Чернівецький політехнічний ФАХОВИЙ коледж

Відділення програмування та радіотехніки

Циклова комісія інженерія програмного забезпечення

Допускається до захисту

завідувач відділення

Андрій ШПАК

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

до дипломного проєкту

молодшого спеціаліста

на тему: Інформаційно довідкова система «Автосервіс»

Виконав: студент IV курсу, групи 541

спеціальність 121

«Інженерія програмного забезпечення»

спеціалізація

«Розробка програмного забезпечення»

Дмитро ВАСЕНКОВ

Керівник: Ірина СЕРЬОГІНА

2022

**Анотація**

Пояснювальна записка складається з 54 сторінок, 3 ілюстрацій, 23 лістингів чистини коду, 2 додатків, 5 джерел, 50 рисунків та 2 таблиці.

Об`єкт проєктування: Інформаційно довідкова система «Автосервіс».

Мета проєктування: проаналізувати процес запису в автосервіс та розробити інформаційно довідкова система «Автосервіс».

Метод проєктування: технологія Spring, бібліотека React, в комбінації з технологіями реалізації патерну MVC. А також середовище розробки IntelliJ IDEA Ultimate, WebStorm, Visual Studio Code, мова програмування Java, мова розмітки HTML, мова стилів CSS, мова програмування JavaScript, бібліотека React та фреймворк React redux.

При проєктуванні програмного забезпечення на основі моделей патерну MVC та технології міграцій було створено базу даних в pgAdmin.

Прогнози розвитку програмного забезпечення в майбутньому: розширення функціоналу, додавання різних мов перекладу, додавання монетизації.

Ключові слова: сайт, Java, Spring, React, MVC, завдання, база даних.

**зміст**

[ВСТУП 4](#_Toc106403079)

[РОЗДІЛ 1. Передпроєктні дослідження 6](#_Toc106403080)

[1.1 Аналіз предметної області 6](#_Toc106403081)

[1.2 Аналіз існуючих аналогів 7](#_Toc106403082)

[1.3 Маркетингове дослідження інновацій 21](#_Toc106403083)

[РОЗДІЛ 2. Технічний проєкт 26](#_Toc106403084)

[2.1 Технічне завдання 26](#_Toc106403085)

[2.2 Моделювання предметної області 27](#_Toc106403086)

[2.3 Моделювання даних 29](#_Toc106403087)

[2.4 Проєктування інтерфейсу 35](#_Toc106403088)

[РОЗДІЛ 3. РОБОЧИЙ ПРОєКТ 38](#_Toc106403089)

[3.1 Засоби розробки 38](#_Toc106403090)

[3.2. Документи на супроводження ПЗ 40](#_Toc106403091)

[3.3 Тестування програмного продукту 49](#_Toc106403092)

[3.4 Охорона праці 55](#_Toc106403093)

[ВИСНОВКИ 57](#_Toc106403094)

[перелік джерел 58](#_Toc106403095)

[Додатки 59](#_Toc106403096)

# ВСТУП

Автосервіс є одним з найперспективніший напрямків розвитку бізнесу на нашому ринку. Автомобільний ремонт та технічне обслуговування – це

процвітаючий бізнес на глобальному рівні.

Метою дипломного проєкту є розробка вебдодатку , в якому реалізовано продаж послуг для ремонту авто .

Об’єктом дослідження є веб сайт автосервісу magnat auto.

Предметом дослідження є робота з інформацією для продажу послуг, сучасні технології розробки веб додатку та можливості їх роботи з реляційними базами даних.

Основним результатом розробки програмного забезпечення є:

1. адміністративна частина, в якій адміністратор має можливості роботи з інформацією про послуги та їх категорії, користувачів та їх замовлення;
2. зручний, інтуїтивно зрозумілий інтерфейс, який дає змогу користувачу безперешкодно користуватись програмним продуктом;
3. розроблений додаток, в якому користувач би мав змогу замовити потрібну йому послгу.

Завданням цього дипломного проєкту є створення мобільного додатку «автосервіс » ». Для створення були використані основні принципи ООП. Всі дані зберігаються на серверіp PostgresSQL для їх подальшого використання.

Для реалізації було використано:

Середовище розробки: IntelliJ IDEA Ultimate, WebStorm. Visual Studio Code

Локальний сервер pgAdmin;

Мова програмування: Java, JS, React.js.

Дипломний проєкт складається з програмного продукту та пояснюючої записки до нього.

В першому розділі представлено аналіз предметної області, що включає в себе аналіз предметної області, аналіз аналогів де представлені вже існуючі інформаційні моделі та маркетингове дослідження інновацій.

У другому розділі вказано вимоги до програмного продукту в технічному завданні, описано процес проєктування програмного забезпечення, наведено концептуальну модель, модель прецедентів, зображено фізичну та логічну моделі бази даних.

Третій розділ містить опис засобів розробки програмного продукту, описано, функції, наведено інструкції для роботи з програмним забезпеченням, а також охорона праці.

У висновках підведено підсумки роботи над розробкою програмного продукту, вказано програмне забезпечення, що було використано в процесі розробки.

# РОЗДІЛ 1. Передпроєктні дослідження

## Аналіз предметної області

У сфері автосервісів навіть у наш час не часто використовують додатки у яких можливий попередній запис та слідкування за статусом роботи. В основному на станціях технічного обслуговування використовують тільки інструменти для звітності та збереження важливих даних.

Станції технчіного обслуговування – це місця де здійснюється ремонт, діагностика та удосконалення автівок.

Автосервіси в основному поділяють на три види:

1. Сервісні центри офіційного дилера – обслуговування автомобіля у цих центрах зберігає гарантію дилера на автомобіль.
2. Професійні автосервіси – це сервісні центри де також можливий ремонт авто, але без збереження гарантії на автомобіль.
3. «Гаражні автосервіс » - це місця де люди без можливості аренди обслуговують та ремонтують автомобілі у гаражних кооперативах.

Також є спеціалізовані автосервіси де надають конкретні послуги, такі як: лакофарбові роботи, ремонт кузова, ремонт ходової, тюнінг та інше.

Принципи роботи веб додатку можна описати так: клієнт реєструється та авторизується на сайті, вводить свої особисті дані. Надалі покупець вибирає послугу, яка його цікавить та оформлює замовлення, після цього з ним зв’язується за допомогою мобільного телефону або електронної адреси адміністратор сервісу, та уточнює умови оплати. При успіху клієнт прибуває на вказану адресу автосервісу.

Комерційне програмне забезпечення – програмне забезпечення, створене комерційною організацією з метою отримання прибутку від його використання.

Електронна комерція – це сфера цифрової економіки, що включає всі фінансові та торгові [транзакції](https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D1%96%D1%8F), які проводяться за допомогою комп’ютерних мереж, та [бізнес-процеси](https://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%B7%D0%BD%D0%B5%D1%81-%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81), пов’язані з проведенням цих транзакцій.

Для веб додатку обов’язковою є наявність таких функціональних частин:

1. оновлення наявного асортименту (послуг);
2. можливість додавати товари до «кошику»;
3. вхід для зареєстрованих користувачів;
4. оплата через Інтернет передбачає необхідність створення кількох облікових записів різних рівнів доступу.

Для забезпечення вказаних функцій веб додатку необхідно, щоб покупця задовольняла ціла низка особистостей, він має мати змогу:

1. ознайомитися з повним каталогом товарів та послуг, в стислому вигляді (ціна, тип послуги , його назва) та в розвернутому (повний опис послуги);
2. товар, складений в кошик можна переглянути в будь-який момент;
3. оплатити послугу на основі різних варіантів.

Для адміністративного управління веб додатком повинна бути передбачена взаємодія за допомогою звичайного браузера через спеціальний закритий вхід у систему, який дозволить користуватись панеллю адміністратора:

1. адміністратору створювати та редагувати каталог послуг;
2. адміністратору додавати, видаляти та модифікувати інформацію про послуги.

Отже, ми проаналізували предметну область та вимоги до веб додатку.

## Аналіз існуючих аналогів

Для аналізу предметної області було розглянуто веб сайт автосервіс

«Magnat auto».

«Magnat auto»– це станція технічного обслуговування в Чернівцях, В якому можливо дистанційно замовити обслуговування та діагностику автомобіля.

На головни сторінці сайту, першим ділом, ми можемо бачити його логотип, та кнопки з посиланнями на інші елементи сайту.

Верхній розділ головної сторінки веб сайту дозволяє:

* Перейти вниз до інформації про переваги даного сервісу
* Перейти до переліку послуг
* Дізнатись адресу сервісу
* Знайти контакти автомайстерні

це зображено на рисунку 1.1.

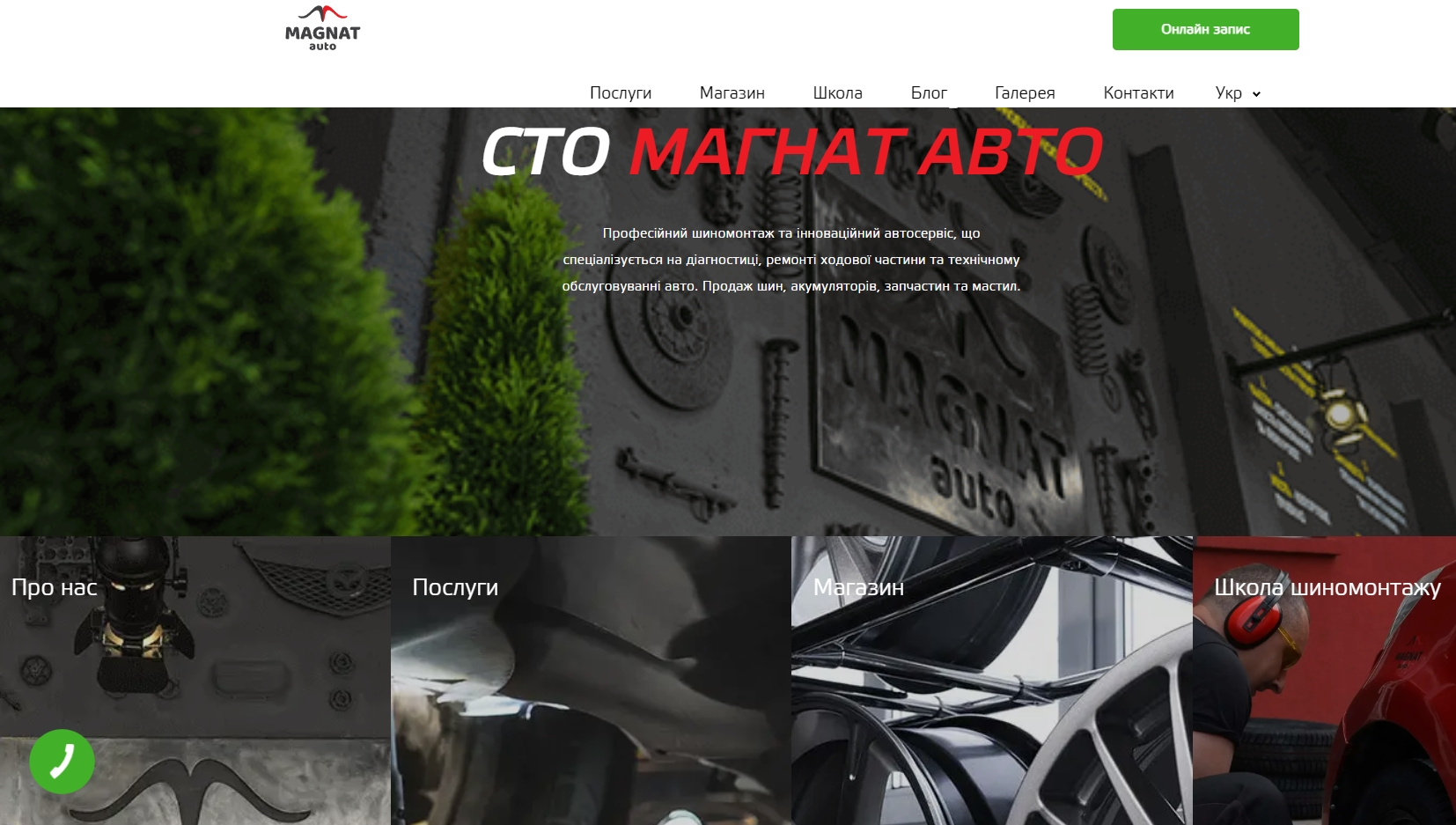


Рисунок 1.1 – Інтерфейс головної сторінки автосервісу «magnat auto»

Інтерфейс каталогу послуг є можливість відкрити з головної сторінки, великий вибір послуг, корзина відкривається модальним вікном, це все зображено нижче (рисунок 1.2).

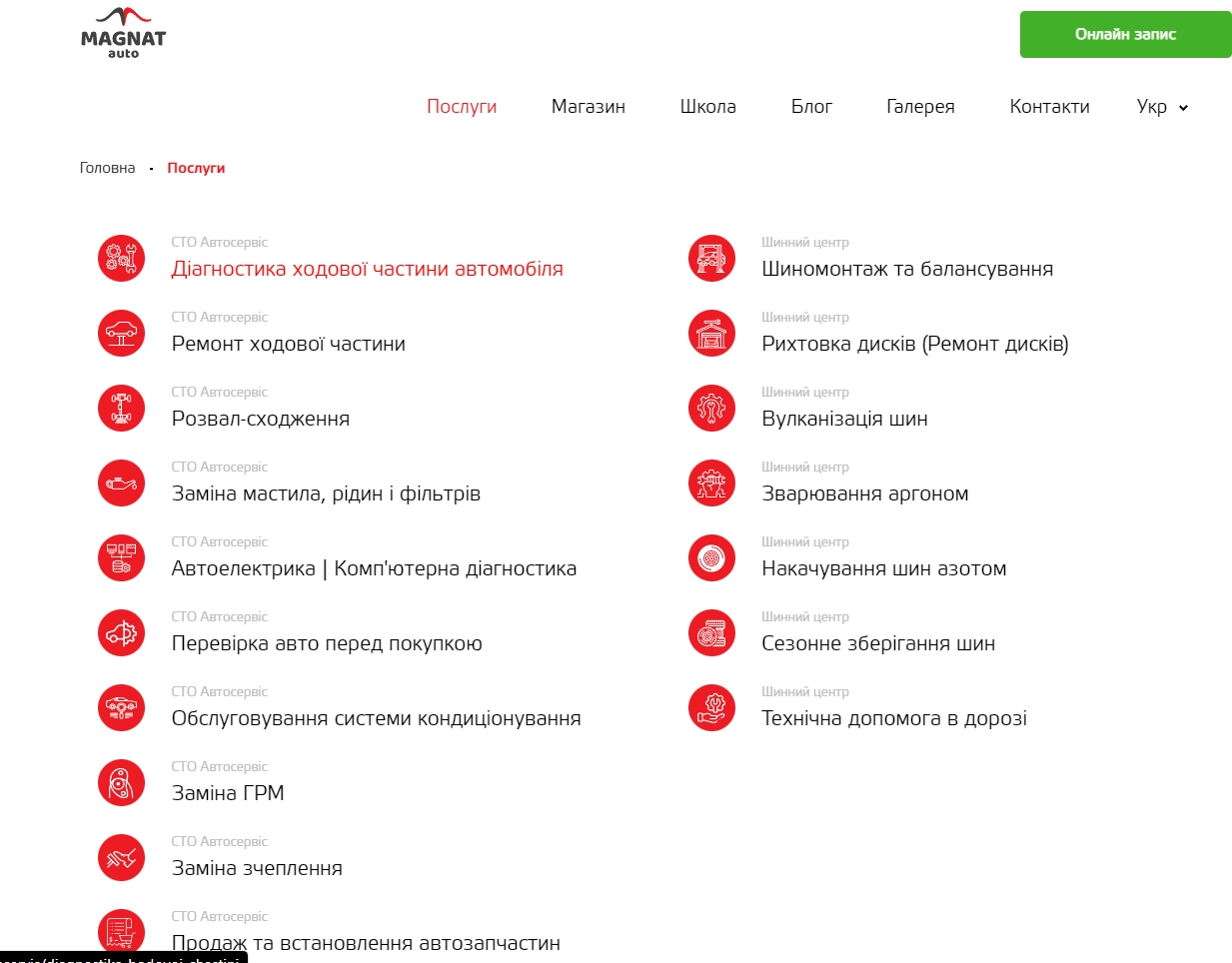


Рисунок 1.2 – Інтерфейс каталогу та вибору послуг сторінки автосервісу «magnat auto»

Ми бачимо перклік послуг які надає даний автосервіс. На цій сторінці клієнт може вибрати послугу яка йому потрібна і записатись онлайн (Рисунок 1.3)

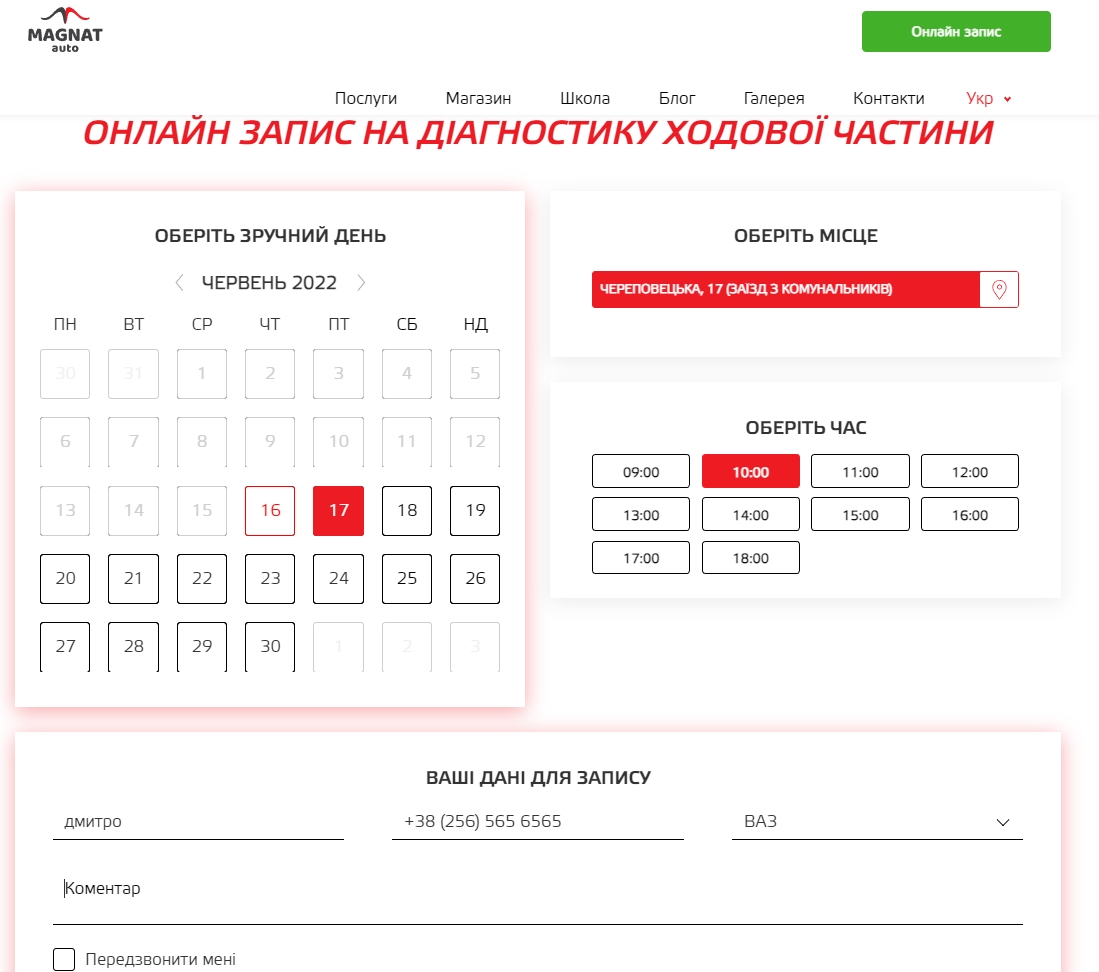


Рисунок 1.3 – «запис на діагностику »

Ми бачимо можливість запису на будь який день і час зручний для клієнта.

Клієнт може вибрати марку авто і лишити коментар якій допоможе спеціалісту

Краще розібратись в проблемі яка виникла у клієнта.

Аналізуючи даний веб додаток, можна зробити висновки, що перевагами є:

* гарний інтерфейс;
* можливість замовлення підтримки, такої як «чату з представником сервісу», так і можливість замовлення персонального дзвінка;
* великий вибір послуг;
* зручний запис на обслуговування авто.

А недоліками можна вважати:

* погана оптимізація;
* дизайн сайту є «важким», навіть для досить швидкого інтернету, адже деякі блоки завантажуються набагато повільніше, ніж інші.

## 1.3 Маркетингове дослідження інновацій

Інформаційний сайт що розробляється, є аналогом вже існуючих сайтів інтернет-магазинів. Потенційним користувачем може бути будь-яка людина, яка цікавиться покупкою товарів для манікюру та педикюру. Вебсайт орієнтований на одноосібне використання недосвідченого користувача.

З часом очікується попит на сайт, так як він буде зручним у використанні та мати інтуїтивно-зрозумілий інтерфейс.

Для оцінки конкурентоспроможності додатку потрібно визначити основні параметри базового та нового варіантів програмного забезпечення (ПЗ), виконати конструювання еталону конкурентоспроможності, розрахувати інтегральний показник конкурентоспроможності, визначити ефективність нового ПЗ порівняно з базовим.

Визначення основних параметрів базового і нового варіантів ПЗ.

У процесі дослідження виділяються основні технічні та економічні параметри базового та нового варіантів ПЗ (таблиця 1.1).

Таблиця 1.1 – Основні техніко-економічні параметри додатку

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Назва параметру | Одиниця виміру | Варіант | | Характеристика параметра нового варіанта відносно базового (↑,↓ чи =) |
| базовий  (аналог) | новий |
| Вартість шаблону | грн. | 18300 | 14903 | ↑ |
| Результати дослідження інтерфейсу методом GOMS | с | 6 | 8 | **↓** |
| Кількість функцій шаблону | шт. | 11 | 11 | = |
| Вага шаблону | Мб | 32 | 12 | ↑ |
| Кількість інформації, що передається | біт | 18 | 20 | ↑ |
| Можливість нарощування функціональних характеристик | бали | 3 | 5 | ↑ |
| Кількість людей, необхідних для обслуговування | чол. | 3 | 2 | ↑ |

Виходячи з отриманих результатів, розробка є кращою відносно базового варіанта за п’ятьма наступними параметрами: вартість шаблону, вага шаблону, можливість нарощування функціональних характеристик, кількість інформації, що передається, кількість людей, необхідних для обслуговування. Нова розробка отримала такий самий результат в одній наступній критерії: кількість функцій шаблону.

Для визначення конкурентоспроможності необхідно сконструювати еталон конкурентоспроможності додатку.

Еталон – це міра для відтворення, зберігання і передачі одиниць будь-якої величини, що утворена за таким правилом: серед показників-стимуляторів (здійснюється позитивний вплив на конкурентоспроможність) відбираємо дані з максимальним значенням, а серед показників дестимуляторів (чинить негативний плив на конкурентоспроможність) відбираємо дані з мінімальним значенням. Визначення еталонного значення наведено у таблиці 1.2 (колонки 2, 3).

Інтегральний показник конкурентоспроможності відносно еталона () визначається за формулою (1.1):

(1.1)

де , – величини за *і*-тим параметром відповідно базового, нового варіанта ПЗ та еталона.

Результати розрахунків заносимо до таблиці 1.2 (колонки 4, 5).

Проведені розрахунки будуть свідчити про те, що базове та нове ПЗ може переважати еталон за параметрами. Результатом буде одна із наступних умов:

1. якщо , то базове ПЗ перевищує новий зразок за конкурентоспроможністю;
2. якщо , то поступається йому;
3. при , нове ПЗ знаходиться на одному рівні з базовим.

Таким чином, базовий додаток переважає за еталон параметрами на «7,39», а новий – на «7». Тобто новий мобільний додаток є безперечно конкуренто-спроможним порівняно з базовим.

Загальна ефективність розробки нового ПЗ порівняно з базовим визначається за формулою (1.2):

(1.2)

Якщо серед параметрів є такі, для яких мінімальне значення є найбільш ефективним, то має місце рівність (1.3):

, (1.3)

де , – величини за *і*-тим параметром відповідно нового та базового варіанту ПЗ.

Проте, якщо серед параметрів є такі, для яких максимальне значення є найбільш ефективним, то має місце рівність (1.4):

, (1.4)

Для розрахунку загальної ефективності визначимо відносне значення параметрів нового ПЗ відносно базового (див. табл. 1.2, колонка 6).

Розрахунок ефективності окремого параметру нового ПЗ здійснюється за формулами (1.3) та (1.4), результати розрахунків заносимо до таблиці 1.2 (колонки 7, 8).

Таблиця 1.2 – Параметри конкурентоспроможності та показники ефективності

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Назва параметра | Еталонне значення | | Варіант | |  |  |  |
|  |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Вартість додатку | min | 14903 | 1.23 | 1 | 0,81 | 0,19 |  |
| Результати дослідження інтерфейсу методом GOMS | min | 8 | 0,71 | 1 | 1,4 | – 0,4 |  |
| Кількість функцій додатку | max | 11 | 1 | 1 | 1 |  | 0 |
| Вага додатку | min | 12 | 3,2 | 1 | 0,31 | 0,69 |  |
| Кількість інформації, що передається | max | 20 | 1,2 | 1 | 0,83 |  | 0,17 |
| Можливість нарощування функціональних характеристик | max | 5 | 0,5 | 1 | 2 |  | 1 |
| Кількість людей, необхідних для обслуговування | min | 2 | 2 | 1 | 0,5 | 0,5 |  |
| Разом: |  |  | 7,39 | 7 |  | 2,14 | |

Оскільки загальна вартість шаблону, що розробляється, є нижчою за вартість аналогу, то виграш від цього буде складати: 1 – 0,81 = 0,4. При розробці шаблону не вдалося покращити інтерфейс відносно базового, програш: 1 – 1,4 = –0,4. При запланованій кількості функцій шаблону покращити його не вдалось, показники однакові. Вага шаблону, що розробляється, є меншою за вагу аналога, тому отримуємо виграш у розмірі: 1 – 0,31 = 0,69. Передбачена кількість інформації, що передається, дозволить отримати виграш: 1 – 0,83 = 0,17, при запланована можливість нарощення функціональних характеристик: 2 – 1 = 1, та кількість людей, необхідних для обслуговування: 1 – 0,5 = 0,5.

Відповідно до цього, загальна економічна ефективність складе:

*Ефзаг* = 0,19 – 0,4 + 0 + 0,69 + 0,17 + 1 + 0,5 = 2,14.

Таким чином, ефективність виробництва нового вебпродукту відносно аналогу складає 2,14.

При виконанні маркетингового дослідження інновації розробки шаблону, виконавши усі необхідні розрахунки, встановлено, що технологія нової розробки відповідає оптимальному рівню витрат, і, у підсумку, розроблений шаблон та його заповнення є економічно доцільним та конкуренто-спроможним для вільного розповсюдження на ринку.

В даному розділі представлено аналіз предметної області, проаналізовано сайти аналогічного призначення, виконано постановку задачі, визначено основні функціональні вимоги до розробки та проведено маркетингове дослідження інноваційної розробки.

Отже, ми розглянули і обчислили коефіцієнт ефективності вебсайту.

# РОЗДІЛ 2. Технічний проєкт

## 2.1 Технічне завдання

Розробити веб додаток «Автосервіс» - в якому користувач зможе продивитись послуги та замовити. Є 2 рівня доступу користувач та адміністратор

Функції користувача:

* реєстрація;
* аторизація;
* перегляд профілю;
* перегляд послу;
* замовлення послуг;
* оформлення замовлення;
* відстеження статусу замовлення.

Функції адміністратора:

* додавання, редагування, видалення замовлень;
* додавання, редагування, видалення послуг;
* зміна статусів замовлень;
* перегляд звітності.

В програмному забезпеченні реалізовуватиметься обмеження між даними користувачів. Користувач не зможе дивитись чужі дані, перейшовши за посиланням в рядку браузера. При спробі це зробити користувач отримає відповідне повідомлення.

Інтерфейс програми повинен відповідати наступним вимогам:

* на сторінках сайту повинно бути не більше 3 основних кольорів;
* на сторінках сайту не повинно бути зайвих елементів інтерфейсу;
* сторінки сайту повинні підтримуватись сучасними браузерами;
* інтерфейс повинен бути інтуїтивно-зрозумілим;
* користувач повинен бачити актуальні дані на сторінках.

Основноюзадачею програмного продукту є значне спрощення пошуку користувачем та надати можливість слідкувати за своїм замовленням. Також додаток повинен надати можливість адміністратору зручно та швидко опрацьовувати замовлення, переглядати та редагувати замовлення.

## 2.2 Моделювання предметної області

Модель взаємодії користувача та адміністратора з програмним продуктом показана на моделі прецедентів(рисунок 2.1)

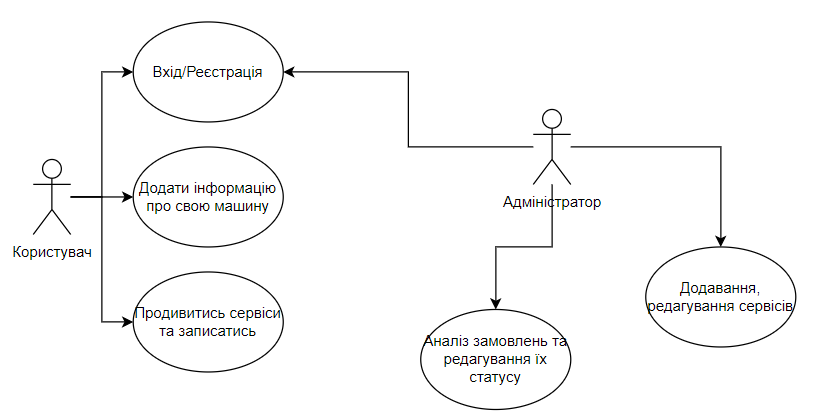


Рисунок 2.1 – Модель прецедентів

Діаграма послідовності процесу авторизації зображено на рисунку 2.2.

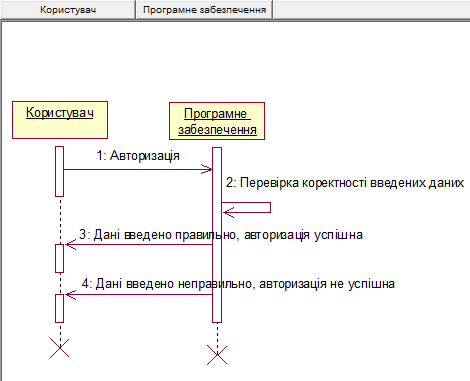


Рисунок 2.2 – Діаграма послідовності процесу авторизації

Під час авторизації користувача відбувається перевірка коректності введених даних. Якщо дані введено правильно, тоді авторизація пройде успішно і користувач зможе ввійти в систему. Якщо дані введено неправильно, тоді авторизація пройде невдало і виведеться відповідне повідомлення.

Діаграма послідовності процесів додавання та редагування даних зображено на рисунку 2.3.

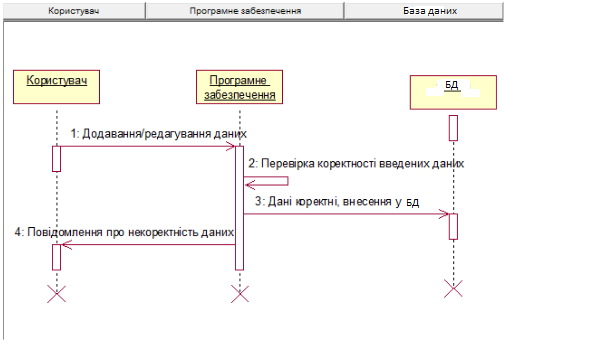


Рисунок 2.3 – Діаграма послідовності процесів обробки даних

Під час додавання чи редагування даних відбувається перевірка коректності введених даних. Якщо дані введено правильно, тоді інформація записується у базу даних. Якщо дані введено неправильно, тоді виведеться відповідне повідомлення.

## 2.3 Моделювання даних

Моделювання даних – це представлення структур даних у таблиці для бази даних і є дуже потужним вираженням бізнес-вимог. Ця модель даних є посібником, що використовується функціональними та технічними аналітиками при розробці та впровадженні бази даних.

Моделі даних використовуються для багатьох цілей, від концептуальних моделей високого рівня до фізичних моделей даних.

Для збереження даних використовується база даних на локальному сервері

pgAdmin з СУБД PostgreSQL.

Зв’язки між таблицями зображено на рисунку 2.4.

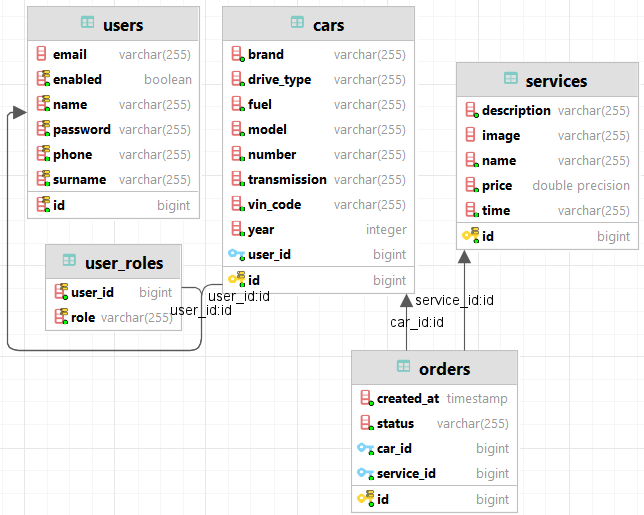


Рисунок 2.4 – Логічна модель даних

Структура таблиці користувачів users зображена на рисунку 2.5.

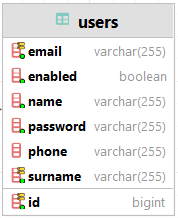


Рисунок 2.5 – Таблиця користувачів users

Вона містить такі поля:

* id – ідентифікатор;
* name – ім’я;
* surname – прізвище;
* email – пошта;
* phone – номер телефону;
* password – пароль;
* enabled – статус акаунта.

Структура таблиці ролів користувачів user\_roles зображена на рисунку 2.6.

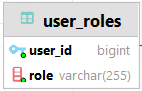


Рисунок 2.6 – Таблиця ролів користувачів user\_roles

Вона містить такі поля:

* user\_id – id користувача;
* role – роль.

Структура таблиці машин cars зображена на рисунку 2.7.

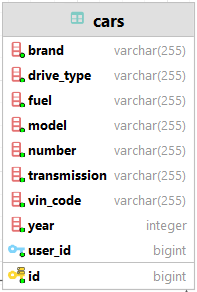


Рисунок 2.7 – Таблиця машин cars

Вона містить такі поля:

* id – ідентифікатор;
* user\_id – id користувача, якому належить машина;
* brand – бренд машини;
* model – модель машини;
* year – рік випуску;
* number – номер машини;
* vin\_code – код машини;
* fuel – паливо;
* transmission – трансмісія;
* drive\_type – тип приводу.

Структура таблиці сервісів services зображена на рисунку 2.8.

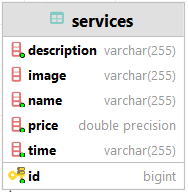


Рисунок 2.8 – Таблиця сервісів services

Вона містить такі поля:

* id – ідентифікатор;
* image – зображення;
* price – ціна;
* time – час виконання;
* description – описю

Структура таблиці прочитаних повідомлень read\_messages зображена на рисунку 2.10.

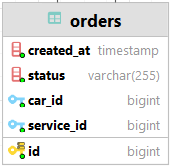


Рисунок 2.10 – Таблиця прочитаних повідомлень read\_messages

Вона містить такі поля:

* id – ідентифікатор;
* car\_id – id машини, яка записується на сервіс;
* service\_id – id сервіса;
* status – статус замовлення;
* created\_at – дата створення замовлення.

На рисунку 2.11 зображена діаграма взаємодій між класами в додатку. На ній можна переглянути поля класів та взаємодії між ними.

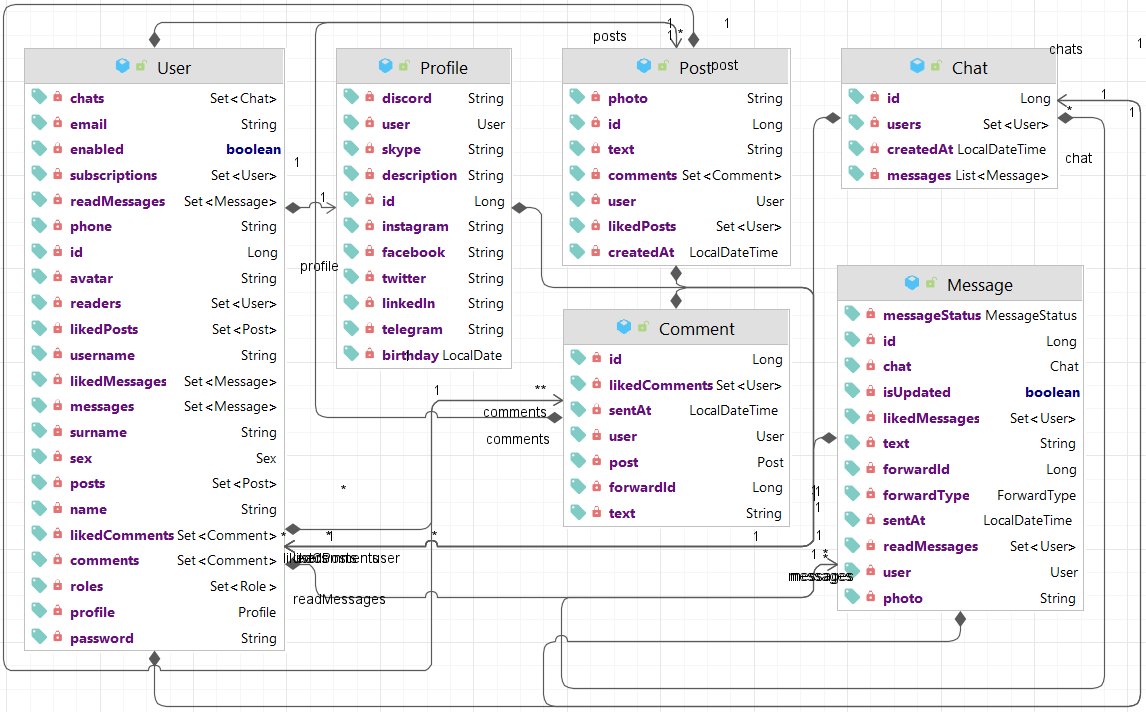


Рисунок 2.11 – Модель взаємодії класів

Отже, вище описано структуру та архітектруру бази даних проекта, взаємодія таблиць в базі даних, та взаємодія класів проекту.

## 2.4 Проєктування інтерфейсу

Необхідність розробки графічного інтерфейсу для сайтів «Instagram» та «Telegram» відпадає, оскільки він вже реалізований.

Дизайн додатку «Instagram» виконаний в білих та сірих тонах, а «Telegram» – блакитних, білих та сірих.

Для розробки інтерфейсу порталу використовувалась мова програмування JavaScript з бібліотекою React.js, мова розмітки HTML, таблиці каскадних стилів CSS.

React – відкрита JavaScript-бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків.

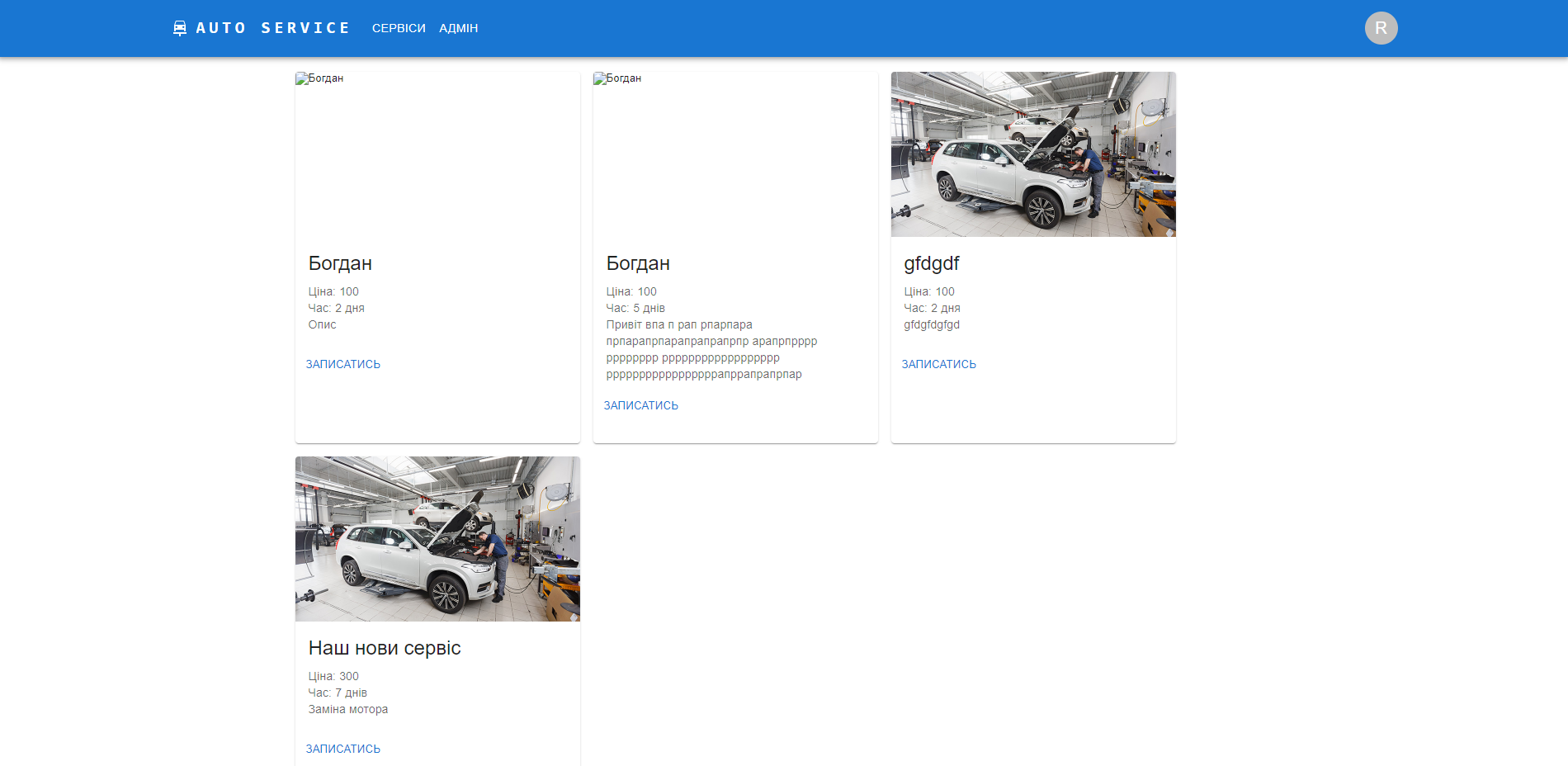


Рисунок 2.12 – Сервіси

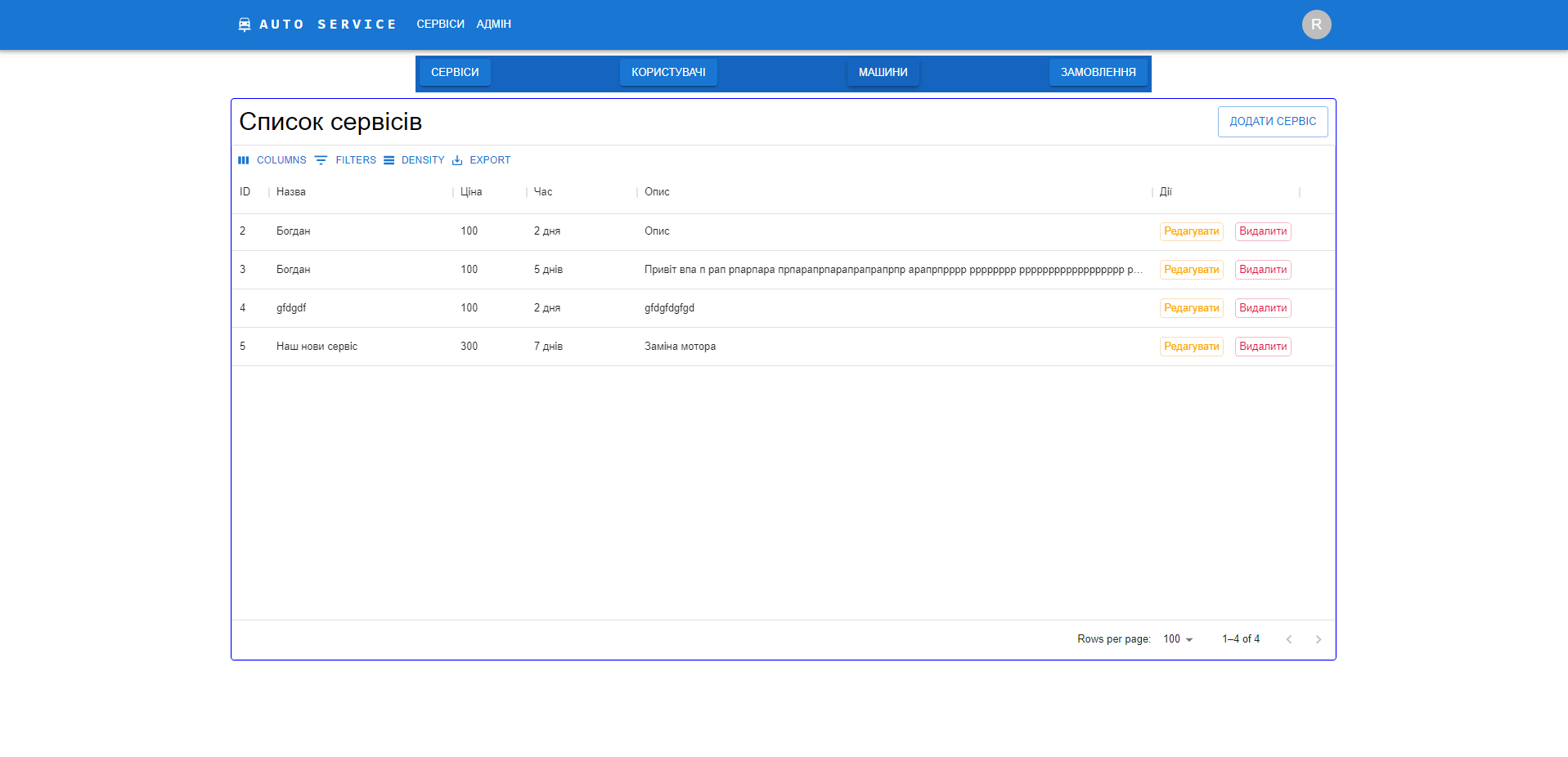


Рисунок 2.13 – Адмін панель(Сервіси)

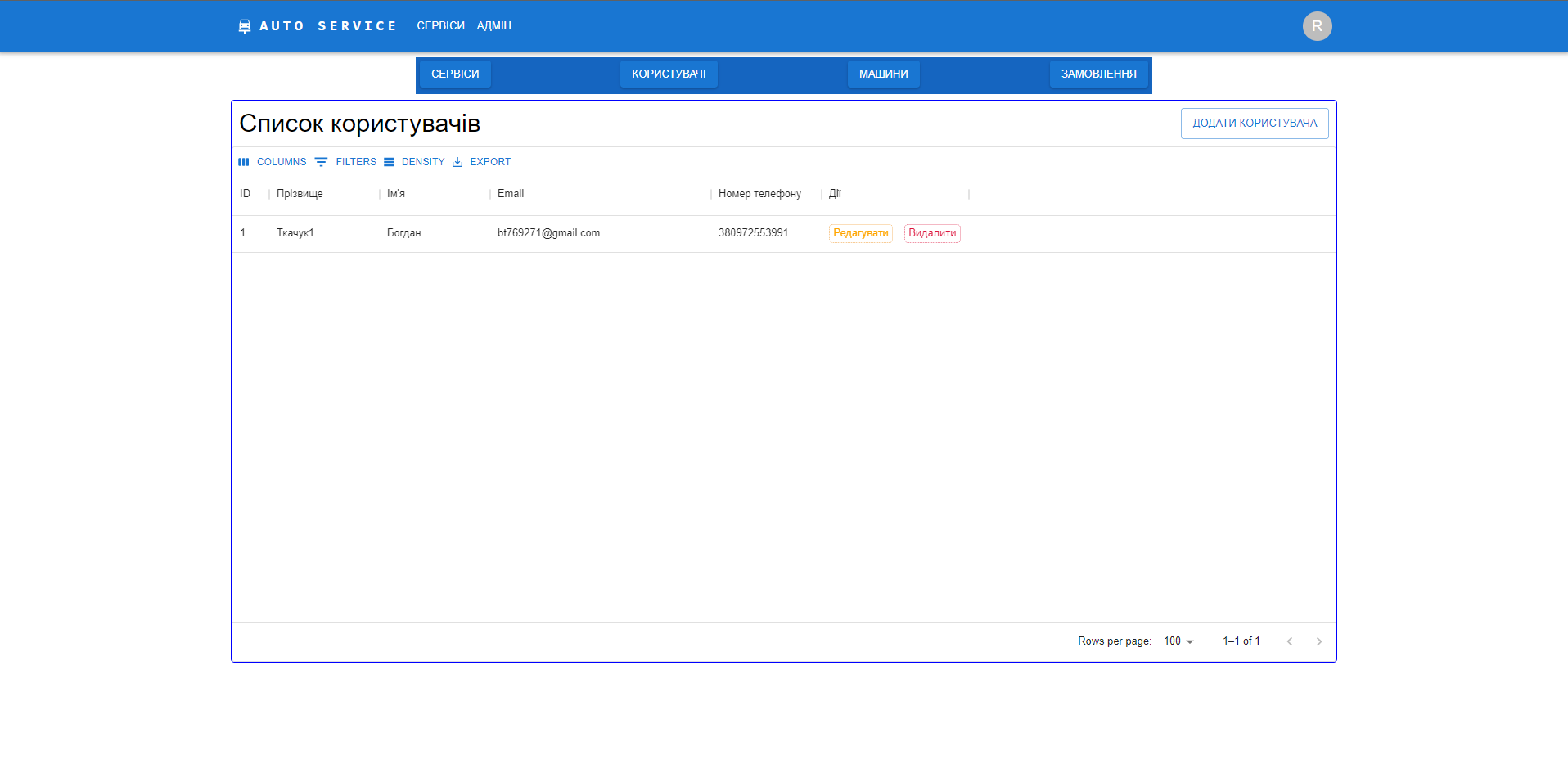


Рисунок 2.14 – Адмін панель(Користувачі)

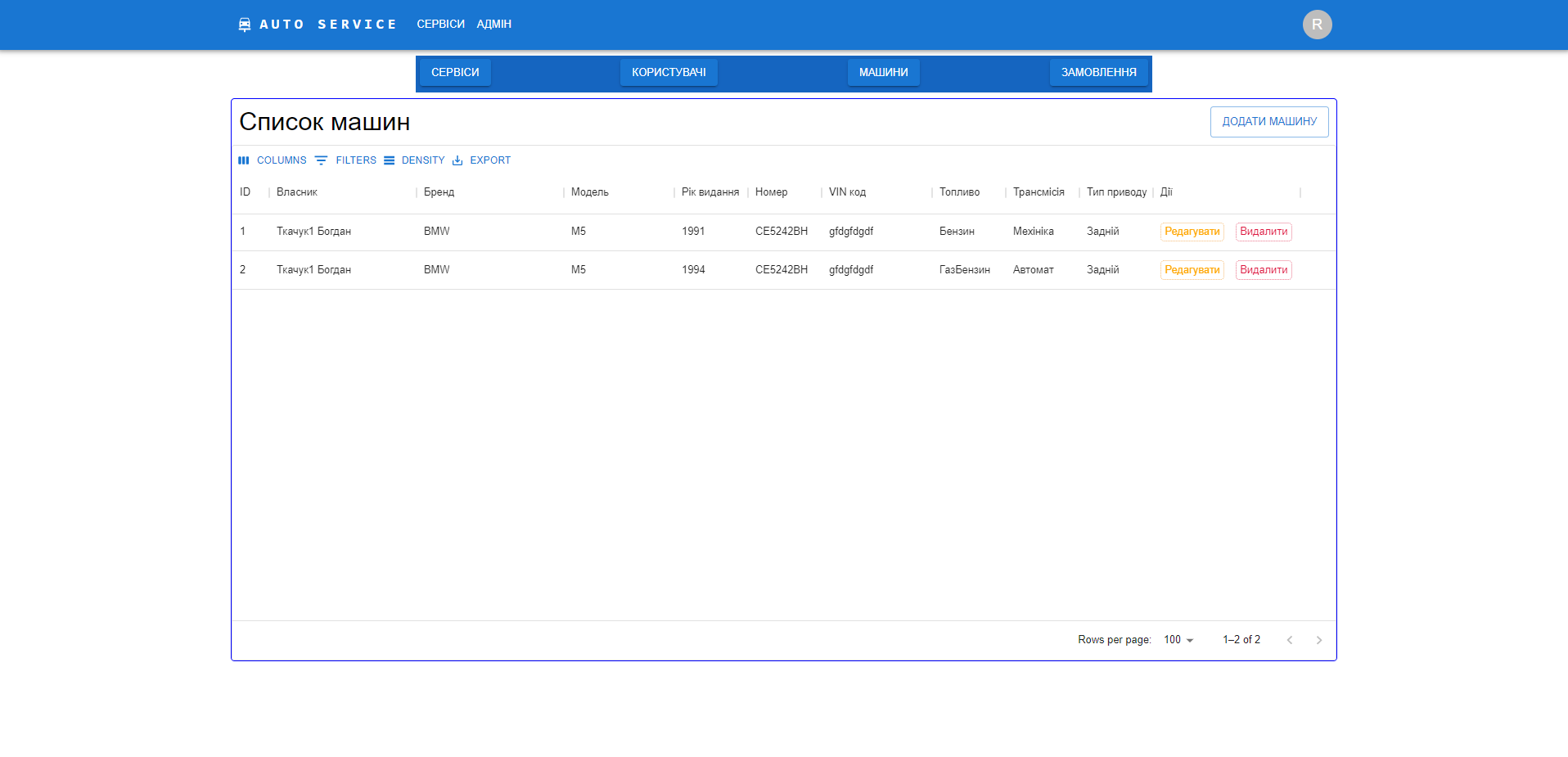


Рисунок 2.15 – Адмін панель(Мишини)

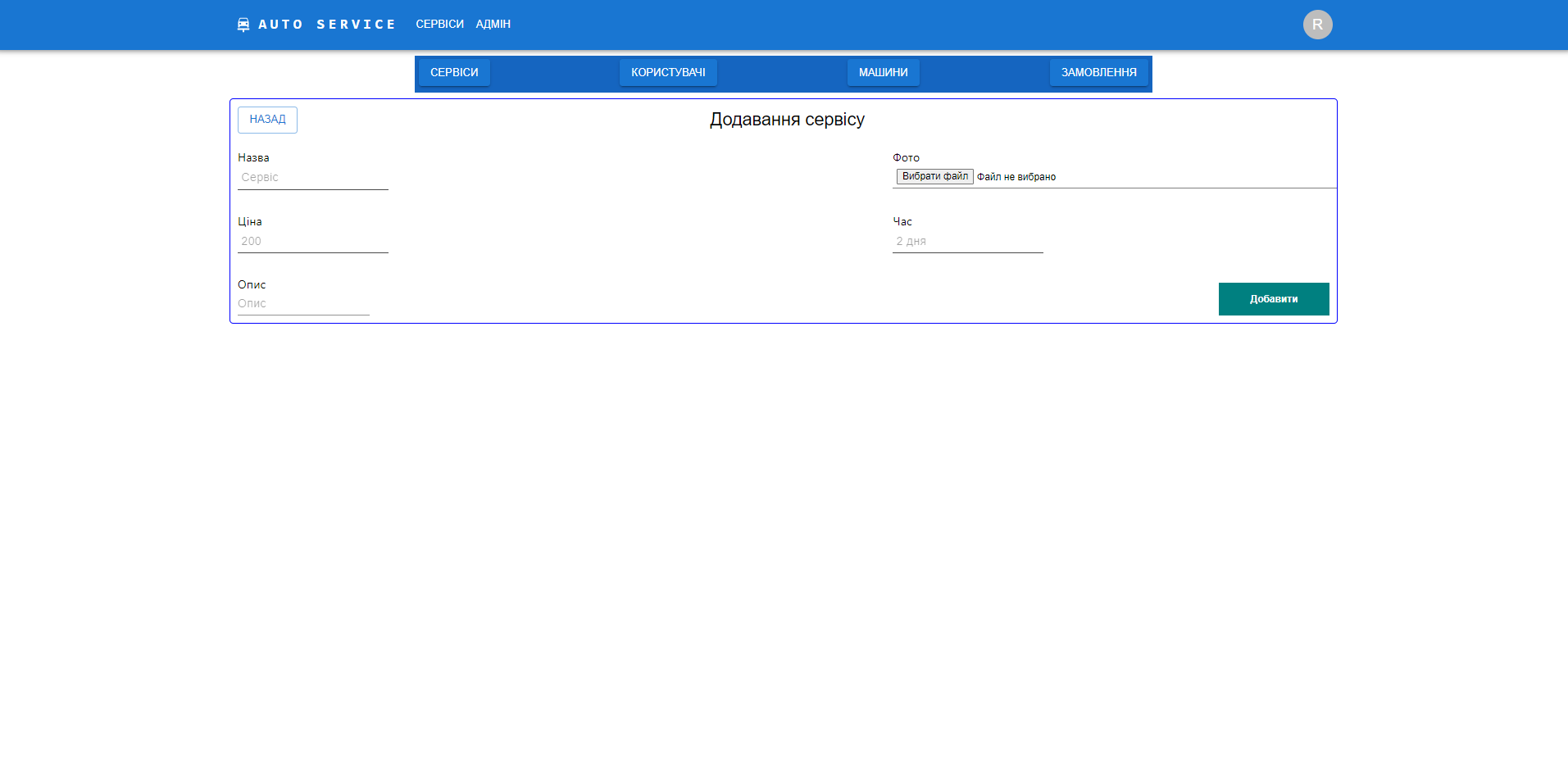


Рисунок 2.16 – Додавання сервісів

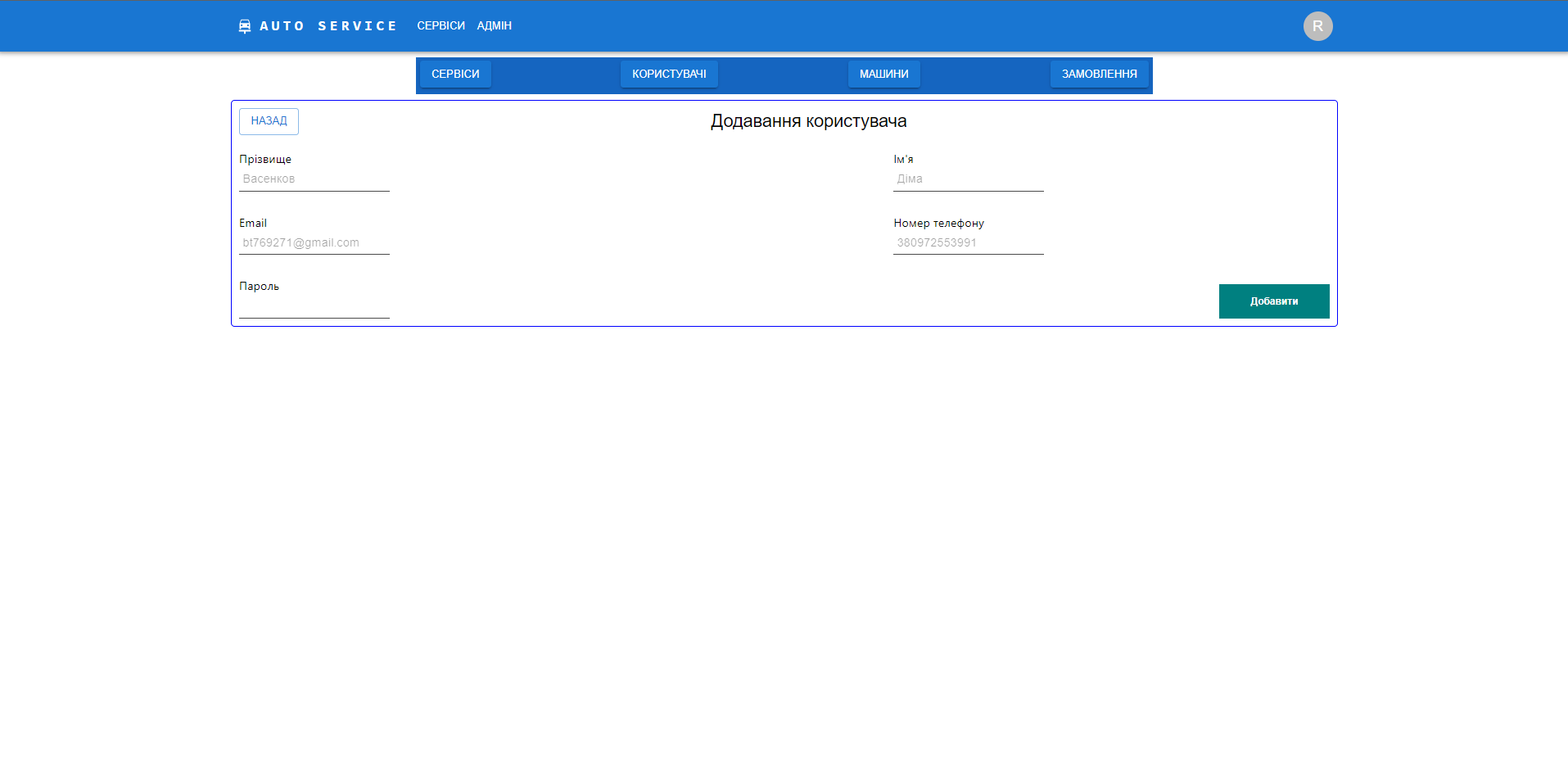


Рисунок 2.17 – Додавання користувачів

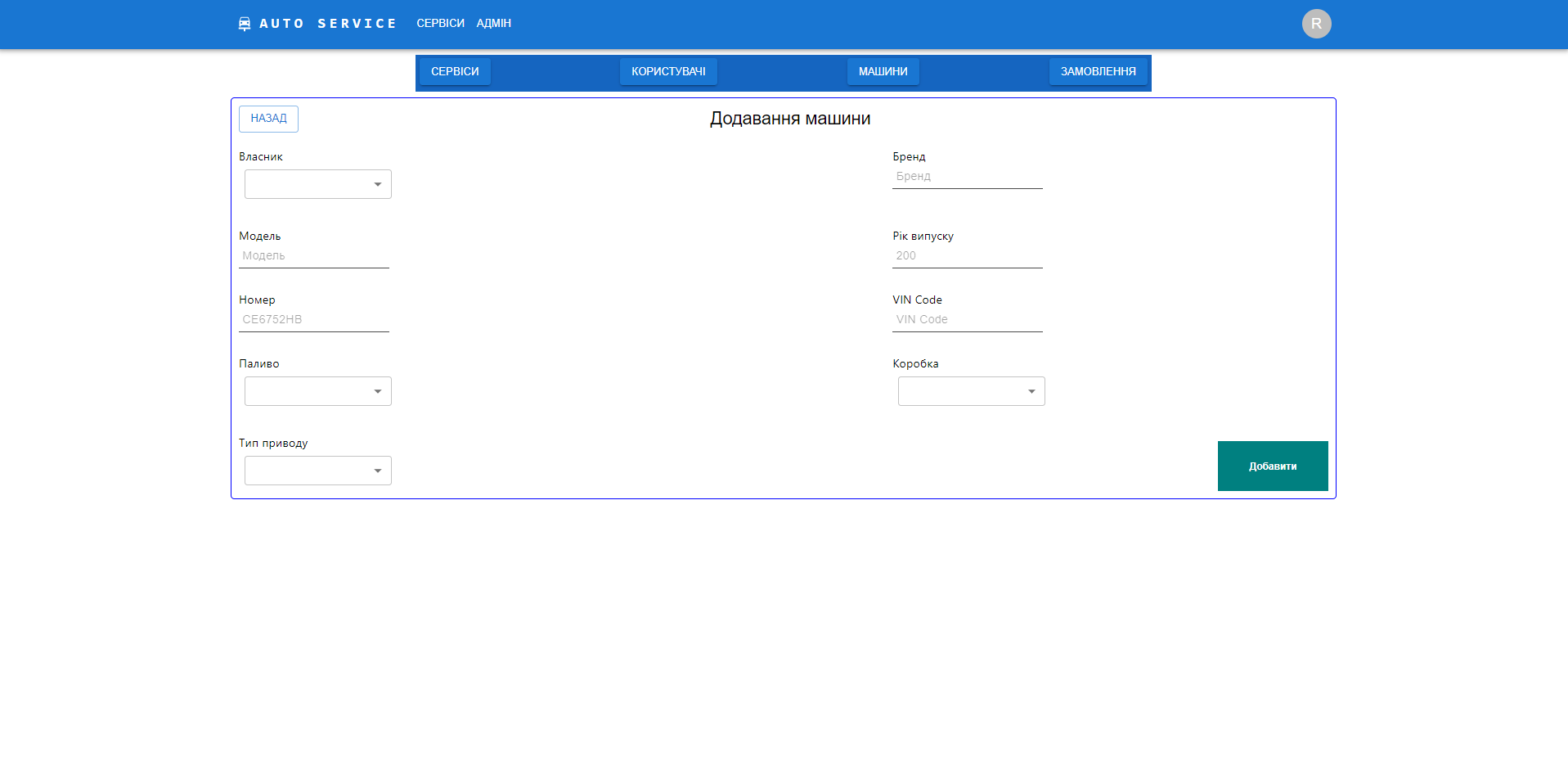


Рисунок 2.18 –Додавання машин

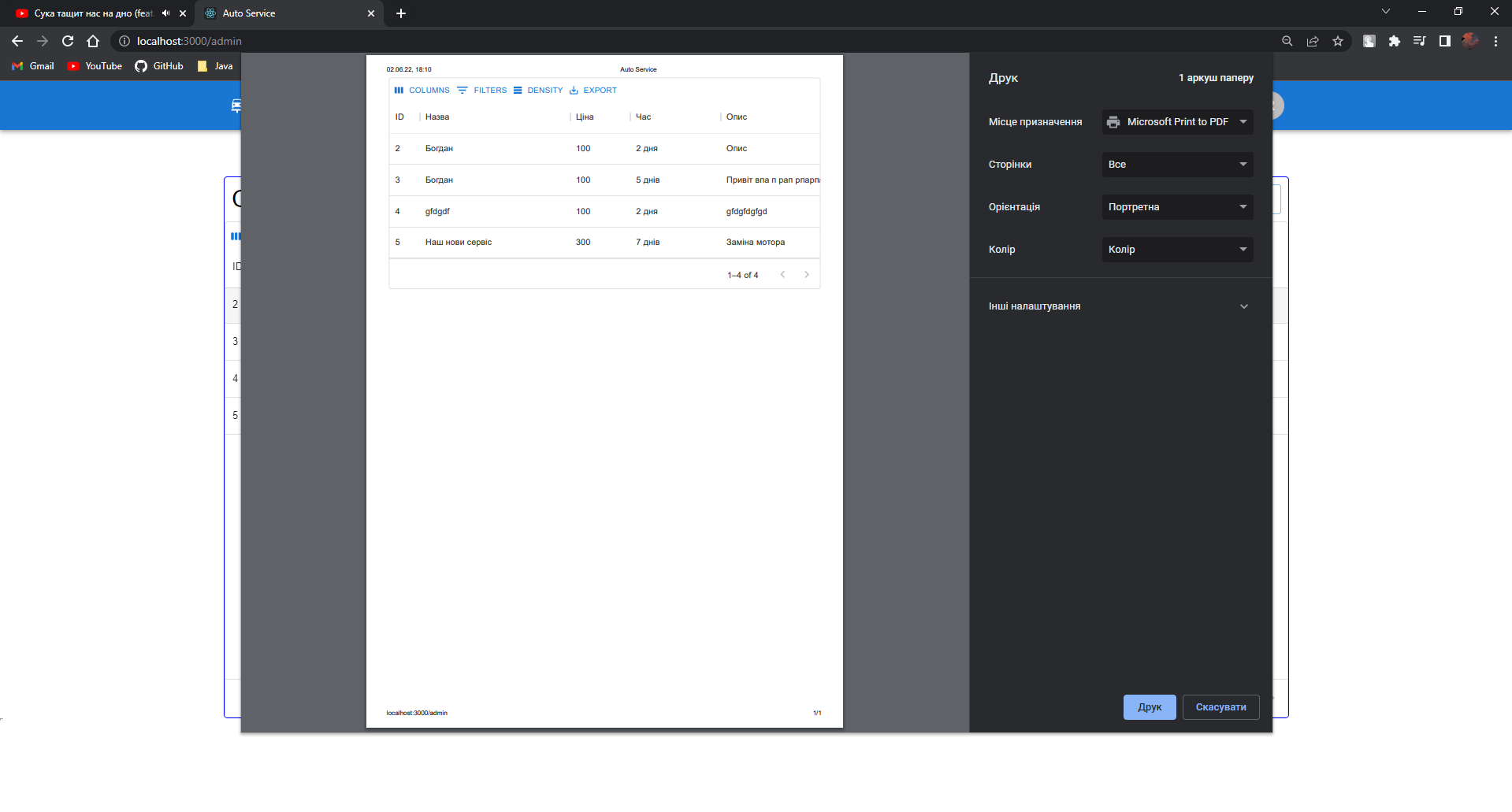


Рисунок 2.19 – Друк звітності

# РОЗДІЛ 3. РОБОЧИЙ ПРОєКТ

## Засоби розробки

Для розробки програмного додатку були обрані редактори коду IntelliJ IDEA Ultimate, WebStorm, Visual Studio Code та мови програмування Java, JavaScript, React.js, HTML, CSS.

IntelliJ IDEA – комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP) від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії «Community Edition» і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Сирцеві тексти Community-версії поширюються рамках ліцензії Apache 2.0. Бінарні збірки підготовлені для Linux, Mac OS X і Windows.

JetBrains WebStorm – інтегроване середовище розробки для JavaScript, HTML та CSS від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA. WebStorm є спеціалізованою версією PhpStorm, пропонуючи підмножину з його можливостей.

Visual Studio Code – засіб для створення, редагування та провадження сучасних вебзастосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X. Компанія Microsoft представила Visual Studio Code у квітні 2015 на конференції Build 2015.

Java – об’єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». Нині її використовують приблизно 9 мільйонів осіб, а самі Java-програми застосовуються на семи мільярдах пристроїв. Найчастіше Java використовується в таких сферах як:

* мобільні застосунки;
* ігри;
* наукові розробки;
* мобільні телефони;
* ігрові консолі;
* суперкомп’ютери.

Java – це основа практично для всіх типів мережевих застосунків. У багатьох дослідженнях її називають мовою програмування номер один серед розробників.

Особливості Java:

1. Кросплатформеність. Java має спеціальну програму, яка виконує код. Це віртуальна машина Java, або Java Virtual Machine. Якщо програмний код написано один раз, то він працює з будь-якою апаратною платформою або операційною системою: від смарткарт до застосунків для розумних будинків.
2. Ком’юніті. Оскільки Java – досить поширена мова, якою користується велика кількість розробників, можна знайти вирішення практично будь-якої проблеми, якщо вона виникне при роботі з Java. У пригоді стануть тисячі бібліотек та форумів. На GitHub, наприклад, є відкриті проекти і документація, а на форумі Stack Overflow можна звернутися за допомогою до ком’юніті.
3. Надійність. Мова Java строго типізована. Тобто будь-яка змінна або вираз має певний тип вже на момент компіляції, що спрощує виявлення якихось проблем. Компілятор сам підказує програмістові, де той припускається помилки.
4. Об’єктно-орієнтованість. Усі бібліотеки, написані колись для Java, – це класи, які відповідають за функціональність мови. Будь-який застосунок на Java – це набір класів, що описують різні об’єкти. Це добре, тому що дозволяє створювати складні, але прості у підтримці програми. І в цілому Java – мультипарадигменна мова, тобто підтримує безліч принципів програмування, що дозволяє ефективно вирішувати різні завдання.
5. Відноснапростота. З Java не так просто подружитися, як з Python, але вона набагато простіша за мову С або С ++. Функціональність мови оновлюється повільно, тому можна легко переходити на нові версії — заново вивчати не доведеться. Java – строго типізована мова, а значить новачок завжди матиме можливість побачити помилку в коді при компіляції.
6. Гнучкість. На Java можна розробити програму будь-якої складності:

* інтернет-магазин;
* банківські застосунки;
* високонавантажені системи;
* штучний інтелект.

JavaScript (JS) – об’єктно-прототипна мова програмування, що здебільшого використовується у веб-розробці. На відміну від більшості серверних мов програмування, JS відноситься до клієнтської сторони, тобто, обробляється на стороні клієнта, а саме інтерпретатором браузера користувача. JS використовують в основному для роботи на frontend-стороні. JS використовується для управління вікном браузера, зміни змісту документа при доступі до DOM’у, обробки різного роду event’ів на сторінці.

React – відкрита JavaScript-бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

HTML – це мова тегів, засобами якої здійснюється розмічання веб-сторінок для мережі Інтернет. Браузери отримують HTML-документи з вебсервера або з локальної пам'яті й передають документи в мультимедійні веб-сторінки. HTML описує структуру веб-сторінки семантично і початково підказки для відображення документа.

CSS – це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду. Самі ж сторінки написані мовами розмітки даних. CSS є основною технологією інтернет, поряд із HTML та JavaScript.

У даному пункті було описано засоби розробки та мови програмування.

## 3.2. Документи на супроводження ПЗ

3.2.1 Інструкції програмісту

Оскільки програмне забезпечення було створено з використанням патерну MVC, то основними частинами фунціоналу є контролери, моделі даних, сервіси та репозиторії.

На рисунку 3.1 зображена загальна структура backend-частини проекту.

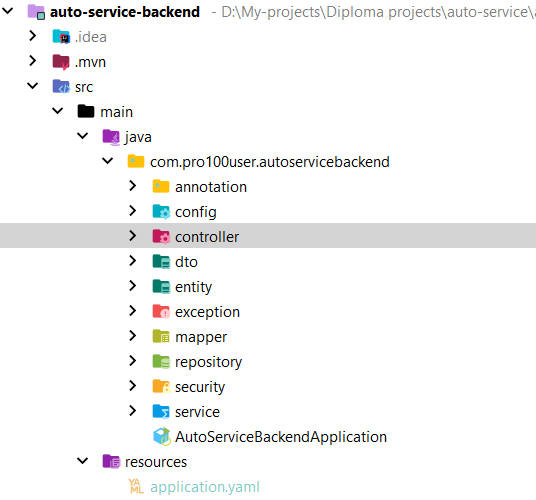


Рисунок 3.1 – Загальна структура backend-частини

Розглянемо призначення цих папок.

Папка annotation містить кастомні анотації. Файл CurrentUser – це анотація, яка витягує дані авторизованого користувача з SecurityContext.

Папка config містить файли конфігурації. MvcConfig – це клас, який конфігурує MVC.

Папка controller містить контролери, які відстежують звернення до backend-частини. До неї належать такі файли:

* AdminController – це клас, який опрацьовує запити від адміністратора;
* AuthorizationController – це клас, який опрацьовує запити на авторизацію та реєстрацію;
* CarController – це клас, який опрацьовує запити на дані машин;
* OrderController – це клас, який опрацьовує запити на дані замовлень;
* ServiceController – це клас, який опрацьовує запити на дані сервісів;
* UserController – це клас, який опрацьовує запити на дані користувача.

Папка dto (Data Transfer Object) містить об’єкти, які вертаються у frontend-частину.

Папка entity містить класи, які відповідають структурі БД. Вона містить такі файли:

* DriveType – перелік типів приводу;
* Fuel – перелік палив;
* Role – перелік ролей;
* Status – перелік статусів замовлення;
* Transmission – перелік типів коробки передач;
* AutoService – це клас, в якому описується дані сервісу;
* Car – це клас, в якому описуються дані машини;
* Order – це клас, в якому описується дані повідомлень;
* User – це клас, в якому описується дані користувача.

Папка exception містить кастомні виключення та обробник виключень. Вона містить такі файли:

* ApplicationExceptionHandler – це клас, який опрацьовує виключення та повертає відповідь про помилку у frontend-частину;

Папка mapper – папка, в якій знаходяться мапери, які перетворюють з dto в entity і навпаки. Вона містить такі файли:

* CarMapper – це мапер, який перетворює car dto в car та навпаки;
* OrderMapper – це мапер, який перетворює order dto в order та навпаки;
* ServiceMapper – це мапер, який перетворює service dto в service та навпаки;
* UserMapper – це мапер, який перетворює user dto в user та навпаки.

Папка repository містить файли рівня репозиторію, які звертаються до бази даних. Вона містить такі файли:

* CarRepository – це інтерфейс, який унаслідується від JpaRepository та працює з даними про машини в БД;
* OrderRepository – це інтерфейс, який унаслідується від JpaRepository та працює з даними про замовлення в БД;
* ServiceRepository – це інтерфейс, який унаслідується від JpaRepository та працює з даними про сервіси в БД;
* UserRepository – це інтерфейс, який унаслідується від JpaRepository та працює з даними про користувачів в БД.

Папка security містить файли конфігурації Spring Security. Вона містить такі файли:

* JwtFilter – це клас, який унаслідується від OncePerRequestFilter та перевіряє JSON Web Token в вхідних запитах;
* JwtProvider – це клас, який генерує JSON Web Token та перевіряє його на валідність;
* SecurityConfig – це клас, який конфігурує Spring Security;
* SecurityConfigure – це клас, який конфігурує Spring Security;
* UserSecurity – це клас, який унаслідується від UserDetails та описує дані користувача, які будуть зберігатися в JSON Web Token та SecurityContext;
* UserSecurityService – це клас, який унаслідується від UserDetailsService, та обробляє пошук користувача по логіну.

Папка service містить файли бізнес-логіки, які зв’язують контролери з рівнем репозиторія. Вона містить такі файли:

* AutoServiceServiceImpl – це клас, який реалізує AutoServiceService;
* CarServiceImpl – це клас, який реалізує CarService;
* ImageServiceImpl – це клас, який реалізує ImageService;
* OrderServiceImpl – це клас, який реалізує OrderService;
* UserServiceImpl – це клас, який реалізує UserService;
* AutoServiceService – це інтерфейс, який описує основні методи пов’язані з AutoService;
* CarService – це інтерфейс, який описує основні методи, пов’язані з Car;
* ImageService – це інтерфейс, який описує основні методи, пов’язані з збереженням, редагуванням, видаленням фотографій;
* OrderService – це інтерфейс, який описує основні методи, пов’язані з Order;
* UserService – це інтерфейс, який описує основні методи, пов’язані з User;

Лістинг програмного коду наведено в додатку А.

3.2.2 Інструкція користувачу

Для початку користувачу потрібно авторизуватись на сайті (рисунок 3.2).

…

Отже, розроблене програмне забезпечення володіє інтуїтивно-зрозумілим інтерфейсом.

## 3.3 Тестування програмного продукту

Тестування програмного забезпечення – це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність.

Об’єктом випробувань є розроблений програмний продукт.

Метою випробувань є виявлення програмних помилок та неточностей в роботі при обробці вхідних даних.

Тестування програмного продукту відбувалось на таких контрольних прикладах як авторизація, реєстрація, додавання, редагування автомобіля, запис на сервіс, пошук замолень.

При вході на сайт першим відкривається вікно авторизації, в якому потрібно увійти в акаунт (рисунок 3.11). Якщо акаунта нема, тоді можна зареєструвати новий акаунт у вікні реєстрації (рисунок 3.14).

…

Проведені тестові завдання демонструють, що програмне забезпечення працює коректно.

## 3.4 Охорона праці

Виконання вимог інструкції є обов’язковим для осіб, які працюють з програмним забезпеченням. Інструкції можуть бути видані на руки, розміщені на робочих місцях, чи зберігатися у визначеному місці, доступному для працюючих.

Напруга живлення 220 В є небезпечною для життя людини. Тому, незважаючи на те, що в конструкції комп’ютера передбачена достатня ізоляція від струмопровідних ділянок, необхідно знати та чітко виконувати ряд правил техніки безпеки.

Забороняється:

* торкатися екрана і тильного боку дисплея, проводів живлення та заземлення, з’єднувальних кабелів;
* порушувати порядок увімкнення й вимикання апаратних блоків;
* класти на апаратуру сторонні предмети;
* працювати за комп’ютером у вологому одязі та вологими руками;
* палити в приміщенні, де знаходяться комп’ютери.

Під час роботи за комп’ютером необхідно:

* суворо дотримуватися інструкції з експлуатації апаратури;
* працювати на клавіатурі чистими сухими руками, не натискати на клавіші без потреби чи навмання;
* коректно завершувати роботу з тим чи іншим програмним засобом.

У разі появи запаху горілого, самовільного вимикання апаратури, незвичних звуків треба негайно вимкнути комп’ютер від’єднати від системи електропостачання та повідомити про можливе виникнення небезпечної ситуації обслуговуючий персонал. Не можна працювати за комп’ютером при недостатньому освітленні, високому рівні шуму тощо.

Організація робочого місця оператора повинна забезпечувати відповідність усіх елементів робочого місця та їх розташування вимогам НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації обчислювальних машин».

Розміщення принтера або іншого пристрою введення-виведення інформації на робочому місці має забезпечувати добру видимість екрана, зручність ручного керування пристроєм введення-виведення інформації у зоні досяжності моторного поля.

Конструкція робочого місця має відповідати сучасним вимогам ергономіки і забезпечувати оптимальне розміщення на робочій поверхні використовуваного обладнання (дисплея, клавіатури, принтера) і документів.

Екран має розташовуватися на оптимальній відстані від очей користувача, що становить 600..700 мм, але не ближче ніж за 600 мм з урахуванням розміру літерно-цифрових знаків і символів.

Розташування екрана має забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30° до нормальної лінії погляду працюючого.

Клавіатуру слід розташовувати на поверхні столу на відстані 100..300 мм від краю, зверненого до працюючого. У конструкції клавіатури має передбачатися опорний пристрій (виготовлений з матеріалу з високим коефіцієнтом тертя, що перешкоджає мимовільному її зсуву), який дає змогу змінювати кут нахилу поверхні клавіатури у межах 5..15°. Висота середнього рядка клавіш не має перевищувати 30 мм. Поверхня клавіатури має бути матовою з коефіцієнтом відбиття 0,4.

Розташування пристроїв введення-виведення інформації має забезпечувати їх добру видимість та зручність ручного керування.

# ВИСНОВКИ

Під час роботи над дипломним проєктом було створено веб-додаток “Автосервіс”.

Метою дипломного проєкту була розробити веб-додаток, який допоможе користувачам з ремонтом авто .

В результаті виконання програмного етапу були створення гнучкий програмник код, який буде далі продовжуватись покращуватись, та додаватись нові частини, чим буде відчутно збільшено можливості.

реалізовані зараз, а саме:

* авторизація;
* реєстрація;
* перегляд, додавання, редагування, видалення сервісів;
* перегляд, додавання, редагування, видалення машин;
* запис на замовлення;
* відстежування статусу замовлення;
* перегляд та зміна замовлення.

Програмний продукт оснащений достатньою кількістю функціоналу, щоб у повній мірі задовільнити поставлені цілі дипломної роботи.

За період реалізації дипломного проекту були вдосконалені навички в роботі з мережевими базами даних, HTTPS-запитами, особливостями «спілкування» різноманітних сервісів між собою, роботою з кодом інших розробників, покращились вміння роботи в команді. При виконанні дипломної роботи отримано досвід роботи з Spring REST Application, Rect.js, JSON та вдосконалено навички програмування мовою Java.

Розроблена система може буде вдосконалена шляхом реалізації додаткового функціоналу та добавлення багатопотокових запитів для меншого навантаження додатку.

# перелік джерел

1. Система питань і відповідей для програмістів – [Електронний ресурс]. – Режим доступу: <https://stackoverflow.com/>
2. Эккель Брюс “Філософія Java” – [Електронний ресурс]. – Режим доступу: <https://www.rulit.me/books/filosofiya-java-read-180907-1.html>
3. Інтернет ресурс для вивчення різних технологій веб-програмування – [Електронний ресурс]. – Режим доступу: <https://metanit.com/web/>
4. Інтернет ресурс для вивчення мови програмування JavaScript – [Електронний ресурс]. – Режим доступу: <https://javascript.info/>
5. Відеохостинг «Youtube» – [Електронний ресурс]. – Режим доступу: <https://www.youtube.com/>

# Додатки

**Додаток А**

**Лістинг програмного коду**

А.1 – Клас User

@Getter

@Setter

@Entity

@NoArgsConstructor

@Table(name = "users")

@Builder(toBuilder = true)

@AllArgsConstructor(access = AccessLevel.PRIVATE)

public class User implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id", nullable = false, updatable = false, unique = true)

private Long id;

@Column(name = "name", nullable = false)

private String name;

@Column(name = "surname", nullable = false)

private String surname;

@Column(name = "email", nullable = false, unique = true)

private String email;

@Column(name = "phone", nullable = false, unique = true)

private String phone;

@Column(name = "password", nullable = false)

private String password;

@ElementCollection(fetch = FetchType.EAGER, targetClass = Role.class)

@CollectionTable(

name = "user\_roles",

joinColumns = @JoinColumn(name = "user\_id", referencedColumnName = "id")

)

@Enumerated(EnumType.STRING)

@Column(name = "role", nullable = false)

private List<Role> roles = new ArrayList<>();

@Column(name = "enabled", nullable = false)

private boolean enabled;

@OneToMany(mappedBy = "user", fetch = FetchType.LAZY, targetEntity = Car.class)

private List<Car> cars = new ArrayList<>();

}

A.2 – Клас Car

@Getter

@Setter

@Entity

@NoArgsConstructor

@Table(name = "cars")

@Builder(toBuilder = true)

@AllArgsConstructor(access = AccessLevel.PRIVATE)

public class Car implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id", nullable = false, updatable = false, unique = true)

private Long id;

@ManyToOne(fetch = FetchType.EAGER, targetEntity = User.class)

@JoinColumn(name = "user\_id", nullable = false)

private User user;

@Column(name = "brand", nullable = false)

private String brand;

@Column(name = "model", nullable = false)

private String model;

@Column(name = "year", nullable = false)

private int year;

@Column(name = "number", nullable = false)

private String number;

@Column(name = "vin\_code", nullable = false)

private String vinCode;

@Enumerated(EnumType.STRING)

@Column(name = "fuel", nullable = false)

private Fuel fuel;

@Enumerated(EnumType.STRING)

@Column(name = "transmission", nullable = false)

private Transmission transmission;

@Enumerated(EnumType.STRING)

@Column(name = "drive\_type", nullable = false)

private DriveType driveType;

@OneToMany(mappedBy = "car", fetch = FetchType.LAZY, targetEntity = Order.class)

private List<Order> orders = new ArrayList<>();

}

A.3 – Клас AutoService

@Getter

@Setter

@Entity

@NoArgsConstructor

@Table(name = "services")

@Builder(toBuilder = true)

@AllArgsConstructor(access = AccessLevel.PRIVATE)

public class AutoService implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id", nullable = false, updatable = false, unique = true)

private Long id;

@Column(name = "name", nullable = false)

private String name;

@Column(name = "image")

private String image;

@Column(name = "price", nullable = false)

private double price;

@Column(name = "time", nullable = false)

private String time;

@Column(name = "description", nullable = false)

private String description;

@OneToMany(mappedBy = "service", fetch = FetchType.LAZY, targetEntity = Order.class)

private List<Order> orders = new ArrayList<>();

}

A.4 – Клас Order

@Getter

@Setter

@Entity

@NoArgsConstructor

@Table(name = "orders")

@Builder(toBuilder = true)

@AllArgsConstructor(access = AccessLevel.PRIVATE)

public class Order implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id", nullable = false, updatable = false, unique = true)

private Long id;

@ManyToOne(fetch = FetchType.EAGER, targetEntity = Car.class)

@JoinColumn(name = "car\_id", nullable = false)

private Car car;

@ManyToOne(fetch = FetchType.EAGER, targetEntity = AutoService.class)

@JoinColumn(name = "service\_id", nullable = false)

private AutoService service;

@Enumerated(EnumType.STRING)

@Column(name = "status", nullable = false)

private Status status;

@CreationTimestamp

@JsonSerialize(using = LocalDateTimeSerializer.class)

@JsonDeserialize(using = LocalDateTimeDeserializer.class)

@Column(name = "created\_at", nullable = false, updatable = false)

private LocalDateTime createdAt;

}

A.5 – Enum Fuel

public enum Fuel {

Бензин,

Дизель,

Газ,

ГазБензин,

Гібрид,

Електро,

Інше;

}

A.6 – Enum Transmission

public enum Transmission {

Мехініка,

Автомат,

Типтронік,

Робот,

Варіатор;

}

A.7 – Enum DriveType

public enum DriveType {

Повний,

Передній,

Задній;

}

A.8 – Enum Role

public enum Role {

ROLE\_USER,

ROLE\_ADMIN;

}

A.9 – Enum Status

public enum Status {

Оформлено,

Доставлено,

Ремонтується,

Завершено,

Архів,

Відмовлено

}

A.10 – Клас UserRepository

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

@Query(value = "SELECT \* FROM users WHERE email=:email", nativeQuery = true)

Optional<User> findByEmail(@Param("email") String email);

@Query(value = "SELECT \* FROM users WHERE phone=:phone", nativeQuery = true)

Optional<User> findByPhone(@Param("phone") String phone);

}

A.11 – Клас CarRepository

@Repository

public interface CarRepository extends JpaRepository<Car, Long> {

}

A.12 – Клас ServiceRepository

@Repository

public interface ServiceRepository extends JpaRepository<AutoService, Long> {

}

A.13 – Клас OrderRepository

@Repository

public interface OrderRepository extends JpaRepository<Order, Long> {

}

A.14 – Інтерфейс UserService

public interface UserService {

UserDTO create(UserCreateDTO dto);

UserDTO getById(Long userId);

UserDTO update(UserUpdateDTO dto);

boolean delete(Long userId);

List<UserDTO> getAll();

User findByEmail(String email);

User findByPhone(String phone);

List<CarDTO> getCars(Long userId);

}

A.15 – Клас CarRepository

public interface CarService {

CarDTO create(CarCreateDTO dto);

CarDTO getById(Long carId);

CarDTO update(CarUpdateDTO dto);

boolean delete(Long carId);

List<CarDTO> getAll();

List<CarListDTO> getAllCarListDTO();

}

A.16 – Клас ImageService

public interface ImageService {

String save(MultipartFile file, Long serviceId);

String update(String filepath, MultipartFile file, Long serviceId);

boolean delete(String filepath);

}

A.17 – Клас AutoServiceService

public interface AutoServiceService {

ServiceDTO create(ServiceCreateDTO dto);

ServiceDTO getById(Long serviceId);

ServiceDTO update(ServiceUpdateDTO dto);

boolean delete(Long serviceId);

List<ServiceDTO> getAll();

ServiceDTO setImage(MultipartFile file, Long serviceId);

boolean updateImage(MultipartFile file, Long serviceId);

boolean deleteImage(Long serviceId);

}

A.18 – Клас OrderService

public interface OrderService {

OrderDTO create(OrderCreateDTO dto);

OrderDTO getById(Long orderId);

OrderDTO update(OrderUpdateDTO dto);

boolean delete(Long orderId);

List<OrderDTO> getAll();

}

A.19 – Клас UserServiceImpl

@Slf4j

@Service

@Transactional

@RequiredArgsConstructor(onConstructor = @\_\_(@Autowired))

public class UserServiceImpl implements UserService {

private final UserRepository userRepository;

private final UserMapper userMapper;

private final CarService carService;

private final CarMapper carMapper;

private final PasswordEncoder passwordEncoder;

@Override

public UserDTO create(UserCreateDTO dto) {

if(findByEmail(dto.getEmail()) != null) {

throw new BadCredentialsException("email");

}

if(findByPhone(dto.getPhone()) != null) {

throw new BadCredentialsException("phone");

}

User entity = userMapper.toEntity(dto);

entity.setRoles(List.of(Role.ROLE\_USER));

entity.setPassword(passwordEncoder.encode(entity.getPassword()));

entity.setEnabled(true);

return userMapper.toUserDTO(

userRepository.save(entity)

);

}

@Override

@Transactional(readOnly = true)

public UserDTO getById(Long userId) {

return userMapper.toUserDTO(

userRepository.findById(userId).orElseThrow()

);

}

@Override

public UserDTO update(UserUpdateDTO dto) {

User userDTO = userMapper.toEntity(dto);

User entity = userRepository.findById(userDTO.getId()).orElseThrow();

if(!entity.getEmail().equals(userDTO.getEmail()) && findByEmail(dto.getEmail()) != null) {

throw new BadCredentialsException("email");

}

if(!entity.getPhone().equals(userDTO.getPhone()) && findByPhone(dto.getPhone()) != null) {

throw new BadCredentialsException("phone");

}

entity = entity.toBuilder()

.surname(userDTO.getSurname() == null ? entity.getSurname() : userDTO.getSurname())

.name(userDTO.getName() == null ? entity.getName() : userDTO.getName())

.email(userDTO.getEmail() == null ? entity.getEmail() : userDTO.getEmail())

.phone(userDTO.getPhone() == null ? entity.getPhone() : userDTO.getPhone())

.build();

return userMapper.toUserDTO(

userRepository.save(entity)

);

}

@Override

public boolean delete(Long userId) {

User entity = userRepository.findById(userId).orElseThrow();

entity.setEnabled(false);

userRepository.save(entity);

return true;

}

@Override

@Transactional(readOnly = true)

public List<UserDTO> getAll() {

return userMapper.toUserDTO(

userRepository.findAll()

);

}

@Override

@Transactional(readOnly = true)

public User findByEmail(String email) {

return userRepository.findByEmail(email).orElse(null);

}

@Override

@Transactional(readOnly = true)

public User findByPhone(String phone) {

return userRepository.findByPhone(phone).orElse(null);

}

@Override

public List<CarDTO> getCars(Long userId) {

return carMapper.toCarDTO(

userRepository.findById(userId).orElseThrow().getCars()

);

}

}

A.20 – Клас CarServiceImpl

@Slf4j

@Service

@Transactional

@RequiredArgsConstructor(onConstructor = @\_\_(@Autowired))

public class CarServiceImpl implements CarService {

private final CarRepository carRepository;

private final CarMapper carMapper;

@Override

public CarDTO create(CarCreateDTO dto) {

Car entity = carMapper.toEntity(dto);

return carMapper.toCarDTO(

carRepository.save(entity)

);

}

@Override

@Transactional(readOnly = true)

public CarDTO getById(Long carId) {

return carMapper.toCarDTO(

carRepository.findById(carId).orElseThrow()

);

}

@Override

public CarDTO update(CarUpdateDTO dto) {

Car entity = carMapper.toEntity(dto);

return carMapper.toCarDTO(

carRepository.save(entity)

);

}

@Override

public boolean delete(Long carId) {

carRepository.deleteById(carId);

return true;

}

@Override

@Transactional(readOnly = true)

public List<CarDTO> getAll() {

return carMapper.toCarDTO(

carRepository.findAll()

);

}

@Override

@Transactional(readOnly = true)

public List<CarListDTO> getAllCarListDTO() {

return carMapper.toCarListDTO(

carRepository.findAll()

);

}

}

A.21 – Клас ImageServiceImpl

@Slf4j

@Service

public class ImageServiceImpl implements ImageService {

private final String filepath;

public ImageServiceImpl() {

try {

filepath = ResourceUtils.getFile("classpath:").getPath() + "/static/files/";

File directory = new File(filepath);

if (!directory.exists()) {

Files.createDirectories(directory.toPath());

}

} catch (FileNotFoundException e) {

throw new RuntimeException(e);

} catch (IOException e) {

throw new RuntimeException(e);

}

}

@Override

public String save(MultipartFile file, Long serviceId) {

try {

File avatar = new File(filepath + serviceId.toString() + file.getOriginalFilename());

if (!avatar.exists()) {

avatar.createNewFile();

}

BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(avatar));

bos.write(file.getBytes());

bos.flush();

bos.close();

return serviceId.toString() + file.getOriginalFilename();

} catch (IOException e) {

return null;

}

}

@Override

public String update(String filename, MultipartFile file, Long serviceId) {

try {

File avatar;

if(filename == null)

avatar = new File(filepath + serviceId.toString() + file.getOriginalFilename());

else

avatar = new File(filepath + filename);

if (!avatar.exists()) {

avatar.createNewFile();

}

BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(avatar));

bos.write(file.getBytes());

bos.flush();

bos.close();

return avatar.getName();

} catch (IOException e) {

return null;

}

}

@Override

public boolean delete(String filename) {

File avatar = new File(filepath + filename);

if (avatar.exists()) {

return avatar.delete();

}

return false;

}

}

A.22 – Клас AutoServiceServiceImpl

@Slf4j

@Service

@Transactional

@RequiredArgsConstructor(onConstructor = @\_\_(@Autowired))

public class AutoServiceServiceImpl implements AutoServiceService {

private final ServiceRepository serviceRepository;

private final ServiceMapper serviceMapper;

private final ImageService imageService;

@Override

public ServiceDTO create(ServiceCreateDTO dto) {

AutoService entity = serviceMapper.toEntity(dto);

return serviceMapper.toServiceDTO(

serviceRepository.save(entity)

);

}

@Override

@Transactional(readOnly = true)

public ServiceDTO getById(Long serviceId) {

return serviceMapper.toServiceDTO(

serviceRepository.findById(serviceId).orElseThrow()

);

}

@Override

public ServiceDTO update(ServiceUpdateDTO dto) {

AutoService entity = serviceMapper.toEntity(dto);

return serviceMapper.toServiceDTO(

serviceRepository.save(entity)

);

}

@Override

public boolean delete(Long serviceId) {

serviceRepository.deleteById(serviceId);

return true;

}

@Override

@Transactional(readOnly = true)

public List<ServiceDTO> getAll() {

return serviceMapper.toServiceDTO(

serviceRepository.findAll()

);

}

@Override

public ServiceDTO setImage(MultipartFile file, Long serviceId) {

AutoService entity = serviceRepository.findById(serviceId).orElseThrow();

entity.setImage(imageService.save(file, serviceId));

return serviceMapper.toServiceDTO(

serviceRepository.save(entity)

);

}

@Override

public boolean updateImage(MultipartFile file, Long serviceId) {

AutoService entity = serviceRepository.findById(serviceId).orElseThrow();

entity.setImage(imageService.update(entity.getImage(), file, serviceId));

serviceRepository.save(entity);

return true;

}

@Override

public boolean deleteImage(Long serviceId) {

AutoService service = serviceRepository.findById(serviceId).orElseThrow();

return imageService.delete(service.getImage());

}

}

A.23 – Клас OrderServiceImpl

@Slf4j

@Service

@Transactional

@RequiredArgsConstructor(onConstructor = @\_\_(@Autowired))

public class OrderServiceImpl implements OrderService {

private final OrderRepository orderRepository;

private final OrderMapper orderMapper;

@Override

public OrderDTO create(OrderCreateDTO dto) {

Order entity = orderMapper.toEntity(dto);

return orderMapper.toOrderDTO(

orderRepository.save(entity)

);

}

@Override

@Transactional(readOnly = true)

public OrderDTO getById(Long orderId) {

return orderMapper.toOrderDTO(

orderRepository.findById(orderId).orElseThrow()

);

}

@Override

public OrderDTO update(OrderUpdateDTO dto) {

Order entity = orderMapper.toEntity(dto);

return orderMapper.toOrderDTO(

orderRepository.save(entity)

);

}

@Override

public boolean delete(Long orderId) {

orderRepository.deleteById(orderId);

return true;

}

@Override

@Transactional(readOnly = true)

public List<OrderDTO> getAll() {

return orderMapper.toOrderDTO(

orderRepository.findAll()

);

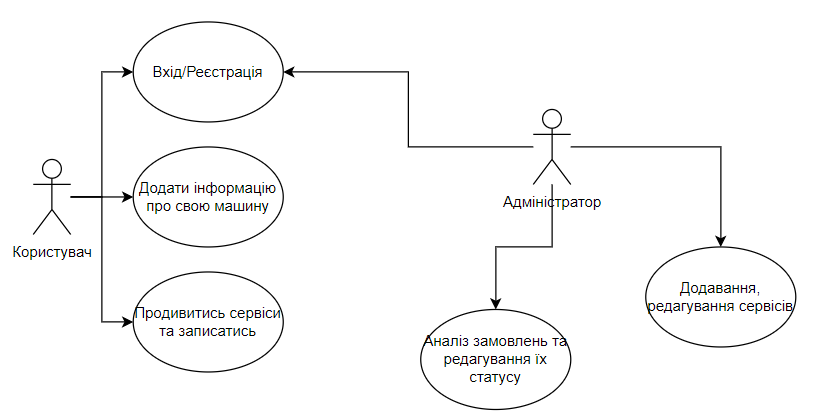
}

}

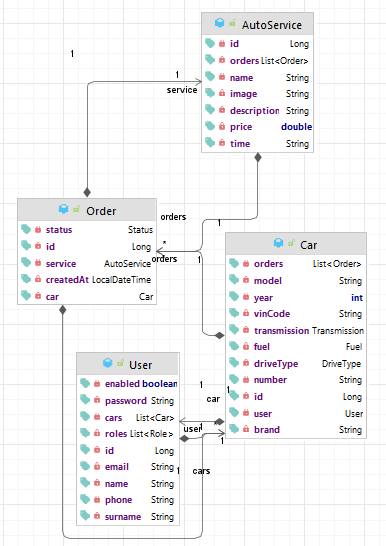
**Додаток Б**

**Рисунки**

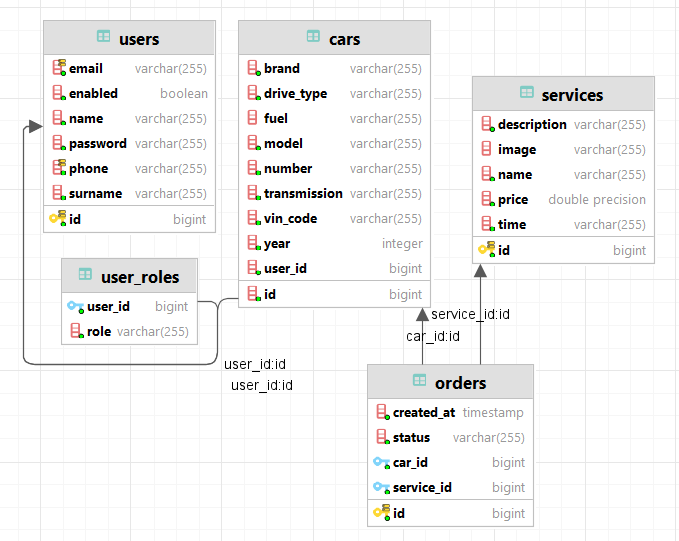
Б.1 – Модель прецедентів



Б.2 – Взаємодія класів



Б.3 – Зв’язки між таблицями



**Декларація академічної чесності**

Я, Васенков Дмитро Олександрович, підтверджую, що мій дипломний проєкт на тему «Веб-додаток «Автосервіс»» написаний моїми власними словами, що відбивають особисті думки, судження та висновки. Я розумію та пам’ятаю, що під час проведення дослідження та оформлення його результатів необхідно уникати плагіату – представлення думок та слів іншої людини як своїх, з особливою увагою та обережністю працювати із інформацією, яка надана у вільному доступі. У разі використання й / або цитування продуктів інтелектуальної праці інших осіб мною було обов’язково подано посилання на джерело інформації.

Я усвідомлюю, що у разі виявлення порушення цих правил розроблений мною дипломний проєкт буде миттєво відхилений від права повторного його захисту із одночасним виставленням оцінки «незадовільно».

*19 червня* 2022 р.

**БІБЛІОГРАФІЧНА ДОВІДКА**

Тема дипломного прокту: Веб-додаток «Автосервіс»

Обсяг пояснювальної записки: 54 аркушів

Дата закінчення проєкту *19 червня* 2022 р.

Підпис студента-дипломника \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_