Хмельницький національний університет

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу

Назва теми

Рівень вищої освіти Перший(бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121«Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.200125.01.08.ПЗ

Виконав студент III курсу група ІПЗс-20-1 _____ В.0. Фіть
Підпис Ініціали, прізвище

Керівник канд. пед. наук, доцент _____ О. Г. Онишко
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер доцент кафедри ІПЗ _____ І.В. Гурман
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

5 червня 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
05 02 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Фітю Василю Олександровичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Програмне забезпезпечення для автоматизації обслуговування клієнтів шиномонтажу

Керівник проекту (роботи) Онишко Оксана Григорівна, канд. пед. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 05.02.2023 р. № 11
2. Строк подання студентом проекту (роботи) на кафедру 05.06.2023 р.
3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Три креслення: діаграма класів, діаграма послідовності, діаграма взаємодії

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання пр
Нормоконтроль	Гурман І. В., доцент кафедри ІПЗ	3.06.2023	5.06.2023
Антиплагіат	Гурман І. В., доцент кафедри ІПЗ	02.06.2023	02.06.2023

7. Дата видачі завдання « 05 » лютого 2023р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) Кваліфікаційної роботи	Строк виконання етапів проекту (роботи)	Прим
1 Ознайомлення з тематикою проектування кваліфікаційної роботи, визначення та узгодження індивідуальних тем	01.12 – 30.12.2022	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2023	
3 Проектування програмного забезпечення	01.02 – 28.02.2023	
4 Програмна реалізація	01.03 – 10.04.2023	
5 Тестування програмного забезпечення	11.04 – 30.04.2023	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки КР згідно вимог стандартів	01.05 – 25.05.2023	
7 Попередній захист КР	Травень 2023 (згідно графіка)	
8 Перевірка КР на плагіат, нормконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки	26.05 – 30.05.2023	
9 Підготовка до захисту та захист КР	з 01.06.2023	

Студент _____ В.О. Фіть
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ О.Г. Онишко
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: Програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу.

Автор роботи: Фіть Василь Олександрович.

Керівник роботи: Онишко Оксана Григорівна.

Пояснювальна записка: 76 с., 27 рис., 2 табл., 4 дод., 40 джерел.

Графічна частина: Три креслення: діаграма класів, діаграма послідвності, діаграма взаємодії.

Метою кваліфікаційної роботи є розробка програмного забезпечення, яке дозволить користувачам автоматизувати роботу шиномонтажу і обслуговування клієнтів, а також полегшить ведення обліку самого шиномонтажу. Система має сучасний, дружній та зрозумілий користувачеві інтерфейс.

У роботі проведений аналіз вимог, де були визначені основні функціональні та нефункціональні вимоги до програмного продукту. На основі цих вимог було розроблено архітектуру програми, включаючи модулі та їх взаємодію. Детальне проектування модулів, включаючи проектування інтерфейсу та бази даних, було проведено з урахуванням забезпечення зручності та зрозумілості користувачу.

У роботі також описана програмна реалізація модулів, включаючи обробку даних, реалізацію основних функцій та взаємодію з користувачем. Була використана реляційна база даних для зберігання та керування інформацією.

Для розробки застосунку була використана мова програмування C#, середовище Visual Studio, Sql Server та середовище Microsoft SQL Management Studio.

В результаті було розроблено програмне забезпечення, яке спрощує та оптимізує процеси шиномонтажу.

25.05.2023
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.200125.01.08.Е8	Пояснювальна записка	76		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А3		Три креслення	3		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.200125.01.08.ВД		
Виконав		Фіть В.О.		5.06	Програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу Відомість документів	Літ.	Арк. 1 / Аркушів 1
Керівник		Онишко О.Г.		5.06			
Н. Контр..		Гурман І.В.		5.06		ХНУ, ІПЗс-20-1	
Зав. Каф.		Бедратюк Л.П.		5.06			

ЗМІСТ

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.200125.01.08.ПЗ		
Виконав		Фіть В.О.		5.06	Програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу	Літ.	Арк. 4 / Аркушів 117
Керівник		Онишко О.Г.		5.06			
Н. Контр.		Гурман І.В.		5.06		ХНУ, ІПЗс-20-1	
Зав. Каф.		Бедратюк Л.П.		5.06			

ВСТУП

Сучасний розвиток технологій і зростання конкуренції у галузі обслуговування автомобілів ставлять перед шиномонтажами все більші виклики. Одним з найважливіших аспектів успішного функціонування шиномонтажу є ефективне обслуговування клієнтів, яке включає в себе швидке прийняття замовлень, точний облік виконаних робіт і взаємодію з клієнтами.

На сьогоднішній день, вирішення економічних проблем у нашій країні стає невідкладним завданням для нашої держави. Особлива увага приділяється розвитку транспортної інфраструктури, яка виступає основним фактором для забезпечення економічного зростання національної економіки та покращення якості життя населення. Підвищення ролі автотранспортного комплексу має важливе значення для розвитку економіки, забезпечення життєдіяльності населення, стійкого економічного прогресу країни, збереження обороноздатності та досягнення високоефективних зовнішньоекономічних відносин.

Однак, багато шиномонтажів все ще оперують застарілими процесами та ручним веденням даних, що може викликати помилки, затримки та незадоволення клієнтів. У зв'язку з цим, появляється потреба в ефективному програмному забезпеченні для автоматизації обслуговування клієнтів шиномонтажу.

Метою даної кваліфікаційної роботи є розробка такого програмного забезпечення, яке допоможе шиномонтажам оптимізувати процес обслуговування клієнтів, покращити точність обліку і підвищити рівень задоволеності клієнтів. Це дозволить підвищити свою конкурентоспроможність та забезпечити більш ефективне управління клієнтськими взаєминами.

Крім того, враховуючи зростання популярності та використання електронних систем управління в різних галузях, програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу стає дедалі більш

актуальним. Воно дозволяє вирішити проблеми, пов'язані з недостатньою ефективністю і помилками, що можуть виникнути при ручному веденні даних, та сприяє автоматизації процесів роботи.

Потенційна галузь застосування розробленого програмного забезпечення включає шиномонтажі різних масштабів і типів. Від невеликих автосервісів до великих шиномонтажних центрів, використання програмного забезпечення дозволить забезпечити єдиною системою керування клієнтами, замовленнями та фінансовою інформацією. Це сприятиме підвищенню ефективності роботи, зменшенню помилок та полегшенню рутинних завдань для співробітників.

Отже, розробка програмного забезпечення для автоматизації обслуговування клієнтів шиномонтажу є актуальною і необхідною завданням, яке відповідає потребам сучасної галузі обслуговування автомобілів. Це дозволить покращити якість обслуговування, підвищити задоволеність клієнтів і сприяти успішному функціонуванню даних підприємств.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Предметна область, пов'язана з програмним забезпеченням для автоматизації обслуговування клієнтів шиномонтажу, включає різноманітні аспекти та процеси, що здійснюються у цій сфері. Розуміння структурних та функціональних особливостей цієї області є важливим для ефективного розробки програмного забезпечення, що задовольняє потреби шиномонтажних підприємств.

Один із основних компонентів предметної області - це сам процес шиномонтажу. Цей процес включає такі етапи:

- прийом автомобілів в ремонт;
- демонтаж та монтаж шин;
- балансування та налаштування коліс;
- проведення інших діагностичних та ремонтних робіт.

Розуміння цих структурних аспектів дозволяє визначити ключові етапи, які потрібно враховувати при розробці програмного забезпечення.

У сучасних умовах людям доводиться працювати з великим обсягом інформації, що стосується розробки програмних продуктів для автоматизованого обліку. Ці системи представляють собою потужні інструменти, які здатні обробляти великі обсяги складної інформації в найкоротший термін, забезпечуючи зручний спілкування з користувачами.

Для аналізу діяльності організації в цілому необхідна чітка та структурована інформація про всі події, що відбуваються всередині організації, щоб правильно розробити стратегії щодо збільшення попиту на послуги та підвищення загальної ефективності роботи. Усі ці завдання можуть бути вирішені шляхом впровадження автоматизованої інформаційної системи.

Основною метою даної роботи є розробка програмного забезпечення для автоматизації обслуговування клієнтів шиномонтажу, що функціонуватиме як аналог автоматизованої системи. Це дозволить полегшити та прискорити процеси обробки даних, забезпечити зручний доступ до інформації та покращити загальний рівень обслуговування клієнтів.

Крім того, у цій області велике значення має взаємодія з клієнтами. Шиномонтажні підприємства повинні вести облік клієнтів, їх контактних даних, історії обслуговування та виконаних робіт. Важливо також забезпечити зручний механізм прийому замовлень, видачі рахунків і керування фінансовою інформацією. Розуміння функціональних особливостей обміну даними з клієнтами і внутрішніми процесами шиномонтажу є необхідним для розробки відповідних функціональних модулів програмного забезпечення.

Шиномонтажі стикаються з низкою проблем та складнощів у процесі обслуговування клієнтів. Вручну виконувати всі операції, пов'язані з прийомом клієнтів, діагностикою автомобілів, роботою з шинами та іншими аспектами, може бути трудомістким та неефективним. Деякі з головних проблем включають:

– довгий час очікування: При великому потоці клієнтів шиномонтаж стикається з проблемою тривалого очікування клієнтів на прийомі та виконання робіт. Це може призвести до невдоволення клієнтів, втрати часу та зниження загальної продуктивності;

– низька точність та помилки: Ручний підхід до обслуговування клієнтів може призвести до помилок та неточностей в інформації про замовлення, вибір та встановлення шин та інших процесів. Це може спричинити незадоволення клієнтів та незадовільну якість обслуговування;

– управління запасами: Ефективне керування запасами шин та інших компонентів є важливим аспектом шиномонтажу. Ручне відстеження та контроль запасів можуть призвести до невідповідності між попитом та наявністю товарів, а також неоптимального використання ресурсів;

– обмежені аналітичні можливості: Вручну збирати та аналізувати дані про клієнтів, замовлення, послуги та інші параметри може бути складно та затратно. Відсутність системи для аналізу даних може перешкоджати ухваленню інформованих рішень та оптимізації бізнес-процесів.

Впровадження програмного забезпечення для автоматизації обслуговування клієнтів шиномонтажу може допомогти вирішити ці проблеми та підвищити ефективність роботи шиномонтажів.

Однією з причин, що підштовхнули до створення даної системи, є те, що більшість існуючих аналогових програм, застосунків та веб-сервісів призначені для використання лише спеціально навченими користувачами через складний інтерфейс взаємодії. Крім того, однією з основних причин розробки такої системи є те, що багато подібних застосунків і сервісів мають складний процес підключення, який вимагає значних затрат часу та фінансових ресурсів.

Для вирішення цих проблем буде створено програмне забезпечення, яке допоможе звичайним працівникам шиномонтажу використовувати необхідний функціонал для ведення обліку інформації про виконані роботи. Це включатиме запис термінів виконання робіт, вартості послуг, зберігання цих даних у базі даних та здійснення звітності. Окрім того, нова система буде спрощувати процес взаємодії з користувачами та забезпечувати їм зручний і доступний інтерфейс.

Додатковою структурною особливістю предметної області є наявність інформаційних систем, що використовуються в шиномонтажних підприємствах. Ці системи можуть бути різною мірою автоматизовані і включати в себе бази даних, програмне забезпечення для обліку та управління, засоби комунікації тощо. Розуміння структурних особливостей цих інформаційних систем допомагає виявити потенційні проблеми та недоліки, які можуть бути вирішені розробкою нового програмного забезпечення.

Також варто враховувати галузь застосування розробленого програмного забезпечення. Вона охоплює шиномонтажні підприємства різних масштабів,

включаючи автосервіси, шиномонтажні центри та автосалони. Переваги автоматизації обслуговування клієнтів у цих підприємствах включають полегшення рутинних процесів, зменшення помилок, покращення точності обліку та забезпечення ефективного взаємодії з клієнтами. Розробка програмного забезпечення здатного відповідати потребам цієї галузі є доцільною та актуальною.

Враховуючи, структурні та функціональні особливості предметної області програмного забезпечення для автоматизації обслуговування клієнтів шиномонтажу мають вирішальне значення для розробки ефективного і пристосованого до потреб рішення. Детальний аналіз цих особливостей допоможе визначити вимоги до програмного забезпечення та покращити якість обслуговування в шиномонтажних підприємствах.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Для початку, проведемо аналіз різних програм, застосунків та веб-сервісів, які використовуються в даній галузі. Це можуть бути внутрішньо розроблені системи, комерційні програмні продукти або веб-платформи. Основною метою цього аналізу є виявлення основних переваг та недоліків наявних рішень.

Перш за все, будуть проаналізовані функціональні можливості існуючого програмного забезпечення. Буде визначено, які конкретні функції воно здійснює, наприклад, реєстрація клієнтів, ведення обліку робіт, розрахунок вартості послуг тощо. Крім того, будуть вивчені можливості інтеграції з іншими системами або пристроями, наприклад, фінансовими системами або обладнанням для шиномонтажу.

Останнім часом спостерігається активний прогрес інформаційних технологій, а їх використання набуває все більшої актуальності. У сфері

навчання це стає наступним кроком у розвитку, спрямованим на надання навчальному процесу характеристик, які включають адаптивність, гнучкість, відкритість та мобільність. Зараз відбувається інтенсивне впровадження "хмарних технологій" та сервісів в систему освіти на рівні середньої та вищої школи, а також активна розбудова єдиного інформаційного простору.

Для досягнення найбільш зручного інтерфейсу програми та максимальної відповідності функціоналу поставленим цілям необхідно ознайомитися з наявними технологіями, методами та програмами, що використовуються для збереження, фільтрування та редагування даних, перегляду інформації, формування звітності та інших функцій.

Одним з варіантів для зберігання облікової інформації є використання хмарних сервісів. Хмарні обчислення - це модель, що надає зручний доступ до мережевих ресурсів на вимогу. Це означає, що користувач може отримати доступ до конфігуруємих обчислювальних ресурсів, таких як мережі передачі даних, сервери, засоби зберігання даних, застосунки і сервіси, безпосередньо через провайдера. Однак, варто враховувати деякі недоліки, пов'язані з хмарними сервісами.

Залежність від компанії: використання хмарних сервісів означає, що ваші дані зберігаються на серверах певної компанії. Ви залежите від цієї компанії для збереження та забезпечення доступу до своїх даних.

Монополізація: існує ризик появи хмарних монополістів, коли великі компанії контролюють значну частину хмарних сервісів. Це може призвести до обмежень в конкуренції та обробці даних.

Залежність від мережі: для роботи з хмарними сервісами потрібний постійний доступ до Інтернету. Відсутність з'єднання може обмежити ваш доступ до даних.

Безпека: існує ризик хакерських атак на сервери хмарних сервісів. При зберіганні даних на локальному комп'ютері ви маєте більшу контроль над безпекою і можливістю захистити свої дані шляхом використання антивірусного програмного забезпечення.

Монетизація ресурсу: у майбутньому компанії можуть ввести плату за використання хмарних сервісів. Це може привести до додаткових витрат для користувачів, які раніше могли користуватися цими сервісами безкоштовно.
Таким чином, хмарні сервіси для зберігання інформації мають свої переваги, такі як зручний доступ і високий рівень захисту даних. Проте, важливо враховувати й недоліки, такі як залежність від компанії, ризик монополізації, потребу у постійному з'єднанні з мережею, можливість хакерських атак та можливу монетизацію ресурсу. Перед використанням хмарних сервісів, користувачам слід уважно оцінити ці переваги та недоліки і вибрати найбільш підходящий сервіс для своїх потреб.

У нашій кваліфікаційній роботі ми розглядаємо систему автоматизації обслуговування клієнтів шиномонтажу. Декілька аналогів таких систем, які можуть мати подібний функціонал.

"AutoShop Management System". Ця система надає функції для керування всіма аспектами шиномонтажного центру, включаючи прийом клієнтів, розрахунок вартості робіт, планування графіка роботи, складський облік шин та запчастин, а також звітність. Перевагою цієї системи є простий інтерфейс, легкість використання та налаштування, а недоліком може бути обмежені можливості налаштування під конкретні потреби.

"Garage Management Software". Ця система спеціально розроблена для автосервісів і включає функціонал для управління клієнтами, розрахунку цін, планування графіка роботи, обліку запчастин та інвентарю, а також збереження історії обслуговування. Перевагою цієї системи є широкі можливості налаштування та інтеграція з іншими системами, а недоліком може бути складність в освоєнні для новачків.

"Tire Shop Management System". Ця система спеціально призначена для шиномонтажних центрів і надає функціонал для керування клієнтами, розрахунку цін, графіка роботи, управління складом шин та запчастин, а також генерації звітів. Перевагою цієї системи є простота використання, швидке

внесення даних та аналітика продажів, а недоліком може бути обмежена функціональність в порівнянні з іншими системами.

Опис переваг та недоліків даних аналогових систем наведено у таблиці 1.1.

Таблиця 1.1 - аналогові системи

Назва системи	Переваги	Недоліки
AutoShop Management System	– Простий інтерфейс; – легкість використання; – швидке налаштування.	– Обмежені можливості налаштування; – відсутність додаткових модулів або розширених аналітичних звітів.
Garage Management Software	– Широкі можливості налаштування; – інтеграція з іншими системами; – загальний функціонал для шиномонтажного центру.	– Складність освоєння для новачків; – вимагає більшої часової та фінансової витрати на налаштування.
Tire Shop Management System	– Зручний інтерфейс; – простота використання; – швидке внесення; – аналітика продажів; – спеціалізований функціонал.	– Обмежена функціональність порівняно з іншими системами; – відсутність модулів для розширення функціоналу.

В результаті детального аналізу наявного програмно-технічного забезпечення предметної області, були виявлені наступні висновки та результати:

– було проаналізовано різноманітні програми, застосунки та веб-сервіси, що використовуються в галузі автоматизованого обслуговування клієнтів шиномонтажу. Виявлено, що на ринку існує широкий спектр рішень з різними функціональними можливостями та характеристиками;

– були ідентифіковані переваги та недоліки наявного програмного забезпечення. Це включає сильні сторони, такі як підтримка основних функцій обслуговування клієнтів, можливість інтеграції з іншими системами, а також слабкі сторони, такі як складний інтерфейс користувача, обмежені можливості адаптації до конкретних потреб підприємства;

– визначено основні проблеми, що виникають при використанні наявного програмного забезпечення, такі як обмежена функціональність, низька продуктивність, висока вартість підтримки та обновлень;

– зроблено порівняння існуючих рішень з майбутнім програмним забезпеченням, що буде розроблено для автоматизації обслуговування клієнтів шиномонтажу. Це дозволило виділити переваги нової системи, такі як покращена функціональність, зручний інтерфейс користувача, ефективність та економічна доцільність.

Були проаналізовані сучасні тенденції та інновації в автоматизованому обслуговуванні клієнтів, які можуть бути використані для розробки нової системи. Цей аналіз відзначив важливість та актуальність створення нового програмного забезпечення для шиномонтажу.

Отже, результати дослідження програмно-технічного забезпечення в даній галузі мають велике значення для подальшої роботи. Вони служать основою для формулювання вимог до нової системи, яка буде розроблена з метою поліпшення обслуговування клієнтів у шиномонтажі. Крім того, отримані результати дослідження дозволяють обговорити замовнику можливі

інвестиції у розробку нового програмного забезпечення і відзначають переваги, які нова система може принести підприємству.

Кінцевим результатом цього підрозділу є збір та систематизація інформації про наявні рішення, виявлення їх переваг та недоліків, а також підкреслення необхідності розробки нового програмного забезпечення для поліпшення автоматизованого обслуговування клієнтів у шиномонтажі. Отримані висновки стануть основою для наступних розділів кваліфікаційної роботи, таких як формулювання вимог, проектування нової системи та її реалізація.

1.3 Визначення вимог до програмного продукту

На даному етапі наша робота зосереджена на проведенні ретельного аналізу потреб і вимог користувачів системи шиномонтажу, щоб досягти успішного результату. Цей аналіз включає глибоке вивчення функціональних та нефункціональних вимог, які пов'язані з роботою програмного продукту.

Функціональні вимоги передбачають опис конкретних функцій та можливостей, які має мати програмний продукт. Наприклад, це можуть бути функції реєстрації клієнтів, ведення обліку робіт, генерації звітів та інші дії, які програма повинна виконувати для задоволення потреб користувачів. Важливо ретельно визначити ці функції і їх поведінку, а також урахувати можливі варіанти використання системи та забезпечити їх правильну реалізацію.

Нефункціональні вимоги охоплюють різні аспекти, які визначають якість програмного продукту. Це можуть бути вимоги до надійності, яка гарантує безперебійну роботу системи, вимоги до продуктивності, які визначають швидкодію та ефективність системи, вимоги до безпеки, які забезпечують захист даних та запобігають несанкціонованому доступу, вимоги до зручності використання, що забезпечують зрозумілість та зручність інтерфейсу

програмного продукту, а також інші аспекти, які впливають на функціональність та задоволення користувачів.

Детальний аналіз функціональних та нефункціональних вимог дозволить нам зрозуміти потреби користувачів і визначити основні цілі, які повинна вирішити нова система шиномонтажу.

Такий підхід дозволяє нам точно визначити, які функції має виконувати програмний продукт і які вимоги він повинен задовольняти. Детальний аналіз потреб і вимог користувачів є основою для подальшого проектування і розробки системи шиномонтажу.

Під час вивчення функціональних вимог ми докладно розглядаємо різні аспекти роботи програмного продукту. Наприклад, ми визначаємо, які операції повинні бути доступні користувачам, які дані необхідно обробляти, які функції мають бути реалізовані для виконання певних завдань. Це допомагає нам створити повний перелік функціональних можливостей, які система має мати.

Нефункціональні вимоги визначаються для забезпечення високої якості програмного продукту. Ми вивчаємо різні аспекти, які впливають на продуктивність, надійність, безпеку і зручність використання системи. Наприклад, ми встановлюємо вимоги до швидкодії системи, максимального часу перерв у роботі, заходів забезпечення безпеки, інтерфейсу користувача та інших аспектів.

В ході детального аналізу потреб і вимог користувачів системи шиномонтажу, ми проводимо систематичне дослідження, щоб усвідомити їхні очікування і визначити ключові аспекти, які впливатимуть на успішне впровадження програмного продукту. Цей процес включає вивчення функціональних та нефункціональних вимог, які встановлюють, як система має працювати та які характеристики має мати.

Під час аналізу функціональних вимог, ми розглядаємо конкретні дії, які користувачі повинні здійснювати у системі, а також функції, які система має забезпечити. Це можуть бути такі дії, як реєстрація клієнтів, планування та

керування роботами, генерація звітів, керування запасами тощо. Важливо чітко визначити, які операції має виконувати система та як вони пов'язані між собою.

Структура та функції шиномонтажу варіюються залежно від конкретного підприємства чи організації, але зазвичай включають такі основні елементи:

– зона прийому клієнтів. Шиномонтаж зазвичай має спеціальну зону, де клієнти можуть припаркувати свої автомобілі та звернутися за допомогою. У цій зоні відбувається початковий етап обслуговування, включаючи прийом замовлень, заповнення документів та консультації з клієнтами;

– діагностика автомобілів. Частиною роботи шиномонтажу є діагностика стану автомобілів клієнтів, особливо щодо шин та коліс. Це може включати перевірку тиску в шинах, знос протектора, балансування коліс та інші аспекти, які можуть вплинути на безпеку та продуктивність автомобіля;

– робота з шинами. Головною функцією шиномонтажу є встановлення, заміна та ремонт шин. Це може включати зняття старих шин з дисків, встановлення нових шин, балансування коліс, перевірку глибини протектора та правильне закріплення коліс;

– додаткові послуги. Деякі шиномонтажі пропонують додаткові послуги, такі як зберігання та обслуговування шин, ремонт дисків, регулювання сходження та розвалу коліс, а також надання консультацій щодо вибору шин та рекомендацій щодо їх експлуатації;

– клієнтське обслуговування та документація. Шиномонтаж забезпечує обслуговування в процесі роботи, відповідаючи на запитання клієнтів, надаючи інформацію про статус замовлення та виконуючи інші адміністративні завдання. Крім того, важливо вести документацію про виконані роботи, використовувані матеріали та послуги з метою обліку та контролю.

Управління всіма цими функціями шиномонтажу може бути складним і потребує ефективної організації та координації. Впровадження програмного забезпечення для автоматизації обслуговування клієнтів шиномонтажу може значно спростити та оптимізувати ці процеси.

Програмне забезпечення може включати такі функції:

– керування клієнтськими даними. За допомогою програмного забезпечення можна створити базу даних клієнтів, де зберігаються контактні дані, історія замовлень, уподобання та інша інформація. Це дозволяє шиномонтажу легко відстежувати та керувати клієнтськими запитами, історією обслуговування та надавати персоналізований підхід;

– розклад та планування. Програмне забезпечення може надавати функціональність для створення розкладу роботи, планування замовлень та призначення спеціалістів на певні роботи. Це дозволяє оптимізувати завантаження персоналу та керувати процесами шиномонтажу більш ефективно;

– управління запасами. Програмне забезпечення допоможе шиномонтажу відстежувати запаси шин, дисків та інших матеріалів, а також автоматично сповіщати про необхідність поповнення запасів при досягненні мінімального рівня. Це допомагає запобігти нестачі товарів та забезпечити наявність необхідних компонентів для обслуговування клієнтів.

– облік та фінансовий контроль. Програмне забезпечення дозволяє вести облік виконаних робіт, виставлення рахунків клієнтам, відстеження фінансових операцій та контроль витрат. Це допомагає шиномонтажу ефективно керувати фінансовою стороною бізнесу та забезпечувати точність ведення бухгалтерії.

– аналітика та звітність. Програмне забезпечення може надавати інструменти для аналізу даних, генерації звітів про виконані роботи, прибуток, клієнтську активність та інші ключові показники продуктивності. Це допомагає приймати поінформовані рішення, оптимізувати бізнес-процеси та планувати стратегічний розвиток.

Для детальнішого аналізу розглянемо Діаграму варіантів використання на рис. 1.1:

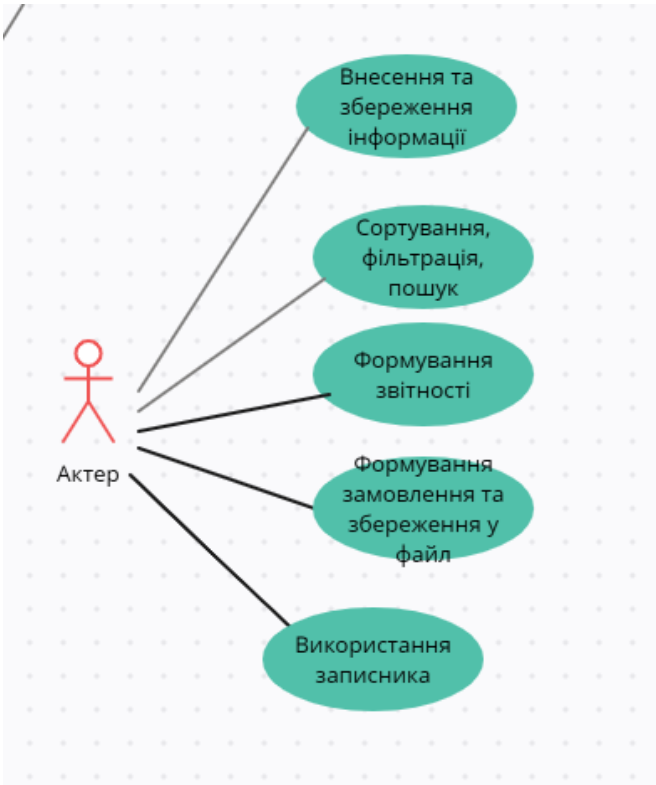


Рисунок 1.1 – діаграма варіантів використання

У діаграмі варіантів використання присутній один користувач, який має різні можливості та функціонал системи. Нижче наведено детальний опис кожної з цих можливостей:

– внесення та збереження інформації. Користувач може вводити нові дані або змінювати існуючу інформацію в системі. Це може включати додавання даних про клієнтів, автомобілів, замовлень, запасних частин тощо. Після внесення змін користувач має можливість зберегти цю інформацію для подальшого використання;

– сортування, фільтрація, пошук. Користувач може використовувати різні методи для організації та швидкого знаходження необхідних даних. Він може сортувати дані за певними критеріями (наприклад, за датою, ім'ям або типом автомобіля), застосовувати фільтри для обмеження видимості певних записів та виконувати пошук за ключовими словами або параметрами;

– формування звітності. Користувач має можливість створювати звіти на основі наявних даних. Це може включати генерацію звітів про замовлення, стан

робіт, клієнтську базу, фінансову інформацію тощо. Звіти можуть бути виконані у вигляді текстових документів, таблиць або графіків для зручного аналізу та подальшого використання;

– формування замовлення та збереження у файл: Користувач може скласти нове замовлення на послуги чи запасні частини. Він може вибрати необхідні пункти зі списку доступних позицій, вказати кількість, а також додаткову інформацію. Після цього замовлення можна зберегти у файловому форматі для подальшого використання або друку. Це дає користувачу зручну можливість зберегти деталі замовлення та використати їх в майбутньому без необхідності повторного введення даних;

– використання записника. Користувач має можливість використовувати вбудований записник в системі. Це дозволяє зберігати примітки, нагадування, список завдань або будь-яку іншу додаткову інформацію, яка може бути корисною під час роботи з системою. Записник дозволяє зручно організовувати та зберігати додаткові дані, які необхідні користувачу для ведення роботи або планування.

Ці можливості спрямовані на полегшення роботи користувача та забезпечення його ефективності в системі шиномонтажу. Вони дозволяють вносити та зберігати інформацію, швидко знаходити необхідні дані, створювати звіти та замовлення, а також мати доступ до записника для збереження додаткових деталей. Цей функціонал робить систему більш гнучкою та привабливою, сприяючи ефективній роботі та підвищенню продуктивності.

1.4 Постановка задачі та висновки

Під час проведення дослідження виявлено, що у сфері автоматизації роботи шиномонтажу існує потреба у програмному продукті, який спрощуватиме та оптимізуватиме робочі процеси в цій галузі. При аналізі наявних рішень виявлено, що багато з них мають обмежену функціональність, не відповідають потребам користувачів та мають не зовсім зручний інтерфейс.

Основна мета нашої кваліфікаційної роботи полягає у розробці програмного продукту для автоматизації роботи шиномонтажу, що буде вирішувати виявлені проблеми та задовольняти потреби користувачів. З метою досягнення цієї мети були поставлені наступні завдання:

– аналіз та вивчення вимог та потреб користувачів у галузі шиномонтажу;

– розробка графічного інтерфейсу для зручної взаємодії користувачів з програмним продуктом;

– реалізація модулів програми, що забезпечують автоматизацію робочих процесів шиномонтажу;

– проектування та реалізація бази даних для зберігання необхідної інформації;

– проведення ручного тестування програмного продукту та виправлення помилок та недоліків.

В результаті виконання цих завдань очікується отримання функціонального та ефективного програмного продукту, що сприятиме вдосконаленню робочих процесів шиномонтажу, забезпечить зручну та ефективну роботу користувачів та підвищить загальну якість надання послуг у цій галузі.

Було проведено аналіз і вивчення предметної області, виявлені існуючі проблеми та визначені цілі і завдання роботи. Було обґрунтовано необхідність розробки програмного продукту для автоматизації роботи шиномонтажу. Також були сформульовані основні задачі, які перед нами стоять у процесі розробки.

Виконання цих задач має на меті поліпшити якість та ефективність роботи шиномонтажу, забезпечити зручну та ефективну роботу користувачів та покращити надання послуг у цій сфері.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір типу архітектури та шаблонів проектування

У ході розробки застосунку важливо враховувати, як він обслуговуватиметься надалі. Слід продумати особливості додавання змін та редагування функціональності. Крім того, важливо розуміти, як зовнішні елементи інтерфейсу взаємодіють із внутрішніми процесами, і за якими принципами користувачі взаємодіятимуть із програмою. У цьому допомагає архітектура програмного забезпечення.

Архітектура програмного рішення виконує низку важливих завдань:

– визначає структуру програми та дозволяє зрозуміти, як вона влаштована, на яких рівнях виконуються ті чи інші завдання та функції;

– визначає поведінку та взаємодію елементів, завдяки чому стає зрозуміло, що відбувається, якщо виконується певна дія;

– визначає значні та другорядні елементи, що дозволяє оцінити вартість розробки, зрозуміти, які елементи обов'язково впроваджувати, а яких можна відмовитися на користь економічного міркування;

– допомагає зрозуміти, наскільки програма масштабується, як складно буде впроваджувати нові функції та який стек технологій використовувати;

– дозволяє задовольнити потреби клієнта та адаптувати програму під взаємовиключні вимоги, наприклад, високий рівень функціональності та визначення меж часу, у такому разі стає зрозуміло, як це реалізувати;

– дозволяє зрозуміти логічні взаємозв'язки у програмі;

– дає можливість коректно вести документацію та чітко розписувати функціонал, що суттєво спрощує подальше обслуговування програми, внесення змін та роботу з існуючим функціоналом.

Основний вибір архітектури залежить від потреб нашого застосунку, масштабуваності, зручності розробки та інших факторів. Ось кілька популярних типів архітектур, які можуть бути використані для реалізації нашого десктопного застосунку. Зображено на таблиці 2.1.

Таблиця 2.1 - порівння наявних архітектур

Тип архітектури	Переваги	Недоліки
Клієнт-серверна архітектура	– Розділення функціональності між клієнтом та сервером; – централізоване управління та контроль; – забезпечує розширення та підтримку.	– Залежність від доступності сервера та мережевого з'єднання; – збільшений обсяг комунікації між клієнтом та сервером.
Модель-представлення-контролер (MVC)	– Розділення між моделлю, представленням та контролером; – полегшує розширення, тестування та підтримку; – підвищує повторне використання коду.	– Може стати складним для менших проектів; – потребує правильного розподілу обов'язків між компонентами.
Компонентно-орієнтована архітектура	– Розділення функціональності; – сприяє повторному використанню та модульності; – полегшує розширення.	– Вимагає додаткового зусилля для управління залежностями; – може призвести до зайвої складності.

Продовження таблиці 2.1

Рівні (Layered)	– Чітке розділення функціональності на різні рівні; – полегшує розширення та підтримку; – можливість заміни окремих рівнів без впливу на інші.	– Зайва складність, якщо рівні не відокремлені належним чином; – збільшений обсяг комунікації між рівнями.
Модель-Вид-Презентер (Model-View-Presenter, MVP)	– Легка тестовість; – полегшує розширення та підтримку; – поліпшена розділення відображення та логіки.	– Потребує більшої кількості класів; – може бути зайвою складністю для простих застосунків.
Сервісно-орієнтована архітектура (Service-Oriented Architecture, SOA)	– Розділення функціональності на сервіси; – полегшує інтеграцію з іншими системами; – сприяє повторному використанню та модульності.	– Потребує використання веб-сервісів або інших механізмів комунікації; – може вимагати додаткових зусиль для організації, керування.
Файл - Сервер	– Централізоване зберігання файлів; – загальний доступ до спільних файлів; – централізоване управління безпекою; – ефективне використання ресурсів.	– Залежність від доступності сервера; – обмежена масштабованість; – проблеми з конфліктами та одночасністю; – залежність від пропускної здатності.

Для розробки нашого застосунку використаємо архітектуру Файл-Сервер та шаблон проектування Модель-представлення-контролер (MVC).

Архітектура файл-сервера - це модель розподіленої системи, в якій файли зберігаються та керуються на централізованому сервері, а клієнтські пристрої звертаються до сервера для доступу до цих файлів. Така архітектура має деякі переваги:

– централізоване зберігання та керування файлами. Архітектура файл-сервера спрощує процес зберігання файлів, оскільки всі вони знаходяться на центральному сервері. Це дозволяє забезпечити єдину точку керування файлами, встановлювати права доступу, контролювати версії та забезпечувати цілісність даних;

– загальний доступ до файлів. Архітектура файл-сервера дозволяє різним користувачам та групам отримувати доступ до спільних файлів. Це сприяє співпраці та обміну інформацією між користувачами, спільними проектами та відділами;

– централізоване управління безпекою: Архітектура файл-сервера дозволяє встановлювати механізми безпеки на центральному сервері, що забезпечує контроль доступу до файлів та захист від несанкціонованого доступу. Адміністратор може встановлювати права доступу до файлів, шифрувати дані та встановлювати інші політики безпеки;

– ефективне використання ресурсів: Архітектура файл-сервера дозволяє ефективно використовувати ресурси мережі та обчислювальні потужності. Замість копіювання файлів на кожен клієнтський пристрій, файли зберігаються на сервері, що зменшує обсяг передачі даних та споживання ресурсів;

– зручність резервного копіювання та відновлення даних. Архітектура файл-сервера спрощує процес резервного копіювання та відновлення даних. Оскільки всі файли знаходяться на центральному сервері, можна легко налаштувати резервне копіювання файлів та систематично зберігати копії

даних. У разі втрати або пошкодження даних, можна відновити їх з резервної копії, що забезпечує безпеку та надійність даних;

– зменшення обсягу передачі даних. У архітектурі файл-сервера, клієнти отримують доступ до файлів, виконуючи запити до сервера. Замість передачі всього файлу клієнту, сервер може передавати лише необхідні частини або блоки даних. Це дозволяє зменшити обсяг передачі даних по мережі та покращити продуктивність, особливо при роботі з великими файлами.

– забезпечення централізованого оновлення. Адміністратор сервера може легко оновлювати та змінювати файли на сервері, а зміни автоматично відображаються на всіх підключених клієнтах. Це дозволяє швидко та одночасно розповсюджувати оновлення, патчі та зміни в програмному забезпеченні або даних.

– забезпечення резервного копіювання та відновлення. Завдяки централізованому зберіганню файлів на сервері, резервне копіювання та відновлення даних стають більш простими та ефективними. Адміністратор може регулярно створювати резервні копії файлів на сервері, що допомагає уникнути втрати даних у разі непередбачених ситуацій.

– підтримка централізованого моніторингу та керування. Завдяки архітектурі файл-сервера, можливо використовувати механізми централізованого моніторингу та керування. Адміністратор може відстежувати активність, стан та використання файлів на сервері, а також застосовувати глобальні налаштування та політики безпеки.

Незважаючи на переваги, архітектура файл-сервера також має деякі недоліки, серед яких можна виділити:

– залежність від сервера. У випадку недоступності сервера або проблем з мережевим з'єднанням, доступ до файлів може бути обмеженим або недоступним. Клієнтські пристрої залежать від працездатності сервера для отримання необхідних файлів;

– пропускна здатність мережі. Збереження та передача файлів через мережу може призвести до збільшення навантаження на мережу, особливо в разі великого обсягу даних або одночасного доступу багатьох користувачів. Це може вплинути на швидкість передачі файлів та загальну продуктивність мережі;

– вразливість до витоку даних. Оскільки всі файли знаходяться на центральному сервері, є певна загроза витоку конфіденційної інформації, якщо сервер стає об'єктом злому або несанкціонованого доступу. Для забезпечення безпеки даних на сервері потрібно вживати відповідні заходи безпеки, такі як шифрування даних та захист від несанкціонованого доступу;

– обмеження масштабованості. Архітектура файл-сервера може мати обмеження щодо масштабованості системи. При збільшенні кількості користувачів та обсягу файлів, сервер може стикатися з навантаженням та обмеженими ресурсами. Це може призвести до зниження продуктивності та часу відповіді, особливо при одночасному доступі багатьох користувачів;

– синхронізація та конфлікти. Завдяки спільному доступу до файлів, можуть виникати проблеми з синхронізацією та виникненням конфліктів. Якщо два або більше користувачі одночасно редагують один файл, можуть виникнути проблеми зі збереженням змін та вирішенням конфліктів. Необхідно встановити механізми синхронізації та контролю версій для уникнення таких проблем;

– залежність від мережевого з'єднання. Для доступу до файлів на сервері необхідне стабільне мережеве з'єднання. У випадку відсутності мережі або проблем зі з'єднанням, клієнтські пристрої можуть втратити доступ до файлів. Це може бути проблематично в ситуаціях, коли потрібно працювати з файлами без підключення до мережі.

Враховуючи переваги та недоліки розглянутих альтернатив, ми прийняли рішення вибрати архітектуру "файл-сервер" для нашої системи зберігання та керування файлами.

Архітектура файл-сервера надає нам централізоване зберігання файлів та єдину точку керування, що спрощує процес резервного копіювання, архівації та забезпечує контроль над файлами. Ми можемо легко встановлювати права доступу, проводити версіонування та інші керуючі дії для файлів на сервері.

Крім того, архітектура файл-сервера дозволяє спільний доступ до файлів, що розташовані на сервері, забезпечуючи співпрацю та одночасний доступ до актуальних версій файлів для спільних груп або організацій.

Централізоване управління безпекою є ще однією перевагою архітектури файл-сервера. Ми можемо застосовувати механізми автентифікації, авторизації та шифрування даних на сервері, що допомагає нам захистити конфіденційні дані та забезпечити контроль доступу до файлів.

Завдяки архітектурі файл-сервера ми можемо ефективно використовувати ресурси мережі та обчислювальні потужності, оскільки не потрібно копіювати файли на кожен клієнтський пристрій.

Таким чином, архітектура файл-сервера відповідає нашим потребам у зберіганні, керуванні та спільному доступі до файлів, забезпечуючи нам централізованість, безпеку та ефективне використання ресурсів.

2.2 Опис декомпозиції

Хороша декомпозиція та її синхронізація зі схемою процесу є важливими чинниками успіху роботи. Декомпозиція залежить від стилю управління проектом, організаційної культури, переваг, фінансових обмежень і від деяких інших труднощів визначення параметрів, специфічних для нашого проекту.

У будь-якому випадку декомпозиція повинна бути зрозуміла і дозволяти збирати проект в цілому з окремих робіт, забезпечувати керованість при його реалізації та розподіл відповідальності за кожною роботою тощо.

Розробка кваліфікаційної роботи на базі архітектури файл-серверу та шаблону проектування MVC передбачає декомпозицію проекту на окремі

компоненти для поліпшення керованості, розширюваності та підтримки коду. Описуючи декомпозиції, можна зосередитися на різних аспектах проекту, таких як модулі, функціональні блоки або шари.

Основні компоненти декомпозиції в нашому проекті включають наступні елементи:

– модель (Model). Цей компонент відповідає за управління даними та бізнес-логікою нашої програми. Він включає в себе структури даних, методи доступу до бази даних, операції зчитування та запису даних, а також логіку обробки та маніпулювання даними;

– представлення (View). Цей компонент відповідає за відображення даних користувачу та взаємодію з ним. Він включає в себе інтерфейс користувача, реалізацію графічного інтерфейсу та компоненти, що відображають дані з моделі користувачу;

– контролер (Controller). Цей компонент відповідає за керування взаємодією між моделлю та представленням. Він обробляє запити користувача, викликає відповідні методи моделі для отримання та обробки даних, а також оновлює представлення залежно від стану моделі;

– база даних. У нашому проекті ми взаємодіємо з базою даних, яка може бути частиною моделі. Вона забезпечує збереження даних, зчитування та запис до бази даних за допомогою SQL-запитів;

– компоненти маршрутизації. Цей компонент відповідає за маршрутизацію запитів користувача до відповідних контролерів. Він визначає шляхи URL і відповідні дії контролерів, що мають бути виконані при отриманні конкретного запиту;

– компоненти збереження файлів. У зв'язку з тим, що обрано архітектуру файл-сервера, можуть бути окремі компоненти, що відповідають за збереження та керування файлами на сервері. Ці компоненти можуть включати функції завантаження, збереження, видалення та оновлення файлів на сервері.

Кожен з цих компонентів може бути розроблений та реалізований окремо, з можливістю підключення і використання в рамках загальної архітектури файл-серверу та шаблону проектування MVC. Така декомпозиція сприяє модульності, розширюваності та підтримці коду, а також полегшує розподіл завдань між розробниками та забезпечує більшу гнучкість в розробці та супроводі проекту.

Розіб'ємо декомпозицію на три умовних рівня. Представлення декомпозиції кваліфікаційної роботи зображено на рис. 2.2:

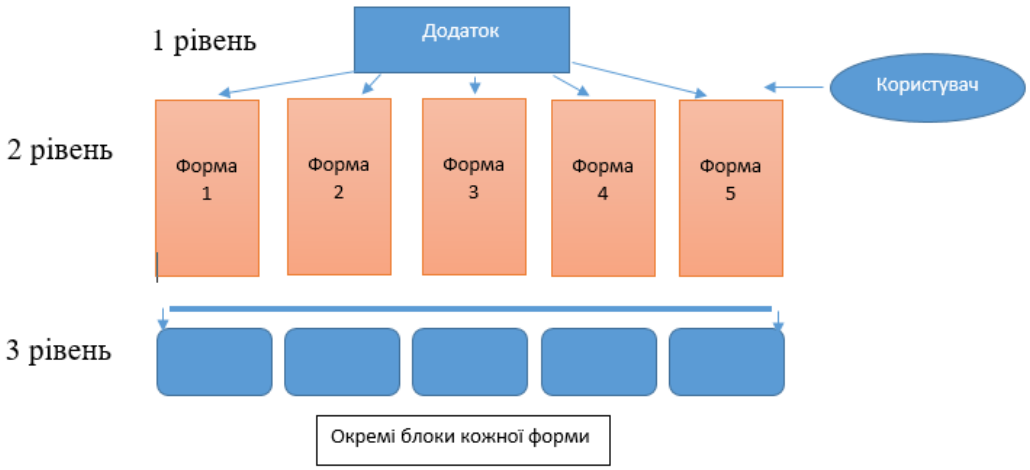


Рисунок 2.2 – Схематичне зображення рівнів декомпозиції

1 рівень - Крупні блоки або компоненти:

– модель (Model). Відповідає за управління даними та бізнес-логікою нашої програми. Цей компонент включає в себе структури даних, методи доступу до бази даних, операції зчитування та запису даних, а також логіку обробки та маніпулювання даними;

– представлення (View). Відповідає за відображення даних користувачу та взаємодію з ним. Цей компонент включає в себе інтерфейс користувача, реалізацію графічного інтерфейсу та компоненти, що відображають дані з моделі користувачу;

– контролер (Controller). Відповідає за керування взаємодією між моделлю та представленням. Цей компонент обробляє запити користувача, викликає відповідні методи моделі для отримання та обробки даних, а також оновлює представлення залежно від стану моделі.

2 рівень - Форми нашої програми:

Форма основного інтерфейсу: Відповідає за відображення головного вікна програми та інтерфейсу користувача. Ця форма містить меню, панель інструментів та область відображення даних.

Форма основного інтерфейсу є центральною точкою взаємодії користувача з програмою. Вона відображає головне вікно програми та надає засоби для навігації, виконання дій та відображення даних. Розглянемо детальніше компоненти, що будуть присутні на формі основного інтерфейсу:

Меню: Меню розташоване зверху форми і містить набір команд та опцій, доступних користувачеві. Воно організоване у вигляді деревоподібної структури, де кожен пункт меню може мати підпункти. Меню надає можливість користувачеві виконувати різноманітні дії, такі як створення, відкриття або збереження документів, налаштування програми, довідкову інформацію тощо.

Панель інструментів: Панель інструментів розташовується під меню і містить іконки або кнопки, які представляють часто використовувані функції або команди. Це дає користувачеві швидкий доступ до важливих опцій і дій. Кожна кнопка або іконка на панелі інструментів пов'язана з певною функцією програми і викликає відповідну дію при натисканні.

Область відображення даних: Ця область розташовується в центрі форми і призначена для відображення даних користувачеві. Вона буде містити таблиці, списки, дерева або інші контролери, які представляють і структурують дані. Область відображення даних може показувати список файлів, розділи документа, результати пошуку або будь-які інші дані, що пов'язані з функціональністю програми.

3 рівень - Наповнення та склад цих форм:

Форма основного інтерфейсу: Ця форма має різні елементи для взаємодії з даними. Наприклад, це можуть бути таблиці для відображення списків таблиць бази даних або файлів, кнопки для створення нового документу, функції пошуку, фільтрації та сортування даних тощо. Користувач може взаємодіяти з цими елементами, здійснюючи різні дії, такі як створення, редагування або видалення документів.

На формі основного інтерфейсу будуть наступні елементи:

– таблиці бази даних. Вона відображає список документів, які зберігаються на файл-сервері. Користувач може переглядати дані цих таблиць, а також виконувати дії, такі як видалення або редагування даних;

– кнопка "Створити". Ця кнопка дозволяє користувачеві створювати нові документи. Після натискання на неї відкривається відповідна форма, де користувач може ввести необхідну інформацію для створення нового документа;

– функція пошуку: Ця функція дозволяє користувачу шукати дані в таблиці бази даних за заданими критеріями. Вона може включати поля для введення пошукового запиту та кнопку;

– форма попередження про помилку: Ця форма відображає повідомлення про помилку, якщо виникають проблеми під час взаємодії з базою даних або іншими операціями. Вона може містити текстове повідомлення про помилку та кнопку для закриття форми або повтору операції.

для форми створення/редагування можуть бути наступні елементи:

– поле введення назви документа. Користувач може ввести назву документа в це поле;

– поле введення дати. Користувач може вибрати дату створення або зміни документа за допомогою календаря або ввести її вручну. Автоматично підтягується сьогоднішня дата, яку можна змінити;

– випадаючий список клієнтів, працівників. Користувач може вибрати дані зі списку наявних або додати нового;

– поле для введення опису: Користувач може ввести додатковий опис або коментар до документа;

– кнопка "Зберегти": Після заповнення полів користувач може натиснути цю кнопку для збереження документа до бази даних;

– кнопка "Скасувати": Користувач може натиснути цю кнопку, щоб скасувати операцію створення або редагування документа та закрити форму.

2.3 Опис залежностей

Для повноцінного розуміння залежностей декомпозиції в нашій архітектурі файл-сервера з використанням шаблону проектування MVC, розглянемо детальний опис залежностей між компонентами.

Модель (Model). Залежності від бази даних: Модель взаємодіє з базою даних для отримання та збереження даних. Вона використовує методи доступу до бази даних, такі як SQL-запити, для виконання операцій читання, запису, оновлення та видалення даних.

Представлення (View). Залежність від моделі: Представлення отримує дані від моделі для їх відображення користувачеві. Воно використовує методи моделі для отримання необхідних даних та відображення їх у відповідних компонентах інтерфейсу користувача.

Контролер (Controller). Залежність від моделі: Контролер взаємодіє з моделлю для отримання та зміни даних. Він викликає методи моделі для виконання операцій з даними, таких як отримання списку файлів, збереження файлу або видалення файлу.

База даних. Залежність моделі від бази даних: Модель використовує функціональність бази даних для зчитування та запису даних. Вона може використовувати SQL-запити або спеціалізовані бібліотеки для взаємодії з базою даних.

Компоненти маршрутизації. Залежність контролера від компонентів маршрутизації: Контролер отримує запити від компонентів маршрутизації, які визначають шляхи URL і відповідні дії контролера. Компоненти маршрутизації відправляють запит.

Компоненти збереження файлів. Залежність моделі від компонентів збереження файлів: Модель використовує компоненти збереження файлів для здійснення операцій з файлами, таких як завантаження, збереження, видалення та оновлення файлів на сервері. Компоненти збереження файлів надають методи інтерфейсу для взаємодії з файловою системою сервера.

Модель взаємодіє з представленням. Модель надає дані, які відображаються відповідними компонентами представлення. Вона оновлює представлення при зміні даних або стану.

Представлення взаємодіє з контролером. Представлення відправляє події та запити контролеру при взаємодії користувача з інтерфейсом. Контролер обробляє ці запити та виконує відповідні дії.

Контролер взаємодіє з моделлю. Контролер викликає методи моделі для отримання або зміни даних. Він передає дані з моделі до представлення для їх відображення.

Компоненти маршрутизації взаємодіють з контролером. Компоненти маршрутизації визначають шляхи URL і відповідні дії контролера. Вони перенаправляють запити від користувача до відповідного контролера для обробки.

Компоненти збереження файлів взаємодіють з моделлю. Компоненти збереження файлів забезпечують функціональність збереження та управління файлами на сервері. Модель використовує ці компоненти для роботи з файлами у контексті своїх операцій.

Залежності в архітектурі вказують на взаємозв'язок між компонентами та вказують, які компоненти використовуються в інших компонентах для виконання певних операцій або отримання необхідних даних. Залежності між

компонентами допомагають забезпечити логічну структуру програми та впорядкувати взаємодію між її частинами.

Зокрема, у нашій архітектурі файл-сервера з використанням шаблону проектування MVC, основні залежності включають:

– Модель також взаємодіє з компонентами збереження файлів, оскільки вона використовує їх функціональність для здійснення операцій з файлами на сервері;

– контролер отримує запити від компонентів маршрутизації, які визначають шляхи URL та відповідні дії контролера;

– представлення взаємодіє з контролером, відправляючи події та запити при взаємодії користувача з інтерфейсом;

– компоненти маршрутизації взаємодіють з контролером, перенаправляючи запити від користувача до відповідного контролера для обробки;

– компоненти збереження файлів використовуються моделлю для здійснення операцій з файлами, таких як завантаження, збереження, видалення та оновлення файлів на сервері.

Ці залежності визначають взаємодію між компонентами системи та покажуть, як кожен компонент використовується для виконання певних функцій та отримання необхідних даних. Вони впорядковують логічну структуру програми і сприяють покращенню її розширюваності, підтримки та модульності.

2.4 Опис інтерфейсів

Тема нашої кваліфікаційної роботи є "Програмне забезпечення для автоматизації роботи шиномонтажу". В рамках цієї теми буде використано графічний тип інтерфейсу.

Графічний інтерфейс користувача - різновид інтерфейсу користувача, в якому елементи інтерфейсу (меню, кнопки, значки, списки і т. п.), представлені користувачеві на дисплеї, виконані у вигляді графічних зображень.

На відміну від інтерфейсу командного рядка, у GUI користувач має довільний доступ (за допомогою пристроїв введення – клавіатури, миші, джойстика тощо) до всіх видимих екранних об'єктів (елементів інтерфейсу) та здійснює безпосереднє маніпулювання ними. Найчастіше елементи інтерфейсу в GUI реалізовані на основі метафор та відображають їх призначення та властивості, що полегшує розуміння та освоєння програм непідготовленими користувачами.

Графічний інтерфейс користувача є частиною інтерфейсу користувача і визначає взаємодію з користувачем на рівні візуалізованої інформації.

Огляд графічного інтерфейсу (GUI) в нашій кваліфікаційній роботі включає докладний опис і пояснення про наступні аспекти.

У розділі, що розглядається, акцентується увага на важливості та ролі графічного інтерфейсу у програмному продукті для автоматизації роботи шиномонтажу. Пояснюється, що графічний інтерфейс є основним засобом взаємодії між користувачем та програмою, забезпечуючи зручний та легко зрозумілий спосіб керування робочим процесом шиномонтажу, спрощуючи використання функцій програми і забезпечуючи зручний доступ до необхідної інформації.

Також, у розділі будуть розглянуті основні візуальні елементи та компоненти, що використовуються в графічному інтерфейсі. До них відносяться кнопки, текстові поля, списки, меню, форми підказок. Ці елементи призначені

для забезпечення зручного доступу до даних, інформації та для забезпечення користувача можливістю комфортно працювати з програмою.

Організація і розташування елементів. В даному розділі пояснюється принцип організації та розташування елементів в графічному інтерфейсі. Зазначається важливість логічного розміщення елементів, створення зручної та інтуїтивно зрозумілої структури інтерфейсу для користувача. Дані з бази в нас будуть відображать в окремому для кожної таблиці DataGridView, поля для введення, редагування та видалення згуртовані та для кожного пункту є своя вкладка, при введенні даних є обмеження та підказки щоб коректно зберігати інформацію.

У графічному інтерфейсі використання кольорів має велике значення для створення певної атмосфери, передачі настрою та забезпечення зручного сприйняття інформації. При виборі кольорової палітри враховуються психологічні аспекти кольорів, їх взаємозв'язок та відповідність до тематики програмного продукту. Наприклад, для створення спокійної та надійної атмосфери можуть бути використані відтінки синього або зеленого кольору, а для акцентів та підсилення уваги - яскраві та насичені кольори. Важливо дотримуватись розумного балансу, щоб уникнути візуального перевантаження або незручностей для користувачів.

Крім того, контрастність між кольорами графічного інтерфейсу має велике значення для забезпечення читабельності тексту та візуальної відмінності між різними елементами. Для полегшення сприйняття будуть використані відповідні кольорові позначення для різних станів елементів, що дозволить користувачеві легко розрізняти їх.

Окрім кольорів, стиль графічного інтерфейсу також грає важливу роль. Він повинен бути однорідним, зрозумілим та естетичним. Для досягнення цього використовуються відповідні шрифти, розміри, вирівнювання, форми та інші елементи стилю. Це допомагає створити єдиний та цілісний вигляд інтерфейсу, що забезпечує зручність його використання.

Враховуючи всі ці фактори, графічний інтерфейс буде ретельно розроблений з метою забезпечення гармонії кольорів та стилю, що максимально забезпечує зручність, комфорт та задоволення користувачів при використанні програмного продукту для автоматизації роботи шиномонтажу.

2.5 Аналіз та вибір технологій і методів реалізації застосунка.

Застосунок, який може бути програмним забезпеченням, працює самостійно на комп'ютері, мобільному пристрої або у браузері. Існують різні типи застосунків, але загалом вони виконують функції для користувача або інших програм/застосунків. Залежно від типу, застосунки можуть вимагати постійного з'єднання з Інтернетом або працювати офлайн. Наприклад, застосунки для робочого столу призначені для використання на комп'ютерах і можуть мати більше можливостей, оскільки вони розраховані на роботу з повним комплектом пристроїв, таких як миша, клавіатура і великий монітор. Мобільні застосунки, з свого боку, пристосовані до роботи на мобільних пристроях з обмеженими розмірами екрану та інтерфейсом, доступним для керування за допомогою пальців. Веб-застосунки, у свою чергу, працюють у веб-браузері і залежать від швидкості Інтернет-з'єднання.

Desktop застосунки – це програми, які вимагають операційну систему настільного комп'ютера для своєї роботи. Вони встановлюються в систему за допомогою спеціального інсталятора та використовують ресурси комп'ютера для своєї роботи.

Однією з основних характеристик таких програм є їхній здатність працювати автономно без доступу до Інтернету. Сучасні застосунки, проте, зазвичай пропонують можливість підключення до Інтернету, що дозволяє використовувати різноманітні пристрої, залучати інших користувачів до проекту та отримувати оновлення програми без необхідності ручної переустановки.

Багато компаній приділяють особливу увагу безпеці і не бажають, щоб застосунки мали доступ до Інтернету. У таких випадках розробка десктопних застосунків може бути виконана "під ключ", а підключення може бути обмежене до захищеної локальної мережі.

Загалом вважається, що такий тип програм є швидшим та має значно більші можливості. Це справді так, проте варто зазначити, що ці параметри в значній мірі залежать від характеристик самого комп'ютера. Якщо спробувати встановити цей софт на менш потужне обладнання, то це швидко стане очевидним особистим досвідом.

Веб-програми представляють собою програмне забезпечення або застосунки, які можна відкрити в будь-якому веб-браузері. Зовнішній інтерфейс веб-програми розробляється з використанням мов програмування, таких як HTML, CSS та JavaScript, які підтримуються всіма популярними веб-браузерами, такими як Opera, Chrome, Mozilla та Yandex. Щодо написання серверної частини (Back-end), використовуються різні мови програмування або фреймворки, такі як Python, PHP, Ruby та Java.

Завдяки веб-застосункам можна заощадити кошти, оскільки вони не вимагають підписки або покупки ліцензій, а можуть функціонувати як SaaS-сервіс, що є набагато доступнішим з фінансової точки зору.

Мобільні застосунки є спеціально розробленим програмним забезпеченням для певної мобільної платформи, такої як iOS, Android, Windows Phone та інші. Вони призначені для використання на смартфонах, фаблетах, планшетах, розумних годинниках та інших мобільних пристроях.

Мобільні програми розробляються з використанням мов програмування високого рівня, які потім компілюються в машинний код операційної системи, щоб забезпечити максимальну продуктивність.

Розробка мобільних застосунків має свої особливості: мобільні пристрої працюють від батареї та оснащені менш продуктивними процесорами, ніж персональні комп'ютери. Крім того, сучасні смартфони та планшети часто

мають додаткові пристрої, такі як гіроскопи, акселерометри та фотокамери, що відкривають унікальні можливості для розширення функціоналу застосунків.

Зазвичай, мобільні пристрої продаються з вже встановленими програмами, а інші можна завантажити за бажанням користувача зі спеціалізованих сервісів, таких як Apple App Store, Google Play, Windows Phone Store та інші, як платні, так і безкоштовні.

З урахуванням усіх доступних методів було вирішено розробляти саме настільні застосунки. Для розробки настільного застосунку, який автоматизує обслуговування клієнтів шиномонтажу, можна використовувати різні технології та мови програмування, такі як C++, C#, Java, Visual Basic, Microsoft Access та Microsoft SQL Server.

Мова програмування C++ була розроблена на основі мови C (Сі). Спочатку вона була створена як розширення основної мови C з можливістю об'єктно-орієнтованого програмування.

C++ має декілька переваг, серед яких можна виділити:

Продуктивність: Мова програмування C++ забезпечує високу швидкість виконання коду завдяки своїй обчислювальній потужності.

Низьке навантаження: Використання мови C++ не ускладнює програми, що дозволяє їх ефективно працювати навіть на старіших пристроях.

Універсальність: C++ є кроссплатформенною мовою, яка підходить для різних цілей розробки. Вона не є вузькоспеціалізованою, як деякі інші мови програмування.

Популярність: C++ є традиційною мовою для розробки програмного забезпечення та різних видів софту. Компілятори для C++ доступні на різних операційних системах, а програми, написані на цій мові, зазвичай легко переносяться з однієї платформи на іншу.

Що стосується мови програмування C#, вона є однією з найбільш швидко зростаючих і популярних мов програмування. C# є модифікацією мови C, розробленою компанією Microsoft. Вона створена з метою створення

універсального інструменту для розробки програмного забезпечення для різних пристроїв і операційних систем.

Таким чином, після аналізу методів та технологій у попередньому розділі і з урахуванням всіх їх недоліків та переваг, було прийнято рішення розробляти настільний застосунок з використанням мови програмування C# та програмного забезпечення Microsoft SQL Server.

2.6 Висновки за розділом 2

У розділі 2 "Проектування програмного забезпечення" було проведено аналіз і вибір типу архітектури та шаблонів проектування для нашого програмного продукту. Під час цього аналізу були враховані вимоги та потреби користувачів, а також здійснено опис декомпозиції програми.

Декомпозиція включає розбиття програмного продукту на окремі модулі, що дозволяє полегшити розробку та підтримку системи. Були визначені залежності між цими модулями, що дозволяє керувати взаємозв'язками та взаємодією між ними.

Детальний опис інтерфейсів був проведений з метою забезпечення зручної та логічної взаємодії користувача з програмним продуктом. Це включало визначення вхідних та вихідних даних, функціональності та обмежень інтерфейсів.

Крім того, проведений аналіз та вибір технологій і методів реалізації застосунку. В результаті цього аналізу були визначені такі технології, як мова програмування C# 7.0 та платформи .NET та .NET Core. Крім того, були використані такі інструменти, як SQL Server, Microsoft SQL Management Studio та Crystal Report.

Загальний висновок у тому, що під час проектування програмного забезпечення були проведені детальні аналізи, вибори технологій та методів, а також описані архітектура, декомпозиція, залежності та інтерфейси

програмного продукту. Всі ці етапи підготовки є важливими для успішної реалізації програми та задоволення потреб користувачів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Детальне проектування модулів

Детальне проектування модулів є критичним етапом у розробці програмного продукту. На цьому етапі проводиться розробка детальних планів і специфікацій для кожного модуля, що визначають його функціональність, інтерфейси, структуру та спосіб взаємодії з іншими модулями.

Під час детального проектування модулів виконуються наступні кроки.

Визначення функцій модуля. Кожен модуль повинен бути чітко визначений щодо його функціональності. Це означає визначення вхідних та вихідних даних, операцій, які він виконує, та очікуваних результатів.

Проектування інтерфейсів. На цьому етапі визначаються методи та параметри, які модуль отримує та повертає. Інтерфейс модуля повинен бути чітким і зрозумілим для інших розробників, які будуть використовувати цей модуль.

Розробка структури модуля. Визначається структура модуля, яка включає класи, функції, змінні та їх взаємозв'язки. Можуть бути використані діаграми класів або інші візуальні засоби для відображення структури модуля.

Визначення логіки модуля. Визначається логіка обробки даних та виконання операцій в межах модуля. Це включає опис алгоритмів, управління потоками, обробку помилок та інші аспекти, пов'язані з функціональністю модуля.

Детальне проектування модулів ПЗ включає ряд етапів, кожен з яких має свої назви та структурні елементи. Опишемо ці етапи та пов'язані з ними структурні елементи з використанням діаграм класів, послідовності, переходів станів.

Етап аналізу вимог є важливим етапом у процесі розробки програмного продукту. Цей етап передує детальному проектуванню модулів і має на меті зрозуміти та уточнити вимоги до системи з точки зору функціональності, інтерфейсів та поведінки системи. Одним із інструментів, що можна використовувати під час аналізу вимог, є діаграма послідовності.

Діаграма послідовності є візуальним інструментом, який дозволяє моделювати взаємодію об'єктів та послідовність повідомлень між ними. Вона допомагає уявити та зрозуміти, як об'єкти взаємодіють між собою для досягнення певної функціональності. Діаграму послідовності зображено на рис. 3.1:

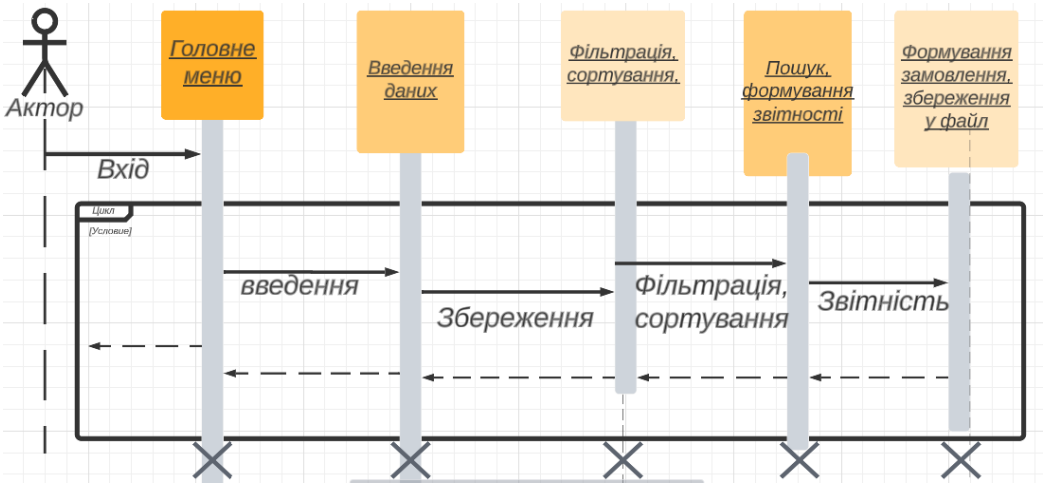


Рисунок 3.1 – діаграма послідовностей

У нашій роботі ми використовуємо діаграму станів для візуалізації поведінки системи шиномонтажу у різних станах. Діаграма станів дозволяє нам моделювати і показувати, як система реагує на зміни вхідних подій і переходить з одного стану в інший.

Завдяки діаграмі станів ми можемо ідентифікувати та відобразити різні стани системи шиномонтажу, наприклад, "Головне меню", "Внесення даних", "Сортування результатів", "Генерація звіту" і т.д. Кожен стан відображає конкретний стан системи на певному етапі взаємодії з користувачем або обробки даних.

Діаграма станів також дозволяє нам показати переходи між станами, які відбуваються в системі внаслідок певних подій чи дій користувача. Наприклад, зміна стану від "Головного меню" до "Внесення даних" може відбутися після того, як користувач обрав опцію "Внести інформацію" у головному меню.

Така діаграма дозволяє нам зрозуміти логіку роботи системи, відстежувати послідовність подій та дій, а також ідентифікувати можливі стани, переходи та умови, що впливають на поведінку системи. Це важливо для проектування та розробки програмного забезпечення, а також для визначення вимог до системи та взаємодії з користувачами. Діграму зображено на рисунку 3.2:

Рис. 3.2 – діаграма Станів

Етап проектування деталей модулів є важливим етапом у розробці програмного продукту. На цьому етапі розробляються детальні плани та специфікації для кожного модуля, що визначають його функціональність, інтерфейси та спосіб взаємодії з іншими модулями. Діаграма взаємодії може бути корисним інструментом для візуалізації цих аспектів.

Діаграма взаємодії, також відома як діаграма послідовності, показує послідовність взаємодій між об'єктами часової осі. Вона дозволяє проектувати

та розуміти, як об'єкти спілкуються між собою для досягнення певного результату. Діаграму зображено на рис. 3.3:

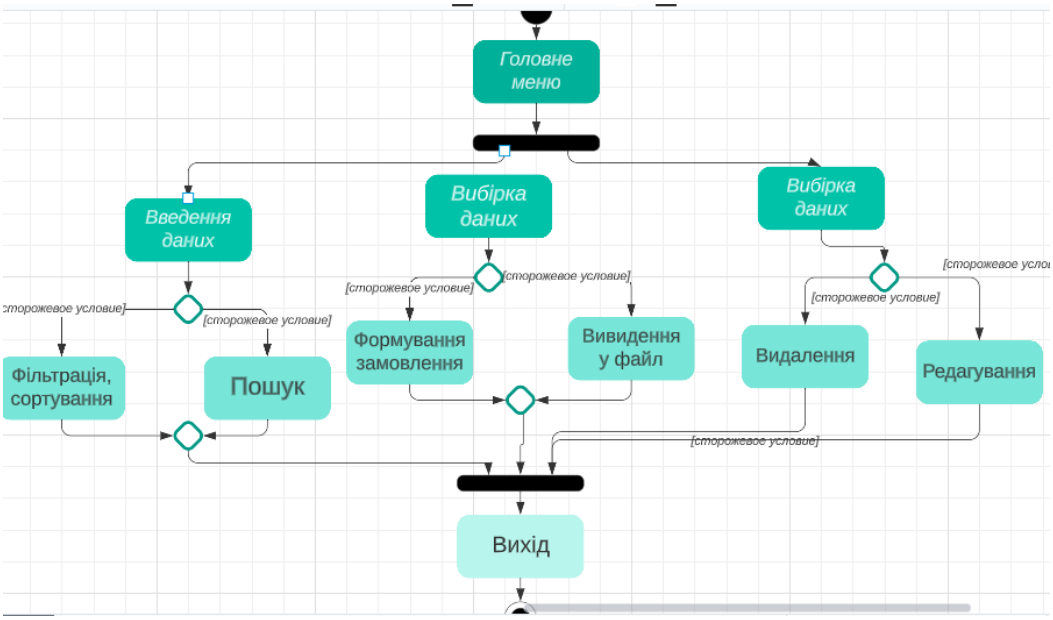


Рисунок 3.3 – діаграма взаємодії

Етап проектування інтерфейсу є важливим етапом у розробці програмного продукту, який включає визначення структури, функціональності та взаємодії між класами. Діаграма класів є ефективним інструментом для візуалізації цих аспектів. Нижче наведено загальний опис етапу проектування інтерфейсу за допомогою діаграми класів.

Визначення класів. Спочатку необхідно визначити класи, які будуть присутні в системі. Класи відображають основні об'єкти, які будуть використовуватися в програмному продукті.

Визначення взаємозв'язків. Далі необхідно визначити взаємозв'язки між класами. Це можуть бути асоціації, агрегації, композиції або спадкування, які показують, які класи взаємодіють між собою.

Визначення методів та атрибутів. Для кожного класу визначаються методи (операції), які він може виконувати, а також атрибути (властивості), які він має. Це допомагає визначити функціональність кожного класу.

Моделювання залежностей. Додатково можна відобразити залежності між класами, такі як залежності від інтерфейсів або використання інших класів.

Це може бути корисно для визначення, які класи взаємодіють між собою на рівні інтерфейсу.

Діаграму класів зображено на рис. 3.4:

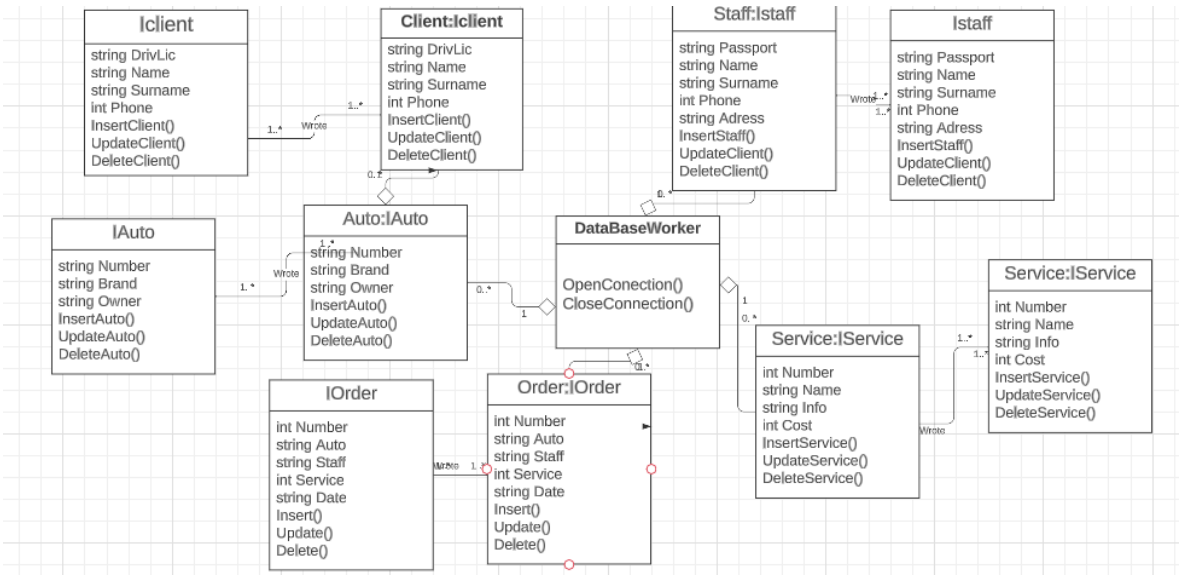


Рисунок 3.4 – діаграма класів

Детальне проектування модулів грає важливу роль у виявленні можливих проблем, помилок або недоліків ще до початку реалізації проекту. Цей етап дозволяє своєчасно вносити корективи та оптимізувати роботу модулів, забезпечуючи високу якість та надійність програмного продукту.

Відділення важливих деталей проекту в окремі модулі та їх детальне пророблення допомагають перетворити початкову концепцію і вимоги до програмного продукту в конкретні плани та специфікації. Це в свою чергу сприяє ефективному процесу розробки, забезпечує високу якість та надійність програмного продукту, а також спрощує подальшу реалізацію та тестування модулів.

3.2 Програмна реалізація модулів

Програмна реалізація модулів включає в себе процес перетворення детального проекту модулів у виконуваний код. Цей етап включає написання програмного коду, вибір технологій та інструментів реалізації, тестування та налагодження модулів.

Програмна реалізація починається зі створення основної структури модулів, включаючи класи, функції та методи, необхідні для виконання функціональності модуля. Залежно від вибраних технологій, буде використана мова програмування C#.

Для розробки застосунку кваліфікаційної роботи буде використаний об’єктно-орієнтований підхід. Об'єктно-орієнтований підхід є популярним і ефективним підходом до розробки програмного забезпечення. Ми вибрали цей підхід для нашого застосунку з кількох причин.

У нашому застосунку об'єктно-орієнтований підхід дозволяє нам:

– організувати функціональність в окремі класи та об'єкти, що полегшує розуміння та керованість коду;

– використовувати інкапсуляцію для захисту даних та функціональності об'єктів, забезпечуючи консистентність та безпеку;

– використовувати спадкування для створення загальних класів, що дозволяють повторне використання коду та легке впровадження нових функціональних можливостей;

– використовувати поліморфізм для заміщення інтерфейсів та забезпечення гнучкості та розширюваності системи;

– легко тестувати окремі класи та об'єкти, що дозволяє забезпечити якість та стабільність програмного забезпечення;

– забезпечити розбиття задач на менші, незалежні модулі, що дозволяє працювати паралельно та зосереджуватися на конкретних аспектах розробки.

Опишемо клас для роботи з базою даних. Що має метод який відкриває підключення до нашої бази даних, код:

```
class DataBaseWorker
    {
        public SqlConnection sqlConnection = new SqlConnection(@"Data
Source=DESKTOP-CK6RHVN;Initial Catalog=AutoService;Integrated Security=True");
        public void OpenConnection()
        {
            if(sqlConnection.State == System.Data.ConnectionState.Closed)
            {
                sqlConnection.Open();
            }
        }
        public void CloseConnection()
        {
            if (sqlConnection.State == System.Data.ConnectionState.Open)
            {
                sqlConnection.Close();
            }
        }
```

Опис класів та інтерфейсів з використанням поліморфізму та наслідування.

Клас Auto і інтерфейс IAuto.

Опис: Клас Auto представляє автомобіль, а IAuto є відповідним інтерфейсом.

Код інтерфейсу IAuto:

```
interface IAuto
    {
        string Number { get; set; }
        string Brand { get; set; }
        string Owner { get; set; }
        void InsertAuto(string Number, string Brand, string Owner,
SqlConnection sqlConnection);
        void UpdateAuto(string Number, string Brand, string Owner,
SqlConnection sqlConnection);
        void DeleteAuto(string Number, SqlConnection sqlConnection);
    }
```

Методи:

InsertAuto(): Додає новий запис про автомобіль до бази даних.

Код методу InnsertAuto():

```
public async void InsertAuto(string Number, string Brand, string Owner,
SqlConnection sqlConnection)
        {
            SqlCommand command2 = new SqlCommand("INSERT INTO [Auto]
(Number, Brand, Owner)VALUES(@Number,@Brand,@Owner)", sqlConnection);
            command2.Parameters.AddWithValue("Number", Number);
            command2.Parameters.AddWithValue("Brand", Brand);
```

```
                command2.Parameters.AddWithValue("Owner", Owner);
                await command2.ExecuteNonQueryAsync();
            }
```

UpdateAuto(): Оновлює існуючий запис про автомобіль у базі даних.

Код методу UpdateAuto():

```
        public async void UpdateAuto(string Number, string Brand, string Owner,
SqlConnection sqlConnection)
            {
                try
                {
                    SqlCommand command = new SqlCommand("UPDATE [Auto] SET
[Brand]=@Brand, [Owner]=@Owner WHERE [Number]=@Number", sqlConnection);
                    command.Parameters.AddWithValue("Number", Number);
                    command.Parameters.AddWithValue("Brand", Brand);
                    command.Parameters.AddWithValue("Owner", Owner);
                    await command.ExecuteNonQueryAsync();
                }
                catch (Exception ty)
                {
                    MessageBox.Show(ty.Message);
                }
            }
```

DeleteAuto(): Видаляє запис про автомобіль з бази даних.

Код методу DeleteAuto():

```
        public async void DeleteAuto(string Number, SqlConnection sqlConnection)
            {
                SqlCommand command = new SqlCommand("DELETE FROM[Auto] WHERE
[Number]=@Number", sqlConnection);
                command.Parameters.AddWithValue("Number", Number);

                await command.ExecuteNonQueryAsync();
            }
```

Методи додавання, редагування та видалення для інших класів описані аналогічно, згідно полів які входять у таблиці бази даних.

Клас Client і інтерфейс IClient.

Опис: Клас Client представляє клієнта ibyjvjynf;e, а IClient є відповідним інтерфейсом.

Методи:

InsertClient(): Додає новий запис про клієнта до бази даних.

UpdateClient(): Оновлює існуючий запис про клієнта у базі даних.

DeleteClient(): Видаляє запис про клієнта з бази даних.

Клас Staff і інтерфейс IStaff.

Опис: Клас Staff представляє співробітника, а IStaff є відповідним інтерфейсом.

Методи:

InsertStaff(): Додає новий запис про співробітника до бази даних.

UpdateStaff(): Оновлює існуючий запис про співробітника у базі даних.

DeleteStaff(): Видаляє запис про співробітника з бази даних.

Клас Service і інтерфейс IService.

Опис: Клас Service представляє послуги сервісу, а IService є відповідним інтерфейсом.

Методи:

InsertService(): Додає новий запис про сервісні послуги до бази даних.

UpdateService(): Оновлює існуючий запис про сервісні послуги у базі даних.

DeleteService(): Видаляє запис про сервісні послуги з бази даних.

Клас Order і інтерфейс IOrder.

Опис: Клас Order представляє замовлення, а IOrder є відповідним інтерфейсом.

Методи:

InsertOrder(): Додає нове замовлення до бази даних.

UpdateOrder(): Оновлює існуючий запис про замовлення у базі даних.

DeleteOrder(): Видаляє запис про замовлення з бази даних.

У нашому проекті використовуються наступні технології ООП:

Поліморфізм:

Відношення поліморфізму відображається у використанні однакових назв методів (Insert(), Update(), Delete()) у різних класах, які реалізують відповідні інтерфейси (IAuto, IClient, IStaff, IService, IOrder).

Це означає, що кожен клас може мати власну реалізацію цих методів, відповідну до його особливостей, але з точки зору зовнішнього коду ми можемо використовувати ці методи однаковим чином, без необхідності звертатися до конкретного класу.

Наслідування:

Відношення наслідування відображається у структурі класів. Наприклад, класи Auto, Client, Staff, Service і Order успадковують (розширюють) функціональність від відповідних інтерфейсів IAuto, IClient, IStaff, IService, IOrder.

Це означає, що кожен з цих класів має доступ до методів, оголошених у відповідному інтерфейсі, та повинен реалізувати їх відповідно до свого контексту.

Таким чином, використання технологій ООП у нашому проекті дозволяє нам створити ієрархію класів, використовувати поліморфізм для заміщення методів та забезпечувати розширюваність та гнучкість системи. Кожен клас має свою функціональність, але вони можуть використовуватися з використанням загального інтерфейсу, що спрощує взаємодію з ними та розробку коду.

Під час програмної реалізації враховуються деталі та вимоги, визначені на етапі аналізу та проектування. Реалізовано функціональність, передбачену для кожного модуля, та забезпечують правильну взаємодію між модулями в рамках програмного продукту.

3.3 Детальне проектування даних

Для забезпечення збереження даних ми використаємо реляційну базу даних. При проектуванні бази даних детально розглянемо вхідні дані. Ми визначимо основні структури даних, необхідні для виконання операцій в програмі.

Вхідна інформація даної кваліфікаційної роботи являє собою певний набір даних, який може містити різноманітні дані про шиномонтаж з якими користувач може взаємодіяти та використовувати.

Вхідну інформацію можна поділити на такі категорії:

– інформація про клієнтів шиномонтажу;

– інформація про автомобілі;

– інформація про персонал;

– інформація про послуги;

– інформація про замовлення.

В подальшому користувач має можливість переглядати цю інформацію, здійснювати пошук та фільтрацію по даній інформації та формувати звітність.

Для мінімізації помилок та накладок при додаванні інформації пов'язаної із зв'язками у базі данних розглянемо алгоритм на рис. 3.5:

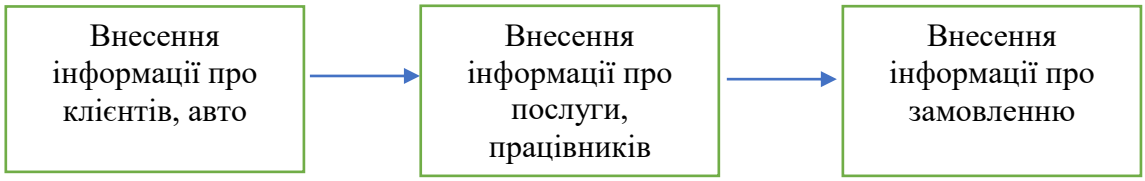


Рисунок 3.5 – Алгоритм послідовності введення даних

У предметній області нашого проекту присутні такі об'єкти: клієнти шиномонтажу, автомобілі, персонал, доступні послуги, замовлення.. Для забезпечення відповідності першій нормальній формі (1НФ) до кожної таблиці буде додано унікальний ідентифікатор. У таблиці "Клієнти" ключовим полем буде номер водійського посвідчення, що спрощує перегляд таблиць користувачем. Кожен водій може мати лише одне посвідчення. У таблиці "Авто", яка містить інформацію про автомоблії клієнтів, є поле номеру посвідчення власника і може містити тільки одного власника. А в одного власника може бути декілька автомобілів. Таким чином, між таблицями "Клієнти" та "Авто" існує зв'язок один до багатьох, який можна побачити на рис. 3.6:

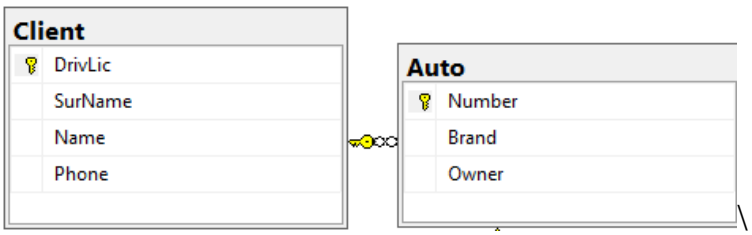


Рисунок 3.6 – зв’язок між таблицями "Клієнти" та "Авто"

По одному авто може бути декілька замовлень. Таким чином відношення між таблицями "Авто" та "Замовлення" містить зв’язок один до багатьох, який можна побачити на рис. 3.7:

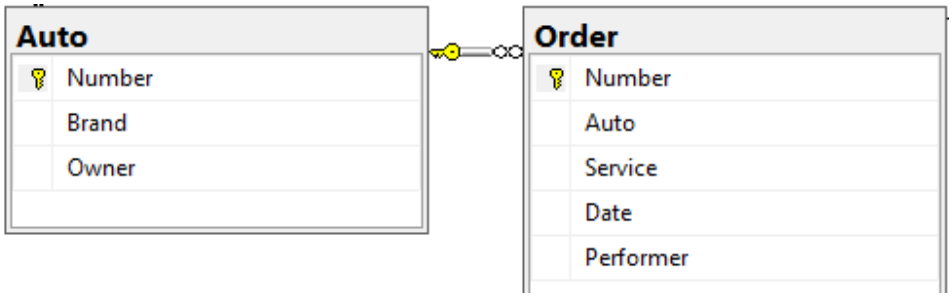


Рисунок 3.7 – зв’язок між таблицями "Авто" та "Замовлення"

По одному працівнику може бути декілька замовлень. Таким чином відношення між таблицями "Працівники" та "Замовлення" містить зв’язок один до багатьох, який можна побачити на рис. 3.8:

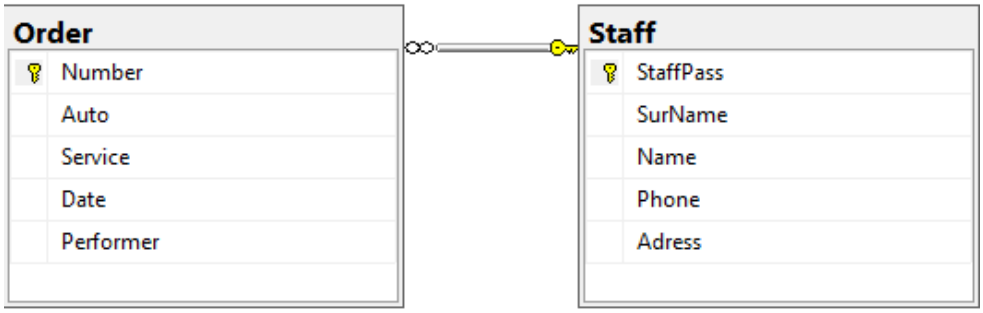


Рисунок 3.8 – зв’язок між таблицями "Працівники" та "Замовлення"

Зв’язок між таблицями "Послуги" та "Замовлення" також буде один до багатьох, який можна побачити на рис. 3.9:

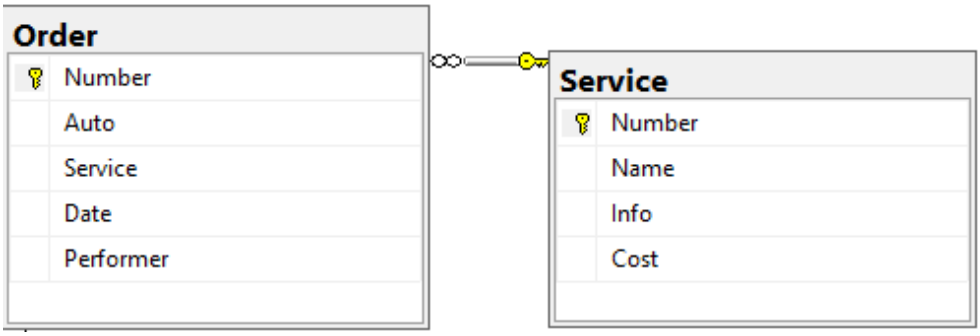


Рисунок 3.9 – зв’язок між таблицями "Послуги" та "Замовлення"

Схематичний результат готової бази даних шиномонтажу зображено на рис. 3.10:

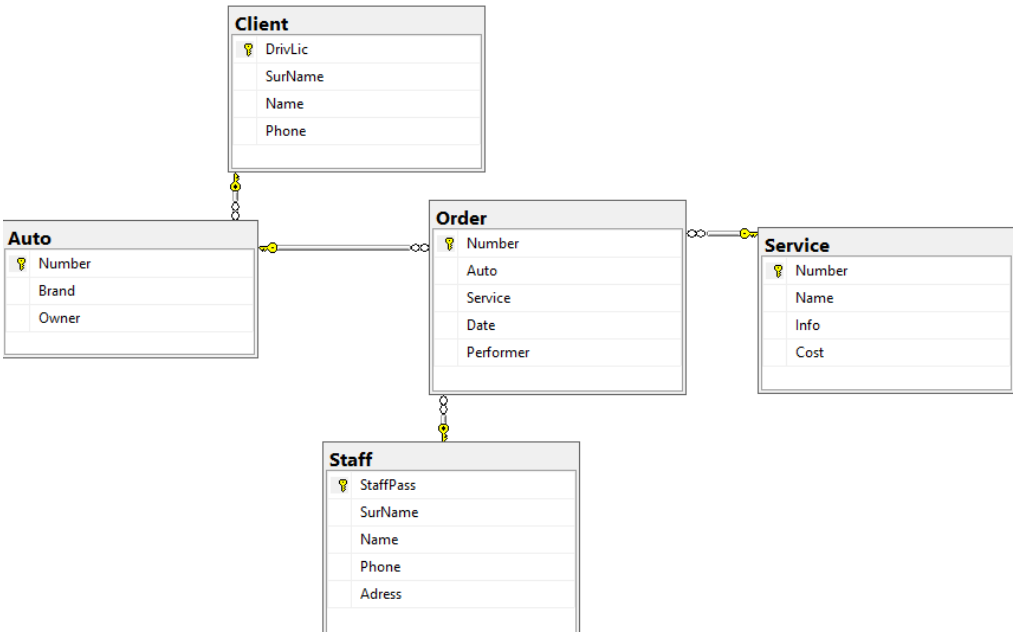


Рисунок 3.3.6 – схема готової бази даних

Результуючою інформацією для користувача є результат збережених даних про шиномонтаж, доданих у програму, з використанням фільтрів та сортувань заданих користувачем та сформованих у звіт. Уся інформація зберігається в різних таблицях бази даних.

Інформація про клієнтів містить номер посвідчення, прізвище, ім’я і номер телефону.

Інформація про авто містить номер авто, модель, та посвідчення власника.

Інформація про працівників містить номер паспорту, прізвище, ім’я, номер телефону та адресу.

Інформація про послуги містить номер, назву, опис та вартість.

Інформація про замовлення містить номер, номер авто, номер паспорту працівника, код послуги та дату. Виводиться в окремий список з можливістю перегляду інформації в окремій, відведеній для цього формі.

Результуюча інформація це уся сукупність інформації котра видається у програмі після того як користувач здійснить потрібну вибірку по даних, які він вносив раніше до програми та які зберігались у базі даних.

3.4 Розробка бази даних

Для реалізації курсового проекту використано інтегроване середовище розробки програмного забезпечення “Visual Studio 2022”, з використанням мови програмування C#.

Створюємо проект Windows Forms, задаємо назву проекту та розташування.

Далі потрібно створити базу даних для збереження інформації про шиномонтаж та клієнтів, в середовищі SQL Management Studio створюємо нашу базу DataBase – New DataBase, рис.3.11:

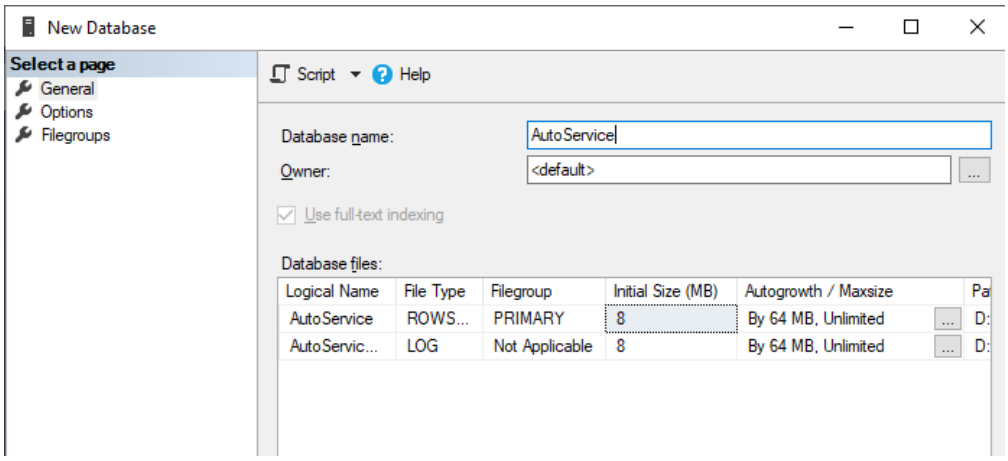


Рисунок 3.11 – створення бази даних AutoService

Після створення бази, потрібно в ній створити всі необхідні нам таблиці з потрібними полями та зв’язати таблиці між собою.

Для того щоб створити таблицю натискаємо Tables – New – Table. Вікно створення таблиці на рис.3.12:

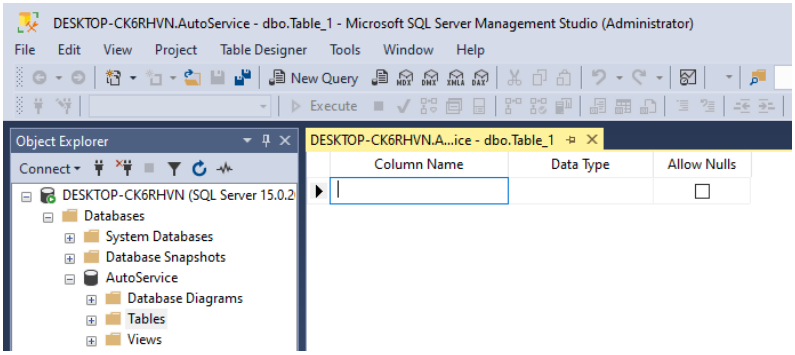


Рисунок 3.12 – Створення таблиць бази даних

Готові таблиці можна переглянути в списку Tables. Наші таблиці: Auto, Client, Staff, Service, Order.

Для створення відношень між таблицями потрібно створити нову діаграму, Diagrams – New Diagram, додати наші таблиці. Вибрать правою кнопкою миші таблицю авто і вибрати Relationships. Створення відношень між таблицями “Auto”,“Client” рис. 3.13:

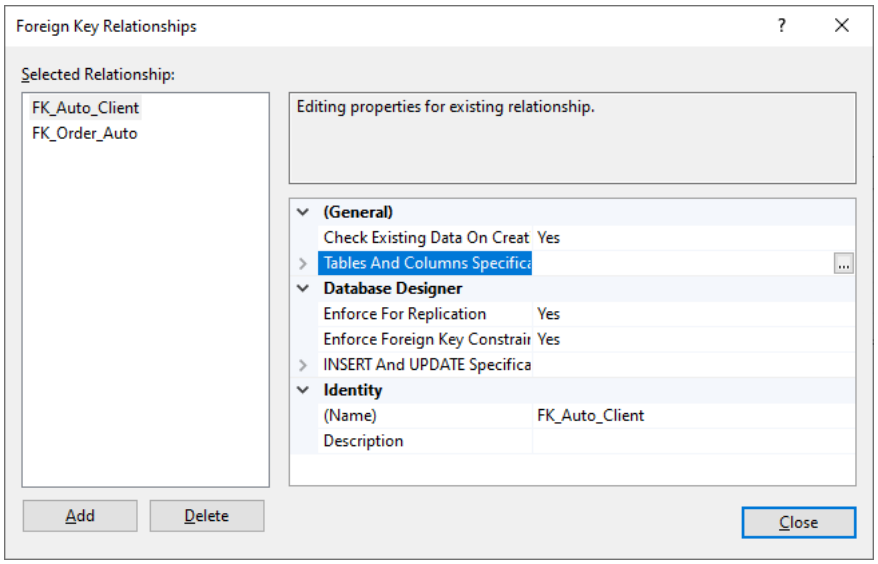


Рисунок 3.13 – Створення відношень між таблиця

Аналогічно проводимо з усіма таблицями і наша база даних готова.

Тепер необхідно підключити її до нашого вже готового проекту Windows Forms. Для цього переходимо до раніше створеного проекта в середовищі Visual Studio, до нашого проекту потрібно підключити нашу вже готову базу даних. Для цього переходимо у розділ Data Sources і обираємо Add new Data Source,

вибираємо Створити нове підключення, вводимо назву сервера та обираємо потрібну нам базу, рис.3.14:

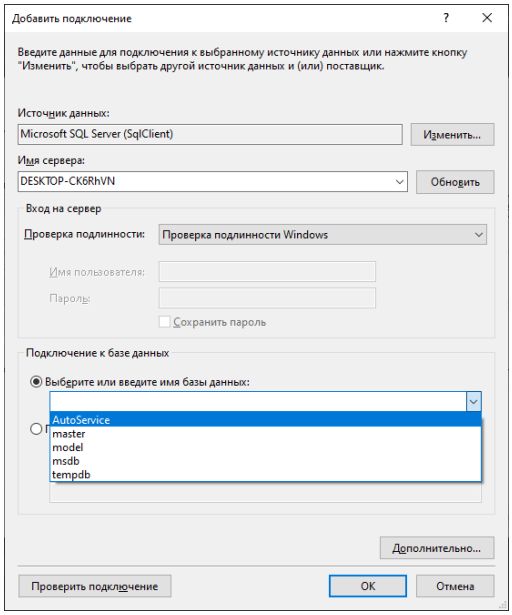


Рисунок 3.14 – Підключення бази даних до проекту

Далі обираємо об’єкти які нам потрібні з бази даних: Tables, Views, Stored Procedures, Functions.

Отже, наша база даних підключена до проекту та готова до використання, рис. 3.15:

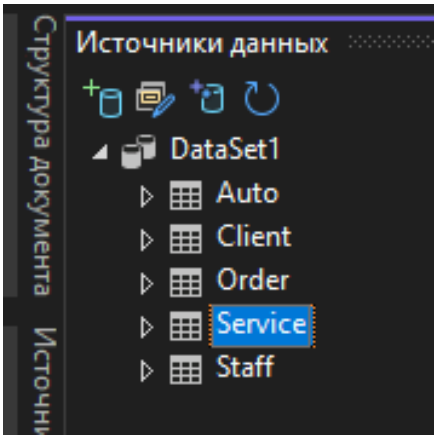


Рисунок 3.15 – База даних підключена до проекту

3.5 Керівництво користувача

Після запуску застосунку користувач потрапляє в головне меню в якому доступні наступні можливості, рис.3.16:

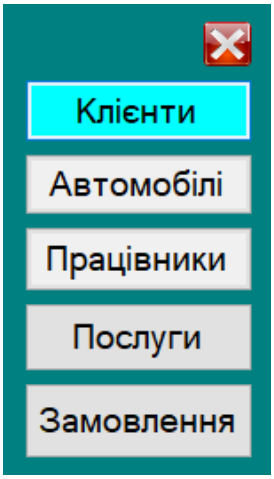


Рисунок 3.16 – Вигляд головного меню

Для початку потрібно додати дані в таблиці: Клієнти, Автомобілі, Працівники та Послуги. На рис. 3.17 зображено додавання інформації в таблицю клієнтів, також можна редагувати або видаляти інформацію, здійснювати пошук:

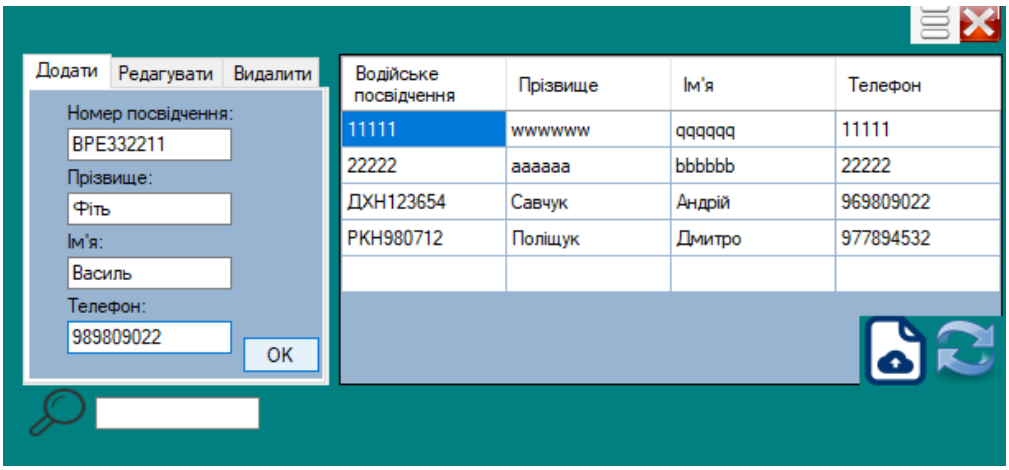


Рисунок 3.17 – Вигляд форми Клієнти

Також на формі є кнопка що дозволяє зробити звіт по даній таблиці за допомогою CrystalReport, також є можливість конвертувати у файли Word, Excel або ж одразу роздрукувати. Вигляд форми звіту по таблиці Клієнти рис 3.18:

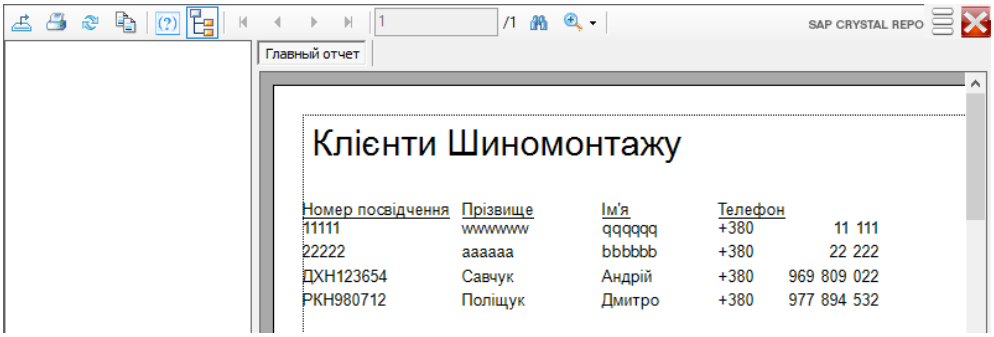


Рисунок 3.18 – Вигляд звіту по таблиці Клієнти

Далі, повернувшись на головне меню, користувач переходить на форму авто на якій доступні такі ж функції і методи, тільки тут необхідно вводити номер водійського поспідчення, яке вже було внесено в базу в таблицю Клієнти рис. 3.19:

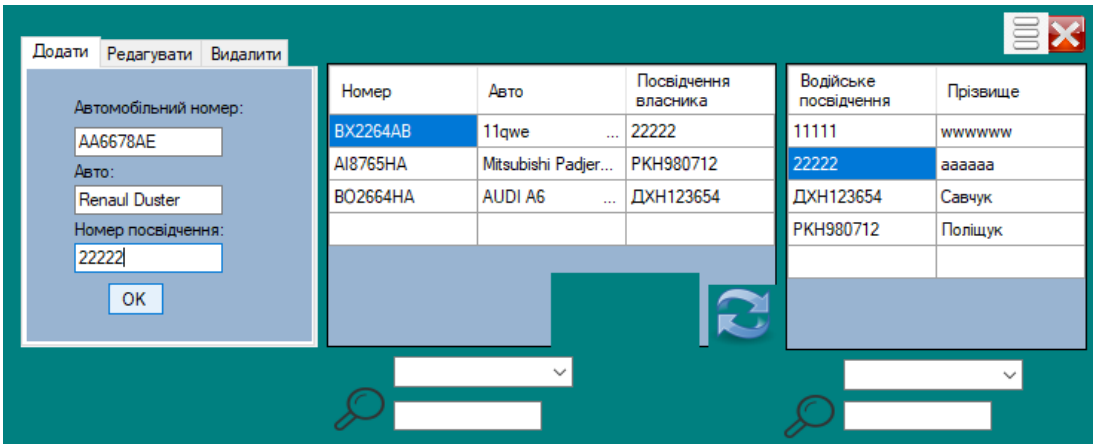


Рисунок 3.19 – Вигляд форми Авто

Сформована звітність по таблиці Авто рис 3.20:

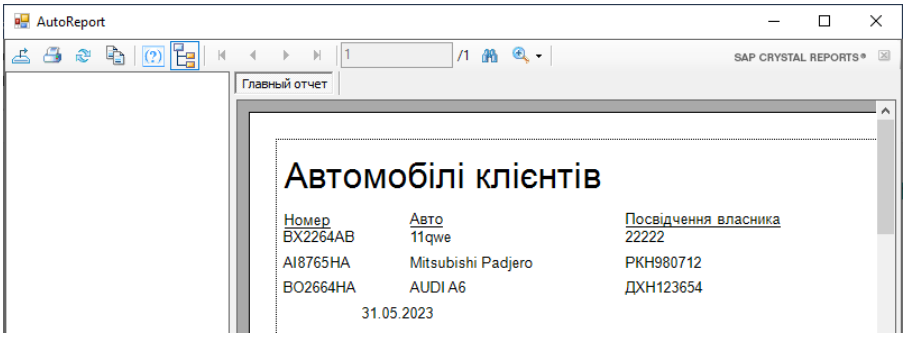


Рисунок 3.20 – Звітність по таблиці Авто

Потім користувачу необхідно ввести інформацію про доступні послуги на автосервісі, опис та ціну послуги, зображено на рис. 3.21:

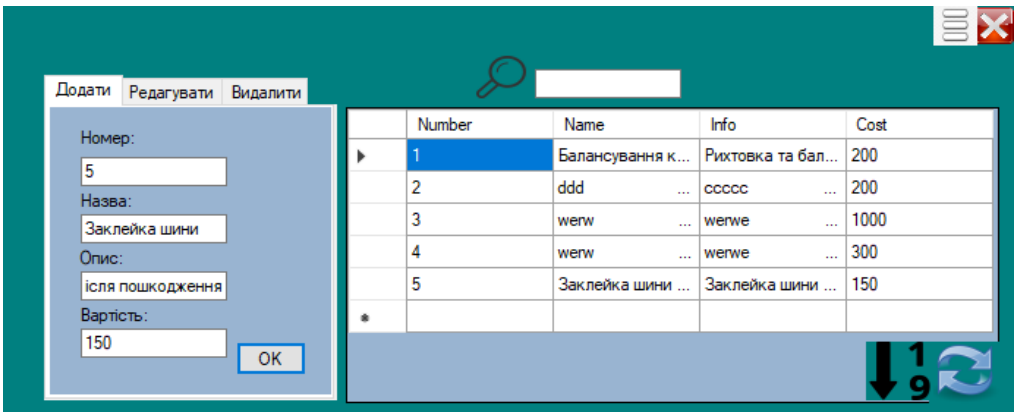


Рисунок 3.21 – Форма послуг

Також в нашій програмі є окрема форма для внесення, редагування та видалення інформації по працівникам шиномонтажу. Можна здіснювати пошук за прізвищем та ім’ям, додавання інформації про нового працівника на рис.3.22:

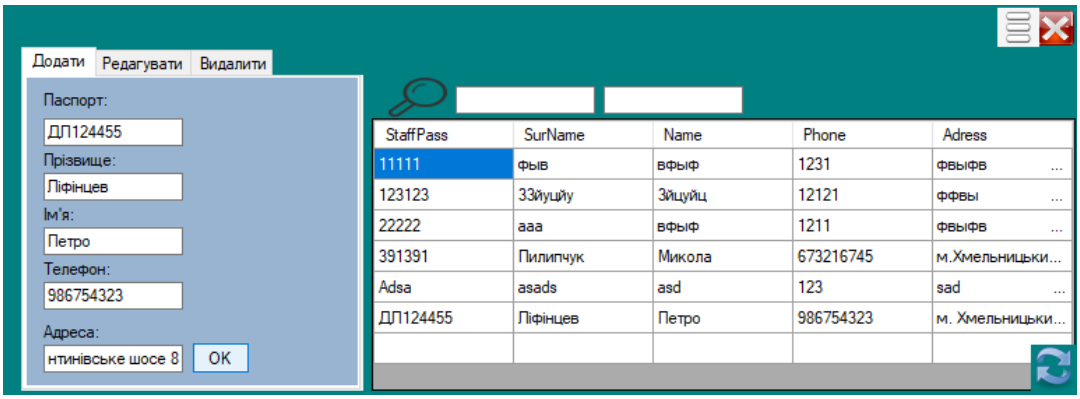


Рисунок 3.22 – Форма працівників

І головною формою у нашому застосунку є форма замовлення, кожний пункт одного замовлення записується у базу даних окремо, потім є можливість встановити фільтр за датою та авто щоб перевірити. Саме ж замовлення для користувача формується в окремому dataGridView, зображено на рис.3.23:

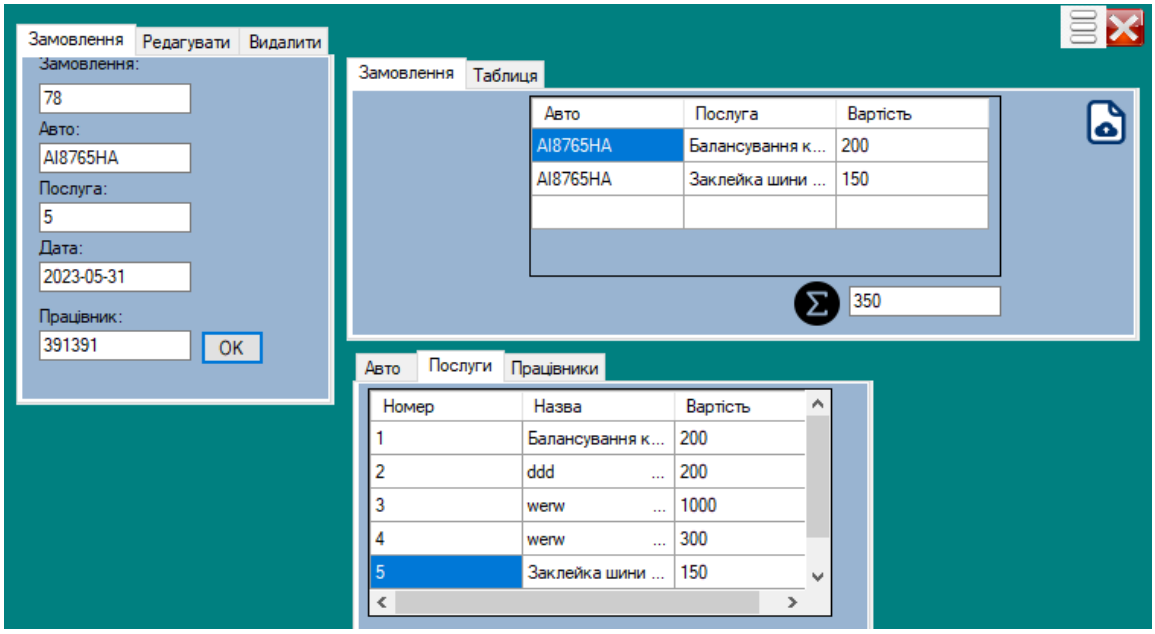


Рисунок 3.23 – Форма замовлення

Вікно збереженого файлу замовлення зображено на рис.3.24:

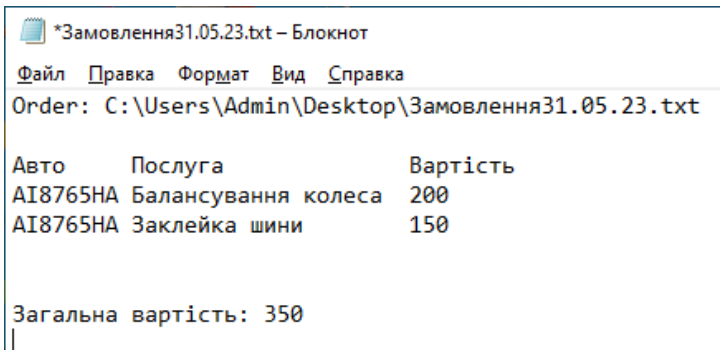


Рисунок 3.24 – Збережний файл замовлення

3.6 Вимоги до технічних та програмних засобів

Для успішного запуску програми для автоматизації обслуговування клієнтів шиномонтажу необхідно мати комп'ютер, що відповідає наступним мінімальним системним вимогам:

– мінімальна тактова частота процесора - 1 ГГц;

– операційна система: одна з наступних версій - Windows 7, 8, 10;

– обсяг оперативної пам'яті не менше 500 МБ;

– вільне місце на жорсткому диску - 20 МБ.

Програма, яка була розроблена, має простий та зрозумілий інтерфейс, що дозволяє користувачам легко з ним взаємодіяти.

Діаграма мінімальної конфігурації використовується для відображення загальної топології розподіленої програмної системи з компонентами, які представлені у вигляді вузлів з фізичними зв'язками для передачі інформації між пристроями. Діаграма мінімальної конфігурації зображена на рис. 3.25:

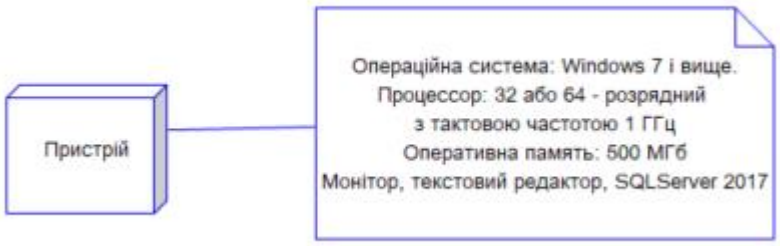


Рисунок 3.25 – Діаграма мінімальної конфігурації.

3.7 Тестування

В кваліфікаційній роботі буде використовуватись ручне тестування для перевірки та валідації розробленого програмного продукту. Ручне тестування є процесом, в якому тестер вручну перевіряє різні функції, функціональності та взаємодію програми з користувачем.

Під час ручного тестування використаємо різні тестові сценарії, спробуємо ввести некоректні дані, перевіримо реакцію програми на різні вхідні дані та ситуації. Використання ручного тестування дозволяє виявити можливі помилки, недоліки та непередбачені ситуації, які можуть виникнути при роботі програмного продукту.

Застосування ручного тестування в кваліфікаційній роботі допоможе переконатись в якості програмного продукту та забезпечити його стабільну та надійну роботу перед його впровадженням.

Перевірку почнемо почергово, з першої форми де здійснюється робота з інформацієї про клієнта. В нас є поле телефон, при ввдені не потрібно вводити символи “+380”, щоб користувач не здійснив помилку, встановимо інформацію про це одраху в textbox. Далі перейшов в розділ редагування та видалення, ми розуміємо що ці функції виконуються за допомогою ключового поля, в нас це номер водійського посвідчення клієнта. Зрозуміло, що це можна зробити тільки вже за існуючим клієнтом, щоб користувач не вводив поле якого немає, встановлюємо значення textbox-Enabled.False, а вибірка ключого поля здійснюється натиском в таблиці, також там доступний пошук на випадок якщо клієнтів багато, код вибору поля при натисканні:

```
       private void clientDataGridView_CellClick(object sender,
DataGridViewCellEventArgs e)
                {
                    if (tabControl1.SelectedTab == tabPage2)
                    {
                        if (e.RowIndex >= 0)
                        {
                            DataGridViewRow row =
clientDataGridView.Rows[e.RowIndex];
```

```
                            string value =
row.Cells[e.ColumnIndex].Value.ToString();
                            textBox5.Text = value;
                        }
                    }
                    if (tabControl1.SelectedTab == tabPage3)
                    {
                        if (e.RowIndex >= 0)
                        {
                            DataGridViewRow row =
clientDataGridView.Rows[e.RowIndex];
                            string value =
row.Cells[e.ColumnIndex].Value.ToString();
                            textBox9.Text = value;
                        }}}
```

Цей ж самий код буде використовуватись на всіх формах де здійснюється редагування або видалення інформації і також встановлюватиметься значення textbox-Enabled.False. Тобто користувач не вводить ключові поля відповідних таблиць вручну, а здійснює це за допомогою натискання поля dataGridView.

Далі користувач потрапляє на форму з авто. А з попередніх розділів ми знаємо що відношення між таблицями “Авто” та “Клієнти” зв’язок один до багатьох. Тобто щоб додати інформацію про авто, завчасно має бути введена інформація про власника. Таблиця “Авто” містить поле номер посвідчення власника, того на цю форму додаємо ще dataGridView в який виводиться інформація про клієнтів і доступний пошук. Користувач зможе таким ж чином за допомогою вибірки обрати власника і номер, що автоматично завантажується в textbox як і на формі клієнтів.

На формі послуг шиномонтажу є поля вартість, значення якого int, тобто користувач має вводити тільки цілі цифри. Для того, щоб обмежити користувача на введення тільки цілих чисел використаємо подію KeyPress та перевіряти код натиснутої клавіші. Ось код:

```
if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar))
                {
                    e.Handled = true; // Відміна введення неправильного символу
                }
```

Такий ж самий код використаємо у розділі редагування де також є поле вартості.

Після цих дій користувач потрапляє на форму де формується замовлення і зберігається у файл. Перша помилка з якою ми стикаємось це те, що кожний пункт замовлення має окреме ключове поле, яке потрібно автоматично збільшувати на одиницю, щоб користувач не робив цього вручну. Тому, що помилково можна ввести повторно таке ж саме ключове поле. Добавимо код який зчитує останнє ключове поле в dataGridView і збільшує на одиницю, та виводить його в textbox. Використовуємо int.TryParse, щоб переконатись, що значення в комірці є дійсним цілим числом. Якщо значення не може бути розпізнане як ціле число, то збільшення не відбудеться.

Цей код дозволить нам зчитувати останнє значення в стовпці DataGridView і збільшувати його на 1:

```
          if (orderDataGridView.Rows.Count > 0)
                    {
                        int lastRowIndex = orderDataGridView.Rows.Count - 1;
                        DataGridViewRow lastRow =
orderDataGridView.Rows[lastRowIndex];
                        if (lastRow.Cells.Count > 0)
                        {
                            DataGridViewCell lastCell = lastRow.Cells[1];
                            int lastValue;
                            if (int.TryParse(lastCell.Value?.ToString(), out
lastValue))
                            {
                                int incrementedValue = lastValue + 1;
                                lastCell.Value = incrementedValue.ToString();
                            }
                        }
                    }
```

Цей код використовуємо двічі, при запуску форми і при додаванні розділів у замовлення.

Остання помилка з якою ми стикаємось при перевірці це не правильний формат вводу дати. Додамо код в FormLoad який одразу буде виводити сьогоднышню дату, код:

```
textBox9.Text = DateTime.Today.ToString("yyyy-MM-dd");
```

Якщо дату необхідно встановити миулу, то користувачу необхідно ввести її в ручну, щоб не стало конфлікту додамо підказку при наведенні у якому форматі необхідно вводити. Код:

```
ToolTip toolTip = new ToolTip();
            toolTip.SetToolTip(textBox4, "2000-03-22");
                   toolTip.SetToolTip(textBox9, "2000-03-22");
```

Результат роботи зображено на рис. 3.26:

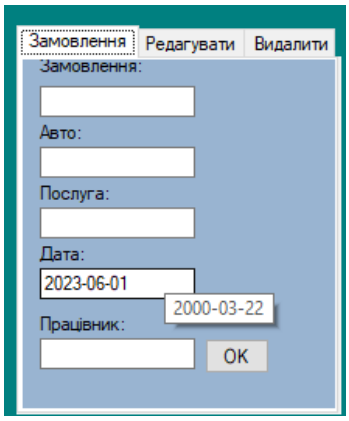


Рисунок 3.26 – Результат роботи підказки

3.6 Висновки за розділом 3

Висновки. У розділі 3 "Програмна реалізація та тестування" було проведено ряд дій, які включали детальне проектування модулів, їх програмну реалізацію, проектування даних та розробку бази даних. Крім того, було розроблено керівництво користувача та визначено вимоги до технічних та програмних засобів.

У процесі детального проектування модулів було враховано вимоги і потреби користувачів. Було створено детальні схеми і структури модулів, визначено їх функціональність та зв'язки з іншими модулями.

Програмна реалізація модулів включала розробку вихідного коду на мові програмування C# 7.0 та використання платформ .NET та .NET Core. Були використані раніше розроблені модулі, а також здійснена програмна інтеграція для забезпечення функціональності.

Під час детального проектування даних було визначено основні структури даних, необхідні для опрацювання в програмі. Об'єкти предметної області були розглянуті, а зв'язки між ними були встановлені. Для забезпечення нормалізації бази даних та унікальності даних було додано унікальний ідентифікатор до кожної таблиці.

Розробка бази даних включала проектування її структури та створення таблиць з необхідними полями і відношеннями. Реляційна база даних була використана для зберігання даних, а зв'язки між таблицями були встановлені за допомогою зовнішніх ключів, що забезпечило зв'язок один до багатьох та багато до одного.

Керівництво користувача було розроблено з метою надання детальної інформації про використання програмного продукту. У ньому були описані основні функції програми, порядок взаємодії з інтерфейсом, правила введення даних та інструкції з використання різних функціональних можливостей. Це допомагає користувачам швидко ознайомитись з програмою і використовувати її належним чином.

Вимоги до технічних та програмних засобів були визначені для забезпечення ефективної роботи програмного продукту. Були вказані необхідні версії операційної системи, мови програмування та додаткових компонентів, таких як база даних і середовище розробки.

Тестування включало проведення ручного тестування програмного продукту з метою виявлення помилок, недоліків та невідповідностей вимогам. Під час тестування перевірялась коректність роботи модулів, правильність збереження та відображення даних, а також взаємодія з базою даних. Були виявлені та виправлені помилки та недоліки, що підвищило якість програмного продукту.

Загалом, розділ 3 "Програмна реалізація та тестування" успішно дозволив розробити та протестувати програмний продукт для автоматизації роботи шиномонтажу. Використані технології, такі як C# 7.0, SQL Server, Microsoft SQL Management Studio та Crystal Reports, допомогли закріпити набуті знання та навички у розробці програмного забезпечення. Розроблений продукт вирішує потреби користувачів, спрощує та оптимізує робочі процеси шиномонтажу, покращує якість надання послуг та забезпечує зручну та ефективну роботу.

ВИСНОВКИ

В рамках нашої кваліфікаційної роботи на тему "Розробка програмного продукту для автоматизації роботи шиномонтажу" ми успішно реалізували програмний продукт, що допомагає впровадити ефективні та автоматизовані процеси в шиномонтажних центрах. Наша робота була спрямована на поліпшення робочого процесу шиномонтажу, забезпечення зручного і надійного інструментарію для обслуговування клієнтів та оптимізацію управління бізнесом.

Протягом роботи ми набули та закріпили практичні навички розробки застосунків з використанням різних технологій. В основі розробленого програмного продукту лежить мова програмування C#, яка виявилась дуже потужною та гнучкою для реалізації бізнес-логіки та інтерфейсу користувача.

Під час розробки ми використали такі технології і інструменти, як C#, SQL Server, Microsoft SQL Management Studio та Crystal Report. Глибокі знання цих технологій та інструментів дозволили нам ефективно розробити програмний продукт з високою якістю та функціональністю. Використання C# дозволило нам створити потужну та гнучку програмну архітектуру, а SQL Server разом з Microsoft SQL Management Studio забезпечили надійне зберігання та керування даними. Крім того, використання Crystal Report дозволило нам створити зручні та інформативні звіти для користувачів.

Наша робота має велику важливість, оскільки спрямована на автоматизацію роботи шиномонтажних центрів. Розроблений програмний продукт дозволяє оптимізувати та прискорити процеси шиномонтажу, забезпечити точність та надійність обробки даних, а також полегшити роботу працівників цих центрів. Це дозволить ефективніше використовувати ресурси, зменшити кількість помилок та підвищити загальну якість обслуговування.

Під час розробки ми провели детальний аналіз вхідних даних, визначили основні структури даних та їх відповідності у базі даних. Ми ретельно розробили модулі програми, забезпечивши їх правильну взаємодію та

функціональність. Також було проведено ручне тестування, щоб перевірити працездатність та коректність роботи програмного продукту.

У підсумку, наша кваліфікаційна робота виявилась успішною і досягла своїх цілей. Ми успішно розробили програмний продукт для автоматизації роботи шиномонтажу, закріпили знання в області розробки програмного забезпечення з використанням C#, SQL Server, Microsoft SQL Server Management Studio та Crystal Reports. Наша робота має велику важливість для шиномонтажних центрів, оскільки допомагає покращити їх ефективність, точність та надійність роботи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Маркіна О.М. Програмування баз даних: навч. посіб. для студ. спеціальності 152 «Метрологія та інформаційно-вимірювальна техніка» / М. В. Добролюбова, М. В. Філіппова, О. М. Маркіна ; КПІ ім. Ігоря Сікорського. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 164 с.

2. Лосєв М.Ю. Бази даних : навчально-практичний посібник для самостійної роботи студентів / М. Ю. Лосєв, В. В. Федько. – Харків : ХНЕУ ім. С. Кузнеця, 2018. – 233 с.

3. Олійник А.В. Інформаційні системи і технології у фінансових установах – Навчальний посібник /А.В.Олійник, В.М.Шацька - Навчальний посібник - Львів: "Новий Світ-2000", 2006 - 436 с.

4. Омельчук Л. Л. «Інструментальні середовища та технології програмування. Лабораторний практикум». / Л. Л. Омельчук, Н. Г. Русіна. Навчальний посібник – Одеса: Айс Принт, 2020. – 175 с

5. Пасічник В. В. Організація баз даних та знань. / В. В. Пасічник, В. А. Резніченко. — К.: Видавнича група BHV, 2006. — 384 с

6. Берко А. Ю. Системи баз даних та знань. Книга 1. Організація баз даних: навч. посібник /О.М. Верес, В.В. Пасічник, А.Ю. Берко – Львів : „Магнолія 2006”, 2008. – 456 с.

7. Берко А.Ю. Організація баз даних: практичний курс: Навч. посібник / А.Ю. Берко, О.М. Верес - Львів: Видавництво Національного університету "Львівська політехніка", 2003. - 152 с.

8. Гайна Г.А. Основи проектування баз даних: Навчальний посібник. − К.: КНУБА, 2005. – 204 с.

9. Бази даних. Поняття ER-моделі. Поняття сутності (entity). Атрибути. Види атрибутів URL: https://www.bestprog.net/uk/2019/01/24/the-concept-of-er-model-the-concept-of-essence-and-communication-attributes-attribute-types-ua/(дата звернення: 12.05.2023)

10. Гайдаржи В. І., Ізварін І. В. Бази даних в інформаційних системах/ І.В. Ізварін, В.І. Гайдаржи – Видавництво. Університет "Україна". 2018. 418 с.

11. Переваги SQL Server 2017 URL: https://itpro.ua/product/microsoft-sql-server2017/?tab=description (дата звернення: 25.05.2023)

12. Що таке SQL? URL: https://avada-media.ua/ua/sql/ (дата звернення: 25.05.2023)

13. SQL база даних. Для чого призначена база даних? URL: https://www.ukraine.com.ua/uk/blog/programming/sql-baza-dannih-dlya-chego-prednaznachena-baza-dannih.html (дата звернення 15.05.2023)

14. Microsoft SQL Server URL: https://docs.microsoft.com/enus/sql/relational-databases/sql-serverguides?view=sql-server-ver15 (дата звернення: 25.05.2023)

15. Сегеда І.В. Системи баз даних: Комп'ютерний практикум: навчальний посібник / І.В.Сегеда, О.А.Дацюк. – Київ: КПІ ім. Ігоря Сікорського, 2019. – 43 с.

16. Ахметшина Л. Г. Навчально-методичний посібник до курсового проектування з дисципліни «Організація баз даних» / Л. Г. Ахметшина, А. О. Єгоров, В. В. Герасимов. — Дніпро, 2018. — 68 с.

17. Що таке файл RPT? URL: https://docs.fileformat.com/uk/reporting/rpt/ (дата звернення 21.05.2023)

18. Грицюк Ю.І. Об'єктно-орієнтоване програмування мовою С++ / Т.Є. Рак, Ю.І. Грицюк – Львів, ЛДУ БЖД. 2011. – 404 с.

19. Вимоги до Microsoft SQL Server URL: https://help.eset.com/protect_install/10.0/uk-UA/sql_server_windows.html (дата звернення: 22.05.2023)

20. Гришанович Т. О. Основи об'єктно-орієнтованого програмування: навч. посіб. Харків: ФОП Панов А.М., 2020. – 104 с.

21. Дудзяний І.М. Об'єктно-орієнтоване моделювання програмних систем: навч. посібник. – Видавничий Центр ЛНУ імені Івана Франка, 2007. – 108.

22. Гришанович Т. О. Основи об’єктно-орієнтованого програмування: навч. посіб. Харків: ФОП Панов А.М., 2020. – 104 с.

23. Онищенко В. В. Спеціалізовані мови програмування: навч. посіб./ Т.П. Довженко, Онищенко В.В. – Київ: ДУТ. 2019. 146 с.

24. Бурлаков А. А. Об’єктно-орієнтований аналіз і проектування. Методичні рекомендації з самостійного вивчення дисципліни студентами напрямку підготовки «Програмна інженерія» – Хмельницький: ХНУ, 2017. – 136 с.

25. Світличний О.О. Основи геоінформатики: Навчальний посібник / С.В. Плотницький, О.О. Світличний — Суми: ВТД «Університетська книга», 2006.

26. Завадський І.О. Основи баз даних: навч. посіб. І.О. Завадський, – К. : Видавець 2011. – 192 с.

27. Бази даних та інформаційні системи. Навчальний курс URL: http://simulation.kiev.ua/dbis/lection06.html (дата звернення: 25.05.2023)

28. Що таке база даних? URL: https://hostiq.ua/wiki/ukr/database/ (дата звернення 24.05.2023)

29. Де використовується SQL і чому він так потрібен програмістам URL: https://kiev.itstep.org/blog/where-sql-is-used-and-why-programmers-need-it-so-much (дата звернення 23.05.2023)

30. Що таке бази даних, їх призначеня та види? URL: https://futurenow.com.ua/shho-take-bazy-danyh-yih-pryznachennya-ta-vydy/ (дата звернення 14.05.2023)

31. Програмування сайтів. Бази даних для сайтів URL: https://webstudio2u.net/ua/programming/494-site-programming.html (дата звернення 25.05. 2023)

32. Сайт про мову програмування C++ URL: http://www.cplusplus.com/. (дата звернення: 25.05.2023)

33. Сидоров М.О. Метод створення ефективного стилю програмування // Інженерія програмного забезпечення/М.М. Костів, М.О Сидоров – 2013. – № 3– 4 (15–16) – С. 17–24.

34. Омельчук Л.Л. Програмування: теорія та практика. Збірник матеріалів за результатами ІТ-проєкту міждисциплінарної інтеграції/ О.М. Ткаченко, Л.Л. Омельчук, О.В. Шишацька – Київ, 2022. – 155 с

35. В.О.Грязнова, Основи методології програмування./ Єфіменко С.В., Грязнова - К.: ВПЦ "Київський університет", 2005 р.

36. Копитко М.Ф. Основи програмування мовою Java: Тексти лекцій./ К.С. Іванків, М.Ф. Копитко. – Львів: Видавничий центр ЛНУ ім. Івана Франка, 2002. – 83 с.

37. Щербаков О.В. Основи об'єктно-орієнтованого програмування: навчальний посібник/ Ю.Е. Парфьонов. В.М. Федорченко – Харків : ХНЕУ ім. С. Кузнеця, 2019.– 237 с.

38. Берко А.Ю., Верес О.М. , Пасічник В.В. Системи управління базами даних та знань. Навчальний посібник (рек. МОН України)./ О.М. Верес, В.В. Пасічник, А.Ю. Берко Львів, ЛПІ, 2021. – 584с.

39. Управління і адміністрування баз даними URL: http://www.interface.ru/home.asp?artId=50&cId=3. (дата звернення: 25.05.2023)

40. Павлишко А.В. Конспект лекцій з дисципліни «Об’єктно-орієнтоване програмування» для студентів спеціальності 122 Комп’ютерні науки та інформаційні технології./ А.В. Павлишко, О.В. Савельєв — Одеса: ОНПУ, 2017. – 92 с.

ДОДАТОК А
(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках розробки програмного забезпечення для автоматизації обслуговування клієнтів шиномонтажу. Технічне завдання розроблено у відповідності до стандарту ГОСТ 19.201–78.

1 Підстава для розробки

Підставою для розробки є «Завдання на кваліфікаційну роботу», затверджене завідувачем кафедри інженерії програмного забезпечення. Найменування розробки: Програмне забезпечення для автоматизації клієнтів шиномонтажу.

2 Призначення розробки

2.1 Функціональне призначення

Функціональним призначенням застосунку є збереження, редагування, видалення інформації про клієнтів, авто, працівників, послуги та замовлення шиномонтажу.

2.2 Експлуатаційне призначення

Програма повинна експлуатуватися на комп’ютерах. Кінцевим користувачем застосунку виступає працівник шиномонтажу.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Для користувача:

- збереження, редагування та видалення інформації про клієнтів, авто, працівників, послуги та замовлення шиномонтажу;
- сортування, фільтрація, пошук;
- формування звітності з можливістю подальшого друку.

3.2 Вимоги до надійності

Застосунок повинен забезпечувати такі вимоги до надійності:

- обробляти невірні дії користувача і попереджати його про можливі наслідки;
- можливість самостійно відновлюватись у разі збою;
- можливість резервного копіювання бази даних.

3.3 Умови експлуатації та вимоги до технічних засобів

Для успішного запуску програми для автоматизації обслуговування клієнтів шиномонтажу необхідно мати комп'ютер, що відповідає наступним мінімальним системним вимогам:

- мінімальна тактова частота процесора - 1 ГГц;
- операційна система: одна з наступних версій - Windows 7, 8, 10;
- обсяг оперативної пам'яті не менше 500 МБ;
- вільне місце на жорсткому диску - 20 МБ.

3.4 Вимоги до інформаційної та програмної сумісності

Для розробки веб-застосунку був використаний мову програмуваппя C#, середовище Visual Studio, базу даних Sql Server та середовищн Microsoft Sql Management Studio, Crystal Report для формування звітів.

3.5 Спеціальні вимоги

Програма повинна мати зручний та зрозумілий зовнішній інтерфейс користувача.

4 Вимоги до програмної документації

У момент здачі проекту замовнику надається наступний набір документів:

- текст програми;
- опис програми;
- технічне завдання;
- керівництво користувача.

5 Стадії та етапи розробки

Стадії та етапи розробки застосунку для автоматизації обслуговування клієнтів шиномонтажу А.1.

Таблиця А.1 – Стадії та етапи розробки проекту

Стадія розробки	Етапи робіт	Зміст робіт
1	2	3
02.01.23 – 31.01.23	Обґрунтування необхідності розробки програми	Коротка характеристика програмного забезпечення; підстава і призначення розробки; вимоги до програмної системи і документація; стадії і етапи розробки програми; порядок контролю і приймання
Ескізний проект 01.02.23 – 26.02.23	Розробка ескізного проекту	Попередня розробка структури вхідних і вихідних даних; уточнення середовища програмування; розробка і опис загальної алгоритмічної структури
Технічний проект 29.02.23 – 19.03.23	Розробка технічного проекту	Уточнення структури вхідних і вихідних даних; розробка докладного алгоритму; розробка структури програми
Робочий проект 20.03.23 – 15.04.23	Розробка програмного забезпечення	Реалізація програмного забезпечення; відлагодження; проведення попереднього тестування
Розробка програмної документації 16.04.23 – 22.04.23	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням
Тестування системи 23.04.23 – 30.04.23	Проведення тестування програмного забезпечення	Розробка методики тестування; проведення основних тестів; коректування програмного забезпечення
Впровадження	Підготовка і передача програми	Підготовка і розгортання програмного забезпечення
Розробка програмної документації 16.04.23 – 22.04.23	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням

6 Порядок контролю та приймання

Контроль здійснюється кінцевими користувачами системи, підключеними на етапі тестування застосунку.

ДОДАТОК Б
(обов’язковий)

ДІАГРАМИ

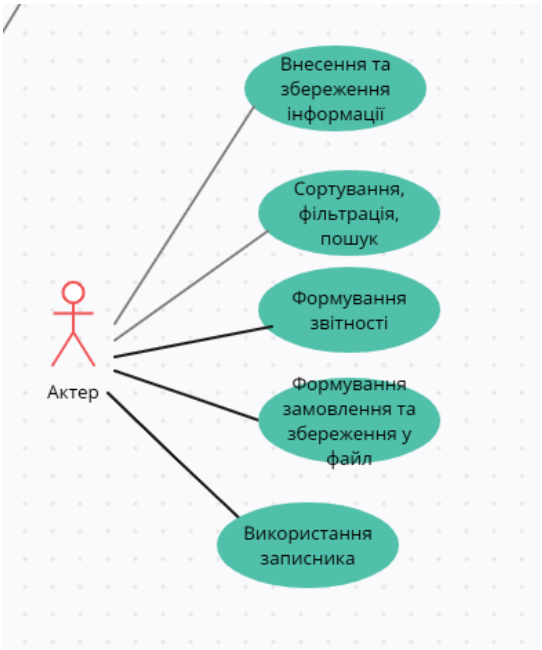


Рисунок Б.1 – Діаграма варіантів використання

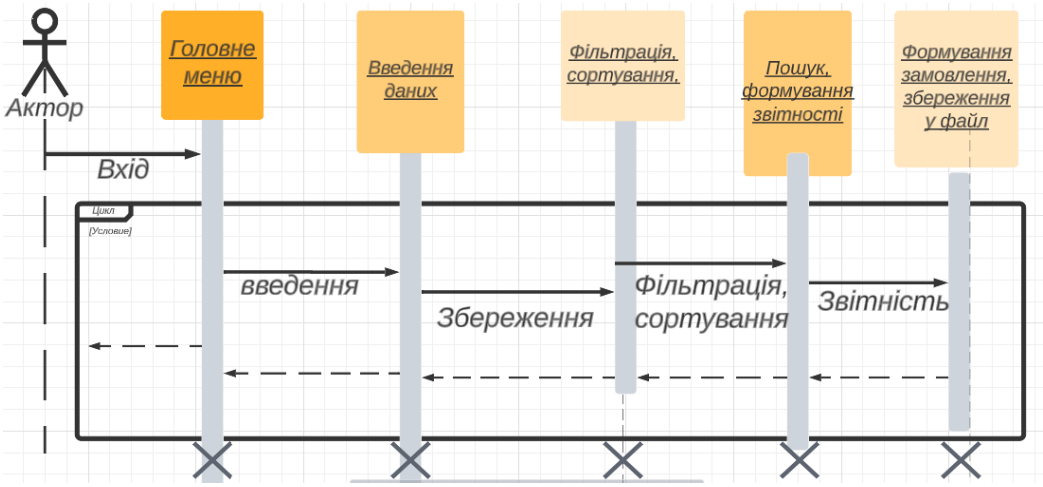


Рисунок Б.2 – Діаграма послідовності

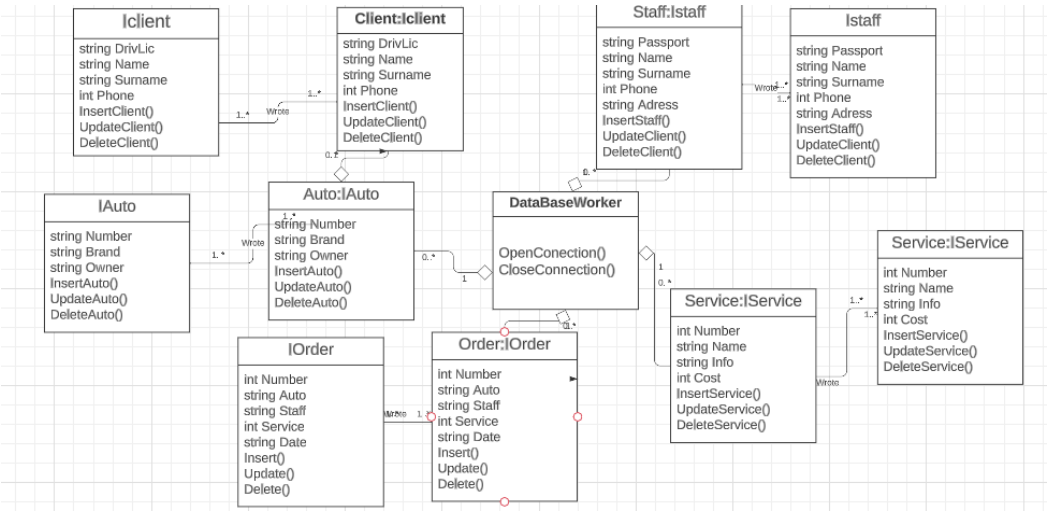


Рисунок Б.3 – Діаграма класів

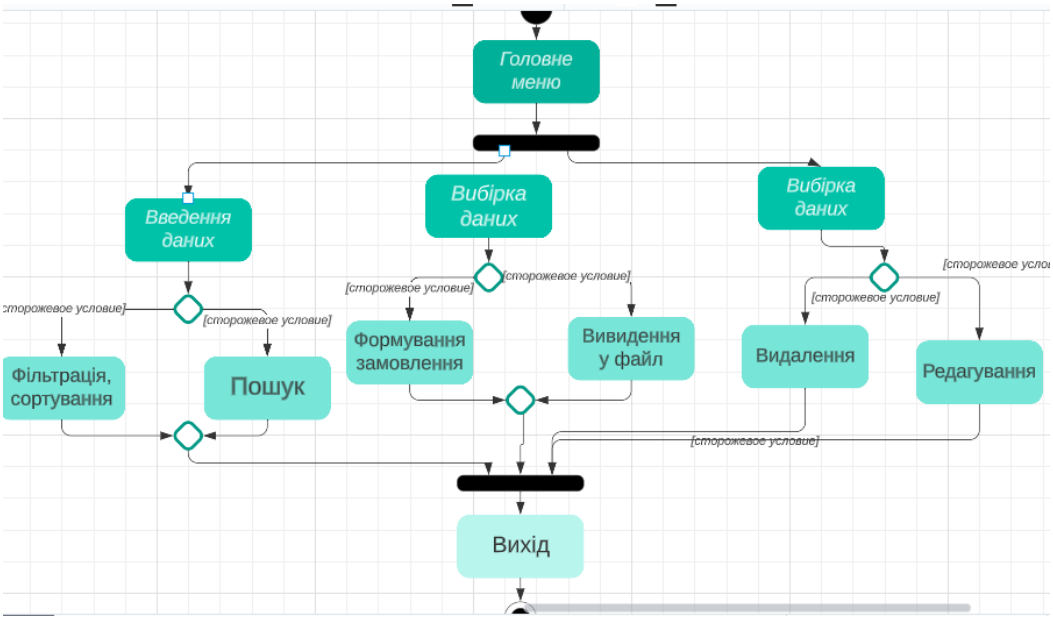


Рисунок Б.4 – Діаграма взаємодії

Рисунок Б.5 – Діаграма Станів

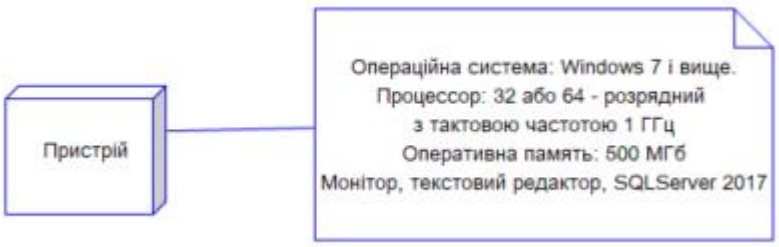


Рисунок Б.6 – Діаграма мінімальної конфігурації

ДОДАТОК В
(обов’язковий)

КОД ПРОГРАМИ
(лістинг)

```
interface IAuto
            {
                string Number { get; set; }
                string Brand { get; set; }
                string Owner { get; set; }
                void InsertAuto(string Number, string Brand, string Owner,
SqlConnection sqlConnection);
                void UpdateAuto(string Number, string Brand, string Owner,
SqlConnection sqlConnection);
                void DeleteAuto(string Number, SqlConnection sqlConnection);
            }
            class AutoClass : IAuto
            {
                public string Number { get; set; }
                public string Brand { get; set; }
                public string Owner { get; set; }

                public async void InsertAuto(string Number, string Brand, string Owner,
SqlConnection sqlConnection)
                {
                    SqlCommand command2 = new SqlCommand("INSERT INTO [Auto] (Number,
Brand, Owner)VALUES(@Number,@Brand,@Owner)", sqlConnection);
                    command2.Parameters.AddWithValue("Number", Number);
                    command2.Parameters.AddWithValue("Brand", Brand);
                    command2.Parameters.AddWithValue("Owner", Owner);
                    await command2.ExecuteNonQueryAsync();
                }
                public async void UpdateAuto(string Number, string Brand, string Owner,
SqlConnection sqlConnection)
                {
                    try
                    {
                        SqlCommand command = new SqlCommand("UPDATE [Auto] SET
[Brand]=@Brand, [Owner]=@Owner WHERE [Number]=@Number", sqlConnection);
                        command.Parameters.AddWithValue("Number", Number);
                        command.Parameters.AddWithValue("Brand", Brand);
                        command.Parameters.AddWithValue("Owner", Owner);
                        await command.ExecuteNonQueryAsync();
                    }
                    catch (Exception ty)
                    {
                        MessageBox.Show(ty.Message);
                    }
                }
                public async void DeleteAuto(string Number, SqlConnection
sqlConnection)
                {
                    SqlCommand command = new SqlCommand("DELETE FROM[Auto] WHERE
[Number]=@Number", sqlConnection);
                    command.Parameters.AddWithValue("Number", Number);

                    await command.ExecuteNonQueryAsync();
                }
            }
        interface IClient
            {
                string DrivLic { get; set; }
                string SurName { get; set; }
                string Name { get; set; }
                int Phone { get; set; }
                void InsertClient(string DrivLic, string SurName, string Name, int
Phone, SqlConnection sqlConnection);
                void UpdateClient(string DrivLic, string SurName, string Name, int
Phone, SqlConnection sqlConnection);
                void DeleteClient(string DrivLic, SqlConnection sqlConnection);
            }
            class ClientClass : IClient
```

```
            {
                public string DrivLic { get; set; }
                public string SurName { get; set; }
                public string Name { get; set; }
                public int Phone { get; set; }

                public async void InsertClient(string DrivLic, string SurName, string
Name, int Phone, SqlConnection sqlConnection)
                {
                    SqlCommand command2 = new SqlCommand("INSERT INTO [Client]
(DrivLic, SurName, Name, Phone)VALUES(@DrivLic,@SurName,@Name,@Phone)", sqlConnection);
                    command2.Parameters.AddWithValue("DrivLic", DrivLic);
                    command2.Parameters.AddWithValue("SurName", SurName);
                    command2.Parameters.AddWithValue("Name", Name);
                    command2.Parameters.AddWithValue("Phone", Phone);
                    await command2.ExecuteNonQueryAsync();
                }
                public async void UpdateClient(string DrivLic, string SurName, string
Name, int Phone, SqlConnection sqlConnection)
                {
                    try
                    {
                        SqlCommand command = new SqlCommand("UPDATE [Client] SET
[SurName]=@SurName, [Name]=@Name, [Phone]=@Phone WHERE [DrivLic]=@DrivLic",
sqlConnection);
                        command.Parameters.AddWithValue("DrivLic", DrivLic);
                        command.Parameters.AddWithValue("SurName", SurName);
                        command.Parameters.AddWithValue("Name", Name);
                        command.Parameters.AddWithValue("Phone", Phone);
                        await command.ExecuteNonQueryAsync();
                    }
                    catch (Exception ty)
                    {
                        MessageBox.Show(ty.Message);
                    }
                }
                public async void DeleteClient(string DrivLic, SqlConnection
sqlConnection)
                {
                    SqlCommand command = new SqlCommand("DELETE FROM[CLient] WHERE
[DrivLic]=@DrivLic", sqlConnection);
                    command.Parameters.AddWithValue("DrivLic", DrivLic);

                    await command.ExecuteNonQueryAsync();
                }

            }
        interface IStaff
            {
                string StaffPass { get; set; }
                string SurName { get; set; }
                string Name { get; set; }
                int Phone { get; set; }
                string Adress { get; set; }
                void InsertStaff(string StaffPass, string SurName, string Name, int
Phone, string Adress, SqlConnection sqlConnection);
                void UpdateStaff(string StaffPass, string SurName, string Name, int
Phone, string Adress, SqlConnection sqlConnection);
                void DeleteStaff(string StaffPass, SqlConnection sqlConnection);
            }
            class StaffClass : IStaff
            {
                public string StaffPass { get; set; }
                public string SurName { get; set; }
                public string Name { get; set; }
                public int Phone { get; set; }
                public string Adress { get; set; }
```

```
                public async void InsertStaff(string StaffPass, string SurName, string
Name, int Phone, string Adress, SqlConnection sqlConnection)
                {
                    SqlCommand command2 = new SqlCommand("INSERT INTO [Staff]
(StaffPass, SurName, Name, Phone,
Adress)VALUES(@StaffPass,@SurName,@Name,@Phone,@Adress)", sqlConnection);
                    command2.Parameters.AddWithValue("StaffPass", StaffPass);
                    command2.Parameters.AddWithValue("SurName", SurName);
                    command2.Parameters.AddWithValue("Name", Name);
                    command2.Parameters.AddWithValue("Phone", Phone);
                    command2.Parameters.AddWithValue("Adress", Adress);
                    await command2.ExecuteNonQueryAsync();
                }
                public async void UpdateStaff(string StaffPass, string SurName, string
Name, int Phone, string Adress, SqlConnection sqlConnection)
                {
                    try
                    {
                        SqlCommand command = new SqlCommand("UPDATE [Staff] SET
[SurName]=@SurName, [Name]=@Name, [Phone]=@Phone, [Adress]=@Adress WHERE
[StaffPass]=@StaffPass", sqlConnection);
                        command.Parameters.AddWithValue("StaffPass", StaffPass);
                        command.Parameters.AddWithValue("SurName", SurName);
                        command.Parameters.AddWithValue("Name", Name);
                        command.Parameters.AddWithValue("Phone", Phone);
                        command.Parameters.AddWithValue("Adress", Adress);
                        await command.ExecuteNonQueryAsync();
                    }
                    catch (Exception ty)
                    {
                        MessageBox.Show(ty.Message);
                    }
                }
                public async void DeleteStaff(string StaffPass, SqlConnection
sqlConnection)
                {
                    SqlCommand command = new SqlCommand("DELETE FROM[Staff] WHERE
[StaffPass]=@StaffPass", sqlConnection);
                    command.Parameters.AddWithValue("StaffPass", StaffPass);

                    await command.ExecuteNonQueryAsync();
                }

            }
        interface IService
            {
                int Number { get; set; }
                string Name { get; set; }
                string Info { get; set; }
                int Cost { get; set; }
                void InsertService(int Number, string Name, string Info, int Cost,
SqlConnection sqlConnection);
                void UpdateService(int Number, string Name, string Info, int Cost,
SqlConnection sqlConnection);
                void DeleteService(int Number, SqlConnection sqlConnection);
            }
            class ServiceClass : IService
            {
                public int Number { get; set; }
                public string Name { get; set; }
                public string Info { get; set; }
                public int Cost { get; set; }

                public async void InsertService(int Number, string Name, string Info,
int Cost, SqlConnection sqlConnection)
                {
```

```
                    SqlCommand command2 = new SqlCommand("INSERT INTO [Service]
(Number, Name, Info, Cost)VALUES(@Number,@Name,@Info,@Cost)", sqlConnection);
                    command2.Parameters.AddWithValue("Number", Number);
                    command2.Parameters.AddWithValue("Name", Name);
                    command2.Parameters.AddWithValue("Info", Info);
                    command2.Parameters.AddWithValue("Cost", Cost);
                    await command2.ExecuteNonQueryAsync();
                }
                public async void UpdateService(int Number, string Name, string Info,
int Cost, SqlConnection sqlConnection)
                {
                    try
                    {
                        SqlCommand command = new SqlCommand("UPDATE [Service] SET
[Name]=@Name, [Info]=@Info, [Cost]=@Cost WHERE [Number]=@Number", sqlConnection);
                        command.Parameters.AddWithValue("Number", Number);
                        command.Parameters.AddWithValue("Name", Name);
                        command.Parameters.AddWithValue("Info", Info);
                        command.Parameters.AddWithValue("Cost", Cost);
                        await command.ExecuteNonQueryAsync();
                    }
                    catch (Exception ty)
                    {
                        MessageBox.Show(ty.Message);
                    }
                }
                public async void DeleteService(int Number, SqlConnection
sqlConnection)
                {
                    SqlCommand command = new SqlCommand("DELETE FROM[Service] WHERE
[Number]=@Number", sqlConnection);
                    command.Parameters.AddWithValue("Number", Number);

                    await command.ExecuteNonQueryAsync();
                }

            }
        interface IOrder
            {
                int Number { get; set; }
                string Auto { get; set; }
                int Service { get; set; }
                string Date { get; set; }
                string Performer { get; set; }
                void InsertOrder(int Number, string Auto, int Service, string Date,
string Performer, SqlConnection sqlConnection);
                void UpdateOrder(int Number, string Auto, int Service, string Date,
string Performer, SqlConnection sqlConnection);
                void DeleteOrder(int Number, SqlConnection sqlConnection);
            }
            class OrderClass:IOrder
            {
                public int Number { get; set; }
                public string Auto { get; set; }
                public int Service { get; set; }
                public string Date { get; set; }
                public string Performer { get; set; }

                public async void InsertOrder(int Number, string Auto, int Service,
string Date, string Performer, SqlConnection sqlConnection)
                {
                    SqlCommand command2 = new SqlCommand("INSERT INTO [Order] (Number,
Auto, Service, Date, Performer)VALUES(@Number,@Auto,@Service,@Date,@Performer)",
sqlConnection);
                    command2.Parameters.AddWithValue("Number", Number);
                    command2.Parameters.AddWithValue("Auto", Auto);
                    command2.Parameters.AddWithValue("Service", Service);
```

```
                    command2.Parameters.AddWithValue("Date", Date);
                    command2.Parameters.AddWithValue("Performer", Performer);
                    await command2.ExecuteNonQueryAsync();
                }
                public async void UpdateOrder(int Number, string Auto, int Service,
string Date, string Performer, SqlConnection sqlConnection)
                {
                    try
                    {
                        SqlCommand command = new SqlCommand("UPDATE [Order] SET
[Auto]=@Auto, [Service]=@Service, [Date]=@Date, [Performer]=@Performer WHERE
[Number]=@Number", sqlConnection);
                        command.Parameters.AddWithValue("Number", Number);
                        command.Parameters.AddWithValue("Auto", Auto);
                        command.Parameters.AddWithValue("Service", Service);
                        command.Parameters.AddWithValue("Date", Date);
                        command.Parameters.AddWithValue("Performer", Performer);
                        await command.ExecuteNonQueryAsync();
                    }
                    catch (Exception ty)
                    {
                        MessageBox.Show(ty.Message);
                    }
                }
                public async void DeleteOrder(int Number, SqlConnection sqlConnection)
                {
                    SqlCommand command = new SqlCommand("DELETE FROM[Order] WHERE
[Number]=@Number", sqlConnection);
                    command.Parameters.AddWithValue("Number", Number);

                    await command.ExecuteNonQueryAsync();
                }
            }
        public partial class Auto : Form
            {
                string Number, Brand, Owner;
                public Auto()
                {
                    InitializeComponent();
                    autoDataGridView.CellClick += new
DataGridViewCellEventHandler(autoDataGridView_CellClick);
                }
                DataBaseWorker db = new DataBaseWorker();

                private void button1_Click(object sender, EventArgs e)
                {

                    if (textBox1.Text == "" || textBox2.Text == "" || textBox3.Text ==
"")
                    {
                        MessageBox.Show("Заповніть усі поля!");
                    }
                    else
                    {
                        AutoClass qw = new AutoClass();
                        Number = Convert.ToString(textBox1.Text);
                        Brand = Convert.ToString(textBox2.Text);
                        Owner = Convert.ToString(textBox3.Text);
                        qw.InsertAuto(Number, Brand, Owner, db.sqlConnection);

                    }
                }

                private void button2_Click(object sender, EventArgs e)
                {
                    if (textBox4.Text == "" || textBox5.Text == "" || textBox6.Text ==
"")
```

```
                    {
                        MessageBox.Show("Заповніть усі поля!");
                    }
                    else
                    {
                        AutoClass qw = new AutoClass();
                        Number = Convert.ToString(textBox4.Text);
                        Brand = Convert.ToString(textBox5.Text);
                        Owner = Convert.ToString(textBox6.Text);
                        qw.UpdateAuto(Number, Brand, Owner, db.sqlConnection);

                    }
                }

                private void autoDataGridView_CellClick(object sender,
DataGridViewCellEventArgs e)
                {
                    if (tabControl1.SelectedTab == tabPage2)
                    {
                        if (e.RowIndex >= 0)
                        {
                            DataGridViewRow row = autoDataGridView.Rows[e.RowIndex];
                            string value = row.Cells[e.ColumnIndex].Value.ToString();
                            textBox4.Text = value;
                        }
                    }
                    if (tabControl1.SelectedTab == tabPage3)
                    {
                        if (e.RowIndex >= 0)
                        {
                            DataGridViewRow row = autoDataGridView.Rows[e.RowIndex];
                            string value = row.Cells[e.ColumnIndex].Value.ToString();
                            textBox7.Text = value;
                        }
                    }
                }

                private void button3_Click(object sender, EventArgs e)
                {
                    if (textBox7.Text == "")
                    {
                        MessageBox.Show("Виберіть поле для видалення!");
                    }
                    else
                    {
                        AutoClass qw = new AutoClass();
                        Number = Convert.ToString(textBox7.Text);
                        qw.DeleteAuto(Number, db.sqlConnection);

                    }

                }

                private void pictureBox1_Click(object sender, EventArgs e)
                {
                    autoTableAdapter.Fill(dataSet1.Auto);
                }


                private void pictureBox2_Click(object sender, EventArgs e)
                {
                    if (comboBox1.Text == "" || textBox8.Text == "")
                    {
                        MessageBox.Show("Виберіть та введіть поле для фільтрації!");
                    }
                    {
                        if (comboBox1.Text == "Номер")
```

```
                        {
                            autoBindingSource.Filter = "CONVERT(Number,
'System.String') LIKE '%" + textBox8.Text + "%'";
                        }
                        if (comboBox1.Text == "Авто")
                        {
                            autoBindingSource.Filter = "Brand LIKE '%" + textBox8.Text
+ "%'";
                        }
                        if (comboBox1.Text == "Посвідчення")
                        {
                            autoBindingSource.Filter = "Owner LIKE '%" + textBox8.Text
+ "%'";
                        }
                    }
                }

                private void pictureBox3_Click(object sender, EventArgs e)
                {

                    if (comboBox2.Text == "" || textBox9.Text == "")
                    {
                        MessageBox.Show("Виберіть та введіть поле для фільтрації!");
                    }
                    {
                        if (comboBox2.Text == "Посвідчення")
                        {
                            clientBindingSource.Filter = "CONVERT(DrivLic,
'System.String') LIKE '%" + textBox9.Text + "%'";
                        }
                        if (comboBox2.Text == "Прізвище")
                        {
                            clientBindingSource.Filter = "SurName LIKE '%" +
textBox9.Text + "%'";
                        }

                    }
                }

                private void pictureBox5_Click(object sender, EventArgs e)
                {
                    Application.Exit();
                }

                private void pictureBox4_Click(object sender, EventArgs e)
                {
                    Form1 form1 = new Form1();
                    form1.Show();
                    Hide();
                }

                private void pictureBox6_Click(object sender, EventArgs e)
                {
                    AutoReport form1 = new AutoReport();
                    form1.Show();
                    Hide();
                }

                private void Auto_Load(object sender, EventArgs e)
                {
                    this.clientTableAdapter.Fill(this.dataSet1.Client);

                    this.autoTableAdapter.Fill(this.dataSet1.Auto);
                    this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.None;
                    db.OpenConnection();
                }
```

```
            private void autoBindingNavigatorSaveItem_Click(object sender,
EventArgs e)
            {  }
        }
    public partial class Client : Form
        {
            string DrivLic, SurName, Name;
            int Phone;



            public Client()
            {
                InitializeComponent();
                clientDataGridView.CellClick += new
DataGridViewCellEventHandler(clientDataGridView_CellClick);
            }
            DataBaseWorker db = new DataBaseWorker();

            private async void Client_Load(object sender, EventArgs e)
            {
                this.clientTableAdapter.Fill(this.dataSet1.Client);
                this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.None;
                db.OpenConnection();

            }

            private void button2_Click(object sender, EventArgs e)
            {
                if (textBox5.Text == "" || textBox6.Text == "" || textBox7.Text ==
"" || textBox8.Text == "")
                {
                    MessageBox.Show("Заповніть усі поля!");
                }
                else
                {
                    ClientClass qw = new ClientClass();
                    DrivLic = Convert.ToString(textBox5.Text);
                    SurName = Convert.ToString(textBox6.Text);
                    Name = Convert.ToString(textBox7.Text);
                    Phone = Convert.ToInt32(textBox8.Text);
                    qw.UpdateClient(DrivLic, SurName, Name, Phone,
db.sqlConnection);
                    clientTableAdapter.Fill(dataSet1.Client);
                }
            }

            private void clientDataGridView_CellClick(object sender,
DataGridViewCellEventArgs e)
            {
                if (tabControl1.SelectedTab == tabPage2)
                {
                    if (e.RowIndex >= 0)
                    {
                        DataGridViewRow row = clientDataGridView.Rows[e.RowIndex];
                        string value = row.Cells[e.ColumnIndex].Value.ToString();
                        textBox5.Text = value;
                    }
                }
                if (tabControl1.SelectedTab == tabPage3)
                {
                    if (e.RowIndex >= 0)
                    {
                        DataGridViewRow row = clientDataGridView.Rows[e.RowIndex];
                        string value = row.Cells[e.ColumnIndex].Value.ToString();
                        textBox9.Text = value;
                    }
```

```
            }
        }

        private void textBox2_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox3_TextChanged(object sender, EventArgs e)
        {

        }

        private void button3_Click(object sender, EventArgs e)
        {
            if (textBox9.Text == "")
            {
                MessageBox.Show("Виберіть поле для видалення!");
            }
            else
            {
                ClientClass qw = new ClientClass();
                DrivLic = Convert.ToString(textBox9.Text);
                qw.DeleteClient(DrivLic, db.sqlConnection);
                clientTableAdapter.Fill(dataSet1.Client);
            }
            clientTableAdapter.Fill(dataSet1.Client);
        }

        private void button1_Click_1(object sender, EventArgs e)
        {
            if (textBox1.Text == "" || textBox2.Text == "" || textBox3.Text ==
"" || textBox4.Text == "")
            {
                MessageBox.Show("Заповніть усі поля!");
            }
            else
            {
                ClientClass qw = new ClientClass();
                DrivLic = Convert.ToString(textBox1.Text);
                SurName = Convert.ToString(textBox2.Text);
                Name = Convert.ToString(textBox3.Text);
                Phone = Convert.ToInt32(textBox4.Text);
                qw.InsertClient(DrivLic, SurName, Name, Phone,
db.sqlConnection);
            }
            clientTableAdapter.Fill(dataSet1.Client);
        }



        private void pictureBox2_Click(object sender, EventArgs e)
        {
            if (textBox10.Text == " ")
            {
                MessageBox.Show("Введіть поле для фільтрації!");
            }
            {
                clientBindingSource.Filter = "SurName LIKE '%" + textBox10.Text
+ "%'";
            }
        }

        private void pictureBox3_Click(object sender, EventArgs e)
        {
            Form1 form1 = new Form1();
            form1.Show();
```

```
                Hide();
            }

            private void pictureBox1_Click(object sender, EventArgs e)
            {
                Application.Exit();
            }

            private void pictureBox4_Click(object sender, EventArgs e)
            {
                clientTableAdapter.Fill(dataSet1.Client);
            }

            private void pictureBox5_Click(object sender, EventArgs e)
            {
                ClientReport form1 = new ClientReport();
                form1.Show();
                Hide();
            }
        public partial class Staff : Form
            {
            string StaffPass, SurName, Name, Adress;
            int Phone;
            public Staff()
            {
                InitializeComponent();
                staffDataGridView.CellClick += new
DataGridViewCellEventHandler(staffDataGridView_CellClick);
            }
            DataBaseWorker db = new DataBaseWorker();

            private void button1_Click(object sender, EventArgs e)
            {
                if (textBox1.Text == "" || textBox2.Text == "" || textBox3.Text ==
"" || textBox4.Text == "" || textBox5.Text == "")
                {
                    MessageBox.Show("Заповніть усі поля!");
                }
                else
                {
                    StaffClass qw = new StaffClass();
                    StaffPass = Convert.ToString(textBox1.Text);
                    SurName = Convert.ToString(textBox2.Text);
                    Name = Convert.ToString(textBox3.Text);
                    Phone = Convert.ToInt32(textBox4.Text);
                    Adress = Convert.ToString(textBox5.Text);
                    qw.InsertStaff(StaffPass, SurName, Name, Phone,Adress,
db.sqlConnection);
                }

            }

            private void button2_Click(object sender, EventArgs e)
            {
                if (textBox6.Text == "" || textBox7.Text == "" || textBox8.Text ==
"" || textBox9.Text == "" || textBox10.Text == "")
                {
                    MessageBox.Show("Заповніть усі поля!");
                }
                else
                {
                    StaffClass qw = new StaffClass();
                    StaffPass = Convert.ToString(textBox6.Text);
                    SurName = Convert.ToString(textBox7.Text);
                    Name = Convert.ToString(textBox8.Text);
                    Phone = Convert.ToInt32(textBox9.Text);
                    Adress = Convert.ToString(textBox10.Text);
```

```
                    qw.UpdateStaff(StaffPass, SurName, Name, Phone, Adress,
db.sqlConnection);

                }

            }

            private void staffDataGridView_CellClick(object sender,
DataGridViewCellEventArgs e)
            {
                if (tabControl1.SelectedTab == tabPage2)
                {
                    if (e.RowIndex >= 0)
                    {
                        DataGridViewRow row = staffDataGridView.Rows[e.RowIndex];
                        string value = row.Cells[e.ColumnIndex].Value.ToString();
                        textBox6.Text = value;
                    }
                }
                if (tabControl1.SelectedTab == tabPage3)
                {
                    if (e.RowIndex >= 0)
                    {
                        DataGridViewRow row = staffDataGridView.Rows[e.RowIndex];
                        string value = row.Cells[e.ColumnIndex].Value.ToString();
                        textBox11.Text = value;
                    }
                }
            }

            private void button3_Click(object sender, EventArgs e)
            {
                if (textBox11.Text == "")
                {
                    MessageBox.Show("Виберіть поле для видалення!");
                }
                else
                {
                    StaffClass qw = new StaffClass();
                    StaffPass = Convert.ToString(textBox11.Text);
                    qw.DeleteStaff(StaffPass, db.sqlConnection);

                }

            }

            private void pictureBox1_Click(object sender, EventArgs e)
            {
                staffTableAdapter.Fill(dataSet1.Staff);
            }

            private void pictureBox2_Click(object sender, EventArgs e)
            {
                if (textBox12.Text == " ")
                {
                    MessageBox.Show("Введіть поле для фільтрації!");
                }
                {
                    //staffBindingSource.Filter = "SurName LIKE '%" +
textBox12.Text + "%'";
                    staffBindingSource.Filter = "SurName LIKE '%" + textBox12.Text
+ "%' AND Name LIKE '%" + textBox13.Text + "%'";

                }
            }

            private void pictureBox3_Click(object sender, EventArgs e)
```

```
            {
                Form1 form1 = new Form1();
                form1.Show();
                Hide();
            }

            private void pictureBox4_Click(object sender, EventArgs e)
            {
                Application.Exit();
            }

            private void Staff_Load(object sender, EventArgs e)
            {

                this.staffTableAdapter.Fill(this.dataSet1.Staff);
                this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.None;
                db.OpenConnection();
            }
        public partial class Service : Form
            {
                string Name, Info;
                int Number, Cost;
                public Service()
                {
                    InitializeComponent();
                    serviceDataGridView.CellClick += new
DataGridViewCellEventHandler(serviceDataGridView_CellClick);
                }
                DataBaseWorker db = new DataBaseWorker();

                private void button1_Click_1(object sender, EventArgs e)
                {

                }

                private void button1_Click(object sender, EventArgs e)
                {
                    if (textBox1.Text == "" || textBox2.Text == "" || textBox3.Text ==
"" || textBox4.Text == "")
                    {
                        MessageBox.Show("Заповніть усі поля!");
                    }
                    else
                    {
                        ServiceClass qw = new ServiceClass();
                        Number = Convert.ToInt32(textBox1.Text);
                        Name = Convert.ToString(textBox2.Text);
                        Info = Convert.ToString(textBox3.Text);
                        Cost = Convert.ToInt32(textBox4.Text);
                        qw.InsertService(Number, Name, Info, Cost, db.sqlConnection);
                    }

                }

                private void button2_Click(object sender, EventArgs e)
                {
                    if (textBox5.Text == "" || textBox6.Text == "" || textBox7.Text ==
"" || textBox8.Text == "")
                    {
                        MessageBox.Show("Заповніть усі поля!");
                    }
                    else
                    {
                        ServiceClass qw = new ServiceClass();
                        Number = Convert.ToInt32(textBox5.Text);
                        Name = Convert.ToString(textBox6.Text);
                        Info = Convert.ToString(textBox7.Text);
```

```
                Cost = Convert.ToInt32(textBox8.Text);
                qw.UpdateService(Number, Name, Info, Cost, db.sqlConnection);

            }
        }

        private void serviceDataGridView_CellClick(object sender,
DataGridViewCellEventArgs e)
        {
            if (tabControl1.SelectedTab == tabPage2)
            {
                if (e.RowIndex >= 0)
                {
                    DataGridViewRow row = serviceDataGridView.Rows[e.RowIndex];
                    string value = row.Cells[e.ColumnIndex].Value.ToString();
                    textBox5.Text = value;
                }
            }
            if (tabControl1.SelectedTab == tabPage3)
            {
                if (e.RowIndex >= 0)
                {
                    DataGridViewRow row = serviceDataGridView.Rows[e.RowIndex];
                    string value = row.Cells[e.ColumnIndex].Value.ToString();
                    textBox9.Text = value;
                }
            }
        }

        private void button3_Click(object sender, EventArgs e)
        {
            if (textBox9.Text == "")
            {
                MessageBox.Show("Виберіть поле для видалення!");
            }
            else
            {
                ServiceClass qw = new ServiceClass();
                Number = Convert.ToInt32(textBox9.Text);
                qw.DeleteService(Number, db.sqlConnection);

            }
            ;
        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {
            serviceTableAdapter.Fill(dataSet1.Service);
        }


        private void pictureBox2_Click(object sender, EventArgs e)
        {
            serviceBindingSource.Sort = "Cost ASC ";
        }

        private void pictureBox3_Click(object sender, EventArgs e)
        {
            if (textBox10.Text == " ")
            {
                MessageBox.Show("Введіть поле для фільтрації!");
            }
            {
                serviceBindingSource.Filter = "Name LIKE '%" + textBox10.Text +
"%'";
            }
        }
```

```
                private void pictureBox5_Click(object sender, EventArgs e)
                {
                    Application.Exit();
                }

                private void pictureBox4_Click(object sender, EventArgs e)
                {
                    Form1 form1 = new Form1();
                    form1.Show();
                    Hide();
                }

                private void textBox4_KeyPress(object sender, KeyPressEventArgs e)
                {
                    if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar))
                    {
                        e.Handled = true; // Відміна введення неправильного символу
                    }
                }

                private void Service_Load(object sender, EventArgs e)
                {

                    this.serviceTableAdapter.Fill(this.dataSet1.Service);
                    this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.None;
                    db.OpenConnection();
                }
        public partial class Order : Form
            {
                string Auto,Date,Performer;
                int Number,Service;
                public Order()
                {
                    InitializeComponent();
                    orderDataGridView.CellClick += new
DataGridViewCellEventHandler(orderDataGridView_CellClick);
                    autoDataGridView.CellClick += new
DataGridViewCellEventHandler(autoDataGridView_CellClick);
                    serviceDataGridView.CellClick += new
DataGridViewCellEventHandler(serviceDataGridView_CellClick);
                    staffDataGridView.CellClick += new
DataGridViewCellEventHandler(staffDataGridView_CellClick);
                }
                DataBaseWorker db = new DataBaseWorker();

                private void button2_Click(object sender, EventArgs e)
                {
                    if (textBox6.Text == "" || textBox7.Text == "" || textBox8.Text ==
"" || textBox9.Text == "" || textBox10.Text == "")
                    {
                        MessageBox.Show("Заповніть усі поля!");
                    }
                    else
                    {
                        OrderClass qw = new OrderClass();
                        Number = Convert.ToInt32(textBox6.Text);
                        Auto = Convert.ToString(textBox7.Text);
                        Service = Convert.ToInt32(textBox8.Text);
                        Date = Convert.ToString(textBox9.Text);
                        Performer = Convert.ToString(textBox10.Text);
                        qw.UpdateOrder(Number, Auto, Service, Date, Performer,
db.sqlConnection);

                    }
                }
```

```
                private void button1_Click(object sender, EventArgs e)
                {
                    if (textBox1.Text == "" || textBox2.Text == "" || textBox3.Text ==
"" || textBox4.Text == "" || textBox5.Text == "")
                    {
                        MessageBox.Show("Заповніть усі поля!");
                    }
                    else
                    {
                        OrderClass qw = new OrderClass();
                        Number = Convert.ToInt32(textBox1.Text);
                        Auto = Convert.ToString(textBox2.Text);
                        Service = Convert.ToInt32(textBox3.Text);
                        Date = Convert.ToString(textBox4.Text);
                        Performer = Convert.ToString(textBox5.Text);
                        qw.InsertOrder(Number, Auto, Service, Date, Performer,
db.sqlConnection);
                    }
                    string[] row = new string[]{
                    textBox2.Text,
                    textBox12.Text,
                    textBox13.Text
                     };
                    dataGridView1.Rows.Add(row);
                    int sum = 0;
                    // Проходження по всім рядкам у стовпці DataGridView та додавання
їх значень до суми
                    foreach (DataGridViewRow row1 in dataGridView1.Rows)
                    {
                        if (row1.Cells[Column3.Index].Value != null) // Переконайтеся,
що вказали індекс вашого стовпця замість ColumnName
                        {
                            int cellValue;
                            if
(int.TryParse(row1.Cells[Column3.Index].Value.ToString(), out cellValue))
                            {
                                sum += cellValue;
                            }
                        }
                    }
                    // Виведення результату суми у TextBox
                    textBox15.Text = sum.ToString();
                    orderTableAdapter.Fill(dataSet1.Order);




                }

                private void button3_Click(object sender, EventArgs e)
                {
                    if (textBox11.Text == "")
                    {
                        MessageBox.Show("Виберіть поле для видалення!");
                    }
                    else
                    {
                        OrderClass qw = new OrderClass();
                        Number = Convert.ToInt32(textBox11.Text);
                        qw.DeleteOrder(Number, db.sqlConnection);

                    }
                }

                private void orderDataGridView_CellClick(object sender,
DataGridViewCellEventArgs e)
                {
```

```
                    if (tabControl1.SelectedTab == tabPage2)
                    {
                        if (e.RowIndex >= 0)
                        {
                            DataGridViewRow row = orderDataGridView.Rows[e.RowIndex];
                            string value = row.Cells[e.ColumnIndex].Value.ToString();
                            textBox6.Text = value;
                        }
                    }
                    if (tabControl1.SelectedTab == tabPage3)
                    {
                        if (e.RowIndex >= 0)
                        {
                            DataGridViewRow row = orderDataGridView.Rows[e.RowIndex];
                            string value = row.Cells[e.ColumnIndex].Value.ToString();
                            textBox11.Text = value;
                        }
                    }
                }

                private void label1_Click(object sender, EventArgs e)
                {

                }

                private void autoDataGridView_CellClick(object sender,
DataGridViewCellEventArgs e)
                {
                    if (tabControl1.SelectedTab == tabPage1)
                    {
                        if (e.RowIndex >= 0)
                        {
                            DataGridViewRow row = autoDataGridView.Rows[e.RowIndex];
                            string value = row.Cells[e.ColumnIndex].Value.ToString();
                            textBox2.Text = value;
                        }
                    }
                    if (tabControl1.SelectedTab == tabPage2)
                    {
                        if (e.RowIndex >= 0)
                        {
                            DataGridViewRow row = autoDataGridView.Rows[e.RowIndex];
                            string value = row.Cells[e.ColumnIndex].Value.ToString();
                            textBox7.Text = value;
                        }
                    }
                }

                private void serviceDataGridView_CellClick(object sender,
DataGridViewCellEventArgs e)
                {
                    if (tabControl1.SelectedTab == tabPage1)
                    {
                        if (e.RowIndex >= 0)
                        {
                            string value1 =
serviceDataGridView.Rows[e.RowIndex].Cells[0].Value.ToString();
                            string value2 =
serviceDataGridView.Rows[e.RowIndex].Cells[1].Value.ToString();
                            string value3 =
serviceDataGridView.Rows[e.RowIndex].Cells[2].Value.ToString();

                            // виводимо значення у відповідні TextBox

                            textBox3.Text = value1;
                            textBox12.Text = value2;
                            textBox13.Text = value3;
```

```
                    }
                }
                if (tabControl1.SelectedTab == tabPage2)
                {
                    if (e.RowIndex >= 0)
                    {
                        DataGridViewRow row = serviceDataGridView.Rows[e.RowIndex];
                        string value = row.Cells[e.ColumnIndex].Value.ToString();
                        textBox8.Text = value;
                    }
                }

            }

            private void staffDataGridView_CellClick(object sender,
DataGridViewCellEventArgs e)
            {
                if (tabControl1.SelectedTab == tabPage1)
                {
                    if (e.RowIndex >= 0)
                    {
                        DataGridViewRow row = staffDataGridView.Rows[e.RowIndex];
                        string value = row.Cells[e.ColumnIndex].Value.ToString();
                        textBox5.Text = value;
                    }
                }
                if (tabControl1.SelectedTab == tabPage2)
                {
                    if (e.RowIndex >= 0)
                    {
                        DataGridViewRow row = staffDataGridView.Rows[e.RowIndex];
                        string value = row.Cells[e.ColumnIndex].Value.ToString();
                        textBox10.Text = value;
                    }
                }
            }

            private void pictureBox1_Click(object sender, EventArgs e)
            {
                orderTableAdapter.Fill(dataSet1.Order);
            }

            private void staffDataGridView_SelectionChanged(object sender,
EventArgs e)
            {

            }

            private void pictureBox2_Click(object sender, EventArgs e)
            {
                if (comboBox1.Text == "" || textBox14.Text == "")
                {
                    MessageBox.Show("Виберіть та введіть поле для фільтрації!");
                }
                {
                    if (comboBox1.Text == "Дата")
                    {
                        orderBindingSource.Filter = "Date LIKE '%" + textBox14.Text
+ "%'";
                    }
                    if (comboBox1.Text == "Номер")
                    {
                        orderBindingSource.Filter = "Auto LIKE '%" + textBox14.Text
+ "%'";
                    }
                }
            }
```

```
        private void pictureBox5_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void pictureBox4_Click(object sender, EventArgs e)
        {
            Form1 form1 = new Form1();
            form1.Show();
            Hide();
        }

        private void pictureBox3_Click(object sender, EventArgs e)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "Text Files|*.txt";
            saveFileDialog.Title = "Save Order";
            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                StringBuilder sb = new StringBuilder();
                sb.AppendLine("Order: " + saveFileDialog.FileName);
                sb.AppendLine();
                sb.AppendLine("Авто\tПослуга\tВартість");

                // Виведення даних DataGridView
                foreach (DataGridViewRow row in dataGridView1.Rows)
                {
                    string auto = row.Cells[0].Value?.ToString() ?? "";
                    string service = row.Cells[1].Value?.ToString() ?? "";
                    string cost = row.Cells[2].Value?.ToString() ?? "";
                    sb.AppendLine($"{auto}\t{service}\t{cost}");
                }

                sb.AppendLine();
                sb.AppendLine("Sum: " + textBox15.Text);

                // Запис рядка з відформатованими даними у файл
                File.WriteAllText(saveFileDialog.FileName, sb.ToString());
            }
        }

        private void pictureBox6_Click(object sender, EventArgs e)
        {
            int sum = 0;
            // Проходження по всім рядкам у стовпці DataGridView та додавання
їх значень до суми
            foreach (DataGridViewRow row in dataGridView1.Rows)
            {
                if (row.Cells[Column3.Index].Value != null) // Переконайтеся,
що вказали індекс вашого стовпця замість ColumnName
                {
                    int cellValue;
                    if (int.TryParse(row.Cells[Column3.Index].Value.ToString(),
out cellValue))
                    {
                        sum += cellValue;
                    }
                }
            }

            // Виведення результату суми у TextBox
            textBox15.Text = sum.ToString();
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
```

```
        }

        private void Order_Load(object sender, EventArgs e)
        {
            db.OpenConnection();
            this.staffTableAdapter.Fill(this.dataSet1.Staff);
            this.serviceTableAdapter.Fill(this.dataSet1.Service);
            this.autoTableAdapter.Fill(this.dataSet1.Auto);
            this.orderTableAdapter.Fill(this.dataSet1.Order);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.None;
            textBox4.Text = DateTime.Today.ToString("yyyy-MM-dd");
            textBox9.Text = DateTime.Today.ToString("yyyy-MM-dd");
            ToolTip toolTip = new ToolTip();
            toolTip.SetToolTip(textBox4, "2000-03-22");
            toolTip.SetToolTip(textBox9, "2000-03-22");
            if (orderDataGridView.Rows.Count > 0)
            {
                int lastRowIndex = orderDataGridView.Rows.Count - 1;
                DataGridViewRow lastRow = orderDataGridView.Rows[lastRowIndex];
                if (lastRow.Cells.Count > 0)
                {
                    DataGridViewCell lastCell = lastRow.Cells[0];
                    int lastValue;
                    if (int.TryParse(lastCell.Value?.ToString(), out
lastValue))
                    {
                        int incrementedValue = lastValue + 1;
                        lastCell.Value = incrementedValue.ToString();
                        textBox1.Text = incrementedValue.ToString();
                    }
                }
            }
```

ДОДАТОК Г
(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Хмельницький Національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

Кваліфікаційна робота на тему: “Програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу”

Виконав студент Фіть Василь Олександрович
Керівник: Онишко О.Г. канд. пед. наук, доцент.

Актуальність теми

Автомобільний транспорт забезпечує широкі можливості для мобільності. Він дозволяє людям подорожувати, діставатися до роботи, школи або інших місць, де вони повинні бути. Автомобіль дозволяє швидко переміщатися і зручно планувати свій час.
Автомобільний транспорт є важливою складовою економіки багатьох країн. Він створює робочі місця у виробництві автомобілів, паливних станціях, автосервісах, дилерських центрах та інших пов'язаних галузях.
У сьогоднішніх реаліях люди змушені працювати із гігантськими обсягами інформації розробка програмних продуктів, що використовуються для автоматизованого обліку. Системи являють собою потужні засоби, які здатні обробляти великі потоки інформації високої структурної складності за мінімум витраченого часу, забезпечуючи дружній діалог із користувачами.
Тема є досить актуальною в сучасному світі. Автоматизація процесів у сфері обслуговування клієнтів стає все більш важливою для підприємств, щоб забезпечити ефективну та якісну роботу.
Завдяки програмному забезпеченню для автоматизації обслуговування клієнтів шиномонтажу, підприємствам стає легше керувати всіма аспектами роботи з клієнтами, включаючи прийом замовлень, планування робіт, ведення обліку та звітності.

Мета та завдання проєкту

Метою даної роботи є автоматизована система виробничого призначення, що здійснюватиме збирання інформації з об'єкта керування, передачу, перетворення і обробку її, формуючи керуючі команди та виконуючи їх на керованому об'єкті, тобто прискорити функції, які піддаються автоматизації. розробка та впровадження спеціалізованої програмної системи, яка допоможе підприємствам шиномонтажу покращити ефективність своєї роботи та надати якісне обслуговування своїм клієнтам.

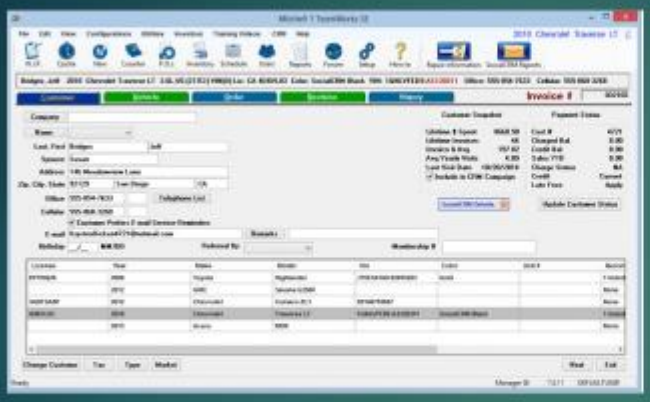
Основні завдання проєкту:

- дослідити предметну область авто траспорту та виявити потреби потенційних користувачів додатку;
- здійснити аналіз існуючих рішень;
- розробити технічне завдання;
- розробити архітектуру веб-застосунку та бази даних;
- обрати технології для розробки;
- Розробити необхідний функціонал програми;
- Розробити зручний користувацький інтерфейс;

Порівняння наявного ПЗ

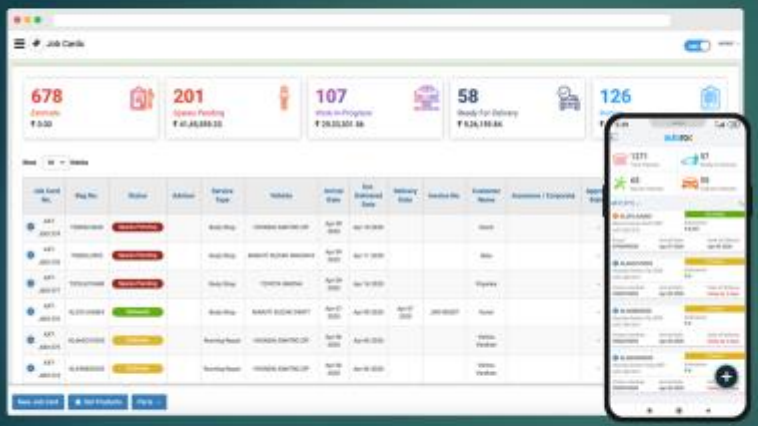
Щоб зрозуміти яким чином можна досягти у програмі максимально зручного інтерфейсу, а функціонал найбільше відповідав меті, необхідно розглянути технології, існуючі методи та інші програми для збереження, фільтрування та редагування даних, перегляду інформації, формування звітності, тощо.

AutoShop Management System



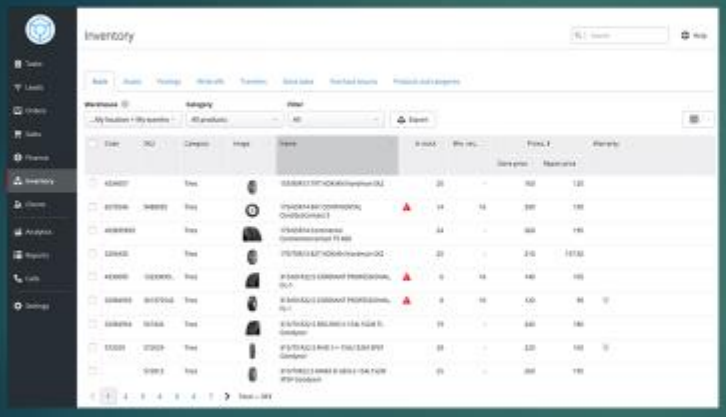
Переваги	Недоліки
- Простий інтерфейс; - легкість використання; - швидке налаштування.	- Обмежені можливості налаштування; - відсутність додаткових модулів або розширених аналітичних звітів.

Garage Management Software



Переваги	Недоліки
- Широкі можливості налаштування; - інтеграція з іншими системами; - загальний функціонал для шиномонтажного центру; -Розширений функціонал для автосервісу	- Складність освоєння для новачків; - вимагає більшої часової та фінансової витрати на налаштування.

Tire Shop Management System

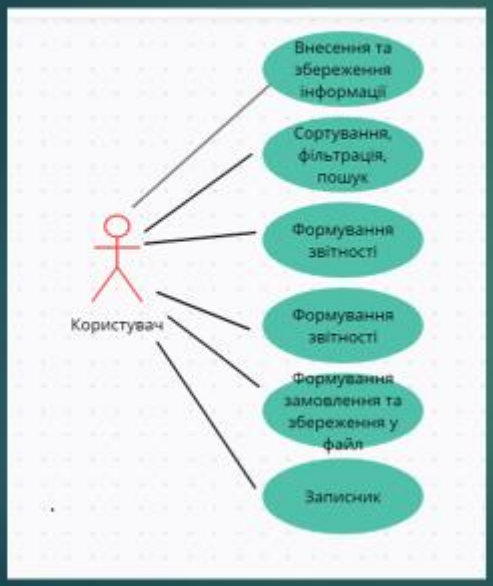


Переваги	Недоліки
- Зручний інтерфейс; - простота використання; - швидке внесення даних; - аналітика продажів; - спеціалізований функціонал.	- Обмежена функціональність порівняно з іншими системами; - відсутність додаткових модулів для розширення функціоналу.

Функціональні вимоги

- Внесення інформації до БД про: клієнтів, авто, послуги сервісу, працівників, та замовлення;
- Редагування, видалення інформації про: клієнтів, авто, послуги сервісу, працівників, та замовлення;
- Пошук даних за заданими атрибутами;
- Фільтрація даних за заданими атрибутами;
- Сортування даних за заданими атрибутами;
- Формування звітності по даним що знаходяться в базі;
- Можливість друку звітності;
- Створення заміток в записнику та збереження у файл.

Діаграма варіантів використання



Використані технології

Microsoft SQL Server

SQL Server Management

crystal reports

Microsoft Visual Studio

Однокористувацька архітектура ПЗ

Така архітектура передбачає три основні компоненти:

- автономність роботи;
- мобільність додатків;
- розвинений користувацький інтерфейс

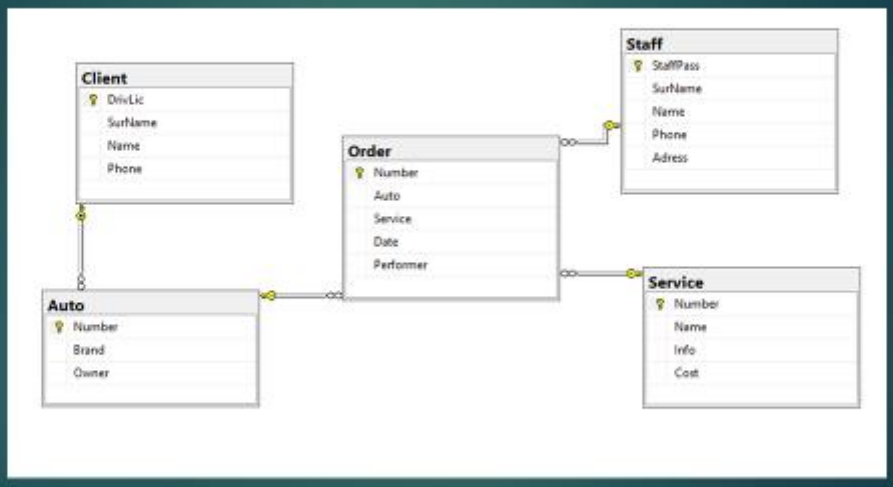
Прикладна програма - це програма, що реалізує прикладні функції (складського обліку, наприклад) Файли БД - це ті файли (на жорстких дисках комп'ютера), де зберігаються дані прикладної програми.

Файл - Сервер

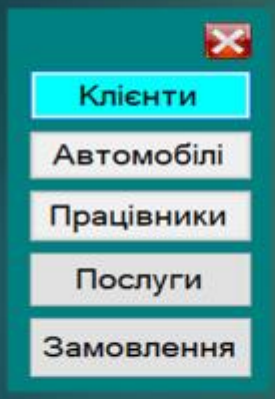
- Централізоване зберігання файлів;
- загальний доступ до спільних файлів;
- централізоване управління безпекою;
- ефективне використання ресурсів;
- зменшення обсягу передачі даних;
- централізоване оновлення і керування;
- підтримка резервного копіювання, відновлення;
- централізований моніторинг.

Рис. 1. Файл-серверна архітектура інформаційної системи

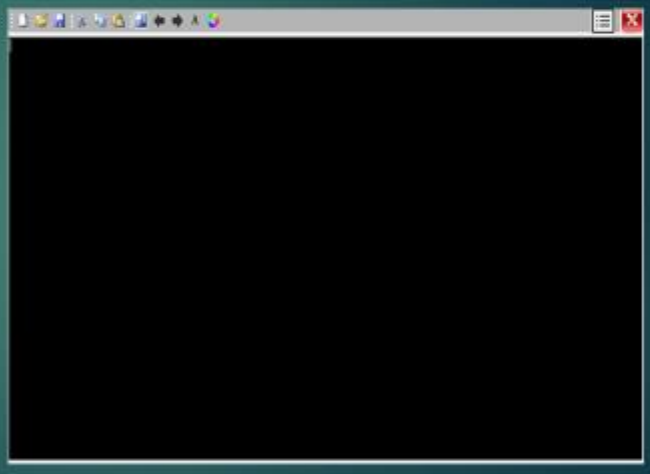
Схема бази даних



Форма “Меню”

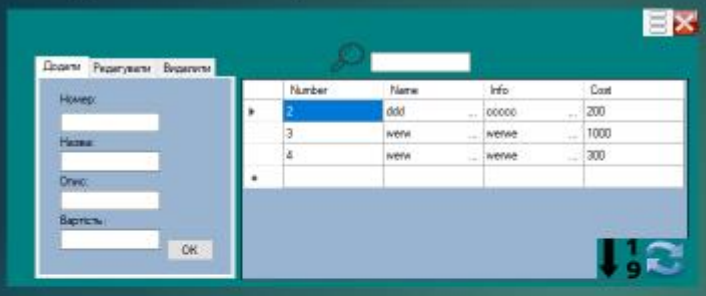


Форма “Записник”

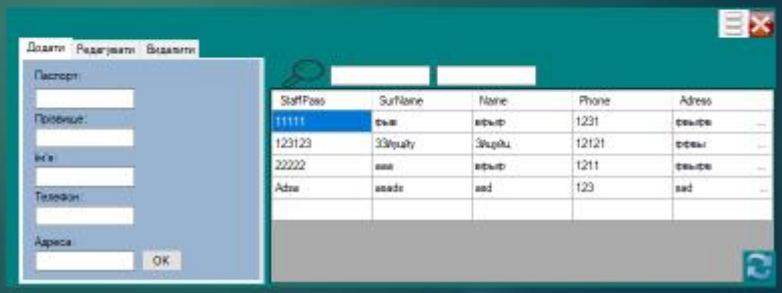


Реалізація

Форма “Послуги”

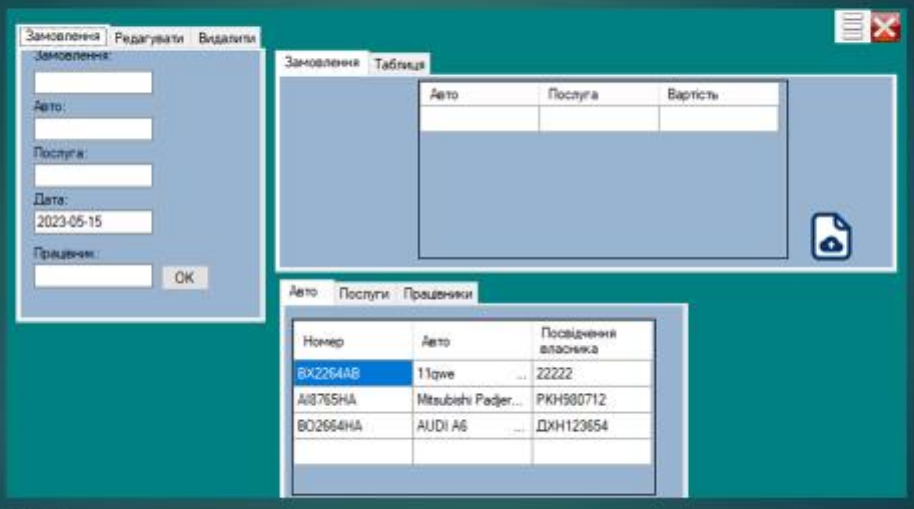


Форма “Працівники”



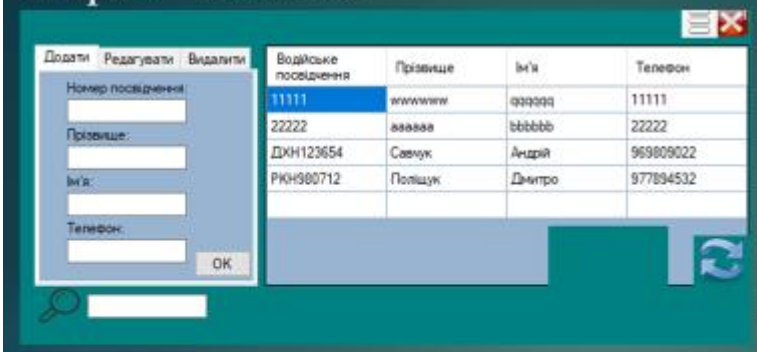
Реалізація

Форма “Замовлення”

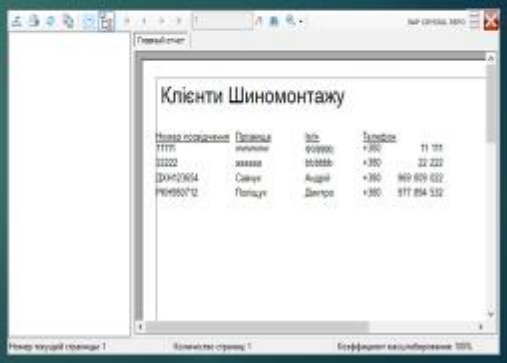


Реалізація

Форма “Клієнти”

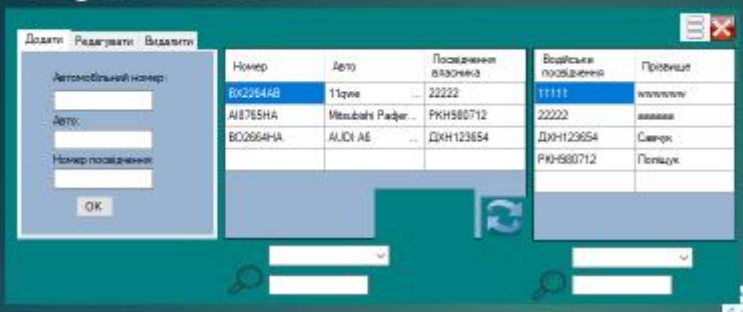


Звіт по клієнтам

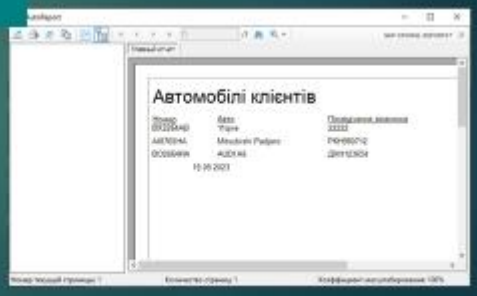


Реалізація

Форма “Авто”



Звіт по авомобілях



Реалізація

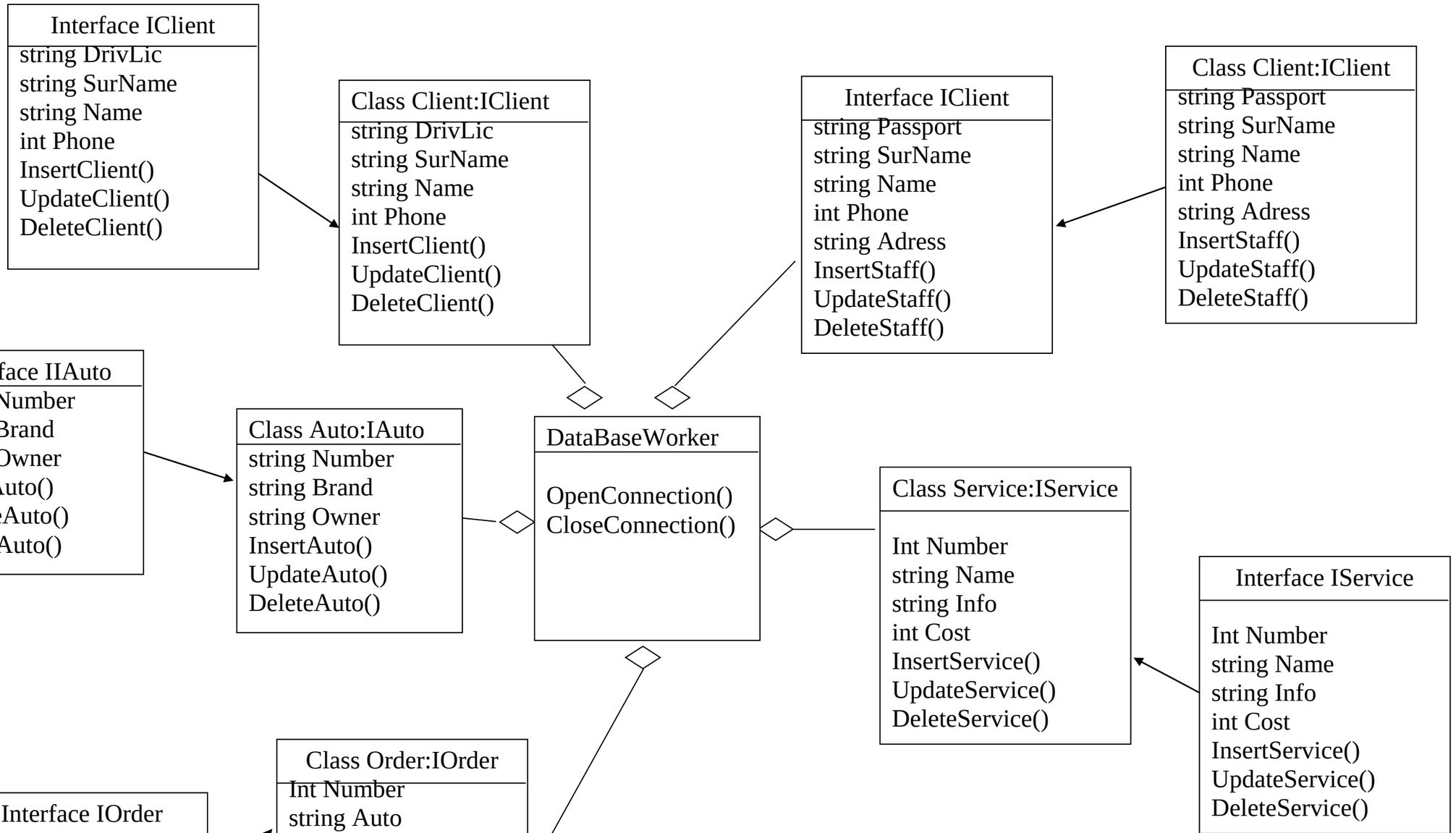
Висновки

Програмне забезпечення розроблено з використанням сучасних технологій та методів програмування. Воно включає в себе модуль для збору та обробки заявок клієнтів, який дозволяє ефективніше слідкувати за діяльністю сервісу. Крім того, в програмному забезпеченні присутні модулі для керування процесом роботи шиномонтажу, контролю якості виконаних робіт та зберігання інформації про клієнтів та їх автомобілі.

Результатом кваліфікайної роботи є спроектована архітектура та логіка проєкту, розроблена база даних, інтерфейс та реалізований додаток, який вимогам та поставленим задачам і повністю відповідає вимогам.

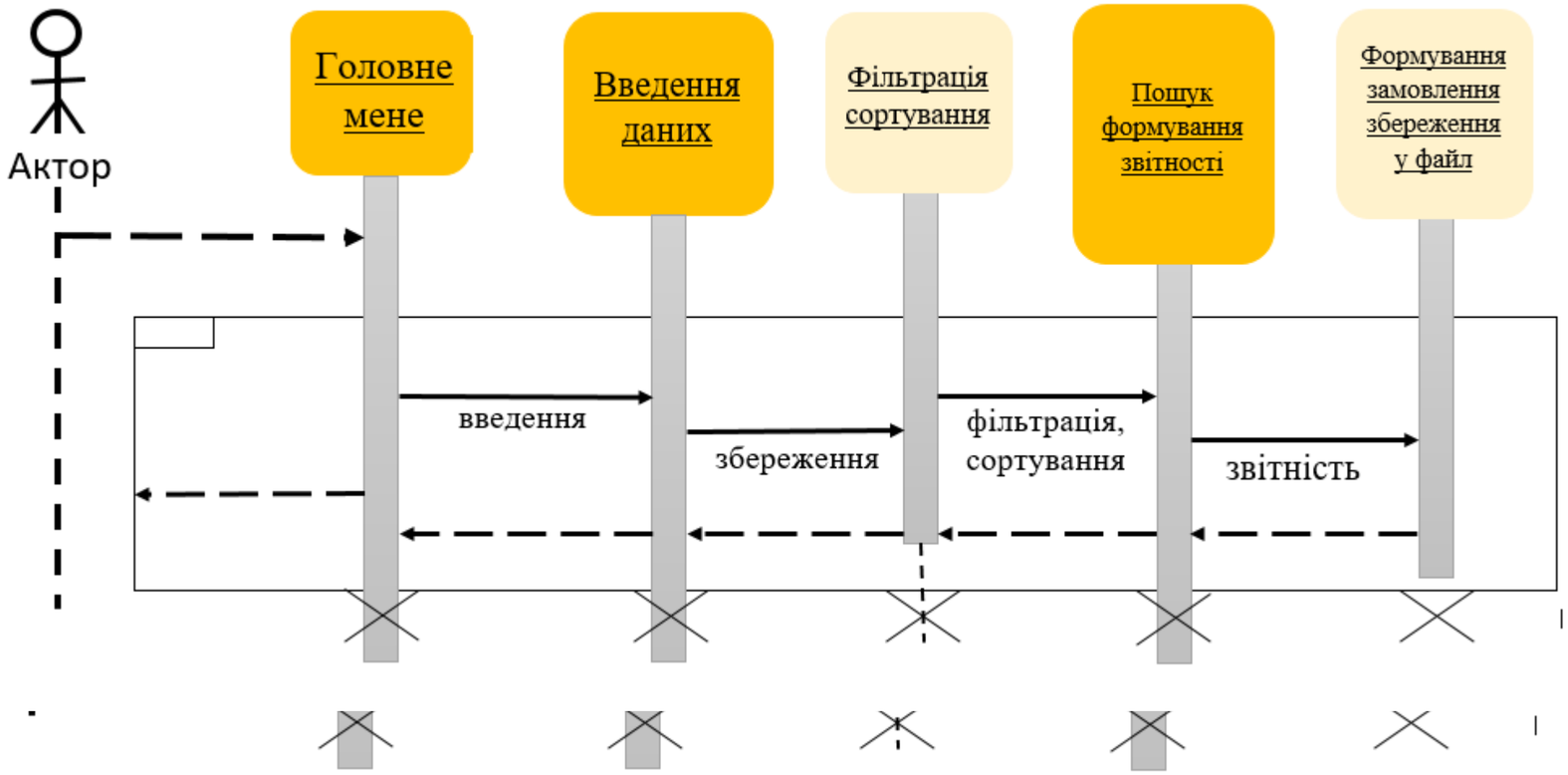
Графічна частина

Діаграма класів



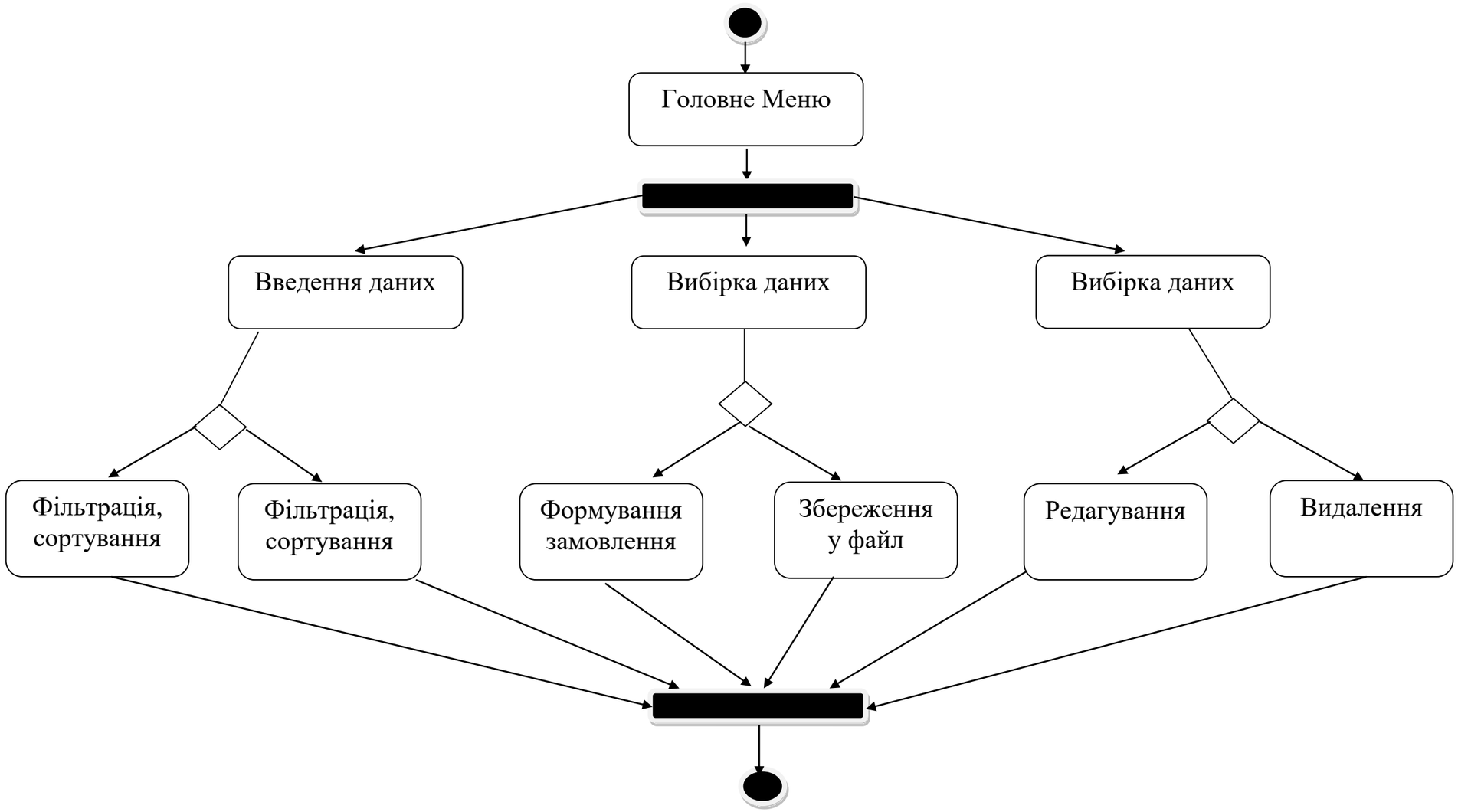
					КвРІП3.200123.01.08.Е8	
Зм.	Арк.	№докум.	Підпис	Дата	Програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу Діаграма класів	Літера / Маса / Масштаб
Розробив		Фіть В.О.		22.05.23		
Керівник		Онишко О.Г.		22.05.23		Аркуш 1 / Аркушів 3
Консульт.						
Н.Контр.		Гурман І.В.		22.05.23		ХНУ, ІПЗс-20-1
Зав.каф.		Бедратюк Л.П.		22.05.23		

Діаграма послідовності



					КвРІП3.200123.01.08.Е8		
Зм.	Арк.	№докум.	Підпис	Дата	Програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу Діаграма послідовності	Літера	Маса / Масштаб
Розробив		Фіть В.О.		22.05.23			
Керівник		Онишко О.Г.		22.05.23		Аркуш 2	Аркушів 3
Консульт.							
Н.Контр.		Гурман І.В.		22.05.23		ХНУ, ІП3с-20-1	
Зав.каф.		Бедратюк Л.П.		22.05.23			

Діаграма взаємодії



					КвРІП3.200123.01.08.Е8		
					Програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу Діаграма взаємодії	Літера	Маса / Масштаб
Зм.	Арк.	№докум.	Підпис	Дата			
Розробив		Фіть В.О.		22.05.23		Аркуш 3	Аркушів 3
Керівник		Онишко О.Г.		22.05.23			
Консульт.							
Н.Контр.		Гурман І.В.		22.05.23		ХНУ, ІПЗс-20-1	
Зав.каф.		Бедратюк Л.П.		22.05.23			

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗс-20-1
Фіть В. О.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: Програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу

(керівник роботи – Онишко Оксана Григорівна)
Прізвище, ім'я, по батькові

05.02.2023
Дата

Підпис студента

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Фіть В. О.
Прізвище, ініціали

факультет ІТ, 3 курс, група ІПЗс-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

05.06.2023
дата

підпис

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Фіть Василь Олександрович
Прізвище, ім'я, по батькові

студент III курсу факультету інформаційних технологій, кафедри інженерії програмного забезпечення
здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)/ науково-педагогічний працівник (назва кафедри)

назва факультету

підтверджую, що ознайомився (- лась) з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

« 05 » лютого 2023 р.

Підпис

Fri Jun 02 12:23:44 EEST 2023, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 7%**

ID: 114572 Назва: БКР Програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу Додано в БД: 2023-06-02 Автора: Фіть В.О. Керівники: Онищко О.Г. к.п.н. доц. Консультанти: Опоненти:	Документ		Сумарний збіг по Бази Даних	
	Символи	Лексеми	Символи	Лексеми
	92534	786	4178 (5%)	53 (7%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

Дата перевірки:
02.06.2023 13:21:37 EEST

Дата звіту:
02.06.2023 13:23:12 EEST

ID перевірки:
1015392720

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100005589

Назва документа: БКР Фіть ІПЗс 20-1

Кількість сторінок: 74 Кількість слів: 13797 Кількість символів: 107334 Розмір файлу: 1.98 MB ID файлу: 1015057200

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

9.22%
Схожість

Найбільша схожість: 1.98% з джерелом з Бібліотеки (ID файлу: 1011233803)

7.8% Джерела з Інтернету 748 Сторінка 76

3.56% Джерела з Бібліотеки 126 Сторінка 80

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування 13 сторінок

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом (ами) перевірки текстів на плагіат:

Назва: Програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу

Автор: Фіть Василь Олександрович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Онишко Оксана Григорівна, кандидат педагогічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 9,22% і адресується до 768 джерел, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 5.06.2023

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ОНИШКО

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Фіть Василь Олександрович

Тема Програмне забезпечення для автоматизації обслуговування клієнтів шиномонтажу

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 76

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі обґрунтовано актуальність теми та детально досліджено предметну область, визначено усі функціональні та нефункціональні вимоги. Був проведений огляд досліджень та робіт пов'язаних з темою, розглянуто їх переваги і недоліки. Визначено проблеми що потребують дослідження. Використані методи та підходи до дослідження. Розглянуто технології та інструменти для реалізації спроектованого застосунку, за допомогою яких було створено програмне забезпечення. проведено експерименти та оцінку ефективності. Результатом роботи є розробка і реалізація системи, яка вирішує актуальну проблему. Проведено тестування. Робота має значний внесок у практичну сферу і має потенціал для розвитку.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступній частині дипломного проектування було обґрунтовано актуальність теми і сформульовано мету та завдання проекту. В першому розділі був проведений аналіз області дослідження, оглянуті наявні рішення і визначені функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення. Другий розділ присвячено аналізу сучасних архітектур, їх перевагам і недолікам, та встановлено, що обрана система буде використовувати монолітну архітектуру та модель клієнт-сервер. У третьому розділі були підготовлені всі залежності для розробки коду та виконана практична реалізація програмних модулів з детальним описом їх особливостей, що призвело до створення програмного продукту. Крім того, у цьому розділі було проведено модульне тестування системи відповідно до функціональних вимог, що підтвердило коректтну роботу програми.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки на сьогодні в Україні Telegram-боти для пошуку роботи не є достатньо розвинутими та не мають достатньої кількості функціональних можливостей. Також було застосовано новітні технології для побудови програмного продукту та актуальні архітектурні рішеня.

5. Негативні сторони роботи Робота розроблена для використання працівника шиномонтажу, але відсутнє вікно авторизації, на майбутнє буде коректно реалізувати авторизацію для користувача щоб відслідковувати який саме працівник виконував ті чи інші дії.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому К Кваліфікаційна робота в цілому заслуговує позитивного оцінювання. Матеріал пояснювальної записки має чітку структуру та послідовність, що допомагає легко розібратись у викладеному матеріалі з питань проектування. Викладений текст є зрозумілим та простим у сприйнятті, що сприяє ясному розумінню тематики проекту. Графічний матеріал надає візуальне уявлення про деталі проектування системи, дозволяючи зрозуміти їх наочно. Матеріали кваліфікаційної роботи були подані вчасно, згідно графіку.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ Кисіль Тетяна Миколаївна, Кандидат фізико-математичних наук, Доцент кафедри комп'ютерної інженерії та інформаційних систем (КНС) ХНУ

"05" червня 2023 р. _____
(підпис)