МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ «АВТОСАЛОН» ІЗ ЗАСТОСУВАННЯМ СУБД MONGO»**

Виконав: студент 4 курсу, групи 6.1260
спеціальності 126 інформаційні системи та технології
(шифр і назва спеціальності)
освітньої програми інформаційні системи та штучний інтелект
(назва освітньої програми)
М.Т. Шепеляковський
(ініціали та прізвище)

Керівник завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 126 інформаційні системи та технології
(шифр і назва)

Освітня програма інформаційні системи та штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Шепеляковському Максиму Тарасовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інформаційної системи «Автосалон» із застосуванням СУБД mongo

керівник роботи Гребенюк Сергій Миколайович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Розробка проєкту з використанням зазначених засобів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі. 2. Аналіз предметної області. 3. Збір та аналіз вимог до ІС.
4. Вибір технологій та інструментів для реалізації. 5. Розробка моделей даних.
6. Реалізація інформаційної системи на основі вибраних технологій.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	19.01.2024	
3.	Обробка методичних та теоретичних	19.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	25.03.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	22.06.2024	

Студент _____ М.Т. Шепеляковський
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.М. Гребенюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка інформаційної системи «Автосалон» із застосуванням СУБД mongo»: 75 с., 114 рис., 7 джерел, 1 додаток.

АВТОСАЛОН, ІНФОРМАЦІЙНА СИСТЕМА, ANGULAR, HTML, JAVA, MONGODB, SCSS, SPRING BOOT, TYPESCRIPT.

Об’єкт дослідження – процес розробки інформаційної системи для автосалону за допомогою Angular, Spring Boot, MongoDB

Мета роботи – створення функціональної інформаційної системи для автосалону, що забезпечує зручний інтерфейс для замовлення автомобілів, динамічну фільтрацію та пошук.

Метод дослідження – методи аналізу та проєктування інформаційних систем, методи розробки вебдодатків з використанням сучасних фреймворків та бібліотек.

Для реалізації мети були поставлені наступні задачі:

– аналіз предметної області та визначення основних вимог до системи;
– проєктування бази даних;
– розробка front-end частини з використанням Angular;
– реалізація back-end частини з використанням Spring Boot;
– розробка функціоналу динамічної фільтрації та пошуку автомобілів.

Для створення програми було обрано середовище розробки IntelliJ IDEA Ultimate, в якому використовуються такі технології, як Angular для front-end розробки, Spring Boot для back-end розробки, MongoDB як база даних, та мови програмування Java, HTML, SCSS, TypeScript.

Інформаційна система «Автосалон» призначена для роботи на пристроях, на яких встановлено один з браузерів: Chrome, Opera, Mozilla, FireFox.

SUMMARY

Bachelor’s qualifying paper “Development of Information System "Autosalon" Using Mongo DBMS”: 75 pages, 114 figures, 7 references, 1 supplement.

ANGULAR, CAR DEALERSHIP, HTML, INFORMATION SYSTEM, JAVA, MONGODB, SCSS, SPRING BOOT, TYPESCRIPT.

The object of the study is the process of developing an information system for a car dealership using Angular, Spring Boot, and MongoDB.

The aim of the study is to create a functional information system for a car dealership that provides a convenient interface for ordering cars, dynamic filtering, and search capabilities.

The methods of research are methods of analysis and design of information systems, methods of web application development using modern frameworks and libraries.

The following tasks were set to achieve the purpose:

- analysis of the subject area and determination of the main requirements for the system;
- database design;
- front-end development using Angular;
- back-end development using Spring Boot;
- development of dynamic filtering and car search functionality.

IntelliJ IDEA Ultimate was chosen as the development environment for creating the program, utilizing technologies such as Angular for front-end development, Spring Boot for back-end development, MongoDB as the database, and programming languages Java, HTML, SCSS, and TypeScript.

The information system “Car Dealership” is designed to work on devices that have one of the following browsers installed: Chrome, Opera, Mozilla, FireFox.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

HTML	HyperText Markup Language
SCSS	Sassy Cascading Style Sheets
ІС	Інформаційна система
СУБД	Система управління базами даних

ВСТУП

Розвиток інформаційних технологій зазнав значного впливу на автомобільну промисловість, що призвело до суттєвих змін у різних аспектах бізнесу. Щоб оптимізувати роботу та задовольнити вимоги клієнтів, сучасні автосалони почали використовувати інформаційні системи, ефективно автоматизуючи та покращуючи свої процеси. Ця інтеграція призвела до підвищення ефективності роботи та можливості задовольняти потреби клієнтів.

Метою цього проєкту є розробка інформаційної системи для автосалону з використанням СУБД MongoDB. Ця система дозволить користувачам легко замовляти автомобілі через зручний інтерфейс і фільтри динамічного пошуку.

Для досягнення цієї мети необхідно розв’язати такі задачі:

- дослідити сучасні тенденції у сфері інформаційних систем для автосалонів;
- провести аналіз предметної області та визначити основні вимоги до системи;
- розробити моделі даних та структуру бази даних;
- розробити фронтенд частину системи з використанням angular;
- реалізувати бекенд частину системи за допомогою spring boot;
- забезпечити інтеграцію з базою даних mongodb;
- розробити функціонал для динамічної фільтрації та пошуку автомобілів;
- провести тестування системи та аналіз її роботи.

При побудові інформаційної системи основна увага буде зосереджена на оптимізації її зручності та ефективності, оскільки ці аспекти є вирішальними для кінцевих користувачів. Крім того, буде ретельно розглянуто потенційні шляхи майбутнього розширення та масштабованості системи, щоб підтримувати її конкурентоспроможність на ринку

Об’єкт дослідження – процес розробки інформаційної системи для

автосалону за допомогою Angular, Spring Boot, MongoDB

Предмет дослідження – розробка інформаційної системи «Автосалон», що дозволяє користувачам здійснювати замовлення автомобілів, використовуючи динамічні фільтри та пошук.

Метод дослідження – методи аналізу та проєктування інформаційних систем, методи розробки вебдодатків з використанням сучасних фреймворків та бібліотек.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ АКТУАЛЬНОСТІ ТЕМИ

1.1 Сучасний стан ринку автомобілів та онлайн-продажів

Світовий автомобільний ринок продовжує залишатися одним з найбільших і найдинамічніших секторів економіки, який адаптується до мінливих умов глобалізації і технічного прогресу. У 2023 році світове виробництво автомобілів збільшилося на 10,2% і досягнувши 76 мільйонів одиниць, при цьому основні виробничі потужності розташовані в Китаї, Сполучених Штатах, Японії та Німеччині.

Ключові ринки і тенденції:

– Азіатсько-Тихоокеанський регіон став лідером продажів у 2023 році завдяки активному ринку в Китаї, де продажі зросли на 11%;
– Європа та Північна Америка також продемонстрували значне зростання, але ці регіони ще не досягли докризового рівня, показників 2019 року.

Останнім часом ринок значно змінився у зв’язку зі зростанням популярності електромобілів, на частку яких у 2023 році припадало майже 30% від загального обсягу продажів. Інноваційні технології, такі як гібридні двигуни і автономні транспортні засоби, розширюють ринкові можливості і будуть стимулювати розробку нових стратегій продажів [1–3].

В Україні після економічних потрясінь, викликаних пандемією та іншими зовнішніми факторами, ринок поступово відновлюється, і попит на вживані і бюджетні автомобілі залишається високим. Електромобілі також починають займати свою нішу, завдяки державній підтримці та розвитку зарядної інфраструктури.

Інтернет-продажі відіграють все більш важливу роль, особливо в умовах, коли споживачі віддають перевагу зручності та безпеці онлайн-покупок. Вони пропонують доступність в режимі 24/7, широкий вибір і

можливість порівнювати ціни і характеристики без необхідності відвідування фізичного магазину. Розвиток таких сервісів, як віртуальні автосалони та онлайн-конфігуратори, підкреслює тенденцію до цифровізації продажів і спонукає автодилерів адаптувати сучасні інформаційні системи для підтримки ефективної взаємодії з клієнтами в онлайн-середовищі.

1.2 Аналіз існуючих інформаційних систем для автосалонів та їх актуальність

Інформаційні системи (ІС) стали невід'ємною частиною сучасних автосалонів, допомагаючи ефективно управляти бізнес-процесами, покращувати обслуговування клієнтів і підвищувати конкурентоспроможність.

Існуючі ІС для автосалонів:

- *CRM-системи:* Salesforce, HubSpot (вони допомагають керувати відносинами з клієнтами, зберігати контактну інформацію та відстежувати історію взаємодій та запитів);
- *ERP-системи:* SAP, Oracle NetSuite (вони комплексно управляють всіма аспектами діяльності автосалону, включаючи фінанси, складування, інвентаризацію та технічне обслуговування);
- *Спеціалізовані платформи:* DealerSocket, CDK Global (вони пропонують комплексні рішення для автодилерів, включаючи управління запасами, онлайн-продажі, маркетинг та обслуговування клієнтів).

Основні функції:

- управління інвентарем та онлайн-продажами;
- взаємодія з клієнтами та crm-функціонал;
- фінансовий менеджмент та сервісне обслуговування.

Переваги:

- підвищення ефективності та продуктивності;

- покращення обслуговування клієнтів;
- зниження витрат та оптимізація процесів;
- аналіз даних для прийняття рішень.

Актуальність розробки ІС «Автосалон»:

- *зростання онлайн-продажів:* сучасна платформа онлайн-продажів зі зручним інтерфейсом, розширеними можливостями пошуку та фільтрації, а також ефективною обробкою замовлень;
- *підвищення конкуренції:* заохочує автодилерів впроваджувати нові технології та оптимізувати бізнес-процеси для залучення клієнтів та підвищення ефективності;
- *необхідність автоматизації:* це дозволяє оптимізувати діяльність, знизити витрати і підвищити якість обслуговування клієнтів;
- *використання сучасних технологій:* Angular, Spring Boot, MongoDB забезпечує гнучкість, масштабованість, надійність і високу продуктивність;
- *економічна ефективність:* автоматизація та оптимізація процесів можуть призвести до значної економії коштів та підвищення прибутковості.

1.3 Висновки до розділу 1

Розвиток ІС «Автосалон» є доцільним і необхідним для успішної роботи автодилерів в умовах цифрової трансформації, допомагаючи адаптуватися до вимог ринку, збільшувати продажі і покращувати обслуговування клієнтів.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ «АВТОСАЛОН»

2.1 Формулювання вимог до системи

Основна мета розробки інформаційної системи «Автосалон» полягає в ефективному управлінні бізнес-процесами автосалону і задоволенні потреб користувачів. Розглянемо ключові вимоги.

Функціональні вимоги:

- *управління інвентарем*: додавання, редагування, видалення інформації про автомобілі; відстеження наявності та управління цінами;
- *пошук та фільтрація*: потужний пошуковий механізм і динамічні фільтри для швидкого знаходження автомобілів;
- *управління замовленнями*: обробка і виконання замовлень на автомобілі;
- *взаємодія з клієнтами* (CRM): зберігання контактної інформації та зворотний зв’язок з клієнтами.

Нефункціональні вимоги:

- *продуктивність*: мінімальні затримки у обробці запитів і швидка обробка великих обсягів даних;
- *масштабованість*: легке розширення функціональності і підтримка зростання користувачів і даних;
- *юзабіліті*: інтуїтивно зрозумілий інтерфейс для користувачів різного рівня підготовки.

2.2 Вибір технологій та інструментів для розробки

Процес вибору технологій та інструментів для розробки інформаційної системи «Автосалон» є важливим етапом у визначенні ефективності, продуктивності та гнучкості майбутньої системи. У цьому розділі

розглядаються основні технології та інструменти, що використовуються при розробці систем, аналізуються їх переваги та обґрунтовується вибір.

Фронтенд технології:

– *Angular*: модульна структура і двостороннє зв’язування даних для ефективної розробки інтерактивних вебдодатків;
– *HTML/SCSS/TypeScript*: стандартизація і стилізація вебсторінок з покращеною підтримкою змінних і вкладених правил.

Бекенд технології:

– *Spring Boot*: швидке налаштування і масштабованість для серверної частини, інтеграція бізнес-логіки;
– *Java*: висока надійність і кросплатформність для розробки корпоративних додатків.

Інструмент розробки:

– *IntelliJ IDEA Ultimate*: підтримка широкого спектру мов програмування і фреймворків, розширення і плагіни для підвищення продуктивності розробки.

Ці технології та інструменти були обрані виходячи з їх популярності, надійності і широких можливостей, що відкриваються при розробці масштабованої, гнучкої і продуктивної інформаційної системи «Автосалон». Використовуючи новітні платформи та інструменти, ви можете створювати високоякісні продукти, які відповідають потребам клієнтів і підтримують розвиток автосалонів.

2.3 Проєктування бази даних та моделей даних

Проєктування бази даних є важливим кроком для забезпечення ефективного зберігання та обробки даних в інформаційній системі «Автосалон». MongoDB була обрана в якості базової, оскільки її гнучкість і висока продуктивність дозволяють легко змінювати схему даних [5].

Вибір технології бази даних. База даних MongoDB NoSQL була обрана для інформаційної системи «Автосалон» через її гнучкість у роботі з даними та високу продуктивність. MongoDB зберігає дані у форматі документів JSON-подібної структури, що дозволяє легко змінювати схему даних без необхідності зупиняти систему.

Розглянемо моделі даних.

Модель Car: зберігає дані про автомобілі, включаючи ідентифікатор, назву, марку, модель, тип кузова, ціну, рік випуску, і статус доступності (див. рис. 2.1).

```
42 usages   Maxim Shepelyakovski
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Document(collection = "cars")
@QueryEntity
public class Car {
    @MongoId
    private String id;
    @Field
    private String fullName;
    @Field
    private String make;
    @Field
    private String model;
    @Field
    private List<String> bodyStyle;
    @Field
    private Long price;
    @Field
    private Integer year;
    @Field
    private Boolean enabled;
}
```

Рисунок 2.1 – Клас Car

Модель Filter: утримує налаштування фільтрів для пошуку автомобілів по критеріях як марка, модель, рік випуску, і ціна (див. рис. 2.2).

```
21 usages   Maxim Shepelyakovski
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Document(collection = "filters")
@QueryEntity
public class Filter {
    @MongoId
    private String id;
    @Field
    FilterType filterType;
    @Field
    List<String> values;
}
```

Рисунок 2.2 – Клас Filter

Модель Make: містить інформацію про марки автомобілів, включаючи назви та країни походження (див. рис. 2.3).

```
25 usages   Maxim Shepelyakovski
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Document(collection = "make")
@QueryEntity
public class Make {
    @MongoId
    private String id;
    private String make;
    private String country;
}
```

Рисунок 2.3 – Клас Make

Модель Request: використовується для зберігання замовлень, з вказівкою користувача, автомобіля та повідомлення (див. рис. 2.4).

Модель User: зберігає контактну інформацію користувачів, таку як ім’я, прізвище, телефон, електронна пошта, країна та місто (див. рис. 2.5).

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Document(collection = "requests")
@QueryEntity
public class Request {
    @MongoId
    private String id;

    @DBRef
    private User user;

    @DBRef
    private Car car;

    @Field
    private String message;
}
```

Рисунок 2.4 – Клас Request

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Document(collection = "users")
@QueryEntity
public class User {

    @MongoId
    private String id;

    @Field
    private String firstName;

    @Field
    private String lastName;

    @Field
    private String phoneNumber;

    @Field
    private String email;

    @Field
    private String country;

    @Field
    private String city;
}
```

Рисунок 2.5 – Клас User

Автоматичне створення таблиць і аннотації:

- **@Document:** визначає клас як документ у MongoDB (наприклад, **@Document(collection = "cars")** вказує на колекцію, де зберігатимуться документи);
- **@MongoId:** використовується для визначення унікального ідентифікатора в колекції;
- **@Field:** маркує поля класу для зберігання в документі MongoDB;
- **@DBRef:** для створення посилань між документами, корисно для встановлення зв’язків між колекціями.

Основні колекції бази даних:

- **cars**: інформація про автомобіль;
- **filters**: фільтри для пошуку;
- **make**: інформація про марки автомобілів;
- **requests**: записи замовленнь автомобілів;
- **users**: дані про користувачів.

Така структура бази даних на основі MongoDB дозволяє інформаційній системі «Автосалон» ефективно масштабуватися, адаптуватися до мінливих вимог і обсягів даних, а також забезпечувати швидкий доступ до необхідної інформації і високий рівень продуктивності.

2.4 Розробка архітектури системи

Архітектура інформаційної системи «Автосалон» спрямована на ефективне управління бізнес-процесами автосалону та забезпечення високої продуктивності, масштабованості та простоти використання. Ключовими компонентами архітектури є фронт-енд, бек-енд та база даних. У цьому розділі наведено детальний опис архітектури системи, використаних технологій та принципів їх взаємодії:

- інтерфейс користувача: забезпечення зручного та швидкого доступу до

функціоналу системи;

– динамічні фільтри та пошук: реалізація потужних інструментів для пошуку та фільтрації автомобілів за різними критеріями;
– інтерактивні компоненти: використання двостороннього зв’язування даних для забезпечення реального часу оновлення інтерфейсу.

Клієнтська частина (Front-end).

Технологія: Angular.

Основні функції:

– інтерфейс користувача: забезпечує швидкий доступ до функціоналу системи;
– динамічні фільтри та пошук: можливості пошуку автомобілів за різними параметрами;
– інтерактивні компоненти: двостороннє зв’язування даних для оновлення інтерфейсу в реальному часі.

Серверна частина (Back-end).

Технологія: Spring Boot.

Основні компоненти:

– API: реалізація RESTful API для взаємодії між клієнтом та сервером;
– управління даними: обробка запитів до бази даних, ке-рування даними.

База даних.

Технологія: MongoDB.

Особливості: гнучкість і висока продуктивність, зберігання даних у форматі документів, що сприяє легкій модифікації схеми даних.

Схема архітектури системи:

– клієнтський шар: інтерфейс користувача розроблений на Angular;
– серверний шар: обробка бізнес-логіки та запитів від клієнта, взаємодія з базою даних;
– шар зберігання даних: використання MongoDB для зберігання всіх необхідних даних системи.

Наведемо приклади основних компонентів.

RESTful API: MakeController – контролер для управління даними про марки автомобілів (див. рис. 2.6).

```
Maxim Shepelyakovski
@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/make")
public class MakeController {

    3 usages
    private final MakeService makeService;

    Maxim Shepelyakovski
    @Contract(pure = true)
    public MakeController(MakeService makeService) { this.makeService = makeService; }

    Maxim Shepelyakovski
    @GetMapping(value = "/all")
    public List<MakeDTO> getAllMake() { return makeService.findAllMakes(); }

    Maxim Shepelyakovski
    @PostMapping(value = "/add")
    public void addMakes(
            @RequestBody List<MakeDTO> makesDTO
    ) {
        makeService.addMakes(makesDTO);
    }

}
```

Рисунок 2.6 – Контролер MakeController

Інтерфейс користувача: показує знімок екрану з інтерфейсом, який взаємодіє з користувачами (див. рис. 2.7).

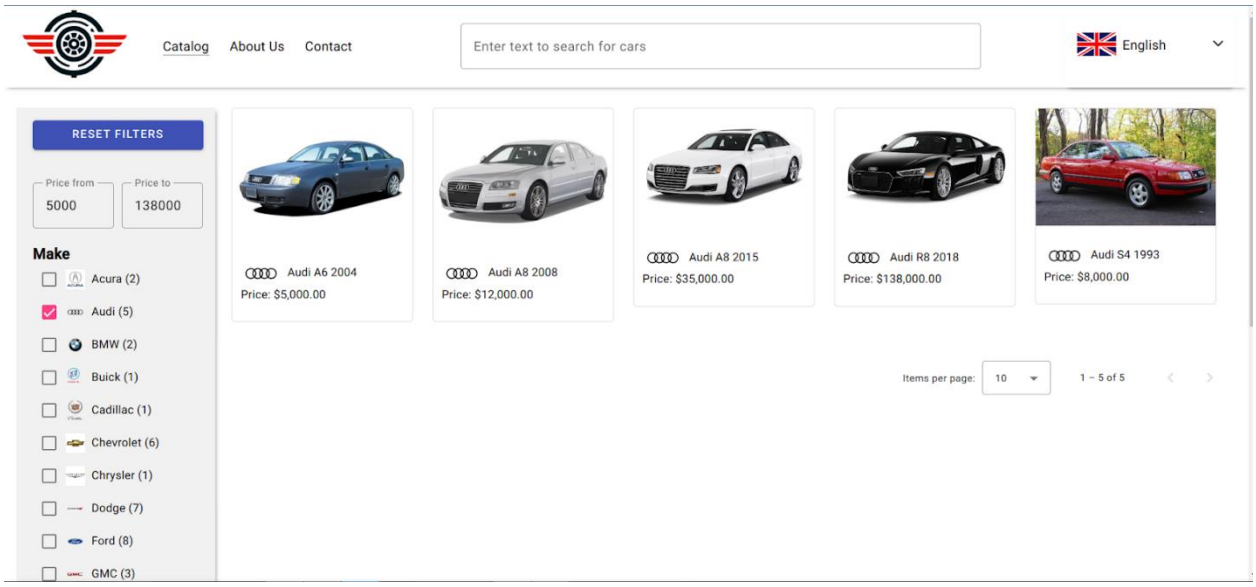


Рисунок 2.7 – Знімок інтерфейсу користувача

Архітектура «Автосалон» розроблена таким чином, щоб забезпечити простоту управління, високу продуктивність і адаптивність до змін в бізнес-процесах. Конфігураційний підхід дозволяє легко розширювати систему і адаптувати її до вимог користувачів і ринків.

2.5 Проєктування інтерфейсу користувача

Дизайн користувальницького інтерфейсу (UI) в інформаційній системі «Автосалон» спрямований на забезпечення зручності та інтуїтивності взаємодії користувача з системою. Нижче описані основні принципи і структура інтерфейсу.

Основні принципи проєктування інтерфейсу:

- *простота та інтуїтивність*: це забезпечує легкий доступ до важливих функцій і робить інтерфейс зрозумілим користувачам різного рівня;
- *консистентність*: єдність стилю та поведінки елементів для зручності та передбачуваності взаємодії;
- *візуальна ієрархія*: виділення важливої інформацію та функції, щоб привернути увагу користувача.

Структура та компоненти інтерфейсу користувача:

- *головна сторінка*: надає загальну інформацію про автосалон та доступні марки;
- *каталог автомобілів*: дозволяє користувачам переглядати автомобілі та застосовувати фільтри для уточнення пошуку;
- *форма замовлення*: можливість для користувачів оформити замовлення, вказати контактні дані та додаткову інформацію;
- *сторінка контактів*: включає інформацію для зв’язку з автосалоном, мапу розташування, та форму для зворотнього зв’язку.

Приклади знімків екрана.

На рисунку 2.8 можна побачити сторінку каталогу автомобілів, на рисунку 2.9 – форму замовлення, на рисунку 2.10 – сторінку «Контакти».

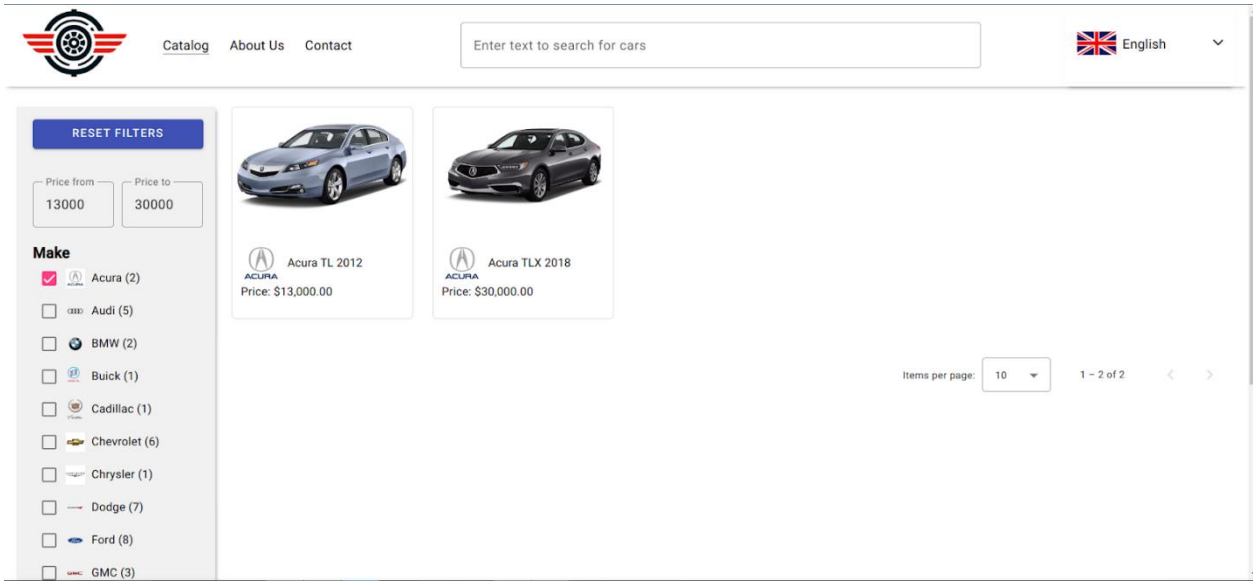


Рисунок 2.8 – Каталог автомобілів марки «Acura»

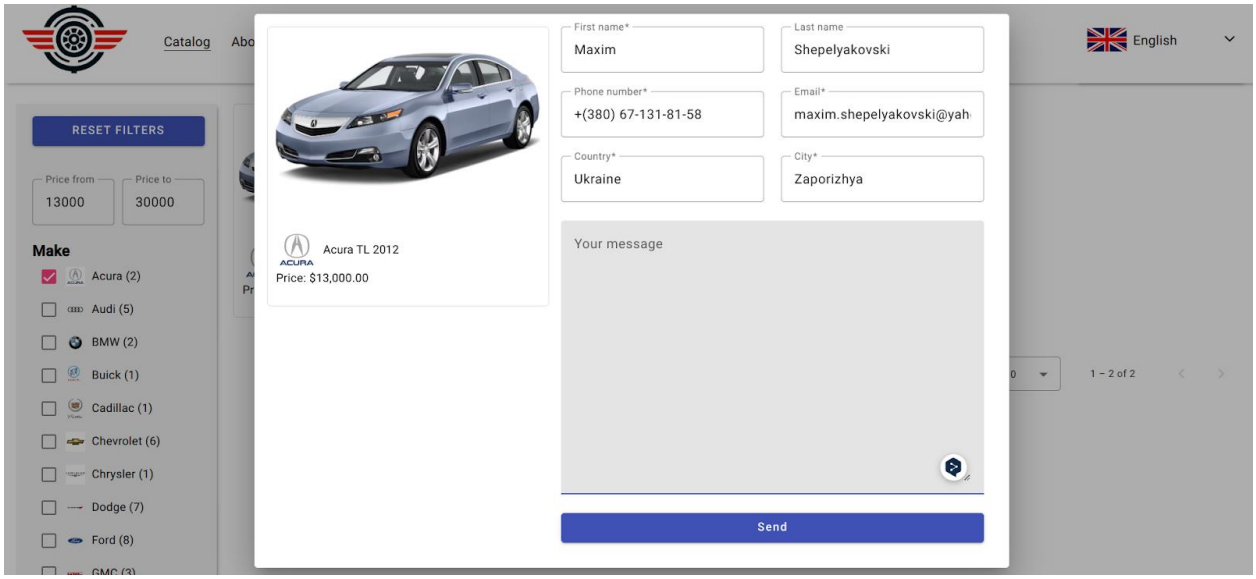


Рисунок 2.9 – Форма замовлення

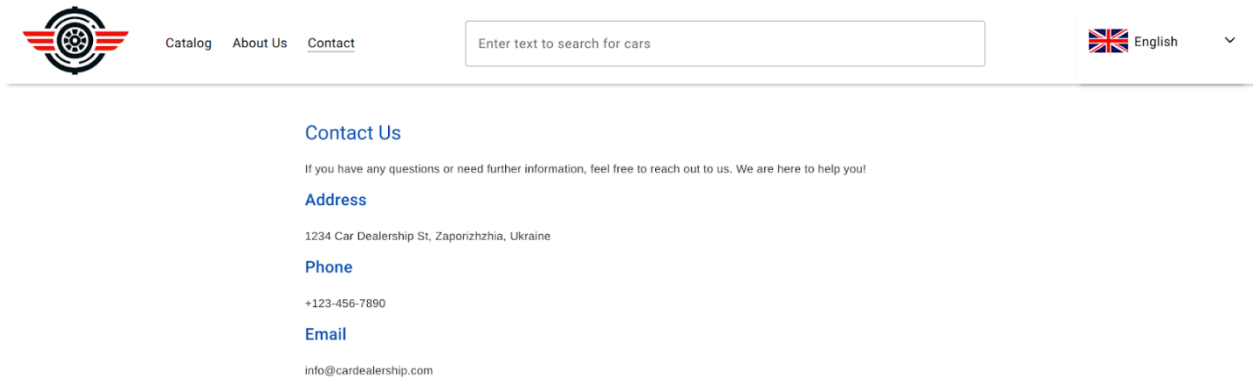


Рисунок 2.10 – Сторінка «Контакти»

На рисунку 2.11 можна побачити сторінку «Про нас».

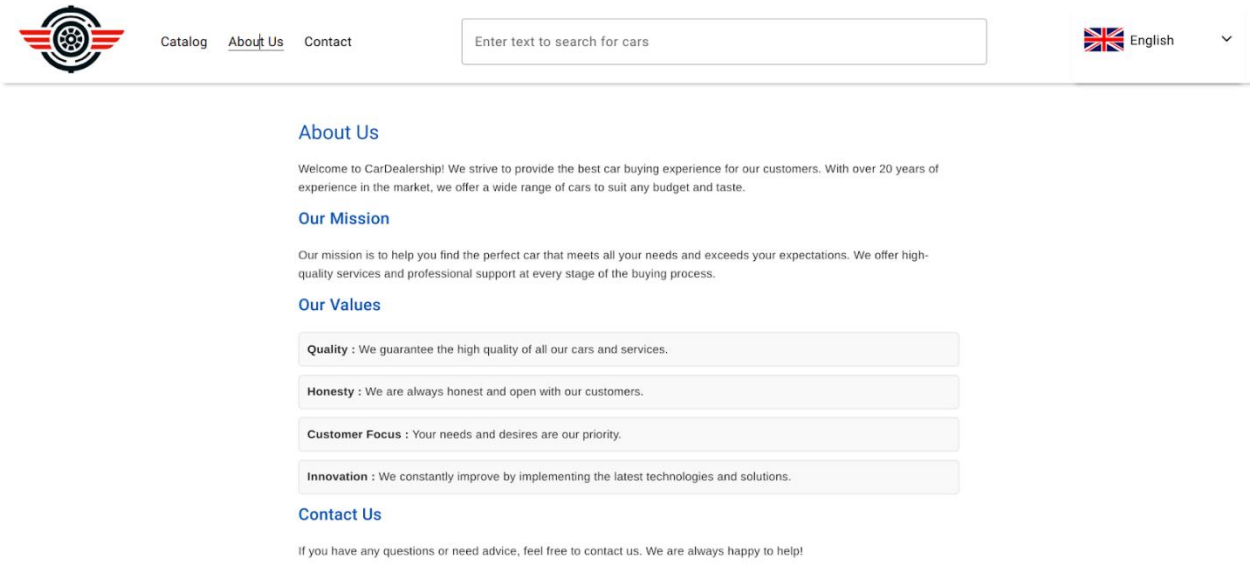


Рисунок 2.11 – Сторінка «Про нас»

Дизайн користувальницького інтерфейсу в системі «Автосалон» спрямований на створення зручного, функціонального і привабливого середовища, що сприяє оптимальній взаємодії користувача з системою і підвищує ефективність роботи автодилера.

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ «АВТОСАЛОН»

3.1 Розробка фронтенд частини системи з використанням Angular

Процес розробки інтерфейсу для системи «Автосалон» в Angular спрямований на створення інтерактивного і зручного для користувача інтерфейсу, який оптимізує взаємодію з системою [4].

3.1.1 Структура проєкту

Проєкт має модульну структуру, що забезпечує простоту розширення та підтримки.

App:

- app.component.ts: кореневий компонент;
- app.module.ts: головний модуль, який організує залежності;
- app-routing.module.ts: конфігурація маршрутизації.

Pages:

- dealership/about-us.component.ts: інформація «Про нас»;
- dealership/car-dealership.component.ts: головна сторінка автосалону;
- dealership/contact.component.ts: контактна інформація.

Services:

- car-service.service.ts: сервіс для роботи з даними автомобілів;
- make-service.service.ts: сервіс для роботи з марками автомобілів.

Directives:

- phone-mask.directive.ts: директива для маски вводу телефону.

Interfaces:

- car.interface.ts: інтерфейси для типів даних (Car, Make та інших).

3.1.2 Конфігураційні файли

Основні конфігураційні файли:

– angular.json: конфігурація Angular CLI, що визначає налаштування проєкту;
– tsconfig.json: налаштування компіляції TypeScript;
– package.json: управління залежностями та конфігурація проєкту.

3.1.3 Основні файли проєкту

Файл app.component.ts (див. рис. А.1).

Функціонал: ініціалізація додатка та управління мовами інтерфейсу.

Ключові аспекти:

– @Component декоратор, який налаштовує HTML-тег, шлях до шаблону та стилів;
– метод initLanguages(), який налаштовує доступні мови відповідно до налаштувань середовища та користувацьких уподобань.

Файл app.module.ts (див. рис. А.2).

Функціонал: конфігурація Angular модулів, компонентів та сервісів.

Особливості:

– @NgModule декоратор, який визначає компоненти та модулі, необхідні для додатку;
– імпорти основних модулів, як BrowserModule, HttpClientModule, та TranslateModule;
– конфігурація маршрутизації та локалізації через AppRoutingModule та TranslateModule.

Важливі аспекти розробки:

– *модульність та розширюваність*: система побудована з використанням модульної архітектури, яка дозволяє легко розширювати і модифікувати

функціональність;

- *локалізація*: використовуючи TranslateService для управління мовами, зручно перемикати мови і забезпечувати гнучкість при локалізації контенту;
- *завантаження та запуск*: AppComponent є відправною точкою для запуску всіх необхідних компонентів та служб під час завантаження програми.

Така структура забезпечує зручність взаємодії користувача з системою, ефективне використання ресурсів і високий рівень продуктивності вебдодатків.

3.1.4 Маршрутизація

Маршрутизація в Angular є ключем до організації навігації між сторінками програми та забезпечення плавного переходу та взаємодії без перезавантаження сторінки.

Файл app-routing.module.ts.

Головні маршрути:

- ‘admin’: завантажує AdminModule для адміністративного інтерфейсу;
- ‘‘(кореневий): завантажує DealershipModule для основних сторінок автосалону.

Файл dealership-routing.module.ts.

Маршрути модуля автосалону:

- ‘main’: завантажує CarDealershipComponent для відображення основної сторінки автосалону;
- ‘about-us’: завантажує AboutUsComponent, яка містить інформацію про компанію;
- ‘contact’: завантажує ContactComponent для відображення контактної інформації та форми зворотнього зв’язку.

3.1.5 Інтерфейси

За інтерфейси відповідає файли car.interface.ts та language.interface.ts. Основні інтерфейси:

- Make: визначає марку автомобіля з полями для назви марки та країни походження;
- Car: вписує автомобіль, включаючи ідентифікатор, повну назву, марку, модель, ціну, рік випуску та країну виробництва;
- User: визначає користувача з полями для імені, прізвища, телефону, електронної пошти, країни та міста;
- Request: описує запит на покупку автомобіля з деталями про користувача, автомобіль та додаткове повідомлення;
- Enum FilterType: визначає типи фільтрів (марка, модель, рік, ціна, країна);
- SearchFilter і FilterCriteria: використовуються для опису фільтрів пошуку з визначенням типу фільтра та значень;
- FilterOption: визначає варіанти фільтра з ключем, розміром та додатковою інформацією;
- Language: описує мову інтерфейсу з полями для коду мови, назви та шляху до іконки прапора.

Використання цих інтерфейсів забезпечує чітку, структуровану організацію даних у додатку, а також інтеграцію з компонентами Angular. Це забезпечує високий рівень читабельності коду та правильну взаємодію з даними в додатку.

3.1.6 Головні сторінки

Сторінка “About Us”. На сторінці «Про нас» інформаційної системи «Автосалон» представлена інформація про компанію, її місію та цінності. Ця

сторінка реалізована з використанням Angular.

Файл about-us.component.ts визначає логіку сторінки «Про нас». Він використовує декоратор @Component для вказівки селектора, HTML-шаблону та стилів. Клас AboutUsComponent відповідає за обробку даних і взаємодію з користувачем.

Файл about-us.component.scss містить стилі, що забезпечують адаптивність та візуальну привабливість сторінки. Він стилізує заголовки, параграфи, списки та інші елементи, щоб досягти консистентного та естетичного вигляду інтерфейсу.

Файл about-us.component.html є HTML-шаблоном, який відображає контент сторінки. Він використовує Angular-пайпи для перекладу тексту, що забезпечує мультиязичну підтримку. Структура шаблону включає заголовки, параграфи та списки, які представляють ключову інформацію про компанію.

Сторінка “Contact”. Сторінка «Контакти» надає користувачам інформацію про адресу компанії, номер телефону та адресу електронної пошти. Ця сторінка реалізована з використанням компонентів Angular і містить три основні файли: contact.component.ts, contact.component.html та contact.component.scss.

Файл contact.component.ts (див. рис. А.6).

Ключові моменти:

– декоратор @Component, який вказує на селектор компоненту, шаблон HTML та стилі;
– клас ContactComponent, який містить основну логіку для роботи компоненту.

Файл contact.component.scss (див. рис. А.7).

Ключові моменти:

– визначення стилів для різних елементів сторінки, таких як заголовки, параграфи, та контейнер;
– забезпечення адаптивності та візуальної привабливості сторінки.

Файл contact.component.html (див. рис. А.8).

Ключові моменти:

– використання Angular-пайпів для перекладу тексту (translate);
– структура сторінки включає заголовки, параграфи, які відображають контактну інформацію компанії.

Сторінка “Catalog”. Сторінка «Каталог» системи «Автосалон» дозволяє користувачам ефективно переглядати, фільтрувати та сортувати автомобілі за допомогою зручних компонентів Angular.

Файл car-dealership.component.ts.

Функціональність: логіка компоненту для управління списком автомобілів, включаючи фільтрацію, сортування та пагінацію.

Методи:

– ngOnInit(): ініціалізація компоненту, створення форм для фільтрації, завантаження даних (див. рис. А.9);
– createForms(): створення форм для фільтрації за ціною та іншими параметрами (див. рис. А.10);
– subscribeToFormChanges(): підпис на зміни в формах для динамічного оновлення списку автомобілів (див. рис. А.11);
– getCars(): завантаження списку автомобілів з сервера (див. рис. А.12);
– processFilterSelectionAndUpdate(): обробка вибору фільтрів та оновлення відфільтрованих результатів (див. рис. А.13);
– resetFilters(): скидання всіх фільтрів до початкового стану (див. рис. А.14);
– openOrderModal(): відкриття модального вікна для замовлення обраного автомобіля (див. рис. А.15);
– onPageChange(): обробка змін сторінки пагінації для оновлення відображення автомобілів (див. рис. А.16).

Файл car-dealership.component.scss (див. рис. А.17–А.19).

Функціональність: стилізація сторінки «Каталог», включаючи контейнери, панелі фільтрів, картки автомобілів, та пагінацію.

Особливості:

– забезпечення адаптивності для різних розмірів екранів;
– візуальна привабливість, що підсилює зручність користування.

Файл car-dealership.component.html (див. рис. А.20–А.23).

Функціональність: HTML-шаблон, який відображає фільтри, список автомобілів, та елементи пагінації.

Особливості:

– використання Angular компонентів для динамічного відображення фільтрів та результатів пошуку;
– розгортання фільтрів та кнопка для скидання дозволяють користувачам легко налаштовувати параметри пошуку.

3.1.7 Компонент та модульне вікно

Компонент “Car Card”. Компонент “Car Card” в системі «Автосалон» призначений для візуалізації інформації про автомобілі у форматі карток, що використовуються на сторінці каталогу. Цей компонент складається з файлів, які управляють логікою, відображенням та стилем.

Файл car-card.component.ts (див. рис. А.24).

Функціональність: компонент обробляє дані автомобіля та управляє їх відображенням.

Особливості:

– @Input() car: Car: приймає об’єкт автомобіля для відображення;
– imageExists: перевіряє наявність зображення автомобіля;
– ngOnInit(): ініціалізація компоненту, активує перевірку на наявність зображення.

Файл car-card.component.html (див. рис. А.25).

Функціональність: HTML-шаблон для відображення картки автомобіля.

Особливості:

– відображення зображення автомобіля або стандартного зображення при

його відсутності;

– використання директиви ngIf для умовного відображення елементів;
– відображення основної інформації про автомобіль, включаючи марку та повну назву.

Файл car-card.component.scss (див. рис. А.26, А.27).

Функціональність: стилізація картки для забезпечення привабливого та адаптивного відображення.

Особливості: стилі для контейнера картки, зображення автомобіля та текстової інформації.

Модальне вікно замовлення (Order Modal) системи «Автосалон» дозволяють користувачам замовляти автомобілі безпосередньо з інтерфейсу каталогу. Цей компонент складається з файлів, які керують логікою, відображенням та стилем.

Файл order-modal.component.ts.

Функціональність: компонент управляє логікою формування замовлення.

Особливості:

– @Input() car: Car: приймає об’єкт автомобіля для замовлення;
– form: реактивна форма для введення даних замовлення;
– ngOnInit(): ініціалізує форму замовлення з валідацією;
– makeOrder(): обробляє подання форми та відправляє замовлення на сервер (див. рис. А.28).

Файл order-modal.component.html.

Функціональність: HTML-шаблон модального вікна, що відображає форму замовлення.

Особливості:

– використання формових контролів для збору інформації від користувача;
– кнопка для відправлення замовлення, що активує метод makeOrder().

Файл order-modal.component.scss.

Функціональність: стилізація модального вікна для забезпечення чіткості та візуальної привабливості.

Особливості: стилізація форми та кнопок для забезпечення зручності та естетичності інтерфейсу.

3.1.8 Сервіси

У програмах Angular сервіси відіграють важливу роль у взаємодії даних та логіки між компонентами.

CarService – сервіс для роботи з даними автомобілів.

Методи CarService:

- saveCar(car: Car): зберігання нового автомобіля в базі даних (див. рис. А.29);
- getExistsMakes(): отримання списку всіх марок автомобілів, наявних у базі (див. рис. А.30);
- getAllCars(): отримання всіх автомобілів з бази даних (див. рис. А.31);
- getCarsByMake(make: string): отримання автомобілів за вказаною маркою (див. рис. А.32);
- updateCar(car: Car): оновлення інформації про автомобіль (див. рис. А.33);
- deleteCar(fullName: string): видалення автомобіля за повною назвою (див. рис. А.34);
- filterCars(searchFilters: SearchFilter[]): фільтрація автомобілів за заданими критеріями (див. рис. А.35);
- getFilterWithValues(searchFilters: SearchFilter[]): отримання критеріїв фільтрації з відповідними значеннями (див. рис. А.36).

MakeService – сервіс для роботи з марками автомобілів.

Метод MakeService: getMakes() отримує всі доступні марки автомобілів (див. рис. А.37).

ModalService – сервіс для роботи з модальними вікнами.

Методи ModalService:

– openOrderModal(config: MatDialogConfig): відкриває модальне вікно для замовлення автомобіля (див. рис. А.38);
– openThankYouModal(): відкриває модальне вікно «Дякуємо за замовлення». (див. рис. А.38).

OrderService – сервіс для обробки замовлень.

Метод: sentOrder(request: Request): Observable<any> відправляє запит на обробку замовлення автомобіля (див. рис. А.39).

Сервіси Angular забезпечують централізоване управління даними та логікою, що робить код компонента більш зрозумілим і простим у обслуговуванні. За допомогою них ви можете ефективно взаємодіяти з сервером, керувати станом програми та обробляти запити користувачів.

3.2 Реалізація бекенд частини системи за допомогою Spring Boot

У цьому розділі описується розробка серверної частини для системи «Автосалон» з використанням Spring Boot. Основні аспекти розробки включають структуру проєкту, контролери, репозиторії, DTO, документи та сервіси [6].

3.2.1 Структура проєкту

Структура заснована на стандартній побудові Spring Boot, що сприяє легкій організації та підтримці коду:

– *основні файли*: build.gradle для налаштувань збірки, CarDealershipApplication.java як головний клас запуску;
– *контролери*: класи, як CarController, FilterController, що управляють

HTTP-запитами;

- *документи та DTO*: використання Spring Data для представлення даних у базі та передачі їх між сервером та клієнтом;
- *мапери*: класи для конвертації даних між ентіті та DTO.;
- *репозиторії та сервіси*: забезпечення взаємодії з базою даних та обробки бізнес-логіки;
- *утиліти*: допоміжні класи для оптимізації роботи з даними.

Такий підхід забезпечує чітку і систематичну реалізацію серверної частини, сприяючи ефективній підтримці і розширенню системи.

3.2.2 Конфігурація додатка

Файл **build.gradle** містить налаштування для збірки проєкту, а саме:

- *плагіни*: підтримка Java, Spring Boot, і управління залежностями (див. рис. А.40);
- *залежності*: набори інструментів Spring Boot для створення вебдодатків, взаємодії з MongoDB, відправки електронної пошти, і тестування (див. рис. А.41);
- *репозиторії*: налаштування завантаження залежностей з Maven Central (див. рис. А.42).

Файл **application.yml** містить детальні налаштування додатку, а саме:

- *MongoDB*: налаштування підключення до бази даних, включаючи URI зі зразками автентифікації та іменем бази даних;
- *Email*: конфігурація SMTP для відправки електронної пошти, включаючи сервер, порт, ім’я користувача, пароль, і параметри безпеки;
- *логування*: встановлення рівнів логування для специфічних компонентів, наприклад, налаштування DEBUG для MongoTemplate для відлагодження.

3.2.3 Документи

Система використовує MongoDB для зберігання моделі даних. Розглянемо основні документи та їх структури.

Модель даних для автомобілів є клас **Car** (див. рис. А.44). Цей клас зберігається у колекції “cars” за допомогою анотацій @Document, @MongoId та @Field, які використовуються для ідентифікації та зберігання полів у MongoDB. Основні поля включають ідентифікатор, повну назву, марку, модель, тип кузова, ціну та рік випуску.

Для фільтрів пошуку використовується клас **Filter** (див. рис. А.45). Цей клас зберігається у колекції “filters” за допомогою анотації @Document. Основні поля включають ідентифікатор, тип фільтра та значення фільтра.

Для марок автомобілів використовується клас **Make** (див. рис. А.46). Цей клас зберігається у колекції “make” за допомогою анотації @Document. Основні поля включають ідентифікатор, назву марки та країну походження.

Модель даних для запитів на автомобілі є клас **Request** (див. рис. А.47). Цей клас зберігається у колекції “requests” за допомогою анотації @Document. Основні поля включають ідентифікатор, користувача (посилання), автомобіль (посилання), повідомлення та мову запиту.

Модель даних для користувачів представлена класом **User** (див. рис. А.48). Цей клас зберігається у колекції “users” за допомогою анотації @Document. Основні поля включають ідентифікатор, ім’я, прізвище, номер телефону, електронну пошту, країну та місто.

Перелік типів фільтрів для пошуку представлений **enum FilterType** (див. рис. А.49). Основні поля включають MAKE, MODEL, YEAR, PRICE та COUNTRY, що відповідають різним критеріям пошуку.

3.2.4 Мапери

Мапери у Spring Boot використовуються для перетворення між об’єктами документів та DTO (Data Transfer Objects). Для реалізації маперів використовується бібліотека MapStruct, яка забезпечує автоматичне генерування коду для перетворення між різними типами об’єктів [7].

CarMapper (див. рис. А.50) відповідає за конвертацію між об’єктами Car і CarDTO. Він використовує спеціальний конвертер у полі та ініціалізує шлях до фотографії після відображення.

FilterMapper (див. рис. А.51) конвертує між об’єктами Filter і FilterDTO.

MakeMapper (див. рис. А.52) конвертує між об’єктами Make і MakeDTO, ігнорує невідповідні поля в DTO для забезпечення точності відображення.

RequestMapper (див. рис. А.53) конвертує між об’єктами Request і RequestDTO;

UserMapper (див. рис. А.54) конвертує між об’єктами User і UserDTO.

3.2.5 Утиліти

Утилітарні класи у Spring Boot містять допоміжні методи, які використовуються для виконання певних завдань у різних частинах програми.

Клас **CarUtils** (див. рис. А.55) відповідає за роботу з даними автомобілів. Він містить методи:

- getPathToPhoto(): генерує шлях до фотографії автомобіля;
- getPathToPhotoCar(): генерує повний шлях до фотографії автомобіля з назвою файлу;
- getPathToPhotoForLogo(): генерує шлях до логотипу марки автомобіля.

CriteriaUtils відповідає за створення критеріїв пошуку у MongoDB. Він включає методи:

– getCriteriaByFilters(): створює критерії пошуку на основі заданого типу фільтру, списку фільтрів та марок;
– getCriteriaByFilters(): створює критерії пошуку на основі списку фільтрів та марок;
– parsedValuesModel(): обробляє значення моделі автомобіля для критеріїв.

Клас **MakeUtils** (див. рис. А.56) працює з даними марок автомобілів. Включає таки метод:

– getPathToPhoto(): генерує шлях до фотографії марки автомобіля.

Крім того, утиліти не тільки спрощує роботу з даними і генерує необхідні шляхи для фотографій і логотипів, але і допомагає створювати критерії пошуку, що значно підвищує ефективність коду і його читабельність.

3.2.6 Контролери

Контролери у Spring Boot відповідають за обробку HTTP-запитів від клієнтів, виклик відповідних сервісів та повернення результатів. Розгляне-мо основні контролери, та їхні ендпоінти.

Контролер **CarController** та його ендпоінти:

– GET /auto/all – отримання списку всіх автомобілів (див. рис. А.57);
– GET /auto/make/{make} – отримання автомобілів за маркою (див. рис. А.58);
– GET /auto/make – отримання списку існуючих марок (див. рис. А.59);
– POST /auto/add – додавання нового автомобіля (див. рис. А.60);
– PATCH /auto/update – оновлення даних автомобіля (див. рис. А.61);
– DELETE /auto/delete/{fullName} – видалення автомобіля за повним ім’ям (див. рис. А.62);
– POST /auto/filter/search – пошук автомобілів за фільтрами (див. рис. А.63);

– POST /auto/filter/options/get – отримання опцій фільтрів для пошуку (див. рис. А.64).

Контролер **FilterController** та його ендпоінти (див. рис. А.65):

– POST /filter/add – додавання нового фільтра;
– GET /filter/all – отримання списку всіх фільтрів.

Контролер **MakeController** та його ендпоінти (див. рис. А.66):

– GET /make/all – отримання списку всіх марок;
– POST /make/add – додавання нових марок.

Контролер **OrderController** та його ендпоінти (див. рис. А.67):

– POST /order/sent – обробка замовлення автомобіля.

3.2.7 Сервіси

Сервіси у Spring Boot реалізує бізнес-логіку програми та забезпечує взаємодію між контролерами та репозиторіями. Давайте розглянемо основні сервіси та їх методи.

CarService – сервіс для управління даними автомобілів.

Методи:

– addCars(): додає новий автомобіль у базу даних (див. рис. А.68);
– updateCars(): оновлює дані автомобіля (див. рис. А.69);
– deleteCar(): видаляє автомобіль за повним ім’ям (див. рис. А.70);
– findAllCars(): отримує всі автомобілі (див. рис. А.71);
– findByFullName(): знаходить автомобіль за повним ім’ям (див. рис. А.72);
– findExistsMakes(): отримує всі наявні марки автомобілів (див. рис. А.73);
– findAllCarsByFilter(): фільтрує автомобілі за заданими критеріями (див. рис. А.74);
– getFilterOptionsBySearchFilter(): отримує опції фільтрів для пошуку (див. рис. А.75).

EmailService – сервіс для відправки електронних листів.

Методи:

– sendMessage(): відправляє електронний лист із підтвердженням замовлення (див. рис. А.76);
– buildEmailText(): створює текст електронного листа на основі шаблону (див. рис. А.77);
– loadMessages(): завантажує повідомлення для електронного листа відповідно до обраної мови (див. рис. А.78).

FilterService – сервіс для управління фільтрами пошуку.

Методи:

– addFilter(): додає новий фільтр у базу даних (див. рис. А.79);
– findAllFilter(): отримує всі фільтри (див. рис. А.80);
– getAllMakeValues(): отримує значення для фільтра «марка» (див. рис. А.81);
– findAllModelValues(): отримує значення для фільтра «модель» (див. рис. А.82);
– findAllYearValues(): отримує значення для фільтра «рік» (див. рис. А.83);
– findAllPriceValues(): отримує значення для фільтра «ціна» (див. рис. А.84);
– findAllCountryValues(): отримує значення для фільтра «країна» (див. рис. А.85).

MakeService – сервіс для управління марками автомобілів (див. рис. А.86).

Методи:

– findAllMakes(): отримує всі марки автомобілів;
– addMakes(): додає нові марки автомобілів у базу даних.

OrderService – сервіс для обробки замовлень (див. рис. А.87).

Метод:

– makeOrder(): обробляє замовлення автомобіля, зберігає його у базі даних та відправляє підтвердження електронною поштою.

UserService – сервіс для управління даними користувачів (див. рис. А.87).

Метод:

– saveUser(): зберігає дані користувача у базу даних.

3.2.8 Репозиторії

Репозиторії у Spring Boot надають доступ до бази даних і виконують CRUD-операції та інші специфічні запити. Цей проєкт використовує репозиторії для MongoDB, які містять основні способи взаємодії з документами в базі даних. Давайте розглянемо основні репозиторії та їх методи.

CarRepository (див. рис. А.89).

CarRepository – інтерфейс репозиторія для управління даними автомобілів.

CarRepositoryImpl – імплементація кастомних методів для CarRepository.

Методи:

– findAllCar(): отримує всі автомобілі (див. рис. А.90);
– deleteCar(): видаляє автомобіль за повним ім’ям (див. рис. А.90);
– addCar(Car car): додає новий автомобіль у базу даних (див. рис. А.90);
– updateCar(): оновлює дані автомобіля (див. рис. А.91);
– findAllModelsByMake(): отримує всі моделі автомобілів за маркою (див. рис. А.91);
– findExistsMakes(): отримує наявні марки автомобілів за фільтрами (див. рис. А.92);
– findAllUniqueMakes(): отримує всі унікальні марки автомобілів (див. рис. А.93);
– findAllCarsByFilter(): фільтрує автомобілі за заданими критеріями (див.

рис. А.93);

– findExistsModels(): отримує наявні моделі автомобілів за фільтрами (див. рис. А.94);
– findExistsYears(): отримує наявні роки випуску автомобілів за фільтрами (див. рис. А.95);
– findMinMaxPrices(): отримує мінімальні та максимальні ціни автомобілів за фільтрами (див. рис. А.96, А.97);
– getAllCars(): отримує всі автомобілі з деталями (див. рис. А.98).

FilterRepository (див. рис. А.99).

FilterRepository – інтерфейс репозиторія для управління фільтрами.

FilterRepositoryImpl – імплементація кастомних методів для FilterRepository.

Метод:

– addFilter(Filter filter): додає новий фільтр у базу даних.

MakeRepository (див. рис. А.101).

MakeRepository – репозиторій для управління марками автомобілів.

MakeRepositoryImpl – імплементація кастомних методів для MakeRepository.

Методи:

– findAllMake(): отримує всі марки автомобілів (див. рис. А.102);
– addMakes(): додає нові марки автомобілів у базу даних (див. рис. А.102);
– findUniqueCountries(): отримує унікальні країни марок автомобілів за фільтрами (див. рис. А.103).

ВИСНОВКИ

У ході даної роботи було розроблено інформаційну систему для автосалону з використанням найсучасніших технологій, таких як Angular, Spring Boot і MongoDB. За допомогою цієї системи можна ефективно управляти процесом роботи автосалону і надавати зручний інтерфейс, динамічну фільтрацію і пошук автомобілів.

Основні результати роботи:

– аналіз предметної області: виявлено основні тенденції на автомобільному ринку і онлайн-продажах; проаналізовано існуючі інформаційні системи для автосалонів і її функції; обґрунтовано актуальність розробки нових інформаційних систем для автосалону в умовах цифрової трансформації;
– проєктування інформаційної системи: сформульовано вимоги до системи, включаючи функціональні та нефункціональні вимоги; були обрані технології та інструменти для розробки системи, такі як Angular для фронтенду, Spring Boot для бекенду та MongoDB для бази даних; розроблено моделі даних та структури баз даних на основі MongoDB; розроблено архітектуру системи, включаючи взаємодію з клієнтською та серверною частинами, а також базою даних; користувальницький інтерфейс розроблений з урахуванням принципів простоти, інтуїтивності, послідовності і адаптивності;
– реалізація інформаційної системи: інтерфейсна частина системи створена з використанням Angular, яка включає в себе ключові компоненти, директиви і сервіси; у додатку реалізована маршрутизація, що забезпечує навігацію між різними сторінками системи; реалізована підтримка багатомовного користувальницького інтерфейсу з використанням NGX-translate; розроблена серверна частина системи, що використовує Spring Boot, включаючи контролери, сервіси та

репозиторії; передбачена інтеграція з базою даних MongoDB для зберігання і обробки даних; реалізовані функції динамічної фільтрації і пошуку автомобілів.

Розробка інформаційних систем для автосалонів з використанням сучасних технологій оптимізує бізнес-процеси, підвищує ефективність роботи автосалонів, а високоякісні технології Angular, Spring Boot і MongoDB дозволяють створювати гнучкі, масштабовані і надійні системи, що відповідають вимогам сучасного авторинку.

В майбутньому можливе розширення функціональності системи за рахунок додавання нових модулів і інструментів для управління автосалоном.

ПЕРЕЛІК ПОСИЛАНЬ

1. Economic and Market Report: Global and EU Auto Industry Full-Year 2023. *ACEA*. URL: https://www.acea.auto/publication/economic-and-market-report-global-and-eu-auto-industry-full-year-2023/ (дата звернення: 20.01.2024).
2. 2023 Full-Year International Worldwide Car Sales**.** *Best Selling Cars*. URL: https://www.best-selling-cars.com/international/2023-full-year-international-worldwide-car-sales/ (дата звернення: 20.01.2024).
3. Visualizing Global Electric Vehicle Sales in 2023 by Market Share**.** *Visual Capitalist*. URL: https://www.visualcapitalist.com/visualizing-global-electric-vehicle-sales-in-2023-by-market-share/ (дата звернення: 20.01.2024).
4. Angular Documentation. *Angular*. URL: https://angular.io/docs (дата звернення: 20.01.2024).
5. MongoDB Documentation. *MongoDB*. URL: https://www.mongodb.com/docs/ (дата звернення: 20.02.2024).
6. Spring Boot Documentation. *Spring*. URL:https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ (дата звернення: 20.02.2024).
7. MapStruct Documentation. *MapStruct*. URL: https://mapstruct.org/documentation/stable/reference/html/(дата звернення: 20.02.2024).

ДОДАТОК А

Програмна реалізація

.

```
1+ usages  Maxim Shepelyakovski
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent implements OnInit {
  title : string  = 'car-dealership-fe';

  no usages  Maxim Shepelyakovski
  constructor(
    private translate: TranslateService
  ) {
  }

  no usages  Maxim Shepelyakovski
  async ngOnInit() : Promise<void>  {
    await this.initLanguages()
  }

  1+ usages  Maxim Shepelyakovski
  async initLanguages() : Promise<void>  {
    this.translate.addLangs( langs: environment.languages.available);
    const platformLang : string  = window.navigator.language.split( separator: '-')[0];
    environment.languages.available.forEach(lang : string  => {
      this.translate.reloadLang( lang: lang)
    });
    if (environment.languages.available.includes(platformLang)) {
      this.translate.use( lang: platformLang);
    } else {
      this.translate.use( lang: environment.languages.default);
    }
  }
}
```

Рисунок А.1 – Клас AppComponent

```
1+ usages  Maxim Shepelyakovski
export function HttpLoaderFactory(http: HttpClient) : TranslateHttpLoader  {
  return new TranslateHttpLoader( http: http,  prefix: './assets/i18n/',  suffix: '.json');
}

1+ usages  Maxim Shepelyakovski
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    BrowserAnimationsModule,
    AppRoutingModule,
    TranslateModule.forRoot( config: {
      loader: {
        provide: TranslateLoader,
        useFactory: HttpLoaderFactory,
        deps: [HttpClient]
      }
    })
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {
}
```

Рисунок А.2 – Клас AppModule

```
1+ usages  Maxim Shepelyakovski
@Component({
  selector: 'app-about-us',
  standalone: true,
  imports: [
    TranslateModule
  ],
  templateUrl: './about-us.component.html',
  styleUrl: './about-us.component.scss'
})
export class AboutUsComponent {

  protected readonly close = close;
}
```

Рисунок А.3 – Клас AboutUsComponent

```
.about-us-container {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
  font-family: Arial, sans-serif;
  color: #333;
}

.about-us-container h1, .about-us-container h2 {
  color: #0056b3;
}

.about-us-container p {
  line-height: 1.6;
}

.about-us-container ul {
  list-style-type: none;
  padding: 0;
}

.about-us-container li {
  background: #f9f9f9;
  margin: 10px 0;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 5px;
}
```

Рисунок А.4 – Стилі сторінки “About us”

```html
<div class="about-us-container">
  <h1>{{ 'about_us.title' | translate }}</h1>
  <p>{{ 'about_us.welcome_message' | translate }}</p>
  <h2>{{ 'about_us.mission_title' | translate }}</h2>
  <p>{{ 'about_us.mission_description' | translate }}</p>
  <h2>{{ 'about_us.values_title' | translate }}</h2>
  <ul>
    <li><strong>{{ 'about_us.values.quality_title' | translate }}
      :</strong> {{ 'about_us.values.quality_description' | translate }}
    </li>
    <li><strong>{{ 'about_us.values.honesty_title' | translate }}
      :</strong> {{ 'about_us.values.honesty_description' | translate }}
    </li>
    <li><strong>{{ 'about_us.values.customer_focus_title' | translate }}
      :</strong> {{ 'about_us.values.customer_focus_description' | translate }}
    </li>
    <li><strong>{{ 'about_us.values.innovation_title' | translate }}
      :</strong> {{ 'about_us.values.innovation_description' | translate }}
    </li>
  </ul>
  <h2>{{ 'about_us.contact_title' | translate }}</h2>
  <p>{{ 'about_us.contact_description' | translate }}</p>
</div>
```

Рисунок А.5 – Код файлу about-us.component.html

```
1+ usages  Maxim Shepelyakovski
@Component({
  selector: 'app-contact',
  standalone: true,
  imports: [
    TranslateModule
  ],
  templateUrl: './contact.component.html',
  styleUrl: './contact.component.scss'
})
export class ContactComponent {

}
```

Рисунок А.6 – Клас ContactComponent

```scss
.contact-container {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
  font-family: Arial, sans-serif;
  color: #333;
}

.contact-container h1, .contact-container h2 {
  color: #0056b3;
}

.contact-container p {
  line-height: 1.6;
}
```

Рисунок А.7 – Стилі сторінки “Contact”

```
<div class="contact-container">
  <h1>{{ 'contact.title' | translate }}</h1>
  <p>{{ 'contact.description' | translate }}</p>

  <h2>{{ 'contact.address_title' | translate }}</h2>
  <p>{{ 'contact.address' | translate }}</p>

  <h2>{{ 'contact.phone_title' | translate }}</h2>
  <p>{{ 'contact.phone' | translate }}</p>

  <h2>{{ 'contact.email_title' | translate }}</h2>
  <p>{{ 'contact.email' | translate }}</p>

</div>
```

Рисунок А.8 – Код файлу contact.component.html

```
no usages  Maxim Shepelyakovski
ngOnInit(): void {
  this.createForms();
  this.subscribeToFormChanges();
  this.subscribeToSearchChanges();
  this.getCars();
}
```

Рисунок А.9 – Метод ngOnInit()

```
1+ usages  Maxim Shepelyakovski
createForms() : void {
  this.searchForm = new FormGroup( controls: {
    search: new FormControl( value: ''),
  });

  this.priceForm = new FormGroup( controls: {
    priceFrom: new FormControl( value: 0),
    priceTo: new FormControl( value: 0),
  });

  this.otherFiltersForm = new FormGroup( contro
    make: new FormControl( value: []),
    model: new FormControl( value: []),
    year: new FormControl( value: []),
    country: new FormControl( value: [])
  });
}
```

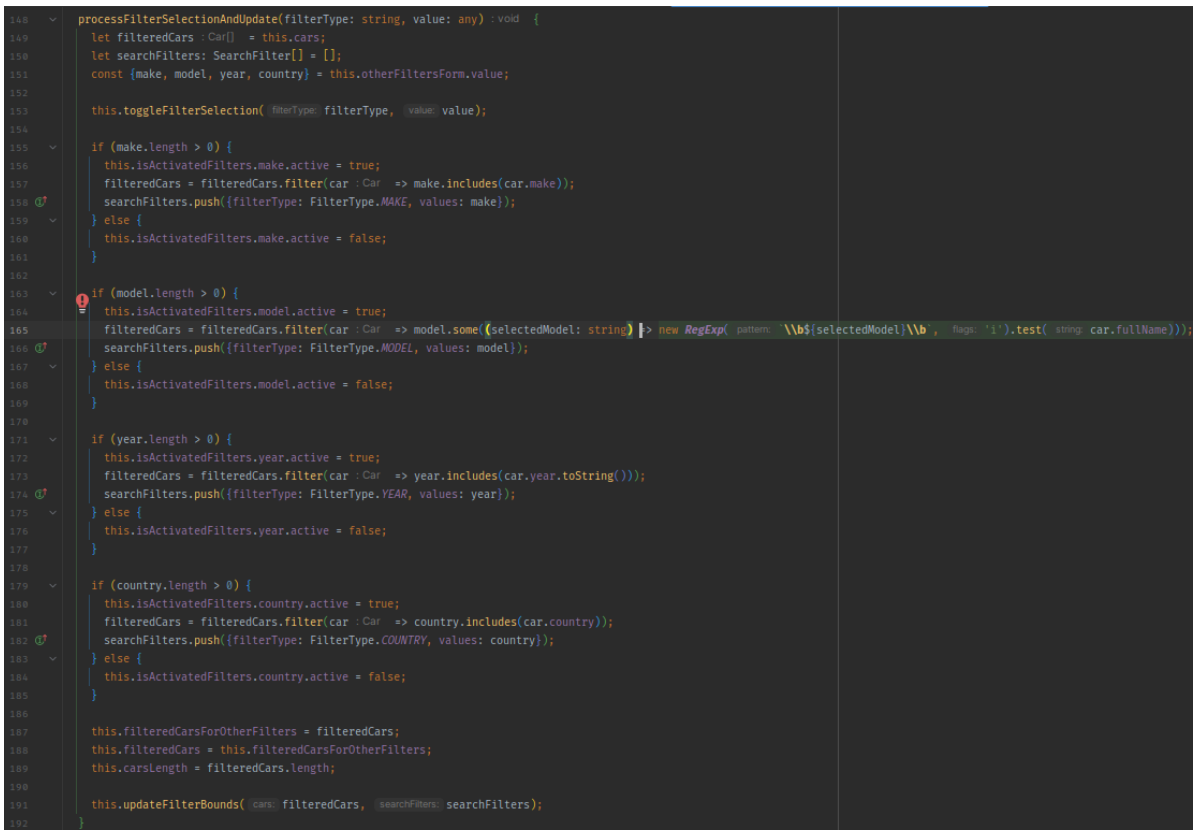
Рисунок А.10 – Метод createForms()

```
1+ usages  Maxim Shepelyakovski *
subscribeToFormChanges() : void {
  this.priceForm.valueChanges.subscribe( observerOrNext: value => {
    let filteredCars : Car[] = this.cars;
    this.filteredCarsForPriceFilter = this.filteredCarsForOtherFilters;

    const {priceFrom, priceTo} = value;
    if (priceFrom > 0 || priceTo > 0) {
      if (this.isActivatedFilters.make.active || this.isActivatedFilters.model.active
        || this.isActivatedFilters.year.active || this.isActivatedFilters.country.active) {
        this.filteredCarsForOtherFilters = this.filteredCarsForPriceFilter;
        filteredCars = this.filteredCarsForOtherFilters.filter(car : Car => car.price >= priceFrom && car.price <= priceTo);
      } else if (!this.isActivatedFilters.make.active && !this.isActivatedFilters.model.active
        && !this.isActivatedFilters.year.active && !this.isActivatedFilters.country.active) {
        filteredCars = filteredCars.filter(car : Car => car.price >= priceFrom && car.price <= priceTo);
        let searchFilters: SearchFilter[] = [{filterType: FilterType.PRICE, values: [priceFrom, priceTo]}];
        this.updateFiltersBasedOnPrice( searchFilters: searchFilters);
      }
    }
    this.filteredCarsForPriceFilter = filteredCars;
    this.filteredCars = this.filteredCarsForPriceFilter;
    this.carsLength = filteredCars.length;
  });
}
```

Рисунок А.11 – Функція subscribeToFormChanges()

```
1+ usages  Maxim Shepelyakovski
getCars(searchFilters: SearchFilter[] = []) : void {
  this.carService.filterCars( searchFilters: searchFilters).subscribe( observerOrNext: (res : Car[] ) : void => {
    this.cars = res;
    this.filteredCars = this.cars;
    this.filteredCarsForOtherFilters = this.cars;
    this.carsLength = this.cars.length;
    this.updateFilterBounds( cars: this.cars, searchFilters: []);
  });
}
```

Рисунок А.12 – Функція getCars()



```
processFilterSelectionAndUpdate(filterType: string, value: any) : void {
  let filteredCars : Car[] = this.cars;
  let searchFilters: SearchFilter[] = [];
  const {make, model, year, country} = this.otherFiltersForm.value;

  this.toggleFilterSelection( filterType: filterType, value: value);

  if (make.length > 0) {
    this.isActivatedFilters.make.active = true;
    filteredCars = filteredCars.filter(car : Car => make.includes(car.make));
    searchFilters.push({filterType: FilterType.MAKE, values: make});
  } else {
    this.isActivatedFilters.make.active = false;
  }

  if (model.length > 0) {
    this.isActivatedFilters.model.active = true;
    filteredCars = filteredCars.filter(car : Car => model.some((selectedModel: string) => new RegExp( pattern: `\\b${selectedModel}\\b`, flags: 'i').test( string: car.fullName)));
    searchFilters.push({filterType: FilterType.MODEL, values: model});
  } else {
    this.isActivatedFilters.model.active = false;
  }

  if (year.length > 0) {
    this.isActivatedFilters.year.active = true;
    filteredCars = filteredCars.filter(car : Car => year.includes(car.year.toString()));
    searchFilters.push({filterType: FilterType.YEAR, values: year});
  } else {
    this.isActivatedFilters.year.active = false;
  }

  if (country.length > 0) {
    this.isActivatedFilters.country.active = true;
    filteredCars = filteredCars.filter(car : Car => country.includes(car.country));
    searchFilters.push({filterType: FilterType.COUNTRY, values: country});
  } else {
    this.isActivatedFilters.country.active = false;
  }

  this.filteredCarsForOtherFilters = filteredCars;
  this.filteredCars = this.filteredCarsForOtherFilters;
  this.carsLength = filteredCars.length;

  this.updateFilterBounds( cars: filteredCars, searchFilters: searchFilters);
}
```

Рисунок А.13 – Функція processFilterSelectionAndUpdate()

Рисунок А.14 – Функція resetFilters ()

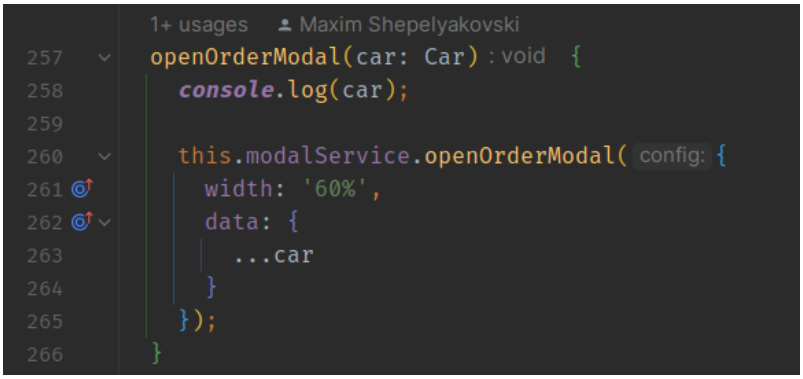


Рисунок А.15 – Функція openOrderModal()

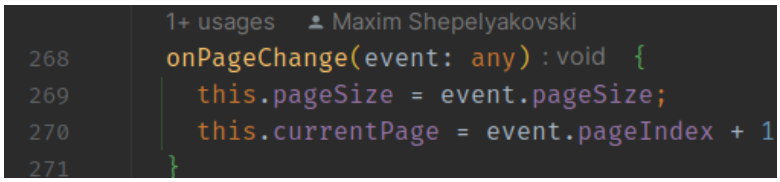


Рисунок А.16 – Функція onPageChange()

```scss
.car-dealership-container {
  display: flex;
  flex-direction: row;
  gap: 20px;

  .filters-panel {
    width: calc(15% - 15px);
    display: flex;
    flex-wrap: wrap;
    align-self: flex-start;
    box-shadow: 5px 0 5px -5px rgba(0, 0, 0, 0.5);
    padding-right: 15px;
    padding-top: 15px;
    padding-left: 20px;
    background: #f1f1f1;

    .button {
      width: 100%;
      margin-bottom: 20px;

      button {
        width: 100%;
      }
    }
```

Рисунок А.17 – Стилі сторінки “Catalog”

```scss
      form {
        width: 100%;

        .input {
          display: flex;
          gap: 10px;
          margin-top: 20px;
        }

        .checkbox {
          display: flex;
          align-items: center;
        }

        .logo {
          width: 24px;
          height: 24px;
          margin-right: 8px;
        }

        .toggle-filters {
          width: 100%;
          margin-bottom: 20px;
        }
      }
    }
```

Рисунок А.18 – Стилі сторінки “Catalog”

```scss
    .cars-container {
      width: 85%;
      display: flex;
      flex-wrap: wrap;
      align-content: flex-start;
      column-gap: 2%;

      .card {
        width: 18%;
      }

      .pagination {
        width: 100%;
        display: flex;
        justify-content: end;
      }
    }

    .filters-title {
      font-size: 18px;
      font-weight: bold;
    }
  }
```

Рисунок А.19 – Стилі сторінки “Catalog”

```
<div class="car-dealership-container">
  <div class="filters-panel">
    <!-- reset filters -->
    <div class="button">
      <button mat-raised-button color="primary"
              (click)="resetFilters()"> {{ 'pages.car_dealership.reset' | translate }}
      </button>
    </div>
    <!-- price filter -->
    <form [formGroup]="priceForm">
      <div class="input">
        <mat-form-field appearance="outline">
          <mat-label translate> pages.car_dealership.filters.price_from</mat-label>
          <input matInput formControlName="priceFrom">
        </mat-form-field>
        <mat-form-field appearance="outline">
          <mat-label translate> pages.car_dealership.filters.price_to</mat-label>
          <input matInput formControlName="priceTo">
        </mat-form-field>
      </div>
    </form>
```

Рисунок А.20 – Html код сторінки “Catalog”

```
    <!-- make, model, year, country filters -->
    <form [formGroup]="otherFiltersForm">
      <ng-container *ngFor="let filterType of filterTypes">
        <ng-container *ngIf="getFiltersByType( filterType: filterType).length > 0">
          <div class="filters-title">{{ 'pages.car_dealership.filters.' + filterType | translate }}</div>
          <div class="checkbox" *ngFor="let filter of getFiltersByType( filterType: filterType)">
            <mat-checkbox
              (click)="processFilterSelectionAndUpdate( filterType: filterType, value: filter.key)"
              [checked]="otherFiltersForm.value[filterType].includes( key: filter.key)">
            </mat-checkbox>
            <img *ngIf="filterType === 'make'"
                 [src]="getLogoPath( additionalInformation: filter.additionalInformation, key: filter.key)" class="logo" alt="{{filter.key}}">
            <img *ngIf="filterType === 'country'"
                 [src]="getFlagPath( country: String( value: filter.key))" class="logo" alt="{{filter.key}}">
            <label>{{ filter.key }} ({{ filter.size }})</label>
          </div>
          <button mat-raised-button color="primary" class="toggle-filters"
                  *ngIf="hasMoreThanTenItems( filterType: filterType)"
                  (click)="toggleFiltersView( filterType: filterType)">
            {{ isFiltersExpanded[filterType] ? ('pages.car_dealership.show_less' | translate) : ('pages.car_dealership.show_more' | translate) }}
          </button>
        </ng-container>
      </ng-container>
    </form>
  </div>
```

Рисунок А.21 – Html код сторінки “Catalog”

```
  <div class="cars-container">
    <ng-container *ngFor="let car of filteredCars | slice: start: (currentPage - 1) * pageSize : end: currentPage * pageSize">
      <div class="card">
        <app-car-card
          [car]="car"
          (click)="openOrderModal( car: car)"
        ></app-car-card>
      </div>
    </ng-container>
    <div class="pagination">
      <mat-paginator
        [length]="carsLength"
        [pageSize]="pageSize"
        [pageSizeOptions]="pageSizeOptions"
        (page)="onPageChange( event: $event)"
      >
      </mat-paginator>
    </div>
  </div>
</div>
```

Рисунок А.22 – Html код сторінки “Catalog”

```ts
import {Component, Input} from '@angular/core';
import {Car} from '@interfaces/car.interface';

1+ usages  Maxim Shepelyakovski
@Component({
  selector: 'app-car-card',
  templateUrl: './car-card.component.html',
  styleUrls: ['./car-card.component.scss']
})
export class CarCardComponent {
  @Input() car!: Car;
  imageExists: boolean = true;

  no usages  Maxim Shepelyakovski
  constructor() {
  }
}
```

Рисунок А.24 – Клас CarCardComponent

```html
<div class="car-tile">
  <img *ngIf="imageExists; else defaultCarImage"
       [src]="car.pathToPhoto"
       alt="Car Photo"
       class="car-photo"
       loading="lazy"
       (error)="imageExists = false">
  Maxim Shepelyakovski
  <ng-template #defaultCarImage>
    <img src="assets/car/noPhoto.svg" alt="Default Car Image" class="car-photo">
  </ng-template>
  <div class="car-info">
    <div class="car-info-header">
      <img [src]="'assets/make/' + car.country + '/' + car.make + '/' + car.make  + '.svg'" alt="{{car.make }}"
           class="make-logo">
      <h3>{{ car.fullName }}</h3>
    </div>
    <p> Price: {{ car.price | currency: currencyCode: 'USD' }} </p>
  </div>
</div>
```

Рисунок А.25 – HTML код компоненту “Car Card”

```scss
.car-tile {
  cursor: pointer;
  border: 1px solid #ddd;
  border-radius: 5px;
  overflow: hidden;
  display: flex;
  flex-direction: column;
  gap: 10px;
  max-height: 700px;
  margin-bottom: 40px;

  .car-info {
    padding: 10px;
    min-height: 30px;
  }
}

.car-photo {
  width: 100%;
  height: auto;
  object-fit: cover;
  max-height: 100%;
}
```

Рисунок А.26 – Стилі компоненту “Car Card”

```css
.car-info-header {
  display: flex;
  align-items: center;
}

.make-logo {
  width: 50px;
  height: auto;
  margin-right: 0.5em;
}

.car-info h3 {
  margin: 0;
  font-size: 1em;
}
```

Рисунок А.27 – Стилі компоненту “Car Card”

```
1+ usages  Maxim Shepelyakovski *
makeOrder() : void {

  const user: User = {
    firstName: this.form.get('firstName')!.value,
    lastName: this.form.get('lastName')!.value,
    phoneNumber: this.form.get('phoneNumber')!.value,
    email: this.form.get('email')!.value,
    country: this.form.get('country')!.value,
    city: this.form.get('city')!.value
  };

  const request: Request = {
    user: user,
    car: this.car,
    message: this.form.get('message')!.value,
    language: this.translateService.currentLang
  }

  this.orderService.sentOrder( request: request).subscribe( observerOrNext: {
    next: (response) : void => {
      console.log('Success', response);
    },
    error: (error) : void => {
      console.error('Error', error);
    },
    complete: () : void => {
      console.log('Completed');
    }
  });

  this.closeOpenModal();
}
```

Рисунок А.28 – Метод makeOrder() класу OrderModal

```
1+ usages  Maxim Shepelyakovski
saveCar(car: Car): Observable<any> {
  return this.http.post( url: this.ADD_CAR_API_URL,  body: car);
}
```

Рисунок А.29 – Метод saveCar() класу CarService

```
1+ usages   Maxim Shepelyakovski
getExistsMakes(): Observable<string[]> {
  return this.http.get<string[]>( url: this.MAKE_API_URL);
}
```

Рисунок А.30 – Метод getExistsMakes() класу CarService

```
1+ usages   Maxim Shepelyakovski
getAllCars(): Observable<Car[]> {
  return this.http.get<Car[]>( url: this.ALL_CARS_API_URL).pipe(
    op1: map( project: cars : Car[]  => cars.map((car : Car , index : number ) : {...}  => ({...car, id: index})))
  );
}
```

Рисунок А.31 – Метод getAllCars() класу CarService

```
1+ usages   Maxim Shepelyakovski
getCarsByMake(make: string): Observable<Car[]> {
  return this.http.get<Car[]>( url: `${this.CARS_BY_MAKE_API_URL}/${make}`).pipe(
    op1: map( project: cars : Car[]  => cars.map((car : Car , index : number ) : {...}  => ({...car, id: index})))
  );
}
```

Рисунок А.32 – Метод getCarsByMake() класу CarService

```
1+ usages   Maxim Shepelyakovski
updateCar(car: Car): Observable<string> {
  return this.http.patch( url: this.UPDATE_CARS_API_URL,  body: car,  options: {responseType: 'text'});
}
```

Рисунок А.33 – Метод updateCar() класу CarService

```
1+ usages   Maxim Shepelyakovski
deleteCar(fullName: string): Observable<any> {
  return this.http.delete( url: `${this.DELETE_CARS_API_URL}/${fullName}`,  options: {responseType: 'text'})
}
```

Рисунок А.34 – Метод deleteCar() класу CarService

```
1+ usages   Maxim Shepelyakovski
filterCars(searchFilters: SearchFilter[]): Observable<Car[]> {
  return this.http.post<Car[]>( url: this.FILTER_CARS_API_URL,  body: searchFilters).pipe(
    op1: map( project: cars : Car[]  => cars.map((car : Car , index : number ) : {...}  => ({...car, id: index})))
  );
}
```

Рисунок А.35 – Метод filterCars() класу CarService

```
58
    1+ usages   Maxim Shepelyakovski
59  getFilterWithValues(searchFilters: SearchFilter[]): Observable<FilterCriteria[]> {
60    return this.http.post<FilterCriteria[]>( url: this.FILTER_OPTIONS_GET_CARS_API_URL,  body: searchFilters);
61  }
```

Рисунок А.36 – Метод getFilterWithValues() класу CarService

```
getMakes(): Observable<Make[]> {
  return this.http.get<Make[]>( url: this.MAKE_ALL_API_URL);
}
```

Рисунок А.37 – Метод getMakes() класу MakeService

```
1+ usages   Maxim Shepelyakovski
openOrderModal(config: MatDialogConfig) : void  {
  this.dialog.open( component: OrderModalComponent,  config: config);
}

1+ usages   Maxim Shepelyakovski *
openThankYouModal() : void  {
  this.dialog.open( component: ThankYouModalComponent,  config: {
    width: '20%',
    disableClose: true
  });
}
```

Рисунок А.38 – Методи openOrderModal() та openThankYouModal() класу ModalService

```
1+ usages   Maxim Shepelyakovski
sentOrder(request: Request): Observable<any> {
  return this.http.post( url: this.SENT_ORDER_API_URL,  body: request);
}
```

Рисунок А.39 – Метод sentOrder() класу OrderService

```
1  plugins {
2      id 'java'
3      id 'org.springframework.boot' version '3.2.4'
4      id 'io.spring.dependency-management' version '1.1.4'
5  }
```

Рисунок А.40 – Плагіни додатку

```
dependencies { Edit Starters...
    implementation 'org.springframework.boot:spring-boot-starter'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'
    implementation 'org.springframework.boot:spring-boot-starter-mail'

    // Querydsl
    implementation "com.querydsl:querydsl-core:${ver.queryDslVersion}"
    implementation("com.querydsl:querydsl-mongodb:${ver.queryDslVersion}") {
        exclude group: "org.mongodb", module: "mongo-java-driver"
    }

    compileOnly 'org.projectlombok:lombok'
    implementation 'org.mapstruct:mapstruct:1.5.5.Final'

    annotationProcessor(
            "com.querydsl:querydsl-apt:${ver.queryDslVersion}:general",
            'org.projectlombok:lombok',
            'javax.annotation:javax.annotation-api:1.3.2',
            'org.mapstruct:mapstruct-processor:1.5.5.Final'
    )

    implementation 'org.apache.commons:commons-collections4:4.4'
    implementation group: 'ch.qos.logback', name: 'logback-classic', version: '1.4.12'

    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

Рисунок А.41 – Залежності проєкту

```
repositories {
    mavenLocal()
    mavenCentral()
}
```

Рисунок А.42 – Репозиторії

```
spring:
  servlet:
    multipart:
      resolve-lazily: true
      max-file-size: 5MB
      max-request-size: 5MB
  data:
    mongodb:
      uri: mongodb://maxim:TcunkZ_%406sKM-92~9_G7@localhost:27017/autodealership?authSource=admin
  mail:
    host: smtp.gmail.com
    port: 587
    username: maxim.shepelyakovski@gmail.com
    password:
    properties:
      mail:
        smtp:
          auth: true
          starttls:
            enable: true


photos:
  uri:
    car: assets/car
    make: assets/make

logging:
  level:
    org.springframework.data.mongodb.core.MongoTemplate: DEBUG
```

Рисунок А.43 – Файл application.yml

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Document(collection = "cars")
@QueryEntity
public class Car {
    @MongoId
    private String id;
    @Field
    private String fullName;
    @Field
    private String make;
    @Field
    private String model;
    @Field
    private List<String> bodyStyle;
    @Field
    private Long price;
    @Field
    private Integer year;
    @Field
    private Boolean enabled;
}
```

Рисунок А.44 – Клас “Car”

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Document(collection = "filters")
@QueryEntity
public class Filter {
    @MongoId
    private String id;
    @Field
    FilterType filterType;
    @Field
    List<String> values;
}
```

Рисунок А.45 – Клас “Filter”

```java
16 usages  Maxim Shepelyakovski
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Document(collection = "make")
@QueryEntity
public class Make {
    @MongoId
    private String id;
    private String make;
    private String country;
}
```

Рисунок А.46 – Клас “Make”

```
11 usages  Maxim Shepelyakovski
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Document(collection = "requests")
@QueryEntity
public class Request {
    @MongoId
    private String id;
    @DBRef
    private User user;
    @DBRef
    private Car car;
    @Field
    private String message;
    @Field
    private String language;
}
```

Рисунок А.47 – Клас “Request”

```
9 usages  Maxim Shepelyakovski *
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Document(collection = "users")
@QueryEntity
public class User {
    @MongoId
    private String id;
    @Field
    private String firstName;
    @Field
    private String lastName;
    @Field
    private String phoneNumber;
    @Field
    private String email;
    @Field
    private String country;
    @Field
    private String city;
}
```

Рисунок А.48 – Клас “User”

```
23 usages   Maxim Shepelyakovski
@Getter
@RequiredArgsConstructor
public enum FilterType {
    MAKE( displayName: "Make"),
    MODEL( displayName: "Model"),
    YEAR( displayName: "Year"),
    PRICE( displayName: "Price"),
    COUNTRY( displayName: "Country");

    private final String displayName;

    Maxim Shepelyakovski
    @Contract(pure = true)
    @Override
    public String toString() { return displayName; }
}
```

Рисунок А.49 – Enum “FilterType”

```
3 usages   Maxim Shepelyakovski
@Component
@Mapper(uses = {CustomConverters.class})
public abstract class CarMapper {

    1 usage
    private CustomConverters customConverters;

    2 usages   Maxim Shepelyakovski
    @Mapping(target = "id", ignore = true)
    @Mapping(target = "enabled", ignore = true)
    @Mapping(target = "fullName", expression = "java(CustomConverters.fullNameConverter(carDTO.getFullName(),carDTO.getMake(), carDTO.getModel(), carDTO.getYear()))")
    public abstract Car mapCarDTOToCar(CarDTO carDTO);

    2 usages   Maxim Shepelyakovski
    @Mapping(target = "country", ignore = true)
    @Mapping(target = "pathToPhoto", ignore = true)
    @Mapping(target = "fullName", expression = "java(CustomConverters.fullNameConverter(car.getFullName(), car.getMake(), car.getModel(), car.getYear()))")
    public abstract CarDTO mapCarToCarDTO(Car car);

    1 usage   Maxim Shepelyakovski
    @AfterMapping
    protected void initializePathToPhoto( @NonNull @MappingTarget CarDTO.CarDTOBuilder carDTO,  @NonNull Car car) {
        carDTO.pathToPhoto(customConverters.pathToPhotoConverter(car.getMake(), car.getModel(), car.getYear()));
    }
}
```

Рисунок А.50 – Мапер “CarMapper”

```
3 usages   Maxim Shepelyakovski
@Mapper
public interface FilterMapper {

    1 usage   Maxim Shepelyakovski
    @Mapping(target = "id", ignore = true)
    @Mapping(target = "values", ignore = true)
    Filter mapFilterDTOToFilter(FilterDTO filterDTO);

    1 usage   Maxim Shepelyakovski
    FilterDTO mapFilterToFilterDTO(Filter filter);

}
```

Рисунок А.51 – Мапер “FilterMapper”

```
3 usages  Maxim Shepelyakovski
@Mapper
public interface MakeMapper {

    1 usage  Maxim Shepelyakovski
    @Mapping(target = "id", ignore = true)
    Make mapMakeDTOToMake(MakeDTO makeDTO);

    1 usage  Maxim Shepelyakovski
    MakeDTO mapMakeToMakeDTO(Make make);

}
```

Рисунок А.52 – Мапер “MakeMapper”

```
3 usages  Maxim Shepelyakovski
@Mapper
public interface RequestMapper {

    1 usage  Maxim Shepelyakovski
    @Mapping(target = "id", ignore = true)
    @Mapping(target = "user", ignore = true)
    @Mapping(target = "car", ignore = true)
    Request mapRequestDTOToRequest(RequestDTO requestDTO);

    no usages  Maxim Shepelyakovski
    @Mapping(target = "car.pathToPhoto", ignore = true)
    @Mapping(target = "car.country", ignore = true)
    RequestDTO mapRequestToRequestDTO(Request request);

}
```

Рисунок А.53 – Мапер “RequestMapper”

```
3 usages  Maxim Shepelyakovski
@Mapper
public interface UserMapper {

    1 usage  Maxim Shepelyakovski
    @Mapping(target = "id", ignore = true)
    User mapUserDTOToUser(UserDTO userDTO);

    no usages  Maxim Shepelyakovski
    UserDTO mapUserToUserDTO(User user);

}
```

Рисунок А.54 – Мапер “UserMapper”

```java
@Component
public class CarUtils {

    @Value("assets/car")
    private String baseCarPhotoPath;
    @Value("assets/make")
    private String baseMakePhotoPath;

    3 usages  Maxim Shepelyakovski
    public String getPathToPhoto(String make, String model, Integer year) {
        return String.format("../%s/%s/%s/%d/", baseCarPhotoPath, make, model, year);
    }

    1 usage  Maxim Shepelyakovski
    public String getPathToPhotoCar(String make, String model, Integer year) {
        return String.format("%s/%s/%s/%d/%s.avif", baseCarPhotoPath, make, model, year, make + model + "_" + year);
    }

    1 usage  Maxim Shepelyakovski
    public String getPathToPhotoForLogo(String make, String country) {
        return String.format("%s/%s/%s/", baseMakePhotoPath, country, make);
    }

}
```

Рисунок А.55 – Утіліта “CarUtils”

```java
@Component
public class MakeUtils {

    @Value("assets/make")
    private String baseMakePhotoPath;

    1 usage  Maxim Shepelyakovski
    public String getPathToPhoto(String make, String country) {
        return String.format("../%s/%s/%s/", baseMakePhotoPath, country, make);
    }
}
```

Рисунок А.56 – Утіліта “MakeUtils”

```java
Maxim Shepelyakovski
@GetMapping(value = "/all")
public List<CarDTO> getAllCar() { return carService.findAllCars(); }
```

Рисунок А.57 – Ендпоінт "/all" контролера CarController

```java
Maxim Shepelyakovski
@GetMapping(value = "/make/{make}")
public List<CarDTO> getAllCarByMake(@PathVariable String make) { return carService.findAllModelCarsByMake(make); }
```

Рисунок А.58 – Ендпоінт "/make/{make}" контролера CarController

```java
Maxim Shepelyakovski
@GetMapping(value = "/make")
public List<String> getExistsMakes() { return carService.findExistsMakes(); }
```

Рисунок А.59 – Ендпоінт "/make" контролера CarController

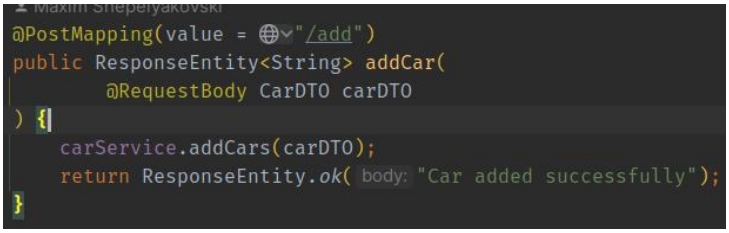


Рисунок А.60 – Ендпоінт "/add" контролера CarController

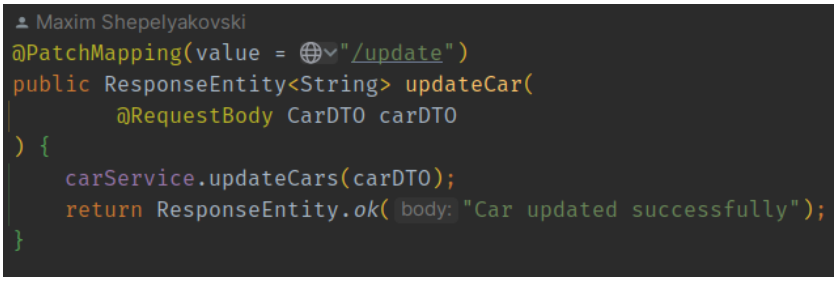
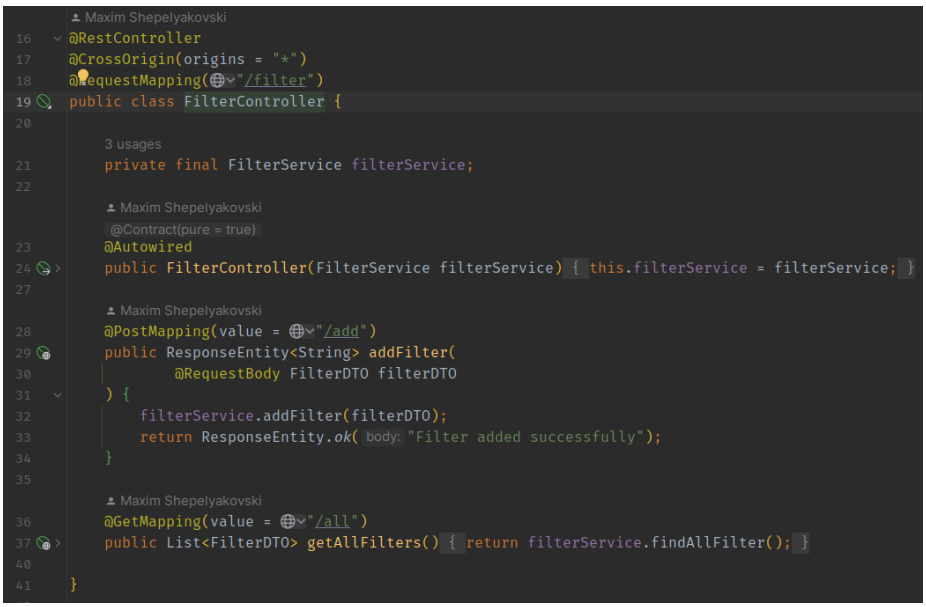


Рисунок А.61 – Ендпоінт "/update" контролера CarController

Рисунок А.62 – Ендпоінт "/delete/{fullName}" контролера CarController

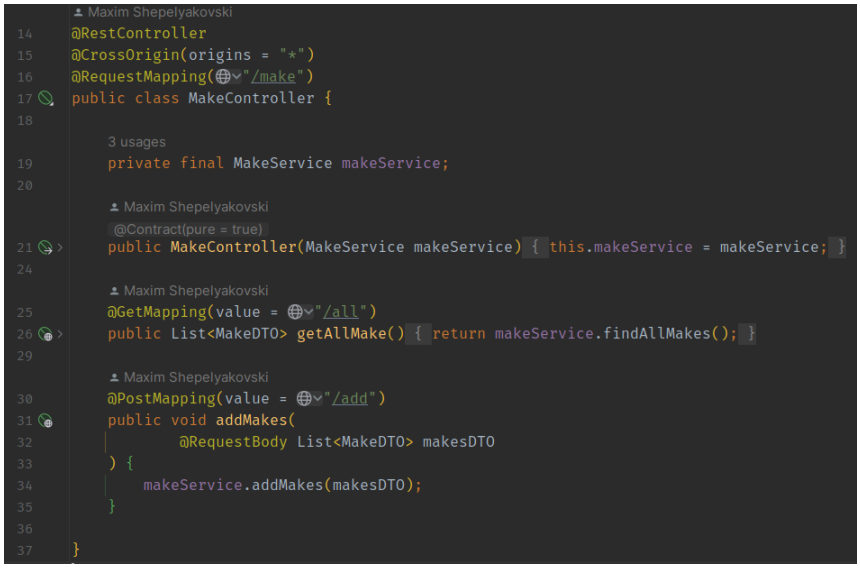
Рисунок А.63 – Ендпоінт "/filter/search" контролера CarController

Рисунок А.64 – Ендпоінт "/filter/options/get" контролера CarController



```
Maxim Shepelyakovski
@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/filter")
public class FilterController {

    3 usages
    private final FilterService filterService;

    Maxim Shepelyakovski
    @Contract(pure = true)
    @Autowired
    public FilterController(FilterService filterService) { this.filterService = filterService; }

    Maxim Shepelyakovski
    @PostMapping(value = "/add")
    public ResponseEntity<String> addFilter(
            @RequestBody FilterDTO filterDTO
    ) {
        filterService.addFilter(filterDTO);
        return ResponseEntity.ok( body: "Filter added successfully");
    }

    Maxim Shepelyakovski
    @GetMapping(value = "/all")
    public List<FilterDTO> getAllFilters() { return filterService.findAllFilter(); }

}
```

Рисунок А.65 – Контролер “FilterController”



```
Maxim Shepelyakovski
@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/make")
public class MakeController {

    3 usages
    private final MakeService makeService;

    Maxim Shepelyakovski
    @Contract(pure = true)
    public MakeController(MakeService makeService) { this.makeService = makeService; }

    Maxim Shepelyakovski
    @GetMapping(value = "/all")
    public List<MakeDTO> getAllMake() { return makeService.findAllMakes(); }

    Maxim Shepelyakovski
    @PostMapping(value = "/add")
    public void addMakes(
            @RequestBody List<MakeDTO> makesDTO
    ) {
        makeService.addMakes(makesDTO);
    }

}
```

Рисунок А.66 – Контролер “MakeController”

```
Maxim Shepelyakovski
@RestController
@RequiredArgsConstructor
@CrossOrigin(origins = "*")
@RequestMapping("/order")
@Slf4j
public class OrderController {

    private final OrderService orderService;

    Maxim Shepelyakovski
    @PostMapping(value = "/sent")
    public ResponseEntity<String> makeOrder(@RequestBody RequestDTO request) throws MessagingException, IOException {
        orderService.makeOrder(request);
        return ResponseEntity.ok().build();
    }
}
```

Рисунок А.67 – Контролер “OrderController”

Рисунок А.68 – Метод addCars() сервісу CarService

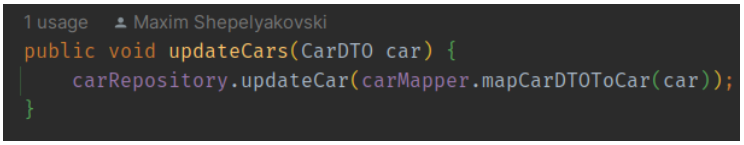


Рисунок А.69 – Метод updateCars() сервісу CarService

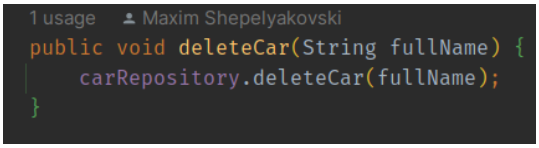


Рисунок А.70 – Метод deleteCar() сервісу CarService

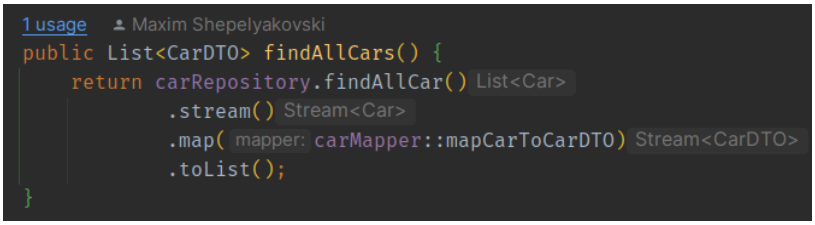


Рисунок А.71 – Метод findAllCars() сервісу CarService

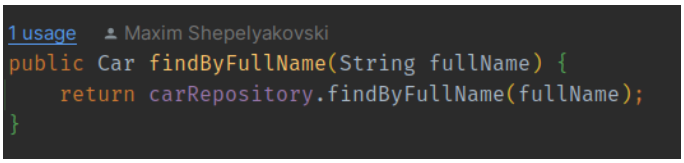


Рисунок А.72 – Метод findByFullName() сервісу CarService

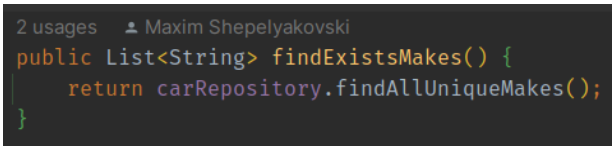


Рисунок А.73 – Метод findExistsMakes() сервісу CarService

```
1 usage   Maxim Shepelyakovski
public List<CarDTO> findAllCarsByFilter(List<SearchFilter> filters) {
    List<String> makes = findExistsMakes();
    return carRepository.findAllCarsByFilter(filters, makes)
            .stream().peek( action: carDTO -> carDTO.setPathToPhoto(
                    carUtils.getPathToPhotoCar(carDTO.getMake(), carDTO.getModel(), carDTO.getYear())
            )).toList();
}
```

Рисунок А.74 – Метод findAllCarsByFilter() сервісу CarService

```
1 usage   Maxim Shepelyakovski
public List<FilterCriteria> getFilterOptionsBySearchFilter(List<SearchFilter> filters) {
    return List.of(
            e1: filterService.getAllMakeValues(filters),
            e2: filterService.findAllModelValues(filters),
            e3: filterService.findAllYearValues(filters),
            e4: filterService.findAllPriceValues(filters),
            e5: filterService.findAllCountryValues(filters)
    );
}
```

Рисунок А.75 – Метод getFilterOptionsBySearchFilter() сервісу CarService

```
1 usage   Maxim Shepelyakovski
public void sendMessage( @NonNull Request request) throws MessagingException, IOException {
    MimeMessage message = mailSender.createMimeMessage();
    MimeMessageHelper helper = new MimeMessageHelper(message,  multipart: true);

    helper.setTo(request.getUser().getEmail());
    helper.setSubject("Order Confirmation");
    helper.setText(buildEmailText(request),  html: true);

    mailSender.send(message);
}
```

Рисунок А.76 – Метод sendMessage() сервісу EmailService

```
1 usage   Maxim Shepelyakovski
private String buildEmailText( @NonNull Request request) throws IOException {
    String lang = request.getLanguage();
    if (!AVAILABLE_LANGUAGES.contains(lang)) {
        lang = "en";
    }

    Map<String, String> messages = loadMessages(lang);
    String template = messages.get("emailTemplate");

    return String.format(template,
            request.getUser().getFirstName(),
            request.getUser().getLastName(),
            request.getCar().getFullName(),
            request.getUser().getPhoneNumber());
}
```

Рисунок А.77 – Метод buildEmailText() сервісу EmailService

```
1 usage  Maxim Shepelyakovski
private Map<String, String> loadMessages(String lang) throws IOException {
    ClassPathResource resource = new ClassPathResource( path: String.format("messages/%s.json", lang));
    ObjectMapper objectMapper = new ObjectMapper();
    return objectMapper.readValue( src: resource.getInputStream(), valueType: Map.class);
}
```

Рисунок А.78 – Метод loadMessages() сервісу EmailService

```
1 usage  Maxim Shepelyakovski
public void addFilter(FilterDTO filterDTO) {
    Filter filter = filterMapper.mapFilterDTOToFilter(filterDTO);
    filterRepository.addFilter(filter);
}
```

Рисунок А.79 – Метод addFilter() сервісу FilterService

```
1 usage  Maxim Shepelyakovski
public List<FilterDTO> findAllFilter() {
    List<Filter> filters = filterRepository.findAll();
    return filters.stream() Stream<Filter>
            .map( mapper: filterMapper::mapFilterToFilterDTO) Stream<FilterDTO>
            .toList();
}
```

Рисунок А.80 – Метод findAllFilter() сервісу FilterService

```
1 usage  Maxim Shepelyakovski
public FilterCriteria getAllMakeValues(List<SearchFilter> filters) {
    return FilterCriteria.builder()
            .filterType( filterType: FilterType.MAKE)
            .values( values: carRepository.findExistsMakes(filters))
            .build();
}
```

Рисунок А.81 – Метод getAllMakeValues() сервісу FilterService

```
1 usage  Maxim Shepelyakovski
public FilterCriteria findAllModelValues(List<SearchFilter> filters)
    return FilterCriteria.builder()
            .filterType( filterType: FilterType.MODEL)
            .values( values: carRepository.findExistsModels(filters))
            .build();
}
```

Рисунок А.82 – Метод findAllModelValues() сервісу FilterService

```
1 usage  Maxim Shepelyakovski
public FilterCriteria findAllYearValues(List<SearchFilter> filters) {
    return FilterCriteria.builder()
            .filterType( filterType: FilterType.YEAR)
            .values( values: carRepository.findExistsYears(filters))
            .build();
}
```

Рисунок А.83 – Метод findAllYearValues() сервісу FilterService

```
1 usage   Maxim Shepelyakovski
public FilterCriteria findAllPriceValues(List<SearchFilter> filters)
    return FilterCriteria.builder()
            .filterType( filterType: FilterType.PRICE)
            .values( values: carRepository.findMinMaxPrices(filters))
            .build();
}
```

Рисунок А.84 – Метод findAllPriceValues() сервісу FilterService

```
1 usage   Maxim Shepelyakovski
public FilterCriteria findAllCountryValues(List<SearchFilter> filters) {
    return FilterCriteria.builder()
            .filterType( filterType: FilterType.COUNTRY)
            .values( values: makeRepository.findUniqueCountries( values: filters))
            .build();
}
```

Рисунок А.85 – Метод findAllCountryValues() сервісу FilterService

```
3 usages   Maxim Shepelyakovski
@Service
@RequiredArgsConstructor
public class MakeService {

    private final MakeRepository makeRepository;
    private final MakeMapper makeMapper;
    private final MakeUtils makeUtils;

    1 usage   Maxim Shepelyakovski
    public List<MakeDTO> findAllMakes() {
        List<Make> makes = makeRepository.findAllMake();
        return makes.stream() Stream<Make>
                .map( mapper: makeMapper::mapMakeToMakeDTO) Stream<MakeDTO>
                .toList();
    }

    1 usage   Maxim Shepelyakovski
    public void addMakes( @NonNull List<MakeDTO> makesDTO) {
        List<Make> makes = makesDTO.stream() Stream<MakeDTO>
                .map( mapper: makeMapper::mapMakeDTOToMake) Stream<Make>
                .toList();

        makes.forEach( action: make -> {
            String photoPath = makeUtils.getPathToPhoto(make.getMake(), make.getCountry());
            createDirectories(photoPath);
        });

        makeRepository.addMakes(makes);
    }

}
```

Рисунок А.86 – Сервіс MakeService

```java
2 usages  Maxim Shepelyakovski
@Service
@RequiredArgsConstructor
public class OrderService {

    private final UserService userService;
    private final CarService carService;
    private final EmailService emailService;
    private final RequestMapper requestMapper;
    private final RequestRepository requestRepository;

    1 usage  Maxim Shepelyakovski
    public void makeOrder(RequestDTO requestDTO) throws MessagingException, IOException {
        Request request = requestMapper.mapRequestDTOToRequest(requestDTO);

        request.setUser(userService.saveUser(requestDTO.getUser()));
        request.setCar(carService.findByFullName(requestDTO.getCar().getFullName()));

        requestRepository.save( entity: request);

        emailService.sendMessage(request);
    }
}
```

Рисунок А.87 – Сервіс OrderService

```java
1 usage  Maxim Shepelyakovski
@Service
@RequiredArgsConstructor
public class UserService {

    private final UserMapper userMapper;
    private final UserRepository userRepository;

    1 usage  Maxim Shepelyakovski
    public User saveUser(UserDTO user) {
        return userRepository.save( entity: userMapper.mapUserDTOToUser(user));
    }
}
```

Рисунок А.88 – Сервіс UserService

```java
6 usages  Maxim Shepelyakovski *
public interface CarRepository extends MongoRepository<Car, String>, QuerydslPredicateExecutor<Car>, CarRepositoryCustom {
    1 usage  Maxim Shepelyakovski
    Car findByFullName(String fullName);
}

2 usages  2 implementations  Maxim Shepelyakovski *
interface CarRepositoryCustom {
    1 usage  1 implementation  Maxim Shepelyakovski
    List<Car> findAllCar();
    1 usage  1 implementation  Maxim Shepelyakovski
    void deleteCar(String fullname);
    1 usage  1 implementation  Maxim Shepelyakovski
    void addCar(Car car);
    1 usage  1 implementation  Maxim Shepelyakovski
    void updateCar(Car car);
    1 usage  1 implementation  Maxim Shepelyakovski
    List<Car> findAllModelsByMake(String make);
    1 usage  1 implementation  Maxim Shepelyakovski
    List<FilterOption> findExistsMakes(List<SearchFilter> filters);
    6 usages  1 implementation  Maxim Shepelyakovski
    List<String> findAllUniqueMakes();
    1 usage  1 implementation  Maxim Shepelyakovski
    List<CarDTO> findAllCarsByFilter(List<SearchFilter> filters, List<String> makes);
    1 usage  1 implementation  Maxim Shepelyakovski
    List<FilterOption> findExistsModels(List<SearchFilter> filters);
    1 usage  1 implementation  Maxim Shepelyakovski
    List<FilterOption> findExistsYears(List<SearchFilter> filters);
    1 usage  1 implementation  Maxim Shepelyakovski
    List<FilterOption> findMinMaxPrices(List<SearchFilter> filters);
    1 usage  1 implementation  Maxim Shepelyakovski
    List<Object> getAllCars();
}
```

Рисунок А.89 – Інтерфейс “CarRepository”

```
1 usage  Maxim Shepelyakovski
@Override
public List<Car> findAllCar() {
    return from( path: qCar)
            .where( e: qCar.enabled.isTrue())
            .fetch();
}

1 usage  Maxim Shepelyakovski
@Override
public void deleteCar(String fullname) {
    mongoOperations.remove(
            query( criteriaDefinition: Criteria.where( key: "fullName").is( value: fullname)),
            entityClass: Car.class
    );
}

1 usage  Maxim Shepelyakovski
@Override
public void addCar(Car car) {
    mongoOperations.insert( objectToSave: car);
}
```

Рисунок А.90 – Методи findAllCar(), deleteCar(), addCar() репозиторію CarRepository

```
1 usage  Maxim Shepelyakovski *
@Override
public void updateCar( @NonNull Car car) {
    Query query = new Query( criteriaDefinition: Criteria.where( key: "fullName").is( value: car.getFullName()));
    Update update = new Update();
    update.set("price", car.getPrice());
    mongoOperations.updateFirst(query, update, entityClass: Car.class);
}

1 usage  Maxim Shepelyakovski
@Override
public List<Car> findAllModelsByMake(String make) {
    Query query = new Query( criteriaDefinition: Criteria.where( key: "make").is( value: make).and( key: "enabled").is( value: true));
    return mongoOperations.find(query, entityClass: Car.class);
}
```

Рисунок А.91 – Методи updateCar(), findAllModelsByMake() репозиторію CarRepository

```
1 usage  Maxim Shepelyakovski
@Override
public List<FilterOption> findExistsMakes(List<SearchFilter> filters) {
    List<String> makes = findAllUniqueMakes();
    Criteria criteria = getCriteriaByFilters( filterType: FilterType.MAKE, filters, makes);

    Aggregation aggregation = Aggregation.newAggregation(
            ...operations: Aggregation.lookup( from: "make", localField: "make", foreignField: "make", as: "makesInfo"),
            Aggregation.unwind( field: "makesInfo"),
            Aggregation.match(criteria),
            Aggregation.group( ...fields: "make")
                    .count().as( alias: "size")
                    .first( reference: "makesInfo.country").as( alias: "additionalInformation"),
            Aggregation.project( ...fields: "size", "additionalInformation")
                    .and( name: "_id").as( alias: "key"),
            Aggregation.sort( sort: Sort.by( direction: Sort.Direction.ASC, ...properties: "key"))
    );

    return mongoOperations.aggregate(aggregation, inputType: Car.class, outputType: FilterOption.class).getMappedResults();
}
```

Рисунок А.92 – Метод findExistsMakes() репозиторію CarRepository

Рисунок А.93 – Методи findAllUniqueMakes(), findAllCarsByFilter() репозиторію CarRepository

Рисунок А.94 – Метод findExistsModels() репозиторію CarRepository

Рисунок А.95 – Метод findExistsYears() репозиторію CarRepository

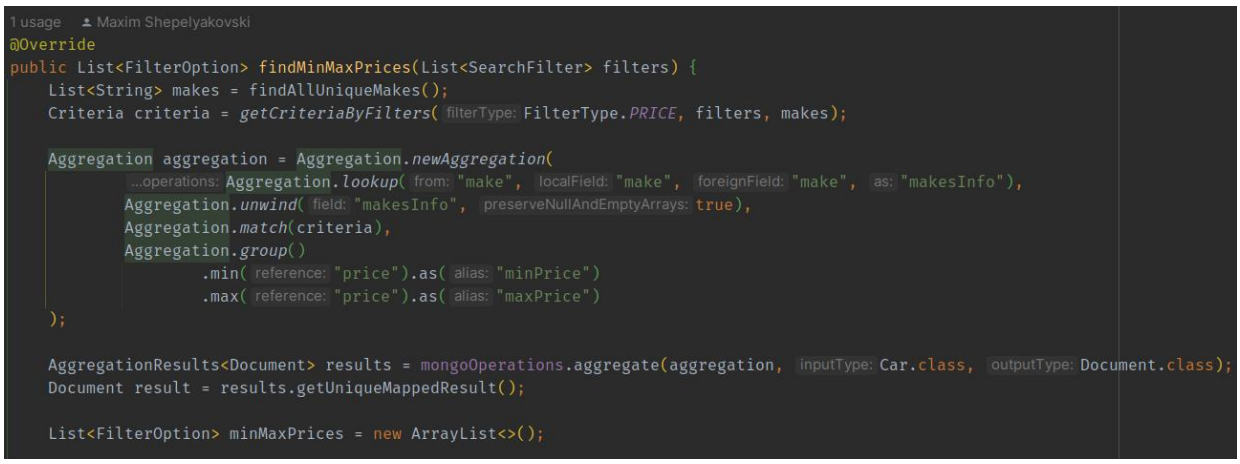


Рисунок А.96 – Метод findMinMaxPrices() репозиторію CarRepository

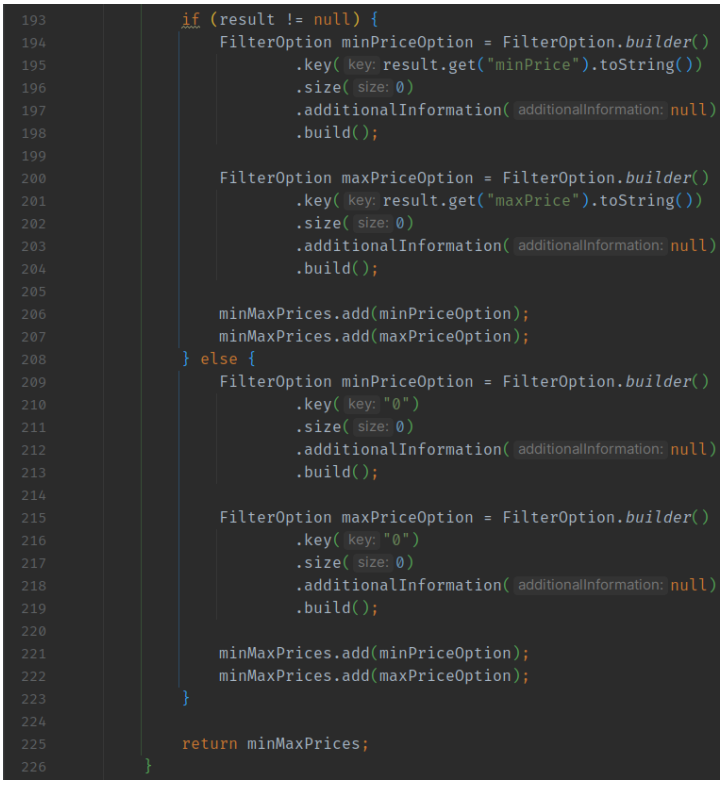


Рисунок А.97 – Метод findMinMaxPrices() репозиторію CarRepository

Рисунок А.98 – Метод getAllCars() репозиторію CarRepository

```
2 usages  Maxim Shepelyakovski
public interface FilterRepository extends MongoRepository<Filter, String>, QuerydslPredicateExecutor<Filter>, FilterRepositoryCustom {

}

2 usages  2 implementations  Maxim Shepelyakovski
interface FilterRepositoryCustom {
    1 usage  1 implementation  Maxim Shepelyakovski
    void addFilter(Filter filter);
}
```

Рисунок А.99 – Інтерфейс FilterRepository

```
no usages  Maxim Shepelyakovski *
public class FilterRepositoryImpl extends QuerydslRepositorySupport implements FilterRepositoryCustom {

    2 usages
    private final MongoOperations mongoOperations;

    Maxim Shepelyakovski
    @Autowired
    public FilterRepositoryImpl(MongoOperations mongoOperations) {
        super(mongoOperations);
        this.mongoOperations = mongoOperations;
    }

    1 usage  Maxim Shepelyakovski
    @Override
    public void addFilter(Filter filter) { mongoOperations.insert( objectToSave: filter); }

}
```

Рисунок А.100 – Репозиторій FilterRepositoryImpl

```
6 usages  Maxim Shepelyakovski
public interface MakeRepository extends MongoRepository<Make, String>, QuerydslPredicateExecutor<Make>, MakeRepositoryCustom {

}

2 usages  2 implementations  Maxim Shepelyakovski *
interface MakeRepositoryCustom {
    2 usages  1 implementation  Maxim Shepelyakovski
    List<Make> findAllMake();
    1 usage  1 implementation  Maxim Shepelyakovski
    void addMakes(List<Make> makes);
    1 usage  1 implementation  Maxim Shepelyakovski
    List<FilterOption> findUniqueCountries(List<SearchFilter> values);
}
```

Рисунок А.101 – Інтерфейс MakeRepository

```
2 usages  Maxim Shepelyakovski
@Override
public List<Make> findAllMake() {
    return from( path: make)
            .orderBy( o: make.make.asc())
            .fetch();
}

1 usage  Maxim Shepelyakovski
@Override
public void addMakes(List<Make> makes) {
    mongoOperations.insertAll( objectsToSave: makes);
}
```

Рисунок А.102 – Методи findAllMake(), addMakes()

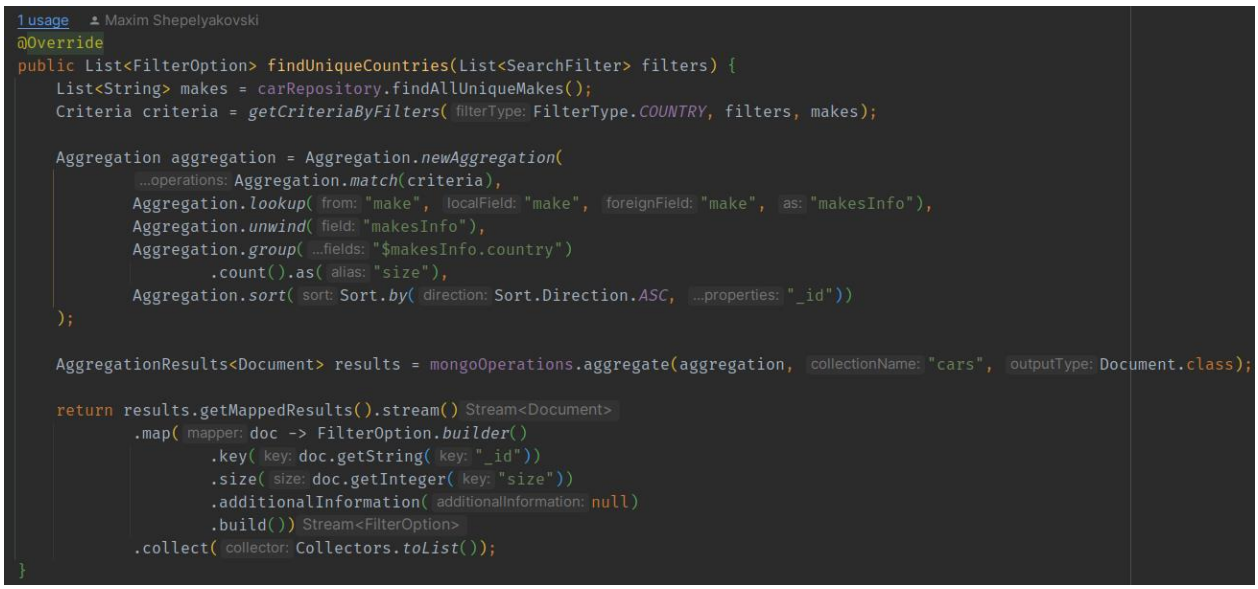


Рисунок А.103 – Метод findUniqueCountries()