НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
_____/Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення для інформаційної системи сервісного центру з гарантійного ремонту велосипедів»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми
_____к.т.н., доцент_____/ Вайганг Г.О./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи :_____/ Кириченко В.В/
підпис ПІБ

Виконав:_____/ Шкарбун М.Д./
підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук

_____/ Голуб Б.Л., доцент, к.т.н /

підпис

" " 2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студента Шкарбуна Максима Дмитровича

Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення для інформаційної системи сервісного центру з гарантійного ремонту велосипедів

Затверджена наказом ректора НУБіП України № 2249 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2025 . 06 . 02
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:
   1. Аналіз проблемної області.
   2. Вибір та обґрунтування засобів для розробки системи.
   3. Проектування інформаційної системи.
   4. Впровадження системи.
   5. Висновки.

Керівник бакалаврської кваліфікаційної роботи_____/ Кириченко В.В. /
підпис ініціали та прізвище

Завдання прийняв до виконання _____/ Шкарбун М.Д. /
підпис ініціали та прізвище

Дата отримання завдання 2025 . 02 . 16
рік, місяць, число

ЗМІСТ

ВСТУП

Обслуговування та ремонт велосипедів стрімко розвивається на тлі зростання популярності екологічного та активного способу життя. Щороку кількість велосипедистів зростає, що зумовлює підвищення попиту на якісний та своєчасний сервіс. Особливу роль у цій сфері відіграє гарантійне обслуговування, адже клієнти очікують не лише професійного ремонту, а й прозорого та зручного процесу взаємодії із сервісним центром.

В умовах зростаючої конкуренції та зростання вимог споживачів підприємства у сфері ремонту велосипедів мають впроваджувати сучасні інформаційні технології. Це дозволяє підвищити рівень обслуговування, оптимізувати внутрішні процеси, зменшити витрати, мінімізувати ймовірність помилок та прискорити обробку замовлень. Автоматизація діяльності сервісного центру з ремонту велосипедів є ефективним інструментом для досягнення зазначених цілей. Запровадження інформаційної системи управління дає змогу організувати чіткий облік замовлень, контролювати хід виконання робіт, вести базу даних запчастин, координувати роботу персоналу та забезпечити зручний електронний зв'язок із клієнтами.

Метою цієї бакалаврської кваліфікаційної роботи є розробка програмного забезпечення для інформаційної системи сервісного центру, що спеціалізується на гарантійному ремонті велосипедів. Проєкт передбачає створення веб-додатку з підтримкою ролей: Клієнт, Менеджер, Механік та Директор. Кожна роль передбачає певний набір функцій:

- Клієнт має можливість зареєструватися, подати заявку на ремонт, відстежувати її статус та переглядати історію звернень.

- Менеджер відповідає за прийом заявок, комунікацію з клієнтами та розподіл завдань між механіками.
- Механік виконує призначені йому ремонти, оновлює статуси заявок та фіксує використані деталі.
- Директор (адміністратор системи) здійснює управління обліковими записами працівників, аналізує ефективність роботи сервісу та має доступ до повної статистики.

Інформаційна система повинна забезпечувати швидкий і надійний доступ до даних, автоматизувати процеси подання та обробки заявок, контролю виконання завдань, облік запчастин. Особлива увага приділяється гарантійним зобов’язанням: система фіксуватиме, за якою заявкою і коли проводився ремонт, з можливістю відслідкувати повторні звернення.

Для досягнення поставленої мети необхідно реалізувати такі завдання:

- Дослідити предметну область та сформувати вимоги до системи;
- Проаналізувати існуючі рішення та виявити їх обмеження;
- Визначити функціональні й нефункціональні вимоги до програмного забезпечення;
- Створити UML-діаграми для опису структури та логіки системи;
- Розробити архітектуру майбутнього програмного продукту;
- Спроєктувати базу даних та реалізувати її засобами SQL-сервера;
- Створити веб-інтерфейс з урахуванням ролей та прав доступу;
- Налагодити взаємодію між клієнтською частиною та сервером;
- Провести тестування розробленої системи та оцінити її ефективність.

У результаті буде створено інформаційну систему, що дозволить автоматизувати ключові процеси сервісного центру. Таке рішення покращить комунікацію з клієнтами, підвищить точність обліку, забезпечить прозорість і допоможе підприємству зміцнити свої конкурентні позиції на ринку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛСТІ

1.1 Постановка задачі

Інформаційна система для сервісного центру гарантійного ремонту велосипедів має на меті спростити та автоматизувати процеси, пов’язані з обслуговуванням клієнтів, обліком замовлень, контролем за виконанням ремонтних робіт, а також управлінням персоналом і запчастинами.

Розроблюване програмне забезпечення у вигляді веб-системи забезпечує зручну взаємодію між усіма учасниками процесу: клієнтами, менеджерами, механіками та директором. Сервіс дозволяє клієнтам взаємодіяти з сервісним центром онлайн, а співробітникам – ефективно керувати внутрішніми процесами.

Функціональні вимоги

1. Система має забезпечувати реєстрацію та авторизацію користувачів з різними ролями (клієнт, менеджер, механік, директор), що дозволяє обмежити доступ до функціоналу згідно з повноваженнями кожної категорії користувачів.
2. Клієнт повинен мати можливість створювати, переглядати та редагувати заявки на ремонт, що дозволить скоротити час взаємодії з сервісом і зробити процес замовлення послуг більш зручним і прозорим.
3. Менеджер повинен мати інструменти для призначення механіків на конкретні заявки, що оптимізує розподіл робочого навантаження серед працівників сервісного центру.

4. Інформація про поточний статус кожного замовлення має відображатися в режимі реального часу, що забезпечить клієнтам прозорість процесу ремонту та дозволить уникнути непорозумінь.
5. Система повинна дозволяти ведення обліку використаних запчастин у межах кожного замовлення, що є необхідним для точного контролю витрат і подальшої аналітики.
6. Користувач (клієнт) має мати змогу переглядати історію власних замовлень, що дозволить йому аналізувати попередні ремонти та підтримувати зворотний зв’язок із сервісом.
7. Механіки повинні мати можливість додавати технічні коментарі до кожного етапу ремонту, що сприятиме кращому розумінню причин несправностей та якості виконаних робіт.
8. Директор має мати розширені можливості керування обліковими записами співробітників, а також контролю доступу до різних частин функціоналу системи для забезпечення організаційної безпеки.

Нефункціональні вимоги

1. Інформаційна система має бути розроблена з використанням сучасного технологічного стеку, зокрема Java та Spring Boot, що забезпечує стабільність, масштабованість і підтримку на довгострокову перспективу.
2. Взаємодія з базою даних повинна здійснюватися через PostgreSQL із використанням Spring Data JPA та Hibernate, що забезпечує ефективне збереження та обробку даних з мінімальними зусиллями з боку розробників.
3. Захист користувацьких даних повинен здійснюватися за допомогою сучасних механізмів автентифікації, управління сесіями та хешування паролів, що сприятиме підвищенню безпеки системи.
4. Користувацький інтерфейс має бути реалізований за допомогою шаблонізатора Thymeleaf, який дозволяє створювати інтуїтивно зрозумілі

сторінки без надлишкового використання JavaScript, з акцентом на простоту й доступність.

5. Система повинна підтримувати підключення до бази даних через пул з’єднань C3P0, що дозволить зменшити навантаження на ресурси сервера та підвищити швидкодію системи.
6. Для передачі та обробки даних має використовуватися формат JSON у поєднанні з бібліотекою Jackson, що забезпечує сумісність із більшістю зовнішніх систем і простоту інтеграції.
7. Валідація користувацького вводу повинна реалізовуватись із використанням Hibernate Validator, щоб гарантувати правильність і повноту введених даних до їх збереження у системі.
8. Усі основні компоненти системи повинні бути протестовані за допомогою модуля spring-boot-starter-test, що забезпечить стабільну роботу проєкту в реальному середовищі.
9. Дані в системі мають зберігатися централізовано з підтримкою функцій фільтрації, сортування та пошуку, що є критично важливим для забезпечення ефективної роботи з великим обсягом інформації.
10. Інтерфейс користувача має бути доступним і зрозумілим для представників усіх ролей, з урахуванням специфіки їх завдань і рівня комп’ютерної грамотності.

Бізнес-вимоги

1. Система повинна зменшити витрати часу на обробку клієнтських звернень, автоматизувавши процес створення та обробки заявок, що дозволить збільшити кількість оброблених замовлень без збільшення штату працівників.
2. Очікується, що впровадження системи підвищить задоволеність клієнтів, надаючи їм більше контролю та прозорості в комунікації з сервісним центром, що своєю чергою покращить репутацію компанії.

3. Інформаційна система має сприяти підвищенню ефективності праці персоналу, завдяки чіткій структуризації завдань, автоматизованому розподілу навантаження та мінімізації ручного вводу.
4. Після впровадження очікується зменшення кількості помилок, пов’язаних з людським фактором, таких як неправильне призначення замовлень, дублювання даних або втрата історії обслуговування.
5. Система має бути достатньо гнучкою для подальшого розширення бізнесу, включаючи можливість масштабування функціоналу, інтеграції з іншими платформами або обліковими системами в майбутньому.

Система повинна бути інтуїтивно зрозумілою, швидкодійною та масштабованою, з можливістю інтеграції з іншими сервісами (наприклад, e-mail нотифікаціями чи SMS), та підтримкою майбутнього розширення функціоналу.

1.2 Моделювання предметної області

Моделювання предметної області відіграє важливу роль на етапі проєктування програмного забезпечення, незалежно від сфери його застосування. Сьогодні для цього існує багато інструментів (так званих CASE-засобів), які допомагають спростити й пришвидшити процес розробки моделей.

Зазвичай побудова моделей програмного забезпечення потребує чималих затрат часу та ресурсів — у деяких випадках це може становити до 80% від загального обсягу робіт. Через це особливу увагу слід приділяти саме якісному моделюванню, адже помилки, допущені на початковому етапі, простіше виправити, ніж ті, які виявляться вже після реалізації системи. Якщо модель виявиться неповною, неточною або суперечливою, це може призвести до необхідності її доопрацювання або навіть до провалу всього проєкту.

Інформаційна модель — це структура, в якій відображені взаємозв’язки між різними складовими програмної системи: компонентами, базами даних, додатками. При її створенні слід враховувати всі елементи та їхню взаємодію.

Чим більша система, тим складніше побудувати логічну та узгоджену модель, яка б відповідала всім вимогам. Практика проєктування інформаційних систем дозволила виділити певні підходи, що дають змогу уникати поширених помилок.

Для опису моделей розроблено низку спеціалізованих мов. У даному випадку використовується UML — уніфікована мова моделювання, що є міжнародним стандартом у сфері розробки програмного забезпечення. Вона особливо зручна для опису систем середньої та великої складності. UML використовується не лише аналітиками, а й розробниками, тестувальниками та іншими учасниками команди. Існує багато програмних продуктів для роботи з UML-діаграмами: Rational Rose Enterprise, MagicDraw, Microsoft Visio та інші. UML дозволяє моделювати як архітектуру ПЗ, так і окремі бізнес-процеси, що робить її універсальним інструментом.

Одним із ключових принципів моделювання є абстрагування — тобто зосередження лише на тих деталях, які важливі для функціонування системи. Це допомагає уникнути зайвої складності. Ще один важливий принцип — побудова кількох моделей для повнішого відображення системи. Жодна окрема модель не здатна передати всі її аспекти, тому зазвичай створюється кілька, кожна з яких висвітлює певну частину.

Отже, UML — це графічна мова, що призначена для створення концептуальних і структурних моделей систем. Вона не є мовою програмування, а застосовується саме для аналізу й проєктування. Основними її елементами є UML-діаграми, які дозволяють візуалізувати різні сторони системи.

Згідно зі стандартом UML 1.5, існує 12 типів діаграм, які умовно поділяють на три групи:

- 4 типи описують статичну структуру;
- 5 — поведінку системи;
- 3 — фізичну реалізацію та функціонування.

На практиці використовується не весь набір діаграм, а лише ті, які потрібні для реалізації конкретного проєкту. У цьому випадку буде обрано відповідні діаграми згідно з вимогами системи.

1.3 Діаграма прецедентів

На етапі проєктування інформаційної системи було створено діаграму прецедентів (Use Case Diagram), яка відіграє ключову роль у візуалізації функціональних можливостей майбутньої системи з точки зору кінцевих користувачів. Дана діаграма дозволяє структуровано відобразити основні сценарії взаємодії між користувачами системи та її функціональними модулями. Це, своєю чергою, дає змогу краще усвідомити логіку побудови програмного забезпечення, його функціональне наповнення та обсяги відповідальності кожної ролі в системі.

Розроблювана система призначена для автоматизації діяльності сервісного центру з гарантійного ремонту велосипедів. Її основною метою є оптимізація процесів обслуговування клієнтів, управління ремонтними роботами, ведення обліку використаних матеріалів і запчастин, а також ефективне адміністрування персоналу. У межах цієї системи передбачено чотири основні ролі користувачів, кожна з яких має доступ до певного переліку функціональних можливостей, які відображені у вигляді прецедентів на відповідній діаграмі.

Роль "Клієнт" - Користувачі, які виступають у ролі клієнтів, взаємодіють із системою переважно для оформлення звернень на ремонт. Вони мають змогу створити заявку, переглянути історію своїх звернень, відстежити статус поточних ремонтів. Окрім того, клієнти можуть керувати власним профілем, редагувати особисті дані, а в разі необхідності — скасувати вже подану заявку. Слід зазначити, що такі дії, як редагування даних користувача або скасування заявки, реалізовано у вигляді розширень (<<extend>>) базових прецедентів. Це вказує на їхню необов’язкову, додаткову природу, яка активується за певних умов.

Роль "Механік" - Представники цієї ролі виконують технічну частину роботи, безпосередньо пов’язану з обслуговуванням велосипедів. Вони мають доступ до переліку заявок, які були їм призначені менеджером, та можуть оновлювати статус виконання ремонтних робіт у реальному часі. Також механіки додають до заявки інформацію про використані під час ремонту запчастини, що дозволяє забезпечити прозорість та точність обліку матеріалів.

Роль "Менеджер" - Менеджер виконує роль координатора у сервісному центрі. Його функціонал охоплює широкий спектр задач, зокрема перегляд усіх наявних у системі заявок, призначення відповідального механіка на кожну з них, ведення обліку замовлень, а також управління наявним складом запчастин. Таким чином, менеджер відіграє ключову роль у забезпеченні оперативності виконання ремонтних робіт і контролю ресурсів.

Роль "Директор" - Директор, як найвища адміністративна ланка, має розширені повноваження. Окрім можливості перегляду всіх заявок і контролю за обліком замовлень та управлінням складом, директор також здійснює управління менеджерами. Він може додавати нових менеджерів, редагувати їхні дані або обмежувати доступ до системи, що є важливим для підтримання належного рівня внутрішньої безпеки та організаційної структури.

Діаграма прецедентів, що представлена на рисунку 1.1, відображає всю сукупність взаємозв’язків між ролями користувачів та функціональними можливостями системи. Завдяки їй розробники, замовники та інші зацікавлені сторони можуть швидко зрозуміти логіку функціонування системи й оцінити обсяги майбутньої реалізації. Такий підхід дозволяє забезпечити прозорість процесу розробки, уникнути функціональних протиріч та підвищити загальну якість проєктного рішення.

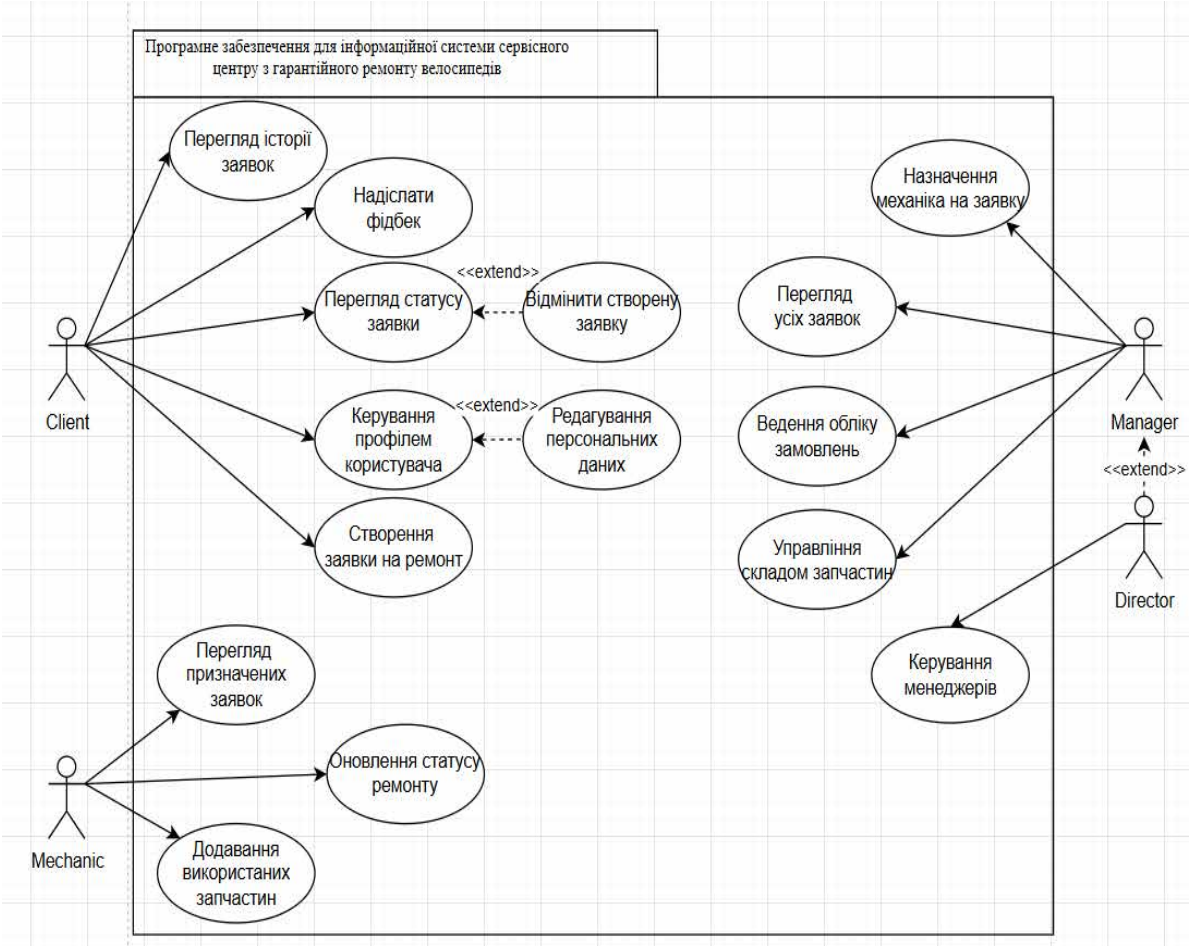

Рис. 1.1 – Діаграма прецендетів

1.4 Діаграма активності

Діаграма активності, що представлена на рисунку 1.2, виступає ключовим інструментом формального опису динамічних процесів у програмних системах, що дозволяє наочно відобразити послідовність дій, логіку переходів між ними, а також альтернативні шляхи розвитку сценаріїв. Вона є ефективним засобом моделювання алгоритмів взаємодії користувачів із системою, охоплюючи як зовнішню, так і внутрішню бізнес-логіку.

У контексті роботи сервісного центру діаграма демонструє взаємозв’язки між ключовими учасниками процесу — клієнтом, менеджером і механіком, розкриваючи послідовність прийняття рішень та реалізації технічних дій. Такий підхід сприяє кращому розумінню організаційної структури обслуговування, виявленню потенційних проблемних ділянок і покращенню внутрішньої координації. Діаграма також створює підґрунтя для аналітики ефективності системи, дозволяючи оптимізувати процеси, підвищити якість сервісу та забезпечити прозорість комунікацій між усіма залученими ролями.

Ініціалізація процесу обслуговування

Початковою точкою активності виступає клієнт, який ініціює процес створення заявки на ремонт. Це може бути здійснено через веб-інтерфейс, де користувач зазначає опис проблеми та дані про велосипед. На цьому етапі важливо забезпечити правильну валідацію вхідних даних, що дозволяє запобігти помилкам на подальших етапах обробки.

Первинна перевірка та ухвалення рішення

Після надходження заявки вона передається на обробку менеджеру. Тут активується блок перевірки валідності інформації, а також аналізу відповідності запиту умовам гарантійного обслуговування. Діаграма демонструє ключову точку розгалуження: у випадку невідповідності заявка відхиляється, а клієнт

отримує відповідне повідомлення. У разі позитивного рішення менеджер підтверджує заявку, забезпечуючи її подальшу обробку.

Призначення відповідального виконавця

Після верифікації заявки менеджер здійснює призначення конкретного механіка, враховуючи його поточну завантаженість, спеціалізацію та складність заявленої проблеми. Це рішення має важливе значення для оптимізації розподілу ресурсів сервісного центру та мінімізації часу очікування з боку клієнта. Таким чином, система функціонує не лише як засіб комунікації, а й як інструмент ефективного управління персоналом.

Виконання ремонту та оновлення статусу

Механік, отримавши завдання, переходить до виконання технічних робіт. По завершенню ремонтного процесу він оновлює статус заявки в системі, фіксуючи результат, витрачені ресурси, а також можливі додаткові рекомендації для користувача. Це оновлення автоматично ініціює тригер для наступного етапу.

Інформування клієнта та завершення процесу

Після оновлення статусу система надсилає повідомлення клієнту про завершення ремонту, включаючи детальну інформацію про виконані роботи та рекомендації щодо подальшої експлуатації велосипеда. Дана активність є фінальною у представленому сценарії та формалізує завершення потоку дій. Таким чином, клієнт отримує повну прозорість щодо ходу виконання заявки, що суттєво підвищує рівень задоволеності сервісом.

Детальна розробка діаграми активності сприяє формалізації логіки сервісного процесу та забезпеченню відповідності між функціональними вимогами та архітектурними рішеннями системи. Впровадження аналітичного підходу до побудови такої діаграми підвищує предиктивність та ефективність

управлінських процесів у сервісному центрі. Крім того, аналіз активності дозволяє виявити вузькі місця в організації роботи, скоротити час обробки заявок, оптимізувати використання людських ресурсів та гарантувати відповідність очікуванням кінцевого користувача.

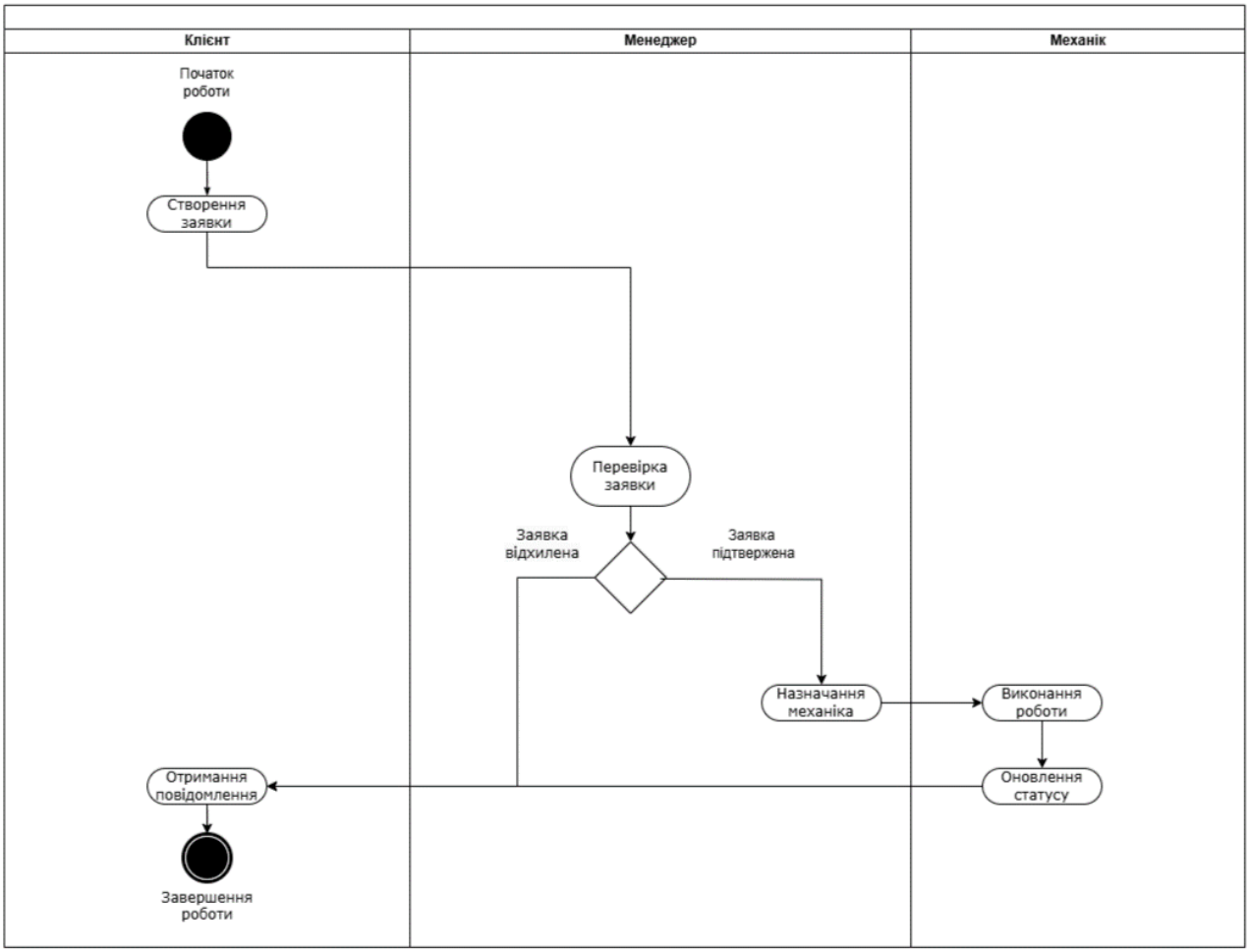

Рис. 1.2 – Діаграма активності

1.5 Діаграма послідовності

Діаграма послідовності, що представлена на рисунку 1.3, є важливим інструментом формального моделювання бізнес-процесів сервісного центру, що дозволяє наочно відобразити порядок обміну повідомленнями між учасниками системи — клієнтом, менеджером, механіком та інформаційною системою. Такий підхід забезпечує чітке розуміння логіки виконання запитів, синхронізації дій між ролями, а також сприяє виявленню потенційних помилок або затримок у процесі обробки заявок.

Ця діаграма відіграє ключову роль на етапі проєктування системи, допомагаючи візуалізувати алгоритм взаємодії у межах єдиного потоку подій, де кожна роль виконує визначену функцію в рамках процесу обслуговування клієнта.

Основні учасники процесу:

- Клієнт — ініціатор звернення, який створює заявку на ремонт велосипеда через інтерфейс системи.
- Менеджер — виконує перевірку отриманої інформації, підтверджує заявку та координує подальші дії.
- Механік — отримує інформацію про підтверджену заявку та оновлює її статус відповідно до ходу виконання ремонту.
- Система — забезпечує обмін повідомленнями між учасниками, фіксує зміни статусу заявки та надсилає сповіщення.

Послідовність операцій

1. Створення заявки
    - Клієнт формує заявку на ремонт, зазначаючи необхідні деталі (тип несправності, дані про велосипед, чи гарантій ремонт).

- Система реєструє заявку та надсилає повідомлення менеджеру про нове звернення.

2. Опрацювання заявки менеджером
    - Менеджер перевіряє правильність заповнених даних, уточнює деталі за потреби, підтверджує заявку.
    - Система оновлює статус заявки та повідомляє механіка про нове завдання.
3. Робота механіка
    - Механік отримує заявку та починає її обробку.
    - Після виконання частини або всього обсягу робіт, механік оновлює статус (наприклад: «Виконується», «Завершено», «Очікує деталі»).
    - Система надсилає клієнту сповіщення про зміни в заявці.

Альтернативні сценарії

- Заявка відхилена — у разі виявлення помилок у заявці або відсутності доступного часу, менеджер може відхилити заявку, і система проінформує клієнта.
- Відсутність механіка — якщо жоден механік не доступний, система генерує повідомлення з відповідною інформацією та зберігає заявку у черзі.

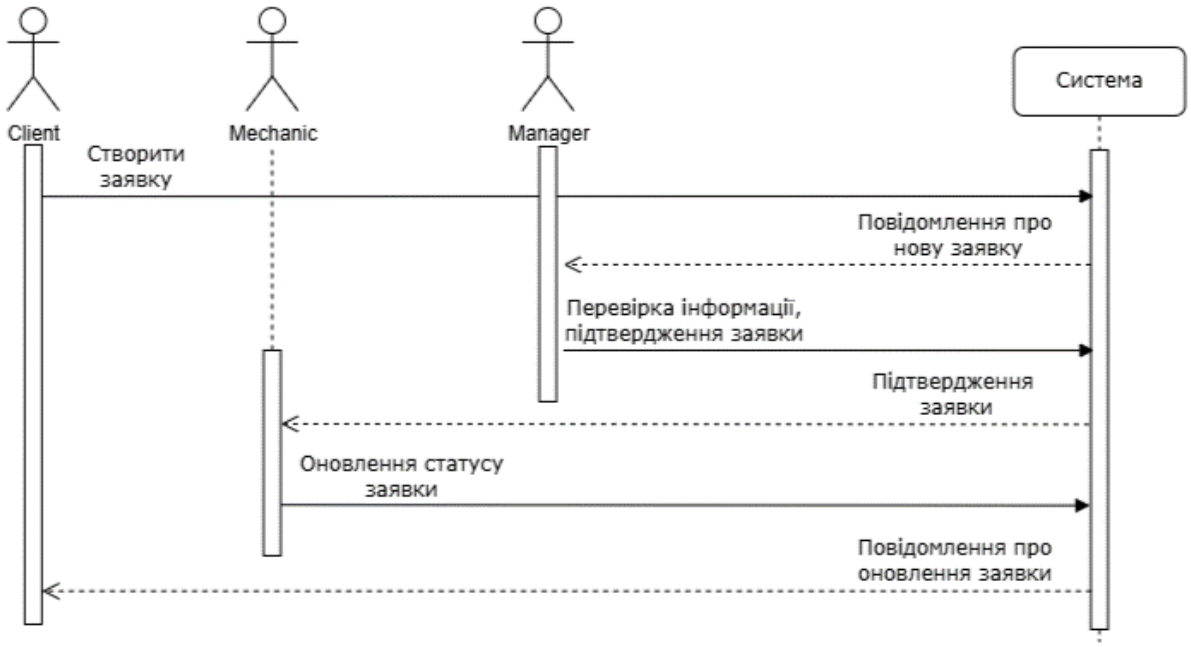

Рис. 1.3 – Діаграма послідовності

1.6 Структурно-функціональна блок-схема алгоритму

Блок-схема на рисунку 1.4 демонструє ключову послідовність дій при створенні заявки на гарантійний ремонт велосипеда через веб-інтерфейс сервісної інформаційної системи. Ця схема охоплює логіку від підтвердження користувачем наміру створити заявку до фіксації інформації про оплату й завершення операції.

В контексті розроблюваної системи, така візуалізація дозволяє формалізувати бізнес-процеси обробки заявок, забезпечити коректну взаємодію між фронтендом Thymeleaf і бекендом (Spring Boot, Hibernate, PostgreSQL), а також узгодити логіку для подальшого модульного тестування.

Процес ініціюється після авторизації користувача (ролі: Клієнт), який підтверджує створення нової заявки. Після цього система перевіряє, чи вже існує в базі даних запис про велосипед, пов’язаний з клієнтом. Якщо велосипед

присутній — використовується наявний запис. Якщо ні — система створює новий запис про велосипед, зберігаючи його в базу даних через сервісний шар.

Далі виконується формування нової заявки на ремонт, де автоматично встановлюється статус заявки як "обробка", а статус платежу — "неоплачено". Ці атрибути використовуються для фільтрації заявок у кабінеті менеджера та механіка. Запис зберігається до бази даних за допомогою відповідного CRUD-методу.

Опис структурно-функціональна блок-схема створення заявки на ремонт велосипеда:

- Початок роботи — ініціація сценарію після входу в систему.
- Користувач підтверджує створення заявки — дія з інтерфейсу (натискання кнопки).
- Умова: велосипед існує в БД?
  - Так → використовуємо запис із бази.
  - Ні → створюється новий велосипед, що зберігається до бази даних.
- Створення нової заявки на ремонт — формується сутність RepairRequest.
- Статус заявки → "обробка"
- Статус платежу → "неоплачено"
- Збереження запису до БД — за допомогою JPA/Hibernate.
- Завершення роботи — повернення результату.

Рис. 1.4 – Структурно-функціональна блок-схема створення заявки на ремонт велосипеда

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Загальні відомості про ER-діаграму

На сучасному етапі розвитку цифрових технологій та активного впровадження інформаційних систем у повсякденне життя надзвичайно важливою є потреба у грамотному, структурованому й логічному підході до проєктування баз даних. Особливо це стосується тих інформаційних систем, які покликані автоматизувати процеси, пов’язані зі зберіганням, обробкою та використанням великої кількості взаємопов’язаних даних. У цьому контексті надзвичайно актуальним є застосування методів концептуального моделювання, серед яких провідну е місце займає побудова ER-діаграм.

ER-діаграма (сутність-зв’язок) — це один з основних інструментів логічного моделювання, який дозволяє наочно представити предметну область системи у вигляді набору сутностей (об'єктів), їхніх властивостей (атрибутів), а також взаємозв’язків між цими сутностями. Вона є початковим етапом при побудові бази даних, що забезпечує цілісне бачення того, як саме повинні взаємодіяти елементи системи, аби забезпечити коректну та ефективну роботу програмного продукту.

У рамках даної дипломної роботи було розроблено ER-діаграму, що відображає логічну структуру бази даних інформаційної системи, призначеної для автоматизації процесу гарантійного обслуговування велосипедів. Основною метою побудови такої діаграми є створення уніфікованої моделі зберігання інформації, яка буде легко масштабуватися, доповнюватися та використовуватися на практиці при реалізації вебзастосунку.

ER-діаграма виконує кілька важливих функцій, серед яких можна виокремити наступні:

- формалізація предметної області через визначення основних об’єктів системи;
- встановлення чітких зв’язків між об’єктами, що дозволяє уникнути логічних помилок у структурі;
- підготовка основи для подальшої реалізації фізичної моделі бази даних, яка буде реалізована у конкретній СУБД;
- полегшення взаємодії між розробниками, аналітиками, замовниками та користувачами програмного продукту шляхом створення наочного уявлення про структуру даних.

Завдяки чіткому розділенню сутностей та їхніх взаємозв’язків, розроблена ER-діаграма дозволяє не лише забезпечити ефективне зберігання даних, але й дає змогу виявити потенційні проблеми ще на етапі проєктування, що істотно знижує ризик виникнення логічних конфліктів або надмірного дублювання інформації при реалізації системи.

ER-діаграма охоплює такі ключові компоненти предметної області, як користувачі, велосипеди, замовлення, деталі, категорії деталей, повідомлення та інші сутності, які відіграють важливу роль у процесах гарантійного обслуговування. Кожна з цих сутностей має чітко визначений набір атрибутів, що дозволяє зберігати відповідну інформацію у структурованому вигляді. Крім того, між сутностями встановлено логічні зв’язки, які відображають реальні взаємовідносини об’єктів у межах предметної області.

Важливо зазначити, що дана ER-модель не є кінцевою чи жорстко фіксованою — у процесі розвитку та масштабування інформаційної системи вона може бути доповнена новими сутностями або зв’язками, що забезпечить

гнучкість архітектури та адаптивність до змін. Проте вже на поточному етапі побудована діаграма дозволяє створити повноцінну реляційну базу даних, яка буде відповідати основним вимогам до збереження, обробки та захисту інформації в межах поставлених задач.

Таким чином, побудова ER-діаграми є важливим етапом у процесі розробки програмного забезпечення, що ґрунтується на системному підході до структурування даних. Це дозволяє не лише підвищити якість програмного продукту, але й забезпечити зручність подальшого супроводу, модифікації та масштабування системи у випадку змін у бізнес-процесах або розширення функціональності.

2.2 Побудова ER- діаграми

У процесі розробки інформаційної системи сервісного центру з гарантійного ремонту велосипедів побудова ER-діаграми (сутність-зв’язок) виступає ключовим етапом концептуального моделювання. Вона дозволяє формалізовано описати основні об’єкти предметної області, визначити їхні атрибути та логічні зв’язки, що, в свою чергу, є основою для побудови фізичної моделі бази даних. Такий підхід забезпечує системність, цілісність і узгодженість при реалізації модулів обробки даних та інтеграції з іншими компонентами вебсистеми.

ER-діаграма надає розробникам уніфіковане уявлення про структуру інформаційної системи. У межах даного проєкту вона охоплює основні сутності: Користувач (User), Замовлення (Order), Велосипед (Bicycle), Запчастина (Part), Категорія запчастин (Part Category), Сессія (UserSession), Повідомлення (Notification) та Використані запчастини (RepairPart). Між цими сутностями встановлені відношення типу один до багатьох та багато до багатьох, які логічно відображають бізнес-процеси сервісного центру.

Особливістю цієї системи є наявність чотирьох чітко розмежованих ролей: Клієнт, Менеджер, Механік та Директор. Користувачі з різними ролями взаємодіють із системою за допомогою окремих інтерфейсів і мають відмінний функціональний доступ. Наприклад, клієнт створює замовлення та отримує повідомлення про його статус, менеджер розподіляє замовлення між механіками, а директор здійснює адміністрування користувачів та контролює якість виконання.

Візуалізацію логічної моделі було виконано за допомогою онлайн-платформи draw.io — інструменту, що забезпечує просте, але функціональне середовище для створення ER-діаграм. Серед переваг даної платформи — підтримка експорту у популярні графічні формати, спільна робота над схемами та швидке редагування елементів моделі. У навчальному та прототипному середовищі цей інструмент є оптимальним вибором, що дозволяє зосередитися на структурі моделі без потреби в складному програмному забезпеченні.

Розроблена ER-діаграма відображає такі ключові аспекти:

- Сутність Order пов’язана з User як клієнтом (customer), менеджером і механіком через зовнішні ключі;
- Bicycle прив’язується до конкретного клієнта;
- RepairPart реалізує зв’язок між замовленням і запчастинами, відображаючи використані деталі в процесі ремонту;
- Notification зберігає повідомлення для користувачів, наприклад, про зміну статусу замовлення;
- Part зв’язується з PartCategory, що дозволяє систематизувати складські одиниці.
- Session зберігає інформацію про активні сесії користувачів, включаючи унікальний ідентифікатор сесії, час створення, час завершення, і

пов’язується з користувачем через зовнішній ключ (user_id). Це необхідно для реалізації механізмів входу в систему, контролю авторизації та безпеки.

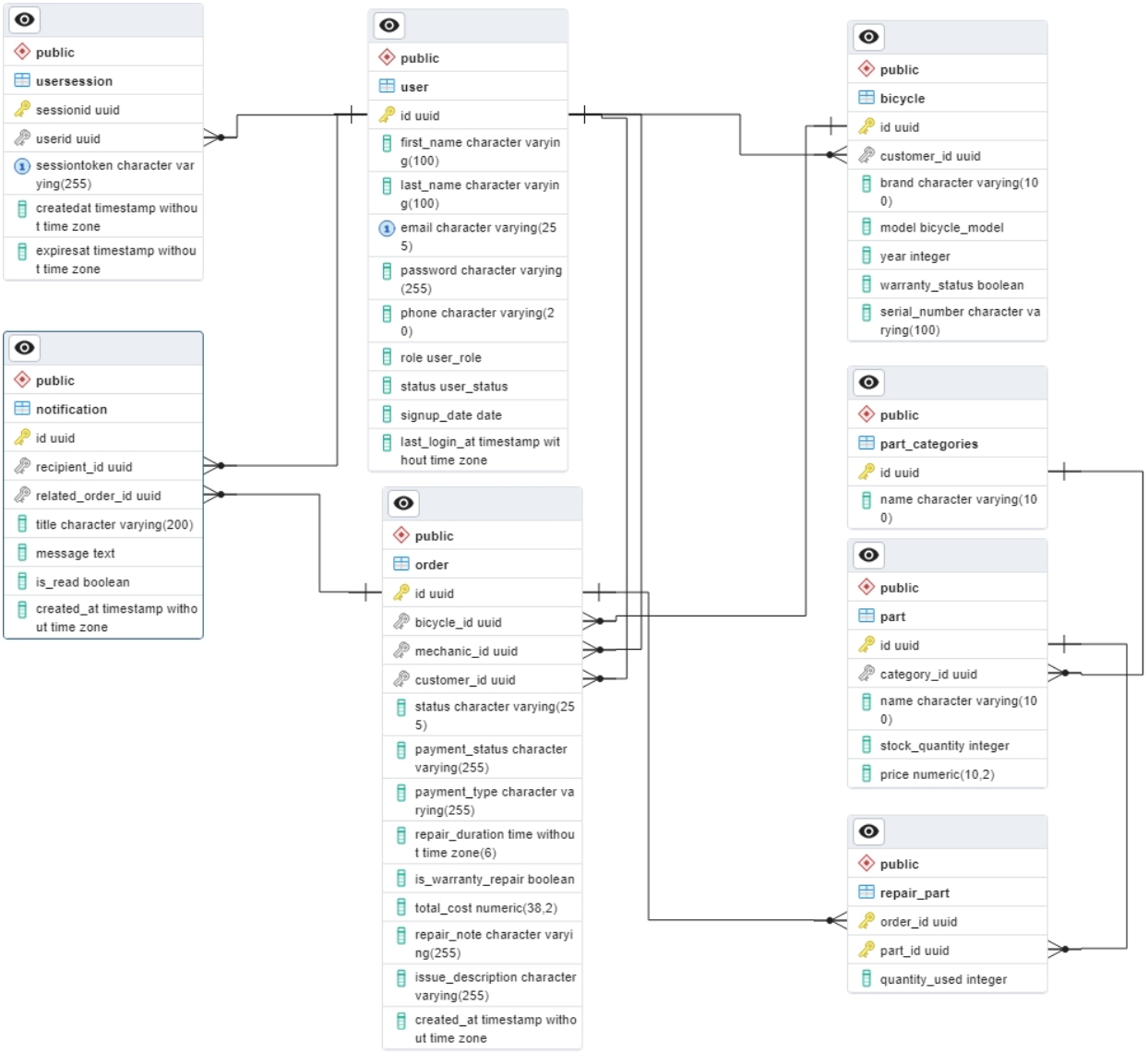

Рис. 2.1 – ER-діаграма сервісного центру з ремонту велосипедів

Таким чином, створена ER-діаграма виступає не лише як засіб документування логічної структури бази даних, а й як аналітичний інструмент для підтримки розробки, тестування та масштабування системи. Вона відіграє важливу роль у забезпеченні правильного відображення бізнес-логіки на рівні

бази даних і дозволяє сформувати надійний фундамент для подальшої реалізації через ORM-рівень (Hibernate) у фреймворку Spring Boot.

2.3 Вибір та обґрунтування СУБД

Для забезпечення стабільного функціонування вебзастосунку, який одночасно обслуговує велику кількість користувачів, обрано систему керування базами даних PostgreSQL. Це сучасна об’єктно-реляційна СУБД з відкритим кодом, яка поєднує високу продуктивність, масштабованість і підтримку розширень. Вона чудово підходить для вебсистем, які вимагають складної логіки обробки даних, надійності й безпеки.

Чому саме PostgreSQL?

- Повна підтримка ACID-гарантій забезпечує цілісність і надійність даних;
- Система легко справляється з багатокористувацьким доступом у режимі реального часу;
- Гнучка система доступу (рольова модель, шифрування, SSL);
- Підтримка JSONB дозволяє зберігати напівструктуровані дані;
- Наявність розширень, таких як PostGIS та Foreign Data Wrappers;
- Можливість масштабування як вертикально, так і горизонтально.

PostgreSQL виконує роль основного сховища даних, на яке спирається уся серверна логіка. Користувачі взаємодіють не з базою напряму, а через REST API, реалізований за допомогою Spring Boot. Це дозволяє централізовано керувати доступом, логікою запитів і відповідей, забезпечуючи чисту архітектуру з чітким розмежуванням обов’язків між фронтендом і бекендом.

Особливості взаємодії та моделі сесій

На відміну від безстанної авторизації (JWT), у системі реалізовано сесійну модель автентифікації. Після входу дані користувача зберігаються на сервері в межах сесії, прив’язаної до клієнта через cookie. Це дозволяє ефективно підтримувати його статус, обмежувати доступ до приватних розділів сайту та централізовано керувати правами доступу.

Кожен запит — чи то перегляд замовлень, редагування профілю, чи доступ до історії — обробляється на сервері. PostgreSQL слугує не просто сховищем, а повноцінною частиною бізнес-логіки: із транзакційною підтримкою, зовнішніми ключами, індексацією й використанням JSON-полів там, де це доцільно.

Переваги централізованої обробки

- Уся логіка фільтрації, перевірок і трансформації даних — на сервері, а не розпорошена по клієнтському коду;
- Зміни в бізнес-логіці не потребують редагування інтерфейсу — достатньо оновити API;
- Сесійна модель дозволяє реалізувати особисті функції (профіль, історія, сповіщення) без ризику витоку даних або дублювання стану.

2.4 Створення БД

Проєктування бази даних стало одним із найвідповідальніших і найвизначальніших етапів у розробці інформаційної системи для сервісного центру, що спеціалізується на гарантійному обслуговуванні велосипедів. Саме ця частина проєкту забезпечує зв'язок між усіма модулями системи, дозволяючи зберігати, оновлювати й аналізувати інформацію в реальному часі. Незважаючи на те, що більшість бізнес-логіки реалізована на боці сервера за допомогою Spring Boot, саме база даних є тією невидимою, але критично важливою частиною системи, яка пов’язує всі дії користувачів у єдину послідовну історію.

У якості основи для збереження даних було обрано СУБД PostgreSQL — надійне та потужне рішення, яке добре зарекомендувало себе в проєктах будь-якої складності. Вона підтримує транзакційність, забезпечує високу продуктивність та має хорошу сумісність із ORM-фреймворками, зокрема Hibernate, який ми використали для автоматичного створення таблиць на основі Java-ентітей. Такий підхід значно знижує ризик людських помилок при моделюванні схеми та пришвидшує розгортання.

Архітектура системи передбачає існування чотирьох основних ролей користувачів: клієнт, менеджер, механік і директор. Кожен із них взаємодіє з системою по-своєму, і тому структура бази даних повинна не лише зберігати інформацію, а й логічно поєднувати всі ці взаємозв’язки. Наприклад, клієнт реєструє велосипед, менеджер приймає замовлення на ремонт, механік виконує роботи та використовує запчастини, а директор має змогу переглядати загальний стан системи та оцінювати ефективність роботи команди.

Щоб забезпечити ці сценарії, були створені наступні таблиці:

- users — центральна таблиця, яка містить детальну інформацію про всіх користувачів: ПІБ, email, номер телефону, роль (визначену через ENUM user_role), статус (активний або заблокований), а також дати створення акаунту та останнього входу. Це дозволяє зручно керувати доступом та відслідковувати активність у системі.
- usersession — відповідає за збереження інформації про сесії авторизації, зокрема токени доступу, час їх створення та строк дії. Це необхідно для підтримки безпечного і стабільного механізму входу.
- bicycle — таблиця, яка містить дані про велосипеди клієнтів: бренд, модель (тип велосипеда визначено через ENUM bicycle_model), рік випуску, серійний номер, а також інформацію про гарантійний статус.

- order — найважливіша таблиця, що описує сам процес ремонту. Вона поєднує клієнта, механіка, велосипед, та містить інформацію про статус замовлення (через ENUM repair_status), тип і статус оплати, тривалість та загальну вартість ремонту, опис несправності й додаткові нотатки.
- notification — таблиця для внутрішніх повідомлень, що дозволяє інформувати користувачів про важливі події, такі як зміна статусу замовлення чи призначення механіка.
- part_categories і part — довідкові таблиці, що дозволяють систематизувати запчастини за категоріями. У таблиці part також ведеться облік доступної кількості на складі, ціни, опису та пов’язаної категорії.
- repair_part — таблиця, яка фіксує, які запчастини були використані для кожного замовлення, в якій кількості, і за якою ціною. Це дозволяє вести точний облік витрат матеріалів та формувати повну історію ремонту.

Окрім таблиць, у системі реалізовано ENUM-типи, такі як user_status, user_role, repair_status, payment_type, payment_status, bicycle_model. Вони дозволяють уникати дублювання та помилок у статусах, а також спрощують аналіз і побудову звітів.

Під час активної розробки було використано механізм ddl-auto=update, який дозволяє автоматично оновлювати схему бази відповідно до змін у Java-моделях. Це значно полегшувало тестування і дозволяло гнучко змінювати структуру без ручного втручання. Водночас, для продакшн-версії було передбачено перехід на керовані міграції, щоб гарантувати стабільність та контрольованість змін.

Для спрощення діагностики було налаштовано логування SQL-запитів та створили набір запитів для швидкого перегляду вмісту основних таблиць. Це дозволяє легко перевірити, чи була створена заявка, змінено статус, або які запчастини використовувалися останнім часом. Таким чином, база даних у

системі — це не просто місце зберігання інформації. Вона виконує роль «розумного посередника» між усіма учасниками процесу ремонту, підтримує узгодженість даних, забезпечує гнучкість та закладає надійний фундамент для подальшого масштабування і розвитку.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕПЕЧЕННЯ

3.1 Вибір інструментарію для розробки програмного забезпечення

Під час реалізації інформаційної системи для сервісного центру гарантійного ремонту велосипедів було критично важливо підібрати такий стек технологій, який забезпечить надійність, масштабованість і відповідність вимогам сучасного веброзробництва. Основна мета — створити зручний вебсервіс, де клієнти можуть реєструватися, створювати заявки на ремонт, відстежувати статус обслуговування, а співробітники — обробляти ці заявки згідно зі своїми ролями (Менеджер, Механік, Директор). З огляду на складну бізнес-логіку, багаторівневу авторизацію, управління замовленнями, запасними частинами та нотифікаціями, було обрано перевірений та структурований інструментарій.

Середовище розробки.

У якості основного середовища розробки використано IntelliJ IDEA — повнофункціональну IDE, яка забезпечує зручну інтеграцію з технологіями Spring Boot, підтримує роботу з базами даних PostgreSQL, автогенерує конфігурації, надає вбудовані інструменти відладки, профілювання та тестування. Система проєкту структурована відповідно до підходу MVC, що дозволяє розділити логіку, відображення та обробку даних.

Мова програмування та серверна частина.

Основною мовою розробки було обрано Java, завдяки її стабільності, широкій підтримці у Spring-екосистемі та відповідності парадигмі об’єктно-орієнтованого програмування. Для побудови серверної частини застосовано Spring Boot**,** який забезпечує швидкий старт проєкту, конфігурацію та інтеграцію з ORM-фреймворком Hibernate. Spring Boot у поєднанні з Spring MVC використано для реалізації контролерів, обробки HTTP-запитів, маршрутизації та побудови REST-інтерфейсів, хоча в рамках даного проєкту основна взаємодія з користувачем відбувається не через REST API, а за допомогою серверного рендерингу HTML-шаблонів.

Робота з базою даних.

Для зберігання даних застосовано PostgreSQL, як надійну, потужну та відкриту СУБД, що підтримує роботу з ENUM-типами, складними зв’язками, транзакціями, індексацією та забезпечує високу продуктивність. Структура бази даних повністю нормалізована та відповідає предметній області:

- users, usersession — керування користувачами, сесіями, ролями та статусами (user_role, user_status);
- bicycle — реєстрація велосипедів, їх модель, бренд, гарантійний статус;
- orders — заявки на ремонт з полями про статус (repair_status), оплату (payment_status, payment_type), призначення механіків;
- part_categories, part, repair_part — облік запасних частин та їх використання;
- notification — реалізація системи повідомлень клієнтів про хід ремонту.

ORM-рівень реалізовано через Hibernate, який дозволяє зручно працювати з сутностями, автоматично створювати таблиці, реалізовувати зв’язки (OneToMany, ManyToOne, ManyToMany) та вести аудит змін.

Шаблонізація та інтерфейс.

Для побудови інтерфейсу користувача було обрано Thymeleaf — Java-шаблонізатор, інтегрований зі Spring Boot, який дозволяє динамічно формувати HTML-сторінки на сервері з урахуванням ролі користувача, стану об'єктів, статусів замовлень тощо. Використання Thymeleaf дає змогу реалізувати перевірки ролей (th:if="${user.role == 'MANAGER'}"), доступність кнопок, відображення форм для різних дій (створення заявки, перегляд історії, оновлення статусу ремонту тощо).

Безпека.

Реалізована базова авторизація через Spring Security: розмежування доступів до функціоналу відповідно до ролей (CLIENT, MANAGER, MECHANIC, DIRECTOR), хешування паролів (BCrypt), захист від CSRF-атак, обробка сесій через usersession.

Контроль версій та збірка.

Управління залежностями та збірка проєкту реалізовані через Maven, що дозволяє автоматизувати компіляцію, запуск тестів, формування .jar-файлів, а також інтегрувати зовнішні бібліотеки (Spring Boot Starter'и, PostgreSQL Driver, Hibernate Validator тощо).

Інші інструменти.

Для перевірки та валідації даних на рівні DTO-класів використано Jakarta Validation API (анотації типу @NotBlank, @Size, @Email). Для логування — SLF4J + Logback, що дає змогу фіксувати дії користувачів, оновлення статусів, помилки доступу, виклики сервісів.

3.2 Принципи роботи у середовищі візуальної розробки програм

Сучасні середовища розробки вже давно перестали бути просто набором базових інструментів для написання коду. Сьогодні це — комплексні, інтелектуальні платформи, які супроводжують розробника на кожному етапі життєвого циклу програмного продукту, починаючи від початкового проектування і прототипування і до фінального тестування, деплойменту та подальшої підтримки. IntelliJ IDEA є одним із таких передових рішень, що пропонує широку функціональність, надійність і гнучкість, суттєво полегшуючи і прискорюючи процес розробки проєктів будь-якого рівня складності, зокрема інформаційних систем з інтегрованими бізнес-процесами, таких як система сервісного центру з гарантійного ремонту велосипедів.

Ключовим фактором, що робить IntelliJ IDEA надзвичайно ефективною, є її здатність об’єднувати в єдиному середовищі різноманітні аспекти розробки: від написання, структуризації та відлагодження коду, до управління версіями, роботи з базами даних і безшовної інтеграції з сучасними веб-технологіями. Це дає можливість розробнику не просто писати код, а зосередитися на логіці бізнесу і архітектурі системи, водночас оптимізуючи робочий процес, скорочуючи кількість помилок і пришвидшуючи ітерації розробки.

Особливу увагу заслуговує інтелектуальна підтримка коду, що реалізована в IntelliJ IDEA. Розширене автодоповнення, миттєвий аналіз синтаксису, контекстні підказки, а також попередження про потенційні помилки на ранньому етапі написання коду значно підвищують продуктивність розробника. Особливо важливо це для проєктів, що використовують Spring Boot — популярний фреймворк для розробки серверної частини, де швидкий доступ до правильних анотацій, методів та конфігурацій критично впливає на якість і швидкість розробки.

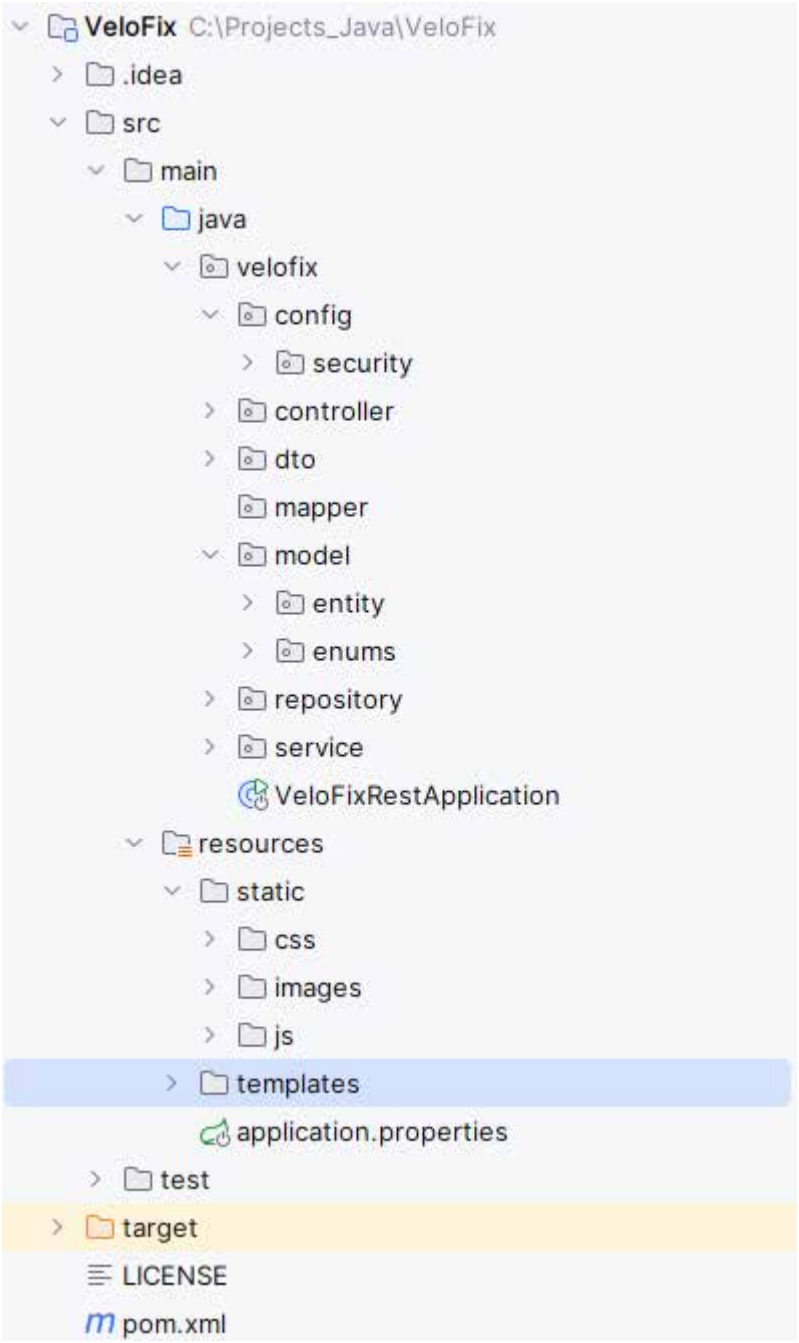

Рис. 3.1 – Візуалізація структури проєкту в IntelliJ IDEA

Інтеграція з Hibernate і PostgreSQL дозволяє розробникам працювати з базою даних напряму в межах IDE. Вбудовані інструменти для перегляду структури бази, написання та тестування SQL-запитів, а також механізми синхронізації сутностей з таблицями надають глибоке розуміння даних і полегшують контроль правильності їх обробки. Це особливо важливо при

реалізації системи обліку замовлень, гарантійних випадків і використання запчастин, де точність і цілісність даних мають вирішальне значення.

Водночас, редактор шаблонів Thymeleaf в IntelliJ робить процес створення веб-інтерфейсів зручним і інтуїтивним. Підсвічування синтаксису, автозаповнення тегів, швидкий перехід між контролерами і шаблонами дозволяють ефективно працювати над користувацьким інтерфейсом, швидко вносити корективи та оперативно тестувати результати без необхідності постійного переключення між різними програмними продуктами.

```
<div th:if="${order.repairParts != null and !order.repairParts.isEmpty()}">
    <h4>Parts</h4>
    <table>
        <thead>
        <tr>
            <th>Part name</th>
            <th>Price</th>
            <th>Quantity</th>
            <th>Sum</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="rp : ${order.repairParts}">
            <td th:text="${rp.part.name}">Part name</td>
            <td>
                <span th:if="${rp.part.price != null}"
                      th:text="${#numbers.formatDecimal(rp.part.price, 0, 'COMMA', 2, 'POINT') + ' USD'}"></span>
                <span th:unless="${rp.part.price != null}">N/A</span>
            </td>
            <td th:text="${rp.quantityUsed}">Qty</td>
            <td>
                <span th:if="${rp.part.price != null and rp.quantityUsed != null}"
                      th:text="${#numbers.formatDecimal(rp.part.price.doubleValue() * rp.quantityUsed, 0, 'COMMA', 2, 'POINT') + ' USD'}">
                </span>
                <span th:unless="${rp.part.price != null and rp.quantityUsed != null}">N/A</span>
            </td>
        </tr>
        </tbody>
    </table>
</div>
```

Рис. 3.2 – Редагування шаблонів Thymeleaf

Важливо також відзначити глибоку інтеграцію IntelliJ з системою контролю версій Git. Можливість відслідковувати зміни, працювати з гілками, швидко вирішувати конфлікти та формувати коміти безпосередньо в середовищі розробки значно підвищує ефективність командної роботи і забезпечує підтримку високого рівня організації процесів управління кодом.

Для забезпечення високої якості програмного продукту IntelliJ надає потужні засоби для запуску і відлагодження юніт-тестів. Тісна інтеграція з фреймворками JUnit та Mockito дозволяє не лише оперативно перевіряти окремі

функціональні модулі, а й отримувати розгорнуті звіти про результати тестування, що сприяє швидкому виявленню та усуненню помилок.

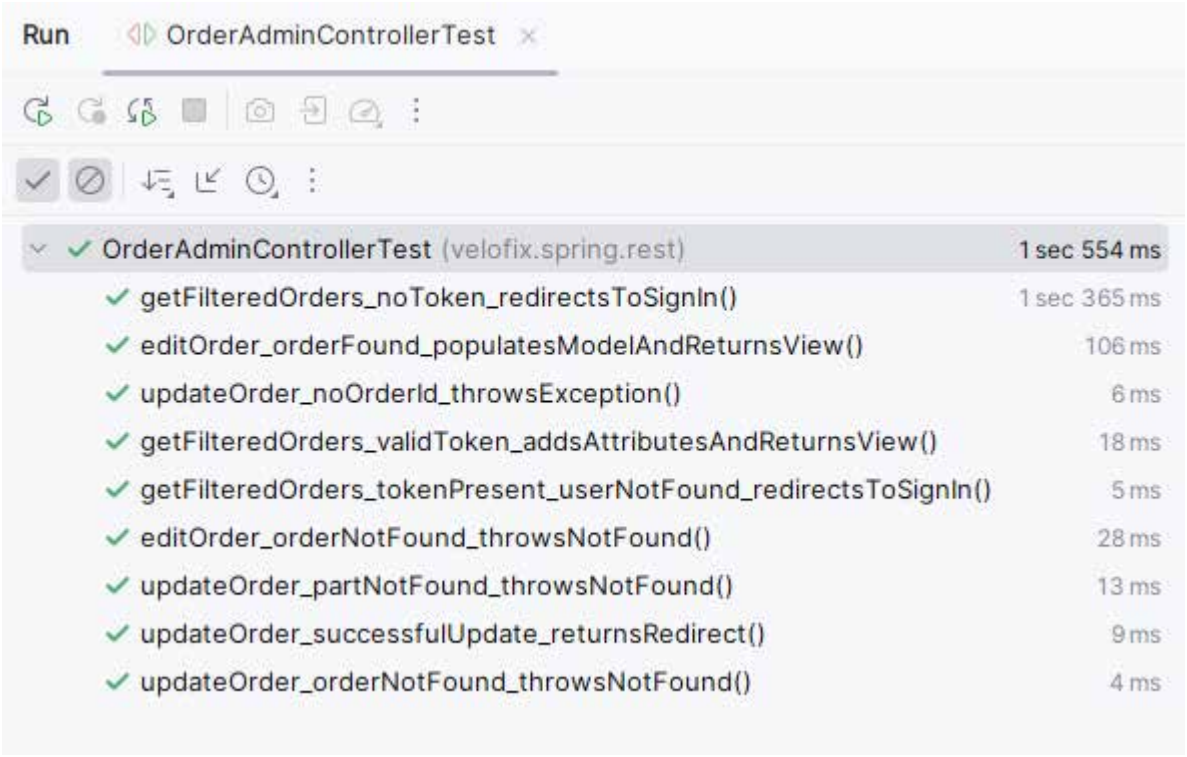

Рис. 3.3 – Запуск тестів та відлагодження

Також IntelliJ IDEA підтримує гнучкі механізми кастомізації та розширення функціоналу через плагіни, шаблони коду, інспекції та інструменти аналізу. Це допомагає дотримуватися єдиного стилю кодування, підвищувати якість проекту та зберігати чистоту архітектури, що є надзвичайно важливим у великих і складних системах, де чіткий поділ відповідальності між компонентами є основою стабільності і підтримуваності коду.

Загалом, IntelliJ IDEA — це не просто середовище розробки. Це комплексна інженерна платформа, яка супроводжує програмний продукт на всіх стадіях його життєвого циклу. Вона робить процес створення, тестування, налагодження і підтримки проєкту більш продуманим, організованим і комфортним, дозволяючи розробникам зосередитися на ключових задачах бізнес-логіки, а не на технічних деталях організації робочого процесу.

4 ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Тестування системи

Тестування програмного забезпечення сьогодні розглядається не просто як одна з фаз у процесі розробки, а як окрема, важлива галузь інженерії програмного забезпечення. Воно тісно пов’язане з такими ключовими напрямами, як управління якістю продукту, оцінка потенційних ризиків, моделювання поведінки системи у різних умовах та формальна верифікація відповідності технічним вимогам. У сучасному світі розробки складних інформаційних систем, подібних до системи сервісного центру з гарантійного ремонту велосипедів, тестування є не лише засобом перевірки правильності функціонування, а й інструментом комплексної оцінки відповідності системи високим стандартам якості. В рамках життєвого циклу програмного продукту, процес тестування виконує ключову роль у визначенні ступеня готовності системи до експлуатації, її стабільності, надійності, особливо в тих сферах, де відмови можуть мати серйозні наслідки — наприклад, у забезпеченні безперервності обслуговування клієнтів і збереженні важливої інформації.

Стандартизація процесу тестування

Для того, щоб забезпечити системність та узгодженість у проведенні тестування, існують міжнародні стандарти, що формалізують і структуризують тестові процедури. Одним із ключових документів є стандарт ISO/IEC/IEEE 29119, який пропонує детальний поділ процесу тестування на окремі фази із визначеними завданнями та методами їх виконання:

- Частина 29119-1 фокусується на встановленні єдиної термінології, що полегшує взаєморозуміння між розробниками, тестувальниками та менеджерами проектів.

- 29119-2 описує організаційні процеси тестування — від стратегічного планування, ідентифікації та аналізу ризиків до тактичного виконання тестів, збору необхідних метрик і формування звітності.
- У 29119-3 наведені вимоги до структури і змісту тестової документації, включно з тест-планами, тест-кейсами, чеклистами, таблицями зв’язку вимог і дефектів.
- 29119-4 класифікує різні техніки тестування, що базуються на моделюванні, досвіді, статистичних методах або аналізі коду (наприклад, контроль покриття умов та логічних шляхів).

Додатково, важливим є стандарт ISO/IEC 25010, який пропонує багатогранну модель якості програмного забезпечення. Ця модель не лише допомагає сформувати вимоги до продукту, але й служить орієнтиром для розробки комплексних стратегій тестування. Вона охоплює такі характеристики:

- Функціональна придатність — повна та коректна реалізація функцій, які задовольняють потреби користувачів;
- Надійність — здатність працювати без відмов навіть у стресових умовах, швидко відновлюватися після помилок;
- Продуктивність — забезпечення ефективної роботи системи при різних навантаженнях, своєчасна реакція на запити;
- Зручність використання — простота і комфорт взаємодії користувачів із системою, інтуїтивно зрозумілий інтерфейс;
- Безпека — захист конфіденційної інформації, належна аутентифікація та контроль доступу;
- Сумісність — коректна взаємодія з іншими програмними рішеннями, дотримання стандартів і протоколів;

- Підтримуваність — легкість внесення змін, адаптація до нових вимог без значних витрат часу і ресурсів;
- Переносимість — здатність функціонувати у різних середовищах з мінімальними змінами.

Види тестування

Оскільки інформаційна система сервісного центру має багаторівневу архітектуру, застосовується кілька рівнів тестування, кожен із яких має свою специфіку та завдання:

- Модульне тестування (Unit-тестування) — перевірка окремих частин системи в ізоляції, що дозволяє впевнитися у правильності реалізації базових функціональних одиниць. Зазвичай цей процес автоматизується за допомогою спеціалізованих фреймворків (наприклад, JUnit).
- Інтеграційне тестування — оцінка коректності взаємодії між модулями, що особливо важливо для перевірки передачі даних і узгодженості бізнес-логіки.
- Системне тестування — комплексне тестування всієї системи в умовах, максимально наближених до реального експлуатаційного середовища.
- Приймальне тестування — підтвердження того, що кінцевий продукт відповідає вимогам і очікуванням замовника, включаючи перевірку на основі конкретних сценаріїв використання.

Методики тестування

Сучасна практика включає комбінування різних підходів для більш повного покриття потенційних помилок та дефектів:

- Black-box testing — оцінка функціоналу без доступу до внутрішньої будови, що дозволяє зосередитися на поведінці системи у відповідь на різні вхідні дані.
- White-box testing — детальна перевірка логіки коду, охоплення всіх можливих шляхів виконання, перевірка циклів і умов.
- Grey-box testing — проміжний варіант, що використовує часткові знання про внутрішню структуру системи для більш цілеспрямованого тестування, наприклад, API або сервісних інтерфейсів.

Основні принципи тестування

Щоб забезпечити максимальну ефективність тестування, слід дотримуватись певних ключових принципів:

- Неможливо протестувати абсолютно все, тому важливо зосереджуватись на найбільш критичних та ризикованих аспектах.
- Раннє залучення тестування у життєвий цикл проекту допомагає виявити і усунути помилки ще до написання коду, що значно знижує витрати на виправлення.
- Вибір методів і стратегій тестування має відповідати особливостям конкретного продукту, галузі та бізнес-вимог.
- Необхідно регулярно оновлювати тестову базу, щоб уникнути "пестицидного ефекту", коли однакові тести перестають виявляти нові дефекти.
- Забезпечення прозорості і зв’язку між вимогами, тестами і дефектами через відповідні механізми трасування є важливою складовою контролю якості.

Отже, тестування інформаційної системи сервісного центру з гарантійного ремонту велосипедів — це ретельно організований, стандартизований і багатогранний процес, що забезпечує високу якість, надійність і безпеку кінцевого продукту, готового до стабільної експлуатації в реальних умовах.

Практична реалізація тестування програмного забезпечення

Практична частина тестування програмного забезпечення в рамках цього дослідження була зосереджена на комплексній перевірці веб-застосунку, розробленого за допомогою середовища IntelliJ IDEA з використанням мови програмування Java. Основною метою тестування було забезпечення високої якості та надійності системи шляхом детальної перевірки її функціональних та нефункціональних аспектів.

У процесі роботи було застосовано сучасні методики модульного та інтеграційного тестування, що дозволяють виявити помилки на ранніх етапах розробки, зменшуючи тим самим загальні витрати на виправлення дефектів і підвищуючи стабільність програми. Для реалізації тестових сценаріїв використовувались широко визнані інструменти — JUnit, який надає гнучкі засоби для організації і виконання автоматизованих тестів, та Mockito, що є потужним фреймворком для створення мок-об’єктів і імітації залежностей компонентів.

Особливості тестового середовища

Розробка та тестування здійснювалися у професійному середовищі розробки IntelliJ IDEA, що забезпечує ефективний цикл збірки, налагодження та автоматичного виконання тестів. Застосунок був побудований на основі

сучасних Java-технологій, а також використовував популярні фреймворки для створення веб-інтерфейсів і серверної логіки.

Із огляду на специфіку веб-застосунку, тестування передбачало перевірку взаємодії між різними модулями системи, зокрема контролерами, сервісами та репозиторіями, а також перевірку коректності бізнес-логіки. Враховувалася необхідність імітації зовнішніх сервісів, баз даних та API для забезпечення ізоляції тестованих компонентів, що реалізовано за допомогою Mockito.

Детальний опис процесу тестування

1. **Модульне тестування бізнес-логіки**

   Ключовим етапом було розроблення і виконання модульних тестів, спрямованих на окремі класи та методи застосунку. Завдяки JUnit була реалізована структура тестів, що охоплювала всі основні функціональні вимоги, а за допомогою Mockito імітувалися зовнішні залежності (наприклад, бази даних, сервіси аутентифікації та інші). Це дозволило не тільки перевірити коректність логіки, але й гарантувати, що зміни в одних компонентах не вплинуть негативно на інші.

2. **Інтеграційне тестування взаємодії компонентів**

   Окрім модульних тестів, здійснювалося інтеграційне тестування, що перевіряло коректність роботи окремих частин системи у спільній взаємодії. Особлива увага приділялась правильності обробки запитів, взаємодії з базою даних та відповідності результатів очікуваним. Тут також активно застосовувались мок-об’єкти, що дозволяло ізолювати середовище тестування від реальних зовнішніх ресурсів.

3. **Перевірка обробки виняткових ситуацій**

Для забезпечення стабільності системи було змодельовано різні варіанти помилкових станів, таких як відсутність даних, некоректні вхідні параметри, несправність зовнішніх сервісів тощо. Тестування підтвердило, що застосунок коректно обробляє виключення, надає інформативні повідомлення про помилки і не допускає аварійних зупинок.

4. **Тестування логіки доступу та безпеки**

   Особливий акцент був зроблений на перевірці механізмів аутентифікації і авторизації користувачів. Було ретельно протестовано ролі і права доступу, щоб гарантувати, що кожен користувач має доступ лише до дозволених функціональних частин системи.

5. **Автоматизація тестів і безперервна інтеграція**

   Усі тести були інтегровані у CI/CD процес, що забезпечувало автоматичне виконання тестів при кожному оновленні коду, дозволяючи оперативно виявляти і усувати дефекти.

Таким чином, комплексний підхід до тестування веб-застосунку із застосуванням JUnit і Mockito дозволив досягти високого рівня надійності, ефективності та відповідності якості сучасним стандартам розробки програмного забезпечення.

4.2 Апаратні та технічі засоби

Для стабільної роботи веб-сайту, особливо такого, де є різні ролі користувачів і багато інтерактивних функцій, важливо мати надійне апаратне та програмне забезпечення. Це допомагає уникнути затримок, зависань і проблем з

безпекою. Нижче наведені ключові вимоги до серверного обладнання, програмного середовища, мережі та клієнтських пристроїв, щоб гарантувати комфортну роботу системи.

Таблиця 1. Апаратні вимоги до серверної платформи

Параметр	**Мінімальні вимоги**	**Рекомендовані вимоги**
Процесор	2 ядра, 2.0 ГГц	4 ядра і вище, від 3.0 ГГц
Оперативна пам’ять	4 ГБ	8 ГБ і більше
Дисковий простір	50 ГБ HDD	100 ГБ SSD (швидкий доступ)
Мережевий інтерфейс	1 Гбіт/с	10 Гбіт/с для навантажених проектів
Операційна система	Linux (Ubuntu/Debian)	Нові версії Linux

Таблиця 2. Програмні вимоги до серверного середовища

Компонент	**Мінімальні вимоги**	**Рекомендовані вимоги**
Версія Java	Java 8	Java 17+
Фреймворк	Spring Boot 2.x	Spring Boot 3.x
База даних	PostgreSQL 13	PostgreSQL 17+
Сервер додатків	Apache Tomcat	Apache Tomcat або інші балансувальники
Система управління версіями	Git	Git + CI/CD (Jenkins, GitHub Actions)
Тестування	JUnit, Mockito	JUnit 5, інтеграційні тести

Таблиця 3. Вимоги до мережевої інфраструктури

Параметр	**Мінімальні вимоги**	**Рекомендовані вимоги**
Швидкість інтернету	100 Мбіт/с	1 Гбіт/с або більше
Протоколи безпеки	HTTPS (SSL сертифікат)	HTTPS з оновленими TLS протоколами
Стабільність мережі	Мінімальні затримки (<100 мс), втрати пакетів < 2%	Затримки <30 мс, майже без втрат пакетів
Балансування навантаження	Відсутнє або базове	Аппаратний/програмний балансувальник

Таблиця 4. Рекомендації по клієнтському обладнанню

Параметр	**Мінімальні вимоги**	**Рекомендовані вимоги**
Оперативна пам’ять	4 ГБ	8 ГБ і вище
Процесор	Двоядерний, 2.0 ГГц	Чотириядерний, від 3.0 ГГц
Браузери	Chrome, Firefox, Edge останніх 2-3 версій	Chrome, Firefox, Edge, Safari останніх версій
Підключення до мережі	Стабільний інтернет 5-10 Мбіт/с	Високошвидкісний інтернет 20 Мбіт/с і більше
Екран	Мінімум 1366x768	Full HD (1920x1080) або вище

Загальний висновок

Щоб веб-сайт працював швидко, без збоїв і був зручним для користувачів, важливо подбати про те, щоб всі складові — від сервера до пристроїв користувачів — відповідали хоча б мінімальним вимогам. Рекомендовані параметри дають запас для росту, дозволяють системі масштабуватися і впевнено витримувати збільшення кількості відвідувачів і функціоналу. Тільки при такому комплексному підході можна забезпечити комфортну, безпечну та стабільну роботу веб-ресурсу, який виконує свої завдання без зайвих пауз і технічних проблем.

4.3 Опис роботи програми

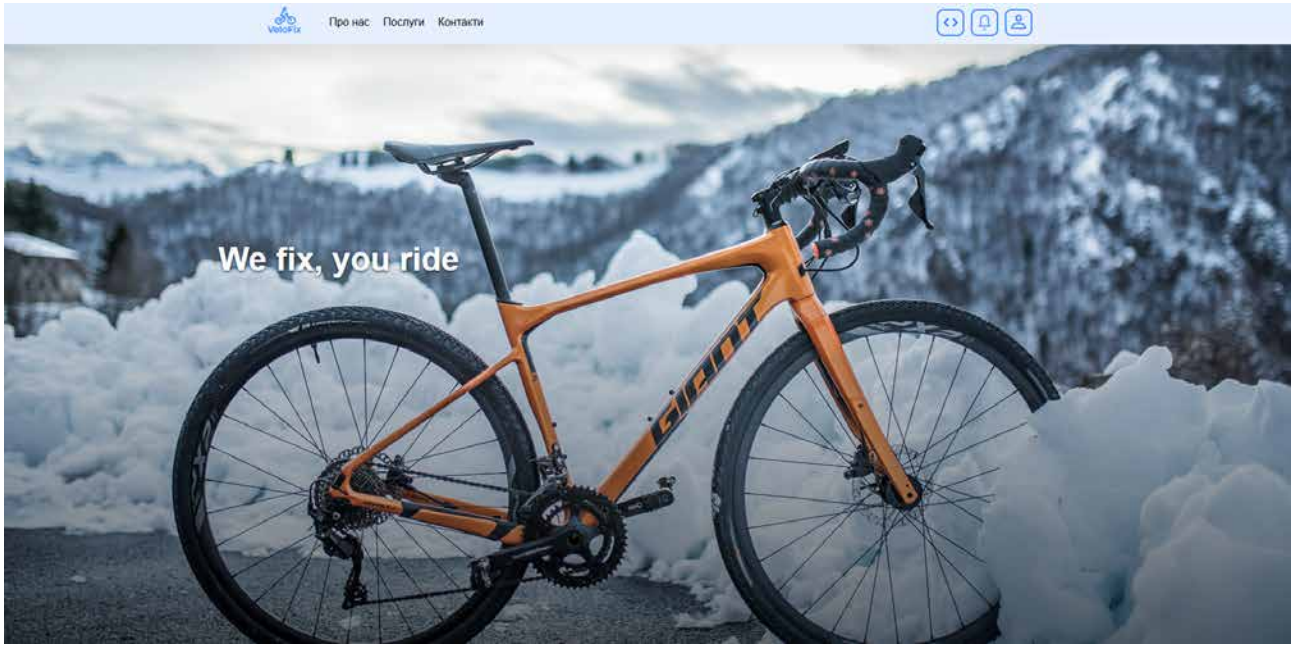

Рис. 4.1 - Головна сторінка

Головна сторінка — Вітрина душі VeloFIX

Коли користувач уперше потрапляє на головну сторінку вебсайту VeloFIX, він одразу занурюється в атмосферу турботи про велосипед і комфорт, що транслюється буквально з першого пікселя. Це не просто сайт — це майже онлайн-веломайстерня, де кожен елемент говорить: “Ми полагодимо, а ти катайся”.

Банер — «We fix, you ride»

На самому верху — велика заставка-банер, де красується зображення велосипеду на фоні світлого простору, що натякає на легкість, свободу й рух. Цей банер — не просто картинка, а фактично перше знайомство користувача з брендом. У центрі — коротке, як удар педалі, але влучне гасло: **“**We fix, you ride**”**. Воно ніби каже: «Тобі не потрібно турбуватись про технічні нюанси — просто довір нам свого двоколісного друга, і ми все зробимо як слід».

Структуровані секції (scroll на якорі)

Кожна навігаційна кнопка вгорі сторінки — Про нас**,** Послуги**,** Контакти— не переносить користувача на нову сторінку, а плавно прокручує вниз до відповідної секції. Це дає змогу не втрачати контекст і залишатися в потоці інтерфейсу, немов на сайті-односторінці з розумним UX-дизайном.

Про нас

Цей блок оформлений у вигляді великого інформаційного блоку з підзаголовком "Про нас"**,** що одразу занурює користувача в історію та суть сервісного центру. Тут пояснюється, що VeloFIX — це не просто пункт обслуговування, а надійне місце, де байки отримують друге дихання.

- Наголос зроблено на гарантійних ремонтах.
- Вказано, що також проводиться платне обслуговування після закінчення гарантії.
- Важлива деталь: описано рівень кваліфікації механіків і типи робіт — від дрібного підкручування до повної заміни трансмісії.
- Використовується емоційно-професійна лексика — “якість”, “прозорість”, “довіра”.

Це створює враження живого, клієнтоорієнтованого сервісу.

Послуги

Блок-перехід від загального до конкретного. Тут користувач дізнається про повний спектр ремонтних та сервісних послуг: гарантійні, негарантійні, діагностика, планове обслуговування. Це свого роду анонс нижчих блоків, що розбивають послуги детальніше на категорії.

Гарантія / Платне обслуговування

Дві основні картки, які структуровано описують послуги:

1. Гарантійний ремонт
    - Безкоштовна діагностика.
    - Ремонт заводських дефектів.
    - Оригінальні запчастини.
    - Прозора документація для клієнта.
2. Платне обслуговування
    - Доступне ціноутворення.
    - Професійний підхід.
    - Діагностика, заміна, модернізація.
    - Орієнтація на довготривалу експлуатацію.

Цей блок ідеально підходить як новачкам, так і досвідченим велосипедистам, бо роз’яснює, що саме можна очікувати й чому VeloFIX — це не просто сервіс, а велокультура.

Контакти

Остання секція — місце для встановлення зв’язку. Тут усе чітко:

- Адреса: вказано фізичне розташування з зображенням мапи.
- Телефон: виділено для термінового зв'язку.
- Email: для офіційних запитів і зворотного зв’язку.
- Графік роботи: зручно для планування візиту.

Поведінка при взаємодії з іконкою профілю

Якщо користувач натискає на іконку акаунту у шапці сайту, система перевіряє наявність активної сесії:

- Якщо сесія є (користувач залогінений), відбувається переадресація до особистого кабінету, залежно від ролі (Клієнт, Менеджер, тощо).

- Якщо сесії нема, користувач автоматично перенаправляється на сторінку авторизації яку зображено на рисунку 4.2 з можливістю реєстрації, сторінку яку зображено на рисунку 4.3.

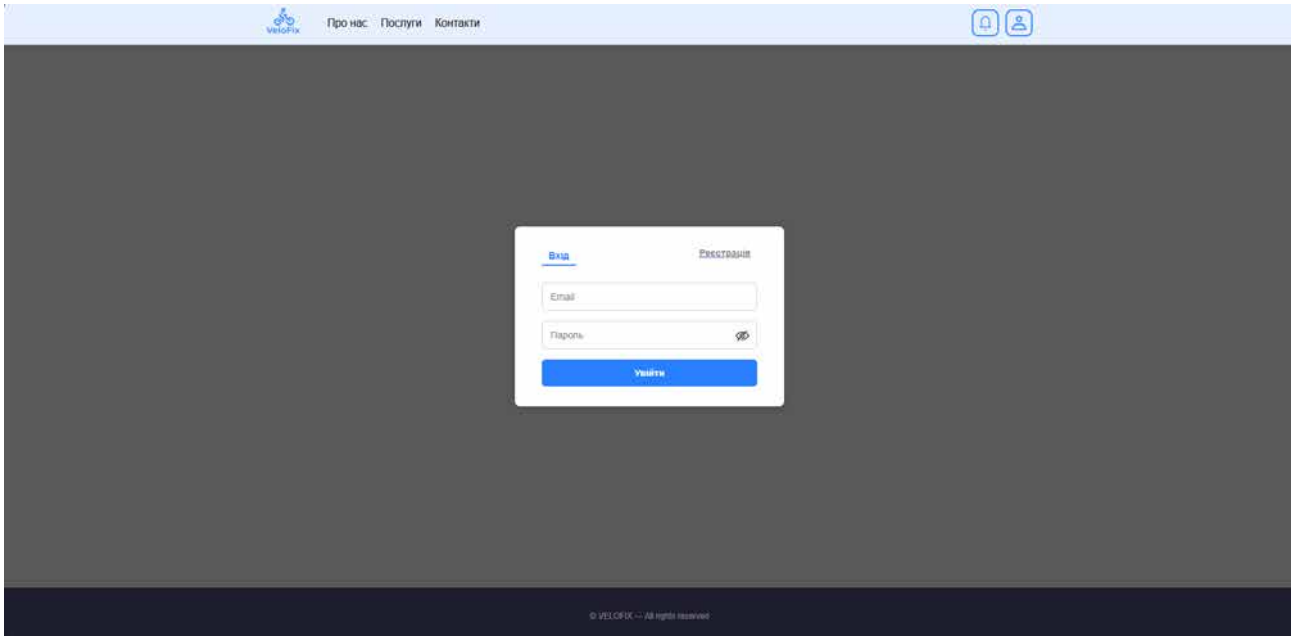

Рис. 4.2 – Сторінка авторизації

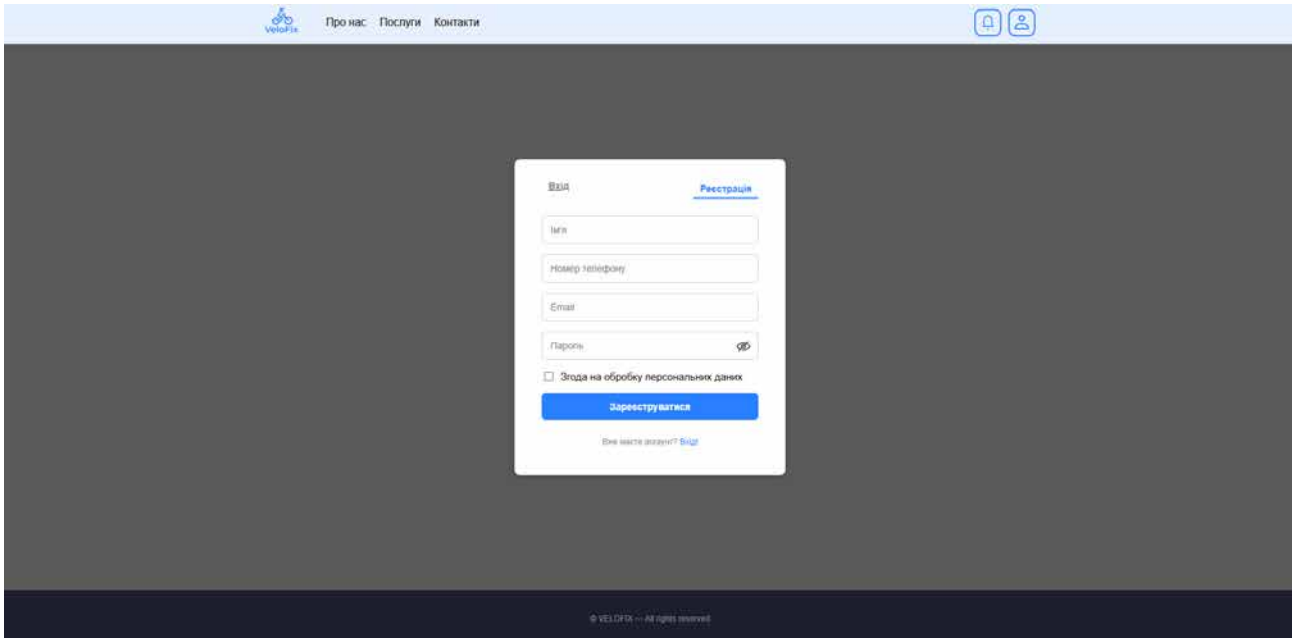

Рис. 4.3 – Сторінка реєстрації

На рисунку 4.4 зображено розділ створення заявки на ремонт — це серце персонального кабінету користувача в системі VeloFIX. Саме тут відбувається той момент, коли турбота про велосипед переходить у цифровий вимір — користувач ініціює процес, який надалі обробляється цілою екосистемою: менеджерами, механіками, директором та системою автоматизованого обліку запчастин.

Візуально — це окрема вкладка у кабінеті, яка одразу активна після входу. Інтерфейс — чистий, стриманий, без перевантаження. Акценти розставлені рівно настільки, наскільки потрібно, щоб користувач без зайвих думок міг швидко створити заявку.

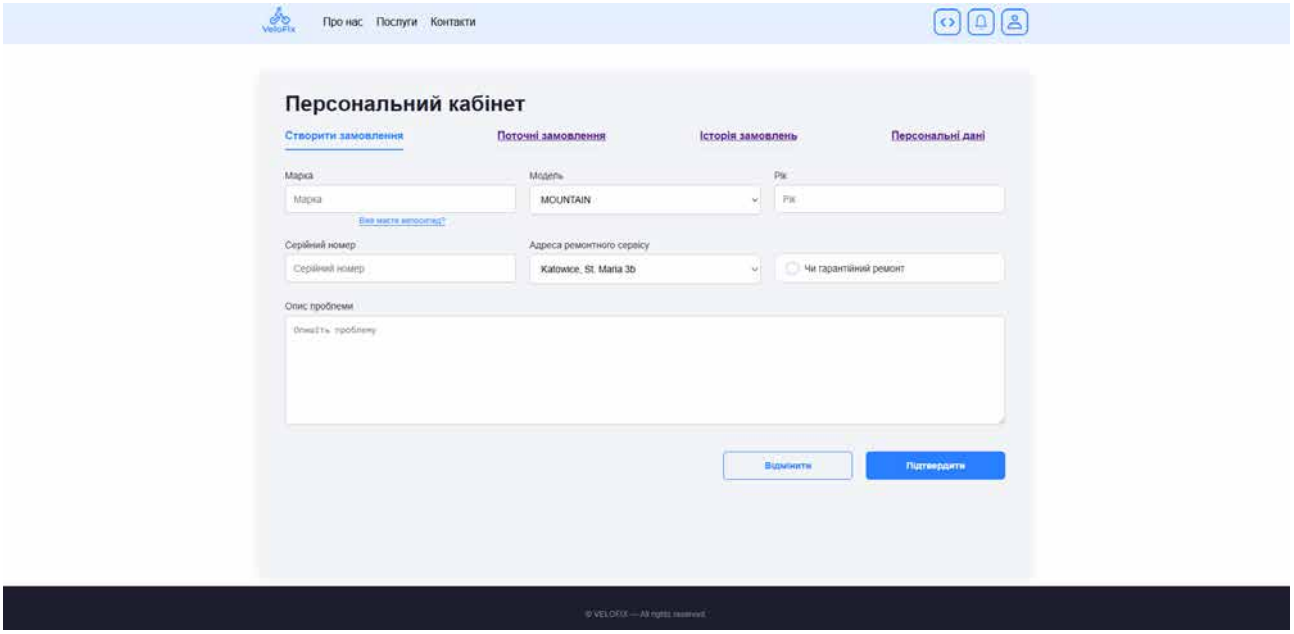

Рис. 4.4 – Сторінка створення нової завяки

Структура сторінки

Головний заголовок: Персональний кабінет, під ним — горизонтальне меню з вкладками:

- Створити замовлення
- Поточні замовлення

- Історія замовлень
- Персональні дані

Активною за замовчуванням є вкладка "Створити замовлення" — саме вона веде до форми створення нової заявки.

Форма створення заявки

Форма складається з кількох логічно структурованих блоків. Вони не просто так розбиті по рядках — це результат обміркованої UX-архітектури, яка не змушує користувача читати інструкції, а веде його поетапно.

Блок 1: Загальна інформація про велосипед

- **Марка:**

  Текстове поле з валідацією. Якщо користувач вже реєстрував велосипед, йому пропонується перейти за посиланням “Вже маєте велосипед” — це відкриває окрему гілку взаємодії з системою, яке зображено на рисунку 4.5, де дані заповнюються автоматично.

- **Модель:**

  Динамічний випадаючий список, наповнений через bicycleModels, який формується на бекенді на основі наявних у системі моделей.

- **Рік випуску:**

  Поле для числового вводу з обмеженнями в межах 1900–2100. Значення зберігається у базі як int.

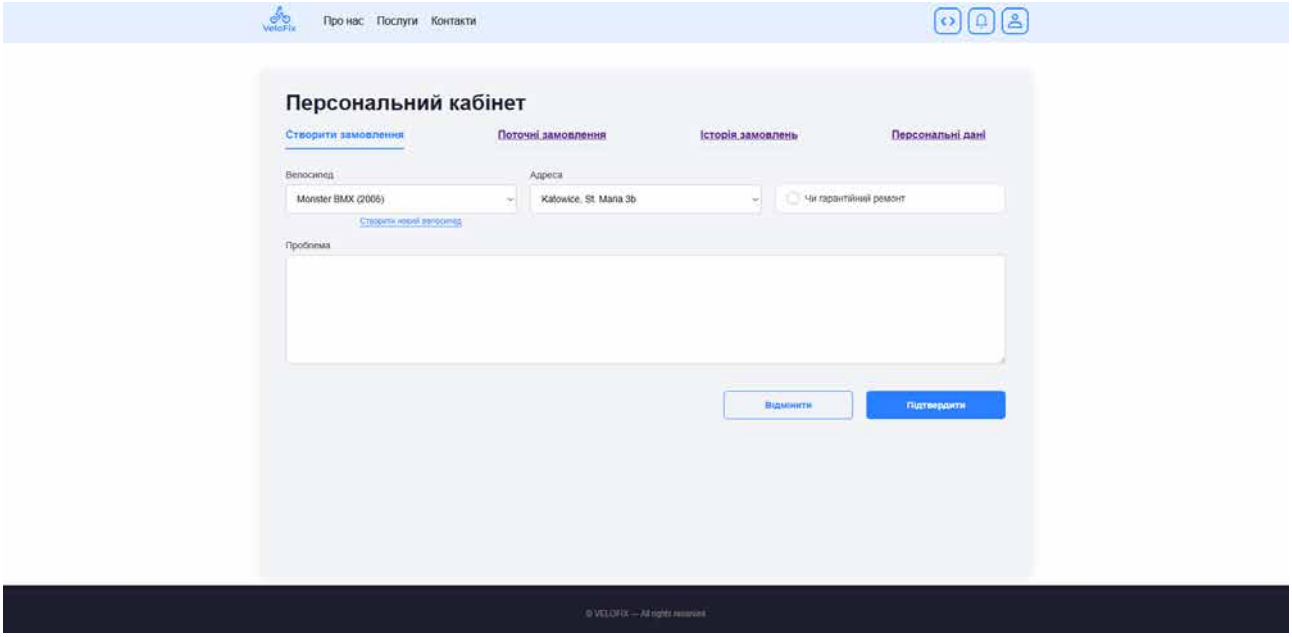

Рис. 4.6 – Сторінка створення заявки з зареєстрованим велосипедом

Блок 2: Технічна прив’язка

- **Серійний номер**:

  Унікальний текстовий ідентифікатор велосипеда, який дозволяє системі відслідковувати гарантійний статус та історію обслуговування.

- **Адреса майстерні**:

  Випадаючий список, що містить адреси наявних ремонтних точок.

- **Гарантійний ремонт**:

  Чекбокс. Його значення напряму впливає на бізнес-логіку: якщо обрано гарантію, заявка надалі обробляється з додатковими перевірками на відповідність умовам гарантії.

Блок 3: Опис проблеми

- Опис несправності:

Багаторядкове поле, де користувач власними словами описує суть проблеми. Це поле передається далі до менеджера, а той — до механіка. Саме тут міститься первинна “медична картка” велосипеда.

Фінальні дії

- Відмінити**:** Кнопка, яка не надсилає форму. Найчастіше повертає користувача назад до кабінету або скидає форму.
- Підтвердити**:** Відправка POST-запиту на endpoint /create із об’єктом OrderRequestDto. Далі — DTO проходить валідацію у Spring, і після успішної обробки — створюється новий запис у таблиці orders, прив’язаний до користувача через зовнішній ключ.

Бекенд-логіка

У Spring Boot ця форма прив’язана до контролера (наприклад, OrderController). Метод @PostMapping("/create") приймає DTO, перевіряє CSRF-токен, валідує всі поля, і лише тоді створює заявку.

Hibernate автоматично мапить DTO на ентіті Order, після чого запис зберігається в базу PostgreSQL. Система одразу асоціює заявку з користувачем (через ID з сесії), встановлює статус "Створено", і передає заявку менеджеру на обробку.

Інтеграція з іншими ролями

Після створення:

- Менеджер бачить нову заявку у своїй панелі.
- Він може призначити механіка, додати/змінити дані.
- Механік оновлює статуси, додає деталі ремонту, фіксує використані запчастини.

- Директор у разі конфліктів або складних випадків може втрутитися або переглянути статистику.

Після того, як користувач створює заявку на ремонт, взаємодія з системою не завершується — вона лише починається. Адже кожне звернення, кожна заявка — це динамічний процес, що розгортається у часі: її потрібно відстежувати, отримувати оновлення, а згодом — зберігати у вигляді історії для зручного доступу. Саме для цього в особистому кабінеті реалізовано три взаємопов'язані вкладки: Поточні замовлення**,** Історія замовлень та Персональні дані. І кожна з них виконує свою чітку, але дуже важливу функцію.

Поточні замовлення

На рисунку 4.7 зображено розділ, який слугує вікном у поточний статус ремонту. Він створений для того, щоб користувач завжди знав, що відбувається з його велосипедом: чи розпочато ремонт, чи очікуються деталі, чи, можливо, вже готовий до видачі.

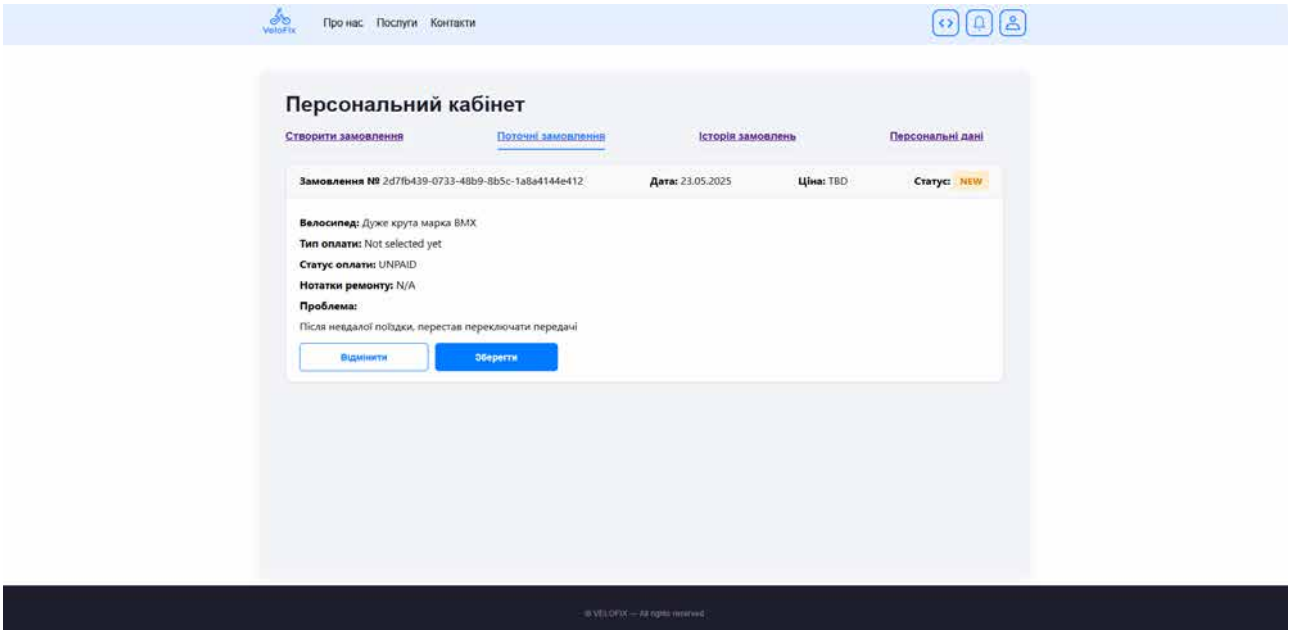

Рис. 4.7 – Сторінка поточних замовлень

У структурі відображення кожного замовлення є кілька інформативних блоків:

- У верхній частині розташовується стислий огляд заявки — номер, дата створення, орієнтовна вартість (якщо вона вже розрахована), та поточний статус.
- Натискання на блок розгортає деталі — відображається модель велосипеда, опис проблеми, тип та статус оплати, а також нотатки механіка.
- Тут же можна побачити і систему дій: якщо заявка ще не у виконанні або не завершена, користувач може підтвердити чи скасувати її. При натисканні на кнопку скасування відкривається модальне вікно з підтвердженням — адже помилкове натискання не повинно мати незворотних наслідків.

Історія замовлень

Як тільки ремонт завершено, відповідна заявка зникає з поточного списку і переміщується до розділу History, який зображено на рисунку 4.8. Цей блок виконує роль архіву, де зберігається хронологія всіх минулих ремонтів.

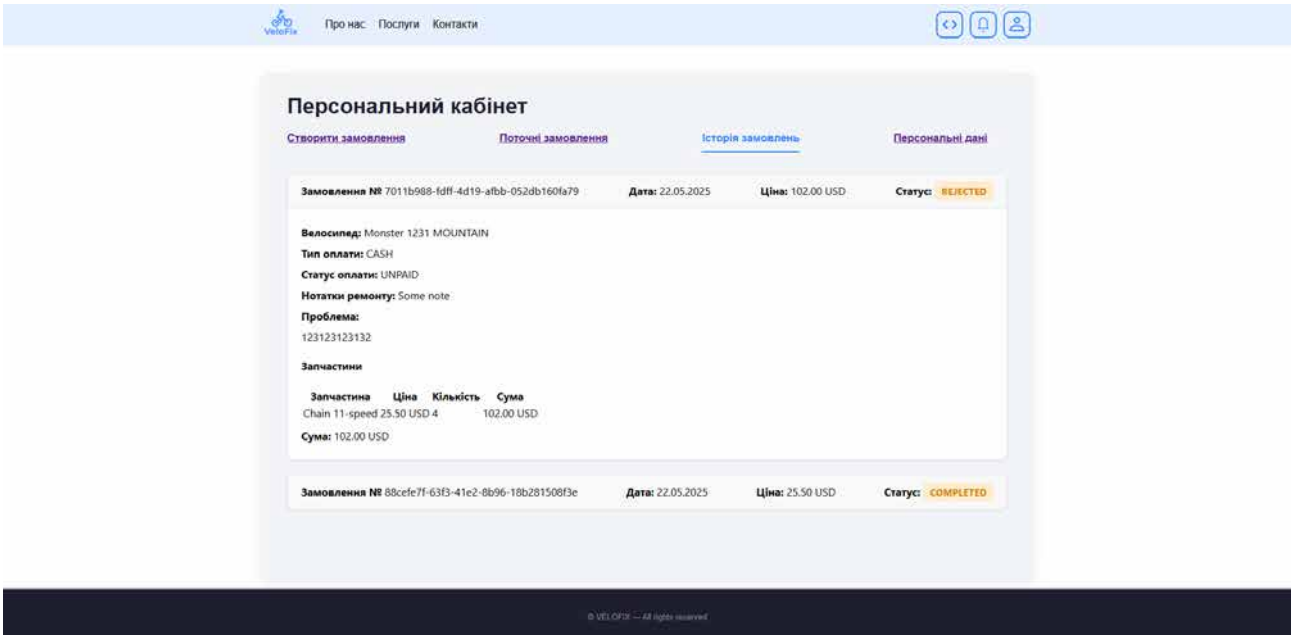

Рис. 4.8 – Сторінка історії замовлень

У цьому розділі користувач має змогу:

- Ознайомитися з повною інформацією по завершеним заявкам.
- Переглянути, які проблеми було усунено, які запчастини використано, скільки це коштувало.
- У разі необхідності — використовувати стару заявку як шаблон для нової (наприклад, якщо потрібно відправити той самий велосипед знову в ремонт).

Таким чином, Історія замовлень — це більше, ніж просто список — це зручний інструмент ретроспективного контролю і цифрова пам'ять користувача про свій велосипед.

Персональні дані

Окрему вкладку займає розділ персональні дані— яку зображено на рисунку 4.9, тут зібрані особисті дані користувача: ім’я, контактна інформація, а також налаштування профілю.

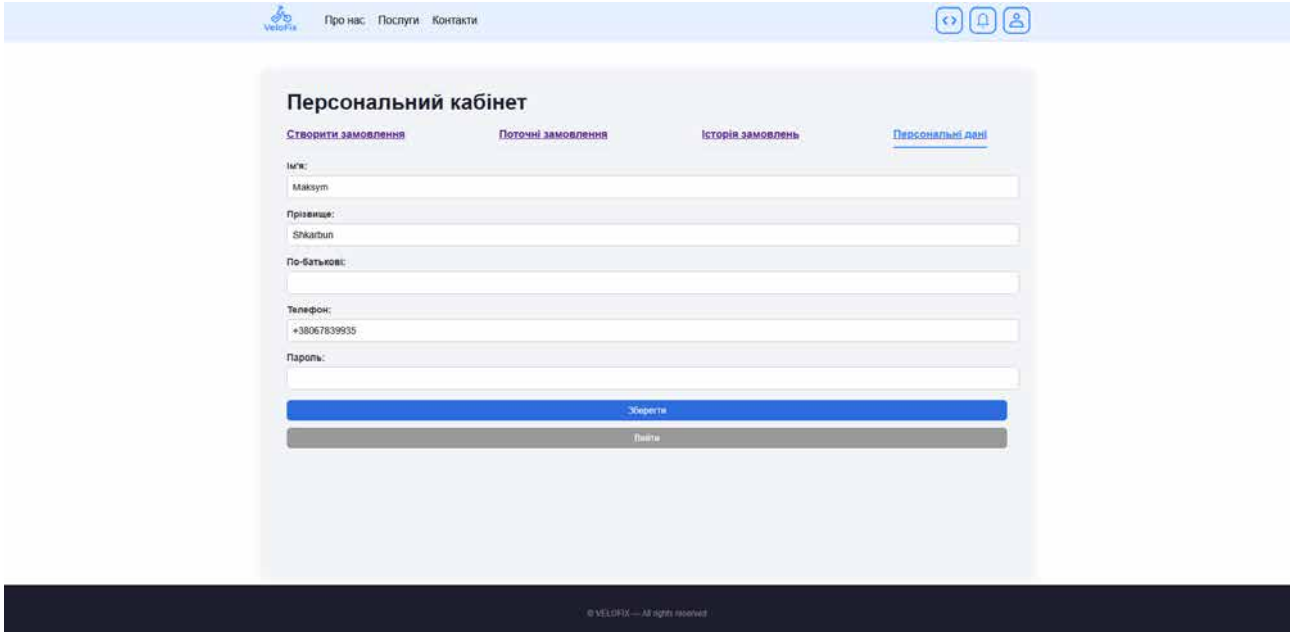

Рис. 4.9 – Сторінка персональних даних

Це місце для:

- Оновлення контактного номеру або ПІБ.
- Зміни пароля.

Цей розділ особливо важливий з точки зору безпеки та персоналізації облікового запису. Своєчасне оновлення даних гарантує, що зв’язок між клієнтом і сервісом буде швидким та ефективним.

Таким чином, вкладки Поточні замовлення, Історія замовлень та Персональні дані створюють повноцінну екосистему супроводу користувача впродовж усього життєвого циклу заявки — від створення до архівування. Це логічна, зрозуміла та максимально зручна структура, що дозволяє контролювати стан ремонту, мати доступ до всієї історії взаємодії із сервісом та залишатися на зв’язку.

Після того, як було детально розглянуто вкладки клієнтського кабінету, логічним кроком є перехід до особливого розділу, що відкривається лише для користувачів з адміністративними повноваженнями. Якщо авторизований користувач належить до однієї з службових ролей — менеджер, механік або директор — у правій частині верхньої панелі, поруч із іконкою профілю, з’являється спеціальний значок, що сигналізує про наявність розширених можливостей.

Натискання на цей значок відкриває випадаюче меню, в якому адміністратор може обрати один із трьох ключових розділів:

Панель керування користувачами

Цей інтерфейс — який зображено на рисунку 4.10, центр адміністрування доступу та ролей у системі.

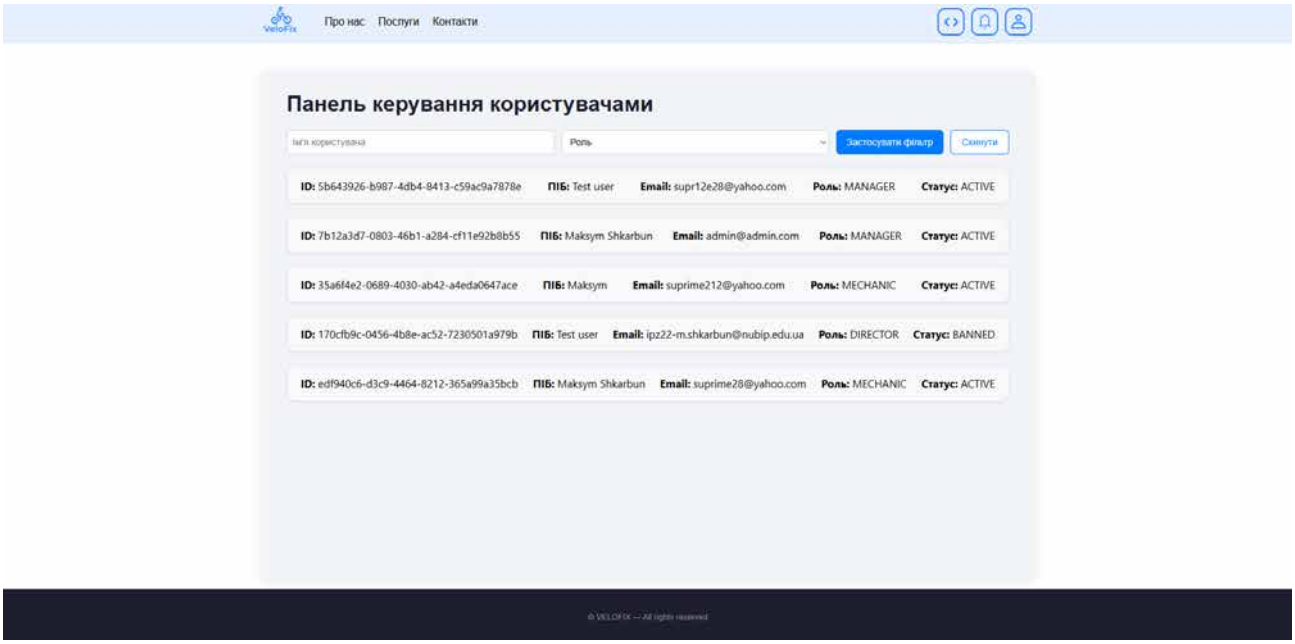

Рис. 4.10 – Сторінка керування користувачами

У верхній частині сторінки розташована фільтраційна форма, що дозволяє знайти користувачів за ім’ям або роллю. Це особливо зручно в компанії з великою кількістю клієнтів та співробітників.

Нижче — список користувачів у вигляді компактних блоків. Кожен блок містить базову інформацію: ID, ПІБ, email, роль, статус акаунту. Клік на користувача розгортає деталі з можливістю редагування ролі та статусу (наприклад, «ACTIVE», «BLOCKED»). Важливий момент: менеджери мають обмежені права — вони не можуть змінювати статус або роль інших менеджерів та директорів. Це реалізовано як логічний бар’єр для запобігання ескалації повноважень.

Панель керування запчастинами

Цей розділ присвячений усьому, що пов’язане з обліком, класифікацією та поповненням складу запчастин, його зображено на рисунку 4.11.

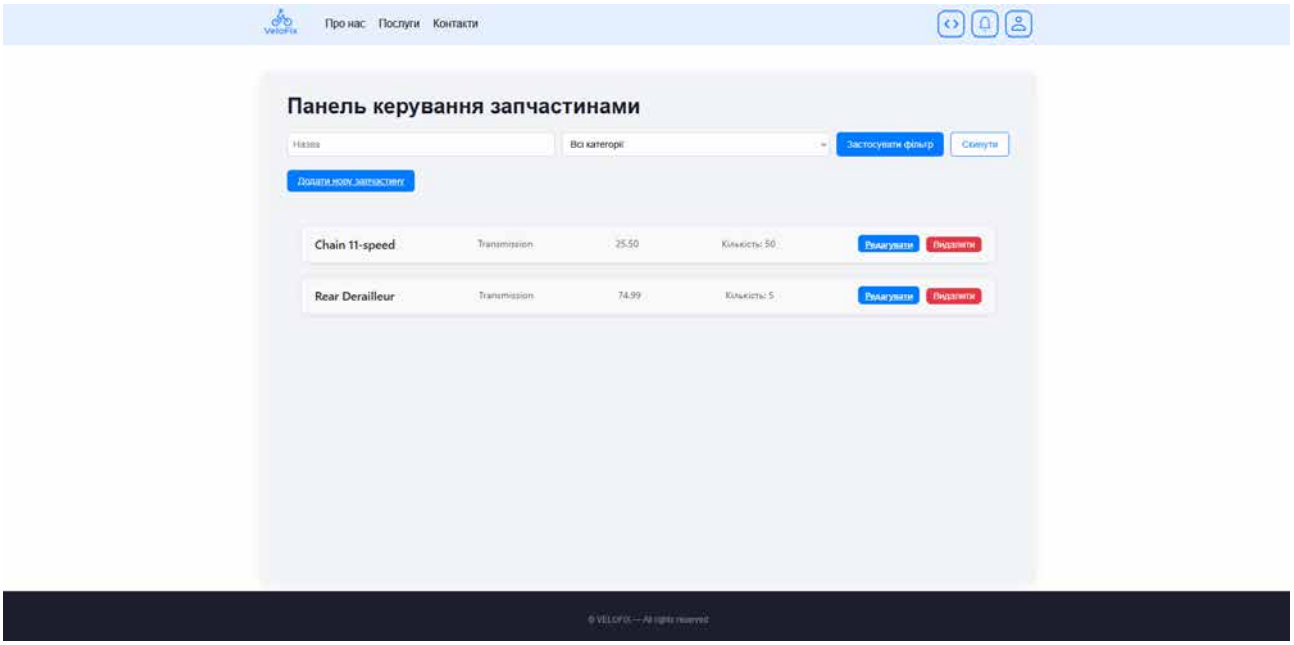

Рис. 4.11 – Сторінка укправління запчастинами

Форма фільтрації дозволяє знаходити деталі за назвою або категорією (наприклад, «гальма», «ланцюги», «перемикачі»). Поруч розташована кнопка Додати нову запчастину — для додавання нової запчастини з формою створення яка зображена на рисунку 4.12.

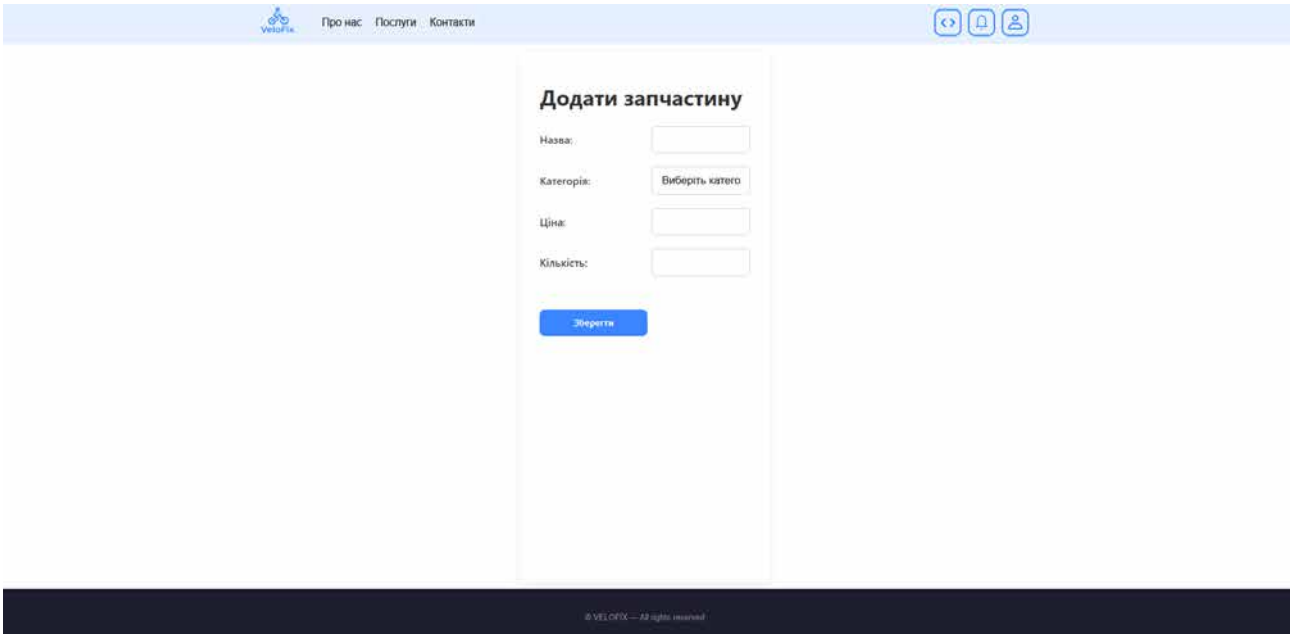

Рис. 4.12 – Сторінка додавання нової запчастини

Кожна деталь виводиться у вигляді блоку з такими даними:

- Назва
- Категорія
- Ціна
- Кількість на складі

Поруч доступні дві кнопки: Редагувати (редагування деталі) та Видалити (видалення з підтвердженням). Ці дії забезпечують гнучкість та оперативність у роботі зі складом.

Order Management (Адміністрування замовлень)

Найбільш функціонально насичена частина адмінки який зображено на рисунку 4.13. Тут сконцентрована вся інформація про створені клієнтами заявки, а також можливість сортування, перегляду і часткового втручання в деталі.

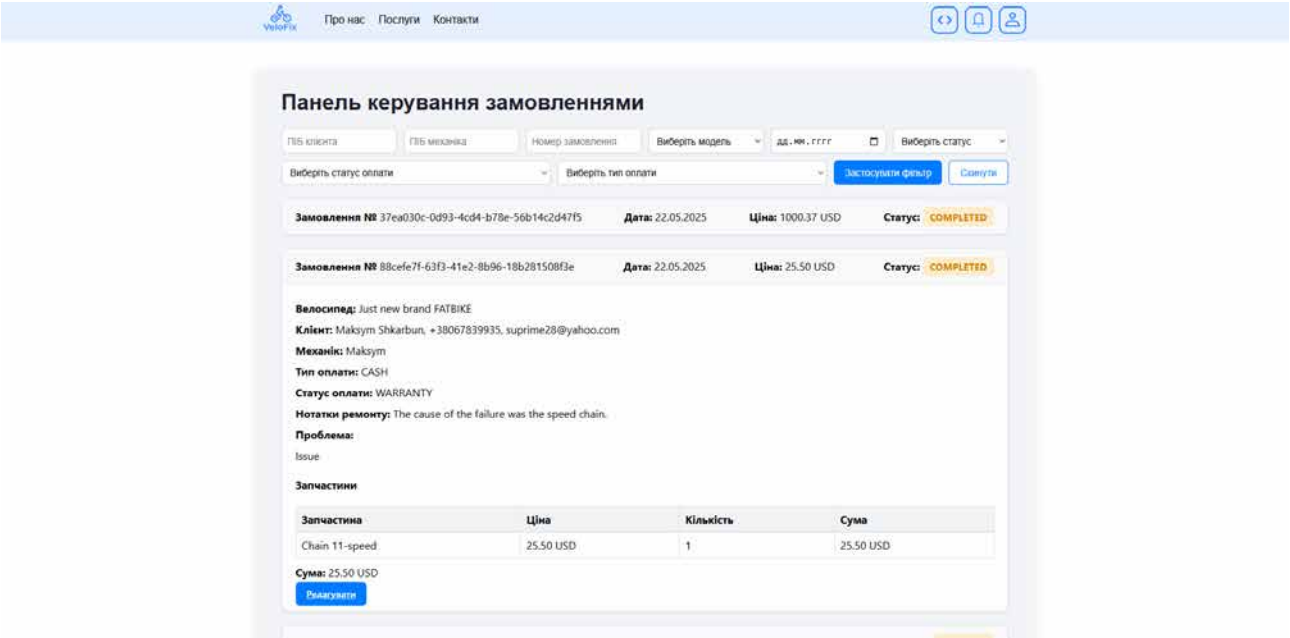

Рис. 4.13 – Сторінка адміністрування замовлень

Інтерфейс фільтрації дозволяє відсіяти замовлення за безліччю параметрів:

- ПІБ клієнта та механіка
- Номер замовлення
- Модель велосипеда
- Дата створення
- Статус ремонту
- Статус і тип оплати

Нижче відображається список замовлень, кожне з яких розгортається по кліку. В розширеному вигляді подається:

- Інформація про велосипед
- Дані клієнта та призначеного механіка
- Примітки по ремонту
- Оплата та використані запчастини (з розрахунком загальної вартості)

Рольова модель визначає, хто з адмінів може редагувати певні поля. Кнопка Редагувати веде до сторінки редагування заявки, яка зображена на рисунку 4.14, де дозволено вносити зміни в статус, оплату, призначення тощо.

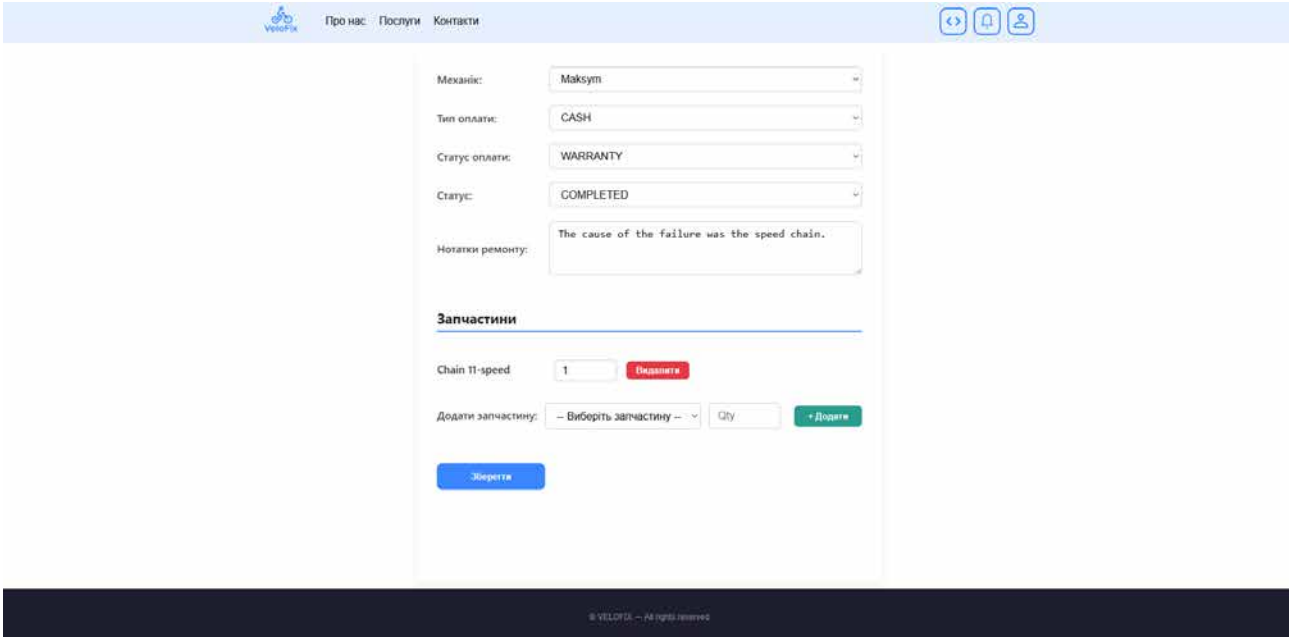

Рис. 4.14 – Сторінка редагування заявки

Таким чином, адміністративна панель — це інструмент контролю, прозорості та швидкої реакції на потреби сервісного процесу. Вона поєднує зручну візуалізацію, розумну систему фільтрів та розмежування прав доступу для забезпечення безпеки й ефективності всієї екосистеми VeloFIX.

ВИСНОВКИ

У межах бакалаврської кваліфікаційної роботи було розроблено повноцінну інформаційну систему для сервісного центру, що спеціалізується на гарантійному ремонті велосипедів. Результатом проєкту стало створення веб-застосунку, що дозволяє ефективно автоматизувати ключові бізнес-процеси сервісного обслуговування, включаючи обробку заявок, контроль за виконанням ремонтних робіт, облік запчастин, управління персоналом і взаємодію з клієнтами.

У ході дослідження:

- Проведено аналіз предметної області та визначено актуальні проблеми в організації гарантійного обслуговування велосипедів.
- Побудовано структурні й поведінкові UML-діаграми, що забезпечили формалізацію функціональних вимог і логіки бізнес-процесів.
- Розроблено та реалізовано ER-модель бази даних у PostgreSQL із застосуванням принципів нормалізації та використанням ENUM-типів.
- Побудовано багаторівневу архітектуру застосунку на основі Spring Boot із підтримкою ролей (Клієнт, Менеджер, Механік, Директор), що гарантує контроль доступу до функціоналу.
- Забезпечено безпеку системи шляхом інтеграції Spring Security та сесійного зберігання авторизації.
- Реалізовано зручний веб-інтерфейс на основі Thymeleaf, що динамічно генерує сторінки залежно від ролі користувача.
- Проведено юніт-тестування ключових компонентів із використанням JUnit та Mockito.

Інформаційна система успішно виконує поставлені завдання: забезпечує прозору взаємодію клієнта з сервісом, мінімізує людський фактор, прискорює

обробку заявок і надає керівництву сервісного центру аналітичні інструменти для оцінки ефективності роботи.

Таким чином, реалізоване рішення є не лише навчальним прикладом сучасної веб-розробки, а й має повноцінний потенціал для впровадження в реальних умовах. Проєкт демонструє високий рівень інтеграції бізнес-аналітики, об'єктно-орієнтованого програмування та практичного застосування інженерії програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bloch, Joshua. *Effective Java*. 3rd Edition. Addison-Wesley, 2018.
2. Walls, Craig. *Spring in Action*. 6th Edition. Manning Publications, 2022.
3. Richards, Mark; Ford, Neal. *Fundamentals of Software Architecture*. O'Reilly Media, 2020.
4. Cosmina, Iuliana et al. *Pro Spring Boot 3*. Apress, 2023.
5. Janssen, Thorben. *Hibernate Tips: More than 70 Solutions to Common Hibernate Problems*. Leanpub, 2018.
6. Mihalcea, Vlad. *High-Performance Java Persistence*. Amazon Digital Services, 2016.
7. Baeldung.com – матеріали та гайди по Spring Boot, Spring Security, Hibernate.
8. PostgreSQL Documentation [Електронний ресурс] – Режим доступу: https://www.postgresql.org/docs/
9. Spring Framework Documentation [Електронний ресурс] – Режим доступу: https://spring.io/guides
10. Hibernate ORM Documentation [Електронний ресурс] – Режим доступу: https://hibernate.org/orm/documentation/
11. Jakarta EE Specification Documents [Електронний ресурс] – Режим доступу: https://jakarta.ee/specifications/
12. Thymeleaf Documentation [Електронний ресурс] – Режим доступу: https://www.thymeleaf.org/documentation.html

ДОДАТОК А

База даних

Сторінок 3

Київ-2025

```
CREATE TYPE user_status AS ENUM ('ACTIVE', 'BANNED');

CREATE TYPE user_role AS ENUM ('CLIENT', 'MECHANIC', 'MANAGER', 'DIRECTOR');

CREATE TYPE payment_type AS ENUM ('CASH', 'CREDIT CARD');

CREATE TYPE repair_status AS ENUM ('NEW', 'IN_PROGRESS', 'WAINTING',
'COMPLETED', 'REJECTED');

CREATE TYPE payment_status AS ENUM ('UNPAID', 'PAID', 'WARRANTY');

CREATE TYPE bicycle_model AS ENUM (

    'MOUNTAIN', 'ROAD', 'HYBRID', 'CITY', 'BMX', 'ELECTRIC',

    'KIDS', 'TOURING', 'CYCLOCROSS', 'FATBIKE'

);


CREATE TABLE users (

    id UUID PRIMARY KEY,

    first_name VARCHAR(100),

    last_name VARCHAR(100),

    email VARCHAR(255),

    password VARCHAR(255),

    phone VARCHAR(20),

    role user_role,

    status user_status,

    signup_date DATE,

    last_login_at TIMESTAMP

);


CREATE TABLE usersession (

    sessionid UUID PRIMARY KEY,

    userid UUID REFERENCES users(id) ON DELETE CASCADE,

    sessiontoken VARCHAR(255),

    createdat TIMESTAMP,

    expiresat TIMESTAMP

);
```

```
CREATE TABLE bicycle (
    id UUID PRIMARY KEY,
    customer_id UUID REFERENCES users(id) ON DELETE CASCADE,
    brand VARCHAR(100),
    model bicycle_model,
    year INTEGER,
    warranty_status BOOLEAN,
    serial_number VARCHAR(100)
);

CREATE TABLE orders (
    id UUID PRIMARY KEY,
    bicycle_id UUID REFERENCES bicycle(id) ON DELETE CASCADE,
    mechanic_id UUID REFERENCES users(id),
    customer_id UUID REFERENCES users(id),
    status repair_status,
    payment_status payment_status,
    payment_type payment_type,
    repair_duration TIME,
    is_warranty_repair BOOLEAN,
    total_cost NUMERIC(38,2),
    repair_note VARCHAR(255),
    issue_description VARCHAR(255),
    created_at TIMESTAMP
);

CREATE TABLE notification (
    id UUID PRIMARY KEY,
    recipient_id UUID REFERENCES users(id) ON DELETE CASCADE,
    related_order_id UUID REFERENCES orders(id) ON DELETE SET NULL,
```

```
title VARCHAR(200),
    message TEXT,
    is_read BOOLEAN,
    created_at TIMESTAMP
);

CREATE TABLE part_categories (
    id UUID PRIMARY KEY,
    name VARCHAR(100)
);

CREATE TABLE part (
    id UUID PRIMARY KEY,
    category_id UUID REFERENCES part_categories(id),
    name VARCHAR(100),
    stock_quantity INTEGER,
    price NUMERIC(10,2)
);

CREATE TABLE repair_part (
    order_id UUID REFERENCES orders(id) ON DELETE CASCADE,
    part_id UUID REFERENCES part(id) ON DELETE CASCADE,
    quantity_used INTEGER,
    PRIMARY KEY (order_id, part_id)
);
```

ДОДАТОК Б

Код програми

Сторінок 3

Київ-2025

Сторінка 1

Конфігурація безпеки Spring Security.

```
@Configuration
public class SecurityConfig {

    private final CustomUserDetailsService userDetailsService;
    private final CustomAuthenticationSuccessHandler successHandler;

    public SecurityConfig(CustomUserDetailsService userDetailsService,
                          CustomAuthenticationSuccessHandler successHandler) {
        this.userDetailsService = userDetailsService;
        this.successHandler = successHandler;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
                .csrf(Customizer.withDefaults())
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers("/admin/**").hasAnyRole("MECHANIC",
"MANAGER", "DIRECTOR")
                        .requestMatchers("/create", "/order",
"order/history","personal").authenticated()
                        .anyRequest().permitAll()
                )
                .formLogin(form -> form
                        .loginPage("/signIn")
                        .loginProcessingUrl("/login")
                        .successHandler(successHandler)
                        .permitAll()
                )
                .logout(logout -> logout
                        .logoutUrl("/logout")
                        .deleteCookies("sessiontoken")
                        .logoutSuccessUrl("/signIn")
                )
                .userDetailsService(userDetailsService)
                .addFilterBefore(sessionTokenAuthFilter,
UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

    @Autowired
    private SessionTokenAuthFilter sessionTokenAuthFilter;

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Сторінка 2

Контролер автентифікації для реєстрації нового користувача.

```
@Controller
public class AuthController {

    private final UserRepository userRepository;
    private final JdbcTemplate jdbcTemplate;
    private final PasswordEncoder passwordEncoder;

    @Autowired
    public AuthController(UserRepository userRepository,
                          JdbcTemplate jdbcTemplate,
                          PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.jdbcTemplate = jdbcTemplate;
        this.passwordEncoder = passwordEncoder;
    }

    @GetMapping("/signup")
    public String showSignUpForm(Model model) {
        model.addAttribute("userDto", new UserRegistrationDto());
        return "signUp";
    }

    @PostMapping("/signup")
    public String registerUser(@ModelAttribute("userDto") @Valid
UserRegistrationDto dto,
                               BindingResult bindingResult,
                               Model model) {

        if (userRepository.findByEmail(dto.getEmail()).isPresent()) {
            bindingResult.rejectValue("email", "error.user", "Email already
registered");}
        if (userRepository.findByPhone(dto.getPhone()).isPresent()) {
            bindingResult.rejectValue("phone", "error.user", "Phone number
already registered");}
        if (bindingResult.hasErrors()) {return "signUp";}
        User user = new User();
        user.setId(UUID.randomUUID());
        user.setFullName(dto.getFullname());
        user.setPhone(dto.getPhone());
        user.setEmail(dto.getEmail());
        user.setPassword(passwordEncoder.encode(dto.getPassword()));
        user.setStatus(UserStatus.ACTIVE);
        user.setRole(UserRole.CLIENT);
        user.setSignupDate(LocalDate.now());

       user.setLastLoginAt(null);
userRepository.save(user);

        return "signIn";
  }}
```

Сторінка 3

Обробка створення нового замовлення на ремонт велосипеда.

```
@PostMapping("/create")
public String createOrder(@ModelAttribute("orderRequestDto") @Valid
OrderRequestDto dto,
                          BindingResult bindingResult,
                          @CookieValue(value = "sessiontoken", required = false)
String token,
                          Model model) {

    if (token == null || authService.getUserIdFromSession(token).isEmpty()) {
        return "redirect:/signIn";
    }

    if (bindingResult.hasErrors()) {
        model.addAttribute("bicycleModels", BicycleModel.values());
        model.addAttribute("repairShops",
userRepository.findByRole(UserRole.MANAGER));
        return "CreateNewBidWithNewBicycle";
    }

    UUID customerId = authService.getUserIdFromSession(token).get();
    User customer = userRepository.findById(customerId)
            .orElseThrow(() -> new UsernameNotFoundException("User not found"));

    Bicycle bicycle = new Bicycle();
    bicycle.setId(UUID.randomUUID());
    bicycle.setCustomer(customer);
    bicycle.setBrand(dto.getBrand());
    bicycle.setModel(dto.getModel());
    bicycle.setYear(dto.getYear());
    bicycle.setSerialNumber(dto.getSerialNumber());
    bicycle.setWarrantyStatus(dto.isWarrantyRepair());
    bicycleRepository.save(bicycle);

    Order order = new Order();
    order.setId(UUID.randomUUID());
    order.setBicycle(bicycle);
    order.setCustomer(customer);
    order.setStatus(RepairStatus.NEW);
    order.setPaymentStatus(PaymentStatus.UNPAID);
    order.setWarrantyRepair(dto.isWarrantyRepair());
    order.setIssueDescription(dto.getIssueDescription());
    order.setCreatedAt(LocalDateTime.now());
    orderRepository.save(order);
    return "redirect:/order";
}
```

ДОДАТОК В

Посилання на репозиторій: https://github.com/Yuklivud/VeloFix/tree/dev