Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного застосунку для менеджменту та обліку послуг автосервісу з використанням технології JavaFX і бази даних MySQL»

Виконав: студент 4 курсу
групи 1ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Сопотніцький О.Є.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Савицька Л.А.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«21» 03 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Сопотніцькому Олексію Євгеновичу

1. Тема роботи – «Розробка програмного застосунку для менеджменту та обліку послуг автосервісу з використанням технології JavaFX і бази даних MySQL»

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: вимоги користувачів; структура бази даних; архітектура застосунку; технології JavaFX і MySQL; макети форм; сценарії взаємодії користувачів; графічний інтерфейс.

4. Зміст розрахунково-пояснювальної записки: вступ; розвитку технологій розробки програмних застосунків для менеджменту та обліку послуг автосервісу та постановка задач дослідження; розробка архітектури та алгоритмів роботи програмної системи менеджменту та обліку послуг автосервісу; розробка програмних модулів для реалізації менеджменту та обліку послуг авто сервісу та постановка задач дослідження; тестування застосунку; висновки; список

використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
	Катєльніков Д.І., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
	Аналіз розвитку технологій розробки програмних застосунків для менеджменту та обліку послуг автосервісу та постановка задач дослідження	25.03.25 - 08.04.25	вик
	Розробка архітектури та алгоритмів роботи програмного застосунку	09.04.25 - 20.04.25	вик
	Варіантний аналіз та вибір мови програмування розробки	21.04.25 - 23.04.25	вик
	Розробка програмних модулів для реалізації менеджменту програмного застосунку та цілісного застосунку	24.04.25 - 20.05.25	вик
	Тестування застосунку	21.05.25 - 25.05.25	вик
	Оформлення матеріалів до захисту БКР	26.05.25 - 30.05.25	вик

Студент _____ Сопотніцький О.Є.
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ Катєльніков Д.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 99 сторінок формату А4, на яких представлено 25 рисунків, 2 таблиці, 4 додатки. Список використаних джерел містить 21 найменування.

У бакалаврській кваліфікаційній роботі було розроблено методи та програмний застосунок для менеджменту та обліку послуг автосервісу з використанням технології JavaFX і бази даних MySQL. Проведено аналіз стану існуючих засобів розробки програмних застосунків для менеджменту та обліку послуг автосервісу їх порівняння та обґрунтовано доцільність створення власного застосунку з інтерактивним інтерфейсом користувача, модулями відстеження статусу обслуговування авто, збереження історії обслуговування транспортного засобу, відповідним модулем фінансової звітності та бази даних.

Розроблено структуру програмного забезпечення, що дозволяє модульно реалізовувати функціонал обліку та управління процесами технічного обслуговування автотранспорту. Система охоплює ключові підсистеми, включаючи реєстрацію замовлень, розподіл завдань між працівниками, фінансовий облік та ведення історії обслуговування. Реалізовано графічний інтерфейс на базі JavaFX з урахуванням ролей користувачів (диспетчер, механік, бухгалтер, клієнт), а також механізм збереження й обробки інформації у реляційній базі даних MySQL.

Забезпечено можливість планування робіт, контролю статусу замовлень у режимі реального часу та автоматичного формування звітної документації. Створена система дозволяє суттєво підвищити ефективність внутрішніх процесів автосервісного підприємства, знизити ймовірність помилок, скоротити час обробки заявок та підвищити якість обслуговування клієнтів. Результати тестування підтвердили стабільність роботи системи, коректність обробки даних та відповідність функціональним вимогам.

Ключові слова: Java, JavaFX, MySQL, автосервіс, автоматизація, облік, диспетчеризація, програмна інженерія.

ABSTRACT

The bachelor’s qualification work consists of 99 A4 pages, including 25 figures, 2 tables, and 4 appendices. The list of references contains 22 sources.

This work presents the development of methods and a software application for the management and accounting of car service operations using JavaFX technology and a MySQL database. An analysis of the current state of existing software tools for managing and tracking automotive services was carried out, and their comparison justified the necessity of creating a custom application featuring an interactive user interface, modules for monitoring service status, storing vehicle maintenance history, generating financial reports, and managing database records.

The software system was designed with a modular architecture to support the implementation of key functionalities for managing and accounting car maintenance processes. The system includes core subsystems such as order registration, task distribution among employees, financial accounting, and maintenance history tracking. A graphical user interface was developed using JavaFX, with role-based access for dispatchers, mechanics, accountants, and clients, while data storage and processing are handled using a relational MySQL database.

The system enables real-time job scheduling, order status tracking, and automated report generation. The implemented solution significantly enhances the efficiency of internal operations at car service enterprises, reduces the risk of errors, accelerates order processing, and improves overall service quality. Testing results confirmed the system's stability, correctness of data handling, and compliance with functional requirements.

Keywords: Java, JavaFX, MySQL, car service, automation, accounting, dispatching, software engineering.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному конкурентному середовищі сфери послуг, особливо в галузі технічного обслуговування та ремонту автотранспорту, зростає потреба в підвищенні ефективності управління робочими процесами. Автосервісні підприємства щоденно обробляють значний обсяг замовлень, координують дії різних учасників процесу (диспетчерів, механіків, бухгалтерів) та взаємодіють із клієнтами. У таких умовах застосування традиційних засобів обліку, як-от ручне ведення документації чи використання електронних таблиць, стає недостатньо ефективним і призводить до зниження продуктивності, затримок у виконанні робіт і незадоволеності клієнтів, що відповідно призводить до втрати доходу та бізнесу.

Основні задачі, з якими стикаються автосервіси, включають: нераціональне планування при розподілі завдань, відсутність прозорості в обслуговуванні клієнтів, слабку інтеграцію між внутрішніми підсистемами (наприклад, обліку, диспетчеризації, фінансів) та неефективне управління історією обслуговування автомобілів. Усе це знижує якість надання послуг і перешкоджає сталому розвитку підприємств.

Для подолання зазначених викликів актуальним є створення комплексної автоматизованої системи обліку та управління, яка дозволить централізувати дані, автоматизувати обробку замовлень, забезпечити своєчасне планування та моніторинг виконання робіт, а також покращити комунікацію з клієнтами. Таке рішення має поєднувати інструменти програмної інженерії, зокрема використання JavaFX для створення інтуїтивно зрозумілого інтерфейсу користувача, та реляційної бази даних MySQL для надійного зберігання і доступу до інформації [1].

У такий спосіб, розробка програмного забезпечення на основі виявлених бізнес-потреб автосервісу спрямована на підвищення загальної ефективності підприємства. Тому, актуальність теми зумовлена необхідністю оптимізації внутрішніх процесів, зменшення витрат часу на рутинні операції, зниження рівня

помилок та забезпечення високого рівня сервісу для клієнтів. Це дослідження базується на поєднанні теоретичних підходів до проектування інформаційних систем і практичних методів розробки програмних продуктів для автоматизації роботи підприємств сфери обслуговування.

Мета та завдання дослідження. Метою роботи є підвищення продуктивності роботи підприємства за рахунок розробки програмної системи менеджменту та обліку послуг автосервісу, що забезпечить автоматизацію обліку, диспетчеризації, розподілу завдань та інтеграцію інформаційних потоків між усіма учасниками процесу. Розроблена система дозволить зменшити кількість помилок, пришвидшити обробку замовлень та покращить якість обслуговування клієнтів.

Відповідно до поставленої мети, в дослідженні необхідно вирішити такі завдання:

- проаналізувати основні завдання та потреби підприємств автосервісу у сфері автоматизації обліку та управління послугами;

- обґрунтувати вибір методів та технологій для створення автоматизованої системи (мови програмування, платформи, СКБД);

- розробити архітектуру та функціональну модель інформаційної системи обліку та управління послугами автосервісу;

- реалізувати прототип програмного забезпечення на базі JavaFX та MySQL, що забезпечує взаємодію між диспетчером, механіками, бухгалтером та клієнтом;

- створити інтерфейси для кожної ролі користувача з урахуванням їхніх потреб;

- забезпечити збереження, обробку та пошук інформації про замовлення, фінансові операції, історію обслуговування;

- протестувати програмні модулі на прикладах реальних сценаріїв та оцінити ефективність впровадженого рішення.

Об’єктом дослідження є процес автоматизації обліку та управління послугами в автосервісних підприємствах.

Предметом дослідження є методи та програмні засоби програмної системи менеджменту та обліку послуг автосервісу.

Методи дослідження

У процесі дослідження використовувалися такі методи:

- методи системного аналізу для вивчення структури процесів автосервісу, формалізації взаємозв'язків між його елементами та визначення функціональних вимог до автоматизованої системи;

- методи програмної інженерії, зокрема моделювання архітектури програмного забезпечення, проєктування бази даних, побудова користувацьких інтерфейсів і логіки взаємодії компонентів системи;

- методи комп'ютерного моделювання для візуалізації та тестування бізнес-процесів у вигляді прототипу системи, що дозволило оцінити поведінку програми в різних робочих ситуаціях;

- методи статистичної обробки даних для виявлення закономірностей у роботі автосервісу, таких як середній час обслуговування, завантаженість персоналу, частота повторних візитів клієнтів тощо;

- методи оптимізації для раціонального розподілу завдань між працівниками з урахуванням їхньої зайнятості та складності робіт. Застосовувались елементи лінійного програмування та евристичних алгоритмів при моделюванні системи планування.

Новизна одержаних результатів:

1. Подальшого розвитку отримав метод автоматизації управління ресурсами автосервісу, який, на відміну від існуючих, використовує інтеграцію планування, обліку та комунікації в єдину інформаційну систему з урахуванням ролей користувачів (диспетчер, механік, бухгалтер, клієнт). Це дозволяє не лише централізовано керувати поточними завданнями, а й ефективно розподіляти навантаження між працівниками та оптимізувати використання ресурсів.

2. Подальшого розвитку отримав метод побудови автоматизованої системи обміну інформацією між підсистемами автосервісу (облік, диспетчеризація, фінанси), який, на відміну від існуючих рішень, передбачає використання

модульної структури програмної системи на основі технології JavaFX і бази даних MySQL, яка забезпечує високий рівень масштабованості та адаптивності, дозволяючи гнучко розширювати її функціональність відповідно до змін бізнес-процесів підприємства. Також це дозволяє мінімізувати дублювання даних, скорочує час обробки замовлень та знижує ризик помилок, пов’язаних із людським фактором, забезпечуючи ефективність і узгодженість усіх компонентів системи.

Практична цінність отриманих результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби менеджменту та обліку послуг автосервісу.

Особистий внесок здобувача

У бакалаврській кваліфікаційній роботі всі результати дослідження отримані автором самостійно.

У друкованій праці, опублікованій у співавторстві[1], автору належать такі результати: розробка методу та програмних засобів для підвищення ефективності управління ресурсами в сервісних підприємствах, зокрема в контексті диспетчеризації та оптимізації розподілу завдань.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні положення БКР доповідалися та обговорювалися на LIV Всеукраїнської науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії (24-27 березня 2025р., м. Вінниця).

Публікації. Основні результати досліджень опубліковано в матеріалах LIV Всеукраїнської науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії (24-27 березня 2025р., м. Вінниця) [1].

Структура та обсяг роботи. Бакалаврська кваліфікаційна робота складається зі вступу, _4_ розділів, висновків, списку використаних джерел, що містить _21_ найменування, _4_ додатка. Робота містить _46_ ілюстрацій, _2_ таблиці.

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ РОЗРОБКИ ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ МЕНЕДЖМЕНТУ ТА ОБЛІКУ ПОСЛУГ АВТО СЕРВІСУ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану технологій розробки програмних модулів системи менеджменту та обліку послуг автосервісу

Сьогодні автосервісні підприємства змушені впроваджувати ефективні методи управління бізнес-процесами за для ефективнішого управління бізнес-процесами, що включають запис клієнтів, управління запасами, облік виконаних робіт, фінансовий контроль та взаємодію з постачальниками [2]. За останні декілька років розвиток інформаційних технологій сприяв появі нових програмних рішень, що автоматизують дані процеси й відповідно спрощують роботу менеджерів та диспетчерів [3]. Дослідження стану технологій розробки таких систем дозволяє оцінити актуальні тенденції , а також окреслити шляхи їхнього вдосконалення та модернізації. Це сприяє підвищенню ефективності та адаптації модулів до нових вимог сьогоднішнього ринку послуг [4].

Ринок програмних продуктів для автосервісів доволі насичений різного роду програмних застосунків, й на сьогоднішній день уже існує декілька підходів для їх розробки, відповідно орієнтованих на сфери використання й вимоги до них. Деякі підприємства використовують локальні програмні застосунки, які встановлюються на окремі комп’ютери й працюють автономно, без необхідності підключення до інтернету [5]. Такі системи є більш захищеними, з точки зору безпеки даних, проте вони не дозволяють зручно синхронізувати інформацію між різними філіями або не надають можливості віддалено керувати процесами автосервісу, через відокремлення даних програмних застосунків від мережі інтернет.

З іншого боку, набувають популярності хмарні рішення, які дозволяють вести облік клієнтів, управляти запасами та контролювати виконання робіт через веб-інтерфейс, веб-застосунок або мобільний додаток. Такі системи забезпечують швидкий доступ до інформації з будь-якого пристрою, на якому

вони встановлені, та дають змогу власникам бізнесу оперативно реагувати на зміни в роботі власного підприємства та отримувати найбільш актуальну інформацію щодо нього [6]. Водночас, їхня ефективність значною мірою залежить від стабільності інтернет-з'єднання та рівня кібербезпеки, що роблять даний тип програмних застосунків не досить надійними в плані захищеності використання.

Попри значні переваги для автоматизації автосервісних процесів, існують і певні проблеми, які стримують впровадження даних програмних рішень. Однією із головних є висока вартість впровадження даних комплексних систем, особливо для малих підприємств, адже купівля ліцензійного програмного забезпечення, навчання персоналу та адаптація бізнес-процесів вимагатимуть значних фінансових та часових ресурсів, що є досить-таки нераціональним в масштабі даного малого підприємства.

Ще однією проблемою є необхідність підготовки співробітників до роботи з новими технологіями та програмним застосунком. Якщо персонал не має достатньо навичок роботи з даними програмними модулями, це може призвести до помилок у веденні обліку, що, у свою чергу, негативно вплине на ефективність роботи автосервісу й, як результат, збільшенню фінансових витрат.

Не менш важливим аспектом є захист персональних даних клієнтів, даних автосервісу та фінансової інформації. Програмні системи, особливо хмарні, повинні мати надійні механізми кібербезпеки, включаючи шифрування даних, багаторівневу автентифікацію та регулярні оновлення програмного забезпечення для захисту від новітніх різновидів загроз. Ще одним викликом є інтеграція нових систем із уже існуючим програмним забезпеченням підприємства [7]. У випадку, якщо автосервіс вже використовує бухгалтерські або CRM-системи, нове програмне забезпечення має бути сумісним із цими інструментами для забезпечення їх безперебійної роботи.

Ринок програмних рішень для автосервісів активно розвивається, і перспективним напрямком є розробка гнучких модульних систем з можливістю їх майбутньої кастомізації. Такі системи дозволять автосервісам підлаштовувати

програмне забезпечення під свої потреби, мінімізуючи витрати на розробку. Інтеграція з сучасними технологіями відкриває нові можливості для оптимізації роботи та покращення обслуговування клієнтів.

1.2 Порівняльний аналіз аналогів

У сучасних умовах цифровізації автосервісів існує широкий вибір програмного забезпечення для управління ремонтними процесами, обліку фінансів та комунікації з клієнтами. Найпоширенішими категоріями таких рішень є хмарні CRM-системи, веб-додатки та десктопні програми з локальною базою даних [8]. Кожен із перелічених варіантів має свої переваги та недоліки, що впливають на ефективність їх використання в автосервісах.

До найбільш відомих рішень відносять:

– АвтоМайстер CRM;

– CarServicePro;

– AutoSoft Web;

– Workshop ERP;

– Мотор Менеджер.

АвтоМайстер CRM – це хмарна система для управління автосервісами, що надає інструменти для ведення клієнтської бази, контролю робочих процесів та фінансів (див.рисунок 1.1). Головна перевага – мобільний додаток для клієнтів, який дозволяє відстежувати статус ремонту [9].

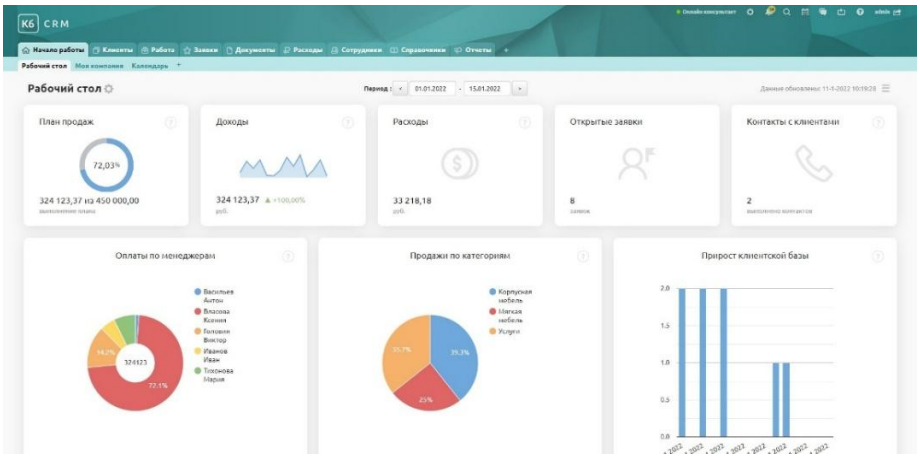


Рисунок 1.1 – Інтерфейс застосунку АвтоМайстер CRM

Інший приклад – CarServicePro, інтерфейс якого зображений на рисунку 1.2. Даний веб-додаток для автосервісів, орієнтований на автоматизацію запису клієнтів, управління персоналом та фінансовий облік. Відрізняється наявністю особистого кабінету для клієнтів [10].

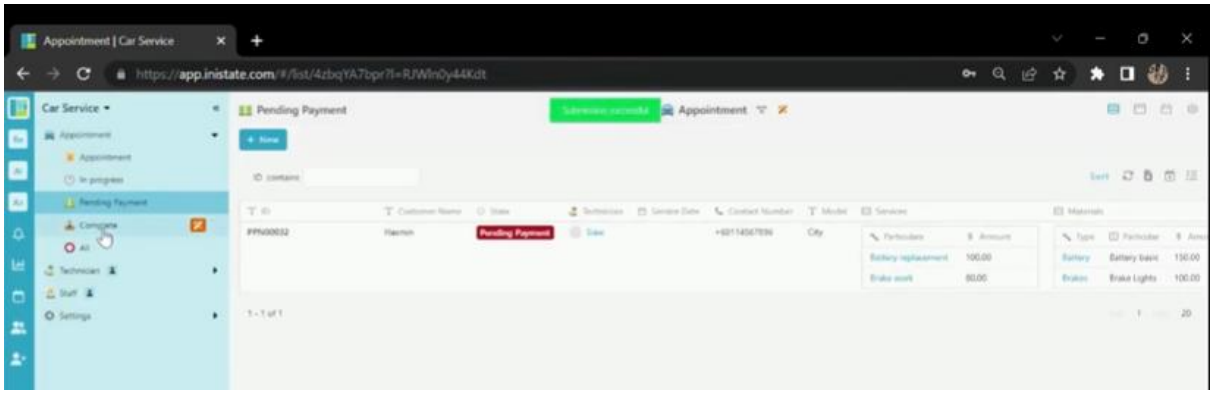


Рисунок 1.2 – Інтерфейс застосунку CarServicePro

AutoSoft Web – це веб-платформа для управління автосервісом, що спеціалізується на обліку виконаних робіт, складуванні та фінансовому аналізі транзакції автосервісу [11]. Відповідний інтерфейс зображено на рисунку 1.3.

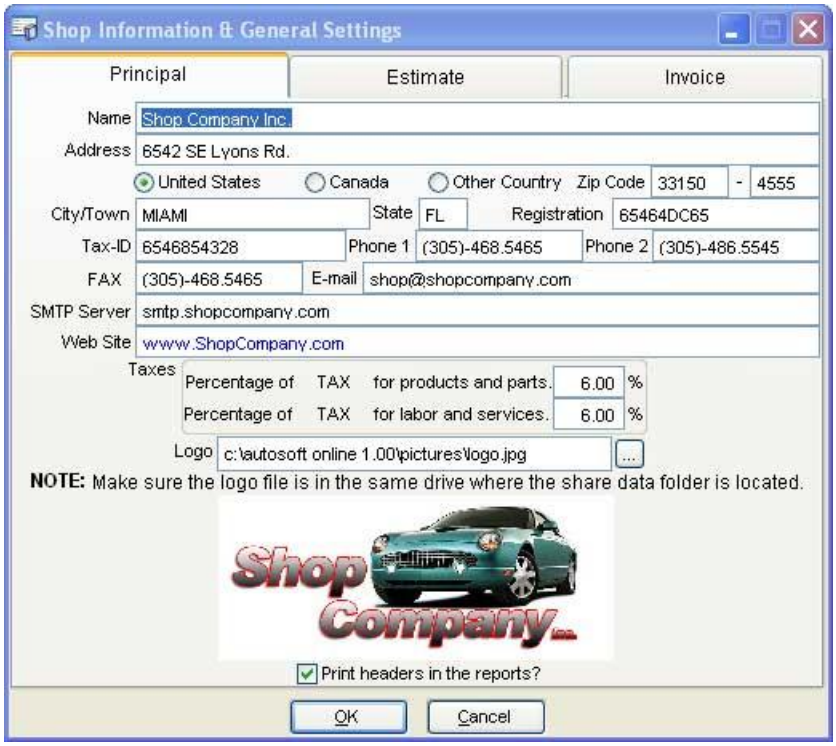


Рисунок 1.3 – Інтерфейс застосунку AutoSoft Web

Workshop ERP – це комплексна система управління автосервісом, яка дозволяє вести облік робіт, керувати складом, взаємодіяти з клієнтами та контролювати фінансові потоки [12]. Відрізняється модульною структурою, що дозволяє підключати додаткові функції за потребою, див. рисунок 1.4.

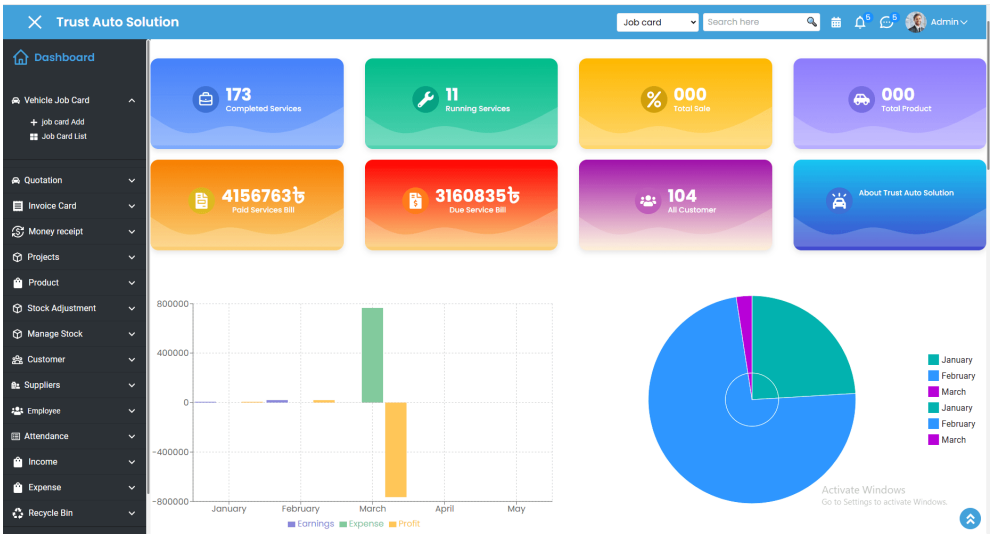


Рисунок 1.4 - Інтерфейс застосунку Workshop ERP

Мотор Менеджер – це спеціалізоване програмне забезпечення для невеликих і середніх автосервісів, яке допомагає автоматизувати робочі процеси та вести облік фінансових операцій [13]. Відрізняється простим інтерфейсом, який зображений на рисунку 1.5, та можливістю швидкого впровадження інформації.

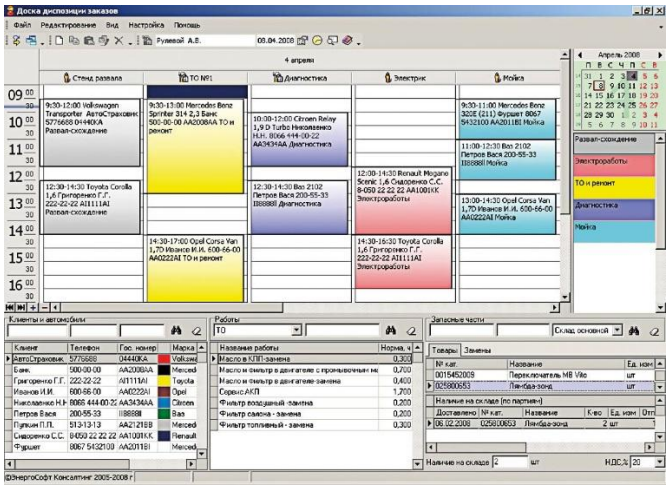


Рисунок 1.4 - Інтерфейс застосунку Мотор Менеджер

Також було проаналізовано функціонал даних додатків на наявність наступних функцій:

– облік клієнтів, замовлень і виконаних робіт;

– автоматичне створення документів (накладні, рахунки, акти виконаних робіт);

– базові інструменти для управління фінансами (облік виплат і доходів);

– проста система для збереження історії ремонту авто;

– повноцінний облік робіт, замовлень і запасних частин;

– наявність інструкцій щодо проведення ремонтних робіт для механіків.

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 - Порівняльна характеристика аналогів

Критерій	АвтоМайстер CRM	CarService Pro	AutoSoft Web	Workshop ERP	Мотор Менеджер	Власна розробка
Відстеження статусу обслуговування авто клієнтом	+	+	-	+	-	+
Прозора історія виконаних робіт	+	+	-	-	+	+
Інструкції для механіків	-	-	+	-	-	+
Фінансова звітність	-	+	-	+	-	+
Загальна оцінка	50%	75%	25%	50%	25%	100%

Відповідно до порівняльної таблиці, розробка власного програмного забезпечення для менеджменту та обліку послуг автосервісу є доцільною. Таке рішення дозволить усунути недоліки існуючих систем, забезпечивши більшу гнучкість, контроль і персоналізацію під конкретні потреби автосервісу.

Отже, завдяки розробці власної бази даних і можливості інтеграції з іншими системами, розроблене програмного забезпечення (ПЗ) покращить взаємодію з клієнтами та дозволить ефективно відслідковувати всі етапи обслуговування автомобіля, забезпечуючи прозорість, точність фінансових розрахунків та зручність у роботі для механіків.

1.3 Аналіз методів розв'язання задачі

Для створення системи менеджменту та обліку послуг автосервісу важливо вибрати відповідні технології для розробки інтерфейсу користувача та управління даними.

За для створення інтерфейсу можна вибрати між веб-технологіями (HTML, CSS, JavaScript) і десктопними технологіями, такими як JavaFX. Хоча веб-рішення дозволяють доступ через браузер, вони можуть мати обмежену продуктивність при роботі з великими обсягами даних. В свою чергу, JavaFX є потужною платформою для створення багатофункціональних, швидких та інтерактивних десктопних застосунків, що ідеально підходить для обробки великих обсягів даних та забезпечення зручного інтерфейсу [1].

Для зберігання даних можна обрати між SQLite та реляційними базами даних, такими як MySQL. SQLite підходить для малих систем, але може виникнути проблема з масштабуванням при великих обсягах даних. MySQL дозволяє ефективно працювати з великими обсягами, підтримує транзакції та збереження цілісності даних, що робить його ідеальним варіантом для автосервісу.

JavaFX дозволяє створювати інтерактивні інтерфейси, а інтеграція з MySQL через ORM фреймворки, наприклад, Hibernate, забезпечує зручну роботу з даними. Така комбінація дозволяє створювати масштабовані рішення з високою продуктивністю.

Отже, розробка системи менеджменту для автосервісу за допомогою JavaFX та MySQL є доцільною. JavaFX дозволяє створювати потужні графічні інтерфейси, а MySQL забезпечує надійне та масштабоване зберігання даних.

Такий підхід гарантує високу продуктивність, зручність у роботі та можливість подальшого розширення системи.

1.4 Постановка задач для розробки методу і засобів для менеджменту та обліку послуг автосервісу

Проаналізувавши переваги та недоліки існуючих програмних засобів для автосервісу, було визначено наступні завдання, які необхідно виконати для успішної розробки програмного застосунку для менеджменту та обліку послуг автосервісу з використанням технології JavaFX і бази даних MySQL, що дозволить ефективно обслуговувати клієнтів, автоматизувати процеси обліку, зберігання та обробки даних, а також покращити взаємодію між клієнтами, механіками та адміністрацією сервісу. До них було віднесено:

– Розробка алгоритму відстеження статусу обслуговування авто, що дозволить клієнтам відслідковувати статус обслуговування їхнього автомобіля, отримуючи інформацію про стан ремонту, проведені роботи та час проведення робіт.

– Розробити модуль для збереження детальної історії виконаних робіт на кожному автомобілі, з можливістю перегляду інформації про проведені ремонтні роботи, замінені деталі та інші роботи, що проводились на автомобілі.

– Створення графічного інтерфейсу користувача, який буде зручний і інтуїтивно зрозумілий у використанні. Також даний інтерфейс спрощуватиме роботу адміністрації, механіків та клієнтів, забезпечуючи швидкий доступ до потрібної інформації та функцій системи.

– Реалізувати інтеграцію програмного забезпечення з надійною базою даних для збереження інформації про робітників автосервісу, клієнтів та їх автомобілі, проведені роботи, фінансові операції тощо. Використання реляційної бази даних, такої як MySQL, забезпечить стабільну роботу з великими обсягами даних.

– Автоматизація процесу обліку послуг, яка дозволить спрощувати процес обліку послуг автосервісу, включаючи автоматичне генерування рахунків та виставлення інвойсів.

– Тестування програмного забезпечення, а саме: проведення всебічного тестування програмного забезпечення, включаючи перевірку коректності роботи всіх функцій, стабільність системи під навантаженням та зручність інтерфейсу.

Також дане програмне забезпечення повинно бути здатне адаптуватися до збільшення обсягів даних та кількості клієнтів.

1.5 Висновки

У даному розділі було розглянуто стан програмних засобів та технологій розробки програмних модулів системи менеджменту та обліку послуг автосервісу. Були розглянуті програмні аналоги, а саме: АвтоМайстер CRM, CarServicePro, AutoSoft Web, Workshop ERP та Мотор Менеджер. Також було проаналізовано їхні переваги та недоліки й відповідно складено порівняльну таблицю.

Відповідно до проаналізованих аналогічних додатків й їхніх переваг та недоліків, було сформульовано завдання для подальшої розробки власного програмного застосунку для менеджменту та обліку послуг автосервісу. Ці завдання включають в розробку алгоритму відстеження статусу обслуговування авто, розробку модуля для збереження історії обслуговування транспортного засобу й відповідного модуля фінансової звітності. Також необхідним є інтегрування бази даних з відповідною реалізацією графічного інтерфейсу користувача в додатку.

Отже, на підставі проведеного аналізу було підтверджено доцільність створення власного програмного застосунку. У результаті отриманих даних сформульовано список завдань, які потрібні бути втілені для виконання розробки власного програмних застосунку.

2 РОЗРОБКА ІНТЕРФЕЙСУ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних даних системи

Для створення програмного забезпечення, що автоматизує менеджмент та облік послуг автосервісу, будуть застосовані технології JavaFX і MySQL. Перш за все, JavaFX є потужним інструментом для розробки інтерактивних графічних інтерфейсів, що гарантує зручність у користуванні та гнучкість налаштувань [14]. У свою чергу, MySQL забезпечує ефективне збереження та обробку великих обсягів даних, гарантуючи швидкий доступ та надійність інформаційних процесів [15].

Процес розробки застосунку передбачає кілька ключових етапів, а саме: на першому етапі буде створений інтуїтивно зрозумілий користувацький інтерфейс, що дозволить диспетчерам автосервісу легко управляти записами клієнтів, переглядати історію замовлень, формувати розклад роботи персоналу та відстежувати фінансові операції.

Наступним етапом є розробка модуля управління замовленнями, що інтегрується з базою даних MySQL. Цей компонент відповідатиме за облік наданих послуг, деталей та витратних матеріалів, а також автоматизуватиме процеси створення, редагування й відстеження статусу виконання ремонтних робіт [16].

Окрему увагу буде приділено фінансовому модулю, який відповідатиме за ведення бухгалтерського обліку, контроль доходів та витрат підприємства. Він дозволить автоматично створювати фінансові документи, такі як накладні та рахунки, а також забезпечить доступ до звітності щодо фінансового стану автосервісу.

Для аналізу ефективності роботи сервісу буде реалізований аналітичний модуль, що використовуватиме дані з MySQL для створення статистичних звітів. Він дозволить отримувати актуальні відомості про завантаженість майстрів, популярність певних послуг та загальний фінансовий стан підприємства.

Розробка та впровадження цієї системи значно оптимізує бізнес-процеси автосервісу, мінімізує ризик помилок, автоматизує ключові операції та покращить взаємодію з клієнтами. Використання JavaFX у поєднанні з MySQL забезпечить високу продуктивність, адаптивність та можливість масштабування системи в майбутньому.

2.2 Розробка інтерфейсу програмного додатку

При розробці інтерфейсу програмного забезпечення було приділено особливу увагу його зрозумілості та зручності для користувача.

Основним кольором інтерфейсу обрано синій, оскільки даний колір асоціюється з автосервісом. Допоміжними кольорами обрано білий, сірий та чорний, оскільки вони контрастують один одному [17].

При відкритті програмного додатку, користувача зустрічає екран завантаження із назвою системи й вибором профілю користувача (рис. 2.1).

Рисунок 2.1 – Головний екран додатку

Інтерфейс оформлено в сучасному стилі з логотипом компанії, назвою системи – «ALEXPLUS», а також зручним розташуванням елементів вибору ролі користувача.

У центральній частині вікна розташовано логотип сервісу, який займає майже весь простір і забезпечує впізнаваність бренду. Над логотипом знаходиться назва системи, стилізована білим шрифтом «System Bold», що вказує на корпоративний стиль та призначення додатку.

Нижче логотипа розташовано три кнопки вибору ролі користувача:

- Client – відкриває інтерфейс клієнта, який дозволяє переглядати та створювати заявки. Також це вікно дозволяє переглядати перелік власних автомобілів та їх історію обслуговування.
- Mechanic – відкриває вікно механіка для перегляду запланованих робіт й відповідних іструкцій щодо їх виконання.
- Dispatch – вкладка диспетчера, яка реалізує перелік заявок клієнтів на ремонт та відповідний інтерфейс для їх розподілення.
- Accountant – використовується для реалізації інтерфейсу бухгалтера.

У верхній частині інтерфейсу зліва розміщена кнопка Info, яка відкриває інформаційне вікно з короткою довідкою про систему, а саме: основні функції застосунку, його мета та використані технології для розробки.

У правому верхньому куті – кнопка Dispatch, що відкриває вікно диспетчера для розподілу заявок.

У нижній частині вікна міститься підпис розробника «Developed by Oleksii Sopotnitskyi, student VNTU 1PI-21B», що вказує на авторство навчального проєкту.

Загалом, вікно є центральним навігаційним хабом програми, з якого користувачі можуть обрати свій сценарій роботи відповідно до ролі в сервісному процесі.

Для забезпечення безпечного доступу до персональних даних клієнта було реалізовано окреме вікно авторизації, призначене спеціально для користувачів, які мають роль клієнта, див. рисунок 2.2.

На цьому етапі користувач бачить привітальний напис «Dear Client», а також інструкцію: «Kindly enter your name and password to open your file», що закликає ввести свої облікові дані. Центрована структура елементів інтерфейсу сприяє зручності взаємодії та інтуїтивному розумінню.

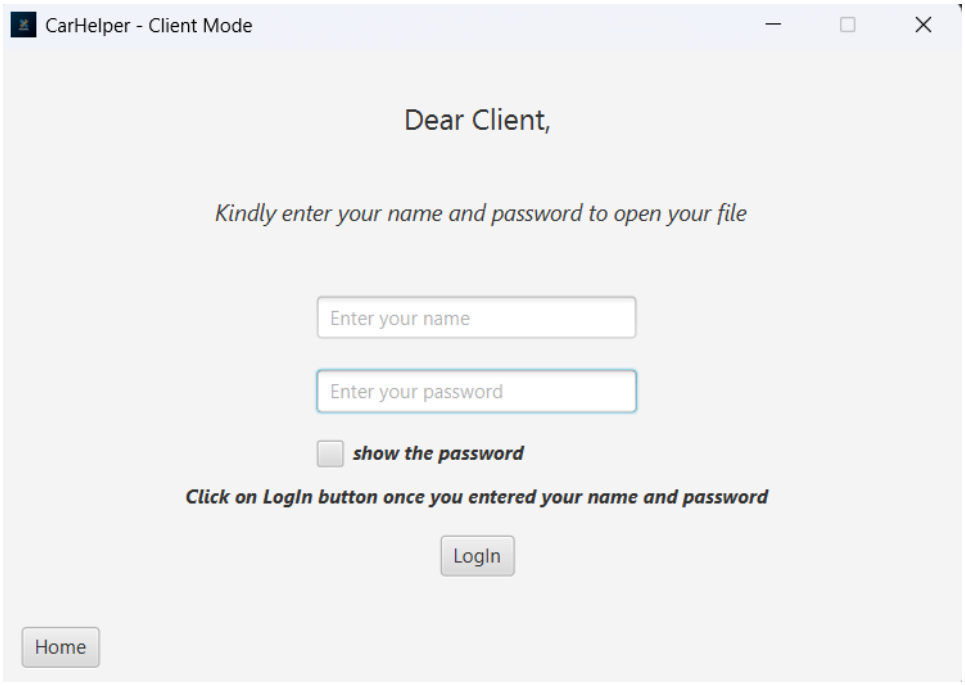


Рисунок 2.2 – Екран логування клієнта

Основні компоненти інтерфейсу:

• Поле введення імені («Enter your name») – текстове поле TextField, у якому клієнт зазначає своє ім’я.

• Поле введення пароля – реалізоване подвійно, як приховане (PasswordField) і як видиме (TextField). Це зроблено для реалізації функціоналу відображення пароля за допомогою прапорця «show the password».

• Прапорець «show the password» – дозволяє користувачу перемикати відображення пароля в текстовому полі. Цей елемент викликає метод, який переключає видимість між прихованим і відкритим варіантом пароля.

У нижній частині вікна розміщено:

• Кнопку «LogIn» – надсилає дані для входу.

• Кнопку «Home» – повертає користувача до головного меню.

• Мітку-підказку – «Click on LogIn button once you entered your name and password», яка виконує роль інструкції та може бути оновлена для виводу повідомлень про помилки авторизації або інші стани.

Інтерфейс логіну клієнта витриманий у мінімалістичному стилі, що відповідає принципам доступності [18–20]: відсутність зайвих візуальних елементів, зрозумілі підказки, чітка логіка дій. Такий підхід дозволяє забезпечити ефективний та безпечний доступ до особистого кабінету клієнта сервісного центру.

Також було створено меню програмного застосунку (рис. 2.3), для полегшення взаємодії із розробленим функціоналом.

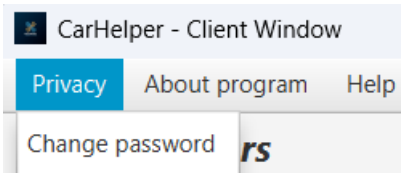


Рисунок 2.3 – Меню клієнта

На рисунку 2.4 відображено початковий стан робочого середовища корисувача (клієнта), що допомагає покращити користувацьку імерсивність та полегшити взаємодію із застосунком. Інтерфейс був розроблений з урахуванням принципів зручності, функціональності та естетики, аби забезпечити безперешкодну взаємодію користувача з програмою, навіть для тих, хто не має досвіду роботи з подібними системами.

Головна частина екрана відведена для відображення списку автомобілів користувача, що розміщено зліва на екрані. Це важлива секція, адже клієнт може побачити всі свої зареєстровані транспортні засоби. Система дозволяє швидко обирати необхідний автомобіль, що значно полегшує процес подальшої роботи.

У правій частині екрану, поруч зі списком автомобілів, знаходиться інша важлива частина інтерфейсу – історія обслуговування клієнта. Тут показуються всі попередні замовлення, що дозволяє користувачеві не тільки переглядати, але й повторно використовувати інформацію про виконані послуги.

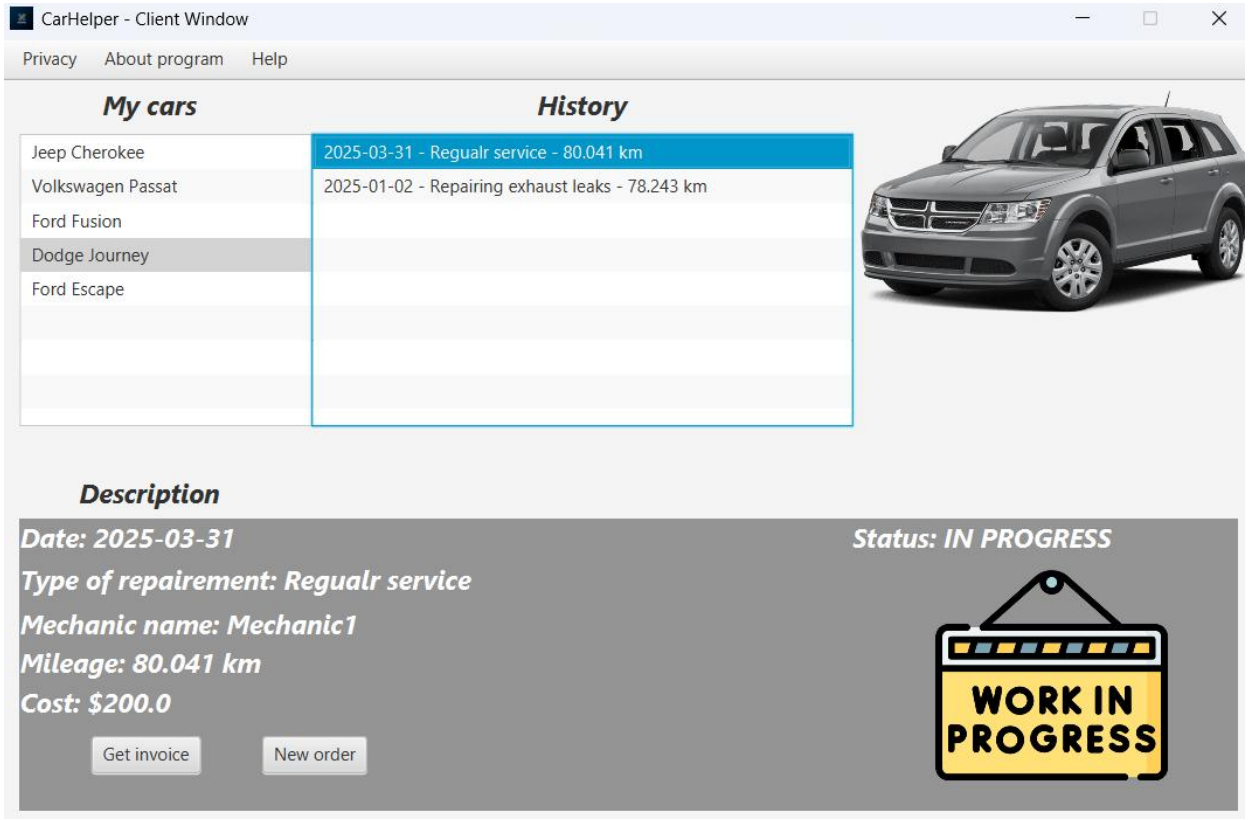


Рисунок 2.4 – Екран робочого середовища клієнта

Нижня частина інтерфейсу містить блок із детальною інформацією про вибране замовлення, зокрема:

- дата замовлення;
- тип ремонту;
- ім’я механіка;
- пробіг автомобіля на момент обслуговування (виконаних робіт);
- вартість робіт;
- статус замовлення.

Ці дані відображаються у вигляді міток з білим текстом на сірому фоні, що покращує читабельність.

Інтерфейс також включає меню для:

- зміни пароля;
- отримання інформації про програму;
- доступу до довідки.

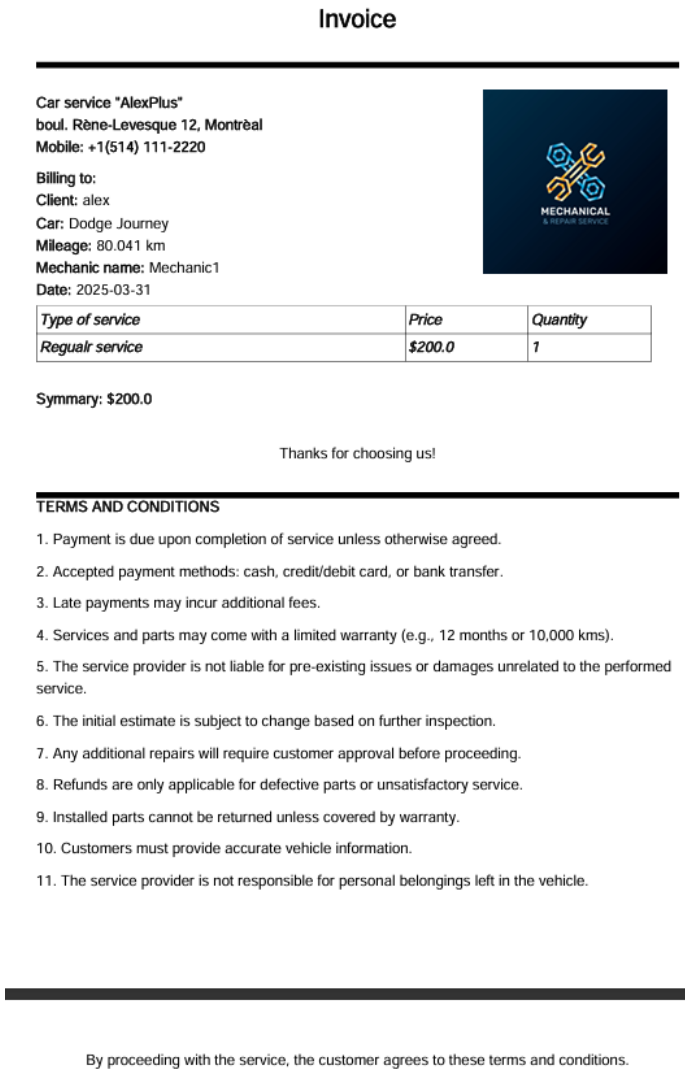
Кнопки в нижній частині екрана дозволяють:

- отримати рахунок-фактуру;
- створити нове замовлення.

Логотип у верхній правій частині екрану додає брендової впізнаваності, а фонове зображення додає візуальної привабливості без порушення зручності читання.

Цей інтерфейс поєднує функціональність та естетичний дизайн, забезпечуючи зручне та інтуїтивне використання системи.

Також, при натисканні кнопки про формування рахунок-фактури – створюється автоматично PDF файл, який відповідно автоматично відкривається в перегляді редактору PDF. Зображення фотографії рахунок-фактури зображено на рисунку 2.5.



Invoice

Car service "AlexPlus"
boul. Rène-Levesque 12, Montrèal
Mobile: +1(514) 111-2220

Billing to:
Client: alex
Car: Dodge Journey
Mileage: 80.041 km
Mechanic name: Mechanic1
Date: 2025-03-31

Type of service	Price	Quantity
Regualr service	$200.0	1

Symmary: $200.0

Thanks for choosing us!

TERMS AND CONDITIONS

1. Payment is due upon completion of service unless otherwise agreed.
2. Accepted payment methods: cash, credit/debit card, or bank transfer.
3. Late payments may incur additional fees.
4. Services and parts may come with a limited warranty (e.g., 12 months or 10,000 kms).
5. The service provider is not liable for pre-existing issues or damages unrelated to the performed service.
6. The initial estimate is subject to change based on further inspection.
7. Any additional repairs will require customer approval before proceeding.
8. Refunds are only applicable for defective parts or unsatisfactory service.
9. Installed parts cannot be returned unless covered by warranty.
10. Customers must provide accurate vehicle information.
11. The service provider is not responsible for personal belongings left in the vehicle.

By proceeding with the service, the customer agrees to these terms and conditions.

Рисунок 2.5 – Сформована рахунок-фактура

При натисканні клавіші «Створити нове замовлення», відкривається вікно, в якому клієнт повинен заповнити заявку на виконання ремонтних робіт, а саме: обрати марку та модель автомобіля, тип ремонту та дату, коли він хотів би відвідати автосервіс. Також є можливість переглядати вже створені заявки на ремонт, їх корегувати та видаляти за необхідності. Зображення інтерфейсу вікна створення нового замовлення продемонстровано на рисунку 2.6.

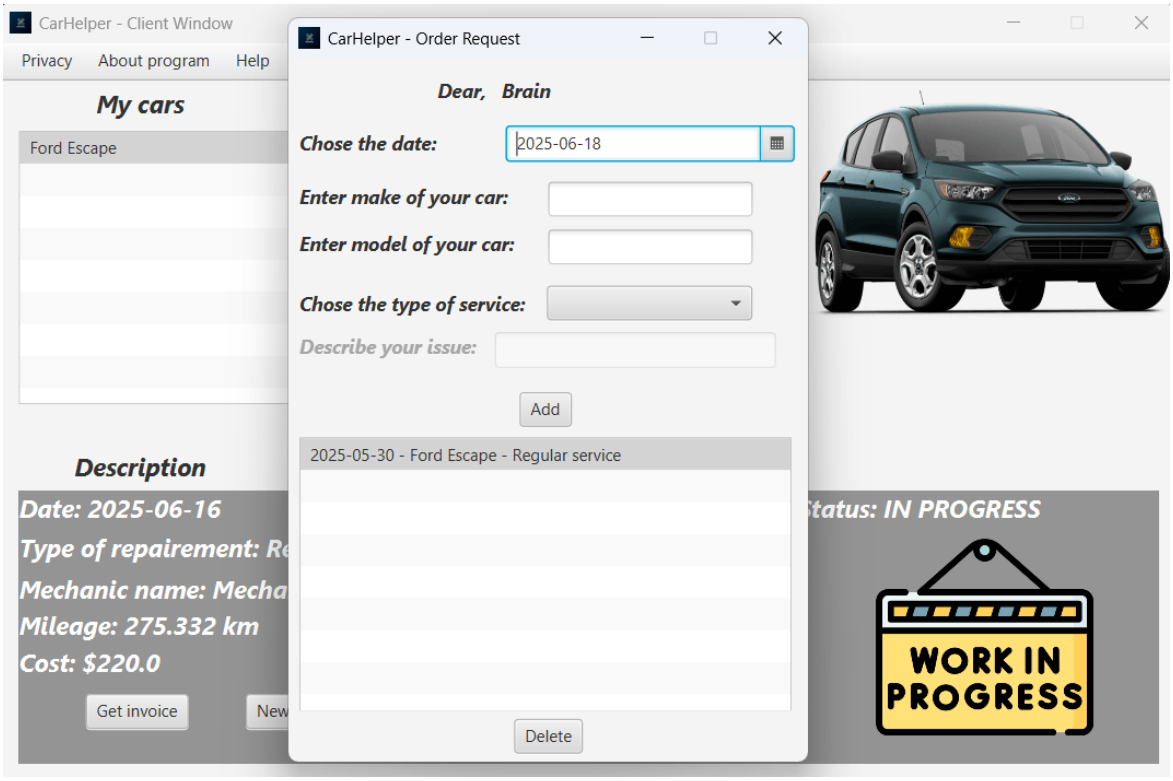


Рисунок 2.6 – Вікно створення замовлення

Також було створено вікно «Про програму», яке відображає короткий опис програмного застосунку, яке розробляється.

Воно містить в собі наступні пункти:

- мету створення програмного застосунку,
- основний його функціонал,
- перелік ролей користувачів,
- використані технології для його розробки.

Відповідний вигляд вікна «Про програму» зображено на рисунку 2.7.

Рисунок 2.7 – Екран інформаційного вікна

Розробка графічного інтерфейсу програмного застосунку була проведена у середовищі розробки IntelliJ IDEA з використанням технології SceaneBuilder та мови програмування JavaFX.

2.3 Розробка моделі системи

Для кращого розуміння роботи модулів програмних засобів системи менеджменту та обліку послуг автосервісу було вирішено розробити модель

роботи системи на прикладі UML діаграми діяльності, зображеної на рисунку 2.7. Згідно із зазначеною діаграмою, користувач для початку взаємодії із програмним застосунком повинен увійти в систему за допомогою аутентифікації. У разі введення некоректних даних користувач отримує повідомлення про помилку і можливість повторного введення.

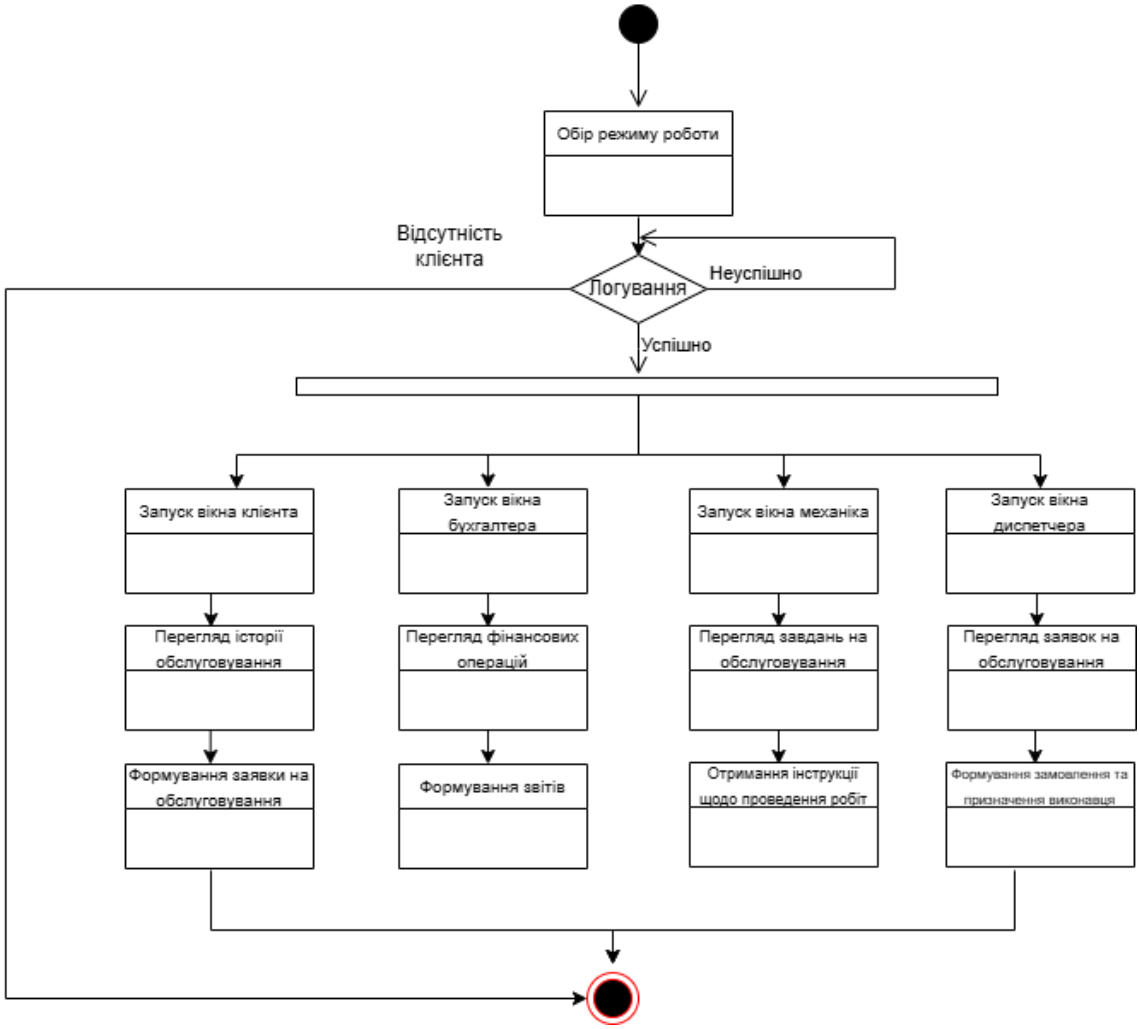


Рисунок 2.7 – Діаграма діяльності програмних засобів

Наступним кроком є вибір режиму роботи: клієнт, механік, диспетчер або бухгалтер. У залежності від вибраного режиму, система надає відповідний набір функцій. Клієнт може переглядати історію обслуговування, записуватись на сервіс, отримувати сповіщення про статус свого замовлення та переглядати історію виконаних робіт. Механік має доступ до інформації про поточні замовлення, перегляду деталей виконуваних робіт та фіксації їхнього виконання.

Бухгалтер може вести фінансовий облік, контролювати надходження та витрати, а також формувати звіти про фінансовий стан автосервісу. Диспетчер має доступ до заявок та розподілення завдань між механіками.

Після завершення обслуговування система автоматично формує звіт про виконані роботи, розраховує загальну вартість послуг і генерує фінальний рахунок для клієнта. Користувач має можливість переглянути деталізацію рахунку й зберегти документ про оплату у форматі .pdf.

Крім того, програмний застосунок містить модуль аналітики, який дозволяє керівництву отримувати статистичні звіти про надані послуги, фінансові показники та завантаженість майстрів. Також передбачена можливість інтеграції з бухгалтерськими системами для автоматизованого обліку витрат і доходів автосервісу.

Таким чином, розроблена система забезпечує ефективне управління послугами автосервісу, спрощує контроль за роботою персоналу та покращує якість обслуговування клієнтів.

2.4 Розробка алгоритмів роботи системи

Для розробки програмного застосунку, для розробки програмних модулів системи менеджменту та обліку послуг автосервісу необхідно розробити загальний алгоритм, згідно якого буде працювати програмний модуль, відповідний алгоритм зображено на рисунку 2.8.

Згідно алгоритму, спочатку відбувається завантаження робочого середовища, а саме – головного вікна програми в якому відповідно потрібно обрати тип користувача з наявних чотирьох: бухгалтер, клієнт, диспетчер та механік. Після цього потрібно здійснити вхід в програму за допомогою логіна й пароля аби отримати доступ до відповідного профіля користувача.

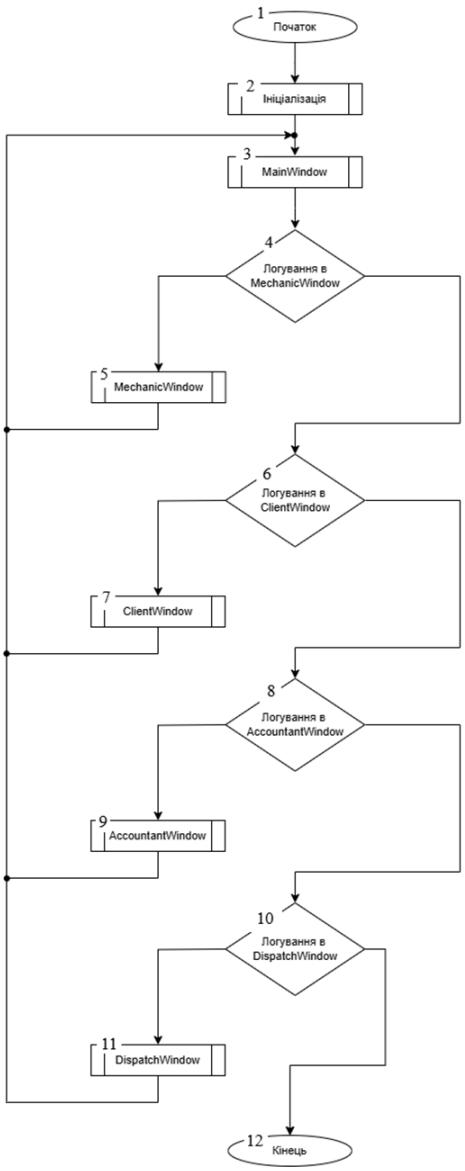


Рисунок 2.8 – Алгоритм роботи програмного застосунку

Якщо користувачем є клієнт – він має змогу побачити власні авто та їх історію обслуговувань, також є можливість запису на сервіс. Також можна отримати інформацію про статус замовлення та відповідний кошторис, відповідний алгоритм зображено на рисунку 2.9.

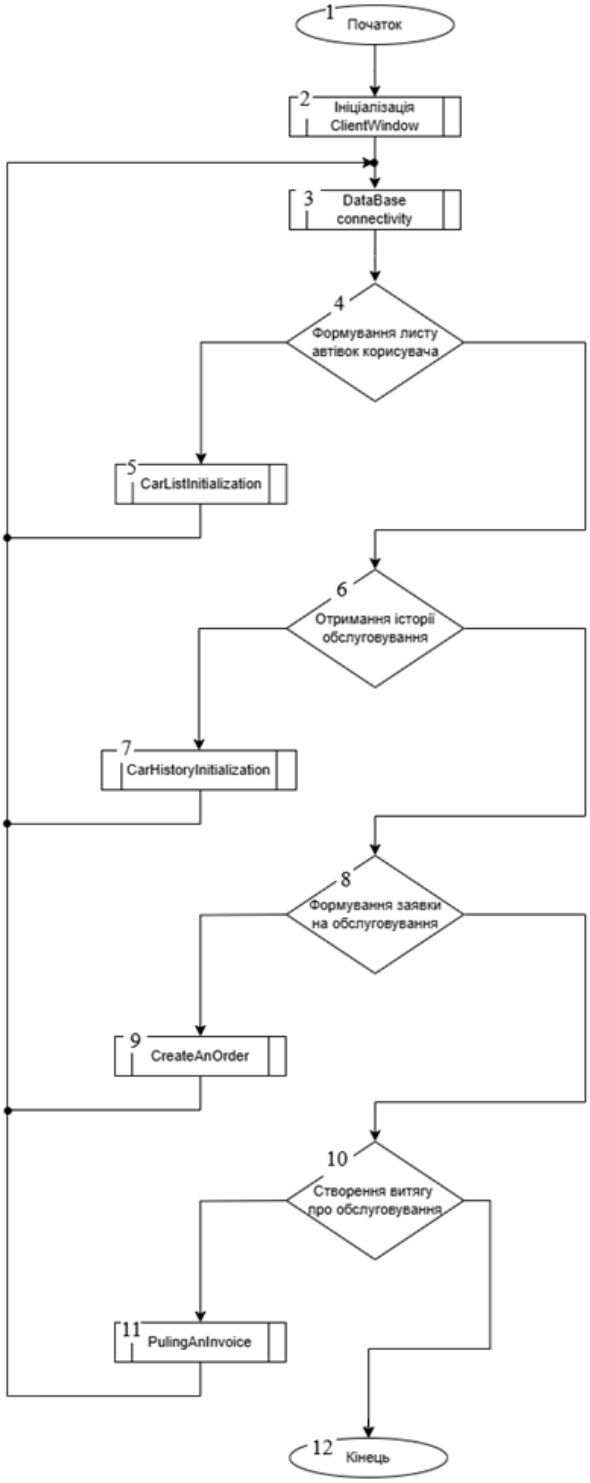


Рисунок 2.9 – Алгоритм роботи модуля клієнта

Під час входу в режим механіка, доступною є історія виконаних робіт даним механіком та наявність інструкцій до виконання наступних ремонтних робіт для автомобіля, відповідний алгоритм модуля зображено на рисунку 2.10.

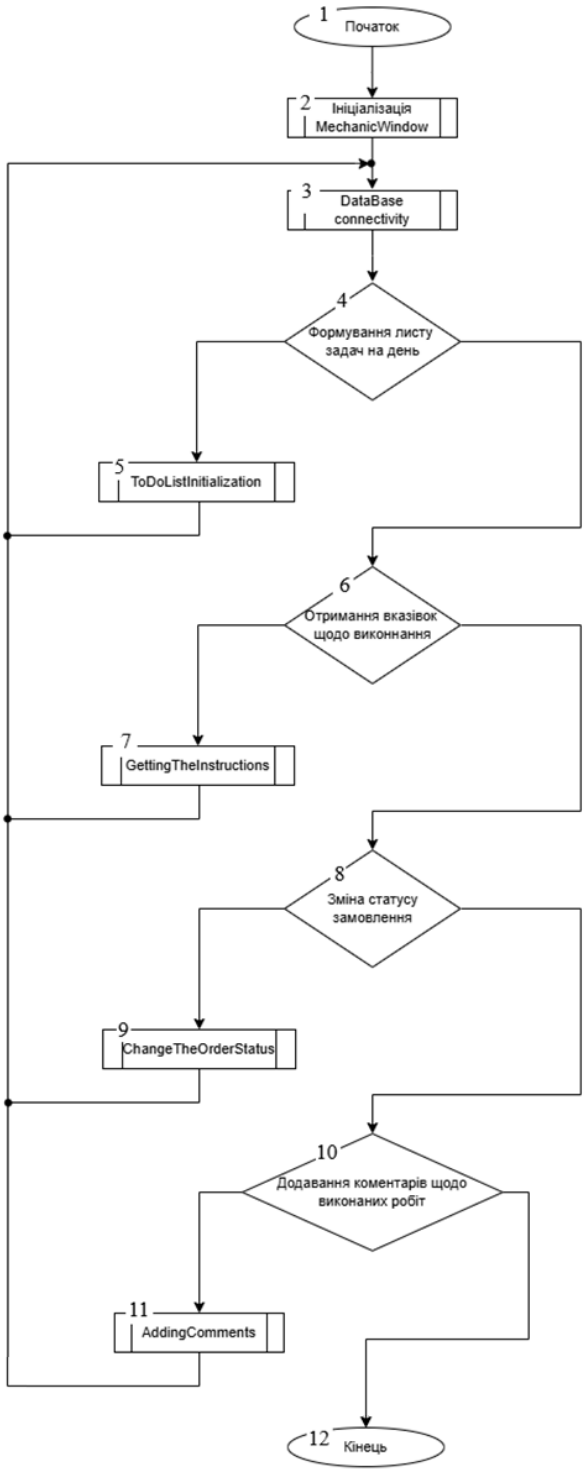


Рисунок 2.10 – Алгоритм роботи модуля клієнта

Для бухгалтера наявним є фінансовий облік автосервісу, а саме – коштові надходження за проведені ремонтні роботи й можливість формування відповідних місячних звітів, які демонструють прибуток даного автосервісу, рисунок 2.11.

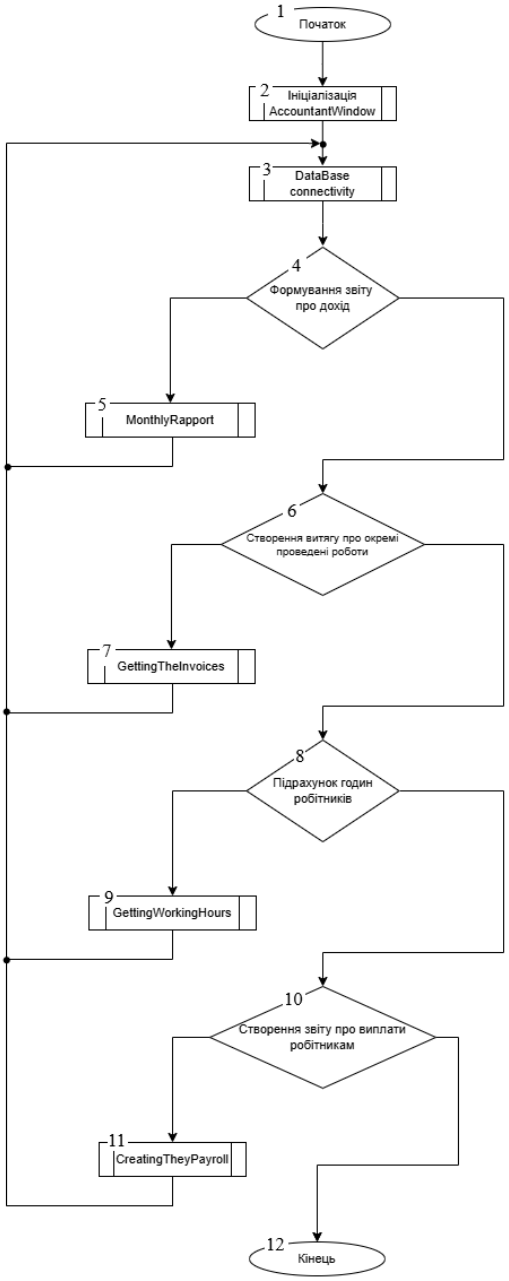


Рисунок 2.11 – Алгоритм роботи модуля бухгалтера

2.5 Висновки

У другому розділі було проведено аналіз вхідних та вихідних даних програмних засобів. Також було розглянуто процес створення графічного інтерфейсу програмного застосунку. Було розроблено основний алгоритм роботи програмних модулів та алгоритм роботи програмного застосунку. Відповідно було розроблену модель роботи програмного продукту за допомогою UML діаграм.

3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ДЛЯ РЕАЛІЗАЦІЇ МЕНЕДЖМЕНТУ ТА ОБЛІКУ ПОСЛУГ АВТО СЕРВІСУ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

При розробці програмного забезпечення для менеджменту та обліку послуг автосервісу важливо обрати найбільш оптимальні технології для реалізації кожного з функціональних модулів. JavaFX є потужною платформою для створення багатофункціональних та інтерактивних графічних інтерфейсів. Використання JavaFX дозволяє реалізувати зручний та інтуїтивно зрозумілий інтерфейс користувача, адаптований під потреби різних типів користувачів – клієнтів, механіків і бухгалтерів [21].

SceneBuilder – це інструмент для візуального проєктування інтерфейсів JavaFX-додатків, який дає можливість створювати графічні інтерфейси шляхом перетягування елементів (кнопок, полів введення, таблиць тощо), без необхідності написання FXML-коду вручну [5]. Його використання значно пришвидшує процес розробки GUI-частини програмного забезпечення та полегшує його подальше супроводження.

Для реалізації логіки обробки даних у проєкті використовується мова програмування Java, яка забезпечує високу продуктивність, надійність та сумісність із сучасними базами даних. Як сховище даних обрано реляційну базу даних MySQL, що підтримує транзакції, забезпечує збереження цілісності даних та зручну інтеграцію з Java-додатками за допомогою JDBC або ORM-фреймворків.

Крім цього, у системі передбачено використання модулів аналітики, які дозволяють відслідковувати ефективність роботи сервісу, будувати статистичні графіки та генерувати фінансову звітність. Для реалізації візуалізації статистичних даних можуть бути використані додаткові бібліотеки, наприклад, JavaFX Charts або сторонні API для побудови інтерактивних графіків.

Поєднання таких технологій, як JavaFX, SceneBuilder, MySQL та Java, дозволяє створити масштабовану програмну систему, яка охоплює всі ключові процеси автосервісу – від реєстрації клієнтів до формування фінансових звітів. Такий підхід забезпечує зручність у використанні, адаптивність, високу швидкість роботи та можливість подальшого розширення функціональності.

Отже, вибір технологій для розробки програмного забезпечення автосервісу має вирішальне значення для створення ефективного інструменту, що дозволяє оптимізувати роботу підприємства, зменшити кількість помилок, підвищити якість обслуговування клієнтів та забезпечити прозорість усіх процесів. Інтеграція сучасних технологій у програмне рішення відкриває нові можливості для цифрової трансформації в сфері технічного обслуговування автомобілів.

3.2 Розробка модуля робочого вікна клієнта

При розробці модуля клієнтського інтерфейсу важливо забезпечити можливість перегляду персоналізованої інформації для кожного клієнта автосервісу, а саме: перелік автомобілів, що йому належать, історію ремонтів кожного авто, зображення транспортного засобу та деталі кожного замовлення. У бакаларвській кваліфікаційній роботі ця функціональність реалізована у вигляді окремого класу ClientWindow, що відповідає за роботу відповідного GUI-вікна програми.

Даний модуль реалізований із застосуванням платформи JavaFX, яка дозволяє створювати сучасні графічні інтерфейси з високою продуктивністю. Для створення зручного, гнучкого та інтуїтивно зрозумілого інтерфейсу користувача використовуються компоненти JavaFX, такі як ListView, Label, ImageView, які забезпечують ефективну взаємодію користувача з програмою.

Клас ClientWindow реалізує функціональність перегляду даних користувача за наступною схемою:

- виведення списку автомобілів, які належать поточному клієнту;
- виведення зображення вибраного автомобіля;

• виведення історії ремонтів обраного автомобіля у вигляді списку;

• виведення детальної інформації про конкретне замовлення ремонту, включно з датою, типом ремонту, ПІБ механіка, пробігом та вартістю.

Модуль реалізує функціональність на основі чіткої архітектури, що дозволяє не тільки переглядати базові дані, але й отримувати деталізовану інформацію по кожному запису. При цьому особлива увага приділяється інтуїтивно зрозумілому інтерфейсу та зручності навігації (рис. 3.1).

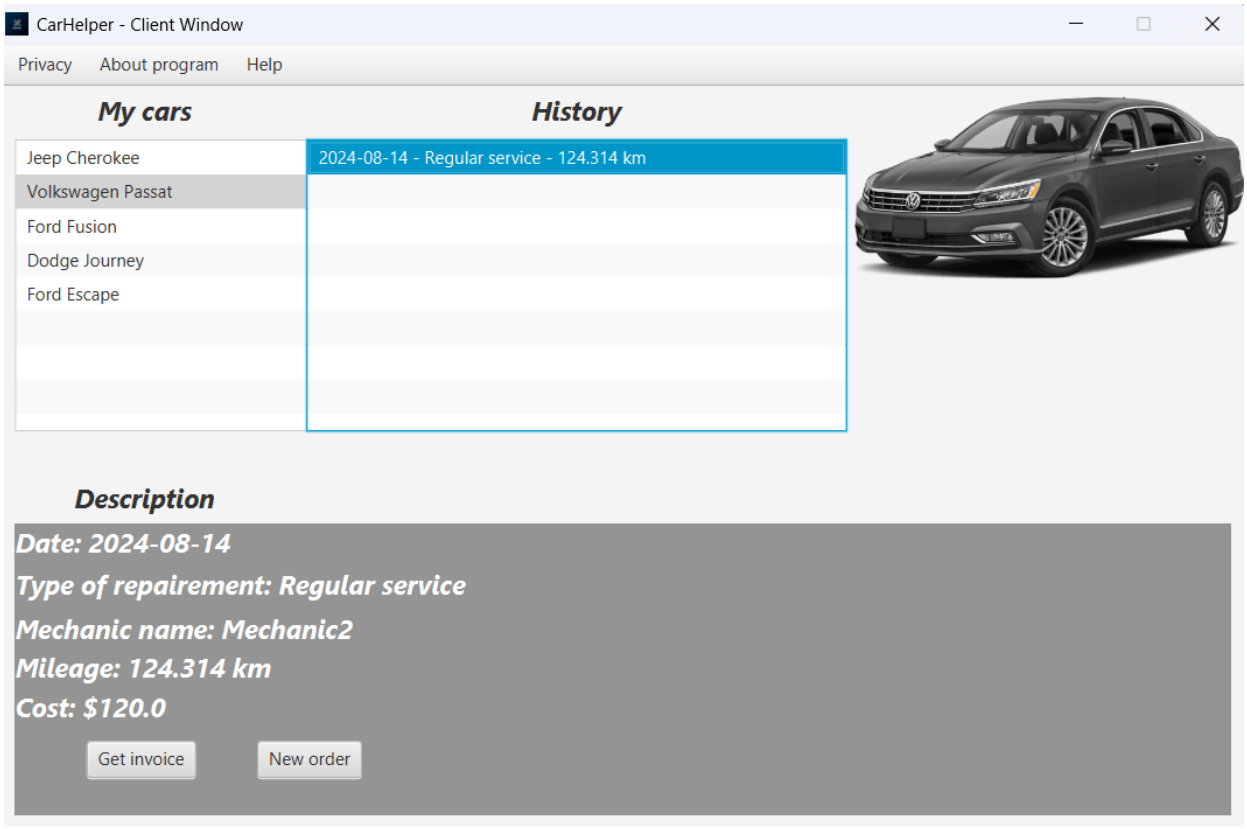


Рисунок 3.1 – Основне вікно відображення інформації клієнта

Після відкриття вікна відразу викликається метод initializeList(), що відповідає за відображення списку автомобілів поточного користувача. Цей метод автоматично виконується під час завантаження вікна. Його основні завдання:

• отримання списку автомобілів поточного користувача з бази даних через виклик DataBase.getCarsByClientName(...);

- очищення попередніх виборів у списку автомобілів та відображення оновленого списку;

підключення обробника подій для кожного елемента списку – при виборі авто завантажується зображення (метод initializeImage(...)) та пов’язана з ним історія ремонтів.

Після того як користувач обирає певний автомобіль, система завантажує дані про всі ремонти, виконані раніше для цього транспортного засобу. Для цього використовується метод DataBase.getRepairementsByClientAndCar(...), який отримує список коротких описів ремонтів, що включають дату ремонту, тип робіт, пробіг автомобіля тощо.

Ці дані відображаються у правому списку (ListView) графічного інтерфейсу, що дозволяє користувачеві зручно переглядати історію обслуговування автомобіля.

При виборі конкретного запису ремонту із списку, виводиться його повна інформація. Це реалізується за допомогою методу updateLabels(...), який:

- приймає масив string, що містять детальну інформацію про ремонт (дата, тип ремонту, механік, пробіг, вартість);
- розбиває кожен запис за допомогою роздільника " - ", щоб отримати окремі складові (дата, тип ремонту, ім’я механіка, пробіг та вартість);
- після цього метод записує отримані значення у відповідні компоненти Label на графічному інтерфейсі, що дозволяє користувачеві бачити деталі обраного ремонту (приклад: дата, тип ремонту, механік, пробіг та вартість).

Візуалізація та взаємодія з користувачем відбувається через програмне вікно, яке містить кілька ключових елементів інтерфейсу для забезпечення комфортної взаємодії:

- Список автомобілів – компонент ListView виводить всі автомобілі, зареєстровані за поточним користувачем. Користувач може вибрати автомобіль, щоб побачити історію його ремонтів.

• Зображення автомобіля – поряд з інформацією про авто відображається його зображення, що допомагає користувачеві ідентифікувати конкретний транспортний засіб.

• Історія ремонту – кожен ремонт виводиться як окремий запис у списку. Вибір конкретного ремонту відкриває детальну інформацію.

• Деталі ремонту – після вибору запису, користувач бачить усю інформацію, що стосується ремонту (дата, тип робіт, механік, пробіг, вартість).

Реалізація модуля ClientWindow орієнтована на найкращі практики UX/UI дизайну. Інтерфейс програми розроблений таким чином, щоб бути інтуїтивно зрозумілим, де кожен елемент сприяє зручності взаємодії з системою. Це дозволяє користувачеві без зайвих зусиль орієнтуватися в інтерфейсі, забезпечуючи максимальну зручність при навігації. Крім того, інтерфейс автоматично оновлюється, що дозволяє отримувати актуальну інформацію без необхідності вручну перезавантажувати дані. Усі елементи розташовані так, щоб користувач міг швидко і легко знайти потрібну інформацію, завдяки чітко структурованим спискам, зображенням і деталі кожного ремонту, що дає можливість ефективно працювати з історією обслуговування автомобілів.

Таким чином, модуль ClientWindow виконує ключову роль у клієнтській частині програмного забезпечення, забезпечуючи користувача візуально зрозумілим та логічно структурованим доступом до всієї історії обслуговування автомобілів. Його реалізація охоплює основні принципи UX/UI-дизайну, обробки подій, роботи з колекціями Java, та інтеграції з базою даних, що робить його універсальним і ефективним рішенням для практичного використання в системах обліку автосервісу.

3.3 Розробка модуля вікна для авторизації клієнта

Одним із базових модулів у структурі програмного забезпечення для автосервісу є модуль авторизації користувача. Надійна та інтуїтивно зрозуміла система входу дозволяє ідентифікувати клієнта, забезпечити конфіденційність облікових даних та надати персоналізований доступ до інформації.

Модуль LogInClientWindow є важливою складовою програмного забезпечення для автосервісу, що забезпечує надійну та зручну систему входу для клієнтів, див. рисунок 2.2. Його основне завдання – надання можливості клієнтам здійснювати авторизацію в системі, забезпечуючи їх конфіденційність і персоналізований доступ до облікових даних та історії обслуговування автомобілів. Цей модуль реалізовано за допомогою платформи JavaFX, що дозволяє створити гнучкий і інтерактивний графічний інтерфейс для користувача.

Інтерфейс модуля LogInClientWindow включає кілька ключових компонентів:

1. TextField clientNameLabel – поле для введення імені користувача. Клієнт має можливість ввести своє ім’я, яке є необхідним для ідентифікації в системі.

2. PasswordField clientPasswordLabel – поле для введення пароля. Це стандартний компонент для захищеного введення пароля, який автоматично приховує введені символи.

3. TextField unhiddenPassword – дане поле для відображення пароля у відкритому вигляді. Цей компонент дозволяє користувачу побачити свій пароль, що особливо корисно при введенні складних або довгих паролів, щоб уникнути помилок при наборі.

4. CheckBox showThePasswordButton – чекбокс, який дозволяє перемикати режим відображення пароля між захованим та відкритим. Коли цей чекбокс активовано, пароль відображається у відкритому вигляді, що допомагає користувачам перевірити правильність введення.

5. Label alertLabel – мітка для відображення повідомлень про помилки або успішні операції. Якщо введено неправильні дані, система відобразить відповідне повідомлення, наприклад, «Невірне ім’я користувача або пароль».

6. Button clientLoginButton – кнопка для підтвердження входу. При натисканні на неї система перевіряє введені дані, здійснює авторизацію і, в разі успіху, надає доступ до особистого кабінету клієнта.

Модуль LogInClientWindow має кілька важливих методів, які забезпечують його функціональність.

showThePassword() – цей метод дозволяє перемикати режим відображення пароля.

Коли користувач активує чекбокс showThePasswordButton, PasswordField (який приховує пароль) замінюється на TextField, в якому відображається введений текст. Це дозволяє клієнту побачити, що саме він ввів у поле пароля. Коли чекбокс знову деактивується, відображення пароля знову стає прихованим.

submit() – є основним метод обробки події входу. Він виконує кілька важливих операцій:

1. Зчитує ім’я користувача та пароль, введені клієнтом, з полів clientNameLabel та clientPasswordLabel.

2. Використовує ці дані для виклику методу DataBase.clientLogInDataBaseCheckUp(...), який перевіряє правильність введених даних за допомогою запиту до бази даних.

Якщо авторизація пройшла успішно:

- зберігає ім’я клієнта у змінну clientName, щоб у подальшому використовувати його для персоналізації інтерфейсу та доступу до даних користувача;
- виводить повідомлення про успіх входу;
- очищає поля введення, щоб підготувати інтерфейс до наступного використання;
- закриває поточне вікно авторизації та відкриває вікно клієнта (ClientWindow.fxml).

Якщо дані невірні:

- виводить повідомлення про помилку на alertLabel, наприклад, «Невірне ім’я користувача або пароль»;
- очищає поля введення, дозволяючи користувачеві спробувати ввести правильні дані знову.

OpenMainWindow() та OpenLogInClientWindow() – дані методи відповідають за відкриття нових вікон в системі. Метод OpenMainWindow() відкриває головне вікно програми, а метод OpenLogInClientWindow() ініціалізує вікно для повторної авторизації, якщо, наприклад, клієнт бажає вийти з системи і зайти знову. Обидва методи включають налаштування сцени, розміру вікна, іконки та інших параметрів.

OpenClientWindow() – метод відкриває основне вікно клієнта після успішної авторизаці. Це вікно містить персоналізовану інформацію для кожного клієнта, таку як список автомобілів, історію ремонтів тощо.

Модуль LogInClientWindow виконує ключову функцію в програмі — забезпечує безпечний та ефективний доступ до персоналізованих даних клієнта. Саме через цей модуль користувачі починають свою взаємодію з програмним забезпеченням, і він гарантує наступні функції:

- захист персональних даних відбувається через перевірку пароля та імені користувача за допомогою бази даних забезпечує конфіденційність інформації клієнта;
- запобігання несанкціонованому доступу – відбувається завдяки використанню захищених полів для пароля та перевірки даних з бази, система не дозволяє входити до неї з помилковими або фальшивими даними;
- персоналізація інтерфейсу відбувається після успішної авторизації й дозволяє налаштувати інтерфейс, що включає дані конкретного клієнта, такі як список автомобілів, історію обслуговування, а також доступ до інших персоналізованих функцій.

інформування користувача про помилки авторизації. Надання чітких і зрозумілих повідомлень про помилки – одна з важливих функцій модуля LogInClientWindow. Коли клієнт вводить неправильне ім’я користувача або пароль, система негайно виводить повідомлення типу «Невірне ім’я користувача або пароль» у полі alertLabel.

Це дозволяє уникнути плутанини та спрямовує користувача до виправлення помилки без потреби в додатковій технічній підтримці. Такий підхід

не лише покращує взаємодію з користувачем, а й зменшує ризик повторних помилок під час входу. Крім того, вчасне повідомлення про невдалу авторизацію дозволяє клієнту оперативно реагувати – перевірити розкладку клавіатури, уважніше ввести пароль або скористатися функцією відновлення доступу, якщо така реалізована. Усе це робить систему більш інтуїтивно зрозумілою, надійною та дружньою до користувача.

Збереження імені клієнта в змінній clientName є важливим аспектом для подальшої роботи системи, оскільки це дає змогу використовувати ці дані для отримання конкретної інформації з бази даних в інших модулях, таких як перегляд історії ремонту автомобілів або зміна персональних даних.

Таким чином, модуль LogInClientWindow не лише забезпечує базову функціональність авторизації, але й відіграє важливу роль у збереженні безпеки даних, інтеграції з іншими модулями системи, а також у персоналізації досвіду користувача в межах програмного забезпечення для автосервісу.

3.4 Розробка модуля робочого вікна диспетчера

Для розробки модуля диспетчерського інтерфейсу важливо забезпечити можливість перегляду інформації про запити на виконання ремонтних робіт в автосервісі, а саме: перелік запитів від клієнтів та їх контакти для уточнення інформації, у випадку її відсутності, перелік наявних механіків в автосервісі та їх календар зайнятості. У бакалаврській кваліфікаційній роботі ця функціональність реалізована у вигляді окремого класу DispatchWindow, що відповідає за роботу відповідного GUI-вікна програми.

Для реалізації робочого вікна диспетчера спочатку потрібно залогуватись в відповідному вікні, після вибору ролі користувача в основному вікні програми. Для цього було створено модуль вікна логування LogInDispatchWindow. Дане рішення забезпечує надійну та зручну систему входу для диспетчера, див. рисунок 3.2.

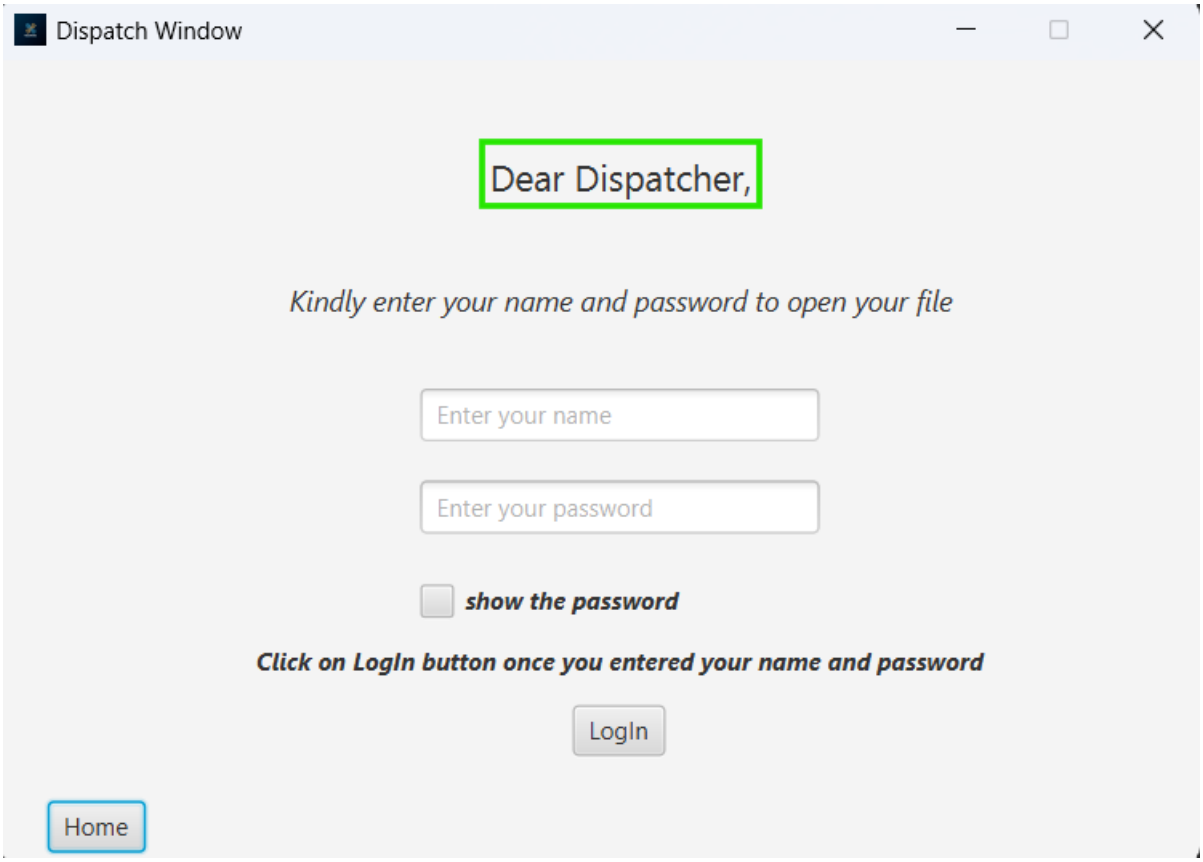


Рисунок 3.2 – Вікно логування диспетчера

Його основне завдання – надання можливості користувачам програми здійснювати авторизацію в системі, забезпечуючи їх конфіденційність і персоналізований доступ до відповідних даних в системі.

Завдяки платформі JavaFX модуль DispatchWindow отримав сучасний графічний інтерфейс, що вирізняється високою продуктивністю. Компоненти JavaFX, зокрема ListView, Label та ImageView, були використані для розробки гнучкого та інтуїтивно зрозумілого інтерфейсу, що значно покращує взаємодію користувача з програмою.

Клас DispatchWindow реалізує функціональність перегляду даних для диспетчера за наступною схемою:

- виведення списку заявок від клієнтів;
- виведення зображення клієнта, який сформував заявку;
- виведення контактної інформації про клієнта;
- виведення переліку наявних механіків, які працюють в автосервісі.

Інтерфейс вікна диспетчера продемонстровано на рисунку 3.3.

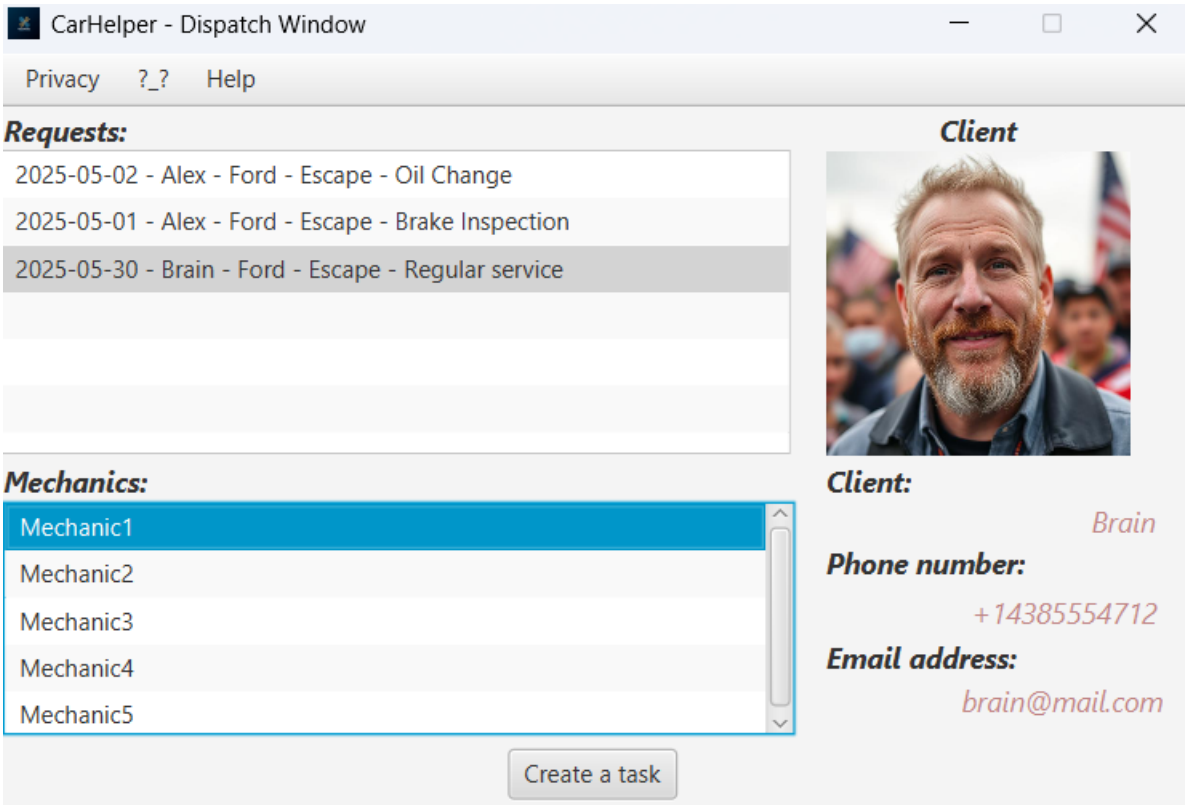


Рисунок 3.3 – Інтерфейс робочого вікна диспетчера

Основні функції та компоненти даного модуля (DispatchWindow):

• openOrderCreationWindow - даний метод, асоційований з кнопкою openOrderCreationWindow, відкриває нове вікно OrderCreationWindow, що дозволяє диспетчеру створювати нові замовлення. Це реалізовано шляхом завантаження відповідного FXML-файлу та встановлення заголовка вікна.

• requests (ListView<String>) - є компонентом ListView, який відображає список поточних запитів на виконання ремонтних робіт. При виборі елемента зі списку, інформація про відповідного клієнта автоматично завантажується та відображається в спеціальних полях, що дозволяє диспетчеру зв’язуватись за необхідністю з клієнтами для уточнення інформації.

• mechanicList (ListView<String>) – є елементом об’єкту ListView, що містить список імен доступних механіків в автосервісі. Вибір механіка зі списку

зберігає його ім'я для подальшого використання при створенні та призначенні замовлення.

• інформаційні поля клієнта (clientNameField, clientPhNumberField, clientEmailAddressField) є об'єктами класу Label, які динамічно оновлюються, відображаючи ім'я, номер телефону та електронну адресу клієнта, відповідно до обраного запиту у списку requests.

• clientImage (ImageView) – є об'єктом ImageView, призначений для відображення зображення (аватара) клієнта, пов'язаного з вибраним запитом раніше.

• requestOrderInfo та mechanicName (статичні поля) – дані статичні змінні використовуються для тимчасового зберігання інформації про вибраний запит на замовлення та ім'я механіка відповідно. Це дозволяє передавати дані між різними вікнами й компонентами системи. В подальшому вони будуть використовуватись в модулі створення замовлення – orderCreationWindow.

• initializeTheListOfRequests() - є методом, який викликається при ініціалізації вікна. Він відповідає за завантаження списку запитів на замовлення з бази даних (DataBase.getOrderRequestDisplayStrings()) та їх відображення у requests.

Реалізацію обробника подій для requests, який при виборі запиту отримує контактну інформацію клієнта (DataBase.getClientContactInfo()) та оновлює відповідні Label та ImageView, що відображають інформацію про клієнта. Також відбувається завантаження списку імен механіків з бази даних (DataBase.getMechanicNames()) та їх відображення у mechanicList.

Реалізацію обробника подій для mechanicList, який зберігає вибране ім'я механіка у статичній змінній mechanicName, що допомагає в створенні замовлення в модулі orderCreationWindow.

Статистичний метод openWindow(String fxmlFile, String title) забезпечує стандартизований спосіб відкриття нового вікна DispatchWindow. Він завантажує FXML-файл, ініціалізує контролер, встановлює іконку, забороняє зміну розміру вікна та відображає його з вказаним заголовком. Важливою частиною цього

методу є виклик controller.initializeTheListOfRequests(), що гарантує відображення актуальних даних при відкритті вікна.

Взаємодія класу DispatchWindow з базою даних відбувається через клас DataBase для отримання необхідних даних, таких як список замовлень, контактна інформація клієнтів та імена механіків. Це підкреслює архітектуру, де контролер відповідає за логіку відображення даних, а доступ до даних інкапсульований в окремому шарі, для відповідності принципам об'єктно-орієнтованого програмування.

Клас OrderCreationWindow є контролером JavaFX, який керує інтерфейсом для створення та перегляду замовлень на ремонт автомобілів, див. рисунок 3.4.

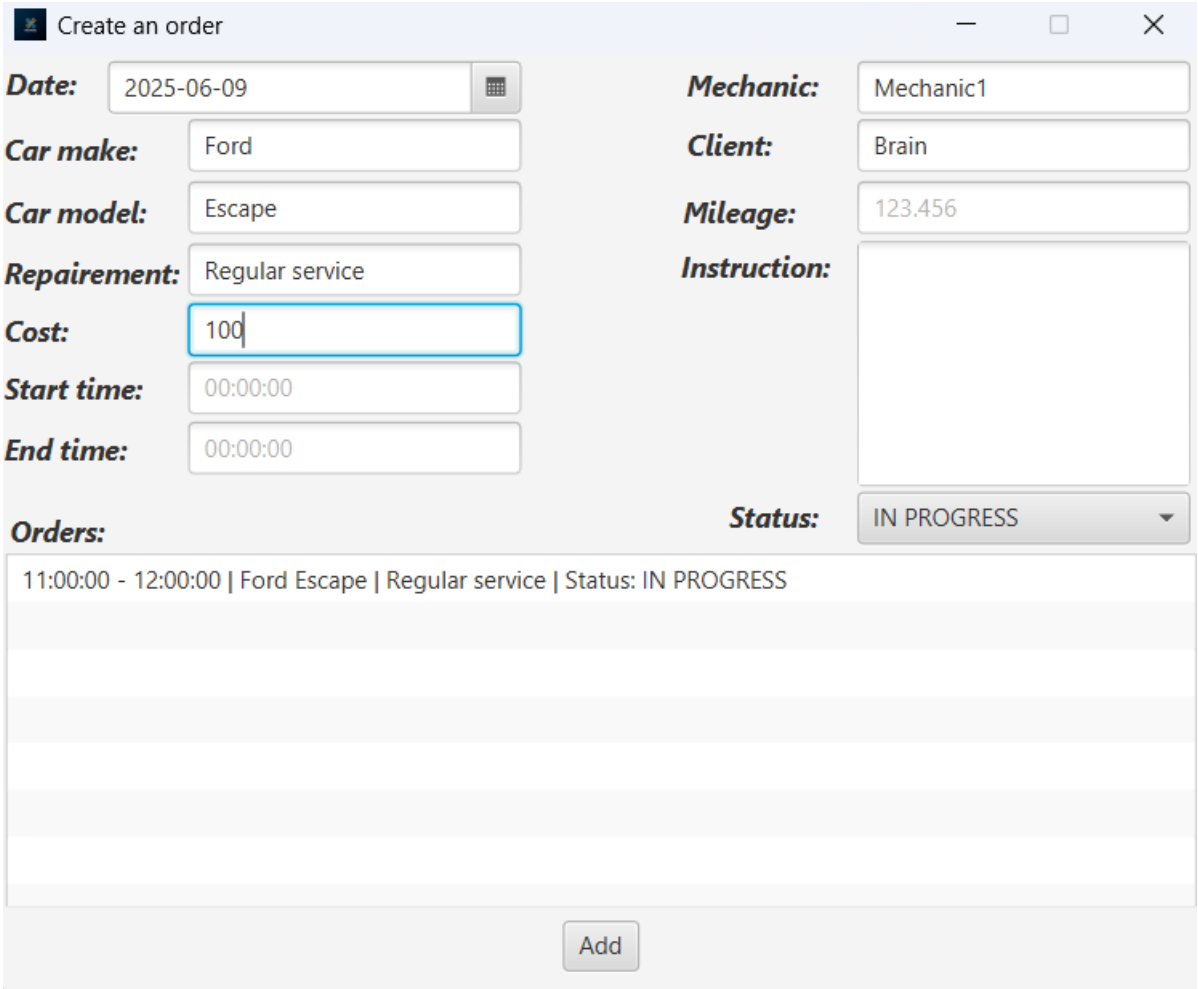


Рисунок 3.4 – Вигляд вікна для створення/призначення замовлення механікам

Він автоматично заповнює поля даними з попереднього вікна, дозволяє користувачу вводити деталі замовлення (інформація про авто, ремонт, вартість, час, механік, клієнт, пробіг, інструкції) та вибирати статус замовлення. Контролер взаємодіє з базою даних для збереження нових замовлень та відображення існуючих, а також забезпечує валідацію введених даних.

Отже, разом DispatchWindow та OrderCreationWindow утворюють цілісну систему для управління життєвим циклом запитів на ремонт транспортних засобів. DispatchWindow являє собою первинний центр сортування та призначення, дозволяючи диспетчеру переглядати нові запити на обслуговування та призначати їх механікам. Після вибору запиту та механіка, OrderCreationWindow бере на себе функцію, надаючи спеціалізований інтерфейс для деталізації та формалізації замовлення на ремонт й відповідного внесення його в базу даних. Такий поділ відповідальності забезпечує структурований робочий процес, від початкової обробки запитів та призначення до детального створення та відстеження ремонтних робіт, що зрештою підвищує ефективність роботи автосервісу.

3.5 Висновки

У третьому розділі було обґрунтовано вибір технологій та інструментів, що використовувались для розробки програмного забезпечення автосервісу. Для створення графічного інтерфейсу застосовано JavaFX у поєднанні з SceneBuilder. В якості мови програмування обрано Java, а для зберігання даних – базу даних MySQL. У розділі описано реалізацію основних модулів: вікна авторизації клієнта, клієнтського інтерфейсу та логіки взаємодії з базою даних. Реалізовані рішення забезпечують зручність використання, модульність і ефективну роботу системи.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування програмного забезпечення є невід'ємним етапом життєвого циклу розробки, який дозволяє виявити помилки, дефекти та невідповідності ще до початку реальної експлуатації системи. Основною метою тестування є перевірка функціональності, стабільності та відповідності розробленого продукту технічним вимогам і очікуванням кінцевого користувача.

Проведення тестування забезпечує впевненість у якості програмного забезпечення, дозволяє вчасно усунути критичні недоліки та підвищує загальну надійність системи. У даному проєкті було проведено комплексне тестування клієнтської частини системи обліку автосервісу, що включало функціональні, відмовостійкі, продуктивні та сумісні випробування.

Перед початком тестування програму було розгорнуто на тестовому середовищі з операційною системою Windows 11. Було також підготовлено усю необхідну документацію: технічне завдання, опис інтерфейсів та сценарії використання, які лягли в основу тест-плану.

4.2 Функціональне тестування

На цьому етапі перевірялась коректність реалізації основних функцій додатку:

- вхід користувача до системи через форму авторизації (LogInClientWindow);
- перевірка правильності обробки введених логінів і паролів (як вірних, так і невірних);
- завантаження списку автомобілів користувача;
- відображення історії обслуговування обраного автомобіля;
- оновлення інформації при виборі запису з історії;
- відображення зображення автомобіля та детальної інформації про замовлення (тип ремонту, дата, механік, вартість, пробіг).

На рис. 4.1 показано результат тестування авторизації клієнта з вірними обліковими даними.

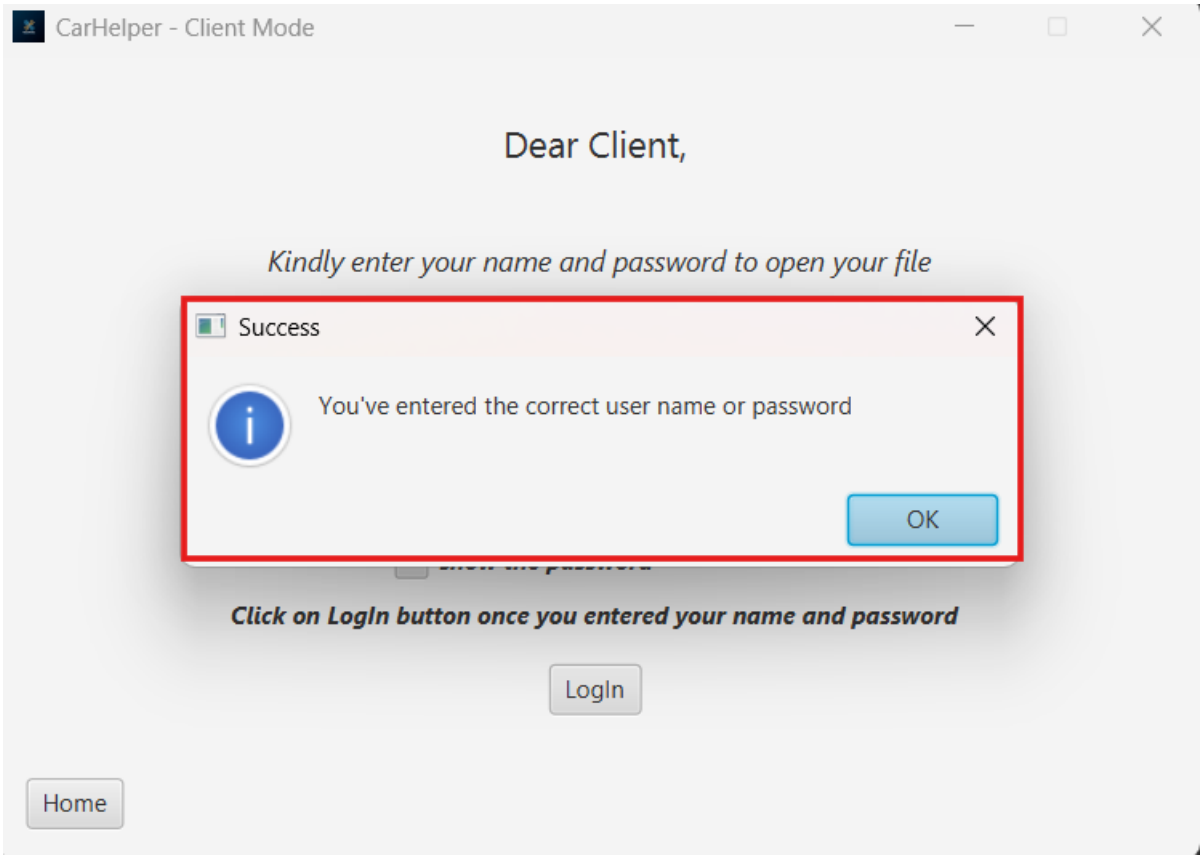


Рисунок 4.1 – Повідомлення про успішний вхід до системи клієнта

Тестування відображення історії обслуговування та автомобіля клієнта. Однією з функціонально важливих частин програмного забезпечення є правильне відображення історії обслуговування автомобіля після входу клієнта в систему. Для перевірки цієї функції було проведено тестування, яке мало на меті визначити, чи правильно:

- завантажується список автомобілів, пов’язаний з користувачем;
- завантажується зображення відповідного автомобіля;
- виводиться повна історія ремонтів для вибраного авто;
- оновлюються текстові поля з деталями обслуговування при виборі запису.

Під час тестування клієнт виконував вхід у систему через вікно авторизації, після чого відкривалося клієнтське вікно. Було обрано один із автомобілів, який

належить даному користувачу, та з переліку історії обрано конкретний запис про ремонт.

На рисунку 4.2 показано результат функціонального тестування, під час якого перевіряється відображення детальної інформації в інтерфейсі після вибору автомобіля та конкретного запису з історії.

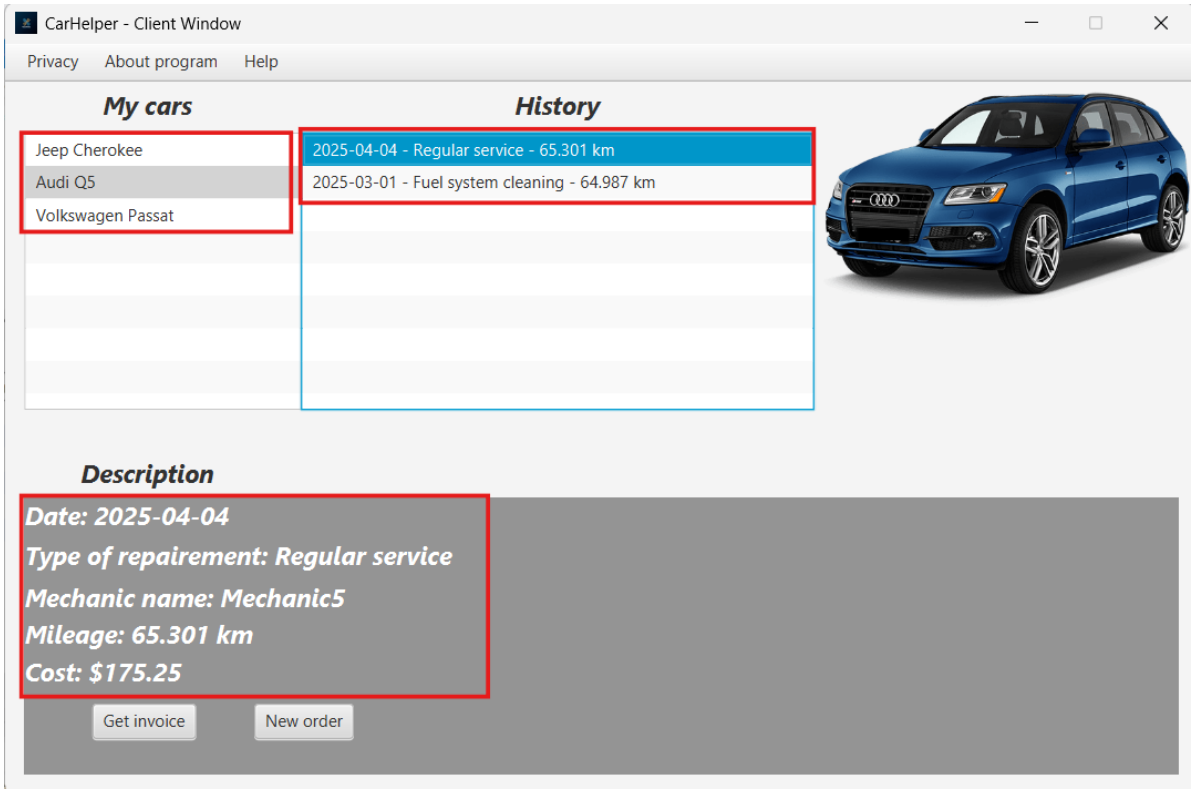
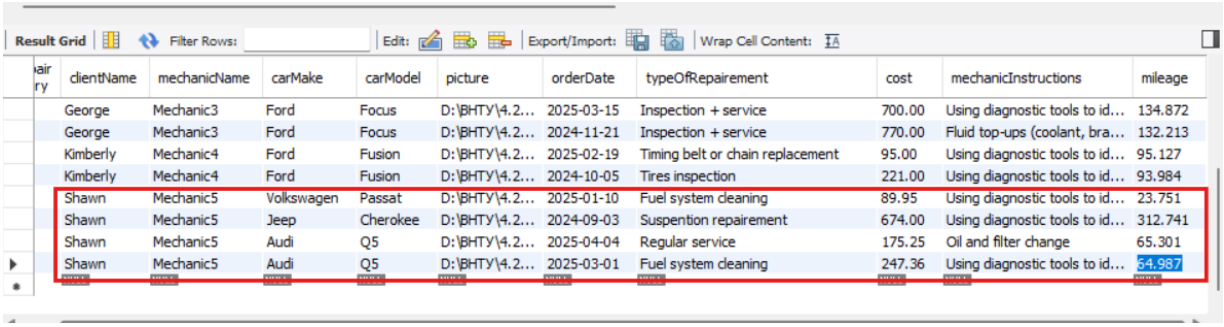


Рисунок 4.2 – Результат функціонального тесту: відображення історії обслуговування та авто у клієнтському вікні

Для перевірки коректності завантажених даних було виконано додатковий аналіз інформації, отриманої з бази даних. За допомогою SQL-запиту до відповідної таблиці (repairments, cars, clients тощо) було виведено записи, що відповідають клієнту, вибраному автомобілю та історії його обслуговування. Це дозволило звірити, чи повністю співпадає інформація, яку бачить користувач в інтерфейсі, з реальною інформацією, що зберігається в базі даних.

На рисунку 4.3 показано результат перевірки бази даних: відображено записи, які відповідають ремонту, обраному у попередньому тесті. Таке

порівняння дозволяє підтвердити, що логіка вибірки в додатку працює коректно, не виникає помилок під час завантаження даних, і жодна інформація не втрачається чи не спотворюється.



clientName	mechanicName	carMake	carModel	picture	orderDate	typeOfRepairement	cost	mechanicInstructions	mileage
George	Mechanic3	Ford	Focus	D:\ВНТУ\4.2...	2025-03-15	Inspection + service	700.00	Using diagnostic tools to id...	134.872
George	Mechanic3	Ford	Focus	D:\ВНТУ\4.2...	2024-11-21	Inspection + service	770.00	Fluid top-ups (coolant, bra...	132.213
Kimberly	Mechanic4	Ford	Fusion	D:\ВНТУ\4.2...	2025-02-19	Timing belt or chain replacement	95.00	Using diagnostic tools to id...	95.127
Kimberly	Mechanic4	Ford	Fusion	D:\ВНТУ\4.2...	2024-10-05	Tires inspection	221.00	Using diagnostic tools to id...	93.984
Shawn	Mechanic5	Volkswagen	Passat	D:\ВНТУ\4.2...	2025-01-10	Fuel system cleaning	89.95	Using diagnostic tools to id...	23.751
Shawn	Mechanic5	Jeep	Cherokee	D:\ВНТУ\4.2...	2024-09-03	Suspention repairement	674.00	Using diagnostic tools to id...	312.741
Shawn	Mechanic5	Audi	Q5	D:\ВНТУ\4.2...	2025-04-04	Regular service	175.25	Oil and filter change	65.301
Shawn	Mechanic5	Audi	Q5	D:\ВНТУ\4.2...	2025-03-01	Fuel system cleaning	247.36	Using diagnostic tools to id...	64.987

Рисунок 4.3 – SQL-запис у базі даних, що відповідає виведеній інформації в клієнтському вікні

Після завершення тестування встановлено, що:

- усі відповідні автомобілі клієнта коректно завантажуються;
- зображення авто відповідає запису в базі;
- історія ремонтів відображається повністю;
- усі поля деталізації замовлення (дата, пробіг, вартість, механік, тип ремонту) точно відображають відповідні значення в базі даних.

Таким чином, результати тестування підтверджують правильність реалізації логіки взаємодії між інтерфейсом користувача та базою даних.

4.3 Тестування відмов

Цей етап був спрямований на перевірку стійкості програми до введення некоректних даних або до виникнення помилкових ситуацій:

- введення неправильного логіна або пароля;
- спроба входу без заповнення обов’язкових полів;
- відсутність підключення до бази даних;
- вибір елементів у списку, що не мають пов’язаних даних в базі;
- завантаження зображення автомобіля з недійсним або порожнім шляхом.

У всіх цих випадках програма обробляла ситуації належним чином: виводила сповіщення, не допускала зависання інтерфейсу або некоректних переходів.

Під час першого тесту перевірялась реакція програми на введення невірного логіна або пароля. Якщо користувач вводив некоректні дані, система виводила відповідне повідомлення та очищувала поля для повторного введення. На рисунку 4.4 показано інтерфейс із повідомленням про помилкову спробу входу.

Рисунок 4.4 – Повідомлення про помилкове введення логіна або пароля

4.4 Тестування продуктивності

Під час тестування продуктивності проводився вимір часу відгуку додатку на:

- вибір елементів списку;
- завантаження зображення автомобіля;
- відображення історії обслуговування;
- відкриття нових вікон (наприклад, ClientWindow.fxml).

Усі операції виконувались миттєво або з незначною затримкою (менше 0,5с), що відповідає сучасним вимогам до десктопних систем.

4.5 Тестування сумісності

Було протестовано працездатність додатку на різних середовищах:

- ОС Windows 10 та Windows 11;
- дисплеї з різною роздільною здатністю;

- різні версії Java Runtime Environment (JRE 11 та JRE 17).

Під час тестування додаток демонстрував стабільну роботу та відсутність конфліктів.

4.6 Розробка інструкції користувача

Інструкція користувача передбачає опис технічних вимог до запуску програмного забезпечення, а також основні етапи взаємодії користувача з системою. У таблиці 4.1 наведено мінімальну та рекомендовану конфігурацію комп’ютера для стабільної роботи програми.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i5 або AMD Ryzen 5	Intel Core i3 або AMD Ryzen 3
Оперативна пам'ять	8 ГБ RAM	4 ГБ RAM
Вільний простір на диску	10 ГБ	3 ГБ
Відеокарта	NVIDIA GeForce GTX 1050 або вище	Вбудована графіка Intel HD Graphics
Операційна система	Windows 10/11 або macOS Big Sur і вище	Windows 7 або macOS Mojave
Монітор	1920×1080, діагональ 19" або більше	1366×768, діагональ 15" або більше

Після встановлення та запуску додатку, користувач потрапляє на головний екран авторизації (рис. 4.5), де йому пропонується обрати вид користувача та згодом ввести ім’я користувача та пароль. У разі введення коректних даних, система виконує авторизацію та відкриває основне клієнтське вікно.

Першим етапом взаємодії користувача є вибір автомобіля зі списку, пов’язаного з його профілем. Після вибору авто автоматично завантажується:

- зображення автомобіля, отримане з бази даних;
- список попередніх обслуговувань, пов’язаних із цим автомобілем.

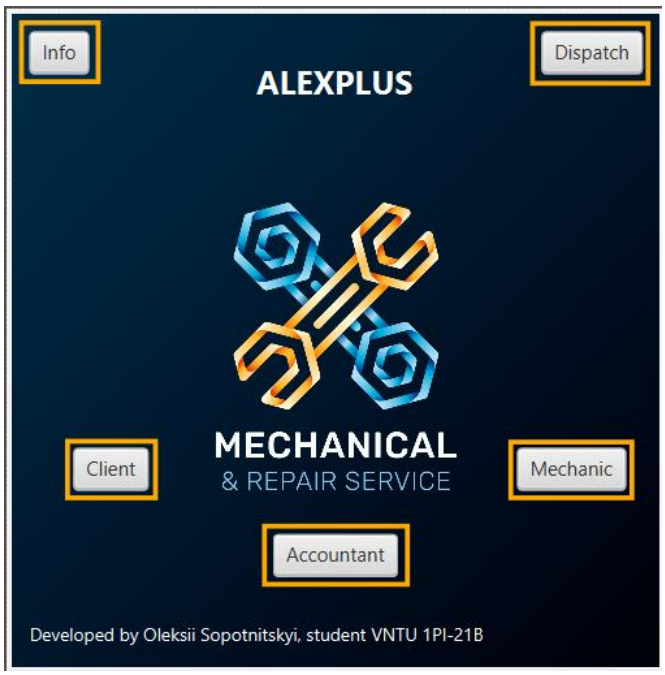


Рисунок 4.5 – Головний екран додатку

Далі користувач може обрати один із записів історії ремонту, після чого в центральній частині інтерфейсу буде відображено детальну інформацію:

- дату ремонту;
- тип виконаних робіт;
- ім’я механіка;
- пробіг автомобіля;
- вартість виконаних послуг (рис. 4.6).

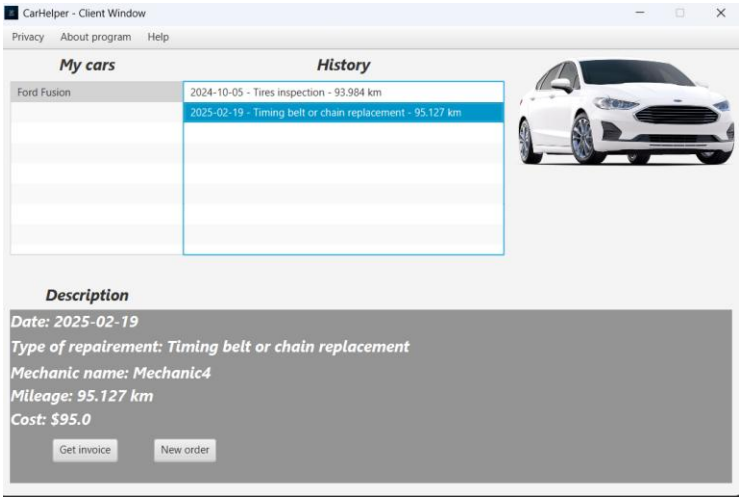


Рисунок 4.6 – Екран клієнта

Усі взаємодії виконуються за допомогою миші або клавіатури, а інтерфейс побудований на основі принципів зручності, інтуїтивності та швидкого доступу до основних функцій.

Після завершення роботи користувач може закрити клієнтське вікно або повернутися до головного меню. Якщо в системі реалізовано функцію експорту, результати обслуговування або історію ремонту можна експортувати у файл формату .xlsx або зберегти в архів для подальшого перегляду.

4.7 Висновки

У четвертому розділі було проведено тестування програмних модулів системи обліку та менеджменту автосервісу. Було перевірено реакцію програмного забезпечення на типові та нетипові сценарії використання, зокрема: введення некоректних даних під час авторизації, вибір автомобіля без історії обслуговування, та взаємодію з елементами інтерфейсу в нестандартних умовах.

Окрему увагу приділено перевірці функціональності взаємодії з базою даних, а саме – правильності відображення інформації про клієнтів, автомобілі та виконані ремонти. За результатами тестування підтверджено стабільну роботу системи та коректність обробки даних.

Також у розділі було розроблено інструкцію користувача щодо встановлення, запуску та початкової роботи з програмним забезпеченням, з описом системних вимог та ключових етапів взаємодії з додатком.

ВИСНОВКИ

У рамках виконання бакалаврської кваліфікаційної роботи було розроблено модулі для автоматизації процесів обліку, управління та взаємодії з клієнтами в системі автосервісу. Для реалізації проєкту використовувалося середовище розробки IntelliJ IDEA, а також мова програмування Java із застосуванням бібліотеки JavaFX для створення графічного інтерфейсу.

Під час виконання роботи було проаналізовано поточний стан проблеми автоматизації в галузі технічного обслуговування автомобілів. Розглянуто існуючі аналоги подібних систем, проаналізовано їхні недоліки та функціональні обмеження, що дозволило сформувати цілі та завдання власного програмного продукту.

У процесі аналізу технологій було обґрунтовано вибір мови Java, використання JavaFX як основної технології для створення інтерфейсу, а також бази даних MySQL для зберігання інформації про клієнтів, автомобілі та обслуговування.

У межах бакалаврської кваліфікаційної роботи проєкту було виконано наступне:

- сформульовано основні функціональні та нефункціональні вимоги до системи;
- розроблено модуль авторизації клієнта з перевіркою даних;
- реалізовано клієнтське вікно з можливістю перегляду автомобілів, історії ремонтів і деталей замовлень;
- забезпечено динамічне відображення зображення авто та інформації з бази даних;
- реалізовано зручний та інтуїтивно зрозумілий графічний інтерфейс;
- проведено комплексне тестування програмного забезпечення згідно з технічним завданням.

Було розроблено схеми загального алгоритму роботи програми, а також UML-діаграми, що відображають логіку та взаємодію основних класів і модулів системи.

Результати тестування підтвердили повну працездатність програмного продукту, його стабільність, відповідність функціональним вимогам і готовність до використання в умовах реального автосервісу. Крім того, було створено інструкцію користувача, що описує етапи встановлення, запуску та використання програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сопотніцький О. Є., Катєльніков Д. І. JAVAFX У 2025 РОЦІ: ЧОМУ ЦЯ ТЕХНОЛОГІЯ НЕОБХІДНА ДЛЯ БІЗНЕС-ДОДАТКІВ // LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії – 2025: матеріали конференції (2025). – С. 341–342.

2. Колодій В.І., Кісіль М.В. Інформаційні технології в управлінні підприємством. – Київ: КНЕУ, 2021. – 184 с.

3. Smith J. Automotive Service Management: Principles and Practice. – New York: McGraw-Hill Education, 2020. – 240 p.

4. Кравченко С.І. Цифровізація автосервісних підприємств: проблеми і перспективи. – Харків: ХНАДУ, 2021. – 152 с.

5. Brown A. Desktop Applications in Modern Auto Service. – Wiley, 2020. – 175 p.

6. Жук О.П. Захист даних у хмарних обчисленнях. – Львів: Видавництво ЛНУ, 2022. – 143 с.

7. Adams C. System Integration in Auto Service Management. – Cambridge University Press, 2021. – 195 p.

8. Nguyen L. Modular Software Design for Automotive Businesses. – Springer, 2021. – 182 p.

9. АвтоМайстер CRM [Електронний ресурс]. – Режим доступу: https://automastercrm.com/

10. CarServicePro [Електронний ресурс]. – Режим доступу: https://carservicepro.com/

11. AutoSoft Web [Електронний ресурс]. – Режим доступу: https://autosoftweb.com/

12. Workshop ERP [Електронний ресурс]. – Режим доступу: https://workshop-erp.com/

13. Мотор Менеджер [Електронний ресурс]. – Режим доступу: https://motormanager.com/

14. MySQL Reference Manual [Електронний ресурс]. – Режим доступу: https://dev.mysql.com/doc/

15. Best Practices for UI Design – Nielsen Norman Group [Електронний ресурс]. – Режим доступу: https://www.nngroup.com/articles/

16. DuBois P. MySQL: The Complete Reference. – McGraw-Hill, 2003. – 1200 p.

17. Любарська Л.С. Психологія кольору в дизайні: практичний посібник. – Київ: КНУКіМ, 2020. – 128 с.

18. Krug S. Don’t Make Me Think: A Common Sense Approach to Web Usability. – New Riders, 2014.

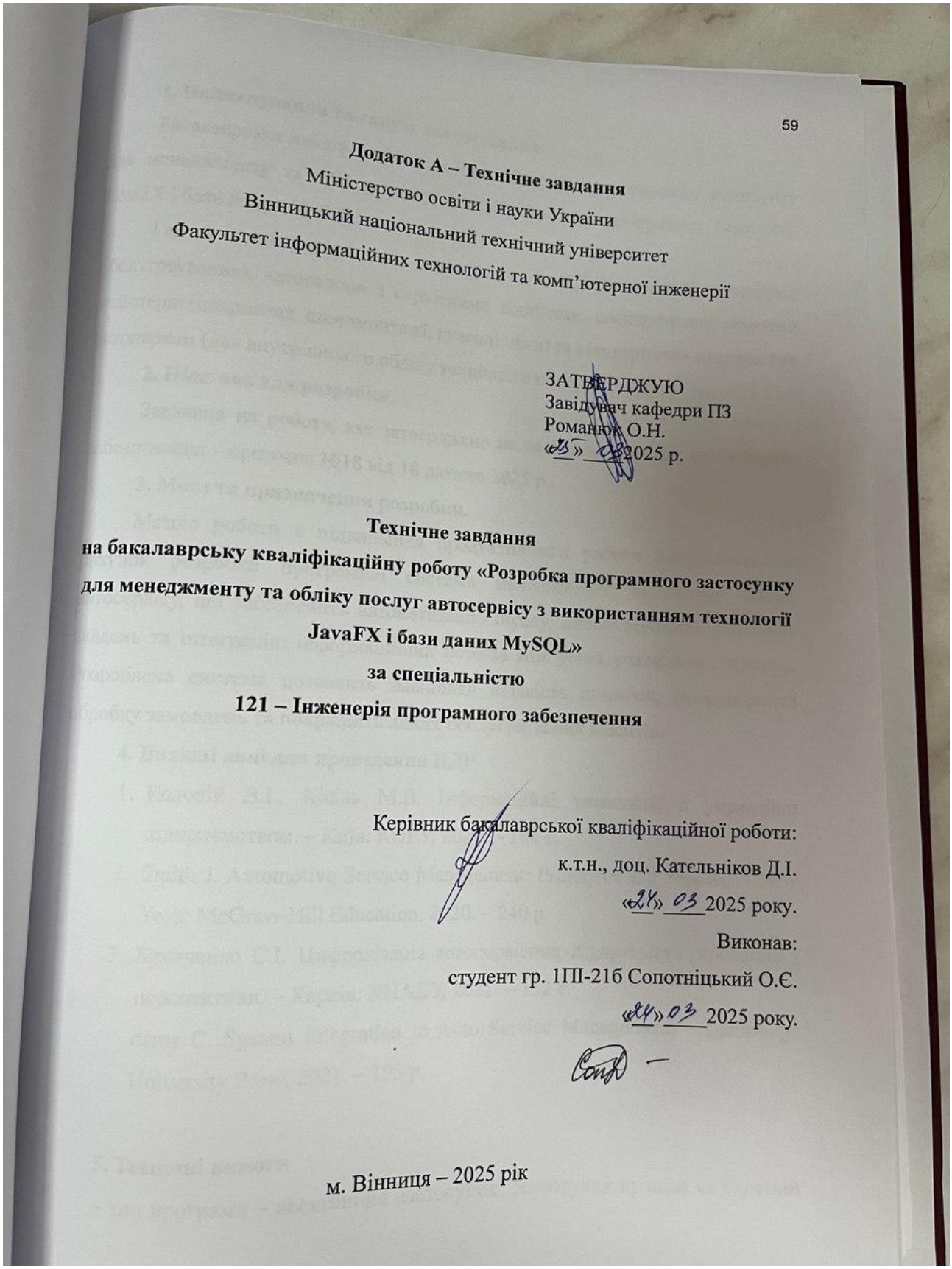
19. Nielsen J. Usability Engineering. – Morgan Kaufmann, 1994.

20. ISO 9241-110:2020. Ergonomics of human-system interaction — Part 110: Interaction principles. – International Organization for Standardization, 2020.

21. Evans, Eric. Domain-Driven Design: Tackling Complexity in the Heart of Software. – Addison-Wesley, 2003. – 560 p.

ДОДАТКИ

Додаток А – Технічне завдання



59

Додаток А – Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» 03 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка програмного застосунку для менеджменту та обліку послуг автосервісу з використанням технології JavaFX і бази даних MySQL»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Катєльніков Д.І.
«24» 03 2025 року.

Виконав:
студент гр. 1ПІ-21б Сопотніцький О.Є.
«24» 03 2025 року.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмного застосунку для менеджменту та обліку послуг автосервісу з використанням технології JavaFX і бази даних MySQL».

Галузь застосування – автосервіси та СТО (Станції технічного обслуговування), автосалони з сервісними відділами, спеціалізовані ремонтні майстерні (наприклад, шиномонтажі, кузовні цехи) та підприємства з управління автопарком (для внутрішнього обліку технічного обслуговування).

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №18 від 18 лютого 2025 р..

3. Мета та призначення розробки.

Метою роботи є підвищення продуктивності роботи підприємства за рахунок розробки програмної системи менеджменту та обліку послуг автосервісу, що забезпечить автоматизацію обліку, диспетчеризації, розподілу завдань та інтеграцію інформаційних потоків між усіма учасниками процесу. Розроблена система дозволить зменшити кількість помилок, пришвидшити обробку замовлень та покращить якість обслуговування клієнтів.

4. Вихідні дані для проведення НДР

1. Колодій В.І., Кісіль М.В. Інформаційні технології в управлінні підприємством. – Київ: КНЕУ, 2021. – 184 с.
2. Smith J. Automotive Service Management: Principles and Practice. – New York: McGraw-Hill Education, 2020. – 240 p.
3. Кравченко С.І. Цифровізація автосервісних підприємств: проблеми і перспективи. – Харків: ХНАДУ, 2021. – 152 с.
4. Adams C. System Integration in Auto Service Management. – Cambridge University Press, 2021. – 195 p.

5. Технічні вимоги

– тип програми – десктопний застосунок. Застосунок працює як окремий

клієнтський застосунок, що встановлюється та запускається на персональному комп’ютері користувача, не потребуючи веб-браузера.

– модель розробки – інкрементна модель. Проєкт реалізується поетапно з послідовним додаванням функціональних модулів, що дозволяє здійснювати поступову інтеграцію, тестування та вдосконалення системи.

– засоби організації даних програми – JavaFX для побудови графічного інтерфейсу користувача та СУБД MySQL для збереження даних. Використання JavaFX забезпечує сучасний, кросплатформний GUI з підтримкою MVC-патерну, а MySQL дозволяє реалізувати надійне централізоване зберігання та обробку інформації про клієнтів, послуги, замовлення тощо.

– середовище розробки – IntelliJ IDEA (або інше середовище з підтримкою JavaFX і Maven/Gradle). Обране середовище забезпечує інтеграцію з JavaFX, MySQL-конектором та зручну роботу з репозиторіями, тестуванням і деплоєм.

– мова програмування – Java. Мова є основною для JavaFX, має високу продуктивність, багату стандартну бібліотеку та підтримку спільноти.

– операційна система – Windows. Основне середовище розгортання та тестування застосунку – ОС Windows, що зумовлено її поширеністю серед користувачів автосервісів.

6. Конструктивні вимоги.

Інтерфейс програми повинен відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до бакалаврської кваліфікаційної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

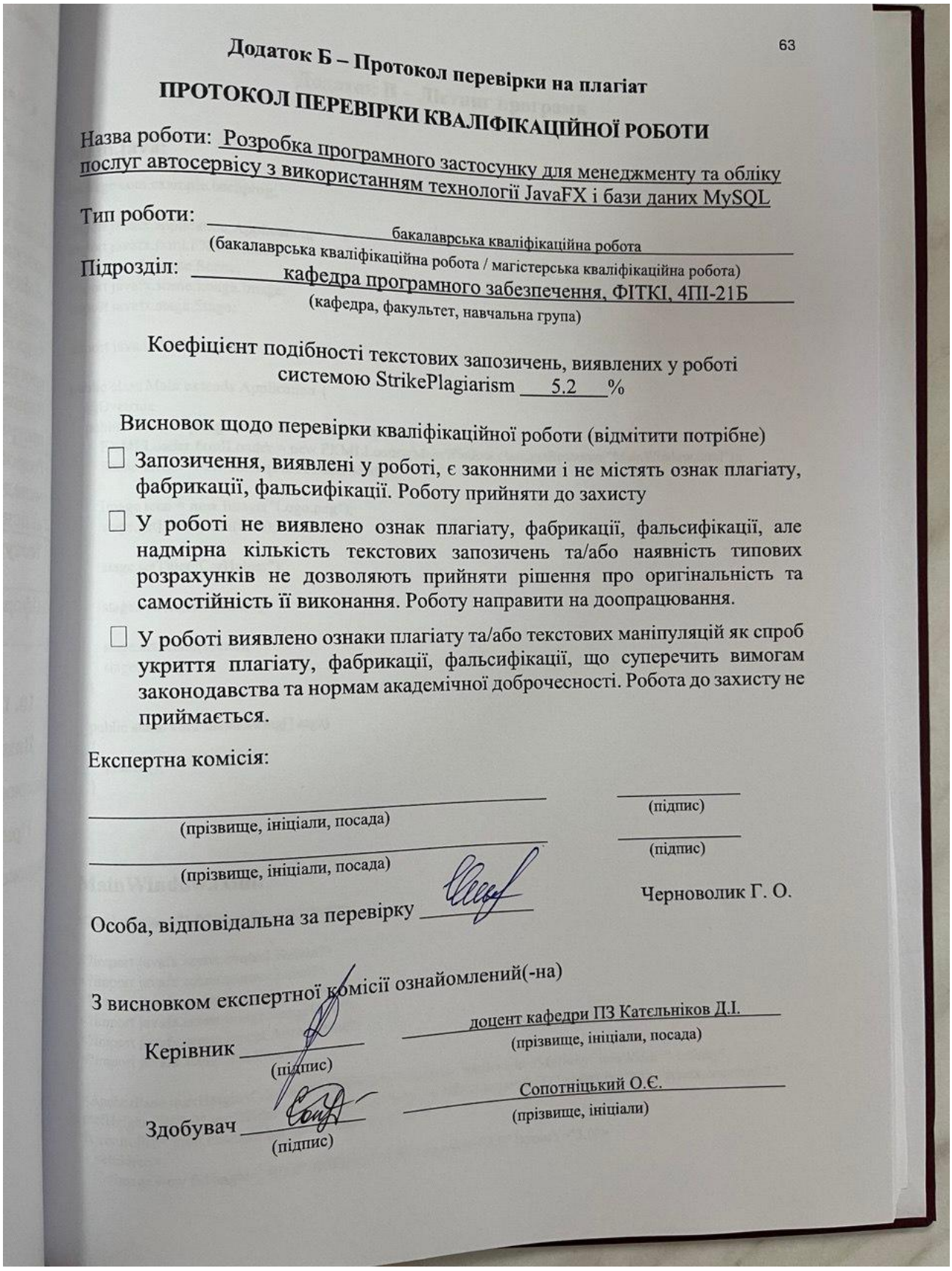
№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз розвитку технологій розробки програмних застосунків для менеджменту та обліку послуг автосервісу та постановка задач дослідження	25.03.25 - 08.04.25
2	Розробка архітектури та алгоритмів роботи програмного застосунку	09.04.25 - 20.04.25
3	Варіантний аналіз та вибір мови програмування розробки	21.04.25 - 23.04.25
4	Розробка програмних модулів для реалізації менеджменту програмного застосунку та цілісного застосунку	24.04.25 - 20.05.25
5	Тестування застосунку	21.05.25 - 25.05.25
6	Оформлення матеріалів до захисту БКР	26.05.25 - 30.05.25

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки на плагіат



Додаток Б – Протокол перевірки на плагіат

63

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного застосунку для менеджменту та обліку послуг автосервісу з використанням технології JavaFX і бази даних MySQL

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: кафедра програмного забезпечення, ФІТКІ, 4ПІ-21Б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 5.2 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☐ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) доцент кафедри ПЗ Катєльніков Д.І.
(прізвище, ініціали, посада)

Здобувач (підпис) Сопотніцький О.Є.
(прізвище, ініціали)

Додаток В – Лістинг програми

Main.java:

```
package com.example.bachprog;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;

import java.io.IOException;

public class Main extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(MainWindow.class.getResource("MainWindow.fxml"));
        Scene scene = new Scene(fxmlLoader.load());

        Image icon = new Image("Logo.png");
        stage.getIcons().add(icon);

        stage.setTitle("CarHelper");

        stage.setResizable(false);

        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args)
    {
        launch();
    }
}
```

MainWindow.fxml:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="407.0" prefWidth="404.0" xmlns="http://javafx.com/javafx/22" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.example.bachprog.MainWindow">
  <children>
    <ImageView fitHeight="402.0" fitWidth="399.0" layoutX="3.0" layoutY="3.0">
      <image>
        <Image url="@../../../Logo.png" />
```

```
        </image>
      </ImageView>
      <Label alignment="CENTER" layoutX="148.0" layoutY="27.0" prefHeight="38.0" prefWidth="107.0"
text="ALEXPLUS" textAlignment="CENTER" textFill="WHITE">
        <font>
          <Font name="System Bold" size="20.0" />
        </font>
      </Label>
      <Button fx:id="clientButton" layoutX="41.0" layoutY="270.0" mnemonicParsing="false"
onAction="#openLogInClientWindow" text="Client" />
      <Button fx:id="mechanicButton" layoutX="314.0" layoutY="270.0" mnemonicParsing="false"
onAction="#openLogInMechanicWindow" text="Mechanic" />
      <Label layoutX="14.0" layoutY="377.0" prefHeight="16.0" prefWidth="282.0" text="Developed by Oleksii
Sopotnitskyi, student VNTU 1PI-21B" textAlignment="CENTER" textFill="WHITE">
        <font>
          <Font size="11.0" />
        </font>
      </Label>
      <Button fx:id="InfoButton" layoutX="14.0" layoutY="14.0" mnemonicParsing="false"
onAction="#openInfoWindow" text="Info" />
      <Button fx:id="accountantButton" layoutX="164.0" layoutY="323.0" mnemonicParsing="false"
onAction="#openLogInAccountantWindow" text="Accountant" />
      <Button fx:id="disaptchButton" layoutX="329.0" layoutY="14.0" mnemonicParsing="false"
onAction="#openLogInDispatchWindow" text="Dispatch" />
    </children>
</AnchorPane>
```

DataBase.java:

```
package com.example.bachprog;

import java.sql.*;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class DataBase
{
    public static void dataBase()
    {
        try
        {
            Connection connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/login_schema",
                    "root",
                    "password123");

            System.out.println("Database has been successfully connected!");

            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery("SELECT * FROM CLIENTS");

            while(resultSet.next())
            {
                System.out.print(resultSet.getString("idclients"));
                System.out.print(" ");
                System.out.print(resultSet.getString("clientName"));
                System.out.print(" ");
```

```
            System.out.println(resultSet.getString("clientPassword"));
        }
    }
        catch (SQLException e)
    {
        e.printStackTrace();
    }
}

public static boolean dispatcherLogInDataBaseCheckUp(String dispatcherLogin, String dispatcherPassword) {
    boolean dispatcherExists = false;

    try {
        Connection connection = DriverManager.getConnection(
                "jdbc:mysql://127.0.0.1:3306/login_schema",
                "root", "password123"
        );

        String query = "SELECT * FROM DISPATCHER WHERE dispatcherLogin = ? AND dispatcherPassword = ?";
        PreparedStatement preparedStatement = connection.prepareStatement(query);
        preparedStatement.setString(1, dispatcherLogin);
        preparedStatement.setString(2, dispatcherPassword);

        ResultSet resultSet = preparedStatement.executeQuery();

        if (resultSet.next()) {
            dispatcherExists = true;
        }

        resultSet.close();
        preparedStatement.close();
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return dispatcherExists;
}

public static boolean clientLogInDataBaseCheckUp(String clientNameLabel, String clientPasswordLabel) {
    boolean clientExists = false;

    try
    {
        Connection connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/login_schema",
                "root", "password123");

        String query = "SELECT * FROM CLIENTS WHERE clientName = ? AND clientPassword = ?";
        PreparedStatement preparedStatement = connection.prepareStatement(query);
        preparedStatement.setString(1, clientNameLabel);
        preparedStatement.setString(2, clientPasswordLabel);


        ResultSet resultSet = preparedStatement.executeQuery();


        if (resultSet.next()) {
            clientExists = true;
```

```
        }


        resultSet.close();
        preparedStatement.close();
        connection.close();
      } catch (SQLException e) {
        e.printStackTrace();
      }

      return clientExists;
    }

    public static boolean mechanicLogInDataBaseCheckUp(String mechanicNameLabel, String
mechanicPasswordLabel) {
      boolean clientExists = false;

      try
      {
        Connection connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/login_schema",
            "root", "password123");

        String query = "SELECT * FROM MECHANICS WHERE mechanicName = ? AND mechanicPassword = ?";
        PreparedStatement preparedStatement = connection.prepareStatement(query);
        preparedStatement.setString(1, mechanicNameLabel);
        preparedStatement.setString(2, mechanicPasswordLabel);


        ResultSet resultSet = preparedStatement.executeQuery();


        if (resultSet.next()) {
          clientExists = true;
        }


        resultSet.close();
        preparedStatement.close();
        connection.close();
      } catch (SQLException e) {
        e.printStackTrace();
      }

      return clientExists;
    }

    public static String[] getCarsByClientName(String clientName) {
      String[] cars = {};
      try {
        Connection connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/login_schema",
            "root", "password123");

        String query = "SELECT carMake, carModel FROM repairements WHERE clientName = ?";
        PreparedStatement preparedStatement = connection.prepareStatement(query);
        preparedStatement.setString(1, clientName);

        ResultSet resultSet = preparedStatement.executeQuery();
```

```
        List<String> carList = new ArrayList<>();

        while (resultSet.next()) {
            String carMake = resultSet.getString("carMake");
            String carModel = resultSet.getString("carModel");
            carList.add(carMake + " " + carModel);
        }

        // list to array
        cars = carList.toArray(new String[0]);

        resultSet.close();
        preparedStatement.close();
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return cars;
}

public static String getCarImageByValue(String carValue) {
    String carImage = null;

    try {
        Connection connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/login_schema",
                "root", "password123");

        String[] carParts = carValue.split(" ");
        if (carParts.length == 2) {
            String carMake = carParts[0];
            String carModel = carParts[1];

            String query = "SELECT picture FROM repairements WHERE carMake = ? AND carModel = ? LIMIT 1";
            PreparedStatement preparedStatement = connection.prepareStatement(query);
            preparedStatement.setString(1, carMake);
            preparedStatement.setString(2, carModel);

            ResultSet resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {
                carImage = resultSet.getString("picture");
            }

            resultSet.close();
            preparedStatement.close();
            connection.close();
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }

    return carImage;
}

public static String[] getRepairementsByClientAndCar(String clientName, String carValue) {
    String[] repairements = {};
```

```
        try {
            Connection connection = DriverManager.getConnection(
                    "jdbc:mysql://127.0.0.1:3306/login_schema",
                    "root",
                    "password123");

            String query = "SELECT orderDate, typeOfRepairement, mileage " +
                    "FROM repairements WHERE clientName = ? " +
                    "AND CONCAT(carMake, ' ', carModel) = ?";
            PreparedStatement preparedStatement = connection.prepareStatement(query);
            preparedStatement.setString(1, clientName);
            preparedStatement.setString(2, carValue);

            ResultSet resultSet = preparedStatement.executeQuery();

            List<String> repairementList = new ArrayList<>();

            while (resultSet.next()) {
                String orderDate = resultSet.getString("orderDate");
                String typeOfRepairement = resultSet.getString("typeOfRepairement");
                double mileage = resultSet.getDouble("mileage"); // Getting mileage
                repairementList.add(orderDate + " - " + typeOfRepairement + " - " + mileage + " km");
            }

            // Convert list to array
            repairements = repairementList.toArray(new String[0]);

            resultSet.close();
            preparedStatement.close();
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return repairements;
    }

    public static String[] getHistoryForCient(String clientName, String carValue) {
        String[] repairementsDetails = {};
        try {
            String[] carValueParts = carValue.split(" - ");
            if (carValueParts.length != 3) {
                throw new IllegalArgumentException("carValue must include 'orderDate', 'typeOfRepairement', and 'mileage',
separated by ' - '.");
            }
            String orderDate = carValueParts[0];
            String typeOfRepairement = carValueParts[1];
            double mileage = Double.parseDouble(carValueParts[2].replace(" km", "").trim());

            Connection connection = DriverManager.getConnection(
                    "jdbc:mysql://127.0.0.1:3306/login_schema",
                    "root",
                    "password123");

            String query = "SELECT orderDate, typeOfRepairement, mechanicName, mileage, cost, status " +
                    "FROM repairements WHERE clientName = ? " +
                    "AND orderDate = ? AND typeOfRepairement = ? AND mileage = ?";
            PreparedStatement preparedStatement = connection.prepareStatement(query);
```

```
        preparedStatement.setString(1, clientName);
        preparedStatement.setString(2, orderDate);
        preparedStatement.setString(3, typeOfRepairement);
        preparedStatement.setDouble(4, mileage);

        ResultSet resultSet = preparedStatement.executeQuery();

        List<String> repairementList = new ArrayList<>();

        while (resultSet.next()) {
            String resultOrderDate = resultSet.getString("orderDate");
            String resultTypeOfRepairement = resultSet.getString("typeOfRepairement");
            String mechanicName = resultSet.getString("mechanicName");
            double resultMileage = resultSet.getDouble("mileage");
            double cost = resultSet.getDouble("cost");
            String status = resultSet.getString("status");

            repairementList.add(resultOrderDate + " - " + resultTypeOfRepairement + " - " + mechanicName +
                    " - " + resultMileage + " km - $" + cost + " - " + status);
        }

        repairementsDetails = repairementList.toArray(new String[0]);

        resultSet.close();
        preparedStatement.close();
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (IllegalArgumentException e) {
        System.out.println(e.getMessage());
    }

    return repairementsDetails;
}

public static String[] getRequestsByClient(String clientName) {
    String[] requests = {};
    try {
        Connection connection = DriverManager.getConnection(
                "jdbc:mysql://127.0.0.1:3306/login_schema",
                "root",
                "password123");

        String query = "SELECT requestedDate, carMakeRequest, carModelRequest, requestedTypeOfService " +
                "FROM order_request " +
                "WHERE clientName = ?";

        PreparedStatement preparedStatement = connection.prepareStatement(query);
        preparedStatement.setString(1, clientName);

        ResultSet resultSet = preparedStatement.executeQuery();

        List<String> requestList = new ArrayList<>();

        while (resultSet.next()) {
            String requestedDate = resultSet.getString("requestedDate");
            String make = resultSet.getString("carMakeRequest");
            String model = resultSet.getString("carModelRequest");
```

```
            String service = resultSet.getString("requestedTypeOfService");

            requestList.add(requestedDate + " - " + make + " " + model + " - " + service);
        }

        requests = requestList.toArray(new String[0]);

        resultSet.close();
        preparedStatement.close();
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return requests;
}

public static boolean insertOrderRequest(String clientName,
                                         String carMake,
                                         String carModel,
                                         LocalDate requestedDate,
                                         String requestedServiceType) {
    try {
        Connection connection = DriverManager.getConnection(
                "jdbc:mysql://127.0.0.1:3306/login_schema",
                "root",
                "password123");

        String query = "INSERT INTO order_request " +
                "(clientName, carMakeRequest, carModelRequest, requestedDate, requestedTypeOfService) " +
                "VALUES (?, ?, ?, ?, ?)";

        PreparedStatement preparedStatement = connection.prepareStatement(query);
        preparedStatement.setString(1, clientName);
        preparedStatement.setString(2, carMake);
        preparedStatement.setString(3, carModel);
        preparedStatement.setDate(4, Date.valueOf(requestedDate));
        preparedStatement.setString(5, requestedServiceType);

        int rowsInserted = preparedStatement.executeUpdate();

        preparedStatement.close();
        connection.close();

        return rowsInserted > 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

public static boolean deleteOrderRequest(String displayValue, String clientName) {
    try {
        String[] parts = displayValue.split(" - ");
        if (parts.length != 3) {
            System.out.println("Невірний формат рядка.");
            return false;
        }
```

```
            String requestedDate = parts[0];
            String[] carParts = parts[1].split(" ", 2);
            if (carParts.length != 2) {
                System.out.println("Невірний формат марки/моделі.");
                return false;
            }

            String make = carParts[0];
            String model = carParts[1];
            String service = parts[2];

            Connection connection = DriverManager.getConnection(
                    "jdbc:mysql://127.0.0.1:3306/login_schema",
                    "root",
                    "password123");

            String sql = "DELETE FROM order_request " +
                    "WHERE clientName = ? " +
                    "AND requestedDate = ? " +
                    "AND carMakeRequest = ? " +
                    "AND carModelRequest = ? " +
                    "AND requestedTypeOfService = ?";

            PreparedStatement stmt = connection.prepareStatement(sql);
            stmt.setString(1, clientName);
            stmt.setString(2, requestedDate);
            stmt.setString(3, make);
            stmt.setString(4, model);
            stmt.setString(5, service);

            int rowsDeleted = stmt.executeUpdate();

            stmt.close();
            connection.close();

            return rowsDeleted > 0;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    public static List<String> getOrderRequestDisplayStrings() {
        List<String> displayList = new ArrayList<>();

        try {
            Connection connection = DriverManager.getConnection(
                    "jdbc:mysql://127.0.0.1:3306/login_schema",
                    "root",
                    "password123");

            String sql = "SELECT requestedDate, clientName, carMakeRequest, carModelRequest,
requestedTypeOfService FROM order_request";
            PreparedStatement stmt = connection.prepareStatement(sql);
            ResultSet rs = stmt.executeQuery();

            while (rs.next()) {
```

```
            String requestedDate = rs.getString("requestedDate");
            String clientName = rs.getString("clientName");
            String carMake = rs.getString("carMakeRequest");
            String carModel = rs.getString("carModelRequest");
            String service = rs.getString("requestedTypeOfService");

            String display = requestedDate + " - " + clientName + " - " + carMake + " - " + carModel + " - " + service;
            displayList.add(display);
        }

        rs.close();
        stmt.close();
        connection.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

    return displayList;
}

public static String[] getClientContactInfo(String clientName) {
    String[] contactInfo = new String[3]; // [0] = phone, [1] = email, [2] = image

    try {
        Connection connection = DriverManager.getConnection(
                "jdbc:mysql://127.0.0.1:3306/login_schema",
                "root",
                "password123");

        String sql = "SELECT clientPhoneNumber, clientEmail, imageClient FROM clients WHERE clientName = ?";
        PreparedStatement stmt = connection.prepareStatement(sql);
        stmt.setString(1, clientName);

        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            contactInfo[0] = rs.getString("clientPhoneNumber");
            contactInfo[1] = rs.getString("clientEmail");
            contactInfo[2] = rs.getString("imageClient");
        }

        rs.close();
        stmt.close();
        connection.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

    return contactInfo;
}

public static List<String> getMechanicNames() {
    List<String> mechanicNames = new ArrayList<>();

    try {
        Connection connection = DriverManager.getConnection(
                "jdbc:mysql://127.0.0.1:3306/login_schema",
                "root",
```

```
            "password123");

        String sql = "SELECT mechanicName FROM mechanics";
        PreparedStatement stmt = connection.prepareStatement(sql);

        ResultSet rs = stmt.executeQuery();

        while (rs.next()) {
            mechanicNames.add(rs.getString("mechanicName"));
        }

        rs.close();
        stmt.close();
        connection.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

    return mechanicNames;
}


public static List<String[]> getRepairementsForMechanicAndDate(String mechanicName, LocalDate orderDate) {
    List<String[]> repairements = new ArrayList<>();

    try {
        Connection connection = DriverManager.getConnection(
                "jdbc:mysql://127.0.0.1:3306/login_schema",
                "root",
                "password123");

        String sql = "SELECT orderDate, startTime, endTime, carMake, carModel, mileage, " +
                "typeOfRepairement, cost, mechanicInstructions, status " +
                "FROM repairements " +
                "WHERE mechanicName = ? AND orderDate = ?";

        PreparedStatement stmt = connection.prepareStatement(sql);
        stmt.setString(1, mechanicName);
        stmt.setDate(2, Date.valueOf(orderDate)); // LocalDate into SQL Date

        ResultSet rs = stmt.executeQuery();

        while (rs.next()) {
            String[] row = new String[10];
            row[0] = rs.getString("orderDate");
            row[1] = rs.getString("startTime");
            row[2] = rs.getString("endTime");
            row[3] = rs.getString("carMake");
            row[4] = rs.getString("carModel");
            row[5] = rs.getString("mileage");
            row[6] = rs.getString("typeOfRepairement");
            row[7] = rs.getString("cost");
            row[8] = rs.getString("mechanicInstructions");
            row[9] = rs.getString("status");

            repairements.add(row);
        }
```

```
            rs.close();
            stmt.close();
            connection.close();

        } catch (Exception e) {
            e.printStackTrace();
        }

        return repairements;
    }

    public static boolean insertRepairement(
            String clientName,
            String mechanicName,
            String carMake,
            String carModel,
            LocalDate orderDate,
            String typeOfRepairement,
            double cost,
            String mechanicInstructions,
            String mileage,
            String status,
            String startTime,
            String endTime) {

        boolean indicator;
        String sql = "INSERT INTO repairements (clientName, mechanicName, carMake, carModel, orderDate, " +
                "typeOfRepairement, cost, mechanicInstructions, mileage, status, startTime, endTime) " +
                "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

        try (Connection connection = DriverManager.getConnection(
                "jdbc:mysql://127.0.0.1:3306/login_schema",
                "root",
                "password123");
             PreparedStatement stmt = connection.prepareStatement(sql)) {

            stmt.setString(1, clientName);
            stmt.setString(2, mechanicName);
            stmt.setString(3, carMake);
            stmt.setString(4, carModel);
            stmt.setDate(5, java.sql.Date.valueOf(orderDate));
            stmt.setString(6, typeOfRepairement);
            stmt.setDouble(7, cost);
            stmt.setString(8, mechanicInstructions);
            stmt.setString(9, mileage);
            stmt.setString(10, status);
            stmt.setTime(11, java.sql.Time.valueOf(startTime));
            stmt.setTime(12, java.sql.Time.valueOf(endTime));

            stmt.executeUpdate();
            indicator = true;

        } catch (Exception e) {
            indicator = false;
            e.printStackTrace();
        }

        return indicator;
```

```
    }

}
```

DispatchWindow.java:

```
package com.example.bachprog;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;

import java.io.IOException;
import java.util.List;

public class DispatchWindow
{
    @FXML
    private Button openOrderCreationWindow;

    @FXML
    private void openOrderCreationWindow() throws IOException
    {
        OrderCreationWindow.openWindow("OrderCreationWindow.fxml", "Create an order");
    }

    @FXML
    private ListView<String> requests;

    @FXML
    private ListView<String> mechanicList;

    @FXML
    private Label clientNameField;

    @FXML
    private Label clientPhNumberField;

    @FXML
    private Label clientEmailAddressField;

    @FXML
    private ImageView clientImage;

    public static String requestOrderInfo;
    public static String mechanicName;

    @FXML
```

```
    public void initializeTheListOfRequests() {
        List<String> orderRequests = DataBase.getOrderRequestDisplayStrings();
        ObservableList<String> observableList = FXCollections.observableArrayList(orderRequests);
        requests.setItems(observableList);

        requests.getSelectionModel().selectedItemProperty().addListener((obs, oldVal, newVal) -> {
            requestOrderInfo = newVal;
            if (newVal != null) {
                String[] parts = newVal.split(" - ");

                String [] clientInfo = DataBase.getClientContactInfo(parts[1]); // 0 - phNum; 1 - email; 2 - image

                clientNameField.setText(parts[1]);
                clientPhNumberField.setText(clientInfo[0]);
                clientEmailAddressField.setText(clientInfo[1]);

                if (clientImage != null) {
                    Image image = new Image(clientInfo[2]);
                    clientImage.setImage(image);
                } else {
                    System.err.println("clientImage is null!");
                }

            }
        });

        List<String> mechanicNames = DataBase.getMechanicNames();
        ObservableList<String> obsNames = FXCollections.observableArrayList(mechanicNames);
        mechanicList.setItems(obsNames);

        mechanicList.getSelectionModel().selectedItemProperty().addListener((obs1, oldVal1, selectedMechanicName) ->
{
            if (selectedMechanicName != null)
            {
                mechanicName = selectedMechanicName;
            }
        });


    }

    @FXML
    private void selectedObjectFromList() {
        String selected = requests.getSelectionModel().getSelectedItem();
        if (selected != null) {
            System.out.println("Selected: " + selected);
        } else {
            System.out.println("Nothing has been selected.");
        }
    }

    public static void openWindow(String fxmlFile, String title) throws IOException {
        FXMLLoader loader = new FXMLLoader(DispatchWindow.class.getResource("/com/example/bachprog/" +
fxmlFile));
        Parent root = loader.load();

        DispatchWindow controller = loader.getController();
```

```
        controller.initializeTheListOfRequests();

        Stage stage = new Stage();
        Scene scene = new Scene(root);

        Image icon = new Image("Logo.png");
        stage.getIcons().add(icon);

        stage.setResizable(false);
        stage.setScene(scene);
        stage.setTitle(title);
        stage.show();
    }

}
```

DispatchWindow.fxml:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ListView?>
<?import javafx.scene.control.Menu?>
<?import javafx.scene.control.MenuBar?>
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>

<AnchorPane prefHeight="378.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/22"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.example.bachprog.DispatchWindow">
  <children>
    <ListView fx:id="requests" layoutX="3.0" layoutY="48.0" onEditStart="#selectedObjectFromList"
prefHeight="152.0" prefWidth="395.0" />
    <MenuBar prefHeight="26.0" prefWidth="600.0">
      <menus>
        <Menu mnemonicParsing="false" text="Privacy">
          <items>
            <MenuItem mnemonicParsing="false" text="Change the password" />
          </items>
        </Menu>
        <Menu mnemonicParsing="false" text="?__?">
          <items>
            <MenuItem mnemonicParsing="false" text="Delete" />
          </items>
        </Menu>
        <Menu mnemonicParsing="false" text="Help">
          <items>
            <MenuItem mnemonicParsing="false" text="About" />
          </items>
        </Menu>
      </menus>
    </MenuBar>
    <Label layoutX="5.0" layoutY="29.0" prefHeight="18.0" prefWidth="75.0" text="Requests:">
      <font>
```

```
            <Font name="System Bold Italic" size="14.0" />
          </font>
        </Label>
        <Label alignment="CENTER" layoutX="454.0" layoutY="29.0" prefHeight="18.0" prefWidth="75.0"
text="Client" textAlignment="CENTER">
          <font>
            <Font name="System Bold Italic" size="14.0" />
          </font>
        </Label>
        <Label layoutX="416.0" layoutY="204.0" prefHeight="20.0" prefWidth="46.0" text="Client: "
textAlignment="CENTER">
          <font>
            <Font name="System Bold Italic" size="14.0" />
          </font>
        </Label>
        <Label layoutX="416.0" layoutY="245.0" prefHeight="20.0" prefWidth="103.0" text="Phone number:"
textAlignment="CENTER">
          <font>
            <Font name="System Bold Italic" size="14.0" />
          </font>
        </Label>
        <Label layoutX="416.0" layoutY="292.0" prefHeight="20.0" prefWidth="103.0" text="Email address:"
textAlignment="CENTER">
          <font>
            <Font name="System Bold Italic" size="14.0" />
          </font>
        </Label>
        <Label fx:id="clientNameField" alignment="CENTER_RIGHT" layoutX="415.0" layoutY="224.0"
prefHeight="20.0" prefWidth="166.0" textAlignment="CENTER" textFill="#bf8686">
          <font>
            <Font name="System Italic" size="14.0" />
          </font>
        </Label>
        <Label fx:id="clientPhNumberField" alignment="CENTER_RIGHT" layoutX="415.0" layoutY="270.0"
prefHeight="20.0" prefWidth="166.0" textAlignment="CENTER" textFill="#bf8686">
          <font>
            <Font name="System Italic" size="14.0" />
          </font>
        </Label>
        <Label fx:id="clientEmailAddressField" alignment="CENTER_RIGHT" layoutX="418.0" layoutY="314.0"
prefHeight="20.0" prefWidth="166.0" textAlignment="CENTER" textFill="#bf8686">
          <font>
            <Font name="System Italic" size="14.0" />
          </font>
        </Label>
        <ImageView fx:id="clientImage" fitHeight="152.0" fitWidth="185.0" layoutX="416.0" layoutY="49.0"
pickOnBounds="true" preserveRatio="true">
          <image>
            <Image url="@../../../avatar_icon.png" />
          </image>
        </ImageView>
        <Label layoutX="5.0" layoutY="204.0" prefHeight="20.0" prefWidth="75.0" text="Mechanics:">
          <font>
            <Font name="System Bold Italic" size="14.0" />
          </font>
        </Label>
        <ListView fx:id="mechanicList" layoutX="5.0" layoutY="224.0" prefHeight="116.0" prefWidth="395.0" />
        <Button fx:id="openOrderCreationWindow" layoutX="257.0" layoutY="347.0" mnemonicParsing="false"
```

```
onAction="#openOrderCreationWindow" text="Create a task" />
  </children>
</AnchorPane>
```

OrderCreationWindow.java:

```
package com.example.bachprog;

import javafx.collections.FXCollections;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.stage.Stage;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class OrderCreationWindow
{
    @FXML
    private DatePicker orderDate;

    @FXML
    private TextField carMake;

    @FXML
    private TextField carModel;

    @FXML
    private TextField repairement;

    @FXML
    private TextField cost;

    @FXML
    private TextField startTime;

    @FXML
    private TextField endTime;

    @FXML
    private TextField mechanic;

    @FXML
    private TextField client;

    @FXML
    private TextField mileage;

    @FXML
    private TextArea instruction;

    @FXML
```

```
    private ChoiceBox<String> orderStatus;

    @FXML
    private ListView<String> currentOrderList;
    @FXML
    public void initializeRequestedServiceList()
    {
        orderStatus.getItems().addAll("IN PROGRESS", "DONE");
        orderStatus.setValue("IN PROGRESS");
    }

    @FXML
    public void initializeOrderCreationWindowFields()
    {
        String info = DispatchWindow.requestOrderInfo;

        String[] requestInfo = info.split(" - "); // 0 - date; 1 - client; 2 - Make; 3 - Model; 4 - ServiceType

        client.setText(requestInfo[1]);
        carMake.setText(requestInfo[2]);
        carModel.setText(requestInfo[3]);
        repairement.setText(requestInfo[4]);

        mechanic.setText(DispatchWindow.mechanicName);
    }

    @FXML
    public void initializeOrderList()
    {
        orderDate.valueProperty().addListener((observable, oldValue, selectedDate) -> {
            List<String[]> repairements =
DataBase.getRepairementsForMechanicAndDate(DispatchWindow.mechanicName, selectedDate);
            List<String> displayList = new ArrayList<>();

            if (repairements.isEmpty()) {
                Alert alert = new Alert(Alert.AlertType.INFORMATION);
                alert.setTitle("No Data");
                alert.setHeaderText(null);
                alert.setContentText("Nothing was set up on that date.");
                alert.showAndWait();

                currentOrderList.getItems().clear();
                return;
            }

            for (String[] row : repairements) {
                String line = row[1] + " - " + row[2] + " | " + row[3] + " " + row[4] +
                        " | " + row[6] + " | Status: " + row[9];
                displayList.add(line);
            }

            currentOrderList.setItems(FXCollections.observableArrayList(displayList));

        });
    }

    @FXML
    private void clearAllFields()
```

```
    {
        //mechanic.setText(null);
        //orderDate.setValue(null);
        client.setText(null);
        carMake.setText(null);
        carModel.setText(null);
        repairement.setText(null);
        cost.setText(null);
        startTime.setText(null);
        endTime.setText(null);
        mileage.setText(null);
        instruction.setText(null);
    }
    @FXML
    private void addAnOrder()
    {
        if ((client.getText() == null) ||(orderDate.getValue() == null) || (carMake.getText() == null)
        || (carModel.getText() == null) || (repairement.getText() == null) || (cost.getText() == null)
        || (startTime.getText() == null) || (endTime.getText() == null) || (mechanic.getText() == null)
        || (mileage.getText() == null) || (instruction.getText() == null))
        {
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Error");
            alert.setHeaderText(null);
            alert.setContentText("Please fill all the fields.");
            alert.showAndWait();
        }
        else
        {
            char costValue = (char) Integer.parseInt(cost.getText());
            //char mileageValue = (char) Integer.parseInt(mileage.getText());
            if (DataBase.insertRepairement(client.getText(), mechanic.getText(),carMake.getText(),carModel.getText(),
                orderDate.getValue(),repairement.getText(),costValue,instruction.getText(),mileage.getText(),
                orderStatus.getValue(), startTime.getText(), endTime.getText()))
            {
                Alert alert = new Alert(Alert.AlertType.INFORMATION);
                alert.setTitle("Success");
                alert.setHeaderText(null);
                alert.setContentText("Order successfully created");
                alert.showAndWait();

                initializeOrderList();
                clearAllFields();
            }

        }
    }

    public static void openWindow(String fxmlFile, String title) throws IOException {
        FXMLLoader loader = new FXMLLoader(OrderCreationWindow.class.getResource("/com/example/bachprog/" +
fxmlFile));
        Parent root = loader.load();

        OrderCreationWindow controller = loader.getController();
        controller.initializeRequestedServiceList();
        controller.initializeOrderCreationWindowFields();
        controller.initializeOrderList();
```

```
        Stage stage = new Stage();
        Scene scene = new Scene(root);

        Image icon = new Image("Logo.png");
        stage.getIcons().add(icon);

        stage.setResizable(false);
        stage.setScene(scene);
        stage.setTitle(title);
        stage.show();
    }

}
```

OrderCreationWindow.FXML:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.ChoiceBox?>
<?import javafx.scene.control.DatePicker?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ListView?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>


<AnchorPane prefHeight="469.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/22"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.example.bachprog.OrderCreationWindow">
  <children>
    <ListView fx:id="currentOrderList" layoutX="3.0" layoutY="249.0" prefHeight="176.0" prefWidth="594.0" />
    <Label layoutX="3.0" layoutY="5.0" prefHeight="18.0" prefWidth="43.0" text="Date:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <DatePicker fx:id="orderDate" layoutX="54.0" layoutY="3.0" prefHeight="26.0" prefWidth="206.0" />
    <Label layoutX="343.0" layoutY="6.0" prefHeight="20.0" prefWidth="73.0" text="Mechanic:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <TextField fx:id="mechanic" layoutX="428.0" layoutY="3.0" prefHeight="26.0" prefWidth="166.0" />
    <Label layoutX="5.0" layoutY="228.0" prefHeight="20.0" prefWidth="73.0" text="Orders:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <Label layoutX="2.0" layoutY="38.0" prefHeight="20.0" prefWidth="73.0" text="Car make:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <TextField fx:id="carMake" layoutX="94.0" layoutY="32.0" prefHeight="26.0" prefWidth="166.0" />
```

```
    <Label layoutX="2.0" layoutY="69.0" prefHeight="20.0" prefWidth="73.0" text="Car model:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <TextField fx:id="carModel" layoutX="94.0" layoutY="63.0" prefHeight="26.0" prefWidth="166.0" />
    <Label layoutX="2.0" layoutY="99.0" prefHeight="20.0" prefWidth="94.0" text="Repairement:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <TextField fx:id="repairement" layoutX="94.0" layoutY="94.0" prefHeight="26.0" prefWidth="166.0" />
    <Label layoutX="2.0" layoutY="128.0" prefHeight="20.0" prefWidth="94.0" text="Cost:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <TextField fx:id="cost" layoutX="94.0" layoutY="124.0" prefHeight="26.0" prefWidth="166.0" />
    <Label layoutX="2.0" layoutY="157.0" prefHeight="20.0" prefWidth="94.0" text="Start time:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <TextField fx:id="startTime" layoutX="94.0" layoutY="153.0" prefHeight="26.0" prefWidth="166.0"
promptText="00:00:00" />
    <Label layoutX="2.0" layoutY="187.0" prefHeight="20.0" prefWidth="94.0" text="End time:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <TextField fx:id="endTime" layoutX="94.0" layoutY="183.0" prefHeight="26.0" prefWidth="166.0"
promptText="00:00:00" />
    <Label layoutX="341.0" layoutY="69.0" prefHeight="20.0" prefWidth="73.0" text="Mileage:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <TextField fx:id="mileage" layoutX="428.0" layoutY="63.0" prefHeight="26.0" prefWidth="166.0"
promptText="123.456" />
    <Label layoutX="340.0" layoutY="96.0" prefHeight="20.0" prefWidth="81.0" text="Instruction:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <TextArea fx:id="instruction" layoutX="428.0" layoutY="93.0" prefHeight="122.0" prefWidth="166.0" />
    <Label layoutX="343.0" layoutY="35.0" prefHeight="20.0" prefWidth="73.0" text="Client:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <TextField fx:id="client" layoutX="428.0" layoutY="32.0" prefHeight="26.0" prefWidth="166.0" />
    <Button layoutX="281.0" layoutY="432.0" mnemonicParsing="false" onAction="#addAnOrder" text="Add" />
    <Label layoutX="364.0" layoutY="221.0" prefHeight="20.0" prefWidth="54.0" text="Status:">
      <font>
        <Font name="System Bold Italic" size="14.0" />
      </font>
    </Label>
    <ChoiceBox fx:id="orderStatus" layoutX="428.0" layoutY="218.0" prefHeight="26.0" prefWidth="166.0" />
```

```
  </children>
</AnchorPane>
```

LogInClientWindow.java:

```
package com.example.bachprog;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.image.Image;
import javafx.stage.Stage;

import java.io.IOException;

public class LogInClientWindow
{
   private Stage stage;
   private Scene scene;
   private Parent root;

   public static String clientName;

   @FXML
   private Label alertLabel;
   @FXML
   private TextField clientNameLabel;
   @FXML
   private PasswordField clientPasswordLabel;
   @FXML
   private Button clientLoginButton;

   @FXML
   private CheckBox showThePasswordButton;

   @FXML
   private TextField unhiddenPassword;

   @FXML
   void showThePassword(ActionEvent event)
   {
      if(showThePasswordButton.isSelected())
      {
         unhiddenPassword.setText(clientPasswordLabel.getText());
         unhiddenPassword.setVisible(true);
         clientPasswordLabel.setVisible(false);
         return;
      }
      clientPasswordLabel.setText(unhiddenPassword.getText());
      clientPasswordLabel.setVisible(true);
      unhiddenPassword.setVisible(false);
   }
```

```
public void OpenLogInClientWindow(ActionEvent event) throws IOException
{
   Parent root = FXMLLoader.load(getClass().getResource("LogInClientWindow.fxml"));
   stage = (Stage) ((Node)event.getSource()).getScene().getWindow();
   scene = new Scene(root);
   stage.setScene(scene);
   stage.show();
}

public void submit(ActionEvent event)
{
   try
   {
      System.out.println("Client name: "+ clientNameLabel.getText());
      System.out.println("Password: " + clientPasswordLabel.getText());
         if (DataBase.clientLogInDataBaseCheckUp((clientNameLabel.getText()),(clientPasswordLabel.getText())))
         {
            clientName = clientNameLabel.getText(); // 4database

            // successfully proceeded
            Alert alert = new Alert(AlertType.INFORMATION);
            alert.setTitle("Success");
            alert.setHeaderText(null);
            alert.setContentText("You've entered the correct user name or password");
            alert.showAndWait();

            clientNameLabel.clear();
            clientPasswordLabel.clear();
            unhiddenPassword.clear();

            Stage currentStage = (Stage) clientNameLabel.getScene().getWindow();
            currentStage.close();

            ClientWindow.openWindow("ClientWindow.fxml", "CarHelper - Client Window");
         }
         else
         {
            clientNameLabel.clear();
            clientPasswordLabel.clear();
            unhiddenPassword.clear();

            Alert alert = new Alert(AlertType.INFORMATION);
            alert.setTitle("Error");
            alert.setHeaderText(null);
            alert.setContentText("You've entered the wrong user name or password");
            alert.showAndWait();
         }
   }
   catch (Exception e)
   {
      System.out.println(e);
   }
}

public void OpenMainWindow(ActionEvent event) throws IOException {
   Parent root = FXMLLoader.load(getClass().getResource("MainWindow.fxml"));
   stage = (Stage) ((Node)event.getSource()).getScene().getWindow();
```

```
        scene = new Scene(root);

        Image icon = new Image("Logo.png");
        stage.getIcons().add(icon);

        stage.setTitle("CarHelper");
        stage.setResizable(false);


        stage.setScene(scene);
        stage.show();
    }

    public void OpenClientWindow() throws IOException {
        ClientWindow.openWindow("ClientWindow.fxml", "Client Window");
    }

    public static void openWindow(String fxmlFile, String title) throws IOException {
        FXMLLoader loader = new FXMLLoader(LogInClientWindow.class.getResource("/com/example/bachprog/" +
fxmlFile));
        Parent root = loader.load();

        LogInClientWindow controller = loader.getController();

        Stage stage = new Stage();
        Scene scene = new Scene(root);

        Image icon = new Image("Logo.png");
        stage.getIcons().add(icon);

        stage.setResizable(false);
        stage.setScene(scene);
        stage.setTitle(title);
        stage.show();
    }

}
```

LogInClientWindow.fxml:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.CheckBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>

<AnchorPane prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/22"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.example.bachprog.LogInClientWindow">
  <children>
    <TextField fx:id="unhiddenPassword" layoutX="198.0" layoutY="200.0" prefHeight="26.0" prefWidth="199.0"
promptText="Enter your password" />
    <Label alignment="CENTER" layoutX="240.0" layoutY="24.0" prefHeight="39.0" prefWidth="116.0" text="Dear
Client,">
```

```
        <font>
          <Font size="18.0" />
        </font></Label>
      <Button layoutX="14.0" layoutY="360.0" mnemonicParsing="false" onAction="#OpenMainWindow"
text="Home" />
      <Button fx:id="clientLogInButton" layoutX="275.0" layoutY="303.0" mnemonicParsing="false"
onAction="#submit" text="LogIn" />
      <Label alignment="CENTER" layoutX="68.0" layoutY="77.0" prefHeight="50.0" prefWidth="464.0"
text="Kindly enter your name and password to open your file" textAlignment="CENTER">
        <font>
          <Font name="System Italic" size="14.0" />
        </font>
      </Label>
      <Label fx:id="alertLabel" alignment="CENTER" layoutX="84.0" layoutY="270.0" prefHeight="18.0"
prefWidth="427.0" text="Click on LogIn button once you entered your name and password"
textAlignment="CENTER">
        <font>
          <Font name="System Bold Italic" size="12.0" />
        </font>
      </Label>
      <TextField fx:id="clientNameLabel" layoutX="198.0" layoutY="154.0" prefHeight="26.0" prefWidth="199.0"
promptText="Enter your name" />
      <PasswordField fx:id="clientPasswordLabel" layoutX="198.0" layoutY="200.0" prefHeight="26.0"
prefWidth="199.0" promptText="Enter your password" />
      <CheckBox fx:id="showThePasswordButton" layoutX="198.0" layoutY="243.0" mnemonicParsing="false"
onAction="#showThePassword" prefHeight="18.0" prefWidth="135.0" text="show the password">
        <font>
          <Font name="System Bold Italic" size="12.0" />
        </font>
      </CheckBox>
    </children>
</AnchorPane>
```

ClientWindow.java:

```
package com.example.bachprog;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.MenuItem;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;

import java.io.File;
import java.io.IOException;
import java.util.*;

public class ClientWindow {
```

```
    private String SelectedInvoicedCar;
    private String SelectedInvoiceOrder;

    @FXML
    private Button getInvoice;

    @FXML
    private void onGeneratePdfButtonClick()
    {
        Invoice.generateCustomerInvoice("invoices/invoice_001.pdf",
DataBase.getHistoryForCient(LogInClientWindow.clientName, SelectedInvoiceOrder), SelectedInvoicedCar);

        Invoice opener = new Invoice();
        opener.openPdf("invoices/invoice_001.pdf");
    }


    @FXML
    private ImageView carImage;

    @FXML
    private ImageView statusImage;

    @FXML
    private ListView<String> listOfCars;

    @FXML
    private ListView<String> listOfHistory;

    @FXML
    private Label orderDate;

    @FXML
    private Label typeOfRepairement;

    @FXML
    private Label mechanicName;

    @FXML
    private Label mileage;

    @FXML
    private Label cost;

    @FXML
    private Label status;

    @FXML
    private MenuItem aboutProg;

    @FXML
    private void handleMenuClick(ActionEvent event) {
        Object source = event.getSource();

        if (source == aboutProg) {
            try {
                Info.openInfoWindow("Info.fxml", "CarHelper - Info");
            } catch (IOException e) {
```

```
            throw new RuntimeException(e);
        }
    }
}

@FXML
private void newOrderWindowOpen()
{
    try {
        OrderRequest.openOrderRequestWindow("OrderRequest.fxml", "CarHelper - Order Request");
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}

@FXML
public void updateLabels(String[] repairDetailsArray) {
    for (String repairDetail : repairDetailsArray) {
        // Split each string in the array into parts
        String[] parts = repairDetail.split(" - ");

        // Extract individual fields from the parts
        String date = parts[0]; // Order date
        String typeofRepairement = parts[1]; // Type of repair
        String mechanicname = parts[2]; // Mechanic name
        String mileAge = parts[3]; // Mileage (e.g., "80.041 km")
        String Cost = parts[4]; // Cost (e.g., "$200.0")
        String Status = parts [5]; // StatusOfOrder

        initializeStatusImage(parts [5]);

        // Update the labels
        orderDate.setText("Date: " + date);
        typeOfRepairement.setText("Type of repairement: " + typeofRepairement);
        mechanicName.setText("Mechanic name: " + mechanicname);
        mileage.setText("Mileage: " + mileAge);
        cost.setText("Cost: " + Cost);
        status.setText("Status: " + Status);


        break;
    }
}

@FXML
public void initializeList() {
    //String[] cars = DataBase.getCarsByClientName(LogInClientWindow.clientName);

    Set<String> carSet = new
HashSet<>(Arrays.asList(DataBase.getCarsByClientName(LogInClientWindow.clientName)));

    List<String> carList = new ArrayList<>(carSet);

    listOfCars.getItems().setAll(carList);

    listOfCars.getSelectionModel().clearSelection();

    listOfCars.getSelectionModel().selectedItemProperty().addListener((observable, oldValue, newValue) -> {
```

```
        if (newValue != null) {
//            System.out.println("Selected car: " + newValue); // carValueName
//            System.out.println("Link: " + DataBase.getCarImageByValue(newValue)); //carLink
            initializeCarImage(DataBase.getCarImageByValue(newValue));
            SelectedInvoicedCar = newValue;
            // small records
            //String[] generalRepairments =
DataBase.getRepairementsByClientAndCar(LogInClientWindow.clientName, newValue);

            Set<String> generalRepairmentsSet = new
HashSet<>(Arrays.asList(DataBase.getRepairementsByClientAndCar(LogInClientWindow.clientName, newValue)));

            List<String> generalRepairments = new ArrayList<>(generalRepairmentsSet);

            listOfHistory.getItems().setAll(generalRepairments);

            listOfHistory.getSelectionModel().clearSelection();

            // full history description
            listOfHistory.getSelectionModel().selectedItemProperty().addListener((observable1, oldValue1, newValue1)
-> {
                if (newValue1 != null)
                {

                    /*
                    for (String detail : DataBase.getHistoryForCient(LogInClientWindow.clientName, newValue1)) {
                        System.out.println(detail);
                    }
                    */

                    String[] historyDetails = DataBase.getHistoryForCient(LogInClientWindow.clientName, newValue1);
                    updateLabels(historyDetails);
                    SelectedInvoiceOrder = newValue1;
                    //listOfHistory.getSelectionModel().clearSelection();
                }
            });

        }
    });

}

@FXML
public void initializeCarImage(String imageLink)
{
    Image image = new Image(imageLink);
    carImage.setImage(image);
}

@FXML
public void initializeStatusImage(String status)
{
    if (status.equals("DONE")) {
        Image image = new Image("doneIcon.png");
        statusImage.setImage(image);
    } else {
        Image image = new Image("inProgress.png");
        statusImage.setImage(image);
```

```
        }
    }

    public static void openWindow(String fxmlFile, String title) throws IOException {
        FXMLLoader loader = new FXMLLoader(ClientWindow.class.getResource("/com/example/bachprog/" +
fxmlFile));
        Parent root = loader.load();

        ClientWindow controller = loader.getController();
        controller.initializeList();

        //controller.initializeCarImage();


        Stage stage = new Stage();
        Scene scene = new Scene(root);

        Image icon = new Image("Logo.png");
        stage.getIcons().add(icon);

        stage.setResizable(false);
        stage.setScene(scene);
        stage.setTitle(title);
        stage.show();
    }
    public static void openWindow(ActionEvent event, String fxmlFile, String title) throws IOException {
        FXMLLoader loader = new FXMLLoader(ClientWindow.class.getResource("/com/example/bachprog/" +
fxmlFile));
        Parent root = loader.load();

        Stage currentStage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        currentStage.close();

        Stage stage = new Stage();
        Scene scene = new Scene(root);

        Image icon = new Image("Logo.png");
        stage.getIcons().add(icon);

        stage.setResizable(false);
        stage.setScene(scene);
        stage.setTitle(title);
        stage.show();
    }


}
```

ClientWindow.fxml:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ListView?>
<?import javafx.scene.control.Menu?>
<?import javafx.scene.control.MenuBar?>
```

```
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.effect.Blend?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>

<AnchorPane prefHeight="541.0" prefWidth="859.0" xmlns="http://javafx.com/javafx/22"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.example.bachprog.ClientWindow">
  <children>
    <ImageView fx:id="carImage" fitHeight="268.0" fitWidth="268.0" layoutX="589.0" layoutY="34.0"
pickOnBounds="true" preserveRatio="true">
      <image>
        <Image url="@../../../Logo.png" />
      </image>
    </ImageView>
    <ListView fx:id="listOfCars" layoutX="14.0" layoutY="64.0" prefHeight="200.0" prefWidth="200.0" />
    <Label alignment="CENTER" layoutX="65.0" layoutY="31.0" prefHeight="18.0" prefWidth="75.0" text="My
cars">
      <font>
        <Font name="System Bold Italic" size="18.0" />
      </font>
    </Label>
    <MenuBar accessibleRole="BUTTON" prefHeight="27.0" prefWidth="859.0">
     <menus>
      <Menu mnemonicParsing="false" text="Privacy">
        <items>
          <MenuItem fx:id="changePassword" mnemonicParsing="false" text="Change password" />
        </items>
      </Menu>
      <Menu mnemonicParsing="false" text="About program">
        <items>
          <MenuItem fx:id="aboutProg" mnemonicParsing="false" onAction="#handleMenuClick" text="About" />
        </items>
      </Menu>
      <Menu fx:id="helpButton" mnemonicParsing="false" text="Help">
        <items>
          <MenuItem fx:id="helpButton" mnemonicParsing="false" text="Help" />
        </items>
      </Menu>
     </menus>
     <effect>
       <Blend />
     </effect>
    </MenuBar>
    <ImageView fitHeight="200.0" fitWidth="834.0" layoutX="14.0" layoutY="327.0">
      <image>
        <Image url="@../../../../../../../../../ВНТУ/4.2/Дипломна/background.jpg" />
      </image>
    </ImageView>
    <ListView fx:id="listOfHistory" layoutX="214.0" layoutY="64.0" prefHeight="200.0" prefWidth="370.0" />
    <Label alignment="CENTER" layoutX="361.0" layoutY="31.0" prefHeight="18.0" prefWidth="75.0"
text="History">
      <font>
        <Font name="System Bold Italic" size="18.0" />
      </font>
    </Label>
    <Label alignment="CENTER" layoutX="45.0" layoutY="297.0" prefHeight="27.0" prefWidth="116.0"
```

```
text="Description">
        <font>
          <Font name="System Bold Italic" size="18.0" />
        </font>
      </Label>
      <Label fx:id="orderDate" alignment="TOP_LEFT" layoutX="14.0" layoutY="327.0" prefHeight="27.0"
prefWidth="577.0" text="Date:" textFill="WHITE">
        <font>
          <Font name="System Bold Italic" size="18.0" />
        </font>
      </Label>
      <Label fx:id="typeOfRepairement" alignment="TOP_LEFT" layoutX="14.0" layoutY="356.0" prefHeight="27.0"
prefWidth="578.0" text="Type of repairement:" textFill="WHITE">
        <font>
          <Font name="System Bold Italic" size="18.0" />
        </font>
      </Label>
      <Label fx:id="mechanicName" alignment="TOP_LEFT" layoutX="14.0" layoutY="386.0" prefHeight="27.0"
prefWidth="578.0" text="Mechanic name:" textFill="WHITE">
        <font>
          <Font name="System Bold Italic" size="18.0" />
        </font>
      </Label>
      <Label fx:id="mileage" alignment="TOP_LEFT" layoutX="14.0" layoutY="413.0" prefHeight="27.0"
prefWidth="578.0" text="Mileage:" textFill="WHITE">
        <font>
          <Font name="System Bold Italic" size="18.0" />
        </font>
      </Label>
      <Button fx:id="getInvoice" layoutX="63.0" layoutY="476.0" mnemonicParsing="false"
onAction="#onGeneratePdfButtonClick" text="Get invoice" />
      <Button fx:id="newOrder" layoutX="180.0" layoutY="476.0" mnemonicParsing="false"
onAction="#newOrderWindowOpen" text="New order" />
      <Label fx:id="cost" alignment="TOP_LEFT" layoutX="14.0" layoutY="440.0" prefHeight="27.0"
prefWidth="578.0" text="Cost:" textFill="WHITE">
        <font>
          <Font name="System Bold Italic" size="18.0" />
        </font>
      </Label>
      <Label fx:id="status" alignment="TOP_LEFT" layoutX="584.0" layoutY="327.0" prefHeight="27.0"
prefWidth="261.0" text="Status:" textFill="WHITE">
        <font>
          <Font name="System Bold Italic" size="18.0" />
        </font>
      </Label>
      <ImageView fx:id="statusImage" fitHeight="200.0" fitWidth="159.0" layoutX="641.0" layoutY="355.0"
pickOnBounds="true" preserveRatio="true">
        <image>
          <Image url="@../../../Logo.png" />
        </image>
      </ImageView>
    </children>
</AnchorPane>
```

Invoice.java:

```
package com.example.bachprog;

import java.awt.Desktop;
import java.io.File;
import java.io.IOException;

import com.itextpdf.io.image.ImageData;
import com.itextpdf.io.image.ImageDataFactory;
import com.itextpdf.kernel.color.Color;
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.kernel.pdf.canvas.PdfCanvas;
import com.itextpdf.layout.Document;
import com.itextpdf.layout.border.SolidBorder;
import com.itextpdf.layout.element.*;
import com.itextpdf.layout.property.TextAlignment;
import javafx.stage.FileChooser;
import javafx.stage.Stage;


import javax.swing.border.Border;

public class Invoice {

    public void openPdf(String path) {
        File file = new File(path);
        if (file.exists() && Desktop.isDesktopSupported()) {
            try {
                Desktop.getDesktop().open(file);
            } catch (IOException e) {
                e.printStackTrace();
                System.out.println("Cannot open PDF");
            }
        } else {
            System.out.println("Cannot find the file or Desktop doesn't support it");
        }
    }
    public static void generateCustomerInvoice(String dest, String[] repairDetailsArray, String CarInfo) {
        String imagePath = "D:\\Java Projects\\programmForBachelor\\BachProg\\src\\main\\resources\\Logo.png";

        try {
            File file = new File(dest);
            file.getParentFile().mkdirs();

            PdfWriter writer = new PdfWriter(dest);
            PdfDocument pdf = new PdfDocument(writer);
            Document document = new Document(pdf);

            Paragraph header = new Paragraph("Invoice")
                    .setTextAlignment(TextAlignment.CENTER).setBold()
                    .setFontSize(20);
            document.add(header);

            PdfCanvas canvas = new PdfCanvas(pdf.getFirstPage());
            canvas.setLineWidth(5);
            canvas.moveTo(36, 750);
```

```
        canvas.lineTo(559, 750);
        canvas.stroke();

        canvas.moveTo(36, 400);
        canvas.lineTo(559, 400);
        canvas.stroke();

        document.add(new Paragraph("\n\n Car service \"AlexPlus\"\nboul. Rène-Levesque 12, Montrèal\nMobile:
+1(514) 111-2220").setBold());


        ImageData imageData = ImageDataFactory.create(imagePath);
        Image image = new Image(imageData);

        image.setWidth(150);
        image.setFixedPosition(400, 580);

        document.add(image);


        String dateOfOrder = null; // Order date
        String typeofRepairement = null; // Type of repair
        String mechanicname = null; // Mechanic name
        String mileAge = null; // Mileage
        String Cost = null; // Cost
        String Status = null; // StatusOfOrder

        for (String repairDetail : repairDetailsArray) {
           // Split each string in the array into parts
           String[] parts = repairDetail.split(" - ");

           // Extract individual fields from the parts
           dateOfOrder = parts[0]; // Order date
           typeofRepairement = parts[1]; // Type of repair
           mechanicname = parts[2]; // Mechanic name
           mileAge = parts[3]; // Mileage
           Cost = parts[4]; // Cost
           Status = parts [5]; // StatusOfOrder


           break;
        }

        Paragraph paragraph = new Paragraph();

        paragraph.add(new Text("Billing to:\n").setBold());
        paragraph.add(new Text("Client: ").setBold());
        paragraph.add(new Text(LogInClientWindow.clientName + "\n"));

        paragraph.add(new Text("Car: ").setBold());
        paragraph.add(new Text(CarInfo + "\n"));

        paragraph.add(new Text("Mileage: ").setBold());
        paragraph.add(new Text(mileAge + "\n"));

        paragraph.add(new Text("Mechanic name: ").setBold());
        paragraph.add(new Text(mechanicname + "\n"));
```

```
        paragraph.add(new Text("Date: ").setBold());
        paragraph.add(new Text(dateOfOrder));

        document.add(paragraph);


        float[] columnWidths = {300F, 100F, 100F};
        Table table = new Table(columnWidths);
        table.addHeaderCell("Type of service").setItalic().setBold();
        table.addHeaderCell("Price").setItalic().setBold();
        table.addHeaderCell("Quantity").setItalic().setBold();

        table.addCell(typeofRepairement);
        table.addCell(Cost);
        table.addCell("1");

        document.add(table);

        document.add(new Paragraph("\nSymmary: " + Cost).setBold());

        document.add(new Paragraph("\nThanks for choosing us!").setTextAlignment(TextAlignment.CENTER));

        //Terms and conditions
        document.add(new Paragraph("\nTERMS AND
CONDITIONS").setTextAlignment(TextAlignment.LEFT).setBold());

        // Payment Terms
        document.add(new Paragraph("1. Payment is due upon completion of service unless otherwise agreed.")
                .setTextAlignment(TextAlignment.LEFT));
        document.add(new Paragraph("2. Accepted payment methods: cash, credit/debit card, or bank transfer.")
                .setTextAlignment(TextAlignment.LEFT));
        document.add(new Paragraph("3. Late payments may incur additional fees.")
                .setTextAlignment(TextAlignment.LEFT));

        // Warranty & Liability
        document.add(new Paragraph("4. Services and parts may come with a limited warranty (e.g., 12 months or
10,000 kms).")
                .setTextAlignment(TextAlignment.LEFT));
        document.add(new Paragraph("5. The service provider is not liable for pre-existing issues or damages unrelated
to the performed service.")
                .setTextAlignment(TextAlignment.LEFT));

        // Estimates & Additional Work
        document.add(new Paragraph("6. The initial estimate is subject to change based on further inspection.")
                .setTextAlignment(TextAlignment.LEFT));
        document.add(new Paragraph("7. Any additional repairs will require customer approval before proceeding.")
                .setTextAlignment(TextAlignment.LEFT));

        // Refunds & Returns
        document.add(new Paragraph("8. Refunds are only applicable for defective parts or unsatisfactory service.")
                .setTextAlignment(TextAlignment.LEFT));
        document.add(new Paragraph("9. Installed parts cannot be returned unless covered by warranty.")
                .setTextAlignment(TextAlignment.LEFT));

        // Customer Responsibilities
        document.add(new Paragraph("10. Customers must provide accurate vehicle information.")
                .setTextAlignment(TextAlignment.LEFT));
        document.add(new Paragraph("11. The service provider is not responsible for personal belongings left in the
```

```
vehicle.")
                .setTextAlignment(TextAlignment.LEFT));

            // Closing
            document.add(new Paragraph("\nBy proceeding with the service, the customer agrees to these terms and
conditions.")
                .setTextAlignment(TextAlignment.CENTER));


            document.close();
            System.out.println("PDF created: " + dest);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

Додаток Г – Ілюстративна частина

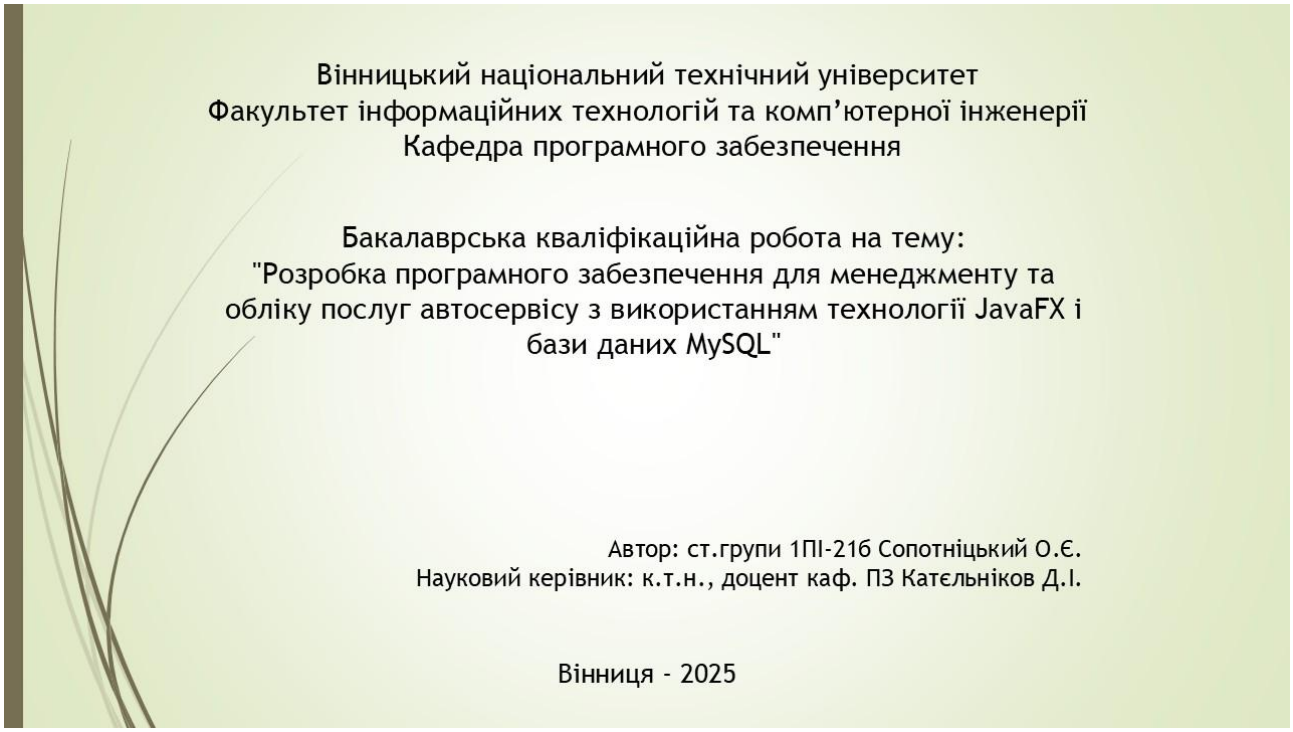


Рисунок Г.1 – Титульний слайд

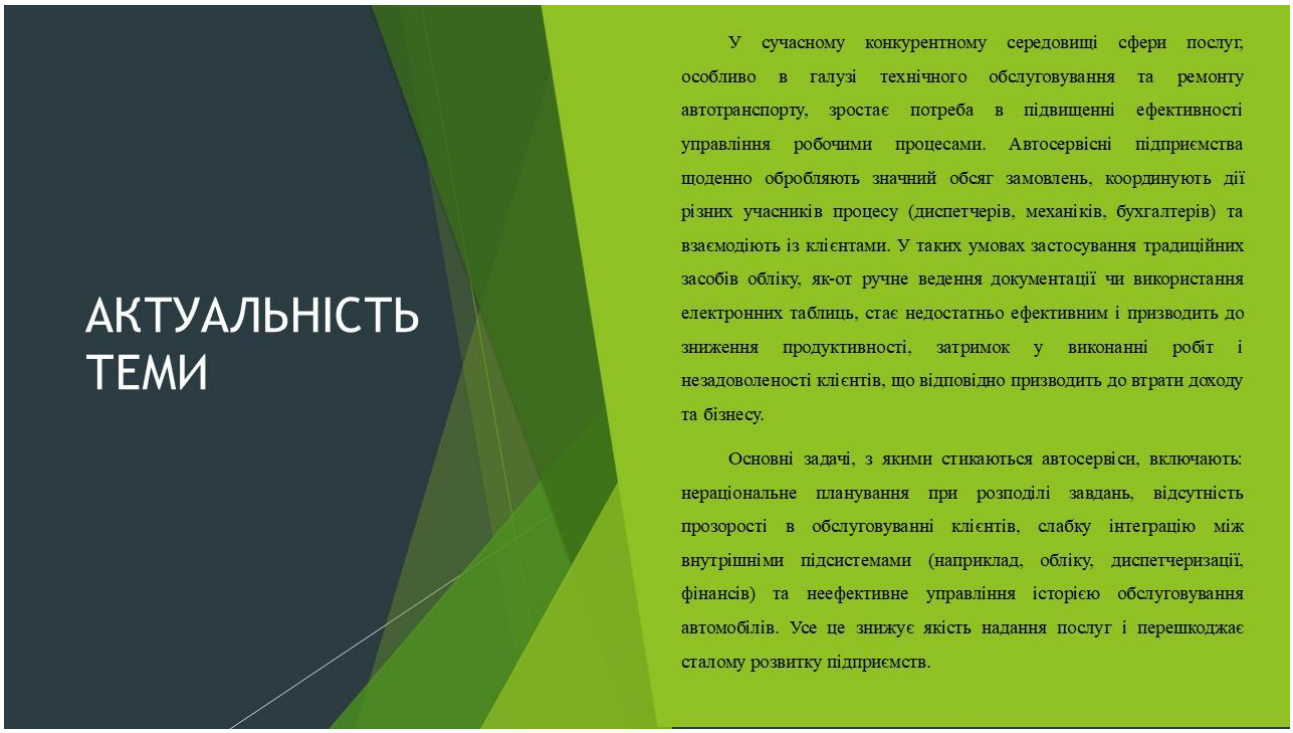


Рисунок Г.2 — Актуальність теми

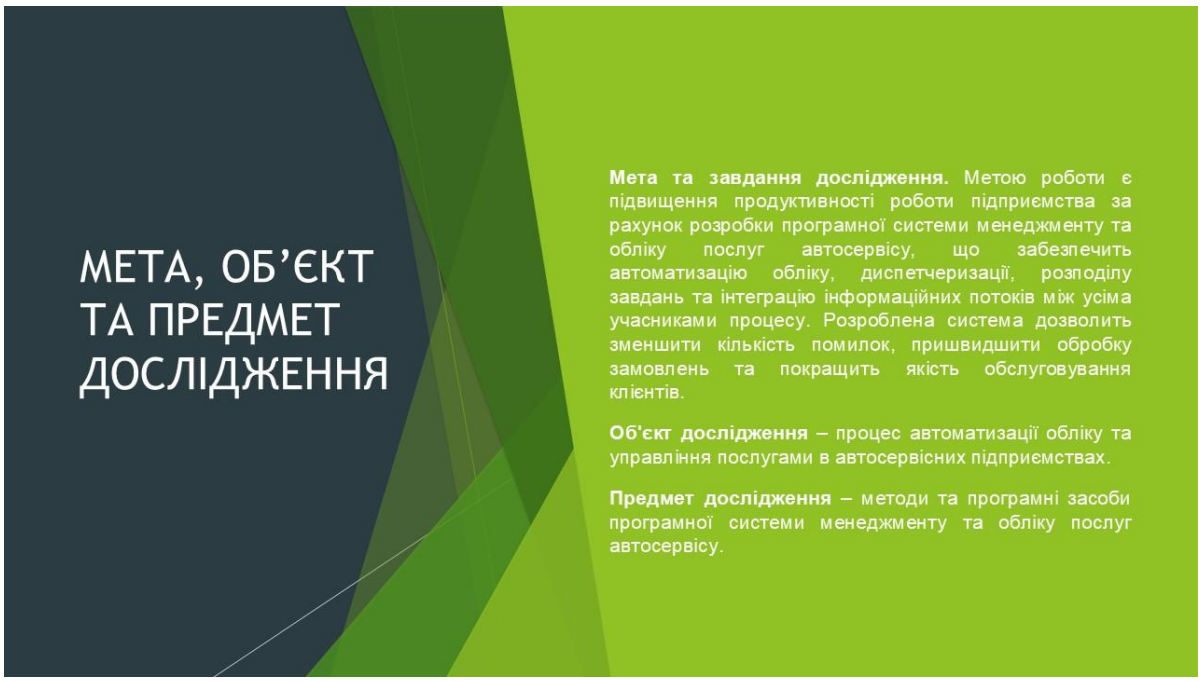


Рисунок Г.3 — Мета, об'єкт та предмет дослідження

Рисунок Г.4 — Завдання дослідження

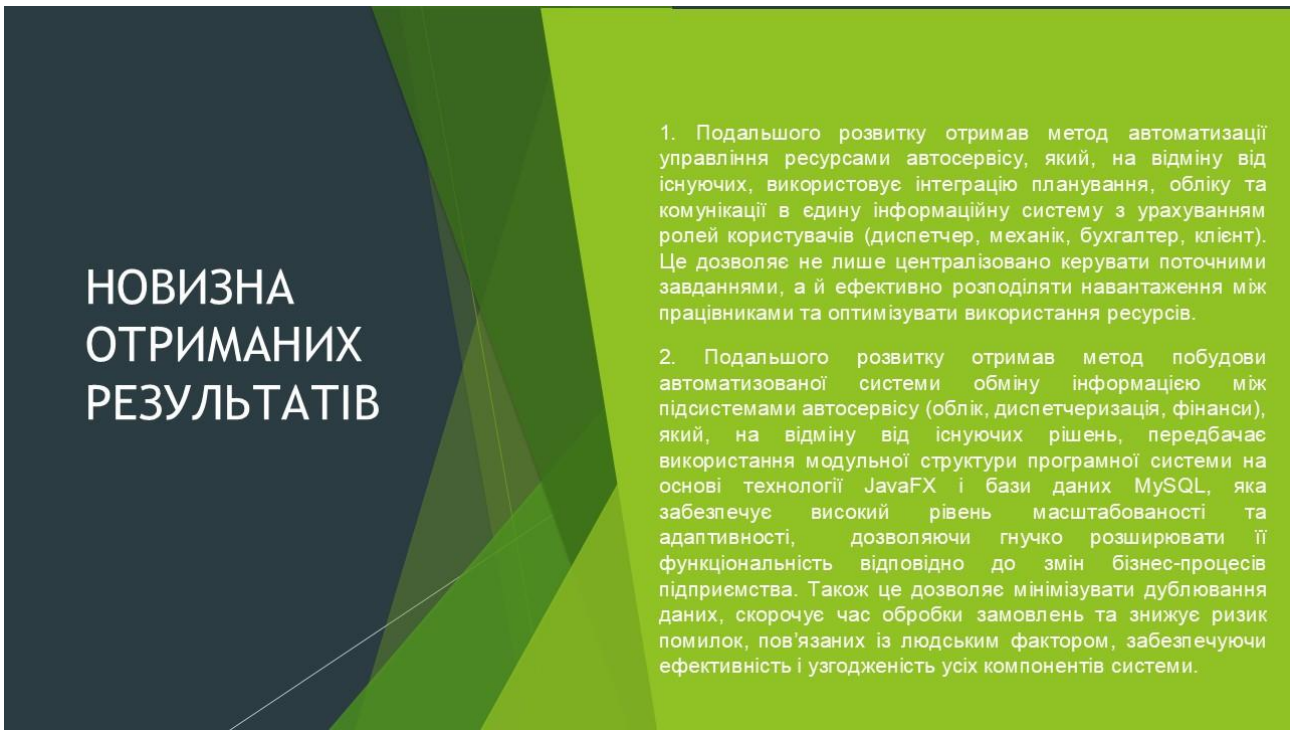


Рисунок Г.5 — Наукова новизна

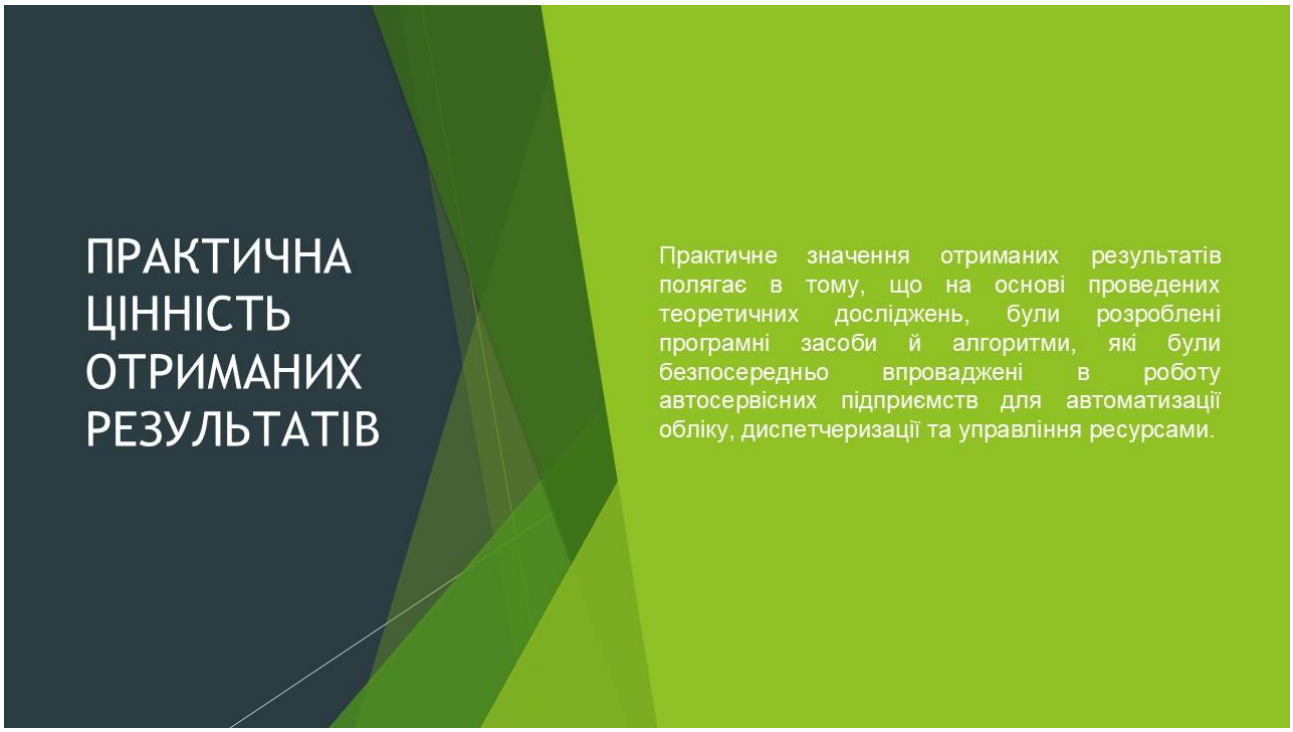


Рисунок Г.6 — Практична цінність

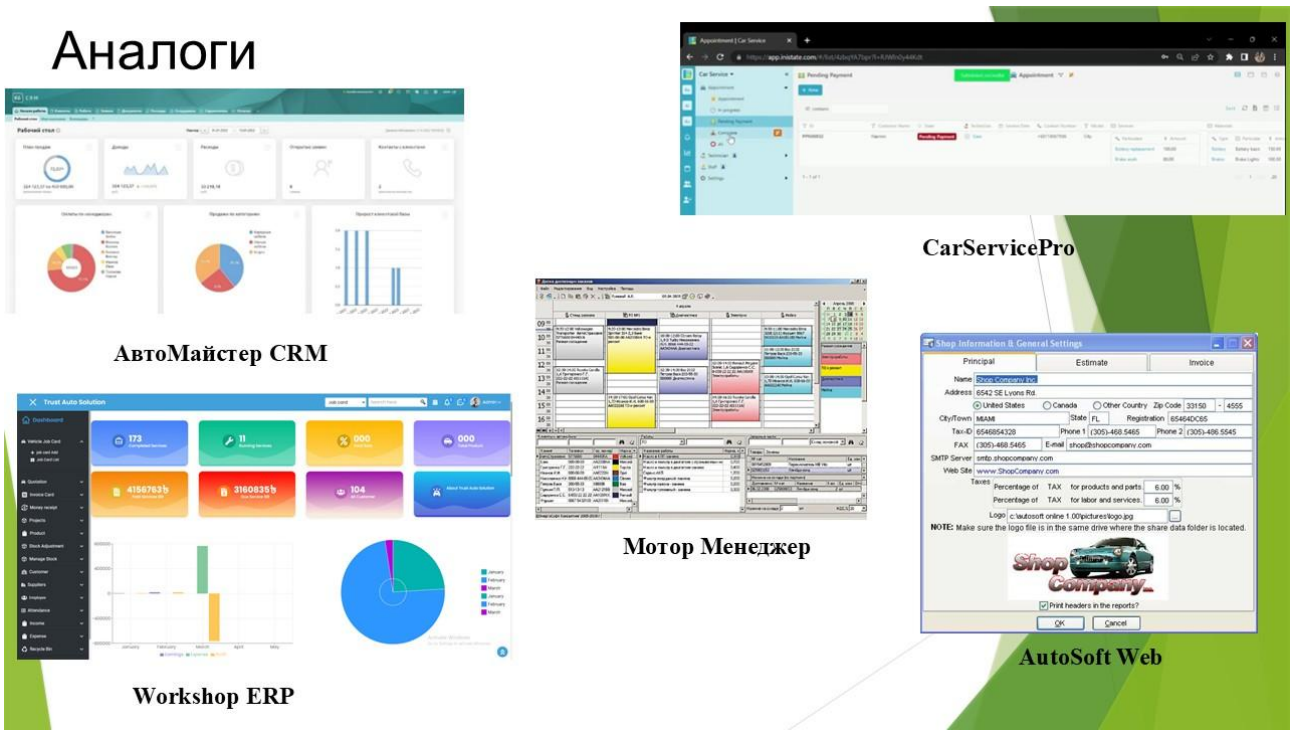


Рисунок Г.7 — Аналоги

Критерій	АвтоМайстер CRM	CarServicePro	AutoSoft Web	Workshop ERP	МоторМенеджер	Власна розробка
Відстеження статусу обслуговування авто клієнтом	+	+	-	+	-	+
Прозора історія виконаних робіт	+	+	-	-	+	+
Інструкції для механіків	-	-	+	-	-	+
Фінансова звітність	-	+	-	+	-	+
Загальна оцінка	50%	75%	25%	50%	25%	100%

Рисунок Г.8 — Порівняльний аналіз аналогів

Рисунок Г.9 — Використані технології

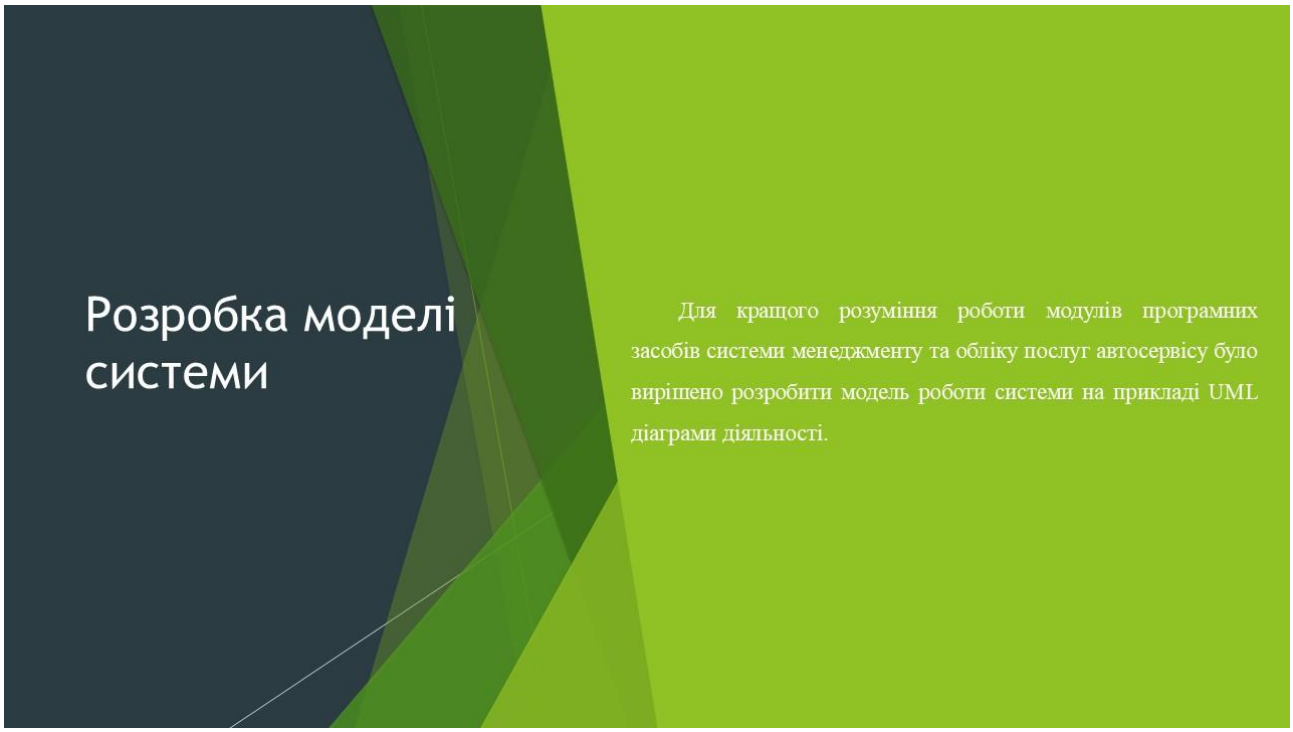


Рисунок Г.10 — Розробка моделі програмного застосунку для менеджменту та обліку послуг автосервісу

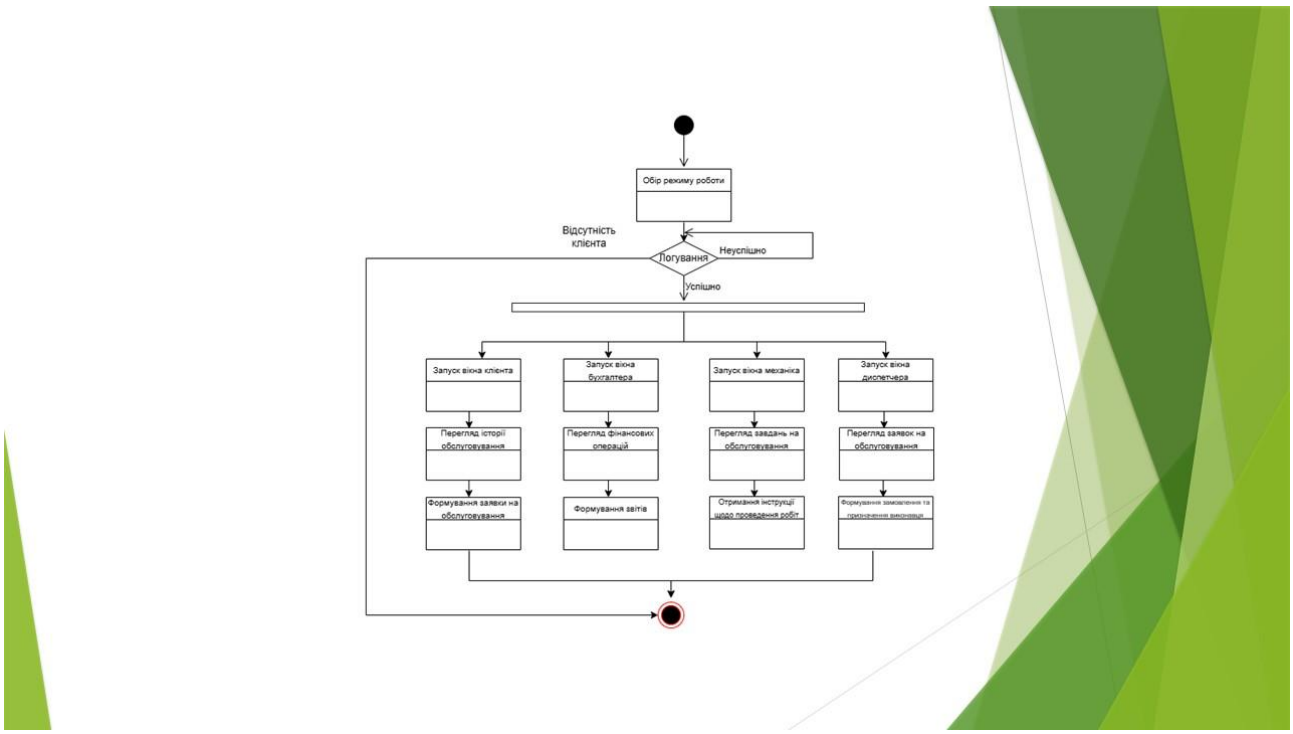


Рисунок Г.11 — Модель програмного застосунку для менеджменту та обліку послуг автосервісу

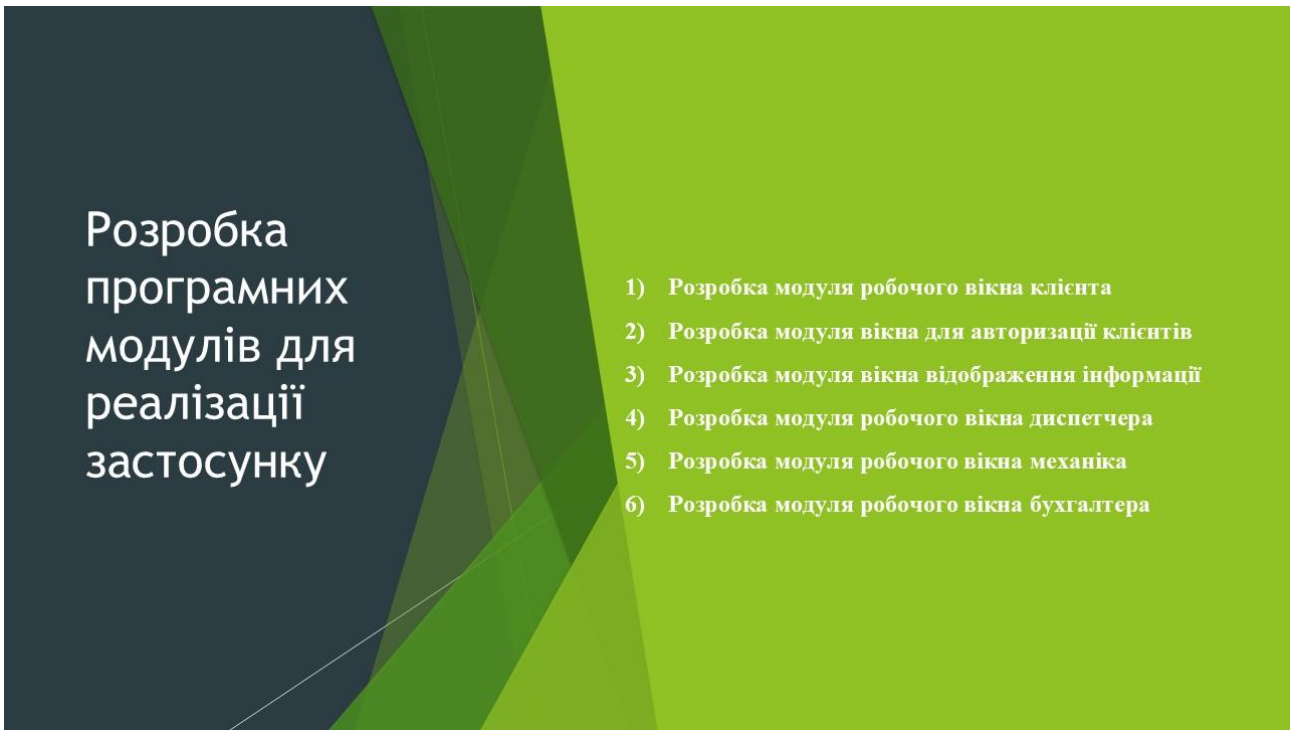


Рисунок Г.12 — Розробка програмних модулів для реалізації програмного застосунку

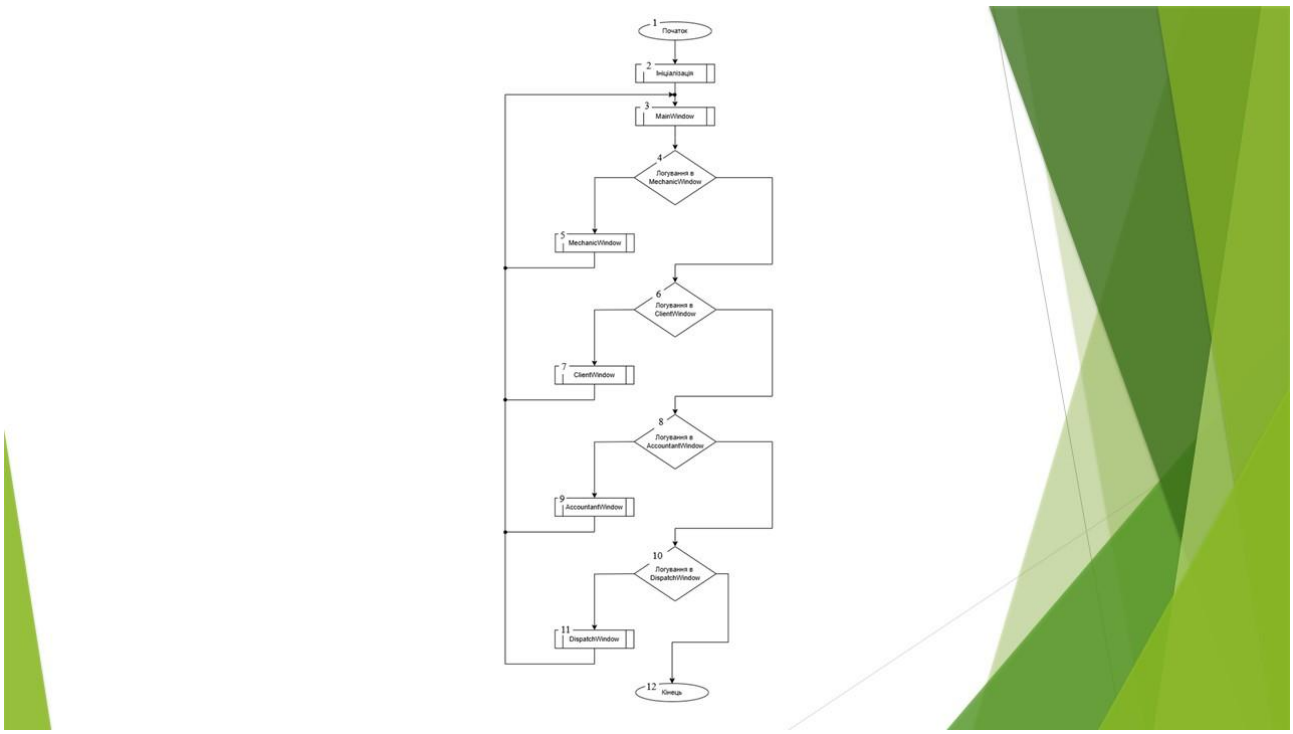


Рисунок Г.13 — Блок-схема програмного застосунку для менеджменту та обліку послуг автосервісу

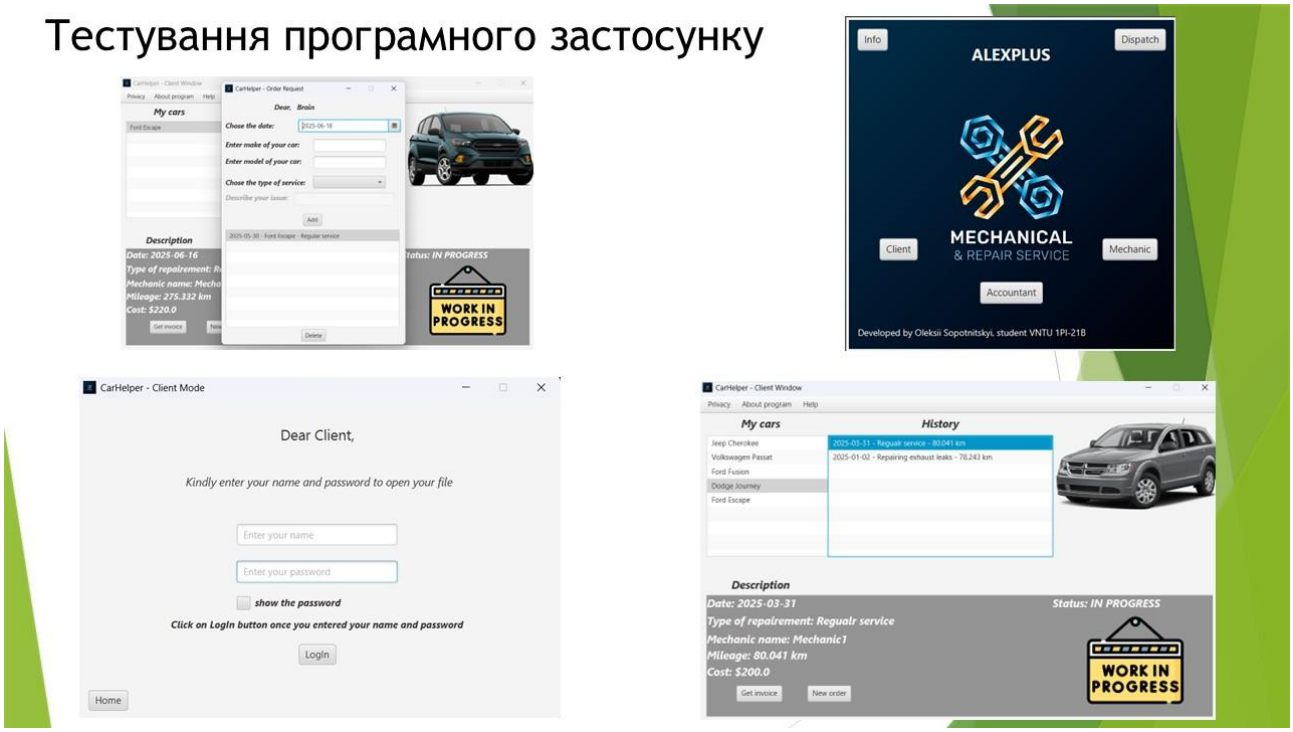


Рисунок Г.14 — Тестування програмного застосунку

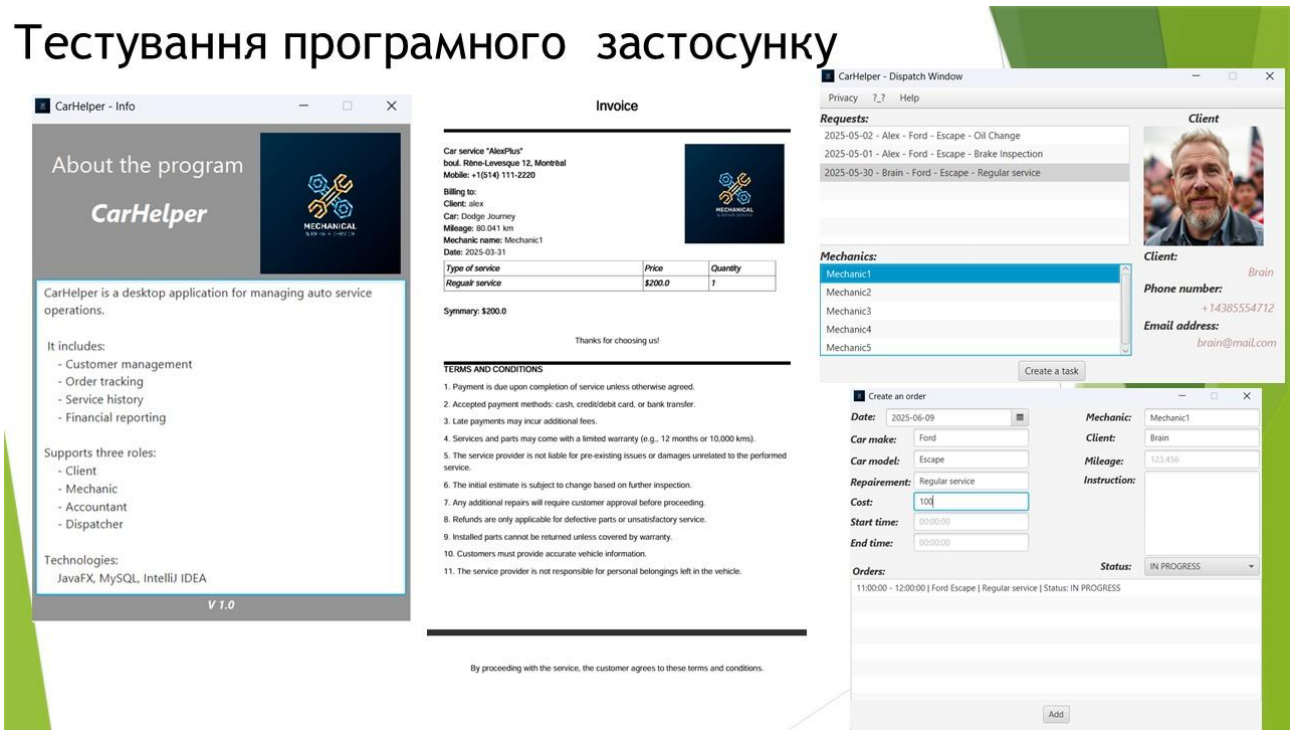


Рисунок Г.15 — Тестування програмного застосунку

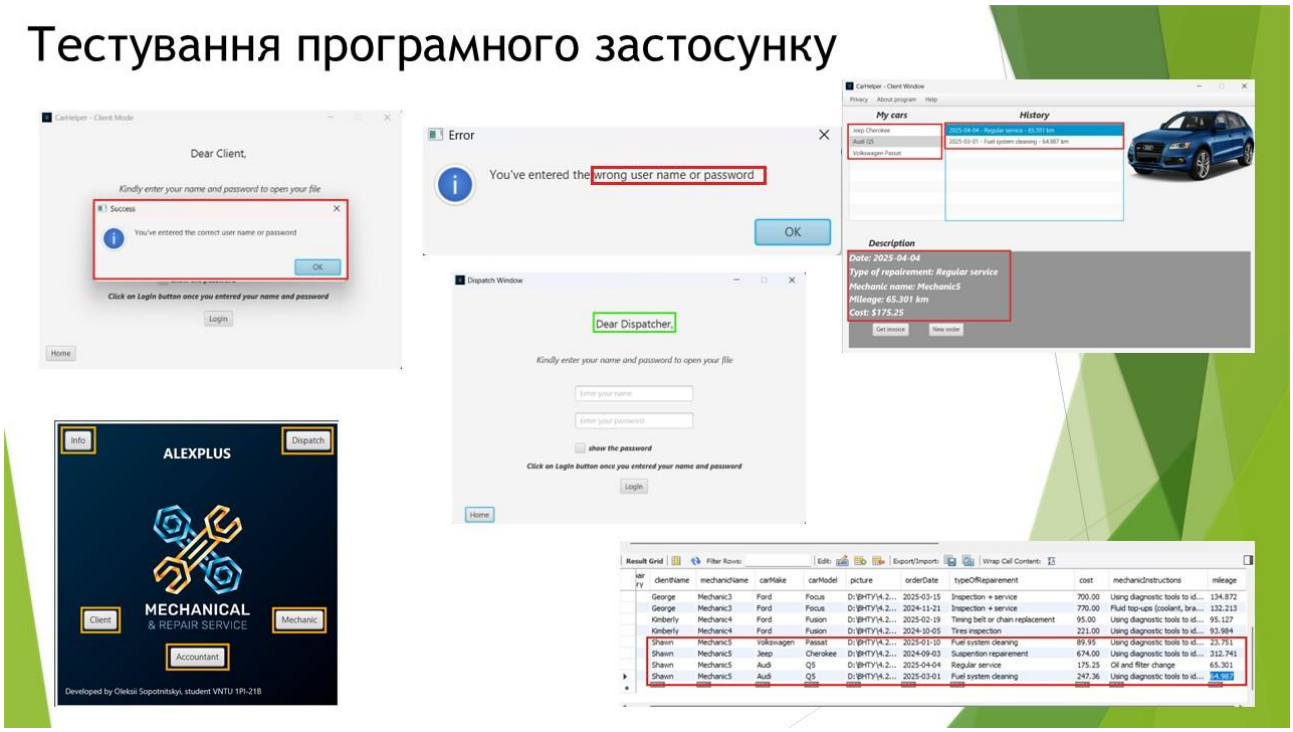


Рисунок Г.16 — Тестування програмного застосунку

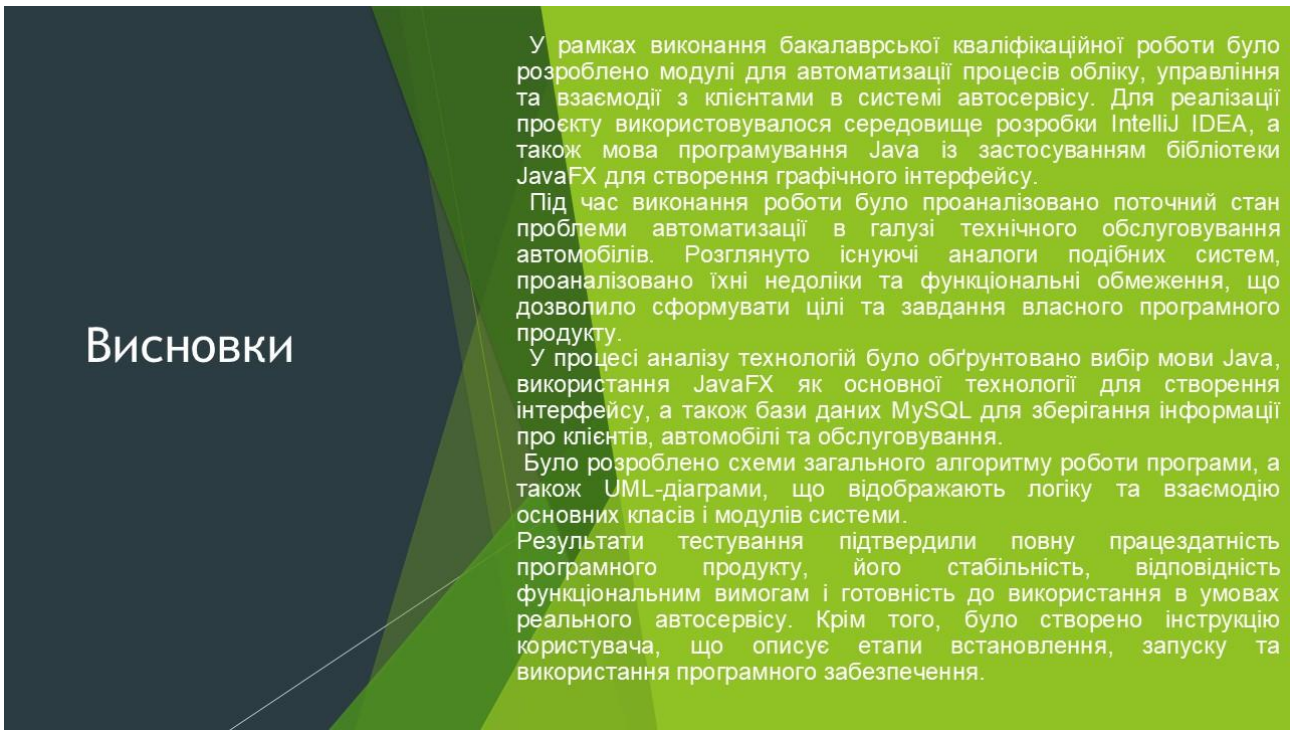


Рисунок Г.17 — Висновки

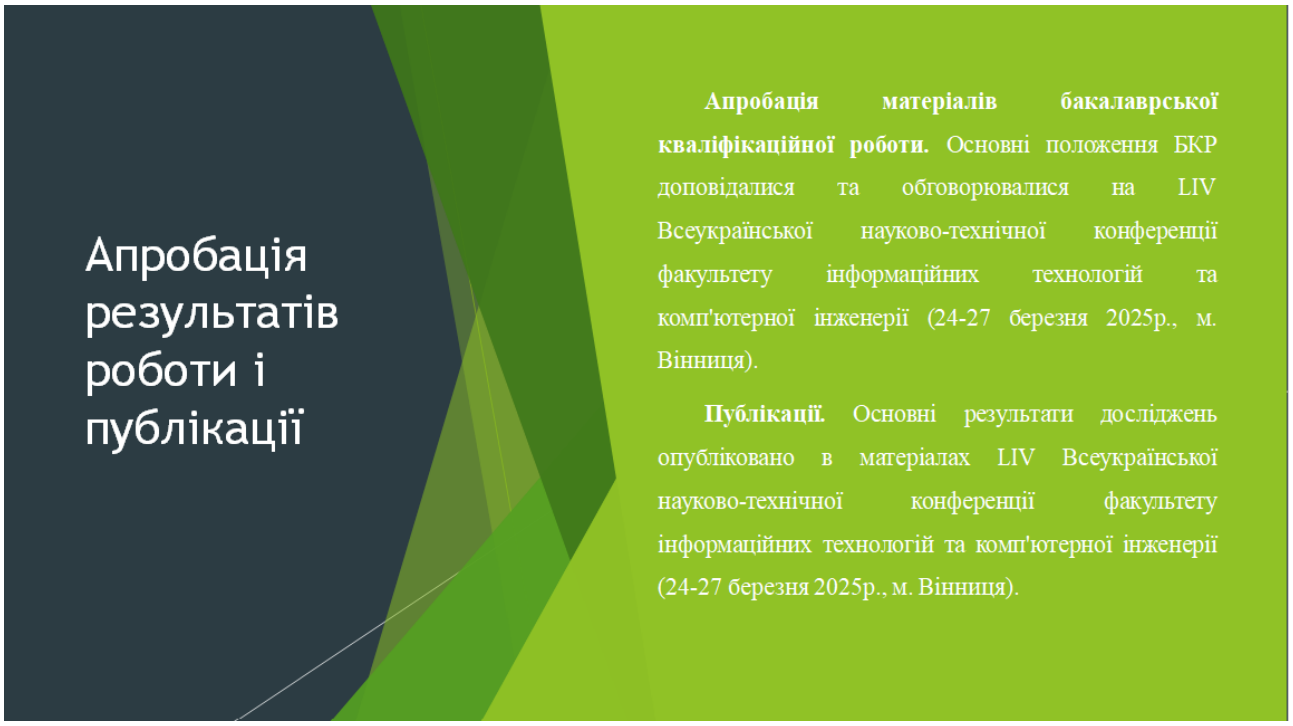


Рисунок Г.18 — Апробація результатів роботи і публікації

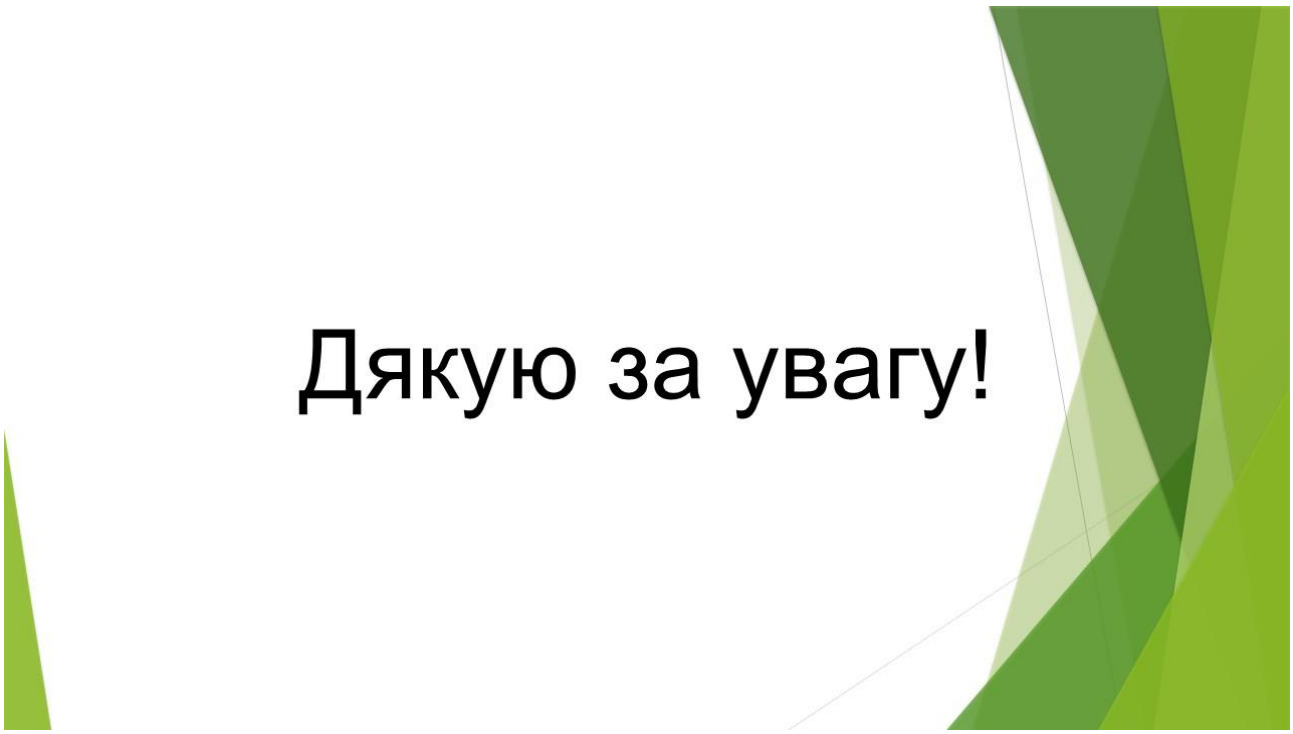


Рисунок Г.18 — Фінальний слайд