Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка вебсайту для автосервісу з використанням Spring

Виконав: студент IV курсу, групи СПс-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Лаповець Я. А.
(підпис) (прізвище та ініціали)

Керівник _____ Бревус В. М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль - 2024

АНОТАЦІЯ

Кваліфікаційна робота бакалавра за спеціальністю 121 – Інженерія програмного забезпечення. Тернопільський національний технічний університет ім. Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПс-42, 2024 рік. Опис кваліфікаційної роботи бакалавра складається з 60 сторінок і включає 41 рисунок, 2 таблиці і 2 додатки. Метою цієї роботи є створення веб-сайту для автосервісу з допомогою Spring Framework.

Основною метою цієї роботи було розробити ефективний та функціональний веб-сайт, який би автоматизував процес управління автомобільними послугами. Для цієї роботи було використано Spring Framework, популярний інструмент для розробки Java-додатків. Цей інстумент забезпечує ефективну інтеграцію системних компонентів, управління залежностями та простоту розробки та тестування програмного забезпечення.

Основні функції розробленого веб-сайту демонструють управління замовленнями на діагностику та ремонт транспортних засобів, управління послугами автосервісу та надання зручного інтерфейсу для клієнтів і техніків автосервісу. У процесі розробки було використано принципи об'єктно-орієнтованого програмування (ООП) та шаблони дизайну для створення гнучкої та розширюваної системи. В якості бази даних вибирається PostgreSQL, що дозволяє зберігати і отримувати доступ до різних типів інформації, що відноситься до обслуговування автомобілів.

Ключові слова: автосервіс, веб-сайт, Spring Framework, Spring Boot, Java, об'єктно-орієнтоване програмування, PostgreSQL, HTML, CSS, JavaScript, Docker, JWT, аутентифікація, PostMan, Maven

ANNOTATION

Bachelor's Thesis in the specialty 121 – Software Engineering. Ternopil Ivan Puluj National Technical University, Faculty of Computer-Information Systems and Software Engineering, Department of Software Engineering, Group SPs-42, 2024. The description of the bachelor's thesis consists of 60 pages and includes 41 figures, 2 tables, and 2 appendices. The aim of this work is to create a website for an auto service using the Spring Framework.

The primary objective of this work was to develop an efficient and functional website that automates the management process of automotive services. The Spring Framework, a popular tool for developing Java applications, was used for this project. This tool provides efficient integration of system components, dependency management, and ease of software development and testing.

The main functions of the developed website demonstrate the management of orders for diagnostics and repair of vehicles, management of auto service services, and providing a user-friendly interface for clients and auto service technicians. Object-oriented programming (OOP) principles and design patterns were used to create a flexible and extensible system. PostgreSQL was chosen as the database, allowing for the storage and retrieval of various types of information related to car maintenance.

Keywords: auto service, website, Spring Framework, Spring Boot, Java, object-oriented programming, PostgreSQL, HTML, CSS, JavaScript, Docker, JWT, authentication, PostMan, Maven

ЗМІСТ

ВСТУП

Технології стали невід'ємною частиною повсякденного життя, програмне забезпечення відіграє важливу роль у багатьох галузях. Індустрія автомобільних послуг не є винятком, для її розвитку важливо впроваджувати Інформаційні технології. Розробка веб-сайтів для управління обслуговуванням автомобілів стає все більш актуальною для покращення управління та оптимізації процесів.

Метою цього кваліфікаційного дослідження є розробка веб-сайту для автосервісу, який надає зручні та ефективні інструменти для реєстрації технічного обслуговування та діагностики, управління замовленнями, перегляду інформації про автомобілі та клієнтів. Цей веб-сайт спрощує процес обслуговування клієнтів, забезпечуючи точний розрахунок вартості запасних частин та послуг та облік заробітної плати працівників.

Для досягнення цієї мети будуть використовуватися стеки сучасних технологій, включаючи Java і Spring Boot для серверної розробки, PostgreSQL в якості бази даних і React.JS для розробки інтерфейсу, Docker для контейнеризації та JWT для аутентифікації користувачів. Такий підхід дозволяє створити потужний і масштабований веб-сайт.

В даний час існує великий попит на веб-сайти з управління автомобільними послугами, які пропонують комплексний підхід до автоматизації процесів. Результати цього дослідження мають практичне значення для автосервісної компанії, допомагаючи підвищити ефективність бізнесу та задовольнити потреби клієнтів. Створення цього веб-сайту допоможе вирішити проблеми, пов'язані з ручним керуванням, помилками та затримками в обслуговуванні.

У цій роботі розглянемо важливі функції веб-сайту автосервісної компанії. Зокрема, був розроблений програмний модуль для відображення службових записів, який дозволяє переглядати зайнятість майстра.

Крім того, сайт надає можливість замовлення послуг, також полегшує процес створення замовлень. Також реалізована можливість перегляду заробітньої плати майстра.

Результатом цієї кваліфікаційної роботи є проект. Який може бути використаний для управління обслуговуванням автомобілів, забезпечення більш ефективної та зручної роботи для власників, майстрів та клієнтів.

У наступному розділі детально описується процес аналізу вимог, проектування, розробки та тестування веб-сайту автомобільної сервісної компанії. Аналіз вимог визначає основні потреби і вимоги, які повинні бути виконані, а також стандарти і норми автомобільного сервісного бізнесу. На основі цих вимог розробляється веб-сайт, розробляється структура бази даних, визначається логіка роботи і взаємодії різних модулів.

Розробка веб-сайтів передбачає створення серверних та інтерфейсних компонентів за допомогою Java, Spring Framework та React, HTML, CSS, JS.

В результаті дослідження будуть оцінені отримані результати, проаналізовані переваги сайту автосервісної компанії, дані рекомендації щодо подальшого розвитку і вдосконалення. Також буде надано список використаних джерел, що підтверджують достовірність і наукову достовірність дослідження.

1 АНАЛІЗ ВИМОГ

1.1 Актуальність програмного забезпечення

Важливість цього питання обумовлена швидким розвитком індустрії автомобільних послуг і зростаючою потребою клієнтів в зручних і ефективних інструментах управління автомобільними послугами. Завдяки впровадженню сучасних технологій автосервісні компанії можуть оптимізувати свою діяльність, покращити обслуговування клієнтів і підвищити продуктивність праці майстрів. Традиційні методи управління обслуговуванням транспортних засобів, такі як ручна обробка замовлень або реєстрація необхідних послуг, вимагають багато часу і часу. Крім того, такий підхід може призвести до помилок і незручностей в роботі персоналу.

Таким чином, для автосервісу потрібен веб-сайт для управління, який спрощує процес і забезпечує точність і швидкість обробки замовлень. Розробка веб-сайту для управління автомобільними послугами на основі фреймворків Spring Boot та React та бази даних допомогою PostgreSQL, оскільки ці технології широко використовуються в програмуванні і мають великий потенціал для реалізації таких рішень. Використання цих технологій дозволить створювати надійні і функціональні вироби, що відповідають потребам автосервісів і сприяють їх успішній роботі.

Крім того, зростаюча популярність онлайн-записів з технічного обслуговування та діагностики робить потребу в ефективних веб-сайтах для управління обслуговуванням автомобілів ще більш нагальною. Завдяки розробленому веб-сайту компанії з обслуговування автомобілів можуть швидше та ефективніше обробляти замовлення, відстежувати їх статус, керувати своїми послугами та керувати зайнятістю ремісників. Це забезпечує високий рівень обслуговування та задоволеності клієнтів.

Веб-сайт для автомобільних сервісів, розроблений на основі мові програмування Java, фреймворку Spring Framework, бази даних PostgreSQL та фреймворку для JS React.JS

Тому розробка веб-сайту для управління автомобільними послугами на основі цих технологій є важливою та відповідає потребам індустрії автомобільних послуг.

1.1.1 Огляд конкурентів

Одним із конкурентів у вирішенні питань розробки програмного забезпечення для автосервісного підприємства є програма ‘Mitchell’(див. рис. 1.1)

Рисунок 1.1 – Логотип компанії конкурента Mitchell 1

Mitchell 1 пропонує програмне забезпечення для автосервісів, що також охоплює широкий спектр функцій:

Історія обслуговування: Mitchell 1 дозволяє зберігати та відстежувати історію обслуговування кожного автомобіля, що спрощує управління клієнтами та надає точну інформацію про попередні роботи.

Управління запасами: Програма дозволяє ефективно керувати запасами автосервісу, відстежувати наявність запасних частин, планувати замовлення та уникати нестачі необхідних компонентів.

Розрахунок вартості робіт: Mitchell 1 надає інструменти для точного розрахунку вартості робіт та матеріалів, що дозволяє уникнути помилок у фінансових розрахунках.

Клієнтський сервіс: Платформа пропонує зручні інструменти для зв'язку з клієнтами, включаючи нагадування про записи, відправку повідомлень та збір зворотного зв'язку.

Аналітика та звіти: Mitchell 1 також забезпечує доступ до аналітики та звітів, що допомагає керівникам автосервісів приймати обґрунтовані рішення на основі даних про продуктивність та ефективність роботи.

Аналіз конкурентів показує, що на ринку існують потужні рішення для управління автосервісами, які пропонують широкий спектр функціональних можливостей. Проте, наш вебсайт може виділитися за рахунок інтеграції новітніх технологій, зручності користування та адаптації до специфічних потреб користувачів. Враховуючи сильні сторони та усуваючи недоліки існуючих рішень, зможемо створити конкурентоспроможний продукт, що задовольнить вимоги сучасного автосервісного бізнесу.

1.2 Пошук основних акторів системи та варіантів використання

1.2.1 Виявлення основних акторів системи

На рисунку 1.2 зображено список усіх основних акторів системи.

Рисунок 1.2 – Основні актори системи

Короткий опис усіх основних акторів системи представлений в таблиці 1.

Табл. 1. Виявлення основних акторів системи

Актор	Короткий опис
Адміністратор	Може створювати, редагувати або видаляти замовлення. Також має доспуп до перегляду активних замовлень, додавання послуг, деталей та майстрів. Крім того, може додавати, редагувати, видаляти користувачів.
Працівник	Може додавати, редагувати замовлення, переглядати інформацію про конкретного користувача, та переглянути свою зарплату.

1.2.2 Виявлення варіантів використання

Виявлення варіантів використання представлені таблиці 2.

Табл. 2. Варіанти використання

Адміністратор	Управління послугами	Можливість адміністратору додавати чи видаляти послуги які виконують в автосервісі.
Адміністратор	Управління речами	Можливість адміністратору додавати чи видаляти речами які використовують в автосервісі.
Адміністратор	Управління замовленнями	Можливість адміністратору додавати чи видаляти замовлення в автосервісі.
Адміністратор	Управління замовленнями	Можливість адміністратору переглядати усі замовлення які виконувалися в автосервісі.
Адміністратор	Управління працівниками	Можливість адміністратору створювати чи видаляти працівників.
Працівник	Управління замовленнями	Можливість працівнику додавати чи редагувати створенні замовлення, змінювати список послуг та речей, обраховувати свою зарплату з замовлення
Працівник	Отримання інформації про замовлення	Працівник отримує всю інформацію про замовлення.

1.2.3 Розробка варіантів використання

Нижче на рисунку 1.3 та рисунку 1.4 зображені діаграми прецедентів системи

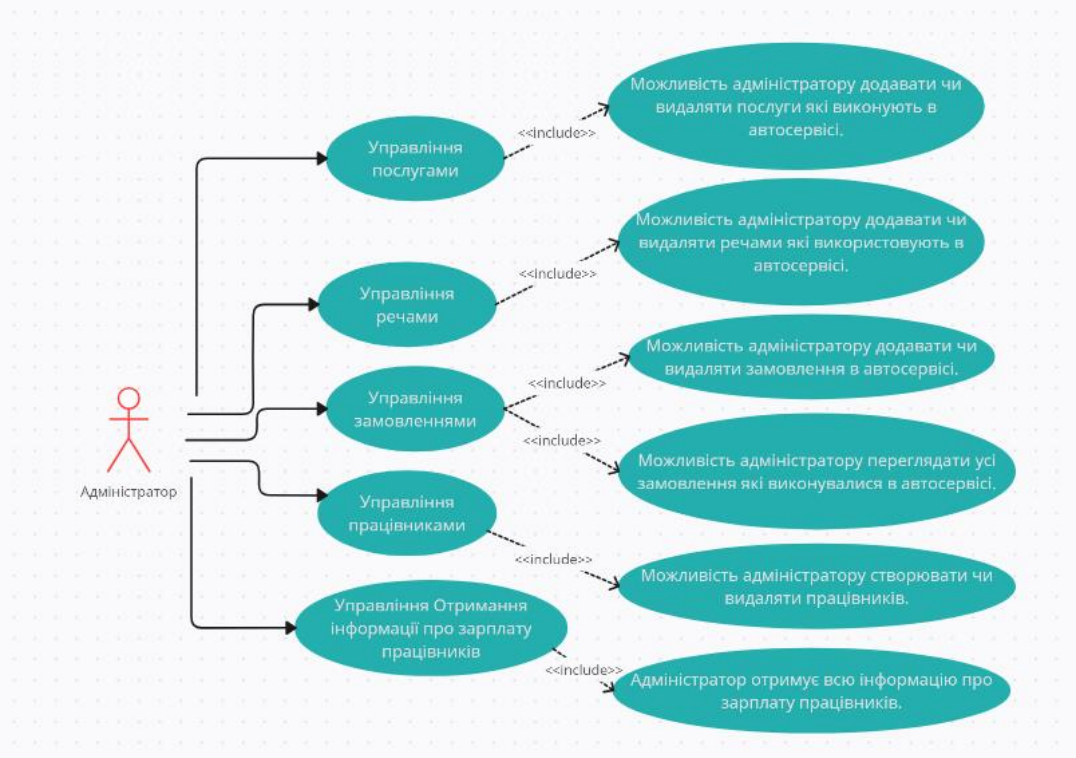

Рисунок 1.3 – Діаграма прецедентів системи

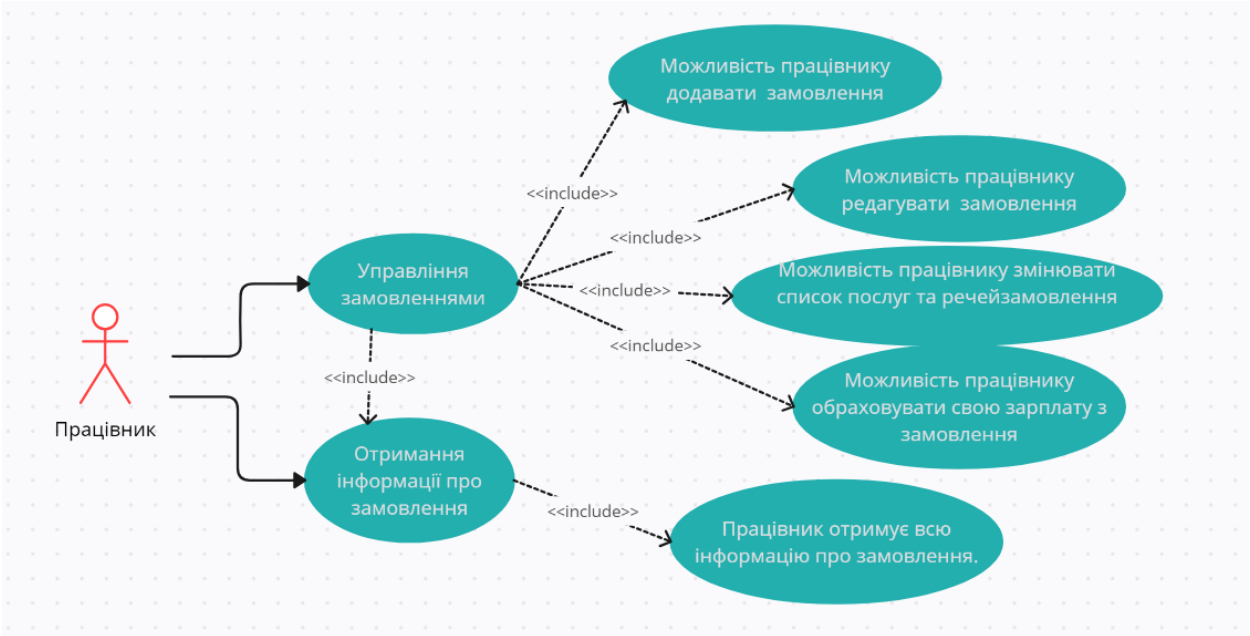

Рисунок 1.4 – Діаграма прецедентів системи

1.3 Діаграми послідовності

Діаграма послідовності — це діаграма, яка показує взаємодії об’єктів, класифіковані за моментом їх прояву. На діаграмі послідовності вісь часу завжди присутня, що дає змогу візуалізувати часовий зв’язок між переданими повідомленнями. За допомогою діаграми послідовності можна представити взаємодію елементів моделі у вигляді своєрідного графіка «життєвого циклу» сукупності взаємопов’язаних об’єктів, які виконують функцію варіантного використання системи програмне забезпечення, досягнення бізнес-цілі або виконання завдання.

Вони допомагають проектувати, аналізувати, документувати та повідомляти про поведінку системи, фіксуючи динамічні аспекти взаємодії об'єктів для досягнення певних функцій. Аналізуючи діаграми, розробники і дизайнери можуть дізнатися взаємодію між об'єктами, порядок обміну повідомленнями і час подій.

Це допоможе вам визначити можливі проблеми та зрозуміти загальну роботу системи. Процес отримання інформації про замовлення складається з декількох етапів. По-перше, користувач входить в систему, щоб переглянути замовлення. Потім система обробляє запит і повертає список дійсних замовлень, як показано на рисунку 1.5.

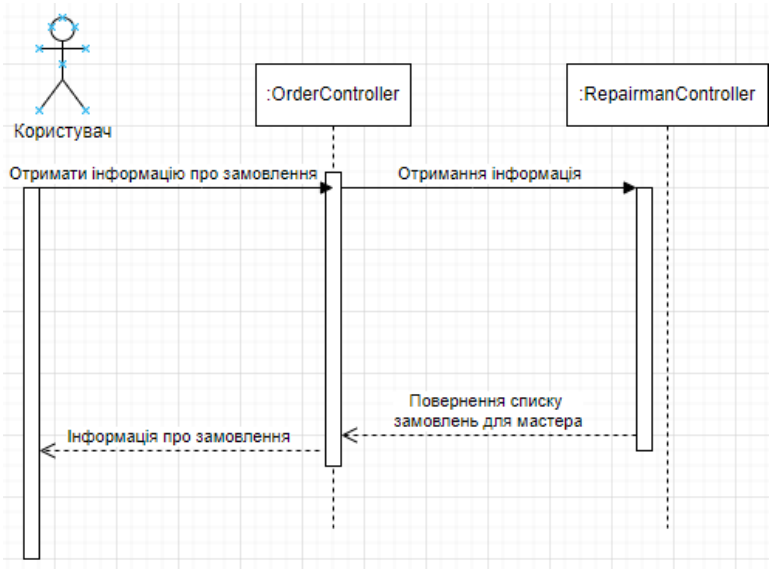

Рисунок 1.5 – Діаграма послідовності отримання інформації про замовлення

Після цього користувач вводить інформацію про себе і автомобілі, і створюється замовлення. Потім система обробляє цей запит і повертає повідомлення про успішне створення замовлення, як показано на рисунку 1.6.

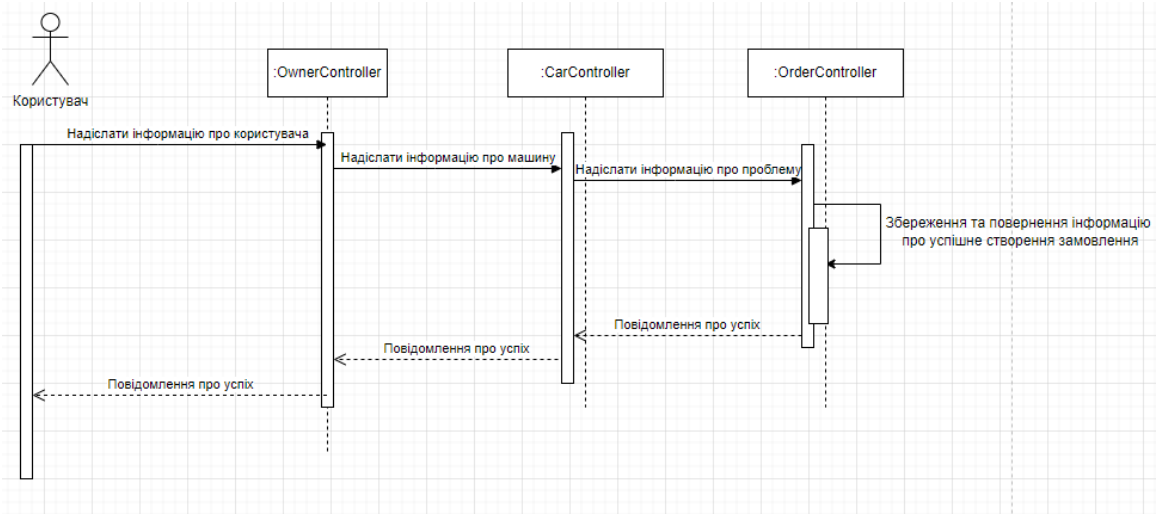

Рисунок 1.6 – Діаграма послідовності створення замовлення

Процес нарахування заробітної плати за поточні виконані замовлення протягом місяця складається з кількох етапів. Спочатку користувач входить у систему, щоб отримати доступ до свого особистого кабінету. Далі користувач натискає кнопку "Обчислити заробітну плату". Після цього система обробляє запит і повертає розраховану суму, як показано на рисунку 1.7.

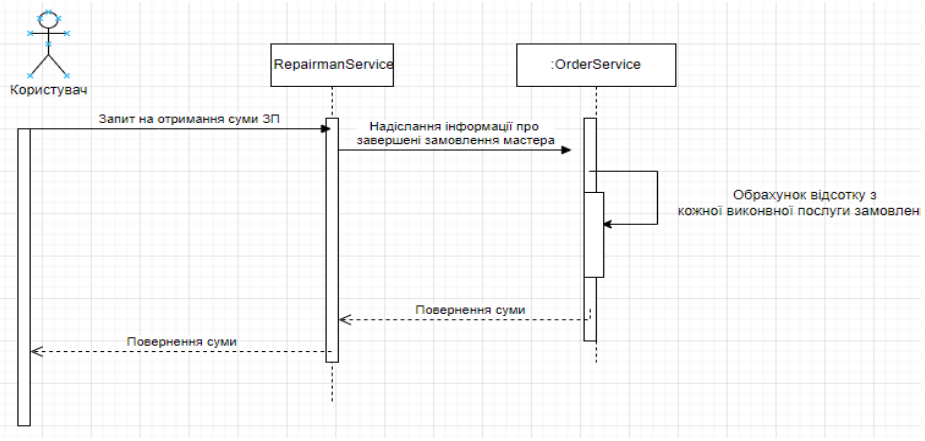

Рисунок 1.7 – Діаграма послідовності відображення

1.4 Вибір технологій розробки

Архітектура клієнт-сервер-це коли клієнтський пристрій або програма взаємодіє з центральним сервером для запиту та отримання послуг, ресурсів або даних. У цій моделі клієнт і сервер виконують різні функції та виконують різні завдання.

Клієнт або інтерфейс - це програма або пристрій, який взаємодіє з користувачем та ініціює запит на сервер. Він відповідає за перегляд інтерфейсу користувача, збір даних та надсилання запитів на сервер для обробки або отримання інформації.

Сервер або серверна частина - це мережа потужних комп'ютерів або комп'ютерів, які надають послуги або ресурси клієнту. Сервер отримує запити від клієнта, обробляє їх, повертає необхідну інформацію або виконує запитувану дію. Він відповідає за управління даними, виконання бізнес-логіки та координацію загальної функціональності системи.

Зв'язок між клієнтом і сервером зазвичай здійснюється через мережу, таку як Інтернет. Клієнт надсилає запити на сервер, і сервер обробляє їх і надсилає відповідь. Цей зв'язок здійснюється за допомогою певних протоколів та стандартів, таких як HTTP для веб-додатків.

Архітектура клієнт-сервер масштабована, оскільки кілька клієнтів можуть підключатися до одного сервера або групи серверів для доступу до спільних ресурсів, а оскільки дані та функції зберігаються та контролюються сервером, це також забезпечує централізоване адміністрування, безпеку та гнучкість даних. Функція зберігається та контролюється сервером.

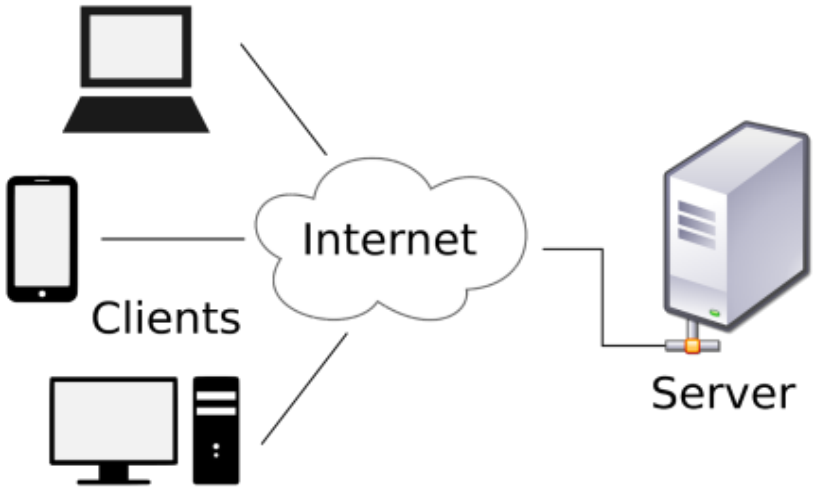

Рисунок 1.8 – Клієнт-серверна архітектура

Архітектура клієнт-сервер використовується в багатьох різних системах та додатках, включаючи веб-додатки, СУБД(системи управління баз даних), поштові сервери та інші різні мережеві системи (див. Рис. 1.8).

Мова програмування Java використовується для серверної частини проекту. Серверна частина дозволяє використовувати та взаємодіяти з базою даних і надавати відповідь клієнту через API. Використання Java забезпечує надійність, ефективність і масштабованість серверної частини проекту. Java-це універсальна, паралельна об'єктно-орієнтована мова програмування на основі класів. Він має спільні риси з мовами C і C++, але є деякі відмінності, такі як виключення певних аспектів або введення нових ідей. Java призначена для використання в реальних проектах, тому вона надійна і перевірена.

Spring Framework, високопродуктивний набір інструментів для розробки програмного забезпечення на платформі Java, використовується для розробки API для веб-додатків. Він забезпечує широке середовище та набір готових компонентів та бібліотек, які полегшують розробку програмних додатків. Структура Spring заснована на принципах інверсії управління (IoC) і аспектно-орієнтованого програмування (AOP).

Spring Framework активно підтримується широкою спільнотою розробників, яка надає документацію, приклади коду та відповіді на запитання. Це один з

найпопулярніших фреймворків для Java-додатків, який використовується багатьма великими проектами і компаніями (див. Рис. 1.9).

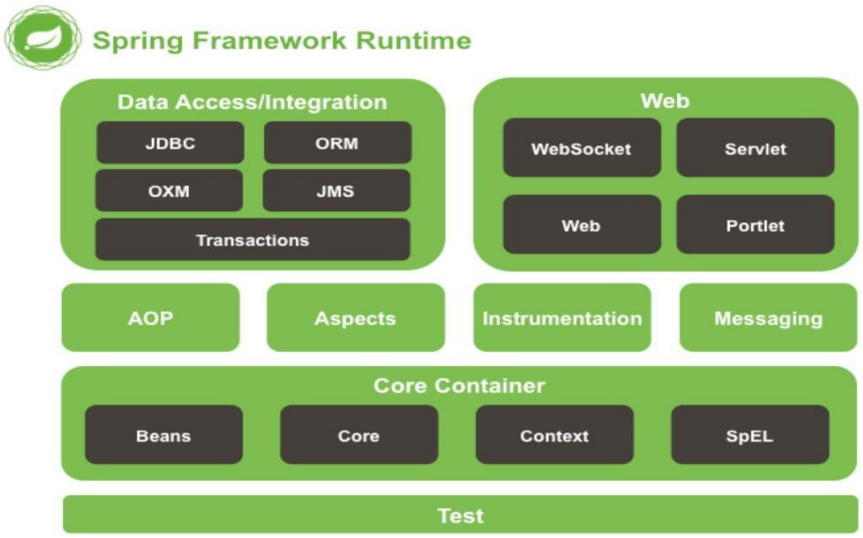

Рисунок 1.9 – Архітектура Spring Framework

Серверна частина є стандартною для багатьох серверних додатків і побудована на широко використовуваній багаторівневій архітектурі.. Компоненти програми організовані у вертикальні шари, причому кожен рівень відповідає за певні аспекти програми, такі як презентація, бізнес-логіка та зберігання даних. Конкретна кількість шарів може змінюватися, але зазвичай використовуються рівні подання, бізнес-логіки, доступу до даних та баз даних (див. Рис. 1.10).

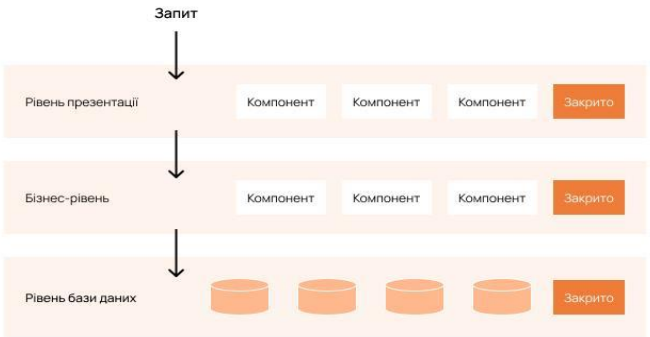

Рисунок 1.10 – Багатошарова архітектура

В якості системи управління базами даних був обраний PostgreSQL. PostgreSQL-це потужна об'єктно-реляційна система з відкритим кодом, яка активно розвивалася протягом багатьох років. Він характеризується високою надійністю, розширеними функціями, високою ефективністю та високою продуктивністю. Існує безліч ресурсів, що пояснюють процес встановлення та використання Postgresql, а також спільнота з відкритим кодом, яка надає безліч корисних матеріалів для ознайомлення з postgresql.

Клієнтська частина реалізована з використанням бібліотеки React (див.розділ React). (див. Рис. 1.11). Щоб відреагувати.js-це фреймворк і бібліотека JavaScript з відкритим кодом, розроблена Facebook. Це дозволяє швидко та ефективно розробляти інтерактивні інтерфейси користувача та веб-додатки, використовуючи менше коду, ніж при використанні чистого JavaScript [10].

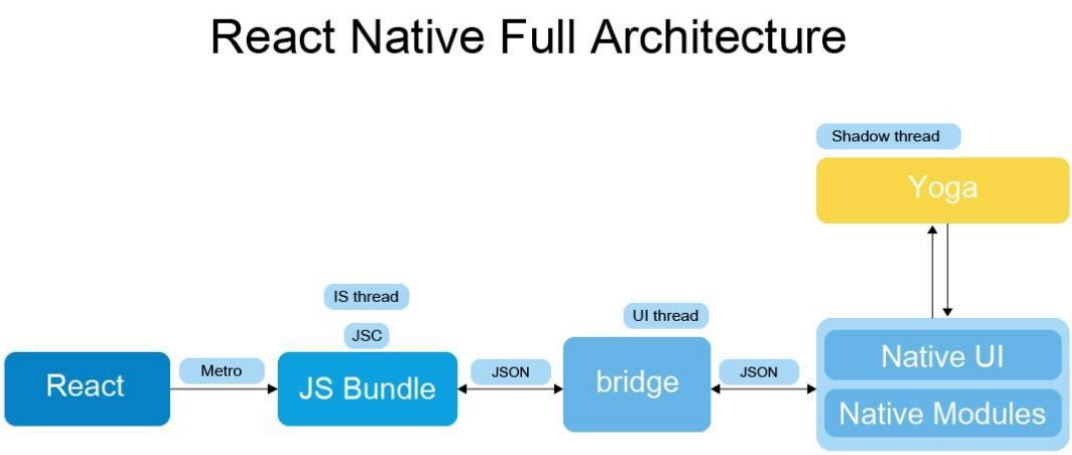

Рисунок 1.11 – Архітектура React

HTML та CSS використовуються для створення розмітки та стилю системи. HTML (мова гіпертекстової розмітки) - це мова розмітки, яка використовується для створення веб-сторінок. Ви можете налаштувати та відобразити вміст веб-сторінки, наприклад текст, зображення, посилання, таблиці та форми. HTML складається з різних елементів, визначених за допомогою тегів.

Кожен елемент HTML має свої особливості. Наприклад, тег<p>відображає абзац тексту,<img>вставляє зображення та<a>створює посилання Теги можуть містити атрибути, які налаштовують зовнішній вигляд і поведінку елемента.

Хоча HTML є основою веб-сторінок, мови програмування, такі як JavaScript, додають динамічність і функціональність, що забезпечує інтерактивність. HTML – це стандарт, який підтримується всіма сучасними веб-браузерами, і він є основою для розробки веб-сторінок і веб-додатків. Існує багато ресурсів і документації, з яких можна дізнатися більше про синтаксис та властивості HTML.

CSS (каскадні таблиці стилів) — це мова стилів, яка використовується для опису зовнішнього вигляду веб-сайту, написаного мовою розмітки HTML.CSS визначає кольори, шрифти, положення елементів та інші властивості, які будуть застосовані до елементів веб-сторінки. За допомогою CSS веб-розробники можуть легко контролювати вигляд і макет веб-сайту [11].

Щоб ефективно використовувати React, контейнер стану зазвичай використовується для управління станом вашої програми. Ресторанна система використовує бібліотеку Redux для уряду штату. (див. Рис. 1.12). Redux-це контейнер із передбачуваним станом, який допомагає писати програми JavaScript, які працюють послідовно в клієнтському, серверному та мобільному середовищах, а також легко тестувати [12].

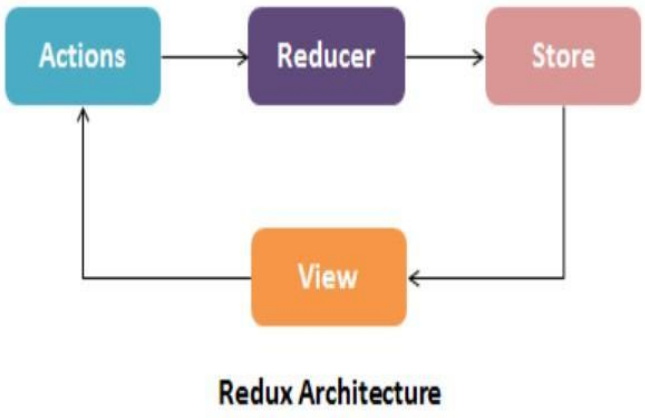

Рисунок 1.12 – Архітектура Redux

Хмарна платформа Heroku буде використовуватися як системний сервер

Heroku — це постачальник хмарних послуг і платформа розробки програмного забезпечення, яка допомагає швидко й ефективно створювати, розгортати та масштабувати веб-додатки.

Він має 140 вбудованих програм, починаючи від сповіщень, інструментів аналітики, служб безпеки, які використовуються для таких цілей, як моніторинг, кешування та електронна пошта, або мережевих доповнень.

Цей інструмент може надати вам інтегровані служби програми миттєвого виконання. Крім того, вам не потрібно думати про інфраструктуру, оскільки нею автоматично керує саме програмне забезпечення. Heroku належить Salesforce.

Основні переваги:

Масштабованість: Heroku полегшує масштабування вашої програми, дозволяючи змінювати розмір ресурсів відповідно до потреб вашої програми. підтримує кілька мов програмування: Heroku підтримує багато різних мов програмування, включаючи Java, Python тощо.

Ви можете використовувати мову, яка найкраще підходить для вашої роботи.

Керування базами даних : Heroku підтримує широкий спектр баз даних, включаючи PostgreSQL, MongoDB та інші.

Просто підключитися та керувати своєю базою даних.

Advanced Features: Heroku спрощує розробку додатків і керування ними, надаючи додаткові інструменти, такі як моніторинг, ведення журналів, керування конфігурацією та доступ до своїх служб [14].

Рисунок 1.13 – Переваги Heroku

Програмне забезпечення JetBrains IntelliJ IDEA є провідним середовищем швидкої розробки для Java.

IntelliJ IDEA — це високотехнологічний комплекс тісно інтегрованих інструментів програмування, включаючи інтелектуальний редактор вихідного коду з розширеними інструментами автоматизації, потужними інструментами рефакторингу коду та вбудованою підтримкою технології J2EE, інтегрованою з тестовими середовищами Ant/JUnit і системами контролю версій .

IntelliJ , унікальний інструмент перевірки та оптимізації коду, а також інноваційний інструмент візуального графічного дизайну Інтерфейс JetBrains Унікальні функції допомагають програмістам полегшити тягар рутинних робочих днів, допомагаючи швидко усувати помилки та покращувати якість коду, підвищуючи продуктивність розробників на висоту рівень.[15].

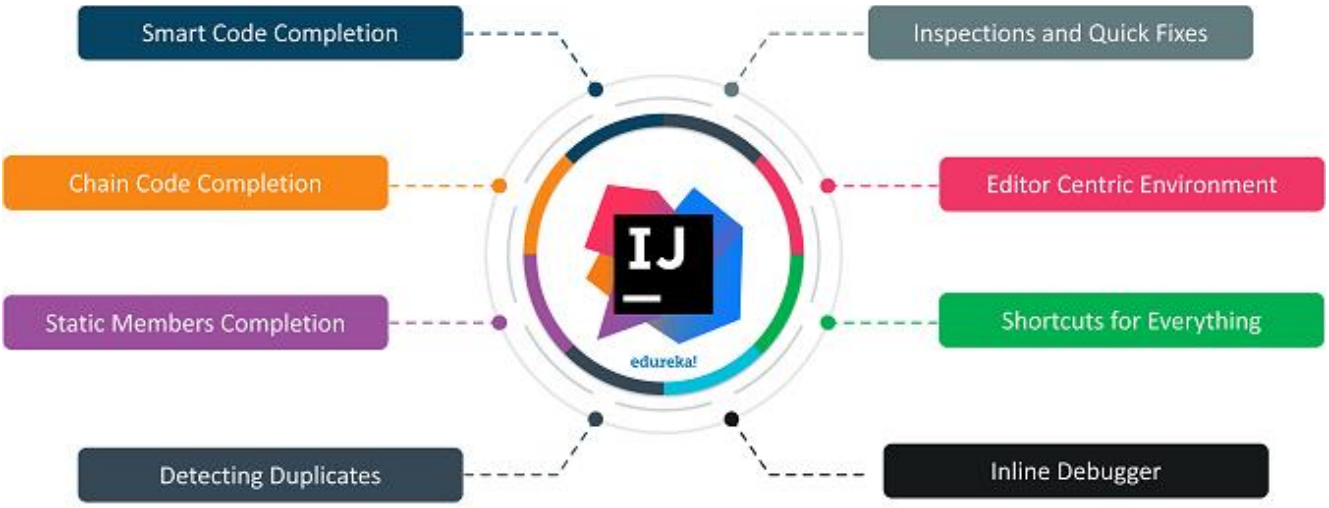

Рисунок 1.14 – Особливості Intellij Idea

Важливість використання Docker і JSON Web Token (JWT) обумовлена швидким розвитком технології контейнеризації і зростаючими вимогами до безпеки і масштабованості веб-додатків. Ці інструменти дозволяють розробникам забезпечувати високий рівень безпеки та гнучкості, значно підвищуючи ефективність розробки, розгортання та управління додатками. Ви також можете використовувати наступні методи:

Docker-це основний інструмент контейнеризації, який дозволяє розробникам упаковувати свої програми та всі їх залежності в контейнери, середовище розробки та продуктивність. Docker допомагає вам уникнути проблем із налаштуванням вашого середовища та гарантує, що ваші програми будуть розгорнуті в будь-якому середовищі, від локальних комп'ютерів до хмарних платформ. Забезпечте швидке та надійне розгортання. Це особливо важливо в контексті сучасних підходів DevOps(див. Рис. 1.15) [18].

Рисунок 1.15 – Docker

Веб-токен JSON (JWT) - це відкритий стандарт токенів, який використовується для безпечної та компактної передачі інформації між сторонами. JWT широко використовується для веб-аутентифікації та авторизації завдяки своїй простоті та ефективності. Це зменшує навантаження на сервер аутентифікації, безпечно передаючи дані про користувачів та інші об'єкти між клієнтом та сервером, а масштабований JWT також дозволяє інтегрувати в розподілену систему, дозволяючи різним службам перевіряти дійсність маркерів без доступу до центрального сервера авторизації. Ви можете це зробити.

Є багато переваг використання Docker та JWT, наприклад::

Портативність та узгодженість: Docker забезпечує однакове середовище для розробки, тестування та підвищення продуктивності, зменшуючи ризик помилок через різницю в конфігурації.

Швидке розгортання: контейнери Docker можна швидко створювати, знищувати та відновлювати, що дає вам гнучкість в управлінні вашими програмами..

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1. Проектування бази даних

У цьому додатку необхідно мати користувачів для обмеження доступу до певних функцій, для цього створимо першу сутність - users. Кожен користувач має мати логін та пароль, представлені у вигляді рядків, а також роль, представлену у вигляді сутності roles.

Для зберігання додаткової інформації про звичайних користувачів створимо сутність owners. Ця сутність буде містити атрибути: ім’я, email та номер телефону, представлені як рядкові типи даних із фіксованою довжиною. Кожен власник повинен мати машину, тому введемо сутність car. Як атрибути ця сутність буде містити бренд, назву та рік випуску, представлені рядковими типами даних.

Для створення замовлення введемо сутність order. Замовлення повинно складатися з опису, необхідних послуг та речей, необхідних для ремонту. Тому введемо ще дві сутності: favor та goods. Вони повинні містити назву, ціну та опис, представлені рядковими типами даних.

Для кожної послуги необхідно додати відповідного майстра, який буде виконувати цю послугу, тому введемо сутність repairman (див. рис. 2.1).

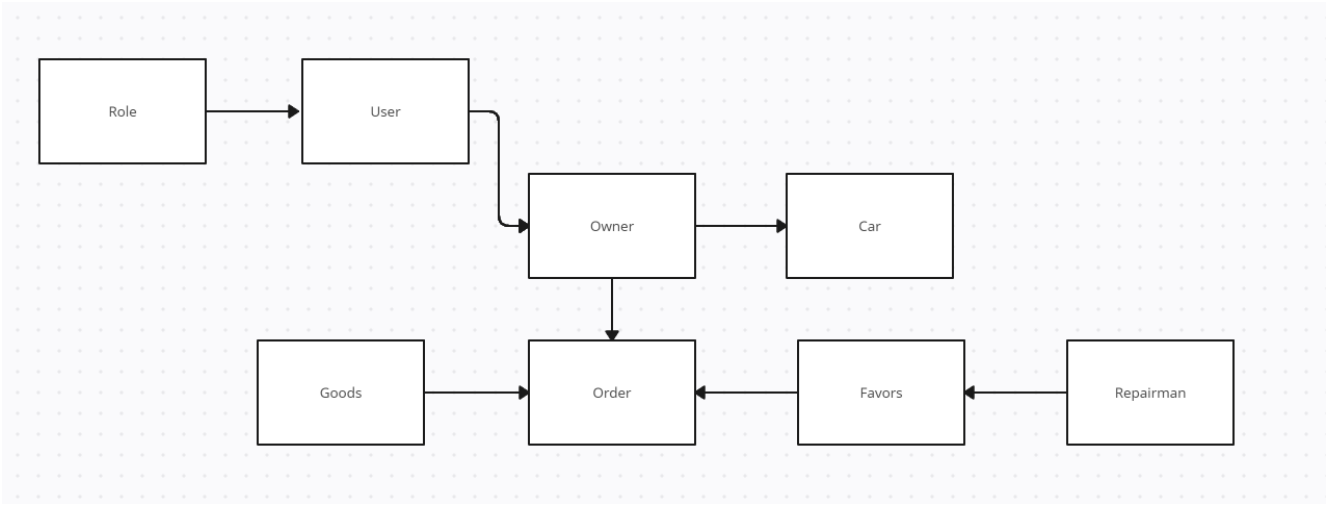

Рисунок 2.1 – Схема бази даних на рівні концепцій

Користувач може мати декілька декілька ролей, та роль може бути декілька користувачів (див. рис. 2.2).

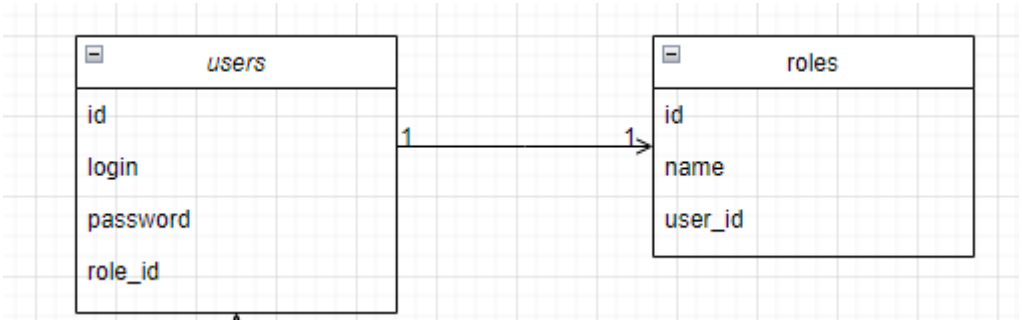

Рисунок 2.2 – Зв’язок users-roles

Кожен власник повинен мати машину, для подальшого сторення замовлень. Оскільки замовлення складається з різних послуг, було вирішено, що одне замовлення може мати кілька послуг тому між ними зв’язок один до багатьох. А також одне замовлення може мати кілька речей тому між ними зв’язок один до багатьох (див. рис. 2.3).

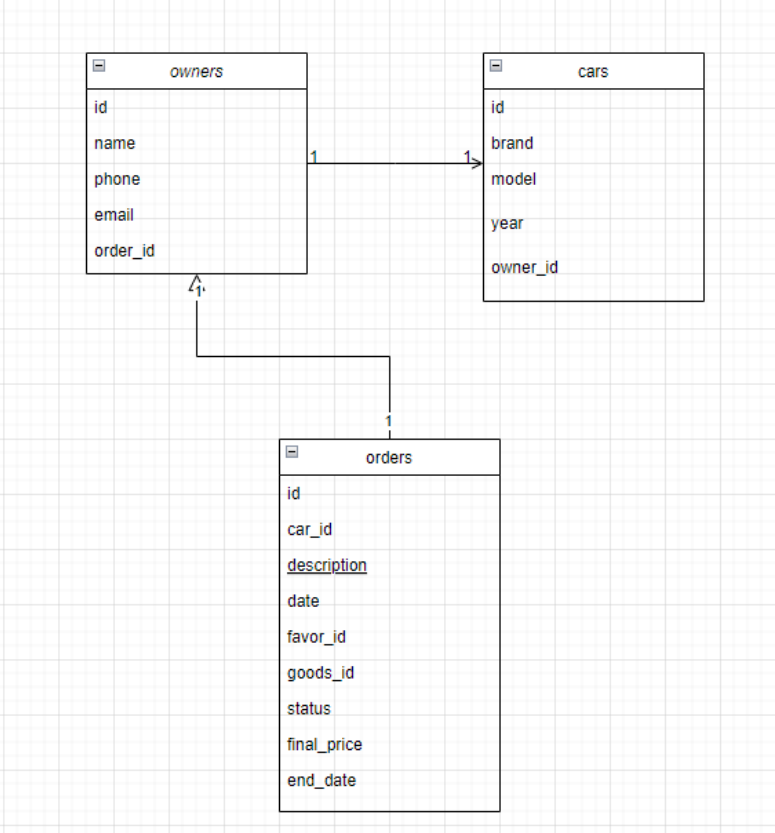

Рисунок 2.3 – Зв’язок owners-cars-orders

Замовлення повинно містити список послуг та речей для подальшого створення замовлень. Оскільки послуги є частиною замовлення, було вирішено, що одне замовлення може мати кілька послуг, тому між ними існує зв’язок "один до багатьох". Аналогічно, речі є частиною замовлення, і одне замовлення може мати кілька речей, тому між ними також існує зв’язок "один до багатьох". Таким чином, замовлення може містити багато послуг та речей, встановлюючи зв’язок "один до багатьох" (див. рис. 2.4).

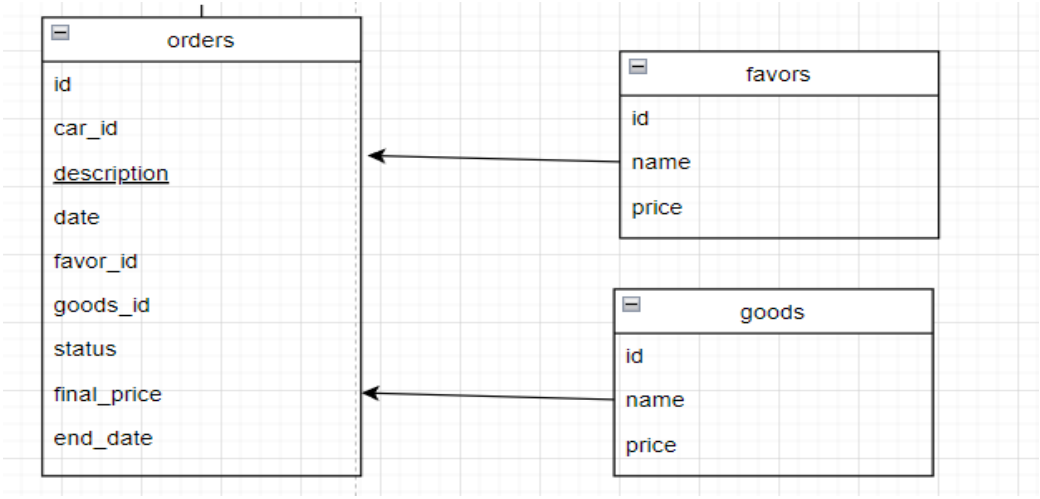

Рисунок 2.4 – Зв’язок orders-favors-goods

Кожна послуга повинна містити майстра який буде її виконувати, оскільки кожен майстра може виконувати декілька послуг, то зв’язок буде один до багатьох (див. рис. 2.5).

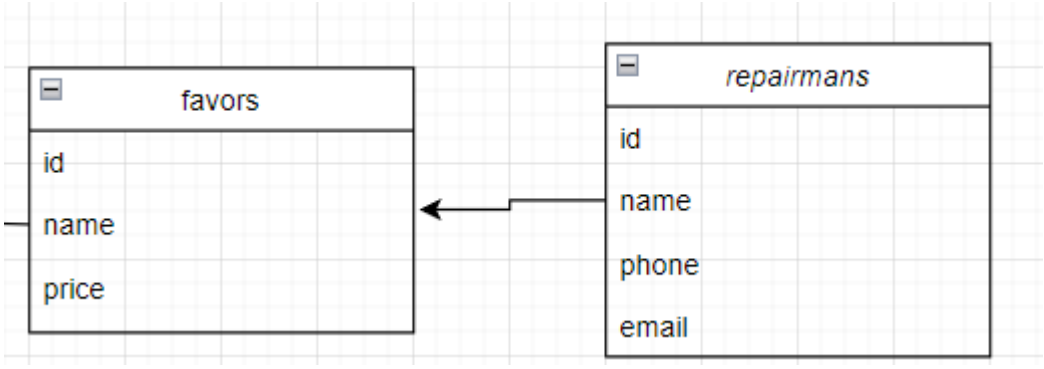

Рисунок 2.5 – Зв’язок favors-repairmans

Логічна модель бази даних в повному обсязі представлена на рисунку 2.6.

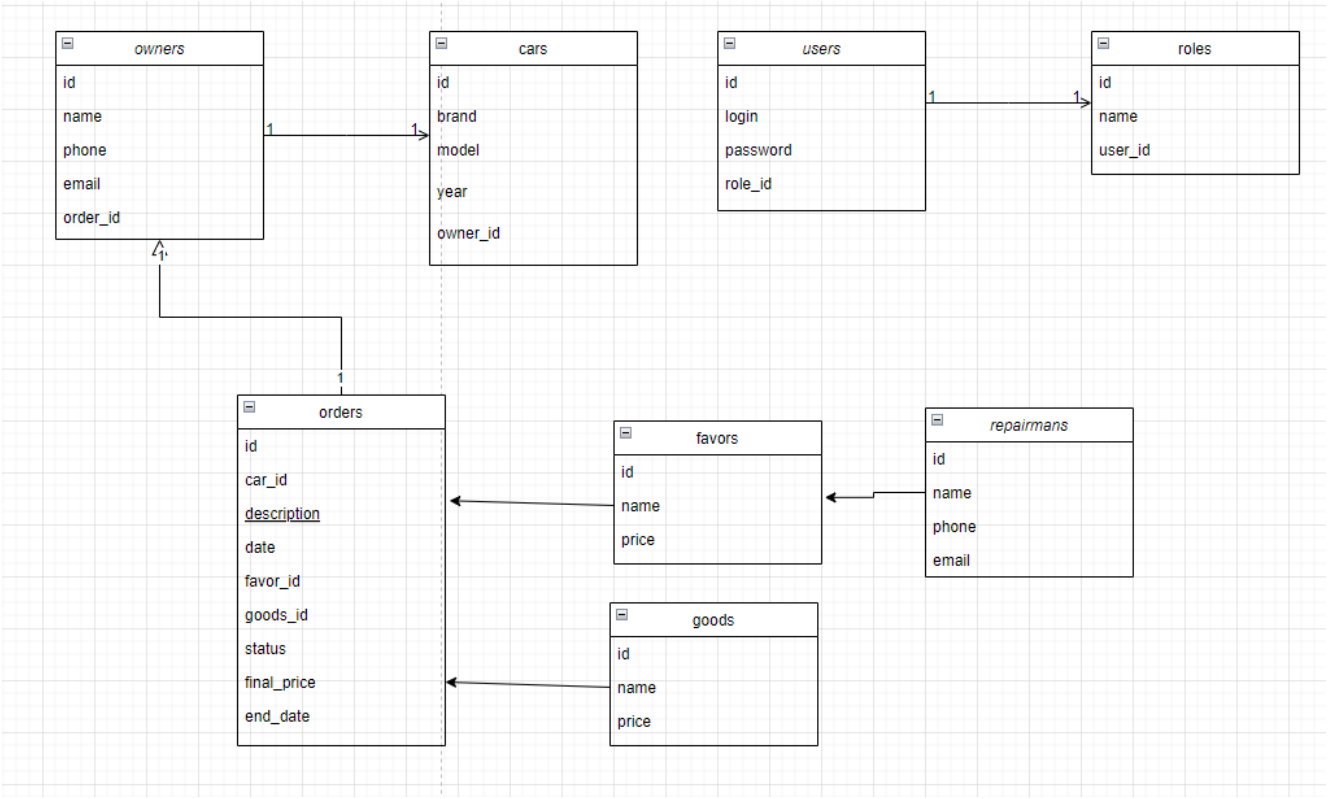

Рисунок 2.6 – Модель бази даних на рівні логіки

2.2. Проектування архітектури системи

Отримавши всю інформацію про систему з першої частини, ми можемо почати проектувати архітектуру системи, включаючи визначення рівнів, які вона містить.Після визначення всіх класів у системі за допомогою UML створюється діаграма класів.

Діаграми класів — це графічні інструменти, які використовуються для статичного відображення структури моделі системи на основі об’єктно-орієнтованого програмування.

Він надає візуальне представлення класів, типів даних, зв’язків і зв’язків між ними. Діаграма класів дозволяє візуалізувати не тільки сам клас і його властивості, а й їхні інтерфейси, об’єкти та поведінку.

У програмуванні класи — це базові елементи, які визначають шаблони для створення об’єктів.Він служить основою для створення екземпляра об’єкта з певним делегатом на основі визначення класу. Класи є важливою концепцією в об’єктно-орієнтованому програмуванні (ООП), парадигмі програмування, яка зосереджена на організації коду в багаторазово використовувані об’єкти.

Це забезпечує модульну, структуровану та ефективну розробку програмного забезпечення, полегшуючи повторне використання, інкапсуляцію та абстракцію цього коду.Створюючи кілька екземплярів класу, ми можемо створювати об’єкти зі спільними властивостями та поведінкою, визначеними класом.

Кожен об’єкт може мати власний стан (значення атрибутів) і може незалежно виконувати методи, визначені класом.Можна використовувати класи для моделювання абстракцій реальних об’єктів, систем або програмного забезпечення.Вони допомагають редагувати код, спрощують повторне використання коду та забезпечують чітку структуру для розробки та впровадження складних систем.

Для серверної частини була використано поєднання багаторівневої архітектури та MVC(Model-View-Controller). Цей архітектурний підхід є стандартним для багатьох серверних додатків і широко використовується. Багаторівнева архітектура відповідає за комунікаційну та організаційну структуру при розробці програмного забезпечення, що робить її природним вибором для багатьох бізнес-додатків. Перевага цього підходу також полягає в тому, що його легко розробляти та тестувати.

MVC (Model-View-Controller) – це шаблон проектування, що розділяє програму на три взаємодіючі компоненти: модель, представлення і контролер. Такий підхід покращує організацію коду, полегшує підтримку та тестування програмного забезпечення.

Модель (Model):

Відповідає за управління даними та бізнес-логікою. Забезпечує доступ до даних, їх зберігання, обробку і правила. Наприклад, клас, що представляє користувача, та методи для збереження, оновлення, видалення даних користувача.

Представлення (View):

Відповідає за відображення даних користувачу. Отримує дані з моделі і формує інтерфейс для користувача. Наприклад, HTML-шаблон, що відображає дані користувача у веб-браузері.

Контролер (Controller):

Відповідає за обробку користувацьких запитів і взаємодію між моделлю і представленням.Отримує вхідні дані від користувача, обробляє їх, оновлює модель і вибирає відповідне представлення. Наприклад, метод в контролері, що обробляє запит на створення нового користувача, взаємодіє з моделлю для збереження даних і вибирає представлення для підтвердження успішного створення.

Основні переваги MVC полягають у розділення відповідальностей: Кожен компонент відповідає за свою частину функціональності, що полегшує розробку і підтримку. Також у модульності, легко змінювати і оновлювати окремі компоненти без впливу на інші.

У багаторівневій архітектурі компоненти програми організовані у вертикальні шари, кожен з яких виконує функції рівня та відповідає за певні аспекти програми, такі як презентація, бізнес-логіка та зберігання даних. Хоча кількість рівнів може відрізнятися від програми до програми, важливо зазначити, що зазвичай використовуються стандартні рівні, такі як подання, бізнес-логіка, доступ до даних та бази даних.

У невеликих додатках використовується менше шарів, тоді як складні проекти можуть мати додаткові шари залежно від потреб програми [18].

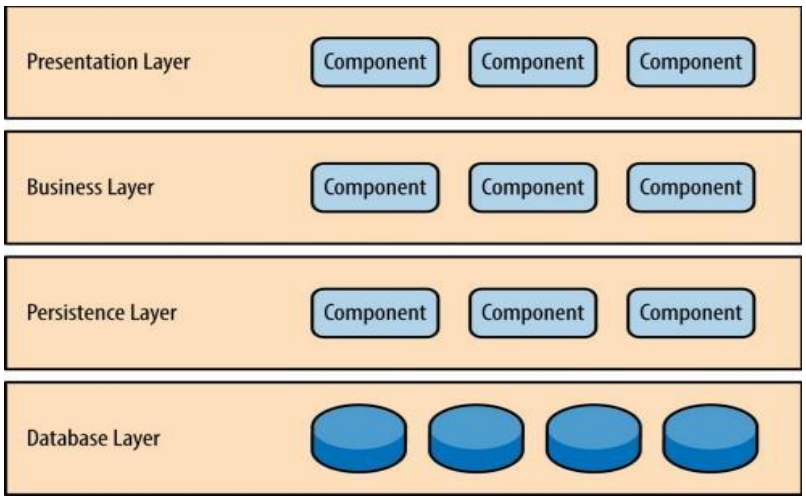

Рисунок 2.7 – Багаторівнева архітектура

На рівні бази даних дані зберігаються та обробляються, і більша частина логіки, пов'язаної з бізнес-процесом, розміщується. Це дозволяє відокремити логічні структури даних від фізичних структур даних, що допомагає підвищити гнучкість і підтримувати масштабованість системи.

Структура класів представлена на рисунку 2.8.

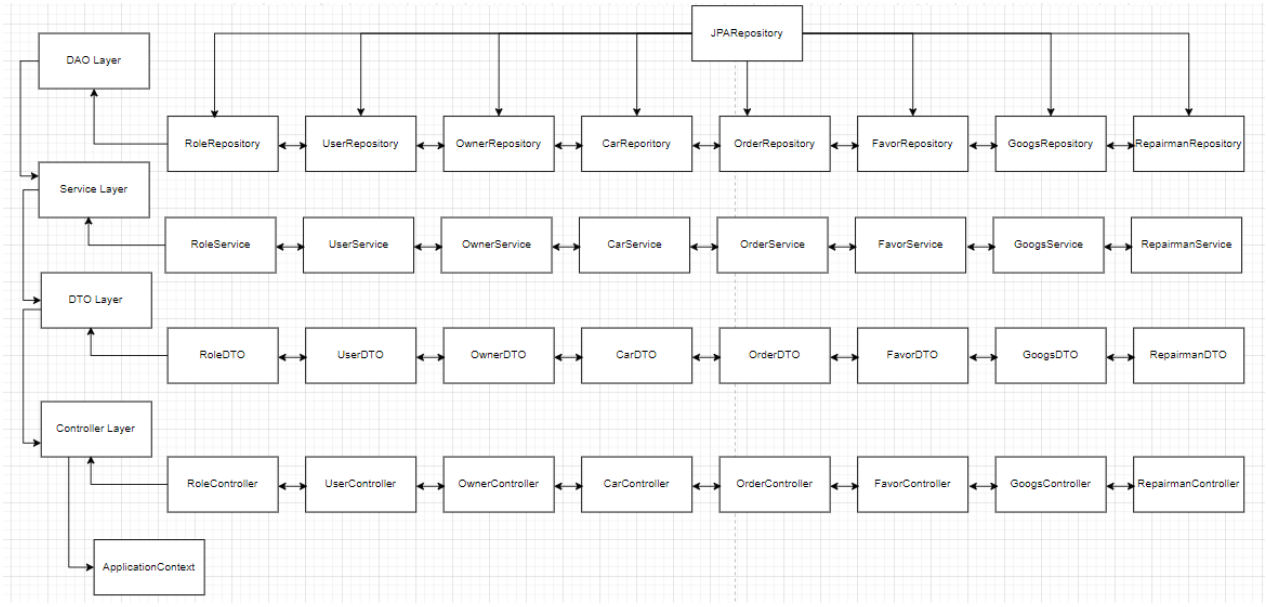

Рисунок 2.8 – Діаграма класів серверного додатку

Архітектура клієнтської частини представлена на рисунку 2.9.

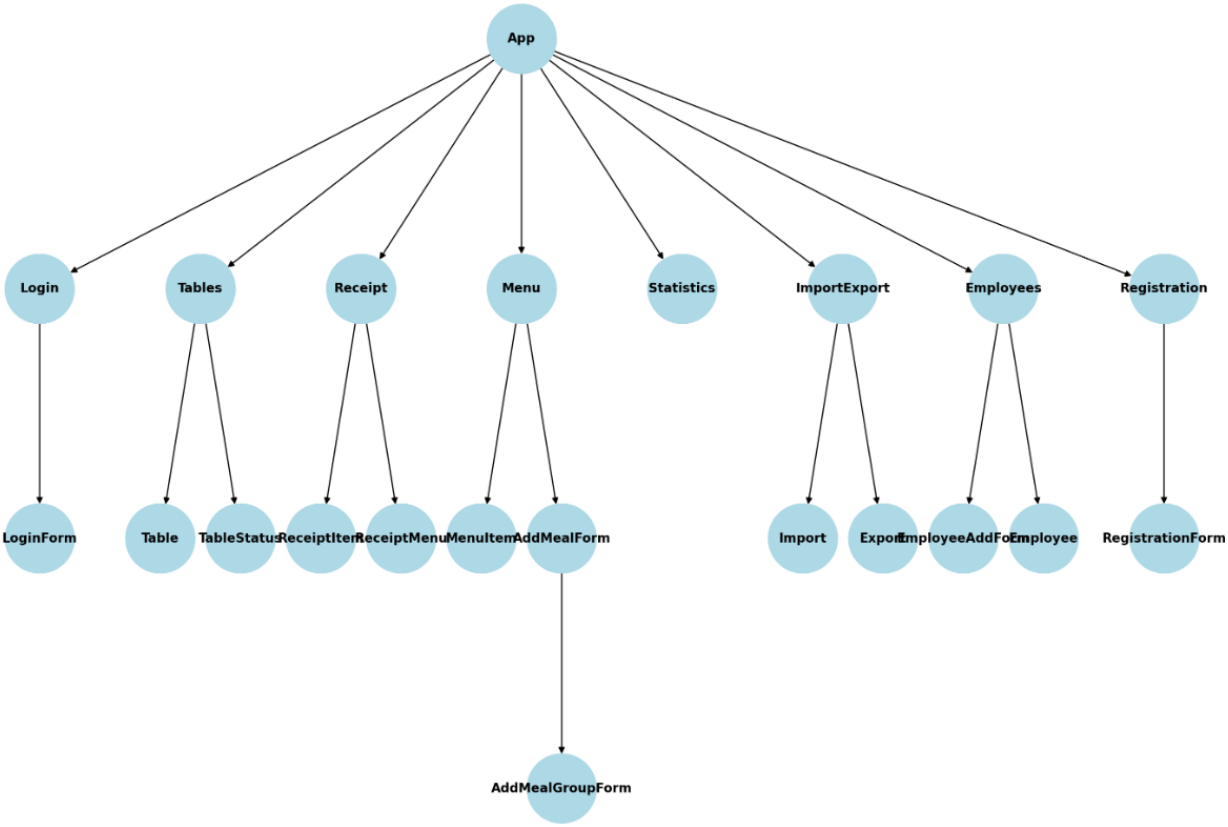

Рисунок 2.9 – Діаграма компонентів клієнтської частини

3 КОНСТРУЮВАННЯ

3.1 Реалізація класів

Лістинг коду для автоматичної зміни статусу замовлення на рисунку 3.1.

```
@Override 1 usage  Yaroslav Lapovets
public Order changeOrderStatus(Long id, Order.Status status) {
    Order order = orderService.getById(id);
    order.setStatus(status);
    List<Repairman> repairmanList;
    switch (status) {
        case COMPLETED: {
            repairmanList = repairmanService.getAllRepairmensByOrderId(id);
            saveData(order, repairmanList);
            break;
        }
        case FAILURE: {
            repairmanList = changeToFailure(order);
            saveData(order, repairmanList);
            break;
        }
        default:
    }
    return order;
}

private List<Repairman> changeToFailure(Order order) {  1 usage  Yaroslav Lapovets
    Favor priceFavor = order.getFavors().stream()  Stream<Favor>
            .findFirst()  Optional<Favor>
            .get();
    priceFavor.setPrice(BigDecimal.valueOf(500));
    return List.of(priceFavor.getRepairman());
}
```

Рисунок 3.1 – Метод зміни статусу замовлення

Лістинг коду для обрахування ціни послуг на рисунку 3.2.

```
private BigDecimal getFavorsPrice(Order order, int bonus) {  1 usage  Yaroslav Lapovets
    order.getFavors().forEach(f -> f.setPrice(f.getPrice()
            .subtract(f.getPrice()
                    .multiply(BigDecimal.valueOf(bonus * FAVOR_DISCOUNT)))));
    return order.getFavors().stream()  Stream<Favor>
            .map(Favor::getPrice)  Stream<BigDecimal>
            .reduce(BigDecimal.ZERO, BigDecimal::add);
}
```

Рисунок 3.2 – Метод для обрахування ціни послуг

Лістинг коду для обрахування ціни деталей на рисунку 3.3.

```
private BigDecimal getGoodsPrice(Order order, int bonus) {  1 usage  Yaroslav Lapovets
    order.getGoods()
            .forEach(g -> g.setPrice(g.getPrice()
                    .subtract(g.getPrice()
                            .multiply(BigDecimal.valueOf(bonus * GOODS_DISCOUNT)))));
    return order.getGoods().stream() Stream<Goods>
            .map(Goods::getPrice) Stream<BigDecimal>
            .reduce(BigDecimal.ZERO, BigDecimal::add);
}
```

Рисунок 3.3 – Метод для обрахування ціни деталей

Лістинг коду для надання ролі USER для нових користувачів на рисунку 3.4.

```
@Override  3 usages
public Collection<? extends org.springframework.security.core.GrantedAuthority> getAuthorities() {
    return user.getRoles().stream() Stream<Role>
            .map(role ->
                    new org.springframework.security.core.authority.SimpleGrantedAuthority(
                            role: "ROLE_" + role.getName())) Stream<SimpleGrantedAuthority>
            .collect(Collectors.toSet());
}
```

Рисунок 3.4 – Метод для надання ролі

Лістинг коду для обрахунку зарплати майстра на основі виконаних послуг у завершених замовленнях на рисунку 3.5.

```
@Override  1 usage  Yaroslav Lapovets
public BigDecimal getSalaryByRepairmanIdAndOrderId(Long repairmanId, Long orderId) {
    List<Favor> favors = favorService.getFavorsByRepairmanIdAndOrderId(repairmanId, orderId);
    favors.forEach(f -> statusService.changeFavorStatus(f.getId(),Favor.Status.PAID));
    return favors.stream() Stream<Favor>
            .map(f -> f.getPrice()
                    .multiply(REPAIRMAN_DISCOUNT)) Stream<BigDecimal>
            .reduce(BigDecimal.ZERO, BigDecimal::add);
}
```

Рисунок 3.5 – Метод для обрахунку зарплати майстра

Загальна структура проекту зображена на рисунках 3.6-3.8.

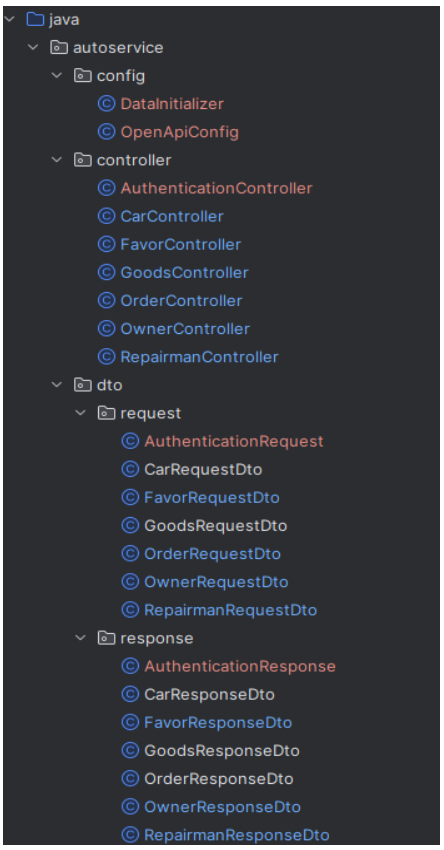

Рисунок 3.6 – Структура проекту

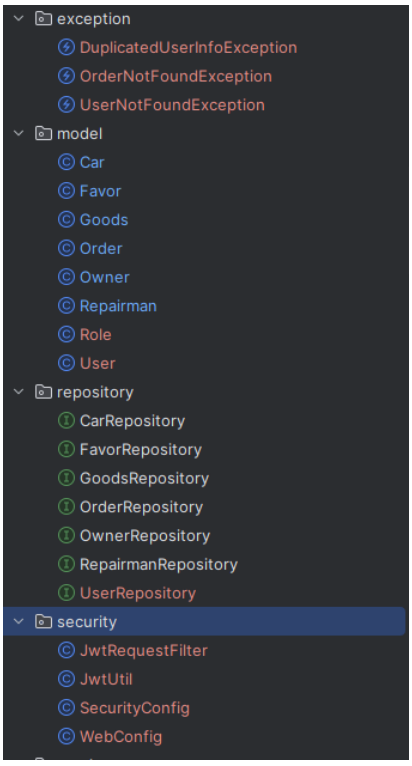

Рисунок 3.7 – Структура проекту

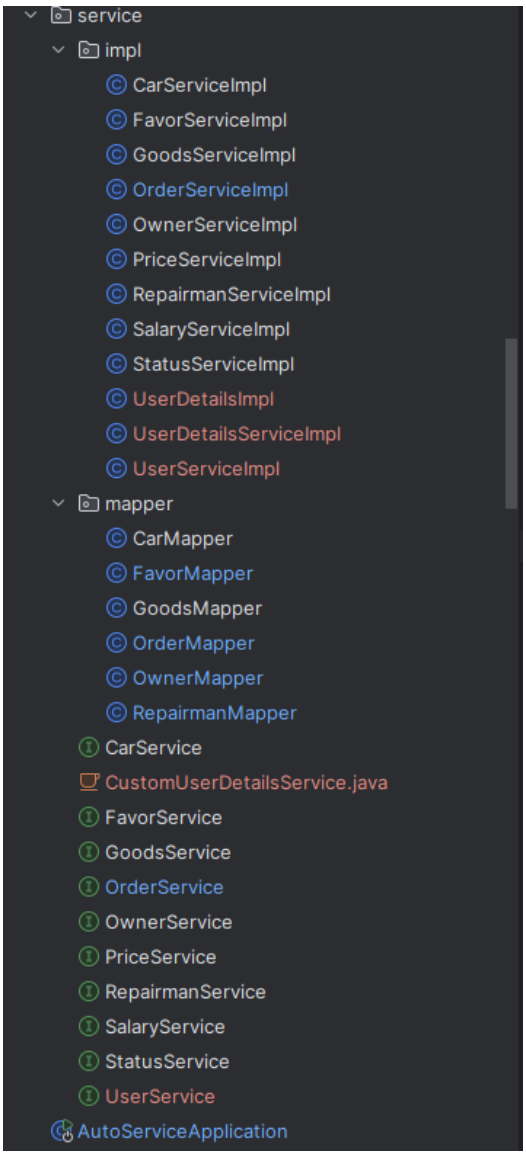

Рисунок 3.8 – Структура проекту

3.2 Забезпечення якості

Тестування - це процес, який дозволяє оцінити якість продукту, що розробляється.

Продукт високої якості повинен відповідати вимогам, які до нього пред'являються.

Програмна система повинна реалізовувати всі необхідні варіанти використання та бути безпомилковою.

Крім того, система програмного забезпечення має бути надійною (без збоїв чи збоїв), безпечною, забезпечувати необхідну продуктивність, простою у використанні та масштабованою.

Отже, тестування – це процес аналізу програмної системи з метою виявлення дефектів та оцінки її характеристик [19].

Завдяки технологічному прогресу інтерфейс користувача стає все більш важливим елементом у проектах.

Дійсно, успіх служби чи веб-сайту значною мірою залежить від задоволеності користувачів та їх взаємодії з інтерфейсом.

Наявність зручного та привабливого інтерфейсу сприяє популярності та успіху проекту, адже користувачі частіше перебувають на сайті та активно взаємодіють із сервісом.

Відгуки та відповіді користувачів допомагають покращити інтерфейс і адаптувати його до потреб і очікувань громадськості.

Тому якісний інтерфейс користувача є важливим чинником популярності та успіху проекту.

User API, також відомий як UI API або UI API, відноситься до набору інтерфейсів або методів програмування, які дозволяють розробникам взаємодіяти та змінювати користувацькі інтерфейси програми або системи.

Він надає спосіб створювати, змінювати та контролювати візуальні та поведінкові елементи інтерфейсу користувача.

Використовуючи спеціальні API, розробники можуть отримати доступ до різних елементів інтерфейсу користувача, таких як кнопки, форми, меню, діалогові вікна та інші інтерактивні елементи.

Вони можуть програмно визначати зовнішній вигляд, розташування та поведінку цих компонентів, а також обробляти введені користувачем дані та реагувати на події, спричинені діями користувача.

Спеціальні API усувають складність взаємодії з інтерфейсом користувача, забезпечуючи простий і стандартизований спосіб роботи з компонентами інтерфейсу.

Зазвичай він включає методи створення та керування елементами інтерфейсу, налаштування властивостей і стилів, керування взаємодією користувача та оновлення інтерфейсу на основі змін у логіці чи функціональності програми.

Використовуючи користувальницькі API, розробники можуть створювати багаті інтерактивні інтерфейси користувача, налаштовувати зовнішній вигляд і поведінку своїх програм і забезпечувати безперебійну взаємодію між користувачами та базовою системою або програмою [20].

При розробці інтерфейсу цього проекту особлива увага приділялася зручності та простоті використання сервісу без проблем.

Основними компонентами інтерфейсу програмної системи є: Під час доступу до сайту користувачі мають 2 варіанти: авторизуватися в системі та зареєструватися в системі.

Сторінка авторизації автоматично відображається користувачам, коли вони відвідують веб-сайт.

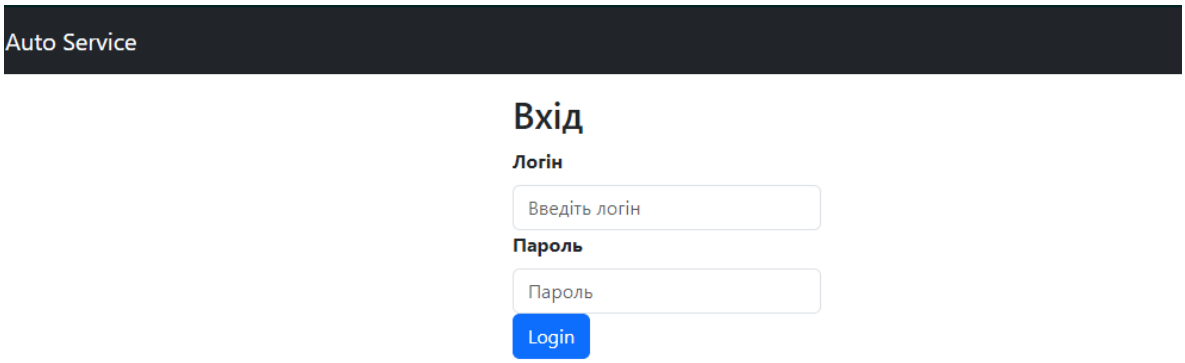

Рисунок 3.9 – Сторінка авторизації

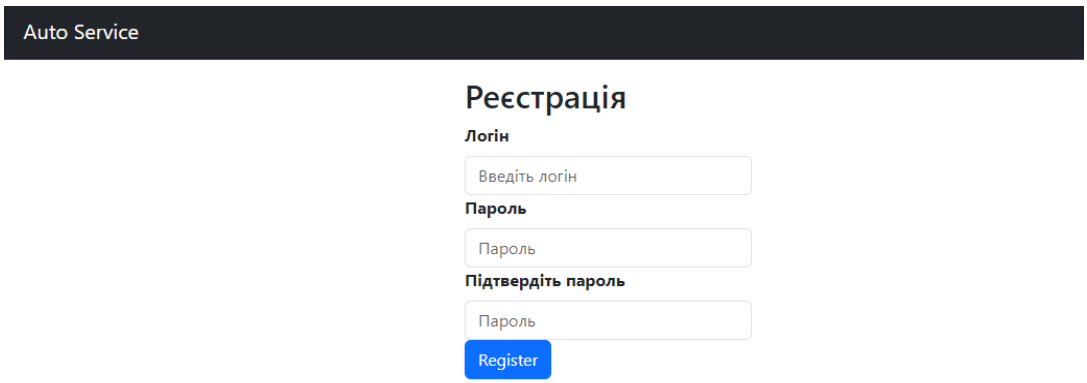

Рисунок 3.10 – Сторінка реєстрації

Щоб користувачі могли використовувати функції програми, вони повинні мати ліцензію. Якщо у користувача немає облікового запису, він повинен спочатку зареєструватися. Кожен зареєстрований користувач автоматично стає простим користувачем системи.

Адміністратори створюються безпосередньо в базі даних і можуть створювати додаткові облікові записи для співробітників з обмеженою функціональністю.

При успішній авторизації адміністратор перенаправляється на сторінку, де він може вибрати необхідну вкладку. У верхній частині сайту розміщено навігаційне меню з різними посиланнями, такими як: "Профіль", "Власники", "Автомобілі", "Послуги", "Товари", "Замовлення", "Майстри". Перейшовши на сторінку "Власники", адміністратор може створювати, редагувати та видаляти власників машин. У списку праворуч відображені всі власники машин, у яких є активні замовлення, як показано на рисунку 3.11.

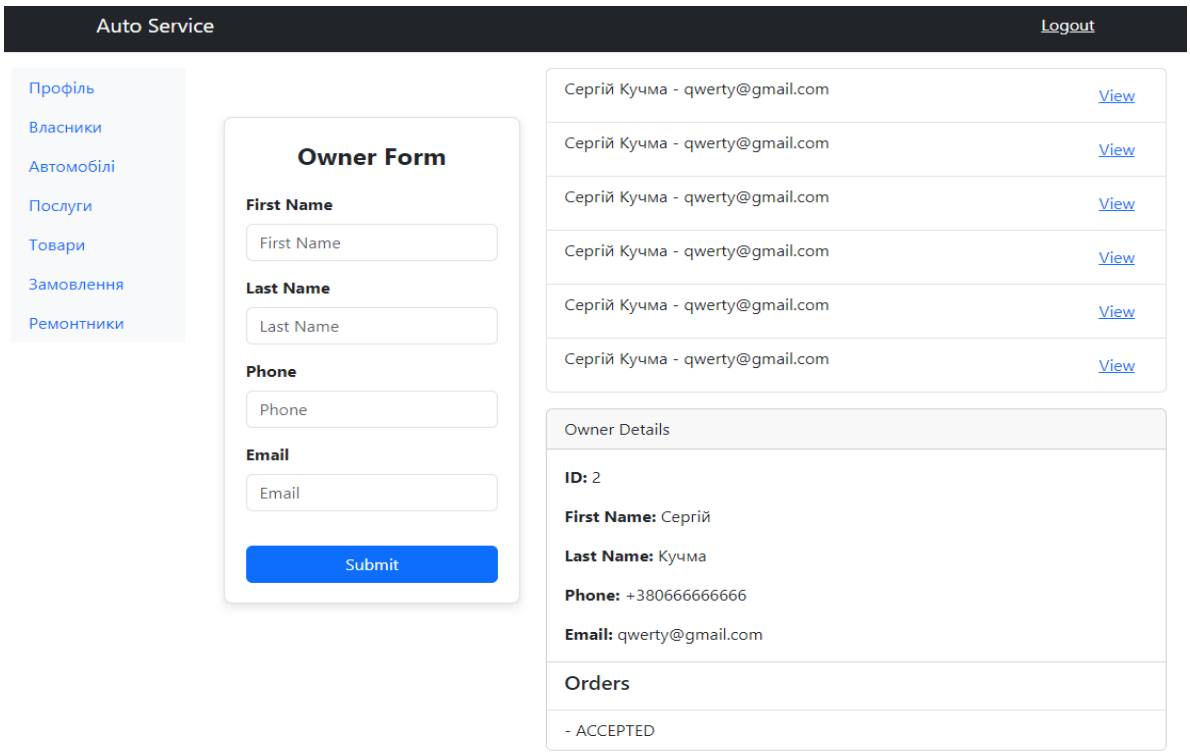

Рисунок 3.11 – сторінка Власники

Перейдемо на сторінку Автомобілі, де зможемо створити, редагувати, видалити машини, також у списку справа зображені усі машини у яких активні замовлення, зображено на рисунку 3.12

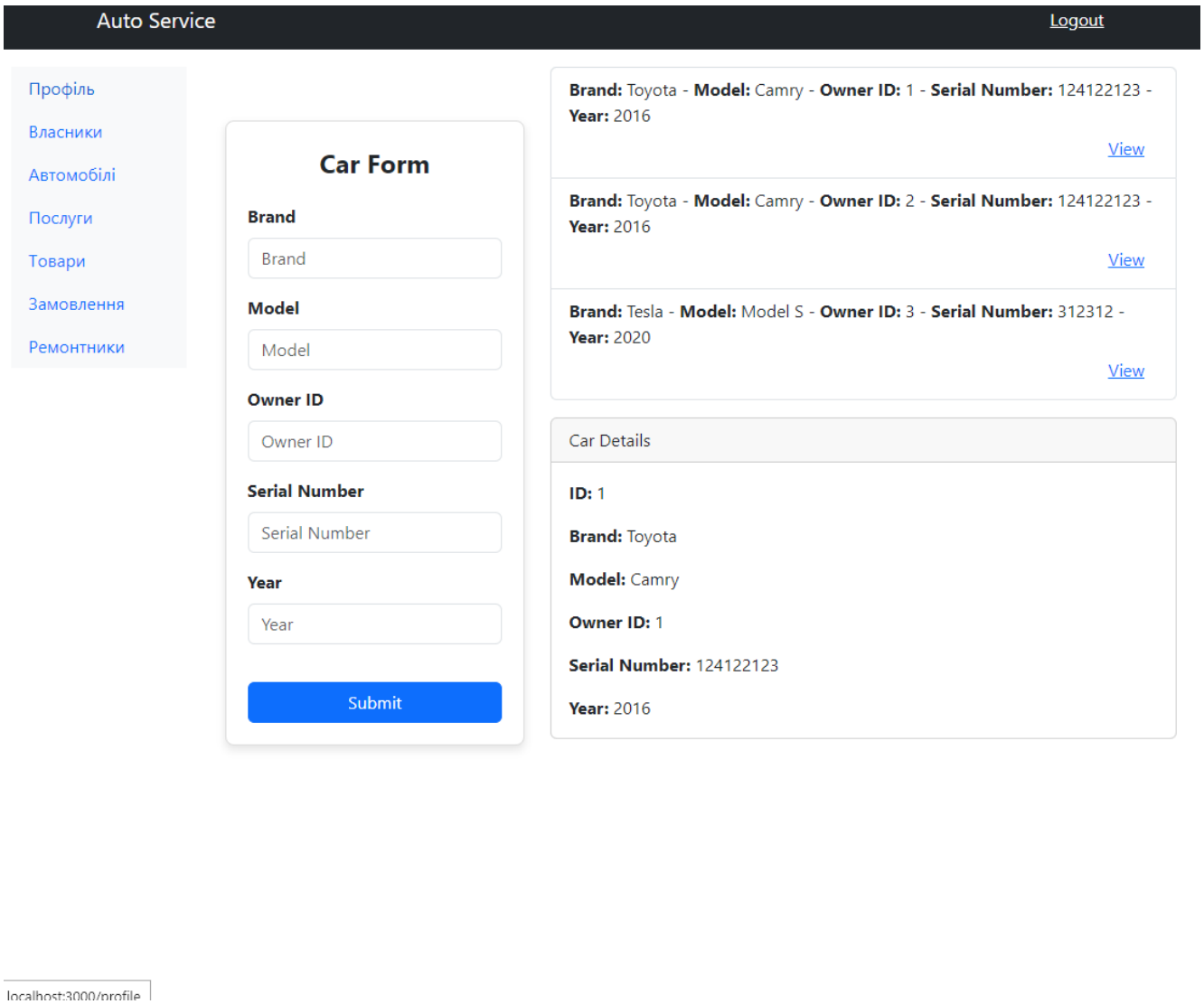

Рисунок 3.12 – сторінка Автомобілі

Перейдемо на сторінку Послуги, де зможемо створити, редагувати, видалити послуги, які надає автосервіс по ремонту машин, також у списку справа зображені усі послуги, також їх можна фільтрувати по назві, наведені тільки ті у яких активні замовлення, зображено на рисунку 3.13

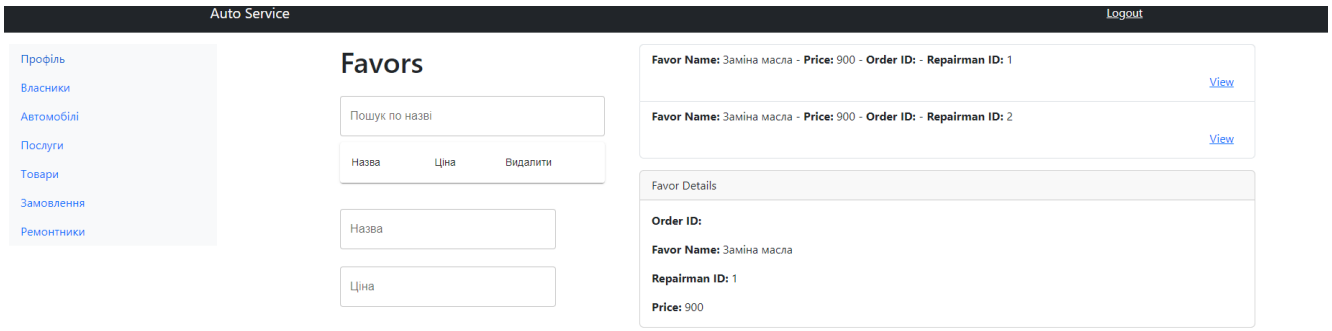

Рисунок 3.13 – сторінка Послуги

Перейдемо на сторінку Товари, де зможемо створити, редагувати, видалити товари необхідні для ремонту, також у списку справа зображені усі товари у активних замовленнях, зображено на рисунку 3.14

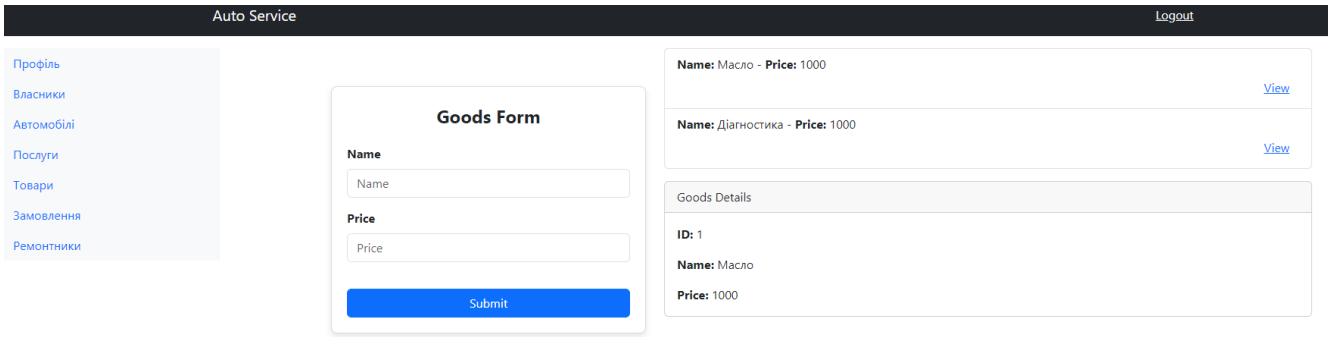

Рисунок 3.14 – сторінка Товари

Перейдемо на сторінку Замовлення, де зможемо створити, редагувати, видалити замовлення, також у списку справа зображені усі активні замовлення, та відповідні послуги та товари які притаманні замовленню, зображено на рисунку 3.15

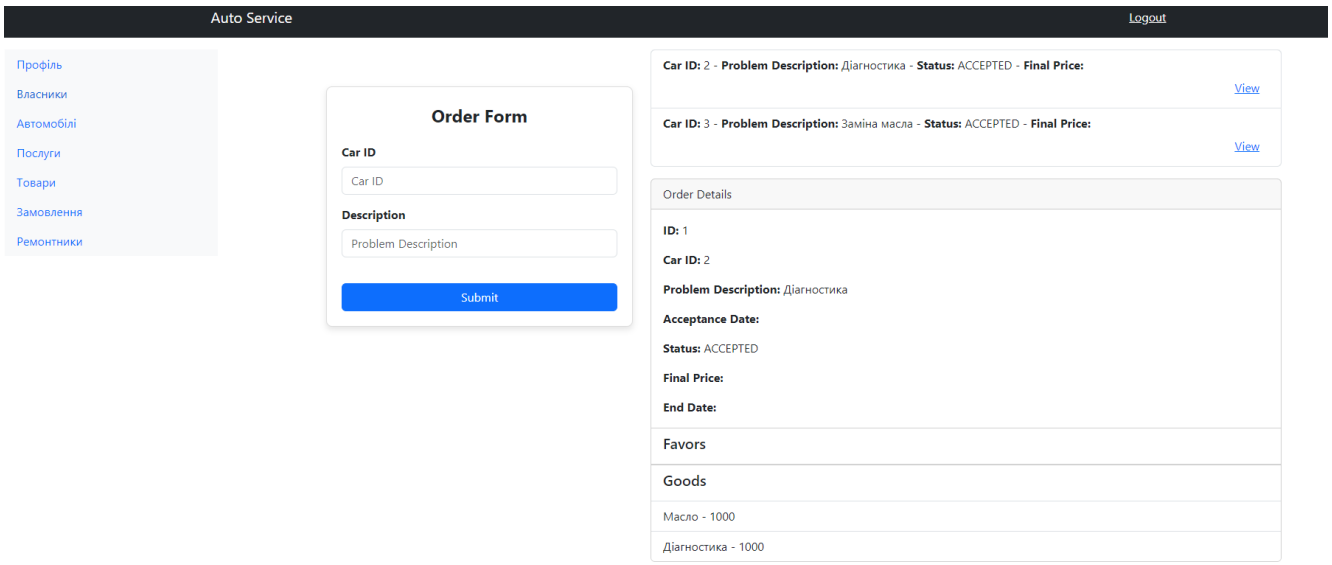

Рисунок 3.15 – сторінка Замовлення

Перейдемо на сторінку Майстри, де зможемо створити, редагувати, майстрів, які працюють у автосервісі, також у списку справа зображено усіх майстрів, які зараз працюють і також відображені усі завершені замовлення протягом дня, зображено на рисунку 3.16

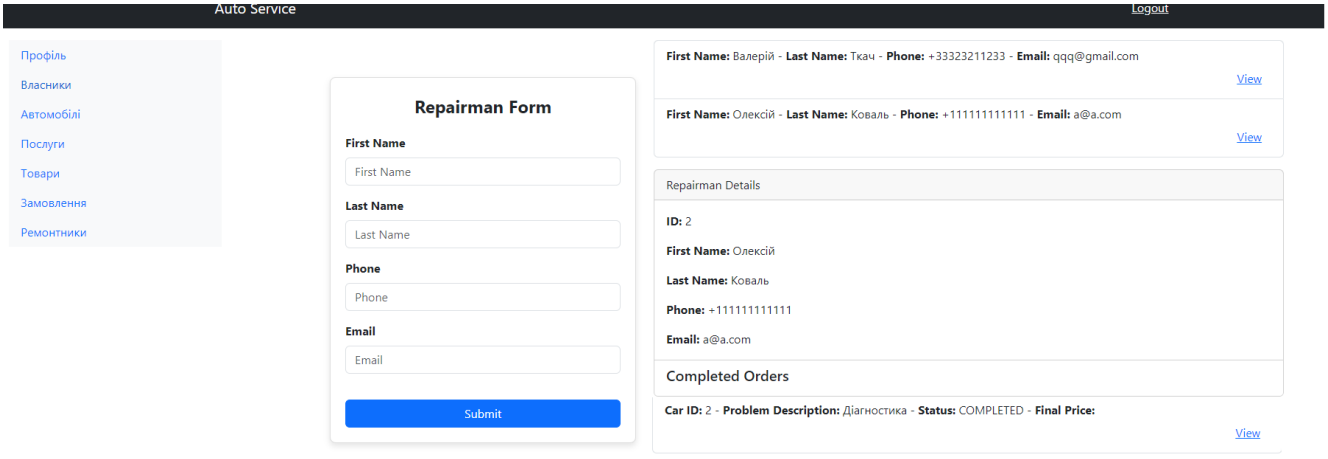

Рисунок 3.16 – сторінка Майстри

Також адміністратор може перейти у персональний кабінет на сторінку Профіль, де можна побачити список усіх машин, які у даних момент ремонтують, та список усіх замовлень, та їх поточний статус, зображено на рисунку 3.17

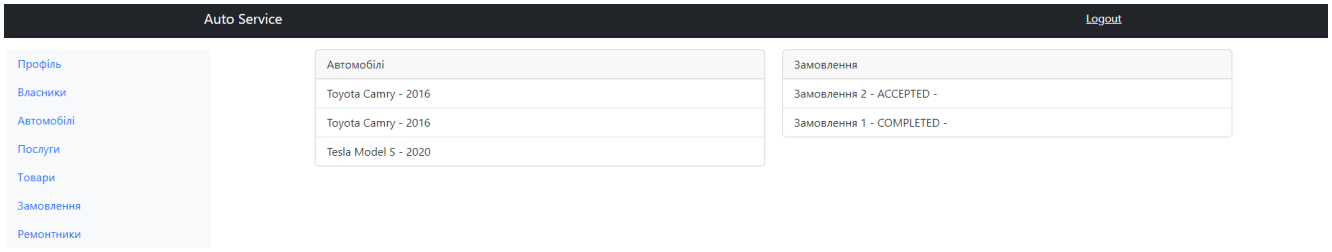

Рисунок 3.17 – сторінка Профіль

Отже, цей веб-сайт надає зручний інтерфейс для управління різними аспектами автосервісу. Він включає розділи для профілів користувачів, власників, автомобілів, послуг, товарів, замовлень і майстрів. Використання React та Bootstrap дозволяє створити інтерактивний і сучасний інтерфейс, що забезпечує легкий доступ до всіх функціональних можливостей системи. Надійність та зручність експлуатації підвищують ефективність роботи та задоволеність користувачів.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при ушкоджені м’яких тканин, суглобів і кісток

Долікарська допомога – це комплекс медичних заходів, спрямованих на надання медичної допомоги при невідкладних станах, які стаються на виробництві, у побуті, під час дорожньо-транспортних пригод, катастроф, техногенних аварій та при гострих неврологічних, терапевтичних, хірургічних та термінальних станах. Не надання долікарської допомоги при нещасних випадках, раптових гострих захворюваннях людини призводить до тяжких наслідків, аж до летальних. Своєчасна допомога відіграє важливу роль у подальшому лікуванні потерпілих і хворих, сприяє скороченню термінів їх медичної та трудової реабілітації [23].

Наслідком пошкодження м'яких тканин є кровотеча. Найчастіше пошкодження м'яких тканин відбувається внаслідок удару. Іншими ознаками удару м'яких тканин є болі в місці удару, розлита припухлість (крововилив), обмежена рухомість ушкодженої частини тіла

Перша допомога у випадку пошкодження м'яких тканин за умови відсутності в цьому місці перелому чи вивиху — холод на місце удару (рушник, змочений холодною водою, пузир з льодом чи снігом) та туга пов'язка на місце крововиливу і спокій ушкодженій частині тіла [24].

Вивих — це стійке зміщення суглобних кінців костей за межі їх нормальної рухомості, інколи з розривом суглобної сумки і зв'язок і виходом однієї з кісток з сумки. Вивихи можливі внаслідок різких, надмірних рухів у суглобі, у випадках падіння з висоти на витягнуту кінцівку, в результаті сильного удару в ділянку суглоба. Часто вивихи супроводжуються переломами [24].

Перша допомога у випадку вивиху: потерпілого необхідно якнайшвидше доставити до медичного закладу, де йому вправлять суглоб. На час транспортування потерпілого до медичного закладу на ушкоджений суглоб потрібно накласти транспортну шину чи пов'язку, що надійно фіксує кінцівку. Для

зменшення болю можна дати потерпілому таблетку анальгіну чи іншого знеболювального засобу. Ні в якому разі не слід вправляти вивих самостійно

Перелом — це порушення цілісності кістки. Розрізняють закриті переломи, коли не відбувається пошкодження шкіри, та відкриті, коли зламана кістка виходить назовні. Ознаки. Біль постійний чи такий, що виникає в разі навантаження на ушкоджену кінцівку або при обмацуванні місця перелому, неможливість рухів в ушкодженій ділянці, зміна форми частини тіла (кінцівки) в ділянці перелому, крововиливи, ненормальна рухомість кістки в місці перелому. Загальний стан потерпілого залежить від характеру перелому і може бути досить важким (особливо в разі переломів кісток черепа, таза, стегна тощо), часто підвищується температура тіла. Слід пам'ятати, що деякі з перелічених ознак іноді можуть бути відсутні. Тому, коли є підозра на перелом, пошкодження слід розцінювати як перелом і надавати потерпілому відповідну допомогу [24].

Долікарняна допомога полягає в забезпеченні повного спокою пошкодженої частини тіла (кінцівки) та усуненні рухомості уламків кісток у місці перелому. Для цього потрібно зробити її нерухомою. Це досягається накладанням утримуючої пов'язки або ще краще — транспортної шини. Стандартні готові шини бувають металеві (дротяні чи з сітки) та дерев'яні.

Для транспортної іммобілізації найкраще користуватися готовими стандартними шинами, в разі їх відсутності шини виготовляють самі, їх можна зробити з будь-яких матеріалів або предметів, які можна знайти на місці нещасного випадку — палиці, дошки, дранки, кори дерева, очерету тощо. Головна вимога — достатня довжина та міцність шини.

Шини найкраще накладати на оголене тіло, але якщо зняття одягу завдає біль чи створює інші проблеми, можна накласти шину поверх одягу. У випадку накладання на оголену частину тіла шину потрібно обгорнути ватою чи тканиною, особливо там, де вона прилягає до виступів кісток. Шина повинна щільно прилягати до пошкодженої частини тіла.

4.2 Використання сучасних методів техніки безпеки в машинобудуванні

Основними подіями, що призводять до травматизму на підприємствах машинобудування, металообробної, оборонної промисловості, є дорожньо-транспортні пригоди, падіння з висоти, нещасні випадки на території підприємств, падіння з висоти, ураження електричним струмом, поранення під час роботи, дія небезпечних предметів, вибухи, пожежі та інші.

Основними причинами нещасних випадків є незадовільна організація виконання робіт та неузгоджені дії, відсутність або недостатня якість проведення навчання та інструктажів з техніки безпеки, невиконання вимог інструкцій та правил техніки безпеки, низький рівень виробничої дисципліни, недостатній контроль за технічним станом обладнання та інструментів, а також інші порушення охорони праці, порушення працівниками технологічної дисципліни, нехтування працівниками вимог безпеки, незадовільний технічний стан обладнання.

Для машинобудівного виробництва може бути використана така узагальнена класифікація технічних причин травматизму:

- Невідповідність нормам безпеки конструкцій технологічного обладнання й підйомно-транспортних пристроїв, технологічного оснащення, ручного механізованого інструменту;
- Відсутність чи недостатня надійність захистних пристроїв, вимикальних пристроїв, відповідного сигналізування тощо;
- Невідповідність конструкцій обладнання ергономічним вимогам (неправильне компоновування пультів управління, незручність місць обслуговування та ін.);
- Неправильний вибір обладнання, оснащення, транспортних засобів, методів, режимів обробки, зборки і транспортування;
- Відсутність указівок про методи і засоби безпечного виконання робіт;

- Недотримання термінів планово-попереджувального ремонту, несправність технологічного обладнання, оснащення, підйомно-транспортних пристроїв, ручного механізованого інструменту;

Реалізація цілей і завдань безпеки життєдіяльності на підприємстві містить у собі такі основні етапи: ідентифікація та опис зон потенційного виникнення небезпек підприємства і окремих його складових (комунікацій, устаткування, приладів тощо); розробка та реалізація найбільш ефективних систем і методів захисту від небезпек; формування систем контролю небезпек і керування станом безпеки підприємства; розробка і реалізація заходів для ліквідації наслідків прояву небезпек; організація навчання персоналу основам безпеки і підготовки фахівців з безпеки життєдіяльності.

Забезпечення безпеки праці в машинобудуванні є одним із найважливіших завдань, адже воно безпосередньо впливає на життя та здоров'я працівників, а також на продуктивність роботи підприємств. Сучасні методи техніки безпеки ґрунтуються на використанні новітніх технологій та досягнень науки, що дозволяє значно знизити ризики травмування та професійних захворювань.

Основні напрямки використання сучасних методів техніки безпеки в машинобудуванні:

Впровадження нового обладнання: У виробничому обладнанні залежно від особливостей виробничого процесу мають бути вбудовані пристрої для видалення шкідливих, вибухо- і пожежонебезпечних речовин безпосередньо від місця їх утворення, а в разі необхідності — пристрої, що скидають небезпечні й шкідливі речовини в приймальники або місця для утилізації чи знешкодження. Для речовин, які не можна скидати разом, проектують окремі складні пристрої.

Конструкція виробничого обладнання має бути такою, щоб забезпечити зниження рівнів шуму, ультра- та інфразвуку і вібрацій до значень, встановлених нормативними актами. Якщо під час роботи виникають шкідливі випромінювання, необхідно обладнати засоби захисту від випромінювання відповідно до вимог стандартів [26].

Навчання персоналу. Традиційні методи, які базуються на передачі знань через лекції та підручники, вже не є достатньо ефективними. Замість цього, сучасні підходи до навчання з охорони праці акцентують увагу на інтерактивних технологіях та методах, які залучають усіх до активної участі та практичного застосування знань.

Інноваційні методи навчання, такі як використання віртуальної реальності, дозволяють отримати реалістичний досвід безпеки на робочому місці, не відходячи від свого місця.

Розширені технології, такі як використання симуляційних тренажерів та інтерактивних програм, допомагають засвоїти та запам'ятати правила безпеки шляхом активної участі у віртуальних ситуаціях.

Прогресивні підходи, такі як групові проекти та колективні тренінги, сприяють розвитку комунікаційних навичок та співпраці між працівниками, що позитивно впливає на безпеку та ефективність роботи.

Переваги використання сучасного обладнанняполягають у з зниженні ризиків травмування та професійних захворювань. Покращення умов праці. Підвищення продуктивності роботи. Зниження витрат на виплату компенсацій за травми та професійні захворювання.

Переваги використання сучасного навчання полягають у забезпеченні більш ефективну та змістовну підготовку працівників до роботи в умовах безпеки та зменшення ризику нещасних випадків на робочому місці. Це важливий крок у напрямку покращення умов праці та забезпечення здоров'я та безпеки працівників у всіх сферах діяльності.

Таким чином, використання сучасних методів техніки безпеки в машинобудуванні є важливою умовою забезпечення безпечних та комфортних умов праці

ВИСНОВКИ

В результаті кваліфікаційної роботи була створена система обслуговування автомобілів з використанням новітніх технологій і передових методів розробки програмного забезпечення, таких як Spring Boot і React. Розроблена система успішно відповідає вимогам і забезпечує ефективне управління автомобільними послугами.

Сайт автосервісу дозволяє автоматизувати процес управління замовленнями, впоратися з наявністю майстрів, а також управляти працівниками і автомобілями, що знаходяться в ремонті. Система призначена для зберігання та управління послугами, що надаються автосервісом, управління деталями, необхідними для ремонту, обліку виконаних замовлень, розрахунку заробітної плати майстрів.

Проектування та розробка системи включали використання архітектури клієнт-сервер, розробку веб-інтерфейсу для створення замовлень та управління системою, інтеграцію з базою даних для зберігання даних автомобільних служб та використання механізмів безпеки. у формі аутентифікації для захисту конфіденційної інформації.

Для розробки системи використовувався широкий спектр технологій, включаючи HTML, CSS, JavaScript для інтерфейсної розробки та Java для серверної розробки. Для контейнеризації використовувався Docker, а для авторизації - JWT. Також використовувались фреймворки React та Spring Boot, а також бібліотеки, які допомогли прискорити розробку та покращити якість програмних продуктів.

Результати роботи свідчать про успішне створення функціонального і надійного веб-сайту для автосервісу, що забезпечує ефективне управління і задоволеність клієнтів. Розроблений веб-сайт може бути використаний в якості основи для подальшого розширення і покращення функцій.

СПИСОК ПОСИЛАНЬ

1. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – Інженерія програмного забезпечення (Освітньо-професійна програма - «Програмне забезпечення систем», Освітньо-наукова програма «Інженерія програмного забезпечення») для студентів усіх форм навчання / Упор.: М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк - Тернопіль: ТНТУ, 2020-51с.

2. Pressman R. S., Maxim B. Software Engineering: A Practitioner's Approach. McGraw-Hill Education, 2019. 704 с.

3. Офіційний сайт Mitchell. URL: https:// mitchell1.com

4. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Pearson Education, Limited, 2018.

5. Клієнт-серверна архітектура. URL: https://www.wikiwand.com/uk/Клієнт-серверна_архітектура

6. Офіційна документація Java. URL: https://docs.oracle.com/javase/specs/jls/se11/html/jls-1.html

7. Офіційна документація Spring. URL: https://spring.io

8. Software architecture patterns / Mark Richards. URL: www.oreilly.com/ideas/software-architecture-patterns/page/2/layered-architecture

9. Офіційна документація PostgreSQL. URL:https://www.postgresql.org

10. Офіційна документація React. URL:https://uk.legacy.reactjs.org

11. Офіційна документація HTML. URL: https://www.w3.org/TR/html53/

12. Документація CSS. URL: https://www.w3.org/Style/CSS/current-work

13. Офіційна документація Redux. URL: https://redux.js.org

14. Офіційна документація Heroku. URL: https://www.heroku.com/what

15. Офіційна документація Intellij Idea. URL: https://www.jetbrains.com/help/idea/discover-intellij-idea.html

16. Official UML Specification. URL: http://www.omg.org/spec/UML/

17. Visual Paradigm: Class DiagramGuide. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

18. Microsoft Docs: Multilayered architecture. URL: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/multilayer

19. Software Testing. URL:https://www.ibm.com/topics/software-testing

20. User API. URL: https://help.userguiding.com/en/userguiding-user-api

21. Docker. URL: https://www.docker.com

22. JWT. URL: https://uk.wikipedia.org/wiki/JSON_Web_Token

23. Перша долікарська допомога. URL: https://www.pharmencyclopedia.com.ua/article/790/persha-dolikarska-dopomoga

24. «Безпека життєдіяльності та основи охорони праці» – Зеркалов Д.В. Київ 2016. – 172-174 с.

25. INCOM Системи промислової безпеки. URL: https://encom-iss.com.ua/pro-kompaniiu/publikatsii/90-suchasni-metodi-navchannya-z-okhoroni-pratsi-vid-traditsijnikh-do-innovatsijnikh

26. «Охорона праці в галузі машинобудування» – Пістун І. П., Стець Р. Є., Трунова І. О., Суми 2012

ДОДАТКИ

ДОДАТОК А. Лістинг моделей

Car.java

```
package autoservice.model;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Getter
@Setter
@Entity
@Table(name = "cars")
public class Car {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String brand;
    private String model;
    private Integer year;
    @Column(name = "serial_number")
    private String serialNumber;
    @ManyToOne
    private Owner owner;

    public Car() {
    }

}
```

Файл Favor.java

```
package autoservice.model;

import lombok.Getter;
import lombok.Setter;

import java.math.BigDecimal;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Getter
@Setter
@Entity
@Table(name = "favors")
```

```
public class Favor {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne
    private Order order;
    @ManyToOne
    private Repairman repairman;
    @Column(name = "favor_name")
    private String favorName;
    private BigDecimal price;
    @Enumerated(EnumType.STRING)
    private Status status;

    public Favor() {
    }

    public enum Status {
        PAID,
        NOT_PAID
    }

    public Favor(Status status) {
        this.status = status;
    }

    public Favor setStatus(Status status) {
        this.status = status;
        return this;
    }
}
```

Файл Goods.

```
java package autoservice.model;

import lombok.Getter;
import lombok.Setter;

import java.math.BigDecimal;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Getter
@Setter
@Entity
@Table(name = "goods")
public class Goods {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private BigDecimal price;

    public Goods() {

    }
}
```

Файл Order.java

```
package autoservice.model;

import lombok.Getter;
import lombok.Setter;
import org.springframework.data.annotation.CreatedDate;

import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Getter
@Setter
@Entity
@Table(name = "orders")
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne
    private Car car;
    @Column(name = "problem_description")
    private String problemDescription;
    @CreatedDate
    @Column(name = "acceptance_date")
    private LocalDateTime acceptanceDate;
    @OneToMany(mappedBy = "order")
    private List<Favor> favors;
    @ManyToMany
    @JoinTable(
            name = "orders_goods",
            joinColumns = @JoinColumn(name = "order_id"),
            inverseJoinColumns = @JoinColumn(name = "goods_id"))
    private List<Goods> goods;
    @Enumerated(EnumType.STRING)
    private Status status;
    @Column(name = "final_price")
    private BigDecimal finalPrice;
    @Column(name = "end_date")
    private LocalDateTime endDate;

    public Order() {
    }

    public enum Status {
        IN_PROCESS,
        ACCEPTED,
        COMPLETED,
```

```
        FAILURE,
        PAID
    }

    public Order(Status status) {
        favors = new ArrayList<>();
        goods = new ArrayList<>();
        this.status = status;
    }

}
```

Файл Owner.java

```
package autoservice.model;

import lombok.Getter;
import lombok.Setter;

import java.util.ArrayList;
import java.util.List;
import javax.persistence.*;

@Getter
@Setter
@Entity
@Table(name = "owners")
public class Owner {
    @Getter
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "first_name")
    private String firstName;
    @Column(name = "last_name")
    private String lastName;
    @Column(name = "phone")
    private String phone;
    @Column(name = "email")
    private String email;
    @OneToMany(mappedBy = "owner")

    private List<Car> cars;
    @OneToMany
    @JoinColumn(name = "owner_id")
    private List<Order> orders;

    public Owner() {
        orders = new ArrayList<>();
        cars = new ArrayList<>();
    }
}
```

}

Файл Repairman.java

```
package autoservice.model;

import lombok.Getter;
import lombok.Setter;

import java.util.ArrayList;
import java.util.List;
```

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;
@Getter
@Setter
@Entity
@Table(name = "repairmans")
public class Repairman {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "first_name")
    private String firstName;
    @Column(name = "last_name")
    private String lastName;
    @Column(name = "phone")
    private String phone;
    @Column(name = "email")
    private String email;
    @ManyToMany
    @JoinTable(
            name = "repairmans_orders",
            joinColumns = @JoinColumn(name = "repairman_id"),
            inverseJoinColumns = @JoinColumn(name = "order_id"))
    private List<Order> completedOrders;

    public Repairman() {
        completedOrders = new ArrayList<>();
    }

}
```

Файл Role.java

```
package autoservice.model;

import lombok.AllArgsConstructor;
import lombok.Data;

import javax.persistence.*;

@Entity
@Table(name = "roles")
@Data
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "role_id")
    private Long id;
    private String name;

    public Role() {

    }

    public Role(String name) {
        this.name = name;
```

```
    }
}
```

Файл User.java

```
package autoservice.model;

import lombok.Getter;
import lombok.Setter;
import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@Getter
@Setter
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "user_id")
    private Long id;
    private String username;
    private String password;
    private boolean enabled;

    @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinTable(
            name = "users_roles",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id")
    )
    private Set<Role> roles = new HashSet<>();

    public User(){
        this.username = "";
        this.password = "";
        this.roles.add(new Role("USER"));
        this.enabled = true;
    }

    public User(String username, String password, boolean enabled, Set<Role>
roles) {
        this.username = username;
        this.password = password;
        this.enabled = enabled;
        this.roles = roles;
    }
}
```

ДОДАТОК Б. Диск із програмними файлами