефективні веб-сервери. Для клієнтської частини використовується фреймворк React.js, який забезпечує динамічний та інтерактивний користувацький інтерфейс.

База даних MongoDB обрана для збереження даних через її здатність ефективно працювати з великими обсягами даних та забезпечувати гнучкість у зберіганні

інформації. MongoDB, як нереляційна база даних, дозволяє легко масштабувати систему та адаптувати її до різних вимог.

Середовище розробки Microsoft Visual Studio Code забезпечує зручні інструменти для програмування, дебагінгу та тестування, що значно спрощує процес розробки та підтримки системи.

На діаграмі розгортання представлена взаємодія між основними компонентами системи, що забезпечують облік робіт з ремонту та обслуговування комп'ютерної техніки в сервісному центрі (рисунок 3.5) [7].

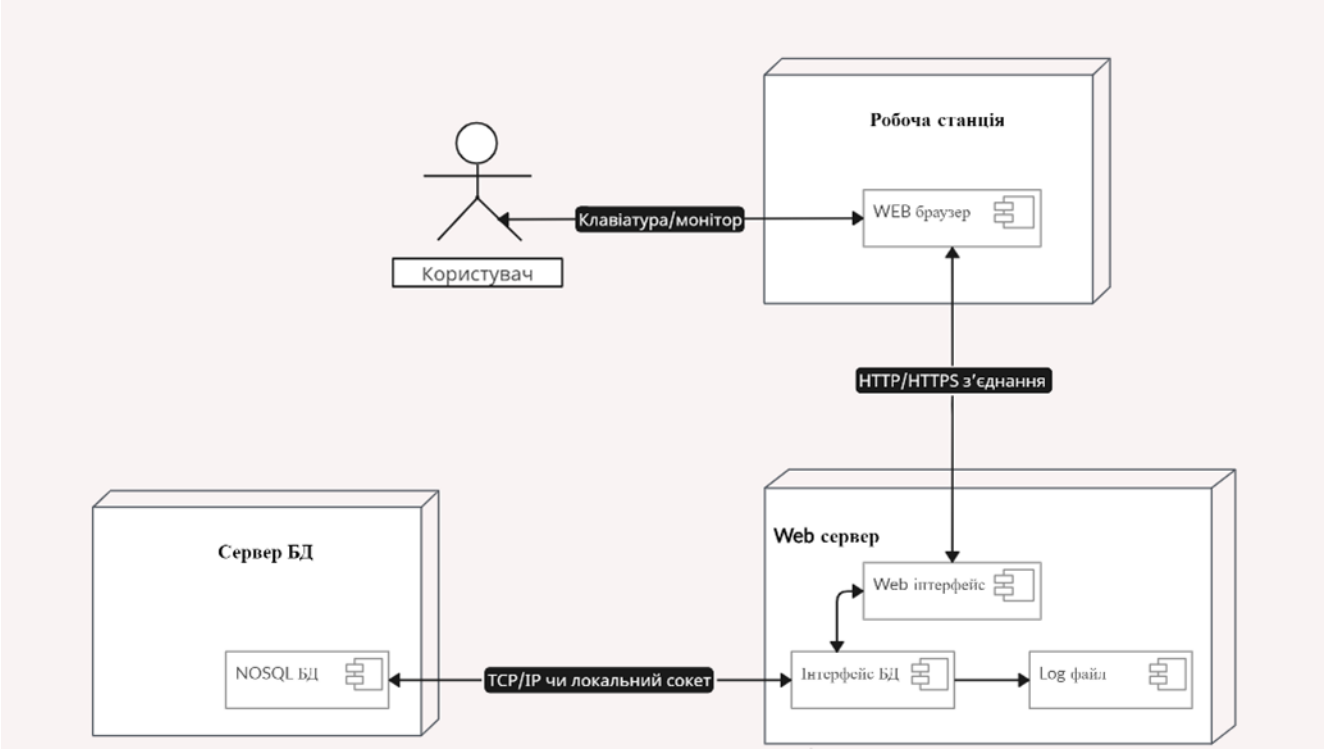


Рисунок 3.5 – Діаграма розгортання системи
(рисунок виконаний самостійно)

Взаємодія користувача із системою відбувається через робоче місце, яке включає в себе веб-браузер. Користувач через веб-браузер надсилає HTTP/HTTPS запити до веб-сервера. Веб-сервер, у свою чергу, складається з декількох компонентів: шару презентації, інтерфейсу бази даних та файлу логів. Шар презентації відповідає за взаємодію з користувачем, надаючи йому веб-інтерфейс для роботи з системою. Інтерфейс бази даних забезпечує зв'язок між веб-сервером та сервером бази даних, де зберігаються всі необхідні дані для роботи системи.

Сервер бази даних містить MongoDB, яка використовується для збереження даних про ремонти, клієнтів, техніку та інші пов'язані дані. Взаємодія між веб-сервером та сервером бази даних відбувається через TCP/IP або локальні сокети, забезпечуючи надійний і швидкий обмін даними. Лог-файл на веб-сервері зберігає всі події та операції, що відбуваються в системі, що дозволяє відстежувати дії користувачів і забезпечує додатковий рівень безпеки та контролю.

Система має кілька ключових функцій, які спрямовані на автоматизацію обліку робіт з ремонту та обслуговування комп'ютерної техніки. Вона дозволяє зберігати та управляти інформацією про стан кожного виробу в будь-який момент часу, що забезпечує прозорість та контроль за процесом ремонту. Зворотний зв'язок між співробітниками та клієнтами є важливим аспектом, який дозволяє оперативно реагувати на запити та побажання клієнтів, що підвищує рівень обслуговування.

Крім того, система мінімізує ручну роботу та зменшує ризик помилок, автоматизуючи багато рутинних завдань. Це дозволяє співробітникам сервісного центру зосередитися на більш складних завданнях та підвищує їх продуктивність. Також система зберігає накопичений досвід і використовує його для навчання нових співробітників, що сприяє підвищенню їх кваліфікації та якості обслуговування.

Загальна архітектура системи та діаграма розгортання демонструють, як сучасні технології можуть бути ефективно застосовані для створення системи обліку робіт з ремонту та обслуговування комп'ютерної техніки. Ця система забезпечує надійний інструмент для управління процесами ремонту, підвищує ефективність роботи сервісного центру та забезпечує високий рівень обслуговування клієнтів.

3.3 Побудова ER-діаграми

На даному етапі проектування БД доцільно за основу взяти загальну діаграму класів з пункту 4.1. Була розроблена ER-діаграма за нотацією Баркера (рисунок 3.6) [8].

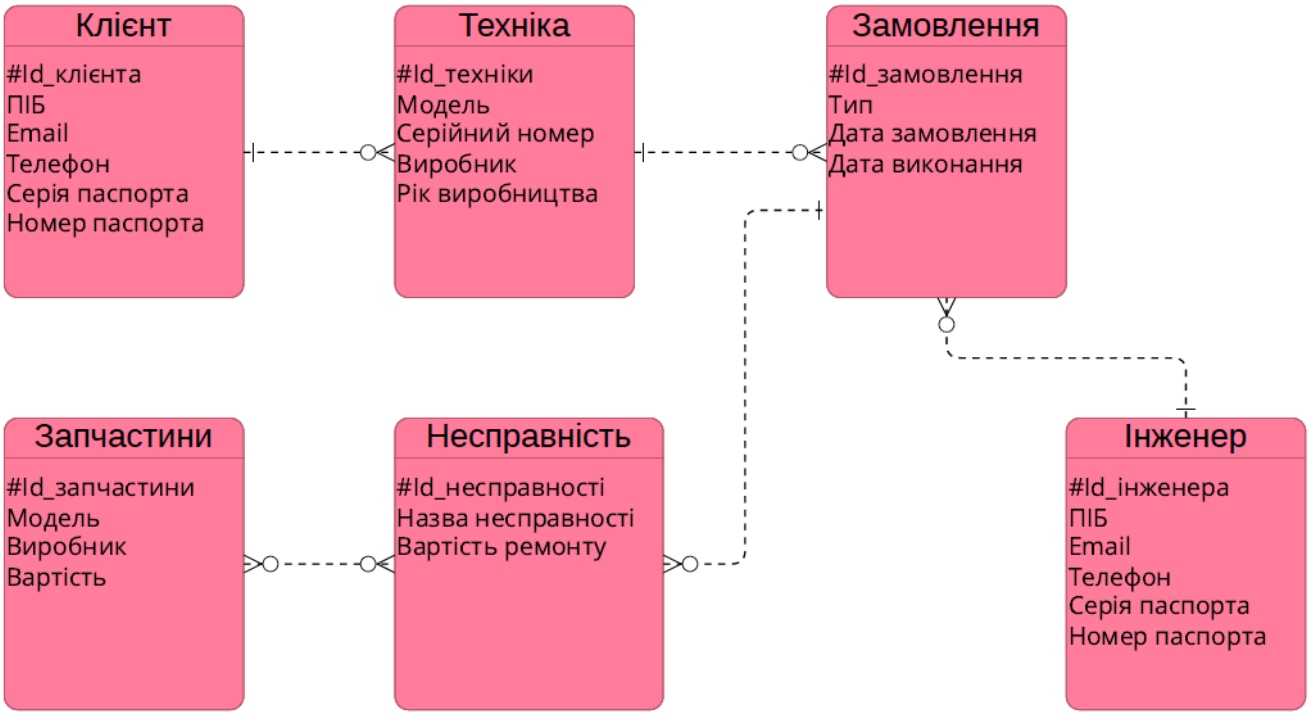


Рисунок 3.6 – ER-діаграма за нотацією Баркера

(рисунок виконаний самостійно)

3.4 Вибір та побудова логічної моделі бази даних на основі ER-діаграми

Нереляційна база даних (NoSQL) – це колекція даних, що зберігається в гнучкому форматі, який не обмежується таблицями. Нереляційна база даних складається з документів, колекцій або інших структур, які містять інформацію про певний набір об'єктів чи подій, пов'язаних між собою [9].

У нереляційній базі даних кожен документ (або запис) може містити різні атрибути та вкладені структури, що дозволяє зберігати більш складні та гнучкі дані. Документи можуть бути організовані в колекції, що нагадує таблиці в реляційних базах даних, але з більшими можливостями для варіацій у структурі даних.

Нереляційні бази даних зазвичай використовуються для зберігання великих обсягів даних з гнучкою структурою, що часто змінюється, або для даних, які не підходять для традиційних реляційних баз. Такий підхід дозволяє здійснювати швидкий пошук та фільтрацію інформації завдяки індексам та іншим механізмам оптимізації [10].

На даталогічному етапі проектування бази даних сервісного центру важливо

розробити чіткий алгоритм реалізації запитів, які можуть виникати у базі даних. Цей алгоритм повинен бути достатньо конкретизованим під потреби конкретної предметної області і водночас містити узагальнення, які дозволяють виконувати будь-який із можливих запитів бази даних. Нижче описані основні кроки, необхідні для програмної реалізації відповідного алгоритму.

У цій моделі дані про клієнтів, техніку, замовлення, інженерів, несправності та запчастини зберігаються в одному документі, що дозволяє уникнути складних зв'язків і значно покращує продуктивність запитів шляхом зберігання пов'язаних даних разом у одному документі.

Першим кроком є ініціалізація та підключення до бази даних MongoDB. Важливо налаштувати стабільне з'єднання з базою даних, а також забезпечити обробку можливих помилок під час з'єднання.

Другим кроком є введення критеріїв для виконання запиту. Користувач вводить необхідні дані через інтерфейс користувача, наприклад, через форми на вебсторінці. Введені дані повинні бути валідаційні для забезпечення їх коректності та безпеки.

Наступним кроком є формування запиту до бази даних. Введені користувачем критерії перетворюються у формат, придатний для MongoDB запиту. Вибирається відповідна колекція (наприклад, clients, techniques, orders, repairparts, crashes, masters) і формується запит на основі отриманих критеріїв.

Після цього виконується сам запит. Система виконує запит до бази даних і отримує результати. Обробляються можливі помилки, якщо запит не вдалося виконати, і забезпечується правильне оброблення отриманих результатів.

Отримані результати обробляються та представляються користувачу. Це включає форматування та підготовку даних до відображення на інтерфейсі користувача, наприклад, у вигляді таблиці або списку.

Після цього виконується логування та аналітика. Виконані запити та їх результати логуються для подальшого аналізу та оптимізації роботи системи. Також збираються аналітичні дані для покращення обслуговування користувачів та ефективності роботи сервісного центру.

Схема реалізації запитів наведена на рисунку 3.7.

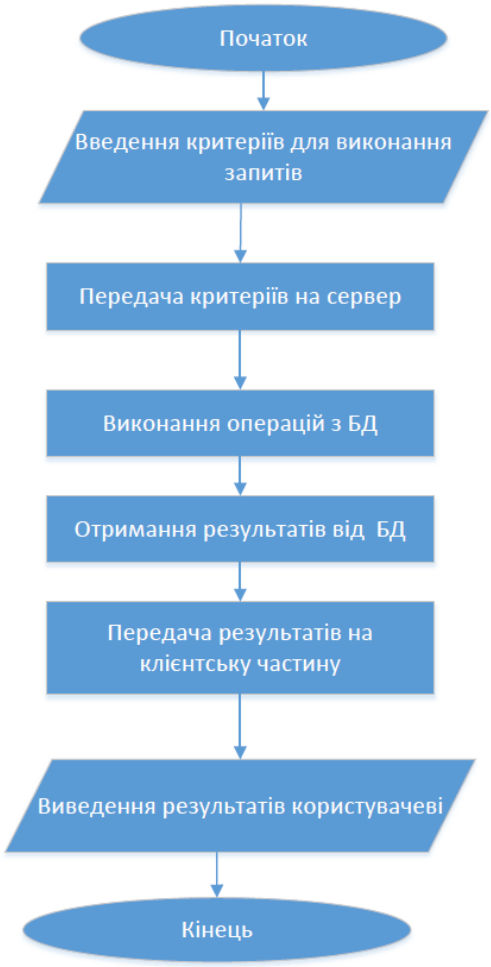


Рисунок 3.7 – Схема алгоритму реалізації запитів бази даних сервісного центру (рисунок виконано самостійно)

3.5 Проектування UI/UX системи

Нижче наведений інтерфейс користувача є прикладом системи управління ремонтом техніки (рисунок 3.8). Він надає зручні інструменти для роботи з даними клієнтів, техніки, замовлень, ремонтних частин та несправностей. Інтерфейс включає кілька вкладок, які допомагають користувачу легко переміщатися між різними розділами системи.

Структура інтерфейсу складається з наступних частин:

а) верхня панель навігації (вкладки верхньої панелі навігації дозволяють швидко переходити між основними розділами системи):

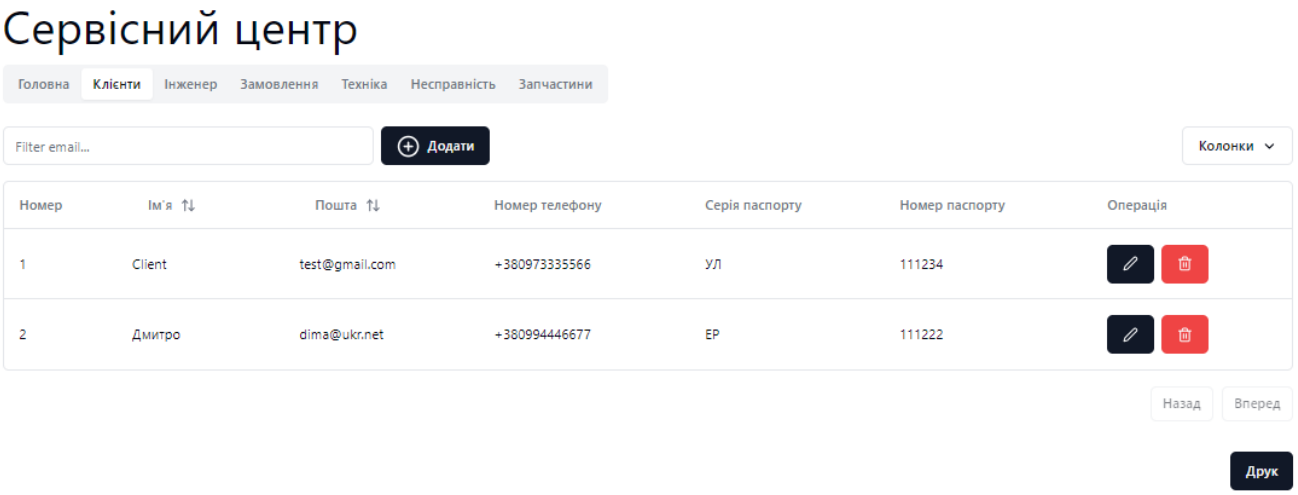


Рисунок 3.8 – Інтерфейс користувача (рисунок виконаний самостійно)

1) головна – повертає користувача на домашню сторінку, де зібрана загальна статистика по роботі з системою;
2) клієнти – розділ для управління даними клієнтів;
3) інженер – розділ для управління даними інженерів;
4) замовлення – активна вкладка, яка відображає замовлення;
5) техніка – розділ для управління даними техніки;
6) несправність – розділ для управління даними несправностей;
7) запчастини – розділ для управління даними запчастин;

б) фільтрація та пошук:

1) поле для пошуку майстра дозволяє вводити ім’я майстра для швидкого знаходження відповідних замовлень;
2) поле для фільтрації замовлень за типом допомагає швидко відфільтрувати замовлення за заданими критеріями;
3) кнопка Add дозволяє додати нове замовлення;
4) фільтр за датою дозволяє вибрати діапазон дат для відображення замовлень, які були створені або виконані у вибраний період;
5) кнопка Search активує фільтрацію за обраними критеріями;

в) таблиця з даними про замовлення:

1) унікальний ідентифікатор кожного замовлення;

2) тип замовлення, який вказує на категорію робіт;

3) дата створення замовлення;

4) дата виконання замовлення;

5) ідентифікатор техніки, пов'язаної із замовленням;

6) ім'я майстра, відповідального за виконання замовлення;

7) дії, які можна виконати з замовленням, такі як редагування та видалення;

г) додаткові функції:

1) колонки (налаштування видимих стовпців у таблиці дозволяє користувачу вибирати, які стовпці будуть відображатися);

2) пагінація (елементи управління для переходу між сторінками дозволяють перегортати сторінки з даними замовлень);

3) кнопка для друку поточного вмісту таблиці дозволяє роздрукувати перелік замовлень.

Цей інтерфейс забезпечує зручний доступ до основних функцій системи управління ремонтом техніки, дозволяючи користувачам ефективно управляти замовленнями та пов'язаними з ними даними.

3.6 Опис найцікавіших алгоритмів та методів системи

3.6.1 CRUD операції

CRUD операції є основними операціями, які виконуються з даними у будь-якій базі даних. У програмній системі підтримки обліку робіт з ремонту та обслуговування комп'ютерної техніки ці операції реалізовані з використанням технологій JavaScript, Node.js, MongoDB та React.js.

Користувач вводить дані нового замовлення через форму у веб-інтерфейсі і натискає кнопку «Add» для створення нового запису. Дані з форми передаються на сервер через HTTP POST запит. Сервер отримує запит і передає дані в метод контролера для створення нового запису. Дані перевіряються на валідність, і якщо дані

валідні, сервер створює новий документ у колекції MongoDB. Після успішного збереження сервер відправляє підтвердження назад клієнту, і клієнт отримує підтвердження та оновлює список замовлень, додаючи новий запис.

```
const newOrder = new OrderModel(orderData);
await newOrder.save();
```

Користувач відкриває сторінку зі списком замовлень або використовує функцію пошуку/фільтрації. Клієнтський додаток відправляє HTTP GET запит на сервер для отримання даних. Сервер отримує запит і звертається до відповідного методу контролера для читання даних з бази даних. Контролер взаємодіє з MongoDB для отримання потрібних документів. Отримані дані відправляються назад клієнту у вигляді JSON, і клієнтський додаток отримує дані та відображає їх у вигляді таблиці або іншого інтерфейсу.

```
const orders = await OrderModel.find(filterCriteria);
```

Користувач вибирає запис для редагування і відкриває форму редагування. Після внесення змін користувач натискає кнопку «Save». Змінені дані передаються на сервер через HTTP PUT або PATCH запит. Сервер отримує запит і передає дані в метод контролера для оновлення запису. Дані перевіряються на валідність. Якщо дані валідні, сервер оновлює відповідний документ у колекції MongoDB. Після успішного оновлення сервер відправляє підтвердження назад клієнту, і клієнт отримує підтвердження та оновлює відображення даних у списку замовлень.

```
await OrderModel.findByIdAndUpdate(orderId, updateData);
```

Користувач вибирає запис для видалення і натискає кнопку «Delete». Клієнтський додаток може відобразити діалогове вікно для підтвердження видалення. Після підтвердження дані передаються на сервер через HTTP DELETE запит. Сервер отримує запит і передає його в метод контролера для видалення запису. Контролер

взаємодіє з MongoDB для видалення відповідного документа. Після успішного видалення сервер відправляє підтвердження назад клієнту, і клієнт отримує підтвердження та видаляє запис зі списку відображених даних.

```
await OrderModel.findByIdAndDelete(orderId);
```

Для забезпечення коректності даних перед збереженням в базі використовуються валідаційні бібліотеки. Обробка помилок здійснюється через try/catch блоки, а також через централізовану систему обробки помилок на сервері.

3.6.2 Авторизація та аутентифікація користувачів

Процес авторизації та аутентифікації користувачів у системі для обліку робіт з ремонту та обслуговування комп'ютерної техніки передбачає кілька етапів. Користувач вводить свої облікові дані, сервер перевіряє ці дані, і якщо вони правильні, сервер генерує токен для подальших запитів. Цей процес забезпечує безпеку та контроль доступу до системи.

Алгоритм авторизації реалізован наступним чином:

- вхід користувача (користувач починає процес авторизації, вводячи свої облікові дані (логін та пароль) у форму на клієнтській стороні. Після цього облікові дані відправляються на сервер через захищений HTTPS POST запит. Це гарантує, що дані передаються безпечно, захищаючи їх від перехоплення);
- перевірка даних (після отримання запиту сервер передає облікові дані у відповідний метод для перевірки. Сервер використовує бібліотеку bcrypt для порівняння введеного пароля з хешованим паролем, що зберігається в базі даних MongoDB. Якщо облікові дані не збігаються, сервер повертає повідомлення про помилку автентифікації);
- генерація токену (якщо введені дані правильні, сервер генерує JWT (JSON Web Token). JWT включає інформацію про користувача та час його дії. Сервер відправляє цей токен назад клієнту, де він зберігається в localStorage

або sessionStorage. Зберігання токену на клієнтській стороні дозволяє уникнути необхідності повторної аутентифікації при кожному запиті);

- використання токену (усі подальші запити до захищених ресурсів системи включають цей токен в заголовках для аутентифікації. Кожен запит перевіряє токен на сервері, щоб підтвердити особу користувача та його права доступу. Це забезпечує безпеку та контроль доступу до даних системи, дозволяючи тільки авторизованим користувачам виконувати певні операції).

4 ОПИС ПРОГРАМИ

4.1 Загальні відомості

Для реалізації даного додатку було використано середовище розробки Microsoft Visual Studio Code та мову програмування JavaScript. Для створення візуального інтерфейсу програми було обрано фреймворк React.js, який забезпечує швидку розробку зручного та інтуїтивно зрозумілого інтерфейсу користувача.

Додаток спрямований на взаємодію з базами даних, тому для їх створення було обрано нереляційну базу даних MongoDB. Ця система управління базами даних була використана завдяки її здатності швидкої обробки інформації, легкості використання та гнучкості у роботі з документами. MongoDB дозволяє ефективно зберігати та обробляти великі обсяги даних, що робить її ідеальною для потреб сучасних додатків.

Доступ до баз даних було здійснено за допомогою платформи Node.js, яка забезпечує асинхронний ввід/вивід та високу продуктивність при роботі з великою кількістю запитів. Однією з головних переваг використання Node.js є можливість реалізації роз’єднаної моделі доступу до даних. Раніше, при створенні застосунків, традиційно використовувалась технологія доступу до джерела даних, за якої з'єднання з базою підтримується постійно. Однак постійне з'єднання може призвести до витрати системних ресурсів. Node.js дозволяє встановлювати з'єднання лише на той час, коли необхідно проводити операції з базою даних, що знижує навантаження на систему.

Таким чином, у цьому проекті були використані сучасні технології та інструменти, які забезпечують високу продуктивність, гнучкість та зручність у розробці додатку. Комбінація JavaScript, React.js, Node.js та MongoDB дозволяє створити ефективне рішення для управління даними та взаємодії з користувачем.

4.2 Розгортання проекту

Перед початком роботи над проектом необхідно підготувати середовище розробки. Для цього потрібно встановити декілька програм, зокрема:

- Node.js, який включає npm. Завантажити його можна з офіційного сайту Node.js;
- MongoDB для управління базою даних. Завантажити можна з MongoDB;
- Microsoft Visual Studio Code (VS Code) як середовище розробки. Завантажити його можна з VS Code.

Після встановлення всіх необхідних програм, необхідно отримати вихідний код проекту. Це можна зробити, клонуючи або копіюючи репозиторій проекту у папку на локальному комп'ютері. Потрібно переконатись, що структура папок проекту відповідає наступній схемі (рисунок 4.1).

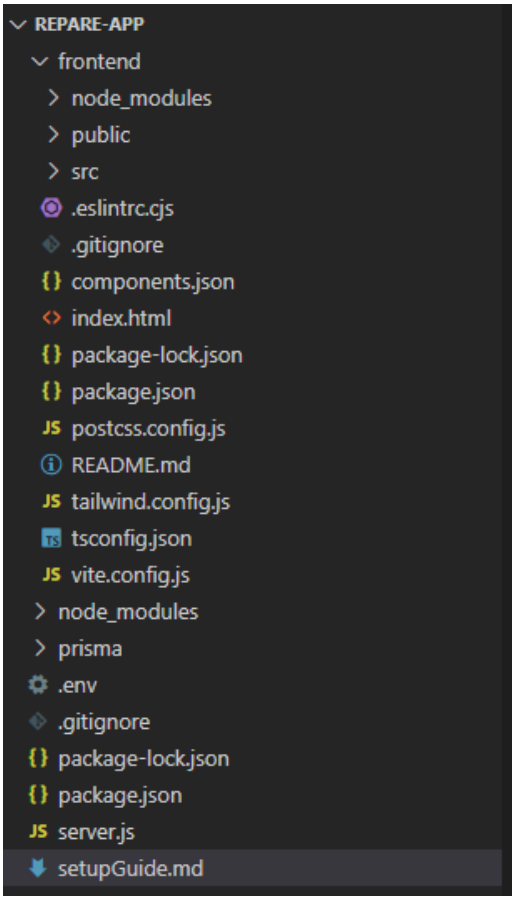


Рисунок 4.1 – Структура проекту (рисунок виконано самостійно)

Після підготовки середовища необхідно перейти до налаштування серверної частини проекту. Для цього необхідно виконати наступні кроки.

Відкрити термінал і перейти до папки проекту repare-app за допомогою команди:

```
cd repare-app
```

Встановити залежності для серверної частини, виконавши команду:

```
npm install
```

Потрібно переконатись, що MongoDB запущений локально або вказати віддалений сервер у файлі конфігурації проекту.

Запустити міграції Prisma для створення необхідних колекцій та схем у MongoDB за допомогою команди:

```
npx prisma migrate dev
```

Згенерувати Prisma клієнт для взаємодії з базою даних, виконавши команду:

```
npx prisma generate
```

Запустити бекенд сервер за допомогою команди:

```
npm run dev
```

Далі необхідно перейти до налаштування клієнтської частини проекту. Для цього виконайте наступні кроки.

У терміналі перейти до папки frontend, використовуючи команду:

```
cd frontend
```

Встановити залежності для клієнтської частини, виконавши команду:

```
npm install
```

Запустити фронтенд сервер за допомогою команди:

```
npm run dev
```

Після запуску серверів потрібно переконатись, що бекенд сервер працює на певному порту (наприклад, 3000). Необхідно відкрити браузер і перейти на адресу фронтенд сервера (зазвичай http://localhost:3000).

Дотримуючись кроків описаних вище, проект успішно розгортається та запускається на локальному комп'ютері або сервері.

4.3 Опис роботи бази даних

При реалізації фізичної моделі бази даних було створено шість таблиць. Нижче наведено команди на їх створення. Для зв'язків між таблицями використовуються відношення типу «один-до-багатьох» або «багато-до-одного». Наприклад, один клієнт може мати кілька технік, і одна техніка може мати кілька замовлень. Всі ідентифікатори (id) в таблицях генеруються автоматично і представлені у вигляді строк.

Ця фізична модель бази даних забезпечує ефективне зберігання та управління даними, необхідними для роботи сервісного центру, включаючи клієнтів, техніку, замовлення, запасні частини, несправності та майстрів.

Для реалізації цієї моделі було використано нереляційну базу даних MongoDB, яка є системою управління базами даних. Всі дані зберігаються у вигляді документів, що дозволяє легко керувати складними структурами даних і забезпечує гнучкість у розробці.

Модель клієнта (Client) зберігає інформацію про клієнтів сервісного центру. Кожен клієнт має унікальний ідентифікатор, який генерується автоматично. У записі про клієнта міститься основна інформація, включаючи ім'я, електронну пошту, телефон, серію та номер паспорта. Крім цього, для кожного клієнта зберігається дата створення запису. Клієнт може мати декілька технік, які він здає на ремонт або обслуговування, а також може мати кілька замовлень, виконаних різними майстрами.

Модель техніки (Technique) зберігає інформацію про кожну одиницю техніки, яку обслуговує сервісний центр. Кожна техніка має унікальний ідентифікатор. Запис про техніку містить дані про модель техніки, серійний номер, виробника та рік виробництва. Кожна техніка пов'язана з певним клієнтом, але цей зв'язок може бути відсутнім, якщо клієнт не вказаний. Для кожної техніки також зберігається дата створення запису. Окрім цього, техніка може мати декілька замовлень на ремонт або обслуговування.

Модель замовлення (Order) містить інформацію про всі замовлення, які обробляє сервісний центр. Кожне замовлення має унікальний ідентифікатор. У записі про замовлення зберігаються дані про тип замовлення, дату замовлення та дату виконання. Кожне замовлення пов'язане з певною технікою та майстром, але ці зв'язки можуть бути відсутніми, якщо техніка або майстер не вказані. Для кожного замовлення також зберігається дата створення запису.

Модель запасних частин (RepairPart) зберігає інформацію про всі запасні частини, які використовуються для ремонту техніки. Кожна ремонтна частина має унікальний ідентифікатор, який генерується автоматично. У записі про ремонтну частину міститься інформація про її модель та виробника. Для кожної ремонтної частини зберігається дата створення запису. Кожна ремонтна частина може бути пов'язана з кількома несправностями, які вона допомагає усунути.

Модель несправності (Crash) зберігає інформацію про всі виявлені несправності техніки. Кожна несправність має унікальний ідентифікатор. У записі про несправність міститься інформація про назву несправності та вартість її ремонту. Кожна несправність пов'язана з певною ремонтною частиною, але цей зв'язок може бути відсутнім, якщо ремонтна частина не вказана. Для кожної несправності також зберігається дата створення запису.

Модель майстра (Master) зберігає інформацію про майстрів, які працюють у сервісному центрі. Кожен майстер має унікальний ідентифікатор, який генерується автоматично. У записі про майстра міститься основна інформація, включаючи ім'я, електронну пошту, телефон, серію та номер паспорта, а також індивідуальний податковий номер (ІПН). Для кожного майстра зберігається дата створення запису. Один майстер може виконувати кілька замовлень і обслуговувати кілька клієнтів.

Таким чином, ці моделі дозволяють ефективно організувати та керувати даними, пов'язаними з клієнтами, технікою, замовленнями, ремонтними частинами та несправностями в сервісному центрі, забезпечуючи при цьому гнучкість і масштабованість системи.

Нижче наведена реалізація системи на основі нереляційної бази даних. Даний код призначений для управління даними клієнтів, технікою, замовленнями, ремонтними частинами та несправностями. Ця схема включає моделі для клієнтів, техніки, замовлень, ремонтних частин і несправностей, а також визначає зв'язки між цими моделями.

```
model Client {
  id             String      @id @default(auto()) @map("_id")
  name           String
  email          String?
  phone          String?
  passportSeries String?
  passportNumber String?
  createdAt      DateTime    @default(now())
  techniques     Technique[]
  masters        Master[]
}

model Technique {
  id              String   @id @default(auto()) @map("_id")
  model           String
  serialNumber    String?
  producer        String?
  yearManufacture String?
  clientId        String?
  createdAt       DateTime @default(now())
  client           Client?  @relation(fields: [clientId], references:
[id], onDelete: SetNull)
  orders          Order[]
}

model Order {
  id            String     @id @default(auto()) @map("_id")
  type          String
  dateOrder     DateTime?  @default(now())
  dateExecution DateTime?  @default(now())
  techniqueId   String?
  masterId      String?
  createdAt     DateTime   @default(now())
  technique    Technique? @relation(fields: [techniqueId], references:
[id], onDelete: SetNull)
  master         Master?    @relation(fields: [masterId], references:
[id], onDelete: SetNull)
}
```

Початок роботи включає ініціалізацію підключення до бази даних MongoDB. Після цього користувач вводить необхідні критерії для пошуку, наприклад, дані про клієнта або техніку. На основі введених критеріїв система формує запит і виконує його до відповідної колекції бази даних. Отримані дані обробляються та форматуються для відображення користувачу. Виконаний запит та його результати логуються для подальшого аналізу. Після завершення всіх етапів процес виконання запиту завершується.

Ця структура алгоритму дозволяє виконувати будь-який із можливих запитів бази даних, забезпечуючи гнучкість та масштабованість для майбутніх розширень і модифікацій.

4.4 Опис серверної частини

Серверна частина (Backend) реалізована за допомогою платформи Node.js. Вона відповідає за обробку запитів від клієнтської частини та взаємодію з базою даних MongoDB. Приклад програмної реалізації для таблиці «Клієнти» наведений нижче.

```
// Endpoints for Clients
app.get("/clients", async (req, res) => {
  const clients = await prisma.client.findMany();
  res.json(clients);
});

app.post("/clients", async (req, res) => {
   const { name, email, phone, passportSeries, passportNumber } =
req.body;
  const client = await prisma.client.create({
    data: { name, email, phone, passportSeries, passportNumber },
  });
  res.json(client);
});

app.put("/clients/:id", async (req, res) => {
  const { id } = req.params;
   const { name, email, phone, passportSeries, passportNumber } =
req.body;
  const client = await prisma.client.update({
    where: { id: parseInt(id, 10) },
    data: { name, email, phone, passportSeries, passportNumber },
  });
  res.json(client);
});

app.delete("/clients/:id", async (req, res) => {
```

```
  const { id } = req.params;
  await prisma.client.delete({
    where: { id: parseInt(id, 10) },
  });
  res.status(204).send();
});
```

Використовуються контролери для обробки логіки додатку, зокрема, для отримання, створення, оновлення та видалення даних.

Маршрутизатори (роути) забезпечують маршрутизацію запитів до відповідних контролерів, що дозволяє розділити логіку обробки різних типів запитів.

У проекті використовується нереляційна база даних MongoDB, що дозволяє зберігати дані в форматі JSON-подібних документів. В базі даних створені колекції для кожної сутності: клієнтів, техніки, замовлень, несправностей та запчастин. Кожна колекція має свою структуру документів з відповідними полями, наприклад, колекція клієнтів містить поля для зберігання інформації про ім'я, номер телефону, електронну пошту тощо.

Для розробки програмної системи використовується середовище Microsoft Visual Studio Code, що забезпечує зручний інтерфейс для написання та налагодження коду. Додаток може бути розгорнутий на будь-якому сервері, що підтримує Node.js, а також забезпечує доступ до бази даних MongoDB.

Програмна система підтримки обліку робіт з ремонту та обслуговування комп’ютерної техніки у сервісному центрі реалізує всі необхідні функції для ефективного управління сервісним центром, забезпечуючи зручний та інтуїтивно зрозумілий інтерфейс для користувачів, а також надійний та швидкий сервер для обробки даних.

4.5 Опис клієнтської частини

Програмна система підтримки обліку робіт з ремонту та обслуговування комп’ютерної техніки у сервісному центрі призначена для управління сервісним центром, що спеціалізується на ремонті техніки. Додаток дозволяє відстежувати клієнтів, техніку, замовлення, несправності та запчастини, а також забезпечує зручний інтерфейс для роботи з цими даними.

У клієнтській частині (Frontend) використовується фреймворк React.js для побудови інтерфейсу користувача. Клієнтська частина відповідає за відображення даних та взаємодію користувача з додатком.

Головна сторінка містить таблицю з даними про клієнтів, техніку, замовлення, несправності та запчастини. Передбачені форми для додавання нових записів у базу даних. Наприклад, форма для додавання нового клієнта включає поля для вводу ПІБ, номеру телефону, електронної пошти, а також інформації про техніку, яка здається на ремонт (рисунок 4.2).

У вкладці «Замовлення» реалізований зручний інтерфейс для управління всіма аспектами замовлень у сервісному центрі, дозволяючи легко додавати, редагувати, видаляти і фільтрувати замовлення, а також переглядати інформацію про техніку та майстрів, пов'язаних із кожним замовленням (рисунок 4.3).

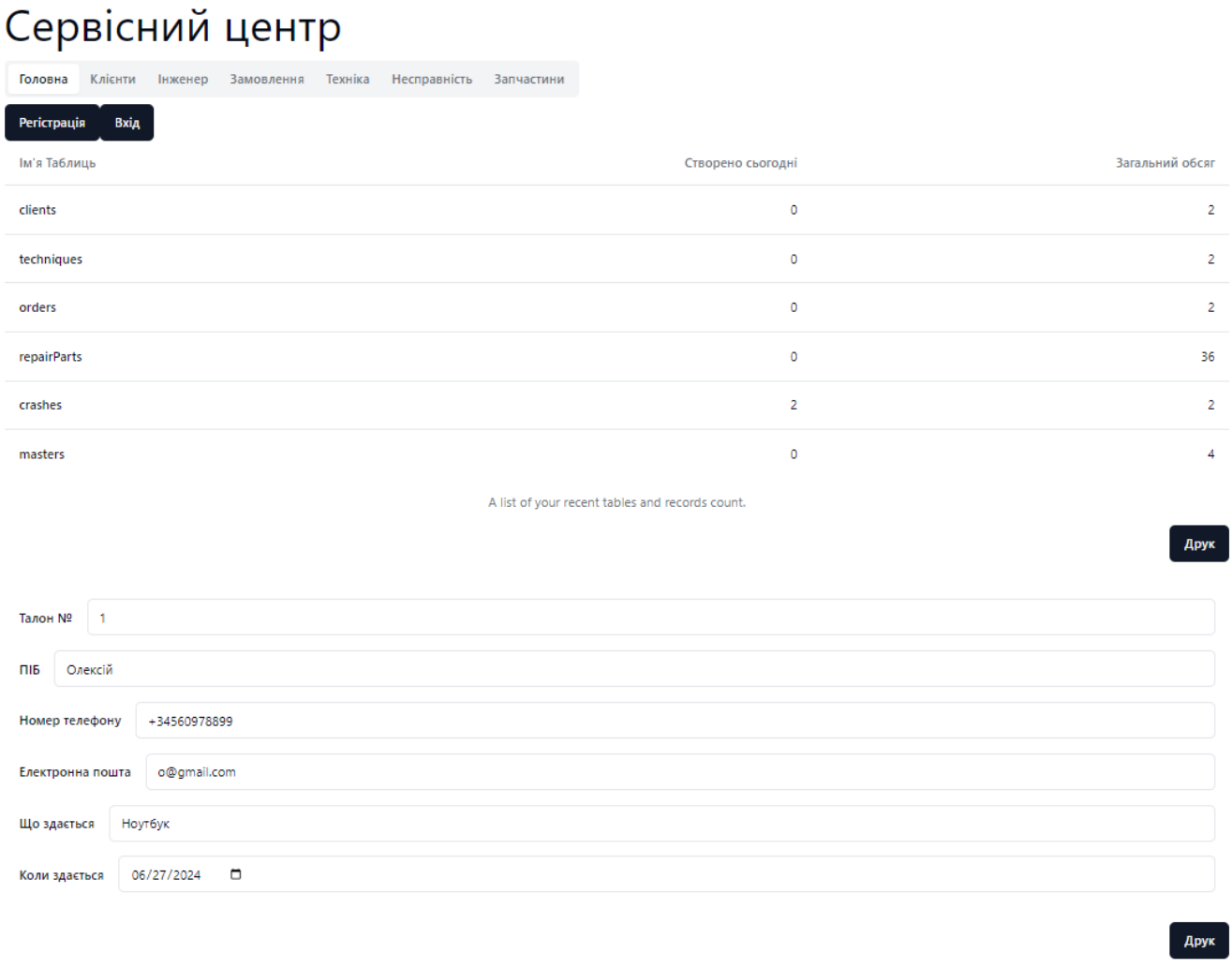


Рисунок 4.2 – Головна сторінка проекту (рисунок виконано самостійно)

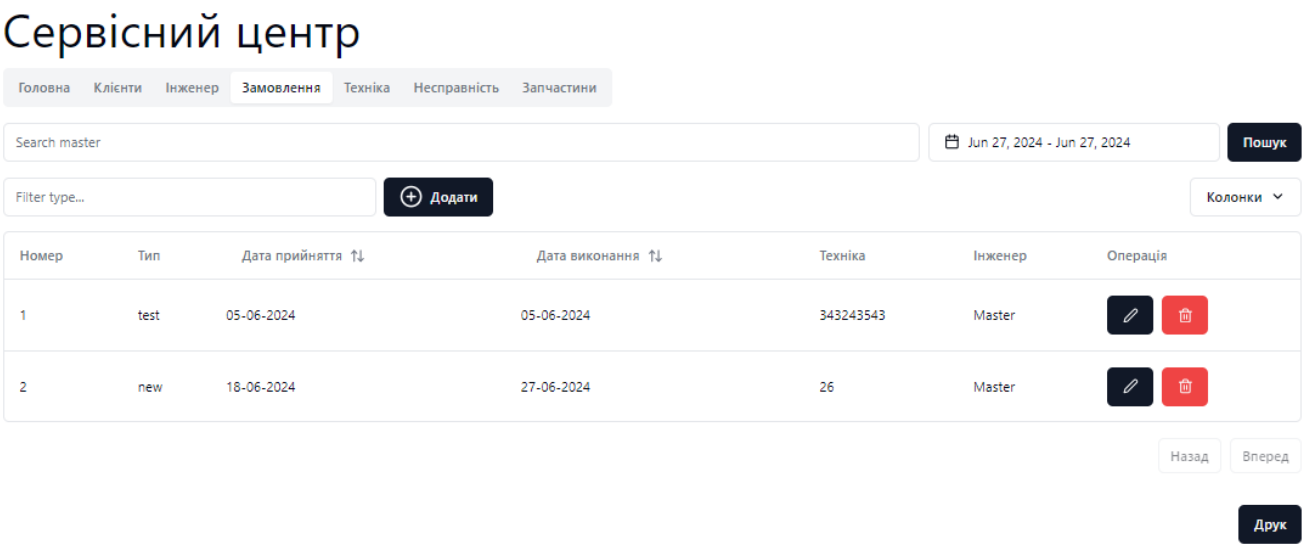


Рисунок 4.3 – Вкладка «Замовлення» (рисунок виконано самостійно)

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональне тестування – це вид тестування програмного забезпечення, який має на меті перевірку правильності виконання функцій, визначених у специфікаціях або вимогах до системи. Основна мета функціонального тестування полягає в тому, щоб переконатися, що програмне забезпечення працює відповідно до очікувань і виконує всі необхідні функції правильно.

Процес проведення функціонального тестування складається з кількох ключових етапів. Першим кроком є аналіз вимог, де тестувальники вивчають специфікації, бізнес-вимоги та технічні описи, щоб зрозуміти функціональність, яку потрібно перевірити. На цьому етапі важливо ідентифікувати всі можливі сценарії використання системи, щоб забезпечити повне покриття функціональних аспектів.

Наступним кроком є планування тестування, яке включає створення детального плану з визначенням мети, обсягу, ресурсів, розкладу та критеріїв успішності тестування. Після цього тестувальники розробляють тестові випадки, які описують конкретні кроки для перевірки кожної функції, вхідні дані та очікувані результати.

Після розробки тестових випадків підготовлюється тестове середовище, яке відтворює реальні умови роботи програмного забезпечення. Це включає налаштування необхідного обладнання, програмного забезпечення та даних для тестування.

Коли тестове середовище готове, тестувальники переходять до виконання тестових випадків. Вони послідовно виконують кожен тестовий випадок, фіксують результати та порівнюють їх з очікуваними. Якщо результати відрізняються від очікуваних, це вказує на наявність дефекту, який потрібно задокументувати.

Важливим етапом є відстеження дефектів, де всі виявлені помилки реєструються у системі відстеження дефектів і передаються розробникам для виправлення. Після виправлення дефектів тестувальники проводять повторне тестування, щоб переконатися, що помилки виправлені, і нові зміни не вплинули на інші частини системи.

Регресійне тестування виконується для перевірки, що нові зміни або виправлення не порушили існуючий функціонал. Це допомагає забезпечити стабільність і надійність програмного забезпечення.

Після завершення тестування проводиться аналіз результатів, готується звіт про тестування, який включає висновки та рекомендації. Завершальним етапом є підтримка тестової документації, щоб забезпечити її актуальність і корисність для майбутніх тестувань.

Таким чином, функціональне тестування є важливим елементом забезпечення якості програмного забезпечення, оскільки дозволяє перевірити правильність виконання всіх функцій і вчасно виявити та виправити дефекти.

5.1 Тестування операцій з БД

У зведеній таблиці наведено результати тестування основних CRUD-операцій (Create, Read, Update, Delete) для програмного забезпечення, розробленого для обліку робіт з ремонту та обслуговування комп’ютерної техніки. Тестування було проведено для перевірки коректності функціонування кожної з цих операцій. Нижче, у таблиці 5.1 наведено детальний опис кожного тестового випадку.

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

Тестовий випадок	Опис	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
TC01: Створення запису	Перевірка створення нового запису в базі даних	Данні про роботу: Назва, Опис, Дата	Запис успішно створено, відображається у списку записів	Запис створено, відображається у списку	Пройдено
TC02: Читання запису	Перевірка читання існуючого запису з бази даних	Ідентифікатор запису	Відображення всіх даних відповідного запису	Дані запису відображені	Пройдено
TC03: Оновлення запису	Перевірка оновлення існуючого запису в базі даних	Ідентифікатор запису, Нові дані	Запис успішно оновлено, зміни збережені	Запис оновлено	Пройдено

Кінець табл. 5.1

Тестовий випадок	Опис	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
ТС04: Видалення запису	Перевірка видалення існуючого запису з бази даних	Ідентифікатор запису	Запис успішно видалено, більше не відображається у списку записів	Запис видалено	Пройдено
ТС05: Читання неіснуючого запису	Перевірка читання запису з неіснуючим ідентифікатором	Ідентифікатор запису: неіснуючий	Повідомлення про помилку: «Запис не знайдено»	Повідомлення про помилку	Пройдено
ТС06: Оновлення неіснуючого запису	Перевірка оновлення запису з неіснуючим ідентифікатором	Ідентифікатор запису: неіснуючий, Нові дані	Повідомлення про помилку: «Запис не знайдено»	Повідомлення про помилку	Пройдено
ТС07: Видалення неіснуючого запису	Перевірка видалення запису з неіснуючим ідентифікатором	Ідентифікатор запису: неіснуючий	Повідомлення про помилку: «Запис не знайдено»	Повідомлення про помилку	Пройдено
ТС08: Масове створення записів	Перевірка масового створення записів	Множинні дані про роботи	Усі записи успішно створено, відображаються у списку записів	Усі записи створено	Пройдено
ТС09: Перевірка відновлення видаленого запису	Перевірка можливості відновлення раніше видаленого запису	Ідентифікатор видаленого запису	Повідомлення про помилку: «Запис не знайдено»	Повідомлення про помилку	Пройдено

Перший тестовий випадок (ТС01) перевіряв можливість створення нового запису. Користувач вводив дані для нового замовлення на ремонт і натискав кнопку

«Зберегти». Система успішно зберігала цей запис у базі даних, відображаючи підтвердження користувачеві. Тест був успішно пройдений, оскільки новий запис був створений і збережений коректно.

У другому тестовому випадку (TC02) перевірялася можливість зчитування даних з бази даних. Користувач запитував інформацію про існуюче замовлення, і система успішно витягала цей запис та відображала його користувачеві. Результати тестування підтвердили, що дані були зчитані і відображені коректно.

Третій тестовий випадок (TC03) був спрямований на перевірку можливості оновлення існуючого запису. Користувач редагував дані існуючого замовлення і натискав кнопку «Оновити». Система зберігала зміни в базі даних. Тестування показало, що всі зміни були збережені коректно.

Четвертий тестовий випадок (TC04) перевіряв можливість видалення існуючого запису. Користувач видаляв існуюче замовлення, і система успішно видаляла цей запис з бази даних. Тест пройшов успішно, підтверджуючи, що запис був видалений коректно.

П'ятий тестовий випадок (TC05) перевіряв, як система обробляє запит на зчитування запису, що не існує. Користувач запитував інформацію про замовлення, якого немає в базі даних. Система вивела повідомлення про відсутність запису, що підтвердило коректну роботу.

Шостий тестовий випадок (TC06) перевіряв можливість оновлення неіснуючого запису. Користувач намагався оновити замовлення, якого немає в базі даних. Система вивела повідомлення про відсутність запису для оновлення, що відповідало очікуванням.

Сьомий тестовий випадок (TC07) перевіряв можливість видалення неіснуючого запису. Користувач намагався видалити замовлення, якого немає в базі даних. Система вивела повідомлення про відсутність запису для видалення, підтверджуючи успішне проходження тесту.

Восьмий тестовий випадок (TC08) перевіряв, як система обробляє масове створення нових записів. Користувач створював відразу кілька нових замовлень. Усі нові записи були створені і збережені в базі даних коректно.

Останній тестовий випадок (TC09) перевіряв можливість відновлення раніше видаленого запису. Користувач намагався відновити замовлення, яке було видалено. Система вивела повідомлення про те, що запис не може бути відновлений, що відповідало очікуванням.

Тестування CRUD-операцій показало, що програмне забезпечення працює коректно при виконанні основних операцій створення, зчитування, оновлення та видалення записів у базі даних. Система правильно обробляє помилки і виконує всі операції згідно з вимогами, що забезпечує надійність і ефективність управління даними в сервісному центрі. Для більшої наочності та систематизації робочого процесу обліку ремонтних робіт у сервісному центрі рекомендується підтримувати регулярне тестування всіх CRUD операцій, що дозволить виявити і усунути можливі проблеми в роботі системи на ранніх етапах.

5.2 Тестування алгоритму авторизації

В ході тестування системи авторизації було перевірено вісім основних тестових випадків, які покривають всі важливі аспекти входу, виходу та управління обліковими записами користувачів (таблиця 5.2).

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

Тестовий випадок	Опис	Очікуваний результат	Результат
TC01: Успішний вхід користувача	Перевірка входу з коректними даними	Користувач успішно входить у систему	Пройдено
TC02: Вхід з некоректним паролем	Перевірка входу з неправильним паролем	Система виводить повідомлення про помилку	Пройдено
TC03: Вхід з некоректним ім'ям користувача	Перевірка входу з неправильним ім'ям користувача	Система виводить повідомлення про помилку	Пройдено
TC04: Блокування облікового запису після кількох невдалих спроб входу	Перевірка блокування облікового запису після кількох невдалих спроб	Обліковий запис блокується після певної кількості спроб	Пройдено

Кінець табл. 5.2

Тестовий випадок	Опис	Очікуваний результат	Результат
TC05: Відновлення пароля	Перевірка процедури відновлення пароля	Система відправляє посилання для відновлення пароля на email користувача	Пройдено
TC06: Вихід з системи	Перевірка коректного виходу з системи	Користувач успішно виходить із системи	Пройдено
TC07: Перевірка сесії користувача	Перевірка збереження сесії після входу	Сесія зберігається до виходу користувача або завершення часу	Пройдено
TC08: Доступ до захищених сторінок без авторизації	Перевірка доступу до сторінок без входу	Система перенаправляє на сторінку входу	Пройдено

Перший тестовий випадок (TC01) перевіряв успішний вхід користувача з коректними даними. Користувач вводив правильні ім'я та пароль, і система дозволяла йому успішно увійти, що підтвердило коректність роботи механізму авторизації.

У другому та третьому тестових випадках (TC02 і TC03) перевірялися сценарії входу з некоректними даними. Користувач вводив неправильний пароль або ім'я, і система виводила повідомлення про помилку, запобігаючи несанкціонованому доступу. Тести пройшли успішно, підтверджуючи надійність системи у випадку помилкових спроб входу.

Четвертий тестовий випадок (TC04) перевіряв блокування облікового запису після кількох невдалих спроб входу. Система блокувала обліковий запис після певної кількості неправильних введень пароля, що забезпечувало додатковий рівень захисту від атак.

П'ятий тестовий випадок (TC05) був спрямований на перевірку процедури відновлення пароля. Користувач запитував відновлення пароля, і система відправляла посилання для відновлення на його email, що підтвердило правильність реалізації цієї функції.

Шостий тестовий випадок (TC06) перевіряв коректний вихід з системи. Ко-

ристувач успішно виходив із системи, і всі його сесії завершувалися, що забезпечувало безпеку облікового запису.

Сьомий тестовий випадок (TC07) перевіряв збереження сесії користувача після входу. Система зберігала сесію до моменту виходу користувача або завершення часу сесії, що гарантувало стабільність роботи.

Восьмий тестовий випадок (TC08) перевіряв доступ до захищених сторінок без авторизації. Система перенаправляла неавторизованих користувачів на сторінку входу, що підтвердило коректну роботу механізму захисту.

Тестування авторизації показало, що система працює коректно і надійно, забезпечуючи безпеку користувацьких даних та захист від несанкціонованого доступу. Регулярне проведення таких тестів допоможе підтримувати високий рівень безпеки та ефективності роботи системи.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було зосереджено увагу на розробці програмної системи для підтримки ведення обліку робіт з ремонту та обслуговування комп'ютерної техніки у сервісному центрі. Цей підхід базується на використанні нереляційної бази даних та логіки об'єктно-орієнтованого програмування.

Для створення додатку використовувалися наступні інструменти та технології: середовище розробки Microsoft Visual Studio Code, платформа Node.js, мова програмування JavaScript, фреймворк для інтерфейсу користувача React.js та система управління базами даних MongoDB. Процес проектування розпочався з концептуального моделювання предметної області, що плавно перейшло до реалізації фізичної моделі за допомогою MongoDB. Особлива увага була приділена розумінню логіки зберігання та роботи з інформацією в нереляційній базі даних.

Виконання цієї роботи дозволило детально ознайомитися з технологією доступу та управління базами даних MongoDB, а також освоїти основні принципи роботи з цим інструментом. Результатом роботи став готовий додаток, який забезпечує ефективний облік робіт з ремонту та обслуговування комп'ютерної техніки у сервісному центрі. Цей додаток не лише демонструє високий рівень експертизи у розробці програмних систем, але й надає практичну користь для задоволення потреб галузі. Його функціональні можливості дозволяють оптимізувати процеси обліку та управління ремонтними роботами, що сприяє підвищенню ефективності роботи сервісного центру та покращенню якості обслуговування клієнтів.

Також було виявлено важливі аспекти, пов'язані з розробкою програмних систем загалом. Це включало в себе управління проектом, комунікацію зі стейкхолдерами, тестування програмного забезпечення та забезпечення його безперебійної роботи. Ці аспекти не лише підтримали успішне виконання проекту, але й надали можливість здобути додатковий досвід у сфері розробки програмного забезпечення та роботи з клієнтами. В цілому, цей проект не лише відображає технічні навички та експертизу, але й підкреслює важливість ефективного управління процесом розробки для досягнення успіху в сучасному програмному середовищі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. CRM система для сервісного центру: сайт. [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/crm-dlja-servisnyh-tsentrov / (дата звернення 23.04.2023).

2. Система «Alldevice». [Електронний ресурс] – Режим доступу до ресурсу: https://as-service.com.ua/programs_ua/alldevice-upravlinnya-tehnichnim-obslugovuvannya-i-remontom.html (дата звернення 25.05.24).

3. Система «Gincore». [Електронний ресурс] – Режим доступу до ресурсу: https://gincore.net/uk (дата звернення 25.05.24)

4. Система «USU Сервісний центр». [Електронний ресурс] – Режим доступу до ресурсу: https://ususoft.com.ua/uk/uchet_servisnogo_tsentra.php (дата звернення 25.05.24).

5. Terry Quatrani. Visual modeling with Rational Rose 200 and UML – Addison-Wesley, 2001. – 256 p.

6. Martin Fowler. UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd Edition – Print2print, 2016. – 208p.

7. Jason Katzer. Learning Serverless: Design, Develop, and Deploy with Confidence. 1st Edition – O'Reilly Media, 2020. – 222p.

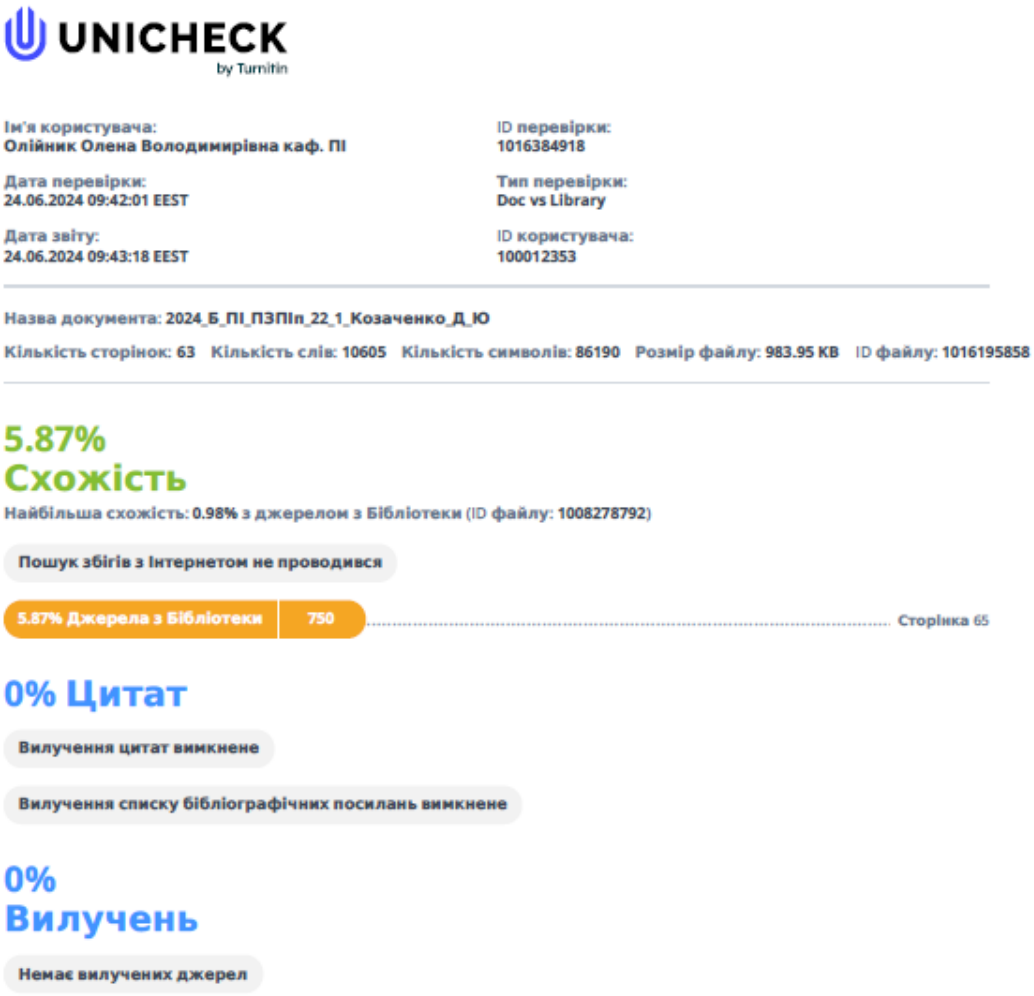
8. Benoir Lecrerc and Jesse Cale. Big Data. – Tailor&Francis, 2022. – 148p.

9. Difference between SQL and NoSQL – GeeksforGeeks. [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/difference-between-sql-and-nosql/ (дата звернення 01.06.2024).

10. Офіційний сайт СУБД MongoDB. [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/what-is-mongodb (дата звернення 01.06.2024).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту у базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016384918

Дата перевірки:
24.06.2024 09:42:01 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
24.06.2024 09:43:18 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІп_22_1_Козаченко_Д_Ю

Кількість сторінок: 63 Кількість слів: 10605 Кількість символів: 86190 Розмір файлу: 983.95 KB ID файлу: 1016195858

5.87%
Схожість

Найбільша схожість: 0.98% з джерелом з Бібліотеки (ID файлу: 1008278792)

Пошук збігів з Інтернетом не проводився

5.87% Джерела з Бібліотеки 750 ..... Сторінка 65

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

ДОДАТОК Б

Фрагмент коду операцій CRUD

```
import { useState, useEffect } from "react";
import axios from "axios";

const useCrudOperations = (endpoint, initialSearchParams = {}) => {
  const [data, setData] = useState([]);
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);
  const [searchParams, setSearchParams] = useState(initialSearchParams);

  const baseURL = "http://localhost:3000";

  const fetchData = async (params = searchParams) => {
    const queryString = new URLSearchParams(params).toString();

    setIsLoading(true);
    try {
      const response = await axios.get(`${baseURL}${end-
point}?${queryString}`);
      setData(response.data);
    } catch (err) {
      setError(err);
    } finally {
      setIsLoading(false);
    }
  };

  const createItem = async (item) => {
    try {
      await axios.post(`${baseURL}${endpoint}`, item);
      fetchData(); // Refetch data after creating item
    } catch (err) {
      setError(err);
    }
  };

  const updateItem = async (id, updatedItem) => {
    try {
      await axios.put(`${baseURL}${endpoint}/${id}`, updatedItem);
      fetchData(); // Refetch data after updating item
    } catch (err) {
      setError(err);
    }
  };

  const deleteItem = async (id) => {
    try {
      await axios.delete(`${baseURL}${endpoint}/${id}`);
      fetchData(); // Refetch data after deleting item
    } catch (err) {
      setError(err);
    }
  };
```

```
  const refetch = (params = searchParams) => {
    fetchData(params);
  };

  useEffect(() => {
    fetchData();
  }, [endpoint, searchParams]);

  return {
    data,
    isLoading,
    error,
    fetchData,
    createItem,
    updateItem,
    deleteItem,
    refetch,
    setSearchParams,
  };
};

export default useCrudOperations;
```

ДОДАТОК В

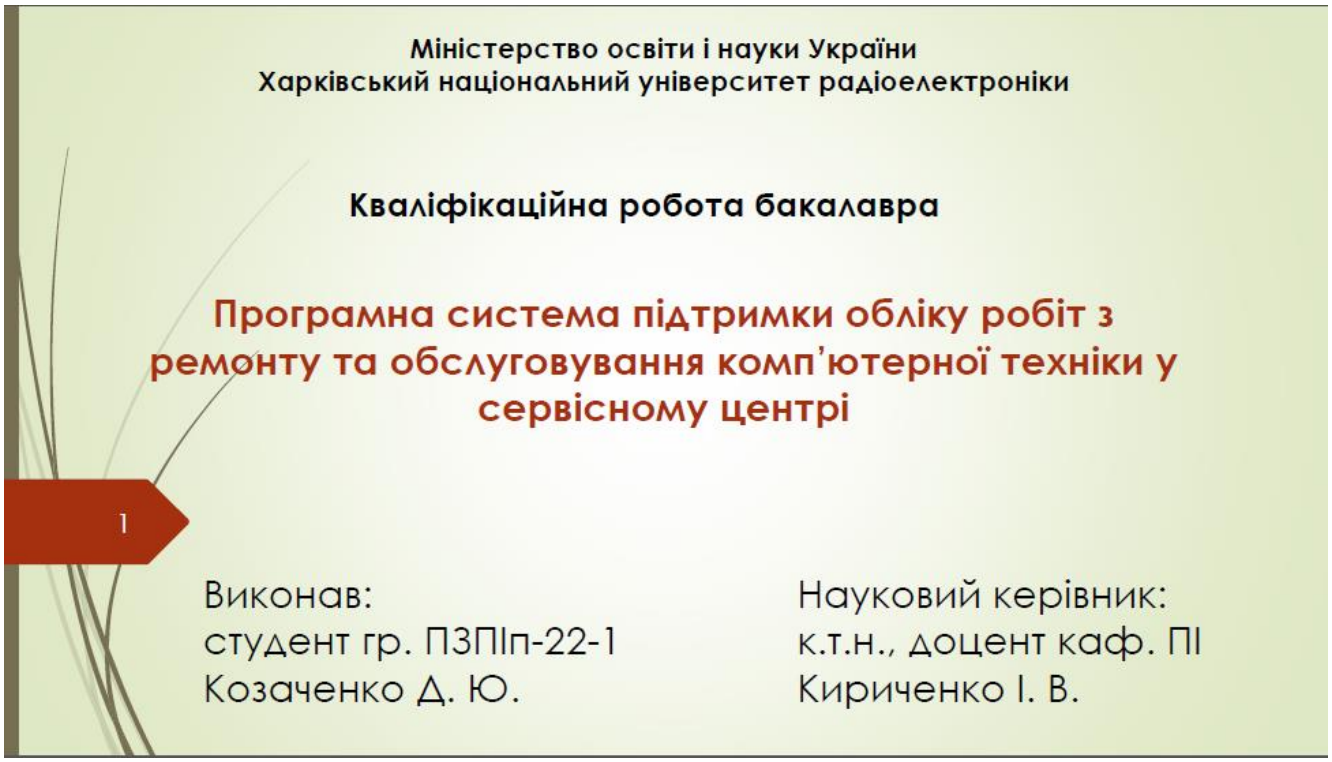
Фрагмент коду реалізації авторизації

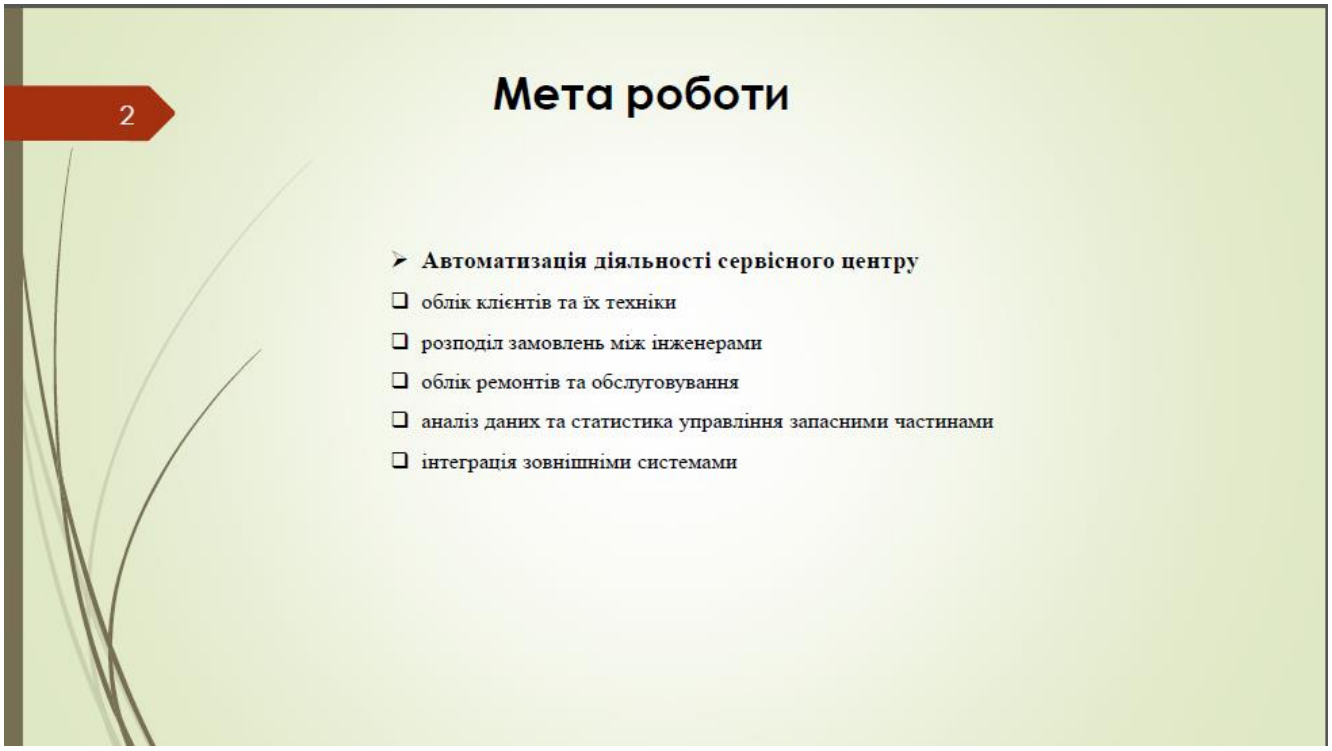
```
import axios from 'axios';
import toast from 'react-hot-toast';
import { createAsyncThunk } from '@reduxjs/toolkit';

axios.defaults.baseURL = 'http://localhost:3000';
const token = {
  set(token) {
    axios.defaults.headers.common.Authorization = `Bearer ${token}`;
  },
  unset() {
    axios.defaults.headers.common.Authorization = ``;
  },
};

const addNewUser = createAsyncThunk(
  'authification/addNewUser',
  async userData => {
    try {
      const { data } = await axios.post('/users/signup', userData);
      token.set(data.token);
      toast.success(`The registration procedure was successful`);
      return data;
    } catch (error) {
      toast.error(`The registration procedure was failture`);
    }
  }
);

const loginUser = createAsyncThunk(
  'authification/loginUser',
  async userData => {
    try {
      const { data } = await axios.post('/users/login', userData);
      token.set(data.token);
      toast.success(`Welcome to the My Contacts application`);
      return data;
    } catch (error) {
      toast.error(`The log in procedure was failture`);
    }
  }
);

const logoutUser = createAsyncThunk('authification/logout', async () => {
  try {
    await axios.post('/users/logout');
    token.unset();
    toast.success(`Log out was successfully`);
  } catch (error) {
    toast.error(`Sorry, we cann't to log out you. Try agan.`);
  }
});

const getUserInformation = createAsyncThunk(
```

```
  'authification/getUserInformation',
  async (_, thunkAPI) => {
    const currentToken = thunkAPI.getState().authification.token;
    if (currentToken === null) {
      return thunkAPI.rejectWithValue();
    }
    token.set(currentToken);
    try {
      const { data } = await axios.get('/users/current');
      return data;
    } catch (error) {
      toast.error(`Something wrong. We have no user data.`);
    }
  }
);

const authOperations = {
  addNewUser,
  loginUser,
  logoutUser,
  getUserInformation,
};
export default authOperations;
```

ДОДАТОК Г

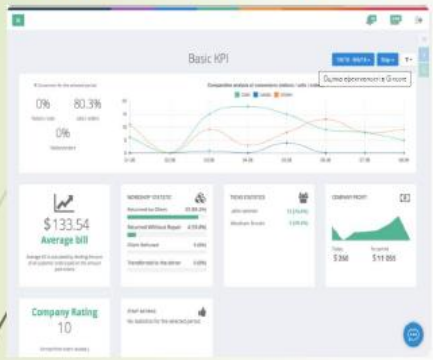
Слайди презентації



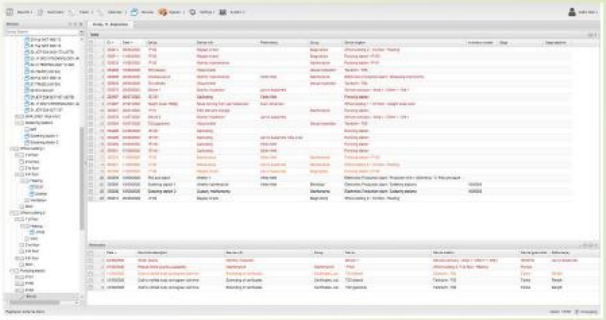


3

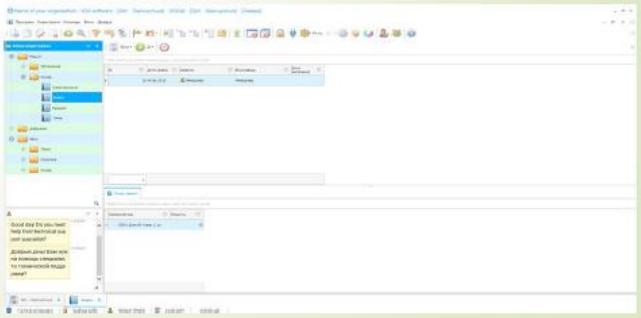
Система Gincore



Система Alldevice



Система USU Сервісний центр



4

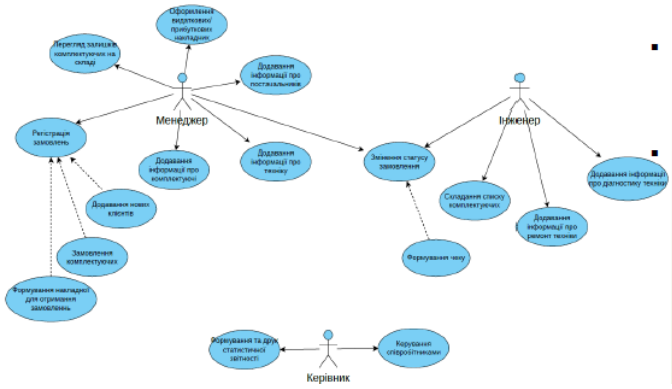
Аналіз аналогів

Система	Alldevice	Gincore	USU Сервісний центр
Облік звернень	так	так	так
Журнал ремонту	так	так	так
Облік гарантійних талонів	так	ні	ні
Клієнтська база	так	так	так
Розмежування прав користувачів	так	так	так
Додавання планового технічного обслуговування	так	ні	ні
Доступ через хмару	так	так	ні
Перехід з іншої облікової системи	ні	ні	так
Інтеграція з іншими сервісами	ні	так	ні
Простота використання	Складна	Складна	Складна
Вартість	$1200	$200/міс.	$40/міс (+сховані доплати)

5

Постановка задачі та опис системи

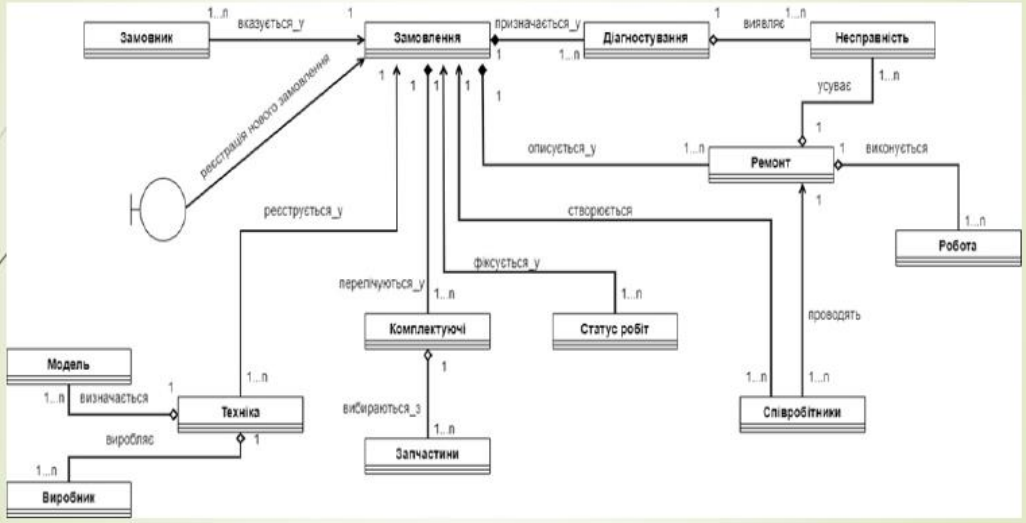
Спроектувати та реалізувати програмну систему підтримки обліку робіт з ремонту та обслуговування комп'ютерної техніки у сервісному центрі



- керівник, який здійснює контроль та управління співробітниками, а також займається формуванням статистичної звітності
- менеджер/адміністратор, який відповідає за взаємодію з клієнтами, реєстрацію та ведення заявок, роботу з постачальниками
- інженер, відповідальний за діагностику та ремонт техніки

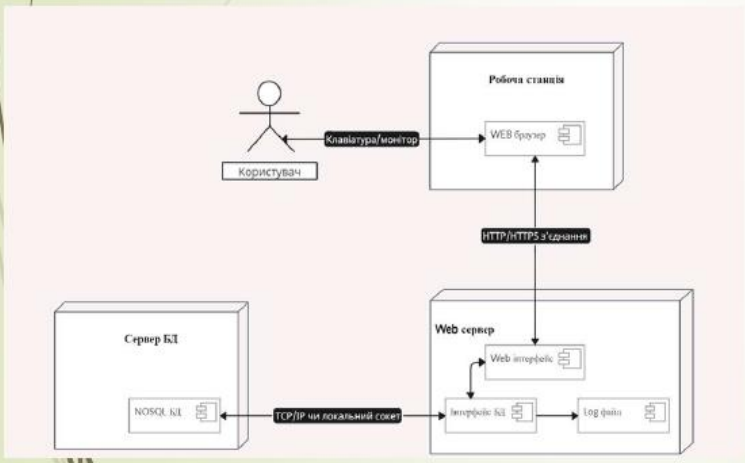
6

Діаграма класів



7

Діаграма розгортання



Вибір технологій розробки

- мова програмування: JavaScript
- система управління базами даних: Mongo DB
- фреймворк для інтерфейсу користувача: React.js
- платформа для серверної частини: Node.js

8

Логічна модель

ER-діаграма

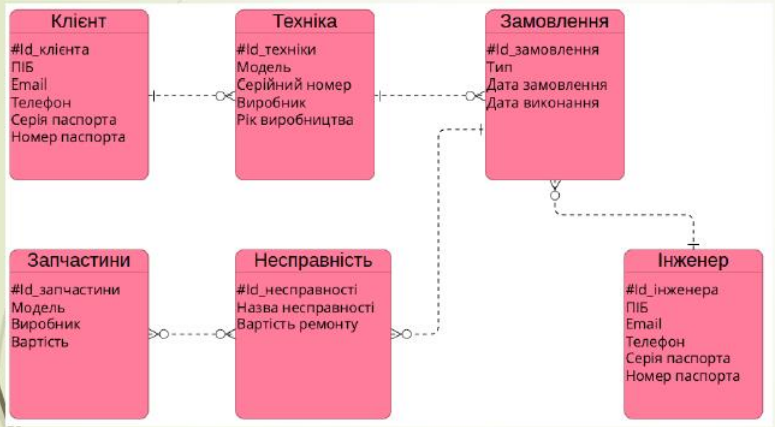


Схема запитів

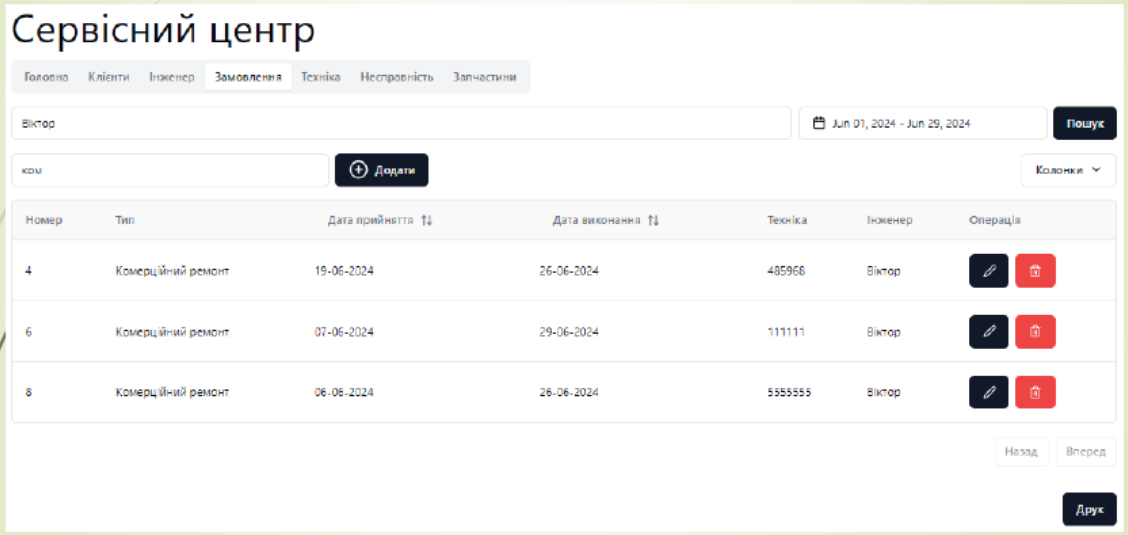
9

Приклад програмної реалізації серверної частини для таблиці «Клієнти»

```
// Endpoints for Clients
app.get("/clients", async (req, res) => {
  const clients = await prisma.client.findMany();
  res.json(clients);
});

app.post("/clients", async (req, res) => {
  const { name, email, phone, passportSeries, passportNumber } =
req.body;
  const client = await prisma.client.create({
    data: { name, email, phone, passportSeries, passportNumber
},
  });
  res.json(client);
});

app.put("/clients/:id", async (req, res) => {
  const { id } = req.params;
  const { name, email, phone, passportSeries, passportNumber } =
req.body;
  const client = await prisma.client.update({
    where: { id: parseInt(id, 10) },
    data: { name, email, phone, passportSeries, passportNumber
},
  });
  res.json(client);
});

app.delete("/clients/:id", async (req, res) => {
  const { id } = req.params;
  await prisma.client.delete({
    where: { id: parseInt(id, 10) },
  });
  res.status(204).send();
});
```

10

Інтерфейс користувача



11

Тестування

Тестовий випадок	Опис	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
TC01: Створення запису	Перевірка створення нового запису в базі даних	Данні про роботу: Назва, Опис, Дата	Запис успішно створено, відображається у списку записів	Запис створено, відображається у списку	Пройдено
TC02: Читання запису	Перевірка читання існуючого запису з БД	Ідентифікатор запису	Відображення всіх даних відповідного запису	Дані запису відображені	Пройдено
TC03: Оновлення запису	Перевірка оновлення існуючого запису в базі даних	Ідентифікатор запису, Нові дані	Запис успішно оновлено, зміни збережені	Запис оновлено	Пройдено
TC04: Видалення запису	Перевірка видалення існуючого запису з бази даних	Ідентифікатор запису	Запис успішно видалено, більше не відображається у списку записів	Запис видалено	Пройдено

12

Практична користь

- **Ефективний облік робіт**
  Готовий додаток забезпечує ефективний облік робіт з ремонту та обслуговування комп'ютерної техніки
- **Оптимізація процесів**
  Оптимізація процсів обліку та управління ремонтними роботами сприяє підвищенню ефективності роботи сервісного центру
- **Покращення обслуговування клієнтів**
  Додаток дозволяє підвищити якість обслуговування клієнтів завдяки швидкому доступу до інформації та поліпшенню контролю за виконанням робіт

13

Висновки

- Проведений аналіз предметної галузі
- Спроектована архітектура програмної системи
- Реалізована та протестована програмна система
- Закладені основи розвитку програмної системи для підтримки роботи сервісного центру