**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА**

**ФАКУЛЬТЕТ УПРАВЛІННЯ ФІНАНСАМИ ТА БІЗНЕСУ**

**Кафедра** **цифрової економіки та бізнес-аналітики**

**КУРСОВА РОБОТА**

**з навчальної дисципліни**

**„Проектування та адміністрування БД і СД“**

**на тему:**

**„Інформаційна система продажу автомобілів (Дилерський центр)“**

**Спеціальність**: 051 «Економіка»

**Спеціалізація**: «Інформаційні технології в бізнесі»

**Освітній ступінь**: бакалавр

**Науковий керівник**: **Виконавець**:

студент(ка) групи УФЕ-31 с

\_\_\_\_\_\_\_\_\_\_ “\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_ 2023 р. Гнатів Н. М

(підпис)

\_\_\_\_\_\_\_\_\_\_\_\_ “\_\_\_\_” \_\_\_\_\_\_\_\_\_ 2023 р.

(підпис)

**Загальна кількість балів** \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис, ПІП членів комісії)

ЛЬВІВ 2023

ЗМІСТ

[ВСТУП 3](#_Toc135935522)

[1 АНАЛІЗ ВИМОГ 5](#_Toc135935523)

[**1.1.** **Постановка завдання** 5](#_Toc135935524)

[**1.2.** **Розробка моделі варіантів використання веб-сайту** 5](#_Toc135935525)

[**1.3.** **Аналіз засобів реалізації (техніко-економічне обґрунтування вибору)**.............................................................................................................................6](#_Toc135935526)

[**1.4.** **SWOT-аналіз** 10](#_Toc135935527)

[2 РОЗРОБКА БАЗИ ДАНИХ 12](#_Toc135935528)

[**2.1.** **Опис моделі даних** 12](#_Toc135935529)

[**2.1.** **Нормалізація реляційних відношень** 16](#_Toc135935530)

[**2.2.** **Обмеження цілісності даних** 17](#_Toc135935531)

[**2.4.** **Реалізація SQL-скрипту** 18](#_Toc135935532)

[3 РОЗРОБКА ВЕБ-ДОДАТКУ 22](#_Toc135935533)

[**3.1.** **Структура веб-сайту** 22](#_Toc135935534)

[**3.2.** **Програмування серверної частини** 22](#_Toc135935535)

[**3.3.** **Програмування клієнтської частини** 27](#_Toc135935536)

[ВИСНОВКИ 31](#_Toc135935537)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 32](#_Toc135935538)

[ДОДАТОК A 33](#_Toc135935539)

[ДОДАТОК B 38](#_Toc135935540)

### ВСТУП

Автомобільна індустрія - це важлива складова економіки країни, а продаж автомобілів є однією з головних галузей цієї індустрії. За останні роки збільшилася кількість автосалонів та дилерських центрів, які займаються продажем автомобілів. Водночас, високий рівень конкуренції вимагає від дилерів не тільки володіти високим рівнем знань про автомобілі та їх характеристики, але й мати зручну та ефективну інформаційну систему продажу автомобілів.

***Метою*** даного дослідження є розробка інформаційної системи продажу автомобілів, яка допоможе підвищити ефективність продажів та покращити якість обслуговування клієнтів.

***Предметом дослідження*** є інформаційна система продажу автомобілів, що включає в себе базу даних автомобілів, засоби пошуку, відображення та фільтрації інформації про автомобілі, можливість оформлення замовлення на автомобіль та інші функції.

***Об'єктом дослідження*** є інформаційна система продажу автомобілів. Ця система включає в себе комплекс програмних та апаратних засобів, які використовуються для збору, обробки та зберігання інформації про автомобілі, їх характеристики, власників, ціни, продавців та покупців.

***Отримані результати******дослідження*** щодо інформаційної системи продажу автомобілів можуть бути корисні в декількох ділянках прикладної діяльності. Перш за все, це може бути корисно для автомобільних компаній, дилерів та інших організацій, що займаються продажем автомобілів. Завдяки розробленим рекомендаціям вони можуть поліпшити свої інформаційні системи, що сприятиме покращенню обслуговування клієнтів і збільшенню продажів.

***Використане програмне забезпечення***: у ході виконання роботи було використано наступний технічний стек: Java, Spring Framework, MySQL, HTML, CSS, JavaScript.

***Структура роботи:*** Курсова робота складається з трьох розділів («Аналіз вимог», «Розробка бази даних» та «Розробка веб-додатку»), висновків, списку використаних джерел та додатків.

### 1 АНАЛІЗ ВИМОГ

## **Постановка завдання**

Завдання курсової роботи на тему "Розробка інформаційної системи продажу автомобілів" включає наступні пункти:

1. Дослідити існуючі інформаційні системи продажу автомобілів та проаналізувати їх переваги та недоліки.
2. Сформулювати вимоги до розроблюваної інформаційної системи продажу автомобілів з урахуванням потреб користувачів та вимог ринку.
3. Розробити концепцію та архітектуру інформаційної системи продажу автомобілів.
4. Розробити базу даних для зберігання інформації про автомобілі, клієнтів, замовлення та операції з продажу.
5. Розробити користувацький інтерфейс системи для зручного та швидкого взаємодії з користувачами.
6. Провести тестування та аналіз ефективності розробленої інформаційної системи продажу автомобілів.

## **Розробка моделі варіантів використання веб-сайту**

Загалом ми маємо трьох акторів використання:

* Звичайний користувач – покупець сайту, основний сценарій якого:

Перегляд каталогу –> Додавання покупки до корзини –> Оформлення замовлення -> Оплата замовлення

* Працівник центру – користувач, що проходить аутентифікацію та до попердніх можливостей у нього додається перегляд та менеджмент замовленнь.
* Менеджер центру – користувач, що теж проходить відповідну аутенифікацію, до попередніх можливостей додається можливість менеджменту елементами каталогу.

Опишемо модель варіантів використання за допомогою Use-Case діаграми:

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Рис. 1. Use-Case діаграма використання системи

## **Аналіз засобів реалізації (техніко-економічне обґрунтування вибору)**

Для реалізації даної роботи було вибрано наступні технології:

* Thymeleaf, Bootstrap (Дані технології для реалізації Front-end частини);
* Java, Spring Framework, MySql, Maven, Tomcat, (Для реалізації Back-end частини).

Для фронт-енд частини було обрано:

* Thymeleaf - це шаблонізатор для веб-програмування, який використовується для створення HTML, XML, Javascript та CSS-файлів з шаблонів, які містять дані та логіку.

Переваги використання Thymeleaf у веб-розробці:

* Синтаксис Thymeleaf є читабельним та легким для вивчення.
* Використання Thymeleaf дозволяє розділити логіку та дані між сервером та клієнтом.
* Thymeleaf надає можливість використовувати звичайні HTML-теги, що робить його легким для розуміння та використання.
* Thymeleaf підтримує підключення до баз даних та збереження даних у шаблонах.
* Thymeleaf може використовуватися як у статичних, так і у динамічних веб-сайтах.
* Thymeleaf підтримує міжнароднізацію, що дозволяє створювати сайти з різними мовами та культурними особливостями.

У загальному, Thymeleaf дозволяє ефективно створювати веб-сайти та додавати логіку до статичних шаблонів, що робить його дуже популярним серед розробників веб-додатків.

* Bootstrap - це найпопулярніший фреймворк для розробки веб-сайтів, який містить набір готових компонентів та стилів для швидкої розробки веб-додатків.

Переваги використання Bootstrap у веб-розробці:

* Bootstrap забезпечує зручний та легкий для використання сітковий систему, що дозволяє створювати респонсивні та мобільні дизайни.
* Bootstrap містить велику кількість компонентів та стилів, що дозволяє зменшити час розробки веб-сайтів та зберегти кошти на дизайні.
* Bootstrap має широкі можливості кастомізації, що дозволяє налаштувати дизайн сайту під власні потреби та вимоги.
* Bootstrap має велику спільноту розробників, що надає документацію, допомогу та швидкий розвиток фреймворку.
* Bootstrap підтримує HTML, CSS та JavaScript, що дозволяє розробникам створювати повноцінні веб-сайти без необхідності використання інших фреймворків.

У загальному, Bootstrap є потужним інструментом для швидкої та ефективної розробки веб-додатків. Він дозволяє розробникам створювати привабливі та респонсивні сайти, зменшуючи час та витрати на дизайн та розробку.

Для бек-енд частини було обрано:

* Java - це об'єктно-орієнтована мова програмування, яка була розроблена в компанії Sun Microsystems у 1990-х роках. Java є однією з найбільш популярних мов програмування у світі.

Переваги використання Java у програмуванні:

* Java є незалежною від платформи, що дозволяє програмістам розробляти програми на будь-якій операційній системі без необхідності змінювати код.
* Java є об'єктно-орієнтованою мовою програмування, що дозволяє програмістам створювати повторно використовувані компоненти та зменшувати час розробки програм.
* Java має вбудовану підтримку багатопоточності, що дозволяє програмістам створювати багатопотокові програми для ефективної роботи з багатопоточними задачами.
* Java має вбудовану систему керування пам'яттю, що дозволяє програмістам уникнути проблем з пам'яттю, таких як витоки пам'яті.
* Java має велику спільноту програмістів, що надає документацію, підтримку та допомогу в розвитку програм.

У загальному, Java є потужним інструментом для програмування, який дозволяє створювати ефективні, надійні та масштабовані програми для різних платформ. Java є популярним вибором для веб-розробки, розробки мобільних додатків та розробки програмного забезпечення для різних індустрій.

* MySQL - це безкоштовна, відкрита реляційна система управління базами даних (СУБД), яка була розроблена компанією Oracle. MySQL використовує мову SQL для управління базами даних.

Переваги використання MySQL для управління базами даних:

* MySQL є надійною та стабільною СУБД, яка здатна працювати з великим обсягом даних та забезпечує високу продуктивність.
* MySQL є безкоштовною та відкритою СУБД, що дозволяє програмістам використовувати її для будь-яких проектів без необхідності платити за ліцензію.
* MySQL має вбудовану підтримку транзакцій, що дозволяє виконувати операції з базами даних без ризику втрати даних.
* MySQL має велику спільноту користувачів та програмістів, яка надає підтримку, документацію та допомогу у вирішенні проблем.
* MySQL має широкі можливості конфігурування та налаштування, що дозволяє програмістам налаштовувати СУБД під конкретні потреби проекту.

У загальному, MySQL є потужним інструментом для управління базами даних, який забезпечує надійність, стабільність та продуктивність. MySQL є популярним вибором для веб-розробки, розробки програмного забезпечення та різних інших проектів, які потребують роботи з базами даних.

* Apache Maven - це інструмент для управління проектами на мові Java, який дозволяє збирати, виконувати та управляти залежностями програмного забезпечення. Maven використовує файл pom.xml (Project Object Model) для опису проекту та його залежностей.

Переваги використання Maven для управління проектами:

* Простота у використанні та конфігурації проекту, що дозволяє програмістам швидко розпочинати розробку.
* Maven дозволяє автоматизувати процес збирання та управління залежностями, що дозволяє зменшити ручну роботу та покращити ефективність проекту.
* Maven має велику кількість плагінів, які дозволяють виконувати різноманітні задачі, такі як тестування, збірка документації, розгортання проекту та інші.
* Maven дозволяє створювати проекти з використанням шаблонів, що дозволяє швидко створювати нові проекти та забезпечувати їхню структуру.
* Maven має велику спільноту користувачів та підтримується Apache Software Foundation, що забезпечує надійність та стабільність інструменту.

У загальному, Apache Maven є потужним інструментом для управління проектами на мові Java, який забезпечує простоту у використанні та автоматизацію процесів збирання та управління залежностями. Maven дозволяє програмістам швидко розпочинати розробку та забезпечувати ефективність проекту завдяки великій кількості плагінів та підтримці спільноти.

## **SWOT-аналіз**

Strengths (Сильні сторони):

1. Ефективне управління інформацією: Інформаційна система дозволяє збирати, оновлювати і аналізувати великий обсяг даних про автомобілі, клієнтів, продажі, запаси тощо, що полегшує процеси управління та прийняття рішень.

2. Автоматизація процесів: Система дозволяє автоматизувати багато рутинних завдань, таких як обробка замовлень, оновлення бази даних, формування звітів, що зменшує можливість помилок та збільшує продуктивність.

3. Збільшення ефективності продажів: Інформаційна система надає зручний доступ до інформації про автомобілі, що дозволяє продавцям швидше та ефективніше обслуговувати клієнтів, надавати необхідні дані та здійснювати продажі.

4. Вдосконалення зв'язку з клієнтами: Система дозволяє вести базу даних клієнтів, збирати та аналізувати дані про їхні потреби та переваги, що дозволяє покращити комунікацію з клієнтами та забезпечити персоналізований підхід.

Weaknesses (Слабкі сторони):

1. Висока вартість впровадження: Розробка та впровадження інформаційної системи може бути дорогою, особливо для невеликих дилерських центрів з обмеженими фінансовими ресурсами.
2. Залежність від технологій: Якщо система базується на спеціалізованому програмному забезпеченні, можуть виникнути проблеми з сумісністю та оновленнями, що потребуватимуть додаткових зусиль для підтримки.
3. Потреба в навчанні: Впровадження нової інформаційної системи вимагає часу та ресурсів для навчання співробітників.

# РОЗРОБКА БАЗИ ДАНИХ

## **Опис моделі даних**

Проаналізувавши предметну область можна виділити наступні сутності:

* Account
* Order
* Vehicle

Виділимо атрибути, необхідні даним сутностям:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Account | Назва | Тип даних | Опис атрибуту | Додаткові обмеження цілісності |
| user\_name | varchar | Назва користувача | Обов'язковий |
| active | boolean | Активність | Обов'язковий |
| Encrypted\_password | varchar | Зашифрований пароль | Обов'язковий |
| User\_role | varchar | Роль акаунта | Обов'язковий |

Таблиця 1. Атрибути сутності «Account»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vehicle | Назва | Тип даних | Опис атрибуту | Додаткові обмеження цілісності |
| code | long | Код машини | Первинний ключ |
| image | blob | Фото машини | Обов'язковий |
| model | varchar | Модель машини | Обов'язковий |
| price | double | Ціна | Обов'язковий |
| Create\_date | datetime | Час створення елементу | Обов'язковий |

Таблиця 2. Атрибути сутності «Vehicle»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Order | Назва | Тип даних | Опис атрибуту | Додаткові обмеження цілісності |
| id | int | Номер | Первинний ключ |
| amount | double | Кількість | Обов'язковий |
| Сustomer\_address | varchar | Адреса покупця | Обов'язковий |
| Сustomer\_email | varchar | Пошта покупця | Обов'язковий |
| Сustomer\_name | varchar | Ім’я покупця | Обов'язковий |
| Сustomer\_phone | varchar | Номер телефону покупця | Обов'язковий |
| date | datetime | Дата | Обов'язковий |
|

Таблиця 3. Атрибути сутності «Order»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Order\_details | Назва | Тип даних | Опис атрибуту | Додаткові обмеження цілісності |
| id | varchar | Номер | Первинний ключ |
| amount | double | Кількість | Обов'язковий |
| price | double | Ціна | Обов'язковий |
| Order\_id | long | Номер замовлення | Зовнішній ключ |
| product\_id | long | Номер машини | Зовнішній ключ |
|

Таблиця 4. Атрибути сутності «Order\_details»

Власне, після проектування отримуємо наступну ER-діаграму:

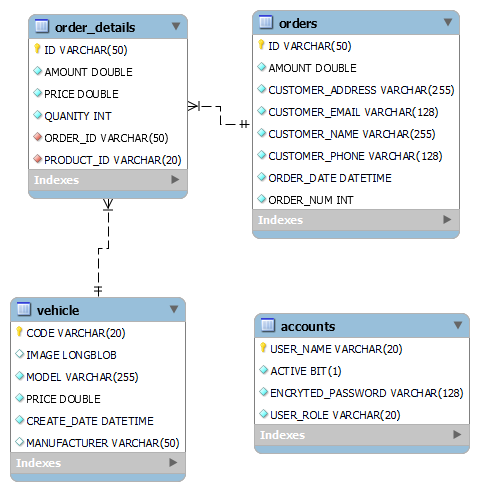


Рис. 2. ER-діаграма бази даних

Надана інформація містить опис схеми бази даних для трьох таблиць: order\_details (деталі замовлення), orders (замовлення) та accounts (акаунти). Розберемо кожну таблицю та її стовпці:

1. Таблиця order\_details:

• ID: Стовпець типу VARCHAR(50), який представляє ідентифікатор для кожної деталі замовлення.

• AMOUNT: Стовпець типу DOUBLE, в якому зберігається сума, пов'язана з деталлю замовлення.

• PRICE: Стовпець типу DOUBLE, який відображає ціну продукту у деталі замовлення.

• QUANTITY: Стовпець типу INT, в якому зберігається кількість продукту у деталі замовлення.

• ORDER\_ID: Стовпець типу VARCHAR(50), який посилається на ідентифікатор замовлення, до якого належить деталь.

• PRODUCT\_ID: Стовпець типу VARCHAR(20), який посилається на ідентифікатор продукту, пов'язаного з деталлю замовлення.

• Індекси: Надана інформація не вказує жодних індексів для цієї таблиці.

2. Таблиця orders:

• ID: Стовпець типу VARCHAR(50), який представляє ідентифікатор для кожного замовлення.

• AMOUNT: Стовпець типу DOUBLE, в якому зберігається загальна сума замовлення.

• CUSTOMER\_ADDRESS: Стовпець типу VARCHAR(255), в якому зберігається адреса клієнта, що робить замовлення.

• CUSTOMER\_EMAIL: Стовпець типу VARCHAR(128), в якому зберігається електронна пошта клієнта.

• CUSTOMER\_NAME: Стовпець типу VARCHAR(255), в якому зберігається ім'я клієнта.

• CUSTOMER\_PHONE: Стовпець типу VARCHAR(128), в якому зберігається телефонний

•

• ORDER\_DATE: Стовпець типу DATETIME, який зберігає дату замовлення.

• ORDER\_NUM: Стовпець типу INT, який представляє номер замовлення.

•

3. Таблиця accounts містить інформацію про користувачів акаунтів. Ось пояснення кожного стовпця:

• USER\_NAME: Стовпець типу VARCHAR(20), який містить ім'я користувача акаунту.

• ACTIVE: Стовпець типу BIT (1), який вказує, чи є акаунт активним (1) або неактивним (0).

• ENCRYPTED\_PASSWORD: Стовпець типу VARCHAR(128), який містить зашифрований пароль для акаунту.

• USER\_ROLE: Стовпець типу VARCHAR(20), який вказує роль користувача.

## **Нормалізація реляційних відношень**

* + 1. Перевірка на 1 нормальну форму (1NF):
* Усі атрибути у таблицях повинні бути атомарними (недільними).
* У кожній таблиці має бути унікальний ідентифікатор - первинний ключ.

Перша таблиця (account) має атомарні атрибути та первинний ключ, тому вона знаходиться в 1NF.

Друга таблиця (vehicle) також має атомарні атрибути та первинний ключ, тому вона також знаходиться в 1NF.

Третя таблиця (order) має атомарні атрибути та первинний ключ, тому вона також знаходиться в 1NF.

Четверта таблиця (order\_details) має атомарні атрибути та первинний ключ, тому вона також знаходиться в 1NF.

Отже, всі три таблиці знаходяться в 1 нормальній формі.

* + 1. Перевірка на 2 нормальну форму (2NF):
* Таблиця повинна знаходитися в 1NF.
* Кожний неключовий атрибут повинен залежати від первинного ключа в цілому, а не від окремих його атрибутів.

У таблиці account всі атрибути залежать від первинного ключа, тому вона знаходиться в 2NF.

У таблиці vehicle також всі атрибути залежать від первинного ключа, тому вона знаходиться в 2NF.

І у таблиці order всі атрибути залежать від первинного ключа, тому вона знаходиться в 2NF.

* + 1. Перевірка на 3 нормальну форму (3NF):
* Таблиця повинна знаходитися в 2NF.
* Кожен неключовий атрибут повинен залежати від первинного ключа безпосередньо, а не через інші атрибути.

У таблиці account всі атрибути залежать від первинного ключа (vin), тому вона знаходиться в 3NF.

У таблиці vehicle всі атрибути залежать від первинного ключа (customer\_id), тому вона знаходиться в 3NF.

У таблиці order всі атрибути залежать від первинного ключа, тому вона знаходиться в 3NF.

## **Обмеження цілісності даних**

Для реалізації цілісності даних нам необхідні наступні обмеження:

Для таблиці **vehicle**:

1. odometer повинен бути додатнім числом або 0, тому додамо обмеження CHECK:

ALTER TABLE vehicle ADD CONSTRAINT odometer\_positive CHECK > 0);

1. price повинен бути додатнім числом, тому додамо обмеження CHECK:

ALTER TABLE vehicle ADD CONSTRAINT price\_positive CHECK (price > 0);

Для таблиці **customer**:

1. purchase\_price повинен бути додатнім числом, тому додамо обмеження CHECK:

ALTER TABLE customer ADD CONSTRAINT purchase\_price\_positive CHECK (purchase\_price > 0);

1. phone\_number повинен бути унікальним, тому додамо обмеження UNIQUE:

ALTER TABLE customer ADD CONSTRAINT phone\_number\_unique UNIQUE (phone\_number);

Для таблиці **sale**:

1. amount повинен бути додатнім числом, тому додамо обмеження CHECK:

ALTER TABLE sale ADD CONSTRAINT amount\_positive CHECK (amount > 0);

1. isDiscounted повинен бути значенням 0 або 1 (тобто, відповідно до типу BOOL), тому додамо обмеження CHECK:

ALTER TABLE sale ADD CONSTRAINT isDiscounted\_boolean CHECK (isDiscounted IN (0,1));

1. customer\_ID повинен посилатися на існуючий запис у таблиці customer, тому додамо обмеження FOREIGN KEY:

ALTER TABLE sale ADD CONSTRAINT customer\_ID\_fk FOREIGN KEY (customer\_ID) REFERENCES customer(customer\_id);

1. vehicle\_ID повинен посилатися на існуючий запис у таблиці vehicle, тому додамо обмеження FOREIGN KEY:

ALTER TABLE sale ADD CONSTRAINT vehicle\_ID\_fk FOREIGN KEY (vehicle\_ID) REFERENCES vehicle(vin)

## **Реалізація SQL-скрипту**

SQL скрипт - це набір команд, які виконуються в базі даних для створення, зміни або отримання інформації. Він дозволяє програмістам взаємодіяти з базою даних, виконуючи запити і оператори для створення, оновлення та видалення даних. SQL скрипти можуть включати створення таблиць, внесення даних, зміну структури таблиць, виконання складних запитів та інші операції, що дозволяють ефективно управляти та аналізувати дані в базі даних.

Типовими для ІС є запити:

* Перегляд усіх замовлень, які були зроблені після певної дати.

SELECT \* FROM orders WHERE order\_date > '2023-01-01';

* Перегляд імен та електронних адресів клієнтів, які замовили автомобіль з кодом X

SELECT customer\_name, customer\_email FROM orders

JOIN order\_details ON orders.id = order\_details.order\_id

WHERE order\_details.product\_id = 'V001';

* Перегляд автомобілів, вартість яких більша за Х

SELECT \* FROM vehicle WHERE price > 100000;

* Перегляд автомобілів з маркою X.

SELECT \* FROM vehicle WHERE model = 'Tesla Model 3';

* Список виробників та кількість моделей автомобілів для кожного виробника.

SELECT manufacturer, COUNT(\*) AS model\_count FROM vehicle

GROUP BY manufacturer;

* Список усіх акаунтів з роллю X

SELECT \* FROM accounts WHERE user\_role = 'USER';

* Список клієнтів, які мають більше 5 замовлень

SELECT customer\_name FROM orders

GROUP BY customer\_name

HAVING COUNT(\*) > 5;

Розширене завдання:

* Модель та ціна автомобіля, який має найвищу ціну в таблиці vehicle

SELECT model, price FROM vehicle

WHERE price = (SELECT MAX(price) FROM vehicle);

* Сумарна кількість та сумарна вартість замовлених автомобілів за кожним виробником

SELECT manufacturer, SUM(quanity) AS total\_quanity, SUM(amount) AS total\_amount FROM vehicle

JOIN order\_details ON vehicle.code = order\_details.product\_id

GROUP BY manufacturer;

* Номер замовлення та дату замовлення для тих замовлень, які мають більше ніж один автомобіль в деталях замовлення

SELECT order\_num, order\_date FROM orders

WHERE id IN (SELECT order\_id FROM order\_details GROUP BY order\_id HAVING COUNT(\*) > 1);

* Імена та телефони клієнтів, які замовили автомобіль з моделлю Х

SELECT customer\_name, customer\_phone FROM orders

JOIN order\_details ON orders.id = order\_details.order\_id

JOIN vehicle ON order\_details.product\_id = vehicle.code

WHERE vehicle.model = 'Tesla Model 3';

* Модель та середня ціна замовлених автомобілів для кожної моделі.

SELECT model, AVG(price) AS average\_price FROM vehicle

JOIN order\_details ON vehicle.code = order\_details.product\_id

GROUP BY model;

* Ім’я користувача та кількість замовлень, які він зробив за Х рік

Повний SQL-скрипт для створення бази даних знаходиться у Додатку А

SELECT user, COUNT(\*) AS order\_count FROM accounts

JOIN orders ON accounts.user = orders.customer\_email

WHERE YEAR(order\_date) = 2023

GROUP BY user;

### 3 РОЗРОБКА ВЕБ-ДОДАТКУ

## **Структура веб-сайту**

Для реалізації зазначеного у пункті 1.2. функціоналу нам необхідні наступні веб-сторінки:

* Додавання машини
* «Домашня» сторінка сайту
* Каталог усіх машин
* Корзина
* Профіль користувача
* Створення та оплата замовлення
* Перегляд усіх замовлень

## **Програмування серверної частини**

Для програмування даної інформаційної системи було використано MVC-архітектуру, що реалізовано за допомогою різних частин фреймворку Spring.

Перш за все, реалізуємо шар entity – класи сутностей, що пов'язані з відповідними таблицями БД.

Далі необхідно реалізувати шар repository. Так як ми користуємося можливостями Spring Data JPA, то репозиторії, пов'язані з відповідними сутностями, імплементують інтерфейс JPARepository, який надає певні методи для взаємодії з БД.

Після цього відповідно реалізуємо модуль services, що відповідатиме за логіку застосунку.

Наступним кроком можемо приступати до реалізації контролерів, що взаємодіятимуть із клієнтською частиною за допомогою ендпоінтів.

Наприклад, endpoint для додавання елементів у каталог:

@RequestMapping(value = { "/admin/product" }, method = RequestMethod.*POST*)  
public String productSave(Model model, //  
 @ModelAttribute("productForm") @Validated VehicleForm vehicleForm, //  
 BindingResult result, //  
 final RedirectAttributes redirectAttributes) {  
  
 if (result.hasErrors()) {  
 return "product";  
 }  
 try {  
 vehicleDAO.save(vehicleForm);  
 } catch (Exception e) {  
 Throwable rootCause = ExceptionUtils.*getRootCause*(e);  
 String message = rootCause.getMessage();  
 model.addAttribute("errorMessage", message);  
 // Show product form.  
 return "product";  
 }  
  
 return "redirect:/productList";  
}

Лістинг 1 – Endpoint з POST-мапінгом

Також, важливими особливостями цього проекту, про які не згадувалось раніше є

* валідація даних при створенні нового елементу каталогу або при заповненні даних для замовлення
* @Override  
  public void validate(Object target, Errors errors) {  
   VehicleForm vehicleForm = (VehicleForm) target;  
    
   // Check the fields of ProductForm.  
   ValidationUtils.*rejectIfEmptyOrWhitespace*(errors, "code", "NotEmpty.productForm.code");  
   ValidationUtils.*rejectIfEmptyOrWhitespace*(errors, "name", "NotEmpty.productForm.name");  
   ValidationUtils.*rejectIfEmptyOrWhitespace*(errors, "price", "NotEmpty.productForm.price");  
    
   String code = vehicleForm.getCode();  
   if (code != null && code.length() > 0) {  
   if (code.matches("\\s+")) {  
   errors.rejectValue("code", "Pattern.productForm.code");  
   } else if (vehicleForm.isNewProduct()) {  
   Vehicle vehicle = vehicleDAO.findVehicle(code);  
   if (vehicle != null) {  
   errors.rejectValue("code", "Duplicate.productForm.code");  
   }  
   }  
   }  
  }

Лістинг 2 – Валідація машин при додаванні

* реалізація можливості оплати за допомогою PayPal.
* пагінація при виводі елементів каталогу або при виведенні списку замовлень
* package com.example.demo.pagination;  
    
  import java.util.ArrayList;  
  import java.util.List;  
  import org.hibernate.ScrollMode;  
  import org.hibernate.ScrollableResults;  
  import org.hibernate.query.Query;  
    
  public class PaginationResult<E> {  
   private int totalRecords;  
   private int currentPage;  
   private List<E> list;  
   private int maxResult;  
   private int totalPages;  
     
   private int maxNavigationPage;  
     
   private List<Integer> navigationPages;  
     
   // @page: 1, 2, ..  
   public PaginationResult(Query<E> query, int page, int maxResult, int maxNavigationPage) {  
   final int pageIndex = Math.*max*(page - 1, 0);  
   int fromRecordIndex = pageIndex \* maxResult;  
   int maxRecordIndex = fromRecordIndex + maxResult;  
   ScrollableResults resultScroll = query.scroll(ScrollMode.*SCROLL\_INSENSITIVE*);  
     
   List<E> results = new ArrayList<>();  
     
   boolean hasResult = resultScroll.first();  
     
   if (hasResult) {  
   // Scroll to position:  
   hasResult = resultScroll.scroll(fromRecordIndex);  
     
   if (hasResult) {  
   do {  
   E record = (E) resultScroll.get(0);  
   results.add(record);  
   } while (resultScroll.next()//  
   && resultScroll.getRowNumber() >= fromRecordIndex  
   && resultScroll.getRowNumber() < maxRecordIndex)  
   }  
     
   // Go to Last record.  
   resultScroll.last();  
   }  
     
   // Total Records  
   this.totalRecords = resultScroll.getRowNumber() + 1;  
   this.currentPage = pageIndex + 1;  
   this.list = results;  
   this.maxResult = maxResult;  
     
   if (this.totalRecords % this.maxResult == 0) {  
   this.totalPages = this.totalRecords / this.maxResult;  
   } else {  
   this.totalPages = (this.totalRecords / this.maxResult) + 1;  
   }  
     
   this.maxNavigationPage = maxNavigationPage;  
     
   if (maxNavigationPage < totalPages) {  
   this.maxNavigationPage = maxNavigationPage;  
   }  
     
   this.calcNavigationPages();  
   }  
     
   private void calcNavigationPages() {  
     
   this.navigationPages = new ArrayList<Integer>();  
     
   int current = Math.*min*(this.currentPage, this.totalPages);  
     
   int begin = current - this.maxNavigationPage / 2;  
   int end = current + this.maxNavigationPage / 2;  
     
   // The first page  
   navigationPages.add(1);  
   if (begin > 2) {  
     
   // Using for '...'  
   navigationPages.add(-1);  
   }  
     
   for (int i = begin; i < end; i++) {  
   if (i > 1 && i < this.totalPages) {  
   navigationPages.add(i);  
   }  
   }  
     
   if (end < this.totalPages - 2) {  
     
   // Using for '...'  
   navigationPages.add(-1);  
   }  
   // The last page.  
   navigationPages.add(this.totalPages);  
   }  
     
   public int getTotalPages() {  
   return totalPages;  
   }  
     
   public int getTotalRecords() {  
   return totalRecords;  
   }  
     
   public int getCurrentPage() {  
   return currentPage;  
   }  
     
   public List<E> getList() {  
   return list;  
   }  
     
   public int getMaxResult() {  
   return maxResult;  
   }  
     
   public List<Integer> getNavigationPages() {  
   return navigationPages;  
   }  
     
   }

Лістинг 3 – Реалізація пагінації

Нижче наведено структуру проекту у вигляді діаграми класів.

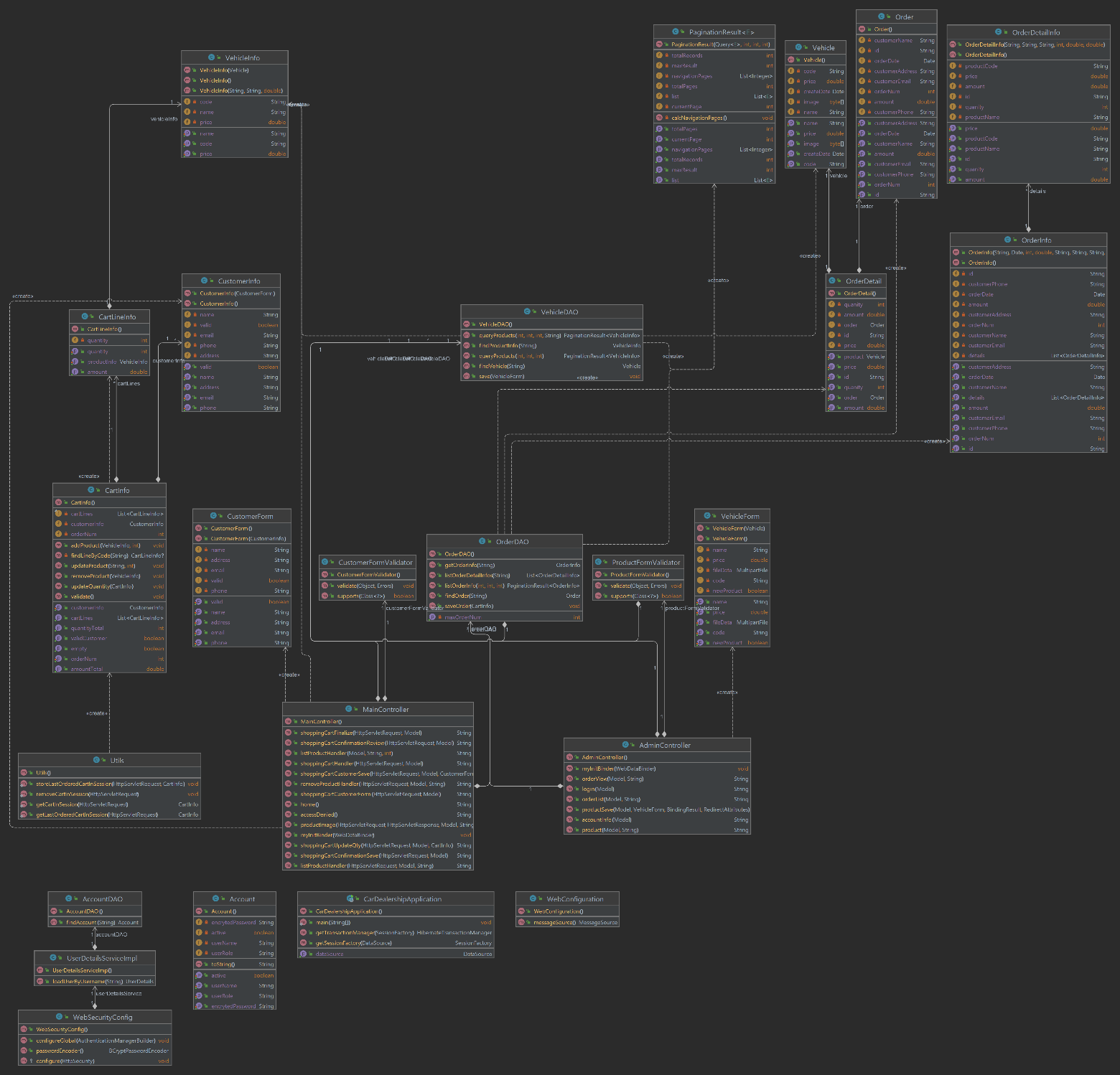


Рис. 3. Діаграма класів додатку

Повний лістинг коду можна знайти і Додатку В.

## **Програмування клієнтської частини**

Для реалізації клієнтської частини було обрано технологію, Thymeleaf, що є технологією веб-розробки на мові Java, яка дозволяє створювати динамічні веб-сторінки.

Результати розробки клієнтської частини



Рис.4. «Домашня» сторінка

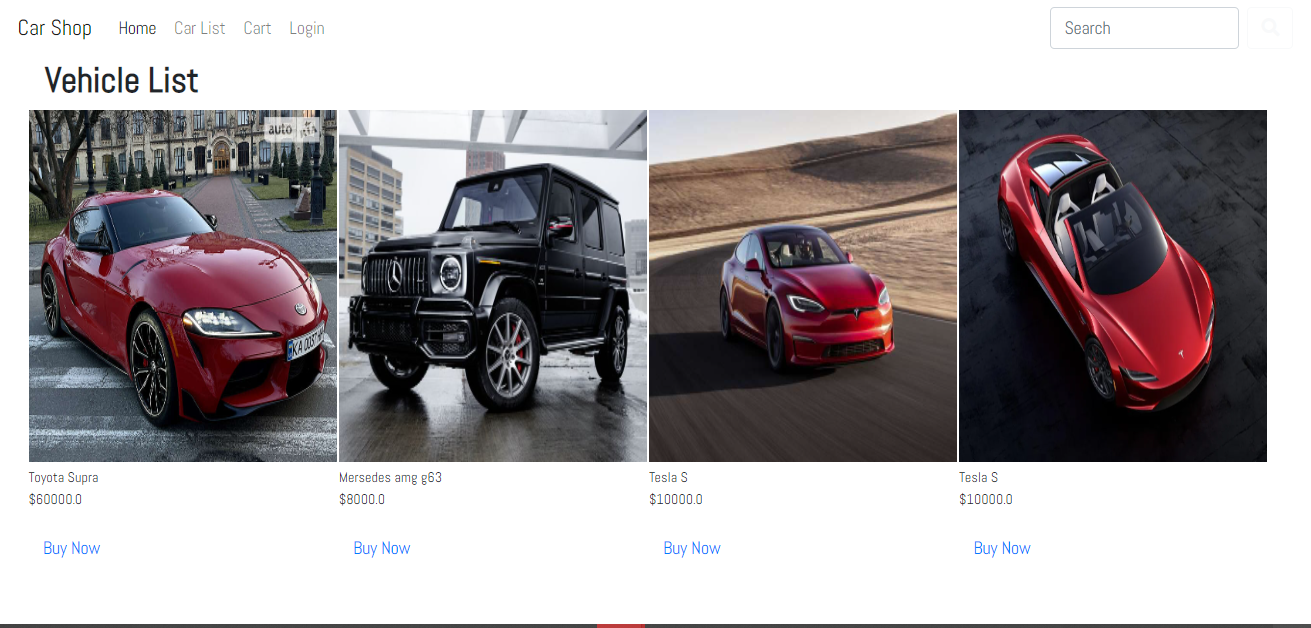


Рис. 5. Каталог машин

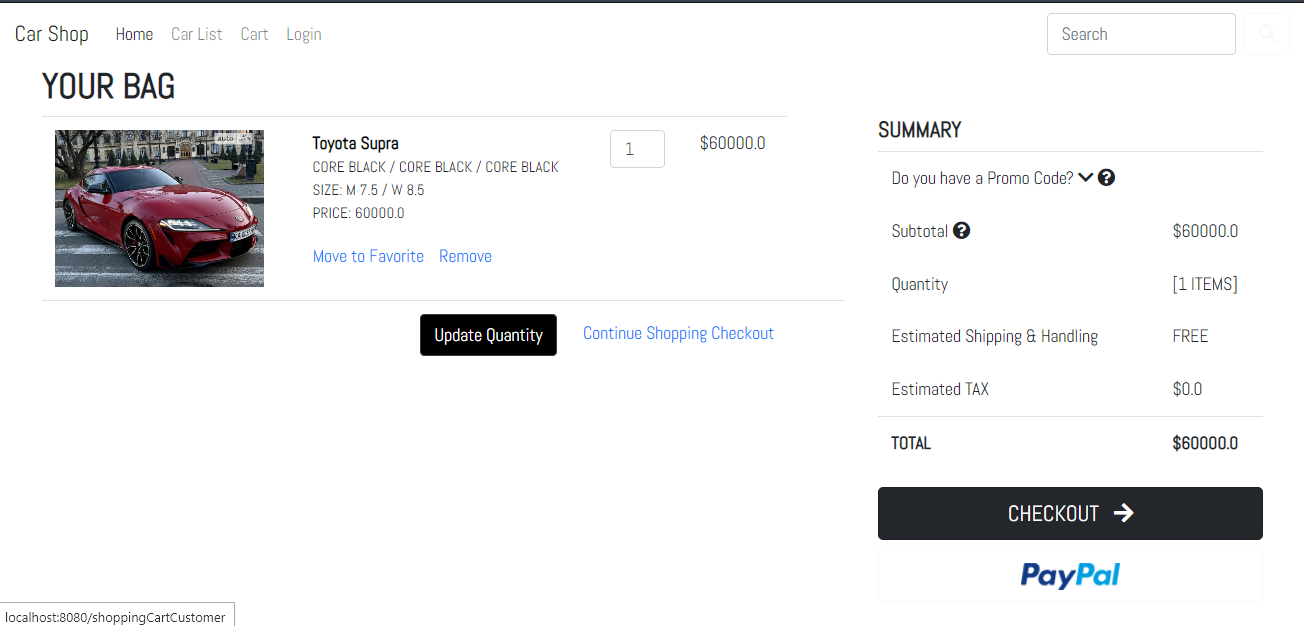


Рис. 6. Корзина покупця

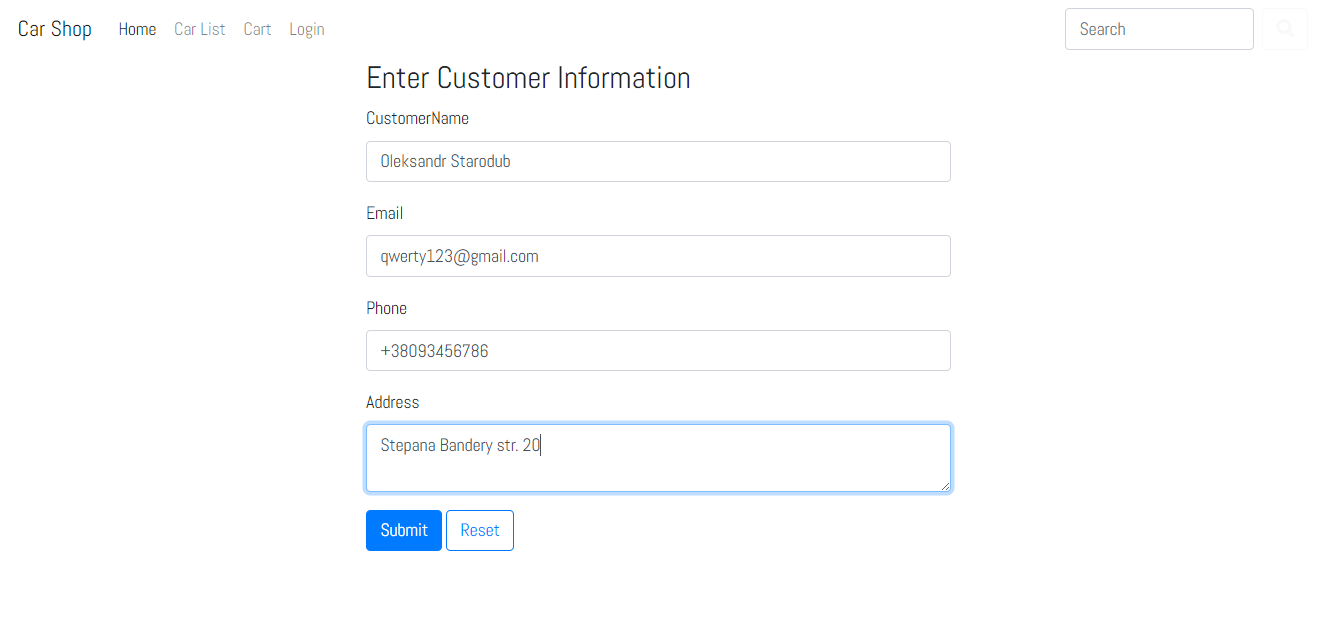


Рис. 7. Оформлення замовлення

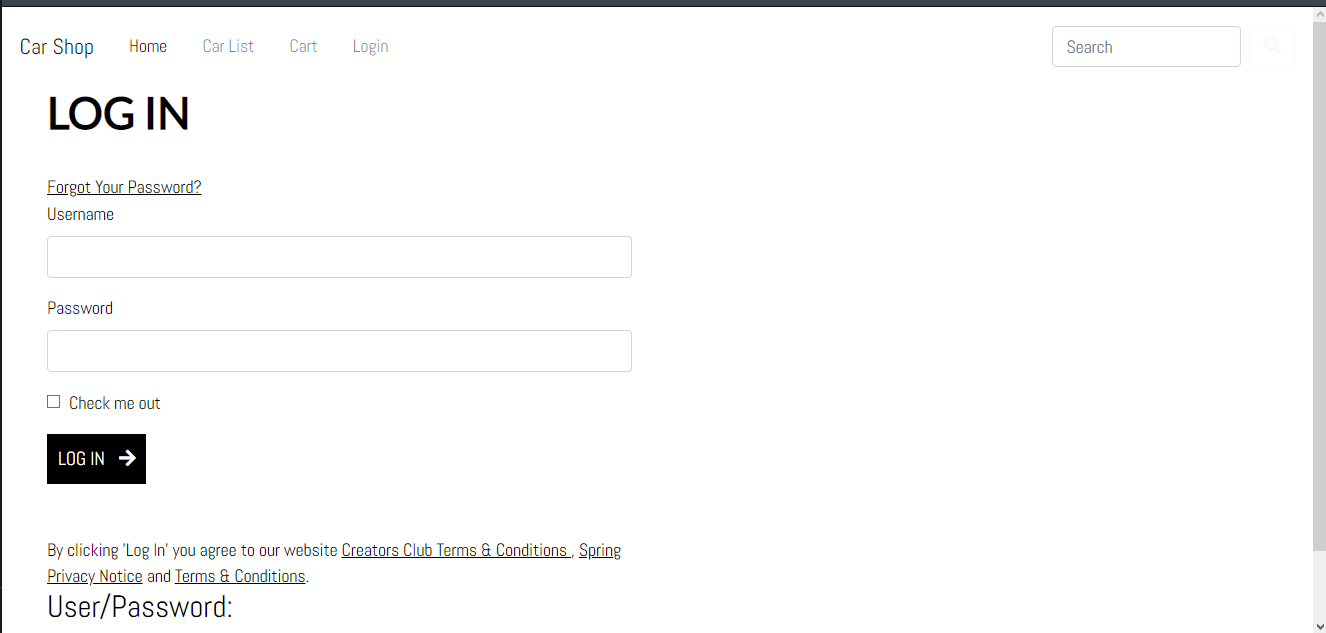


Рис. 8. Авторизація

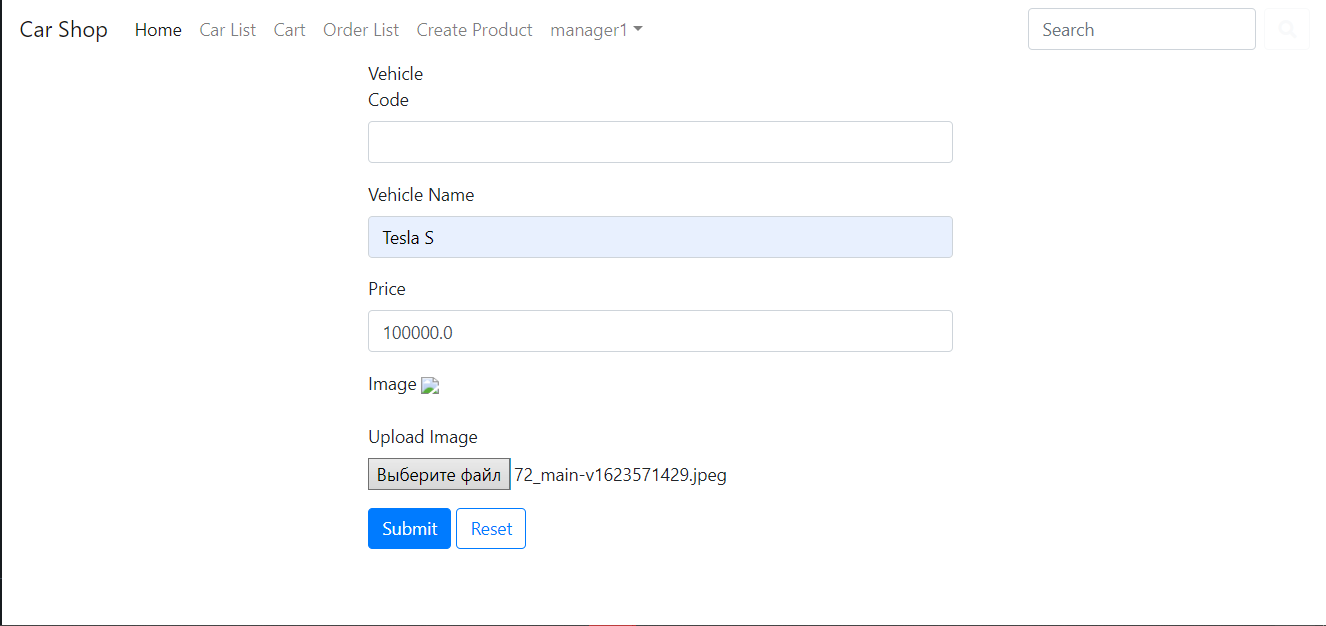


Рис. 9. Створення нового елементу каталогу

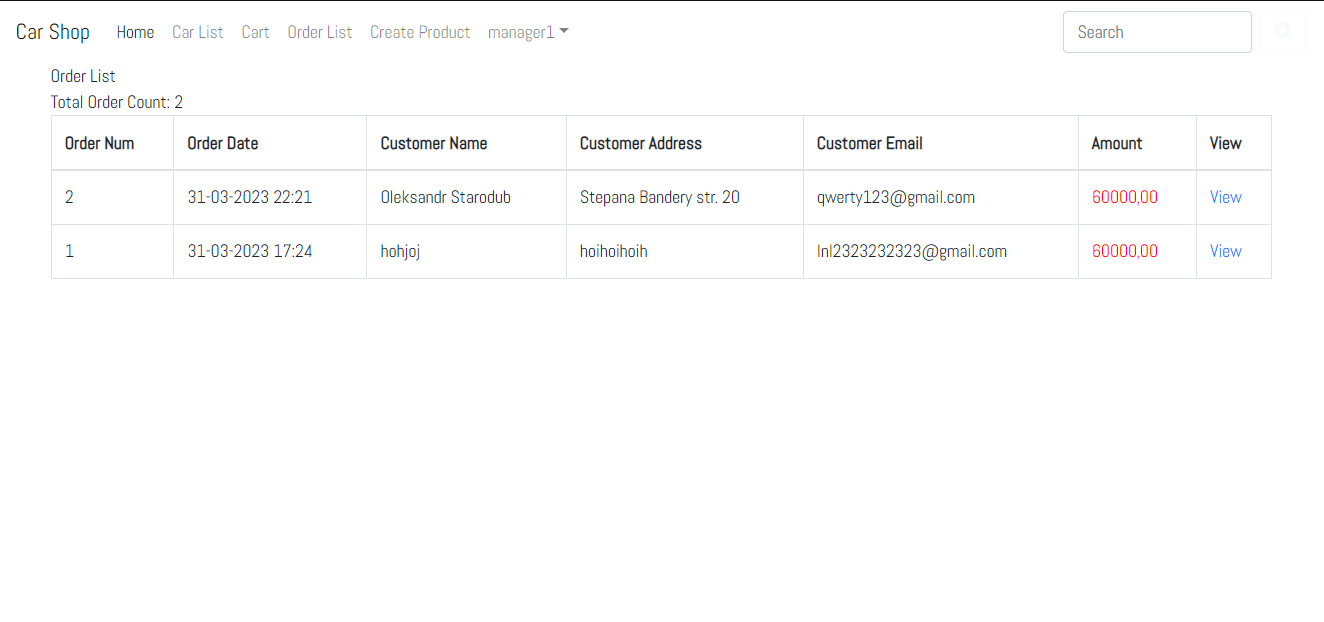


Рис. 10. Перегляд списку замовлень

### ВИСНОВКИ

Якщо розглядати висновки до курсової роботи з розробки інформаційної системи продажу авто, то основні пункти, які можна виділити, наступні:

* Розробка інформаційної системи продажу авто є складним та багатоетапним процесом, який вимагає від розробника високого рівня знань та досвіду в області веб-розробки, баз даних, архітектури та дизайну.
* В процесі розробки було створено інформаційну систему продажу авто, яка дозволяє користувачам швидко та зручно здійснювати пошук та покупку автомобілів.
* Під час розробки було використано ряд сучасних технологій та інструментів, включаючи мову програмування Java, фреймворк Spring, базу даних MySQL, сервер додатків Apache Tomcat, Thymeleaf, Bootstrap.
* Результатом розробки є повноцінна інформаційна система продажу авто, яка може бути використана як для комерційних, так і для особистих цілей.
* Наступним кроком може бути розширення функціональності системи, додавання нових функцій та модулів, а також покращення інтерфейсу користувача та оптимізація продуктивності системи.

### ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Патерни проектування [Електронний ресурс] – Режим доступу: <https://refactoring.guru/uk/design-patterns/factory-method>
2. for-each.dev. (2021). The Persistence Layer with Spring Data JPA. [Оnline]. Режим доступу: <https://for-each.dev/lessons/b/-the-persistence-layer-with-spring-data-jpa/>
3. Java [Електронний ресурс] – Режим доступу:

https://uk.wikipedia.org/wiki/Java

1. IntelliJ IDEA [Електронний ресурс] – Режим доступу:

https://uk.wikipedia.org/wiki/IntelliJ\_IDEA

### ДОДАТОК A

**Скрипт Orders**

CREATE DATABASE IF NOT EXISTS `car\_dealership` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;  
USE `car\_dealership`;  
-- MySQL dump 10.13 Distrib 8.0.29, for Win64 (x86\_64)  
--  
-- Host: 127.0.0.1   
-- ------------------------------------------------------  
-- Server version 8.0.29  
  
/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;  
/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;  
/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;  
/\*!50503 SET NAMES utf8 \*/;  
/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;  
/\*!40103 SET TIME\_ZONE='+00:00' \*/;  
/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;  
/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;  
/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;  
/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;  
  
--  
-- Table structure for table `orders`  
--  
  
DROP TABLE IF EXISTS `orders`;  
/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  
/\*!50503 SET character\_set\_client = utf8mb4 \*/;  
CREATE TABLE `orders` (  
 `ID` varchar(50) CHARACTER SET latin1 COLLATE latin1\_general\_ci NOT NULL,  
 `AMOUNT` double NOT NULL,  
 `CUSTOMER\_ADDRESS` varchar(255) CHARACTER SET latin1 COLLATE latin1\_general\_ci NOT NULL,  
 `CUSTOMER\_EMAIL` varchar(128) CHARACTER SET latin1 COLLATE latin1\_general\_ci NOT NULL,  
 `CUSTOMER\_NAME` varchar(255) CHARACTER SET latin1 COLLATE latin1\_general\_ci NOT NULL,  
 `CUSTOMER\_PHONE` varchar(128) CHARACTER SET latin1 COLLATE latin1\_general\_ci NOT NULL,  
 `ORDER\_DATE` datetime NOT NULL,  
 `ORDER\_NUM` int NOT NULL,  
 PRIMARY KEY (`ID`),  
 UNIQUE KEY `ORDER\_UK` (`ORDER\_NUM`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1\_general\_ci;  
/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;  
  
--  
-- Dumping data for table `orders`  
--  
  
LOCK TABLES `orders` WRITE;  
/\*!40000 ALTER TABLE `orders` DISABLE KEYS \*/;  
/\*!40000 ALTER TABLE `orders` ENABLE KEYS \*/;  
UNLOCK TABLES;  
/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;  
  
/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;  
/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;  
/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;  
/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;  
/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;  
/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;  
/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;  
  
-- Dump completed on 2023-03-28 23:50:39

**Скрипт Accounts**

CREATE DATABASE IF NOT EXISTS `car\_dealership` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;  
USE `car\_dealership`;  
-- MySQL dump 10.13 Distrib 8.0.29, for Win64 (x86\_64)  
--  
-- Host: 127.0.0.1 Database: spring\_shoppingcartdb2  
-- ------------------------------------------------------  
-- Server version 8.0.29  
  
/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;  
/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;  
/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;  
/\*!50503 SET NAMES utf8 \*/;  
/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;  
/\*!40103 SET TIME\_ZONE='+00:00' \*/;  
/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;  
/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;  
/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;  
/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;  
  
--  
-- Table structure for table `accounts`  
--  
  
DROP TABLE IF EXISTS `accounts`;  
/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  
/\*!50503 SET character\_set\_client = utf8mb4 \*/;  
CREATE TABLE `accounts` (  
 `USER\_NAME` varchar(20) CHARACTER SET latin1 COLLATE latin1\_general\_ci NOT NULL,  
 `ACTIVE` bit(1) NOT NULL,  
 `ENCRYTED\_PASSWORD` varchar(128) CHARACTER SET latin1 COLLATE latin1\_general\_ci NOT NULL,  
 `USER\_ROLE` varchar(20) CHARACTER SET latin1 COLLATE latin1\_general\_ci NOT NULL,  
 PRIMARY KEY (`USER\_NAME`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1\_general\_ci;  
/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;  
  
--  
-- Dumping data for table `accounts`  
--  
  
LOCK TABLES `accounts` WRITE;  
/\*!40000 ALTER TABLE `accounts` DISABLE KEYS \*/;  
INSERT INTO `accounts` VALUES ('employee1',\_binary '','$2a$10$PrI5Gk9L.tSZiW9FXhTS8O8Mz9E97k2FZbFvGFFaSsiTUIl.TCrFu','ROLE\_EMPLOYEE'),('manager1',\_binary '','$2a$10$PrI5Gk9L.tSZiW9FXhTS8O8Mz9E97k2FZbFvGFFaSsiTUIl.TCrFu','ROLE\_MANAGER');  
/\*!40000 ALTER TABLE `accounts` ENABLE KEYS \*/;  
UNLOCK TABLES;  
/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;  
  
/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;  
/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;  
/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;  
/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;  
/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;  
/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;  
/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;  
  
-- Dump completed on 2023-03-28 23:50:39

**Cкрипт Vehicle**

CREATE DATABASE IF NOT EXISTS `car\_dealership` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;  
USE `car\_dealership`;  
-- MySQL dump 10.13 Distrib 8.0.29, for Win64 (x86\_64)  
--  
-- Host: 127.0.0.1   
-- ------------------------------------------------------  
-- Server version 8.0.29  
  
/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;  
/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;  
/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;  
/\*!50503 SET NAMES utf8 \*/;  
/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;  
/\*!40103 SET TIME\_ZONE='+00:00' \*/;  
/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;  
/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;  
/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;  
/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;  
  
--  
-- Table structure for table `vehicle`  
--  
  
DROP TABLE IF EXISTS `vehicle`;  
/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  
/\*!50503 SET character\_set\_client = utf8mb4 \*/;  
CREATE TABLE `vehicle` (  
 `CODE` varchar(20) CHARACTER SET latin1 COLLATE latin1\_general\_ci NOT NULL,  
 `IMAGE` longblob,  
 `MODEL` varchar(255) COLLATE latin1\_general\_ci NOT NULL,  
 `PRICE` double NOT NULL,  
 `CREATE\_DATE` datetime NOT NULL,  
 `MANUFACTURER` varchar(50) COLLATE latin1\_general\_ci DEFAULT NULL,  
 PRIMARY KEY (`CODE`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1\_general\_ci;  
/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;  
  
--  
-- Dumping data for table `vehicle`  
--  
  
LOCK TABLES `vehicle` WRITE;  
/\*!40000 ALTER TABLE `vehicle` DISABLE KEYS \*/;  
/\*!40000 ALTER TABLE `vehicle` ENABLE KEYS \*/;  
UNLOCK TABLES;  
/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;  
  
/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;  
/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;  
/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;  
/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;  
/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;  
/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;  
/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;  
  
-- Dump completed on 2023-03-28 23:50:39

**Скрипт Order\_details**

CREATE DATABASE IF NOT EXISTS `car\_dealership` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;  
USE `car\_dealership`;  
-- MySQL dump 10.13 Distrib 8.0.29, for Win64 (x86\_64)  
--  
-- Host: 127.0.0.1 Database: spring\_shoppingcartdb2  
-- ------------------------------------------------------  
-- Server version 8.0.29  
  
/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;  
/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;  
/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;  
/\*!50503 SET NAMES utf8 \*/;  
/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;  
/\*!40103 SET TIME\_ZONE='+00:00' \*/;  
/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;  
/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;  
/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;  
/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;  
  
--  
-- Table structure for table `order\_details`  
--  
  
DROP TABLE IF EXISTS `order\_details`;  
/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  
/\*!50503 SET character\_set\_client = utf8mb4 \*/;  
CREATE TABLE `order\_details` (  
 `ID` varchar(50) CHARACTER SET latin1 COLLATE latin1\_general\_ci NOT NULL,  
 `AMOUNT` double NOT NULL,  
 `PRICE` double NOT NULL,  
 `QUANITY` int NOT NULL,  
 `ORDER\_ID` varchar(50) CHARACTER SET latin1 COLLATE latin1\_general\_ci NOT NULL,  
 `PRODUCT\_ID` varchar(20) CHARACTER SET latin1 COLLATE latin1\_general\_ci NOT NULL,  
 PRIMARY KEY (`ID`),  
 KEY `ORDER\_DETAIL\_ORD\_FK` (`ORDER\_ID`),  
 KEY `ORDER\_DETAIL\_PROD\_FK` (`PRODUCT\_ID`),  
 CONSTRAINT `ORDER\_DETAIL\_ORD\_FK` FOREIGN KEY (`ORDER\_ID`) REFERENCES `orders` (`ID`),  
 CONSTRAINT `ORDER\_DETAIL\_PROD\_FK` FOREIGN KEY (`PRODUCT\_ID`) REFERENCES `vehicle` (`CODE`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1\_general\_ci;  
/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;  
  
--  
-- Dumping data for table `order\_details`  
--  
  
LOCK TABLES `order\_details` WRITE;  
/\*!40000 ALTER TABLE `order\_details` DISABLE KEYS \*/;  
/\*!40000 ALTER TABLE `order\_details` ENABLE KEYS \*/;  
UNLOCK TABLES;  
/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;  
  
/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;  
/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;  
/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;  
/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;  
/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;  
/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;  
/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;  
  
-- Dump completed on 2023-03-28 23:50:39

### ДОДАТОК B

**Клас WebConfiguration**

package com.example.demo.config;  
  
import org.springframework.context.MessageSource;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.context.support.ReloadableResourceBundleMessageSource;  
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
  
@Configuration  
public class WebConfiguration implements WebMvcConfigurer {  
   
 @Bean  
 public MessageSource messageSource() {  
 ReloadableResourceBundleMessageSource messageSource = new ReloadableResourceBundleMessageSource();  
 // Load file: validation.properties  
 messageSource.setBasename("classpath:validation");  
 messageSource.setDefaultEncoding("UTF-8");  
 return messageSource;  
 }  
   
}

**Клас WebSecurityConfig**

package com.example.demo.config;  
  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
  
import com.example.demo.service.UserDetailsServiceImpl;  
  
@Configuration  
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  
 @Autowired  
 UserDetailsServiceImpl userDetailsService;  
   
 @Bean  
 public BCryptPasswordEncoder passwordEncoder() {  
 BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();  
 return bCryptPasswordEncoder;  
 }  
   
 @Autowired  
 public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {  
   
 // Setting Service to find User in the database.  
 // And Setting PassswordEncoder  
 auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());  
   
 }  
   
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
   
 http.csrf().disable();  
   
 // Requires login with role ROLE\_EMPLOYEE or ROLE\_MANAGER.  
 // If not, it will redirect to /admin/login.  
 http.authorizeRequests().antMatchers("/admin/orderList", "/admin/order", "/admin/accountInfo")//  
 .access("hasAnyRole('ROLE\_EMPLOYEE', 'ROLE\_MANAGER')");  
   
 // Pages only for MANAGER  
 http.authorizeRequests().antMatchers("/admin/product").access("hasRole('ROLE\_MANAGER')");  
   
 // When user login, role XX.  
 // But access to the page requires the YY role,  
 // An AccessDeniedException will be thrown.  
 http.authorizeRequests().and().exceptionHandling().accessDeniedPage("/403");  
   
 // Configuration for Login Form.  
 http.authorizeRequests().and().formLogin()//  
   
 //  
 .loginProcessingUrl("/j\_spring\_security\_check") // Submit URL  
 .loginPage("/admin/login")//  
 .defaultSuccessUrl("/admin/accountInfo")//  
 .failureUrl("/admin/login?error=true")//  
 .usernameParameter("userName")//  
 .passwordParameter("password")  
   
 // Configuration for the Logout page.  
 // (After logout, go to home page)  
 .and().logout().logoutUrl("/admin/logout").logoutSuccessUrl("/");  
   
 }  
}

**Клас Account**

package com.example.demo.entity;  
  
import java.io.Serializable;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Table(name = "Accounts")  
public class Account implements Serializable {  
   
 private static final long *serialVersionUID* = -2054386655979281969L;  
   
 public static final String *ROLE\_MANAGER* = "MANAGER";  
 public static final String *ROLE\_EMPLOYEE* = "EMPLOYEE";  
   
 @Id  
 @Column(name = "User\_Name", length = 20, nullable = false)  
 private String userName;  
   
 @Column(name = "Encryted\_Password", length = 128, nullable = false)  
 private String encrytedPassword;  
   
 @Column(name = "Active", length = 1, nullable = false)  
 private boolean active;  
   
 @Column(name = "User\_Role", length = 20, nullable = false)  
 private String userRole;  
   
 public String getUserName() {  
 return userName;  
 }  
   
 public void setUserName(String userName) {  
 this.userName = userName;  
 }  
   
 public String getEncrytedPassword() {  
 return encrytedPassword;  
 }  
   
 public void setEncrytedPassword(String encrytedPassword) {  
 this.encrytedPassword = encrytedPassword;  
 }  
   
 public boolean isActive() {  
 return active;  
 }  
   
 public void setActive(boolean active) {  
 this.active = active;  
 }  
   
 public String getUserRole() {  
 return userRole;  
 }  
   
 public void setUserRole(String userRole) {  
 this.userRole = userRole;  
 }  
   
 @Override  
 public String toString() {  
 return "[" + this.userName + "," + this.encrytedPassword + "," + this.userRole + "]";  
 }  
   
}

**Клас Vehicle**

package com.example.demo.entity;  
  
import java.io.Serializable;  
import java.util.Date;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.Lob;  
import javax.persistence.Table;  
import javax.persistence.Temporal;  
import javax.persistence.TemporalType;  
  
@Entity  
@Table(name = "Vehicle")  
public class Vehicle implements Serializable {  
   
 private static final long *serialVersionUID* = -1000119078147252957L;  
   
 @Id  
 @Column(name = "Code", length = 20, nullable = false)  
 private String code;  
   
 @Column(name = "MODEL", length = 255, nullable = false)  
 private String name;  
   
 @Column(name = "Price", nullable = false)  
 private double price;  
   
 @Lob  
 @Column(name = "Image", length = Integer.*MAX\_VALUE*, nullable = true)  
 private byte[] image;  
   
 @Temporal(TemporalType.*TIMESTAMP*)  
 @Column(name = "Create\_Date", nullable = false)  
 private Date createDate;  
   
 public Vehicle() {  
 }  
   
 public String getCode() {  
 return code;  
 }  
   
 public void setCode(String code) {  
 this.code = code;  
 }  
   
 public String getName() {  
 return name;  
 }  
   
 public void setName(String name) {  
 this.name = name;  
 }  
   
 public double getPrice() {  
 return price;  
 }  
   
 public void setPrice(double price) {  
 this.price = price;  
 }  
   
 public Date getCreateDate() {  
 return createDate;  
 }  
   
 public void setCreateDate(Date createDate) {  
 this.createDate = createDate;  
 }  
   
 public byte[] getImage() {  
 return image;  
 }  
   
 public void setImage(byte[] image) {  
 this.image = image;  
 }  
   
}

**Клас Order**

package com.example.demo.entity;  
  
import java.io.Serializable;  
import java.util.Date;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.Table;  
import javax.persistence.UniqueConstraint;  
  
@Entity  
@Table(name = "Orders", //  
 uniqueConstraints = { @UniqueConstraint(columnNames = "Order\_Num") })  
public class Order implements Serializable {  
   
 private static final long *serialVersionUID* = -2576670215015463100L;  
   
 @Id  
 @Column(name = "ID", length = 50)  
 private String id;  
   
 @Column(name = "Order\_Date", nullable = false)  
 private Date orderDate;  
   
 @Column(name = "Order\_Num", nullable = false)  
 private int orderNum;  
   
 @Column(name = "Amount", nullable = false)  
 private double amount;  
   
 @Column(name = "Customer\_Name", length = 255, nullable = false)  
 private String customerName;  
   
 @Column(name = "Customer\_Address", length = 255, nullable = false)  
 private String customerAddress;  
   
 @Column(name = "Customer\_Email", length = 128, nullable = false)  
 private String customerEmail;  
   
 @Column(name = "Customer\_Phone", length = 128, nullable = false)  
 private String customerPhone;  
   
 public String getId() {  
 return id;  
 }  
   
 public void setId(String id) {  
 this.id = id;  
 }  
   
 public Date getOrderDate() {  
 return orderDate;  
 }  
   
 public void setOrderDate(Date orderDate) {  
 this.orderDate = orderDate;  
 }  
   
 public int getOrderNum() {  
 return orderNum;  
 }  
   
 public void setOrderNum(int orderNum) {  
 this.orderNum = orderNum;  
 }  
   
 public double getAmount() {  
 return amount;  
 }  
   
 public void setAmount(double amount) {  
 this.amount = amount;  
 }  
   
 public String getCustomerName() {  
 return customerName;  
 }  
   
 public void setCustomerName(String customerName) {  
 this.customerName = customerName;  
 }  
   
 public String getCustomerAddress() {  
 return customerAddress;  
 }  
   
 public void setCustomerAddress(String customerAddress) {  
 this.customerAddress = customerAddress;  
 }  
   
 public String getCustomerEmail() {  
 return customerEmail;  
 }  
   
 public void setCustomerEmail(String customerEmail) {  
 this.customerEmail = customerEmail;  
 }  
   
 public String getCustomerPhone() {  
 return customerPhone;  
 }  
   
 public void setCustomerPhone(String customerPhone) {  
 this.customerPhone = customerPhone;  
 }  
   
}

**Клас OrderDeatail**

package com.example.demo.entity;  
  
import java.io.Serializable;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.ForeignKey;  
import javax.persistence.ManyToOne;  
import javax.persistence.Table;  
  
@Entity  
@Table(name = "Order\_Details")  
public class OrderDetail implements Serializable {  
   
 private static final long *serialVersionUID* = 7550745928843183535L;  
   
 @Id  
 @Column(name = "ID", length = 50, nullable = false)  
 private String id;  
   
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "ORDER\_ID", nullable = false, //  
 foreignKey = @ForeignKey(name = "ORDER\_DETAIL\_ORD\_FK"))  
 private Order order;  
   
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "PRODUCT\_ID", nullable = false, //  
 foreignKey = @ForeignKey(name = "ORDER\_DETAIL\_PROD\_FK"))  
 private Vehicle vehicle;  
   
 @Column(name = "Quanity", nullable = false)  
 private int quanity;  
   
 @Column(name = "Price", nullable = false)  
 private double price;  
   
 @Column(name = "Amount", nullable = false)  
 private double amount;  
   
 public String getId() {  
 return id;  
 }  
   
 public void setId(String id) {  
 this.id = id;  
 }  
   
 public Order getOrder() {  
 return order;  
 }  
   
 public void setOrder(Order order) {  
 this.order = order;  
 }  
   
 public Vehicle getProduct() {  
 return vehicle;  
 }  
   
 public void setProduct(Vehicle vehicle) {  
 this.vehicle = vehicle;  
 }  
   
 public int getQuanity() {  
 return quanity;  
 }  
   
 public void setQuanity(int quanity) {  
 this.quanity = quanity;  
 }  
   
 public double getPrice() {  
 return price;  
 }  
   
 public void setPrice(double price) {  
 this.price = price;  
 }  
   
 public double getAmount() {  
 return amount;  
 }  
   
 public void setAmount(double amount) {  
 this.amount = amount;  
 }  
   
}

**Клас CustomerForm**

package com.example.demo.form;  
  
import com.example.demo.model.CustomerInfo;  
  
public class CustomerForm {  
   
 private String name;  
 private String address;  
 private String email;  
 private String phone;  
   
 private boolean valid;  
   
 public CustomerForm() {  
   
 }  
   
 public CustomerForm(CustomerInfo customerInfo) {  
 if (customerInfo != null) {  
 this.name = customerInfo.getName();  
 this.address = customerInfo.getAddress();  
 this.email = customerInfo.getEmail();  
 this.phone = customerInfo.getPhone();  
 }  
 }  
   
 public String getName() {  
 return name;  
 }  
   
 public void setName(String name) {  
 this.name = name;  
 }  
   
 public String getEmail() {  
 return email;  
 }  
   
 public void setEmail(String email) {  
 this.email = email;  
 }  
   
 public String getAddress() {  
 return address;  
 }  
   
 public void setAddress(String address) {  
 this.address = address;  
 }  
   
 public String getPhone() {  
 return phone;  
 }  
   
 public void setPhone(String phone) {  
 this.phone = phone;  
 }  
   
 public boolean isValid() {  
 return valid;  
 }  
   
 public void setValid(boolean valid) {  
 this.valid = valid;  
 }  
   
}

**Клас VehicleForm**

package com.example.demo.form;  
  
import com.example.demo.entity.Vehicle;  
import org.springframework.web.multipart.MultipartFile;  
  
public class VehicleForm {  
 private String code;  
 private String name;  
 private double price;  
   
 private boolean newProduct = false;  
   
 // Upload file.  
 private MultipartFile fileData;  
   
 public VehicleForm() {  
 this.newProduct= true;  
 }  
   
 public VehicleForm(Vehicle vehicle) {  
 this.code = vehicle.getCode();  
 this.name = vehicle.getName();  
 this.price = vehicle.getPrice();  
 }  
   
 public String getCode() {  
 return code;  
 }  
   
 public void setCode(String code) {  
 this.code = code;  
 }  
   
 public String getName() {  
 return name;  
 }  
   
 public void setName(String name) {  
 this.name = name;  
 }  
   
 public double getPrice() {  
 return price;  
 }  
   
 public void setPrice(double price) {  
 this.price = price;  
 }  
   
 public MultipartFile getFileData() {  
 return fileData;  
 }  
   
 public void setFileData(MultipartFile fileData) {  
 this.fileData = fileData;  
 }  
   
 public boolean isNewProduct() {  
 return newProduct;  
 }  
   
 public void setNewProduct(boolean newProduct) {  
 this.newProduct = newProduct;  
 }  
   
}

**Клас CartInfo**

package com.example.demo.model;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class CartInfo {  
   
 private int orderNum;  
   
 private CustomerInfo customerInfo;  
   
 private final List<CartLineInfo> cartLines = new ArrayList<CartLineInfo>();  
   
 public CartInfo() {  
   
 }  
   
 public int getOrderNum() {  
 return orderNum;  
 }  
   
 public void setOrderNum(int orderNum) {  
 this.orderNum = orderNum;  
 }  
   
 public CustomerInfo getCustomerInfo() {  
 return customerInfo;  
 }  
   
 public void setCustomerInfo(CustomerInfo customerInfo) {  
 this.customerInfo = customerInfo;  
 }  
   
 public List<CartLineInfo> getCartLines() {  
 return this.cartLines;  
 }  
   
 private CartLineInfo findLineByCode(String code) {  
 for (CartLineInfo line : this.cartLines) {  
 if (line.getProductInfo().getCode().equals(code)) {  
 return line;  
 }  
 }  
 return null;  
 }  
   
 public void addProduct(VehicleInfo vehicleInfo, int quantity) {  
 CartLineInfo line = this.findLineByCode(vehicleInfo.getCode());  
   
 if (line == null) {  
 line = new CartLineInfo();  
 line.setQuantity(0);  
 line.setProductInfo(vehicleInfo);  
 this.cartLines.add(line);  
 }  
 int newQuantity = line.getQuantity() + quantity;  
 if (newQuantity <= 0) {  
 this.cartLines.remove(line);  
 } else {  
 line.setQuantity(newQuantity);  
 }  
 }  
   
 public void validate() {  
   
 }  
   
 public void updateProduct(String code, int quantity) {  
 CartLineInfo line = this.findLineByCode(code);  
   
 if (line != null) {  
 if (quantity <= 0) {  
 this.cartLines.remove(line);  
 } else {  
 line.setQuantity(quantity);  
 }  
 }  
 }  
   
 public void removeProduct(VehicleInfo vehicleInfo) {  
 CartLineInfo line = this.findLineByCode(vehicleInfo.getCode());  
 if (line != null) {  
 this.cartLines.remove(line);  
 }  
 }  
   
 public boolean isEmpty() {  
 return this.cartLines.isEmpty();  
 }  
   
 public boolean isValidCustomer() {  
 return this.customerInfo != null && this.customerInfo.isValid();  
 }  
   
 public int getQuantityTotal() {  
 int quantity = 0;  
 for (CartLineInfo line : this.cartLines) {  
 quantity += line.getQuantity();  
 }  
 return quantity;  
 }  
   
 public double getAmountTotal() {  
 double total = 0;  
 for (CartLineInfo line : this.cartLines) {  
 total += line.getAmount();  
 }  
 return total;  
 }  
   
 public void updateQuantity(CartInfo cartForm) {  
 if (cartForm != null) {  
 List<CartLineInfo> lines = cartForm.getCartLines();  
 for (CartLineInfo line : lines) {  
 this.updateProduct(line.getProductInfo().getCode(), line.getQuantity());  
 }  
 }  
   
 }  
   
}

**Клас CartLineInfo**

package com.example.demo.model;  
  
public class CartLineInfo {  
   
 private VehicleInfo vehicleInfo;  
 private int quantity;  
   
 public CartLineInfo() {  
 this.quantity = 0;  
 }  
   
 public VehicleInfo getProductInfo() {  
 return vehicleInfo;  
 }  
   
 public void setProductInfo(VehicleInfo vehicleInfo) {  
 this.vehicleInfo = vehicleInfo;  
 }  
   
 public int getQuantity() {  
 return quantity;  
 }  
   
 public void setQuantity(int quantity) {  
 this.quantity = quantity;  
 }  
   
 public double getAmount() {  
 return this.vehicleInfo.getPrice() \* this.quantity;  
 }  
   
}

**Клас CustomerInfo**

package com.example.demo.model;  
  
import com.example.demo.form.CustomerForm;  
  
public class CustomerInfo {  
   
 private String name;  
 private String address;  
 private String email;  
 private String phone;  
   
 private boolean valid;  
   
 public CustomerInfo() {  
   
 }  
   
  
 public CustomerInfo(CustomerForm customerForm) {  
 this.name = customerForm.getName();  
 this.address = customerForm.getAddress();  
 this.email = customerForm.getEmail();  
 this.phone = customerForm.getPhone();  
 this.valid = customerForm.isValid();  
 }  
   
 public String getName() {  
 return name;  
 }  
   
 public void setName(String name) {  
 this.name = name;  
 }  
   
 public String getEmail() {  
 return email;  
 }  
   
 public void setEmail(String email) {  
 this.email = email;  
 }  
   
 public String getAddress() {  
 return address;  
 }  
   
 public void setAddress(String address) {  
 this.address = address;  
 }  
   
 public String getPhone() {  
 return phone;  
 }  
   
 public void setPhone(String phone) {  
 this.phone = phone;  
 }  
   
 public boolean isValid() {  
 return valid;  
 }  
   
 public void setValid(boolean valid) {  
 this.valid = valid;  
 }  
   
}

**Клас OrderDetailInfo**

package com.example.demo.model;  
  
public class OrderDetailInfo {  
 private String id;  
   
 private String productCode;  
 private String productName;  
   
 private int quanity;  
 private double price;  
 private double amount;  
   
 public OrderDetailInfo() {  
   
 }  
   
 // Using for JPA/Hibernate Query.  
 public OrderDetailInfo(String id, String productCode, //  
 String productName, int quanity, double price, double amount) {  
 this.id = id;  
 this.productCode = productCode;  
 this.productName = productName;  
 this.quanity = quanity;  
 this.price = price;  
 this.amount = amount;  
 }  
   
 public String getId() {  
 return id;  
 }  
   
 public void setId(String id) {  
 this.id = id;  
 }  
   
 public String getProductCode() {  
 return productCode;  
 }  
   
 public void setProductCode(String productCode) {  
 this.productCode = productCode;  
 }  
   
 public String getProductName() {  
 return productName;  
 }  
   
 public void setProductName(String productName) {  
 this.productName = productName;  
 }  
   
 public int getQuanity() {  
 return quanity;  
 }  
   
 public void setQuanity(int quanity) {  
 this.quanity = quanity;  
 }  
   
 public double getPrice() {  
 return price;  
 }  
   
 public void setPrice(double price) {  
 this.price = price;  
 }  
   
 public double getAmount() {  
 return amount;  
 }  
   
 public void setAmount(double amount) {  
 this.amount = amount;  
 }  
}

**Клас OrderInfo**

package com.example.demo.model;  
  
import java.util.Date;  
import java.util.List;  
   
public class OrderInfo {  
   
 private String id;  
 private Date orderDate;  
 private int orderNum;  
 private double amount;  
   
 private String customerName;  
 private String customerAddress;  
 private String customerEmail;  
 private String customerPhone;  
   
 private List<OrderDetailInfo> details;  
   
 public OrderInfo() {  
   
 }  
   
 // Using for Hibernate Query.  
 public OrderInfo(String id, Date orderDate, int orderNum, //  
 double amount, String customerName, String customerAddress, //  
 String customerEmail, String customerPhone) {  
 this.id = id;  
 this.orderDate = orderDate;  
 this.orderNum = orderNum;  
 this.amount = amount;  
   
 this.customerName = customerName;  
 this.customerAddress = customerAddress;  
 this.customerEmail = customerEmail;  
 this.customerPhone = customerPhone;  
 }  
   
 public String getId() {  
 return id;  
 }  
   
 public void setId(String id) {  
 this.id = id;  
 }  
   
 public Date getOrderDate() {  
 return orderDate;  
 }  
   
 public void setOrderDate(Date orderDate) {  
 this.orderDate = orderDate;  
 }  
   
 public int getOrderNum() {  
 return orderNum;  
 }  
   
 public void setOrderNum(int orderNum) {  
 this.orderNum = orderNum;  
 }  
   
 public double getAmount() {  
 return amount;  
 }  
   
 public void setAmount(double amount) {  
 this.amount = amount;  
 }  
   
 public String getCustomerName() {  
 return customerName;  
 }  
   
 public void setCustomerName(String customerName) {  
 this.customerName = customerName;  
 }  
   
 public String getCustomerAddress() {  
 return customerAddress;  
 }  
   
 public void setCustomerAddress(String customerAddress) {  
 this.customerAddress = customerAddress;  
 }  
   
 public String getCustomerEmail() {  
 return customerEmail;  
 }  
   
 public void setCustomerEmail(String customerEmail) {  
 this.customerEmail = customerEmail;  
 }  
   
 public String getCustomerPhone() {  
 return customerPhone;  
 }  
   
 public void setCustomerPhone(String customerPhone) {  
 this.customerPhone = customerPhone;  
 }  
   
 public List<OrderDetailInfo> getDetails() {  
 return details;  
 }  
   
 public void setDetails(List<OrderDetailInfo> details) {  
 this.details = details;  
 }  
   
}

**Клас VehicleInfo**

package com.example.demo.model;  
  
import com.example.demo.entity.Vehicle;  
  
public class VehicleInfo {  
 private String code;  
 private String name;  
 private double price;  
   
 public VehicleInfo() {  
 }  
   
 public VehicleInfo(Vehicle vehicle) {  
 this.code = vehicle.getCode();  
 this.name = vehicle.getName();  
 this.price = vehicle.getPrice();  
 }  
   
 // Using in JPA/Hibernate query  
 public VehicleInfo(String code, String name, double price) {  
 this.code = code;  
 this.name = name;  
 this.price = price;  
 }  
   
 public String getCode() {  
 return code;  
 }  
   
 public void setCode(String code) {  
 this.code = code;  
 }  
   
 public String getName() {  
 return name;  
 }  
   
 public void setName(String name) {  
 this.name = name;  
 }  
   
 public double getPrice() {  
 return price;  
 }  
   
 public void setPrice(double price) {  
 this.price = price;  
 }  
   
}

**Клас AccountDao**

package com.example.demo.dao;  
  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Repository;  
import org.springframework.transaction.annotation.Transactional;  
  
import com.example.demo.entity.Account;  
  
@Transactional  
@Repository  
public class AccountDAO {  
   
 @Autowired  
 private SessionFactory sessionFactory;  
   
 public Account findAccount(String userName) {  
 Session session = this.sessionFactory.getCurrentSession();  
 return session.find(Account.class, userName);  
 }  
   
}

**Клас OrderDao**

package com.example.demo.dao;  
  
import java.util.Date;  
import java.util.List;  
import java.util.UUID;  
  
import com.example.demo.entity.Vehicle;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.query.Query;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Repository;  
import org.springframework.transaction.annotation.Transactional;  
  
import com.example.demo.entity.Order;  
import com.example.demo.entity.OrderDetail;  
  
import com.example.demo.model.CartInfo;  
import com.example.demo.model.CartLineInfo;  
import com.example.demo.model.CustomerInfo;  
import com.example.demo.model.OrderDetailInfo;  
import com.example.demo.model.OrderInfo;  
import com.example.demo.pagination.PaginationResult;  
  
@Transactional  
@Repository  
public class OrderDAO {  
   
 @Autowired  
 private SessionFactory sessionFactory;  
   
 @Autowired  
 private VehicleDAO vehicleDAO;  
   
 private int getMaxOrderNum() {  
 String sql = "Select max(o.orderNum) from " + Order.class.getName() + " o ";  
 Session session = this.sessionFactory.getCurrentSession();  
 Query<Integer> query = session.createQuery(sql, Integer.class);  
 Integer value = (Integer) query.getSingleResult();  
 if (value == null) {  
 return 0;  
 }  
 return value;  
 }  
   
 @Transactional  
 public void saveOrder(CartInfo cartInfo) {  
 Session session = this.sessionFactory.getCurrentSession();  
   
 int orderNum = this.getMaxOrderNum() + 1;  
 Order order = new Order();  
   
 order.setId(UUID.*randomUUID*().toString());  
 order.setOrderNum(orderNum);  
 order.setOrderDate(new Date());  
 order.setAmount(cartInfo.getAmountTotal());  
   
 CustomerInfo customerInfo = cartInfo.getCustomerInfo();  
 order.setCustomerName(customerInfo.getName());  
 order.setCustomerEmail(customerInfo.getEmail());  
 order.setCustomerPhone(customerInfo.getPhone());  
 order.setCustomerAddress(customerInfo.getAddress());  
   
 session.persist(order);  
   
 List<CartLineInfo> lines = cartInfo.getCartLines();  
   
 for (CartLineInfo line : lines) {  
 OrderDetail detail = new OrderDetail();  
 detail.setId(UUID.*randomUUID*().toString());  
 detail.setOrder(order);  
 detail.setAmount(line.getAmount());  
 detail.setPrice(line.getProductInfo().getPrice());  
 detail.setQuanity(line.getQuantity());  
   
 String code = line.getProductInfo().getCode();  
 Vehicle vehicle = this.vehicleDAO.findVehicle(code);  
 detail.setProduct(vehicle);  
   
 session.persist(detail);  
 }  
   
 // Order Number!  
 cartInfo.setOrderNum(orderNum);  
 // Flush  
 session.flush();  
 }  
   
 // @page = 1, 2, ...  
 public PaginationResult<OrderInfo> listOrderInfo(int page, int maxResult, int maxNavigationPage) {  
 String sql = "Select new " + OrderInfo.class.getName()//  
 + "(ord.id, ord.orderDate, ord.orderNum, ord.amount, "  
 + " ord.customerName, ord.customerAddress, ord.customerEmail, ord.customerPhone) " + " from "  
 + Order.class.getName() + " ord "//  
 + " order by ord.orderNum desc";  
   
 Session session = this.sessionFactory.getCurrentSession();  
 Query<OrderInfo> query = session.createQuery(sql, OrderInfo.class);  
 return new PaginationResult<OrderInfo>(query, page, maxResult, maxNavigationPage);  
 }  
   
 public Order findOrder(String orderId) {  
 Session session = this.sessionFactory.getCurrentSession();  
 return session.find(Order.class, orderId);  
 }  
   
 public OrderInfo getOrderInfo(String orderId) {  
 Order order = this.findOrder(orderId);  
 if (order == null) {  
 return null;  
 }  
 return new OrderInfo(order.getId(), order.getOrderDate(), //  
 order.getOrderNum(), order.getAmount(), order.getCustomerName(), //  
 order.getCustomerAddress(), order.getCustomerEmail(), order.getCustomerPhone());  
 }  
   
 public List<OrderDetailInfo> listOrderDetailInfos(String orderId) {  
 String sql = "Select new " + OrderDetailInfo.class.getName() //  
 + "(d.id, d.product.code, d.product.name , d.quanity,d.price,d.amount) "//  
 + " from " + OrderDetail.class.getName() + " d "//  
 + " where d.order.id = :orderId ";  
   
 Session session = this.sessionFactory.getCurrentSession();  
 Query<OrderDetailInfo> query = session.createQuery(sql, OrderDetailInfo.class);  
 query.setParameter("orderId", orderId);  
   
 return query.getResultList();  
 }  
   
}

**Клас VehicleDao**

package com.example.demo.dao;  
  
import java.io.IOException;  
import java.util.Date;  
  
import javax.persistence.NoResultException;  
  
import com.example.demo.entity.Vehicle;  
import com.example.demo.form.VehicleForm;  
import com.example.demo.model.VehicleInfo;  
import org.hibernate.query.Query;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Repository;  
import org.springframework.transaction.annotation.Propagation;  
import org.springframework.transaction.annotation.Transactional;  
  
import com.example.demo.pagination.PaginationResult;  
  
@Transactional  
@Repository  
public class VehicleDAO {  
   
 @Autowired  
 private SessionFactory sessionFactory;  
   
 public Vehicle findVehicle(String code) {  
 try {  
 String sql = "Select e from " + Vehicle.class.getName() + " e Where e.code =:code ";  
   
 Session session = this.sessionFactory.getCurrentSession();  
 Query<Vehicle> query = session.createQuery(sql, Vehicle.class);  
 query.setParameter("code", code);  
 return (Vehicle) query.getSingleResult();  
 } catch (NoResultException e) {  
 return null;  
 }  
 }  
   
 public VehicleInfo findProductInfo(String code) {  
 Vehicle vehicle = this.findVehicle(code);  
 if (vehicle == null) {  
 return null;  
 }  
 return new VehicleInfo(vehicle.getCode(), vehicle.getName(), vehicle.getPrice());  
 }  
   
 @Transactional(propagation = Propagation.*REQUIRES\_NEW*, rollbackFor = Exception.class)  
 public void save(VehicleForm vehicleForm) {  
   
 Session session = this.sessionFactory.getCurrentSession();  
 String code = vehicleForm.getCode();  
   
 Vehicle vehicle = null;  
   
 boolean isNew = false;  
 if (code != null) {  
 vehicle = this.findVehicle(code);  
 }  
 if (vehicle == null) {  
 isNew = true;  
 vehicle = new Vehicle();  
 vehicle.setCreateDate(new Date());  
 }  
 vehicle.setCode(code);  
 vehicle.setName(vehicleForm.getName());  
 vehicle.setPrice(vehicleForm.getPrice());  
   
 if (vehicleForm.getFileData() != null) {  
 byte[] image = null;  
 try {  
 image = vehicleForm.getFileData().getBytes();  
 } catch (IOException e) {  
 }  
 if (image != null && image.length > 0) {  
 vehicle.setImage(image);  
 }  
 }  
 if (isNew) {  
 session.persist(vehicle);  
 }  
 // If error in DB, Exceptions will be thrown out immediately  
 session.flush();  
 }  
   
 public PaginationResult<VehicleInfo> queryProducts(int page, int maxResult, int maxNavigationPage,  
 String likeName) {  
 String sql = "Select new " + VehicleInfo.class.getName() //  
 + "(p.code, p.name, p.price) " + " from "//  
 + Vehicle.class.getName() + " p ";  
 if (likeName != null && likeName.length() > 0) {  
 sql += " Where lower(p.name) like :likeName ";  
 }  
 sql += " order by p.createDate desc ";  
 //   
 Session session = this.sessionFactory.getCurrentSession();  
 Query<VehicleInfo> query = session.createQuery(sql, VehicleInfo.class);  
   
 if (likeName != null && likeName.length() > 0) {  
 query.setParameter("likeName", "%" + likeName.toLowerCase() + "%");  
 }  
 return new PaginationResult<VehicleInfo>(query, page, maxResult, maxNavigationPage);  
 }  
   
 public PaginationResult<VehicleInfo> queryProducts(int page, int maxResult, int maxNavigationPage) {  
 return queryProducts(page, maxResult, maxNavigationPage, null);  
 }  
   
}

**Клас UserDeatailsServiceImpl**

package com.example.demo.service;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.User;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Service;  
  
import com.example.demo.dao.AccountDAO;  
import com.example.demo.entity.Account;  
  
@Service  
public class UserDetailsServiceImpl implements UserDetailsService{  
 @Autowired  
 private AccountDAO accountDAO;  
   
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 Account account = accountDAO.findAccount(username);  
 System.*out*.println("Account= " + account);  
   
 if (account == null) {  
 throw new UsernameNotFoundException("User " //  
 + username + " was not found in the database");  
 }  
   
 // EMPLOYEE,MANAGER,..  
 String role = account.getUserRole();  
   
 List<GrantedAuthority> grantList = new ArrayList<GrantedAuthority>();  
   
 // ROLE\_EMPLOYEE, ROLE\_MANAGER  
 GrantedAuthority authority = new SimpleGrantedAuthority(role);  
   
 grantList.add(authority);  
   
 boolean enabled = account.isActive();  
 boolean accountNonExpired = true;  
 boolean credentialsNonExpired = true;  
 boolean accountNonLocked = true;  
   
 UserDetails userDetails = (UserDetails) new User(account.getUserName(), //  
 account.getEncrytedPassword(), enabled, accountNonExpired, //  
 credentialsNonExpired, accountNonLocked, grantList);  
   
 return userDetails;  
 }  
}

**Клас AdminController**

package com.example.demo.controller;  
  
import java.util.List;  
  
import com.example.demo.entity.Vehicle;  
import com.example.demo.form.VehicleForm;  
import org.apache.commons.lang.exception.ExceptionUtils;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.stereotype.Controller;  
import org.springframework.transaction.annotation.Transactional;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.validation.annotation.Validated;  
import org.springframework.web.bind.WebDataBinder;  
import org.springframework.web.bind.annotation.InitBinder;  
import org.springframework.web.bind.annotation.ModelAttribute;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.servlet.mvc.support.RedirectAttributes;  
  
import com.example.demo.dao.OrderDAO;  
import com.example.demo.dao.VehicleDAO;  
import com.example.demo.model.OrderDetailInfo;  
import com.example.demo.model.OrderInfo;  
import com.example.demo.pagination.PaginationResult;  
import com.example.demo.validator.ProductFormValidator;  
  
@Controller  
@Transactional  
public class AdminController {  
   
 @Autowired  
 private OrderDAO orderDAO;  
   
 @Autowired  
 private VehicleDAO vehicleDAO;  
   
 @Autowired  
 private ProductFormValidator productFormValidator;  
   
 @InitBinder  
 public void myInitBinder(WebDataBinder dataBinder) {  
 Object target = dataBinder.getTarget();  
 if (target == null) {  
 return;  
 }  
 System.*out*.println("Target=" + target);  
   
 if (target.getClass() == VehicleForm.class) {  
 dataBinder.setValidator(productFormValidator);  
 }  
 }  
   
 // GET: Show Login Page  
 @RequestMapping(value = { "/admin/login" }, method = RequestMethod.*GET*)  
 public String login(Model model) {  
   
 return "login";  
 }  
   
 @RequestMapping(value = { "/admin/accountInfo" }, method = RequestMethod.*GET*)  
 public String accountInfo(Model model) {  
   
 UserDetails userDetails = (UserDetails) SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();  
 System.*out*.println(userDetails.getPassword());  
 System.*out*.println(userDetails.getUsername());  
 System.*out*.println(userDetails.isEnabled());  
   
 model.addAttribute("userDetails", userDetails);  
 return "accountInfo";  
 }  
   
 @RequestMapping(value = { "/admin/orderList" }, method = RequestMethod.*GET*)  
 public String orderList(Model model, //  
 @RequestParam(value = "page", defaultValue = "1") String pageStr) {  
 int page = 1;  
 try {  
 page = Integer.*parseInt*(pageStr);  
 } catch (Exception e) {  
 }  
 final int MAX\_RESULT = 5;  
 final int MAX\_NAVIGATION\_PAGE = 10;  
   
 PaginationResult<OrderInfo> paginationResult //  
 = orderDAO.listOrderInfo(page, MAX\_RESULT, MAX\_NAVIGATION\_PAGE);  
   
 model.addAttribute("paginationResult", paginationResult);  
 return "orderList";  
 }  
   
 // GET: Show product.  
 @RequestMapping(value = { "/admin/product" }, method = RequestMethod.*GET*)  
 public String product(Model model, @RequestParam(value = "code", defaultValue = "") String code) {  
 VehicleForm vehicleForm = null;  
   
 if (code != null && code.length() > 0) {  
 Vehicle vehicle = vehicleDAO.findVehicle(code);  
 if (vehicle != null) {  
 vehicleForm = new VehicleForm(vehicle);  
 }  
 }  
 if (vehicleForm == null) {  
 vehicleForm = new VehicleForm();  
 vehicleForm.setNewProduct(true);  
 }  
 model.addAttribute("productForm", vehicleForm);  
 return "product";  
 }  
   
 // POST: Save product  
 @RequestMapping(value = { "/admin/product" }, method = RequestMethod.*POST*)  
 public String productSave(Model model, //  
 @ModelAttribute("productForm") @Validated VehicleForm vehicleForm, //  
 BindingResult result, //  
 final RedirectAttributes redirectAttributes) {  
   
 if (result.hasErrors()) {  
 return "product";  
 }  
 try {  
 vehicleDAO.save(vehicleForm);  
 } catch (Exception e) {  
 Throwable rootCause = ExceptionUtils.*getRootCause*(e);  
 String message = rootCause.getMessage();  
 model.addAttribute("errorMessage", message);  
 // Show product form.  
 return "product";  
 }  
   
 return "redirect:/productList";  
 }  
   
 @RequestMapping(value = { "/admin/order" }, method = RequestMethod.*GET*)  
 public String orderView(Model model, @RequestParam("orderId") String orderId) {  
 OrderInfo orderInfo = null;  
 if (orderId != null) {  
 orderInfo = this.orderDAO.getOrderInfo(orderId);  
 }  
 if (orderInfo == null) {  
 return "redirect:/admin/orderList";  
 }  
 List<OrderDetailInfo> details = this.orderDAO.listOrderDetailInfos(orderId);  
 orderInfo.setDetails(details);  
   
 model.addAttribute("orderInfo", orderInfo);  
   
 return "order";  
 }  
   
}

**Клас MainController**

package com.example.demo.controller;  
  
import java.io.IOException;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
import com.example.demo.entity.Vehicle;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.transaction.annotation.Transactional;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.validation.annotation.Validated;  
import org.springframework.web.bind.WebDataBinder;  
import org.springframework.web.bind.annotation.InitBinder;  
import org.springframework.web.bind.annotation.ModelAttribute;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.servlet.mvc.support.RedirectAttributes;  
  
import com.example.demo.dao.OrderDAO;  
import com.example.demo.dao.VehicleDAO;  
import com.example.demo.form.CustomerForm;  
import com.example.demo.model.CartInfo;  
import com.example.demo.model.CustomerInfo;  
import com.example.demo.model.VehicleInfo;  
import com.example.demo.pagination.PaginationResult;  
import com.example.demo.utils.Utils;  
import com.example.demo.validator.CustomerFormValidator;  
  
@Controller  
@Transactional  
public class MainController {  
   
 @Autowired  
 private OrderDAO orderDAO;  
   
 @Autowired  
 private VehicleDAO vehicleDAO;  
   
 @Autowired  
 private CustomerFormValidator customerFormValidator;  
   
 @InitBinder  
 public void myInitBinder(WebDataBinder dataBinder) {  
 Object target = dataBinder.getTarget();  
 if (target == null) {  
 return;  
 }  
 System.*out*.println("Target=" + target);  
   
 // Case update quantity in cart  
 // (@ModelAttribute("cartForm") @Validated CartInfo cartForm)  
 if (target.getClass() == CartInfo.class) {  
   
 }  
   
 // Case save customer information.  
 // (@ModelAttribute @Validated CustomerInfo customerForm)  
 else if (target.getClass() == CustomerForm.class) {  
 dataBinder.setValidator(customerFormValidator);  
 }  
   
 }  
   
 @RequestMapping("/403")  
 public String accessDenied() {  
 return "/403";  
 }  
   
 @RequestMapping("/")  
 public String home() {  
 return "index";  
 }  
   
 // Product List  
 @RequestMapping({ "/productList" })  
 public String listProductHandler(Model model, //  
 @RequestParam(value = "name", defaultValue = "") String likeName,  
 @RequestParam(value = "page", defaultValue = "1") int page) {  
 final int maxResult = 8;  
 final int maxNavigationPage = 10;  
   
 PaginationResult<VehicleInfo> result = vehicleDAO.queryProducts(page, //  
 maxResult, maxNavigationPage, likeName);  
   
 model.addAttribute("paginationProducts", result);  
 return "productList";  
 }  
   
 @RequestMapping({ "/buyProduct" })  
 public String listProductHandler(HttpServletRequest request, Model model, //  
 @RequestParam(value = "code", defaultValue = "") String code) {  
   
 Vehicle vehicle = null;  
 if (code != null && code.length() > 0) {  
 vehicle = vehicleDAO.findVehicle(code);  
 }  
 if (vehicle != null) {  
   
 //  
 CartInfo cartInfo = Utils.*getCartInSession*(request);  
   
 VehicleInfo vehicleInfo = new VehicleInfo(vehicle);  
   
 cartInfo.addProduct(vehicleInfo, 1);  
 }  
   
 return "redirect:/shoppingCart";  
 }  
   
 @RequestMapping({ "/shoppingCartRemoveProduct" })  
 public String removeProductHandler(HttpServletRequest request, Model model, //  
 @RequestParam(value = "code", defaultValue = "") String code) {  
 Vehicle vehicle = null;  
 if (code != null && code.length() > 0) {  
 vehicle = vehicleDAO.findVehicle(code);  
 }  
 if (vehicle != null) {  
   
 CartInfo cartInfo = Utils.*getCartInSession*(request);  
   
 VehicleInfo vehicleInfo = new VehicleInfo(vehicle);  
   
 cartInfo.removeProduct(vehicleInfo);  
   
 }  
   
 return "redirect:/shoppingCart";  
 }  
   
 // POST: Update quantity for product in cart  
 @RequestMapping(value = { "/shoppingCart" }, method = RequestMethod.*POST*)  
 public String shoppingCartUpdateQty(HttpServletRequest request, //  
 Model model, //  
 @ModelAttribute("cartForm") CartInfo cartForm) {  
   
 CartInfo cartInfo = Utils.*getCartInSession*(request);  
 cartInfo.updateQuantity(cartForm);  
   
 return "redirect:/shoppingCart";  
 }  
   
 // GET: Show cart.  
 @RequestMapping(value = { "/shoppingCart" }, method = RequestMethod.*GET*)  
 public String shoppingCartHandler(HttpServletRequest request, Model model) {  
 CartInfo myCart = Utils.*getCartInSession*(request);  
 CartInfo cartInfo = Utils.*getCartInSession*(request);  
   
 model.addAttribute("cartForm", myCart);  
 model.addAttribute("myCart", cartInfo);  
 return "shoppingCart";  
 }  
   
 // GET: Enter customer information.  
 @RequestMapping(value = { "/shoppingCartCustomer" }, method = RequestMethod.*GET*)  
 public String shoppingCartCustomerForm(HttpServletRequest request, Model model) {  
   
 CartInfo cartInfo = Utils.*getCartInSession*(request);  
   
 if (cartInfo.isEmpty()) {  
   
 return "redirect:/shoppingCart";  
 }  
 CustomerInfo customerInfo = cartInfo.getCustomerInfo();  
   
 CustomerForm customerForm = new CustomerForm(customerInfo);  
   
 model.addAttribute("customerForm", customerForm);  
   
 return "shoppingCartCustomer";  
 }  
   
 // POST: Save customer information.  
 @RequestMapping(value = { "/shoppingCartCustomer" }, method = RequestMethod.*POST*)  
 public String shoppingCartCustomerSave(HttpServletRequest request, //  
 Model model, //  
 @ModelAttribute("customerForm") @Validated CustomerForm customerForm, //  
 BindingResult result, //  
 final RedirectAttributes redirectAttributes) {  
   
 if (result.hasErrors()) {  
 customerForm.setValid(false);  
 // Forward to reenter customer info.  
 return "shoppingCartCustomer";  
 }  
   
 customerForm.setValid(true);  
 CartInfo cartInfo = Utils.*getCartInSession*(request);  
 CustomerInfo customerInfo = new CustomerInfo(customerForm);  
 cartInfo.setCustomerInfo(customerInfo);  
   
 return "redirect:/shoppingCartConfirmation";  
 }  
   
 // GET: Show information to confirm.  
 @RequestMapping(value = { "/shoppingCartConfirmation" }, method = RequestMethod.*GET*)  
 public String shoppingCartConfirmationReview(HttpServletRequest request, Model model) {  
 CartInfo cartInfo = Utils.*getCartInSession*(request);  
   
 if (cartInfo == null || cartInfo.isEmpty()) {  
   
 return "redirect:/shoppingCart";  
 } else if (!cartInfo.isValidCustomer()) {  
   
 return "redirect:/shoppingCartCustomer";  
 }  
 model.addAttribute("myCart", cartInfo);  
   
 return "shoppingCartConfirmation";  
 }  
   
 // POST: Submit Cart (Save)  
 @RequestMapping(value = { "/shoppingCartConfirmation" }, method = RequestMethod.*POST*)  
   
 public String shoppingCartConfirmationSave(HttpServletRequest request, Model model) {  
 CartInfo cartInfo = Utils.*getCartInSession*(request);  
   
 if (cartInfo.isEmpty()) {  
   
 return "redirect:/shoppingCart";  
 } else if (!cartInfo.isValidCustomer()) {  
   
 return "redirect:/shoppingCartCustomer";  
 }  
 try {  
 orderDAO.saveOrder(cartInfo);  
 } catch (Exception e) {  
   
 return "shoppingCartConfirmation";  
 }  
   
 // Remove Cart from Session.  
 Utils.*removeCartInSession*(request);  
   
 // Store last cart.  
 Utils.*storeLastOrderedCartInSession*(request, cartInfo);  
   
 return "redirect:/shoppingCartFinalize";  
 }  
   
 @RequestMapping(value = { "/shoppingCartFinalize" }, method = RequestMethod.*GET*)  
 public String shoppingCartFinalize(HttpServletRequest request, Model model) {  
   
 CartInfo lastOrderedCart = Utils.*getLastOrderedCartInSession*(request);  
   
 if (lastOrderedCart == null) {  
 return "redirect:/shoppingCart";  
 }  
 model.addAttribute("lastOrderedCart", lastOrderedCart);  
 return "shoppingCartFinalize";  
 }  
   
 @RequestMapping(value = { "/productImage" }, method = RequestMethod.*GET*)  
 public void productImage(HttpServletRequest request, HttpServletResponse response, Model model,  
 @RequestParam("code") String code) throws IOException {  
 Vehicle vehicle = null;  
 if (code != null) {  
 vehicle = this.vehicleDAO.findVehicle(code);  
 }  
 if (vehicle != null && vehicle.getImage() != null) {  
 response.setContentType("image/jpeg");  
 response.getOutputStream().write(vehicle.getImage());  
 }  
 response.getOutputStream().close();  
 }  
   
}

**Клас Util**

package com.example.demo.utils;  
  
import javax.servlet.http.HttpServletRequest;  
  
import com.example.demo.model.CartInfo;  
  
public class Utils {  
   
 // Products in the cart, stored in Session.  
 public static CartInfo getCartInSession(HttpServletRequest request) {  
   
 CartInfo cartInfo = (CartInfo) request.getSession().getAttribute("myCart");  
   
   
 if (cartInfo == null) {  
 cartInfo = new CartInfo();  
   
 request.getSession().setAttribute("myCart", cartInfo);  
 }  
   
 return cartInfo;  
 }  
   
 public static void removeCartInSession(HttpServletRequest request) {  
 request.getSession().removeAttribute("myCart");  
 }  
   
 public static void storeLastOrderedCartInSession(HttpServletRequest request, CartInfo cartInfo) {  
 request.getSession().setAttribute("lastOrderedCart", cartInfo);  
 }  
   
 public static CartInfo getLastOrderedCartInSession(HttpServletRequest request) {  
 return (CartInfo) request.getSession().getAttribute("lastOrderedCart");  
 }  
   
 }

**Клас PaginationResult**

package com.example.demo.pagination;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import org.hibernate.ScrollMode;  
import org.hibernate.ScrollableResults;  
import org.hibernate.query.Query;  
  
public class PaginationResult<E> {  
   
 private int totalRecords;  
 private int currentPage;  
 private List<E> list;  
 private int maxResult;  
 private int totalPages;  
   
 private int maxNavigationPage;  
   
 private List<Integer> navigationPages;  
   
 // @page: 1, 2, ..  
 public PaginationResult(Query<E> query, int page, int maxResult, int maxNavigationPage) {  
 final int pageIndex = Math.*max*(page - 1, 0);  
   
 int fromRecordIndex = pageIndex \* maxResult;  
 int maxRecordIndex = fromRecordIndex + maxResult;  
   
 ScrollableResults resultScroll = query.scroll(ScrollMode.*SCROLL\_INSENSITIVE*);  
   
 List<E> results = new ArrayList<>();  
   
 boolean hasResult = resultScroll.first();  
   
 if (hasResult) {  
 // Scroll to position:  
 hasResult = resultScroll.scroll(fromRecordIndex);  
   
 if (hasResult) {  
 do {  
 E record = (E) resultScroll.get(0);  
 results.add(record);  
 } while (resultScroll.next()//  
 && resultScroll.getRowNumber() >= fromRecordIndex  
 && resultScroll.getRowNumber() < maxRecordIndex);  
   
 }  
   
 // Go to Last record.  
 resultScroll.last();  
 }  
   
 // Total Records  
 this.totalRecords = resultScroll.getRowNumber() + 1;  
 this.currentPage = pageIndex + 1;  
 this.list = results;  
 this.maxResult = maxResult;  
   
 if (this.totalRecords % this.maxResult == 0) {  
 this.totalPages = this.totalRecords / this.maxResult;  
 } else {  
 this.totalPages = (this.totalRecords / this.maxResult) + 1;  
 }  
   
 this.maxNavigationPage = maxNavigationPage;  
   
 if (maxNavigationPage < totalPages) {  
 this.maxNavigationPage = maxNavigationPage;  
 }  
   
 this.calcNavigationPages();  
 }  
   
 private void calcNavigationPages() {  
   
 this.navigationPages = new ArrayList<Integer>();  
   
 int current = Math.*min*(this.currentPage, this.totalPages);  
   
 int begin = current - this.maxNavigationPage / 2;  
 int end = current + this.maxNavigationPage / 2;  
   
 // The first page  
 navigationPages.add(1);  
 if (begin > 2) {  
   
 // Using for '...'  
 navigationPages.add(-1);  
 }  
   
 for (int i = begin; i < end; i++) {  
 if (i > 1 && i < this.totalPages) {  
 navigationPages.add(i);  
 }  
 }  
   
 if (end < this.totalPages - 2) {  
   
 // Using for '...'  
 navigationPages.add(-1);  
 }  
 // The last page.  
 navigationPages.add(this.totalPages);  
 }  
   
 public int getTotalPages() {  
 return totalPages;  
 }  
   
 public int getTotalRecords() {  
 return totalRecords;  
 }  
   
 public int getCurrentPage() {  
 return currentPage;  
 }  
   
 public List<E> getList() {  
 return list;  
 }  
   
 public int getMaxResult() {  
 return maxResult;  
 }  
   
 public List<Integer> getNavigationPages() {  
 return navigationPages;  
 }  
   
 }