МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра економічної кібернетики та інформатики

МІЖДИСЦИПЛІНАРНА КУРСОВА РОБОТА

на тему:
Розробка Web базовної інформаційної системи електронної комерції

Студента 4 курсу групи СА-41
спеціальності 124 «Системний аналіз»

Яскілка О.В.
(прізвище та ініціали)

Керівник
_____

_____
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала _____

Кількість балів: _____Оцінка: ECTS _____

Члени комісії: _____(_____)
прізвище та ініціали підпис
_____(_____)
прізвище та ініціали підпис
_____(_____)
прізвище та ініціали підпис

м. Тернопіль – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра економічної кібернетики та інформатики

ЗАВДАННЯ НА МІЖДИСЦИПЛІНАРНУ КУРСОВУ РОБОТУ

група СА -41 курс 4

Студента Яскілки О.В.

Тема міждисциплінароної курсової роботи:
Розробка Web базовної інформаційної системи електронної комерції

Основні розділи міждисциплінарної курсової роботи:

Вступ
1. Аналіз технологій та платформ електроної комерції
2. Інструментальні засоби розробки інформаційної системи
3. Проектування та тестування інформаційної системи електронної комерції

Висновки
Рекомендована література:
1. Что такое веб-приложения и динамические веб-страницы
2. Spring Framework
3. Як працює оцінювання задач у Kanban на практиці
Додатки

Дата видачі завдання "____"_____ 2024р.

Термін представлення роботи на кафедру "____"_____ 2024р.

Керівник міжисциплінарної курсової роботи _____

ЗМІСТ

ВСТУП

Проблема автоматизації виробничих процесів і процесів управління як засобу підвищення продуктивності праці завжди була і залишається актуальною.

Автоматизація – один з напрямів науково-технічного прогресу, спрямований на застосування саморегульованих технічних засобів, економіко-математичних методів і систем керування, що звільняють людину від участі в процесах отримання, перетворення, передачі і використання енергії, матеріалів чи інформації, істотно зменшують міру цієї участі чи трудомісткість виконуваних операцій. Разом з терміном автоматичний, використовується поняття автоматизований, що підкреслює відносно великий ступінь участі людини в процесі.

Сьогодні продуктивність є головним фактором, який визначає ефективність системи. Правильне проектне рішення служить основою високопродуктивної системи.

В реальних умовах проектування – це пошук способу забезпечити вимоги функціональності системи засобами наявних технологій із врахуванням заданих обмежень.

Метою даної курсової роботи є розробка інформаційної системи електронної комерції.

Розділ 1
Аналіз технологій та платформ електронної комерції

1.1 .Web-базовані платформи електронної комерції

Веб-додаток — розподілений додаток, в якому клієнтом виступає браузер, а сервером — веб-сервер. Браузер може бути реалізацією так званих тонких клієнтів — логіка додатку зосереджується на сервері, а функція браузера полягає переважно у відображенні інформації, завантаженої мережею з сервера, і передачі назад даних користувача. Однією з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-додатки є міжплатформовими сервісами. Унаслідок цієї універсальності та відносної простоти розробки веб-застосунки стали широко популярними в кінці 1990-х — початку 2000-х років. [1]

Сьогодні веб-додатки набирають популярність. найбільш відвідуваними сайтами стають не чисто інформаційні, гіпертекстові сайти, а ті, які надають будь-які сервіси, будь-яким чином взаємодіють з користувачем. Але навіть і звичайні інформаційні сайти часто використовують системи управління контентом для зручності управління інформацією, так що і їх теж можна зарахувати до веб-додатків. Веб-додаток являє собою веб-сайт, на якому розміщені сторінки з частково або повністю несформованим вмістом. Остаточний вміст формується тільки після того, як відвідувач сайту запросить сторінку з веб-сервера. У зв'язку з тим що остаточний вміст сторінки залежить від запиту, створеного на основі дій користувача, така сторінка називається динамічної.

Будь-який веб-додаток являє собою набір статичних і динамічних вебсторінок. Статична веб-сторінка - це сторінка, яка завжди відображається перед користувачем в незмінному вигляді. Веб-сервер відправляє сторінку за

запитом веб-браузера без будь-яких змін. На противагу цьому, сервер вносить зміни в динамічну веб-сторінку перед відправкою її браузеру. У зв'язку з тим що сторінка змінюється, вона називається динамічною. Коли веб-сервер отримує запит на видачу статичної веб-сторінки, він відправляє сторінку безпосередньо браузеру. Однак, коли запитується динамічна сторінка, дії веб-сервера не настільки однозначні. Сервер передає сторінку спеціальною програмою, яка і формує остаточну сторінку. Така програма називається сервером додатків.

Сервер додатків виконує читання коду, що знаходиться на сторінці, формує остаточну сторінку відповідно до прочитаним кодом, а потім видаляє його з сторінки. В результаті всіх цих операцій виходить статична сторінка, яка передається веб-сервера, який в свою чергу відправляє її клієнтському браузеру.

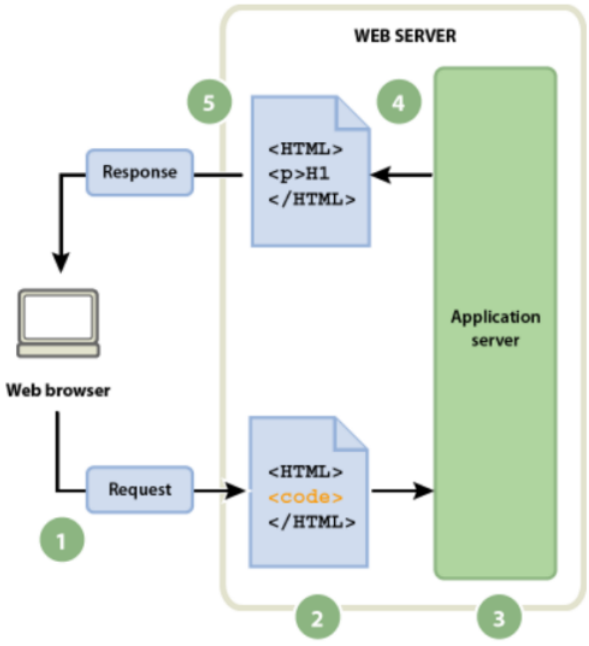

Рис.1.1 Схематичне зображення процесу

Процес включає наступні етапи:

1. Веб-браузер запитує динамічну сторінку.

2. Веб-сервер знаходить сторінку і передає її сервера додатків.

3. Сервер додатків переглядає сторінку на наявність інструкцій і виконує її створення.

4. Сервер додатків повертає підготовлену сторінку на веб-сервер.

5. Веб-сервер відправляє підготовлену сторінку запити її браузеру.

Всі сторінки, які отримує браузер, містять тільки HTML-код. Схематичне зображення процесу на рис.1.1.

Сервер додатків надає можливість використовувати такі ресурси сервера, як бази даних. Наприклад, динамічна сторінка може містити програмні інструкції для сервера додатків, слідуючи яким сервера необхідно отримати певні дані з бази даних і помістити їх в HTML-код сторінки. Зберігання вмісту в базі даних дозволяє відокремити оформлення вебсайту від вмісту, яке будуть бачити користувачі. Замість того щоб створювати всі сторінки у вигляді окремих HTML-файлів, пишуться тільки шаблони сторінок для кожного виду інформації, що представляється. Потім вміст завантажується в базу даних, після чого веб-сайт буде витягувати його при запитах користувачів. Крім того, можна оновити інформацію в одному джерелі і продублювати це зміна на всіх веб-сайті без редагування кожної сторінки вручну.

Програмна інструкція, призначена для отримання даних з бази даних, називається запитом до бази даних. Запит складається з критеріїв пошуку, виражених за допомогою мови баз даних, званого SQL (мова структурованих 11 запитів). Текст SQL-запиту розташовується в сценаріях сторінок на стороні сервера або в тегах. Сервер додатків не може безпосередньо отримати дані з бази, оскільки бази даних використовують специфічні формати зберігання даних. Тому для підключення до бази даних сервер додатків використовує посередника - драйвер бази даних.

Драйвер бази даних являє собою програмний модуль, за допомогою якого встановлюється взаємодія між сервером додатків і базою даних. Після того як драйвер встановить з'єднання, виконується запит до бази, в результаті чого

формується набір записів. Набір записів є безліч даних, отриманих з однієї або декількох таблиць бази даних. Набір записів повертається сервера додатків, який використовує отримані дані для формування сторінки. [2]

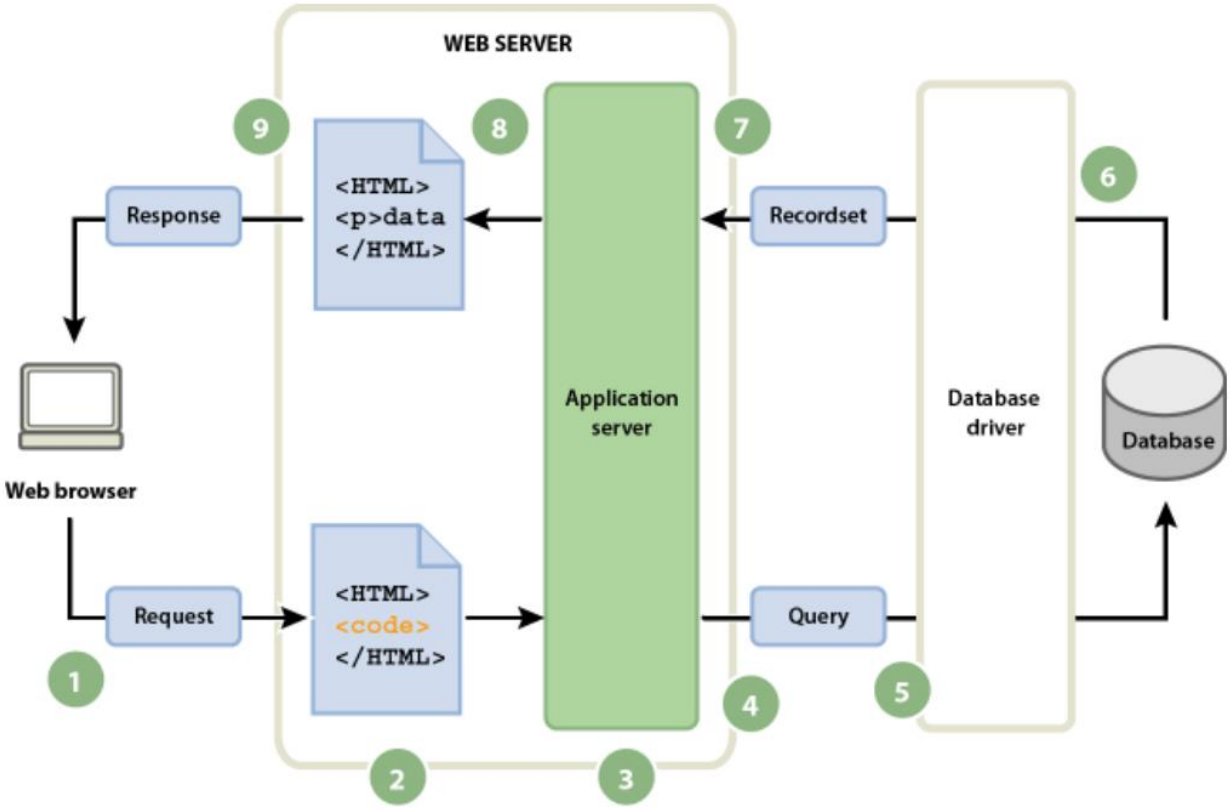

Рис.1.2 Процес формування сторінки

1. Веб-браузер запитує динамічну сторінку.

2. Веб-сервер знаходить сторінку і передає її сервера додатків.

3. Сервер додатків переглядає сторінку на наявність інструкцій і виконує її підготовку.

4. Сервер додатків відправляє запит драйверу бази даних.

5. Драйвер виконує запит в базі даних.

6. Драйвера повертається набір записів.

7. Драйвер передає набір записів сервера додатків.

8. Сервер додатків вставляє дані в сторінку і передає сторінку вебсервера. 9. Веб-сервер відправляє підготовлену сторінку запити її браузеру.

Для використання в веб-додатку придатна будь-яка база даних за умови, що на сервері встановлений відповідний драйвер бази даних. Для створення нашого додатку було використанно PostgressSql.

1.2 Архітектурний шаблон Model-view-Controller

Шаблон проектування MVC передбачає поділ даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: Модель, Представлення і Контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно (рис.1.3).

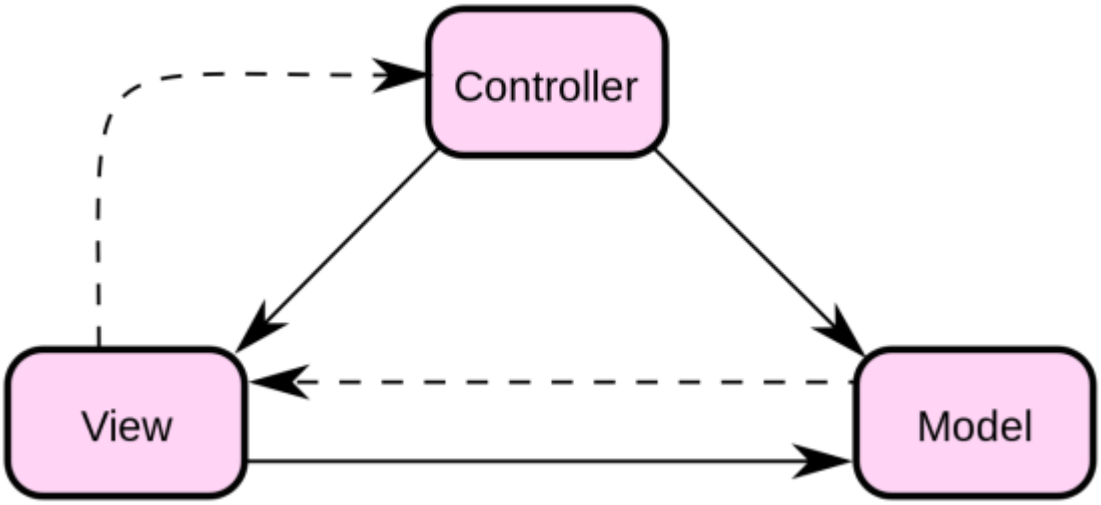

Рис.1.3 Шаблон проектування MVC

В даному шаблоні модель не залежить від представлення або керуючої логіки, що робить можливим проектування моделі як незалежного компонента і, наприклад, створювати кілька представленнь для однієї моделі.

Вперше цей шаблон був застосований в фреймворку, що розробляється для мови Smalltalk в кінці 1970-х років. З цього моменту він відіграє основну роль в більшості фреймфорков з призначеним для користувача інтерфейсом. Він докорінно змінив погляд на проектування додатків.

Модель інкапсулює ядро даних і основний функціонал їхньої обробки і не залежить від процесу вводу чи виводу даних.

Вигляд може мати декілька взаємопов'язаних областей, наприклад різні таблиці і поля форм, в яких відображаються дані.

У функції контролера входить відстеження визначених подій, що виникають в результаті дій користувача. Контролер дозволяє структурувати код шляхом групування пов'язаних дій в окремий клас. Наприклад у типовому MVC-проекті може бути користувацький контролер, що містить групу методів, пов'язаних з управлінням обліковим записом користувача, таких як реєстрація, авторизація, редагування профілю та зміна пароля. [3]

Зареєстровані події транслюються в різні запити, що спрямовуються компонентам моделі або об'єктам, відповідальним за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Таким чином, якщо користувач через контролер внесе зміни до моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися.

Для реалізації шаблону було використано фреймворк Spring MVC. Він забезпечує роботу цього паттерну. Вся логіка роботи Spring MVC побудована навколо DispatcherServlet, який приймає і обробляє всі HTTP-запити (з UI) і відповіді на них.

Робочий процес обробки запиту DispatcherServlet'ом проілюстрований на наступній діаграмі:

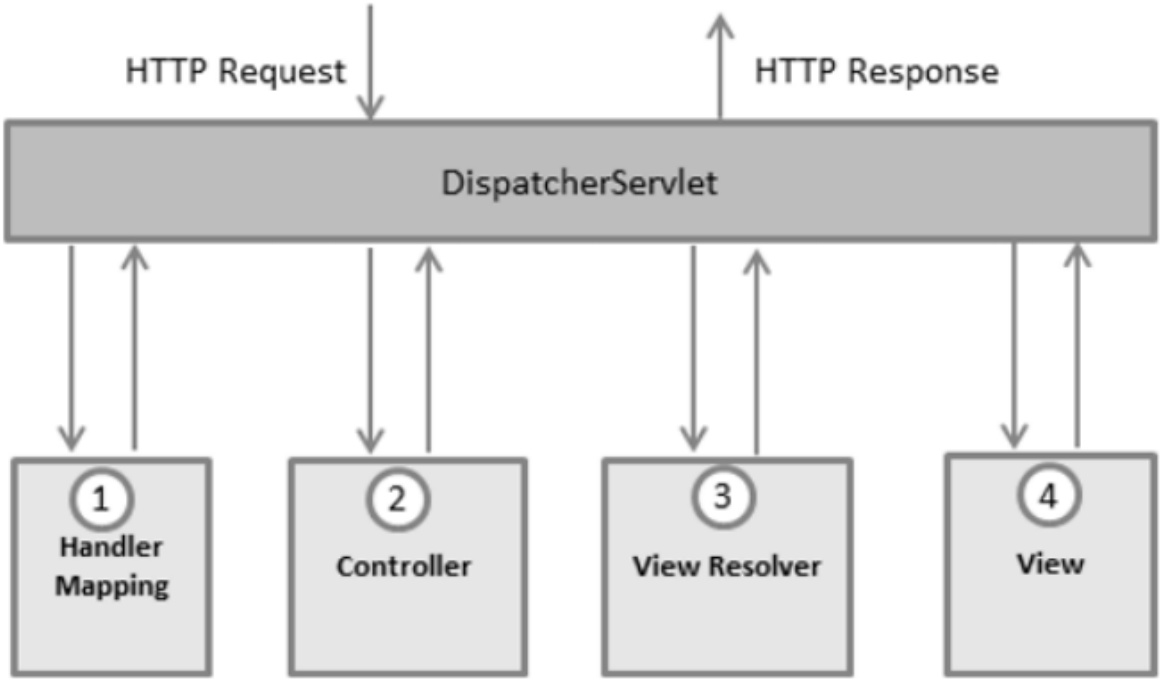

Рис.1.4 Робочий процес обробки запиту DispatcherServlet'ом

Нижче наведена послідовність подій, відповідна входить HTTP-запиту:

- Після отримання HTTP-запиту DispatcherServlet звертається до інтерфейсу HandlerMapping, який визначає, який Контролер повинен бути викликаний, після чого, відправляє запит в потрібний Контролер.
- Контролер приймає запит і викликає відповідний службовий метод, заснований на GET або POST методах. Викликаний метод визначає дані Моделі, засновані на певній бізнес-логіці і повертає в DispatcherServlet ім'я Виду (View).
- За допомогою інтерфейсу ViewResolver DispatcherServlet визначає, який Вид потрібно використовувати на підставі отриманого імені.
- Після того, як Вид (View) створений, DispatcherServlet відправляє дані Моделі у вигляді атрибутів в Вид, який в кінцевому підсумку відображається в браузері.

Всі вище згадані компоненти, а саме, HandlerMapping, Controller і ViewResolver, є частинами інтерфейсу WebApplicationContext extends

ApplicationContext, з деякими додатковими особливостями, необхідними для створення web-додатків.

1.3 Мікросервісна архітектура для Web-базованої системи

Мікросервіси — архітектурний стиль за яким єдиний застосунок будується як сукупність невеличких сервісів кожен з яких працює у своєму власному процесі і комунікує з рештою використовуючи легковагові механізми, зазвичай HTTP. Ці сервіси будуються навколо бізнес-потреб і розгортаються незалежно з використанням зазвичай повністю автоматизованого середовища. Існує абсолютний мінімум централізованого керування цими сервісами. Самі по собі вони можуть бути написані з використанням різних мов і технологій зберігання даних.

Мікросервісна архітектура добре підходить для процесу безперервної поставки, на відміну від сервіс-орієнтовної архітектури мікросервісна спрямована на створення одного застосунка в той час як сервісно орієнтована система — являє собою множину застосунків які взаємодіють між собою.

Основні властивості:

- Високий рівень незалежності: незалежна розробка, незалежне розгортання
- Незалежне масштабування
- Невелика кодова база зменшує кількість конфліктів та дозволяє швидко залучати нових розробників
- Простота заміни однієї реалізації сервісу іншою
- Простота додавання нового функціоналу в систему
- Ефективне використання ресурсів
- Еластичність: вихід з ладу одного сервісу зазвичай не призводить до виходу з ладу всієї системи
- Сервіси організовані відносно бізнес логіки яку вони виконують

- Кожен сервіс незалежно від інших може бути реалізований за допомогою будь-якої мови програмування, СУБД, та ін.
- Архітектурно побудовані за симетричним принципом (виробникспоживач)

1.4 Предметна область Web- додатку

Веб додаток буде функціонувати як віртуальний торговельна площадка де у відкритому доступі розміщуються оголошення про продаж і характеристики автомобілів.

Стурктура веб-додатку:

- Повнотекстовий пошук автомобілей;
- Детальний пошук, за типом машини, рік випуску, ціна і тд.
- Реєстрація нових користувачів;
- Авторизація користувачів;

У веб-додатків є чимало переваг у порівнянні зі звичайними десктопзастосуноками .

Наприклад, розробка веб-додатків забезпечує повну сумісність при роботі на різних платформах. Якщо звичайне оффлайн-додаток доводиться «заточувати» під певні системні вимоги конкретної платформи, то веб-додаток завжди і всюди може бути доступним для будь-якої локальної машини. Все що потрібно - це браузер і включений доступ до інтернету.

Всі сучасні веб-додатки мають важливі властивості, серед яких особливо велике значення мають:

- масштабованість веб-системи;
- інтеграція з іншими системами;
- розмежування прав доступу до різного функціоналу;
- зручне розгортання і обслуговування системи.

Наявність подібних властивостей дозволяє використовувати веб-додатки і веб-системи максимально ефективно і зручно. Наприклад, завдяки властивості масштабованості можна без внесення кардинальних змін розширювати вебсистему для роботи з постійно зростаючим числом користувачів, додавати в неї нові функції.

Розробка веб-додатків надає широкі можливості по створенню для компаній багатофункціональних онлайн-інструментів для оптимізації або вирішення різних бізнес-задач.

Веб-додаток, буде заснований на сучасних базах даних, з програмно-технічним комплексом, і буде призначений для зручного пошуку та продажу нових та вживаних автомобілей.

Завданням системи є:

1) Забезпечети безпеку обробки та зберігання персональних данних користувача;

2) Облік користувачів;

3) Облік колонок для карток. Колонки повинні забепечити сортування карток за пріорітетом;

4) Облік карток. Користувач має можливість додавати до картки: опис завдання, прикріплення у вигдляді посилання, пріорітет виконання цієї картки;

Проаналізувавши предметну область, можна сказати, що розробка даного веб-додатку є вельми актуальною і дозволить користувачам зручно керувати своїми оголошеннями та зв’язуватись з іншими коричстувачами.

Розділ 2
Інструментальні засоби розробки інформаційної системи

2.1 Аналіз Java орієнтовані серверні застосування.

Java - це потужна та популярна мова програмування, яка була розроблена компанією Sun Microsystems (зараз належить Oracle Corporation) у 1995 році. Відома своєю надійністю, портативністю та великою спільнотою розробників. Вона має свої переваги та недоліки при створенні веб-застосунків, включаючи блоги.

Огляд основних переваг та недоліків Java:

переваги Java:

1) Платформо незалежність: Однією з головних переваг Java є його платформо незалежність. Програми, написані на Java, можуть працювати на різних операційних системах без потреби перекомпіляції або змін вихідного коду. Це означає, що ваш блог може бути запущений на будь- 40 якій платформі, що підтримує Java, зменшуючи залежність від конкретної ОС.

2) Велика спільнота та підтримка: Java має велику спільноту розробників та багато ресурсів, таких як документація, форуми та бібліотеки, що допомагають у розробці веб-застосунків. Це означає, що ви можете швидко знайти допомогу, розв'язати проблеми та скористатися готовими рішеннями для свого блогу.

3) Великий вибір фреймворків та бібліотек: Java має широкий вибір різноманітних фреймворків та бібліотек, які полегшують розробку вебзастосунків. Наприклад, фреймворк Spring надає потужні інструменти для розробки веб-застосунків, включаючи підтримку інверсії керування та внедрення залежностей. Це дозволяє прискорити розробку блогу та забезпечити його ефективність.

недоліки Java:

1) Складність: Java може бути дещо складним для вивчення та розуміння, особливо для початківців. Вона вимагає знання об'єктноорієнтованого програмування та інших концепцій, що можуть затримувати розробку для деяких людей.

2) Громіздкість: Java, порівняно з іншими мовами програмування, може вимагати більшого обсягу коду для досягнення тих самих функціональних можливостей. Це може призвести до більшої тривалості розробки та обслуговування коду.

3) Важкість в розумінні: Для деяких розробників Java може викликати відчуття "важкості", оскільки вона має багато надлишкових конструкцій та строгих правил, які потрібно дотримуватися. Це може призвести до більшої складності у вирішенні простих задач або прискоренні розробки. 41 Враховуючи переваги та недоліки Java, важливо звернути увагу на особливості вашого проекту та ваш рівень володіння мовою програмування, перш ніж обрати Java для створення блогу.

Рис. 2.1. Логотип Java

Spring Framework забезпечує вирішення багатьох завдань, з якими стикаються Java-розробники та організації, які хочуть створити інформаційну систему, засновану на платформі Java. Через широку функціональність важко визначити найважливіші структурні елементи, з яких він складається. Spring

Framework не повністю пов'язана з платформою Java Enterprise, незважаючи на його масштабну інтеграцію з нею, що є важливою причиною його популярності.

2.2 Використання SpringBoot для проектування структури системи

Spring Framework - вірогідно, найбільш відомий як джерело розширень (особливостей), необхідних для ефективної розробки складних бізнес-додатків, що не мають важких програмних моделей, які історично були домінуючими в галузі.

Рис. 2.2. Логотип Spring Boot

Ще одна його достоїнство в тому, що він вніс раніше невикористані функціональні можливості в сьогоднішні основні методи розробки, навіть поза платформою Java.

Цей фреймворк пропонує послідовну модель і робить її прикладною для більшості типів додатків, які вже створені на базі платформи Java.

Вважаэться, що Spring Framework реалізує модель розробки, засновану на кращих стандартах промисловості, і робить її доступною у багатьох сферах Java.

Spring Framework може бути розглянутий як колекція менших фреймворків або фреймворків у фреймворку. Більшість цих фреймворків може

працювати незалежно один від одного, проте, вони забезпечують більшу функціональність при спільному їх використанні.

Ці фреймворки діляться на структурні елементи типових комплексних програм. Схематичне зображення модулей фреймворку на рис. 2.3

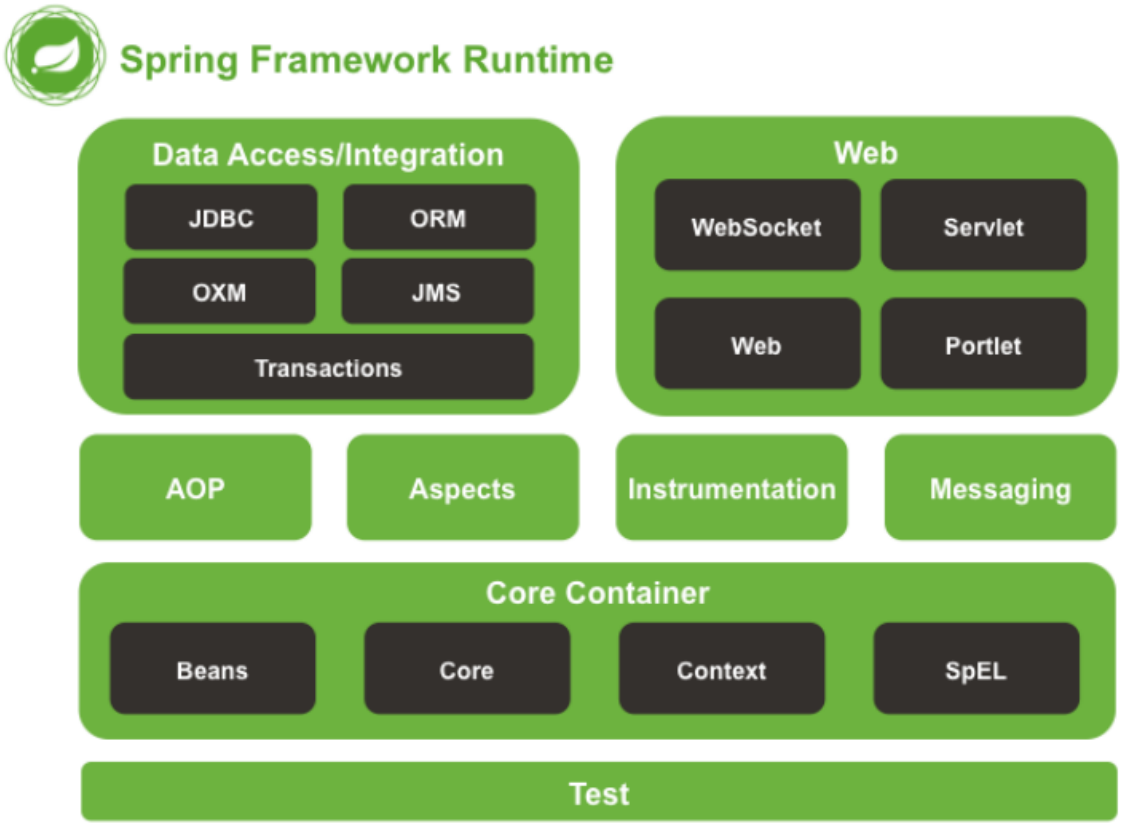

Рис. 2.3 Модулі Spring Fremework

- Inversion of Control контейнер: конфігурація компоненту додатків і управління життєвим циклом Java об'єктів.
- Фреймворк аспектно-орієнтованого програмування: працює з функціональністю, яка не може бути реалізована можливостями об'єктноорієнтованого програмування на Java без втрат.
- Фреймворк доступу до даних: працює з системами керування базами даних на Java платформі використовуючи JDBC і Object-relational mapping кошти забезпечуючи вирішення завдань, які повторюються в великому числі Java-based environments.

• Фреймворк управління транзакціями: координація різних API керування транзакціями і інструментарій настроюваного управління транзакціями для об'єктів Java.

• Фреймворк Model-view-controller: каркас, заснований на HTTP і сервлетах надає безліч можливостей для розширення та налаштування (customization).

• Фреймворк віддаленого доступу: конфігурується передача Javaоб'єктів через мережу в стилі RPC, підтримуюча RMI, CORBA, HTTP-based протоколи, включаючи web-сервіси (SOAP).

• Фреймворк аутентифікації і авторизації: конфігурується інструментарій процесів аутентифікації і авторизації, що підтримує багато популярних і стали індустріальними стандартами протоколів, інструментів, практик через дочірній проект Spring Security (раніше відомий як Acegi).

• Фреймворк віддаленого управління: конфігурується уявлення і управління Java об'єктами для локальної або віддаленої конфігурації за допомогою JMX.

• Фреймворк роботи з повідомленнями: конфігурується реєстрація об'єктів-слухачів повідомлень для прозорої обробки повідомлень з черги повідомлень за допомогою JMS, поліпшена відправлення повідомлень за стандартом JMS API.

• Тестування: каркас, що підтримує класи для написання модульних і інтеграційних тестів.

Центральною частиною Spring Framework є Inversion of Control контейнер, який надає кошти конфігурації і управління об'єктами Java за допомогою зворотних викликів. Контейнер відповідає за управління життєвим циклом об'єкта: створення об'єктів, виклик методів ініціалізації і конфігурація об'єктів шляхом зв'язування їх між собою[4].

Об'єкти створюються контейнером також називаються Керовані об'єкти або Beans. Зазвичай конфігурація контейнера здійснюється шляхом

завантаження XML файлів, що містять Визначення Bean'ов і надають інформацію необхідну для створення bean'ов

Об'єкти можуть бути отримані або за допомогою Пошуку залежності, або Впровадження залежності. Пошук залежності - шаблон проектування, коли викликає об'єкт запитує у об'єкта-контейнера екземпляр об'єкта з певним ім'ям або певного типу. Впровадження залежності - шаблон проектування, коли контейнер передає екземпляри об'єктів по їх імені іншим об'єктам або за допомогою конструктора, або властивості, або фабричного методу

2.3 Використання системи керування базою данних PostgreeSQL

PostgreSQL — об'єктно-реляційна система керування базами даних (СКБД). Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite).

Рис. 2.4 Логотип PostgreSQL

Огляд її основних переваг та недоліків у контексті роботи з вебзастосунком типу блог:

переваги PostgreSQL:

1)Розширена функціональність: PostgreSQL підтримує розширення SQL, як-от CTE (Common Table Expressions), JSON, JSONB, Window Functions, а також підтримує складніші типи даних.

2)Надійність та стійкість даних: PostgreSQL забезпечує повну відповідність ACID, що робить його відмінним вибором для проектів, де дані мають залишатися незмінними та цілісними.

3)Масштабованість та потужність: Підтримує вертикальне та горизонтальне масштабування, що робить його оптимальним вибором для великих проектів.

3)Гнучкість у налаштуваннях: PostgreSQL пропонує можливість налаштування під специфічні завдання.

недоліки MySQL:

1)Продуктивність для простих задач: PostgreSQL може бути повільнішим, ніж MySQL, для простих операцій читання/запису в базах з невеликою кількістю користувачів.

3)Вимоги до ресурсів: PostgreSQL потребує більше оперативної пам’яті та обчислювальних ресурсів для роботи в порівнянні з MySQL.

4)Складність налаштування: Початкове налаштування PostgreSQL може бути складнішим для новачків у порівнянні з MySQL.

2.4 Інтегроване середовище розробки Intellij IDEA

IntelliJ IDEA представляє собою один з найпопулярніших інтегрованих середовищ розробки (IDE) для Java програмування. Розроблений компанією JetBrains, IntelliJ IDEA надає розширений набір інструментів та функцій, що полегшують процес розробки програмного забезпечення.

Рис. 2.5 Логотип PostgreSQL

Огляд основних переваг та недоліків IntelliJ IDEA:

переваги IntelliJ IDEA:

1) Великий функціонал: IntelliJ IDEA надає багатий набір функцій, що полегшують розробку Java-програм. Вона підтримує автодоповнення коду, рефакторинг, аналіз коду, налагодження, керування залежностями та багато іншого. Це дозволяє розробникам працювати швидше та ефективніше.

2) Підтримка різних мов програмування: IntelliJ IDEA не обмежується лише Java і має вбудовану підтримку інших мов програмування, таких як Kotlin, Groovy, Scala, JavaScript, HTML, CSS та інші. Це дозволяє розробникам працювати з різними технологіями та стеками технологій в одному IDE.

3) Інтеграція з інструментами розробки: IntelliJ IDEA легко інтегрується з іншими інструментами розробки, такими як системи контролю версій (наприклад, Git), засоби автоматичної збирання (наприклад, Maven або Gradle), сервери додатків та багато іншого. Це спрощує роботу з іншими інструментами розробки в рамках одного інтерфейсу.

4) Підтримка плагінів: IntelliJ IDEA підтримує велику кількість сторонніх плагінів, що дозволяють розширити його функціонал і адаптувати до потреб конкретного проекту або технології.

недоліки IntelliJ IDEA:

1) Ресурсоємність: IntelliJ IDEA вимагає певних обчислювальних ресурсів, таких як пам'ять та процесор. На менш потужних комп'ютерах вона може працювати повільно або вимагати додаткових настроювань.

2) Обмежений функціонал безкоштовної версії: IntelliJ IDEA має платну версію, яка надає додаткові функції та інструменти. Безкоштовна версія, IntelliJ IDEA Community Edition, має обмежений функціонал.

3) Час на освоєння: Встановлення та налаштування IntelliJ IDEA може бути складним для новачків, особливо якщо вони не знайомі з певними конфігураційними параметрами або інструментами розробки. Враховуючи перечисленні вище недоліки, IntelliJ IDEA є потужним інструментом для розробки веб-застосунків типу блогу та надає багато переваг, що полегшують процес розробки та підвищують продуктивність розробника.

Розділ 3
Проектування та тестування інформаційної системи електронної комерції

3.1 Проектування та створення бази даних

Першим етапом розробки бази даних є концептуальне (інфологічне) проектування бази даних. Це побудова семантичної (формалізованої) моделі предметної області, тобто інформаційної моделі найбільш високого рівня абстракції. Така модель створюється без орієнтації на якусь конкретну СКБД і модель даних.

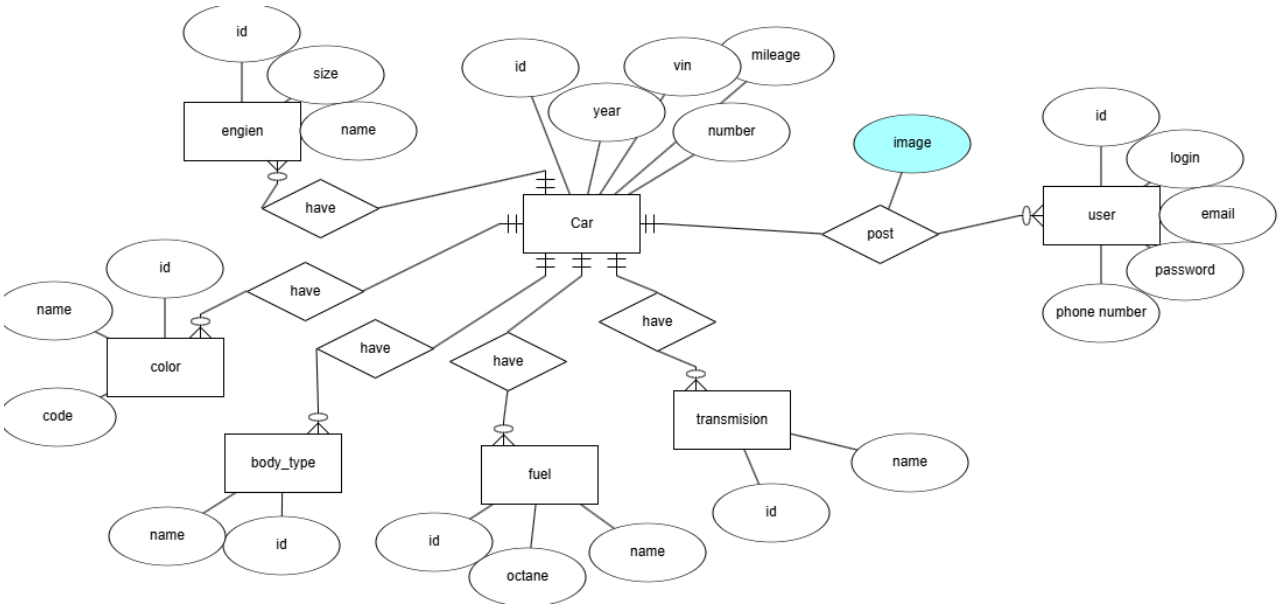

Рис. 3.1 Концептуальна модель БД

Другим етапом є створення ER діграми класів.

Схема реляційних відношень називається коректною, якщо в ній відсутні небажані функціональні залежності. Виправити некоректні схеми можна за допомоги декомпозиції (розкладання) реляційних відношень на множину

реляційних відношень, які не містять не коректностей. Це і є суттю процесу нормалізації. Код визначив 3 рівні нормалізації: 1НФ, 2НФ, 3НФ.

Реляційне відношення перебуває в 1НФ, якщо всі його атрибути – прості, тобто кожен атрибут повинен містити лише одне значення.

Реляційне відношення перебуває у 2НФ, якщо воно перебуває у 1НФ і всі його не ключові атрибути функціонально повно залежать від ключа.

Реляційне відношення перебуває у 3НФ, якщо воно перебуває в 2НФ і не містить транзитивних залежностей неперервних атрибутів від можливих ключів.[6]

Після проведення нормалізації реляційних залежностей виділено сім основних сутностей з атрибутами:

1) Користувач (код користувача, логін, пароль, ім’я, прізвище, контакти користувача);

2) Оголошення (код оголошення, код користувача, код машини, код головної фотокартки, ціна, опис, статус,);

3) Машина (код машини, код моделі, ціна, код валюти, рік випуску, опис);

4) Фотокартка (код фотокартки, шлях фотокартки, назва фото);

5) Модель (код моделі, код марки, назва моделі);

6) Тип кузова (код кузова, назва кузова);

7) Тип пального (код пального, навза пального);

8) Обєм двигуна (код, розмір двишуна);

9) Трансмісія (код трансмісії, рназва трансмісії);

10) Колір (код кольору, назва кольору);

Схема структури бази даних набуває остаточного вигляду(Рис. 3.2).

Далі розроблену схему необхідно представити у вигляді SQL-сценарію. Для цього використовується Export → SQL в налаштуваннях якого вказана цільова база даних MySQL. Після натискання кнопки Generate SQL необхідно завантажити файл SQL-сценарію в якому є всі необхідні SQL-запити для створення БД(текст данного файлу наведено в Додатку А) Для створення БД

використовуючи середовище розробики Intellij IDEA необхідно запустити на виконання збережений до цього файл SQL-сценарію

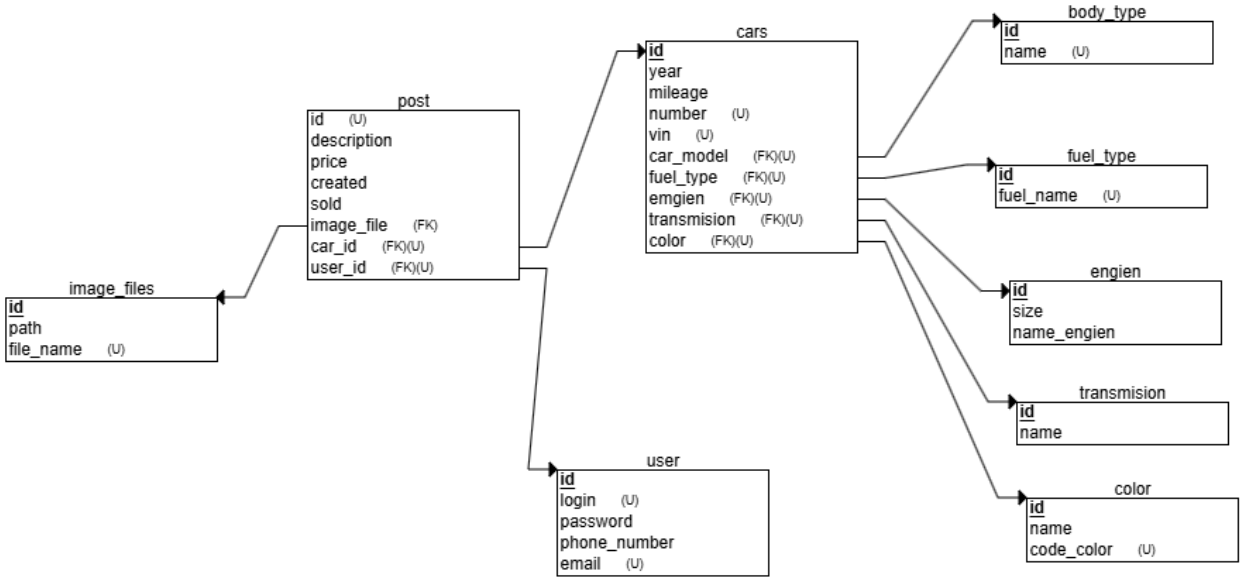

Рис. 3.2 ER діграма класів

3.2 Реалізація інформаційної системи

Серверне налаштування web-додатка

Для реалізації проекту були обрані такі технології:

- Spring Boot. Використовується для запуску серверу додатків;
- Postman Connector – забезпечує зв’язок бекенду з БД.
- Spring Data – ORM фреймворк, який зв’язує таблиці БД з сутностями
- Spring Tomcat – контейнер сервлетів, забезпечує компіляцію представлення для фронтенду;
- Java Servlet – стандартизований API для створення динамічного контенту до веб-сервера;
- Junit – бібліотека для тестування програмного забезпечення для мови Java;

Для побудови та підключення всіх залежностей які прераховані вище, було обрано технологію «Maven». Це засіб автоматизації роботи з програмними проектами. Використовується для управління (management) та складання (build) програм. Maven конфігурує проекти за допомогою конструкції ProjectObjectModel, що зберігається в файлі POM.xml.

Для підключення потрібних елементів з центрального репозиторія https://search.maven.org/ потрібно знайти необхідну технологію та скопіювати залежність. Пошук технології забражено на рис.3.3

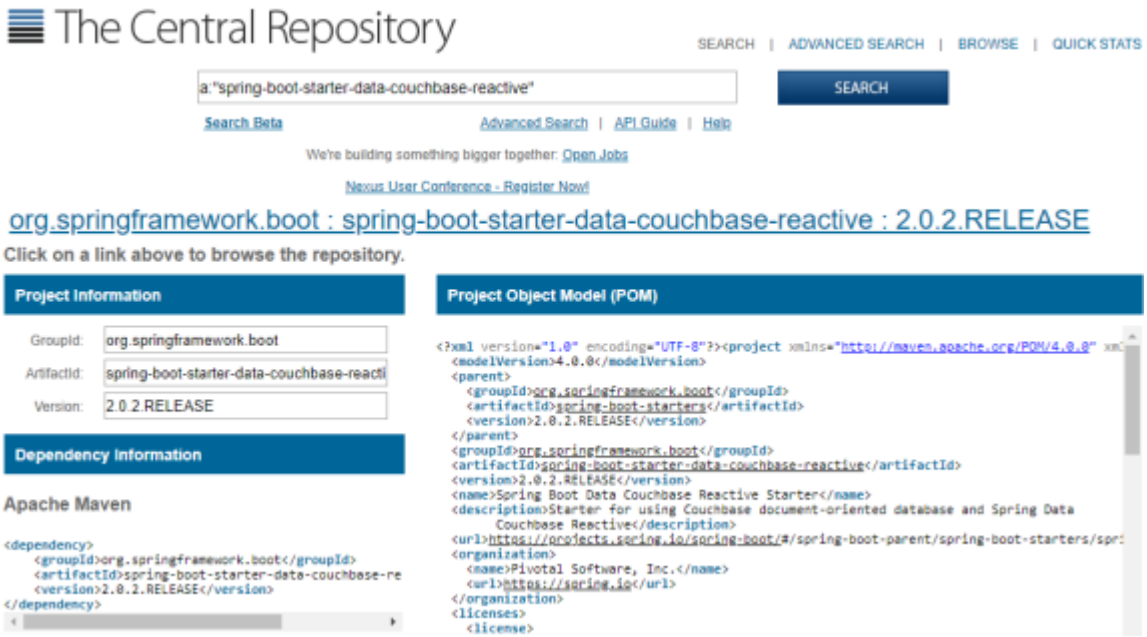

Рис. 3.3 Пошук технології

Для компіляції проекту в терміналі треба виконати команду mvn clean, під час виконання maven завантажує всі залежності до локального репозиторію. Після успішного виконання команди необхідно сконфігурувати файл налаштування application.properies (Код представленно в додатку В). В данному файлі вказується назва протоколу з’єднання(jdbc:mysql), хост і порт підключення(localhost:3306), ім’я бази даних(task-list), ім’я користувача та пароль(root, mysql).

Для того щоб зупустути сервер додатків було створенно головний клас SpringBootApp.java. Аннотація @SpringBootApplication допомагає Springконтейнеру розпізнати цей клас. В класі об`явленний метод main(), котрий створює окремий потік JVM. Він і є головною точкою запуску web-додатку.

Щоб запустити web-додаток треба виконати команду mvn spring-boot:run. Web-додаток запуститься на локальному сервері комп`ютера і буде займати порт № 8080. Строка Tomcat started on port(s): 8080 (http): Started SpringBootApp in 20.948 seconds (JVM running for 61.987) в терміналі повідомляє про те що додаток успішно запустився.

Реалізація моделей:

Після створення БД всі сутності потрібно відобразити у вигляді java класів. Було виділено такі об`єкти як : користувач, роль користувача, дошка, список задач, завдання, прикріплення.

Java Persistence API (JPA) - специфікація API Java EE, надає можливість зберігати в зручному вигляді Java-об'єкти в базі даних. Існує кілька реалізацій цього інтерфейсу, одна з найпопулярніших використовує для цього Hibernate, Eclipse Link, Spring Data. JPA реалізує концепцію ORM.

Entity (сутність/утворення) — об'єкт для якого забезпечується ORM. Класи Entity задаються аннотацією @Entity або перелічуються у XML дескрипторі. Клас Entity повинен мати конструктор без аргументів, з рівнем доступу — public або protected. Якщо сутність передається як віддалений об'єкт 55 (remote object), вона має реалізувати інтерфейс Serializable. Клас Entity не може бути завершеним (final) або мати завершені методи.

Поля класу повинні відповідати колонкам сутності з БД. Аннотації @Entity та @Table(name = “user”) вказують на те що данний клас є відображенням таблиці в БД. В аннотацію @Table передається атрибут name значення якого повинно відповідати назві таблиці користувача. Для поля з унікальним індитифікатором потрібно присвоїти аннотації @Id та

@GeneretedValue. В аннотацію @GeneretedValue передається атрибут strategy , існує декілька констант для атрибуту, а саме AUTO, INDENTITY, SEQUENCE.

Сутності в БД мають зв`язки для забезпечення зв`язків на стороні java в специфікації JPA виділенно три види аннотацій:

- @OneToOne – Визначає однозначну асоціацію іншому об'єкту, який має кратність "один до одного".
- @OneToMany Визначає багатозначну асоціацію з одного до багатьох об`єктів.
- @ManyToMany Визначає багатозначну асоціацію з багатьох до багатьох об`єктів.

Приклад реалізації класу Car.java наведенно на рис 3.4. Поля id, year, класу відповідають сутності Car. Так як car_model зіншої таблиці, було додано поле car_model з аннотаціями @ManyToOne та @JoinColumn з атрибутом name який повинен відповідати зовнішньому ключу в БД.

```java
import ...

@Entity
@Table(name = "cars")
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
public class Car {

    @Id
    @EqualsAndHashCode.Include
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne
    @JoinColumn(name = "car_model_id")
    private CarModel carModel;

    @ManyToOne
    @JoinColumn(name = "body_type_id")
    private BodyType bodyType;

    @Column(name = "car_year")
    private int carYear;

    @ManyToOne
    @JoinColumn(name = "fuel_type_id")
    private FuelType fuelType;

    @ManyToOne
    @JoinColumn(name = "engine_size_id")
    private EngineSize engineSize;

    @ManyToOne
    @JoinColumn(name = "transmission_id")
    private Transmission transmission;

    private long mileage;

    @ManyToOne
    @JoinColumn(name = "car_colour_id")
    private CarColour carColour;
```

Рис. 3.4 Клас-сутність Car.

CRUD операції для моделей

CRUD — (англ. create read update delete) 4 базові функції управління даними «створення, зчитування, зміна і видалення». Термін «CRUD» також вживають стосовно інтерфейсу користувача. Для прикладу, в web-додатку task list, один із базових об'єктів — це запис про завдання.

Як мінімум, програма повинна надавати корисувачу функції для:

- Додавання оголошень;
- Пошук і зчитування оголошень;
- Редагування оголошень,
- Видалення існуючих оголошень.

Без цих базових операцій програма не може вважатися придатною до користування.

Для створення цих операцій було обрана технологія Spring Data. Вона забезпечує підтримку репозиторіїв для Java Persistence API (JPA). Це полегшує розробку додатків, які потребують доступу до джерел даних JPA. Мета абстракції даних полягає у суттєвому зменшенні кількості коду, необхідного для введення шарів доступу до даних. Центральний інтерфейс у абстрактному репозиторію Spring Data – це інтерфейс Repository.

Цей інтерфейс діє, насамперед, як маркерний інтерфейс для захоплення типів данних, а також допомагає виявити інтерфейс для моделі, який і створить реалізацію CRUD операцій. Типові операції описувати не потрібно такі як збрерігання, видалення, редагування. Якщо ж розробнику потрібен специфічний метод, то він має вказати в назві методу ключові слова, наприклад deleteById буде видяти сутність за індифікатором. Розробнику не потрібно реалізовувати тіло методу, Spring-контейнер зробить це за нього.

Інтерфейс JpaRepository є насілідником батьківського інтерфейсу Repository. Тому для опису CRUD операцій необхідно наслідуватись від JpaRepository, так як він є узагальненим класом йому потрібно передати два

аргументи це модель та тип унікального індтифікатора який використовується в данній моделі

Інтерфейс UserRepository буде описувати CRUD операції для сутності User. Методи інтерфейсу:

Optional<User> findById(int userId) на вхід приймає ід користувача та знаходить відповідний запис в БД і повертає об`єкт користувача, якщо запис не знайдено повертає посилання на null.

Optional<User> findByEmail(int userId) на вхід приймає email користувача та знаходить відповідний запис в БД і повертає об`єкт користувача, якщо запис не знайдено повертає посилання на null.

Colection<User> findOrderById() знаходить всі запис в БД і повертає об`єкти користувачі, якщо запис не знайдено повертає посилання на null.

Інтерфейс CarRepository буде описувати CRUD операції для сутності Car. Методи інтерфейсу:

Optional<Car> findById (Long id) на вхід приймає індитифікатор та знаходить відповідний запис в БД і повертає об`єкт машини, якщо запис не знайдено повертає посилання на null;

Boolen save (Car car) на вхід приймає клас Car та записує його в БД;

Boolen delete (int id) на вхід приймає індитифікатор та знаходить відповідний запис в БД та видаляє його

Інтерфейс CarRepository буде описувати CRUD операції для сутності Post. Методи інтерфейсу:

Colection<Post> findAllById (Long id) на вхід приймає індитифікатор та знаходить відповідний запис в БД і повертає об`єкт Post, якщо запис не знайдено повертає посилання на null;

Colection<Post> findPostsForLast24Hours (Long id) на вхід приймає індитифікатор та знаходить відповідний запис в БД і повертає об`єкт Post, якщо запис не знайдено повертає посилання на null;

Collection<Post> findAllSoldByUserId(int id) на вхід приймає індитифікатор та знаходить відповідний запис в БД і повертає об`єкт Post, якщо запис не знайдено повертає посилання на null;

Collection<Post> findAllNotSold() знаходить відповідний запис в БД і повертає об`єкт Post, якщо запис не знайдено повертає посилання на null;

Інтерфейс ImageFileRepository буде описувати CRUD операції для сутності files. Методи інтерфейсу:

boolean save(ImageFile imageFile)

на вхід приймає фото та записує його в БД;

Boolen delete (int id) на вхід приймає індитифікатор та знаходить відповідний запис в БД та видаляє його

Optional<ImageFile> findById(int id) знаходить відповідний запис в БД і повертає об`єкт ImageFile, якщо запис не знайдено повертає посилання на null;

Інтерфейс BodyTypeRepository буде описувати CRUD операції для сутності files. Методи інтерфейсу:

Collection <BodyType> findById(int userId) на вхід приймає ід та знаходить відповідний запис в БД і повертає об`єкт користувача, якщо запис не знайдено повертає посилання на null.

Optional < BodyType > findById (int id) на вхід приймає індитифікатор та знаходить відповідний запис в БД і повертає об`єкт Post, якщо запис не знайдено повертає посилання на null

Інтерфейс CarColorRepository буде описувати CRUD операції для сутності files. Методи інтерфейсу:

Collection <CarColor> findById(int userId) на вхід приймає ід та знаходить відповідний запис в БД і повертає об`єкт користувача, якщо запис не знайдено повертає посилання на null.

Optional < CarColor> findById (int id) на вхід приймає індитифікатор та знаходить відповідний запис в БД і повертає об`єкт Post, якщо запис не знайдено повертає посилання на null

Інтерфейс CarModelRepository буде описувати CRUD операції для сутності files. Методи інтерфейсу:

Collection <CarModel> findById(int userId) на вхід приймає ід та знаходить відповідний запис в БД і повертає об`єкт користувача, якщо запис не знайдено повертає посилання на null.

Optional < CarModel> findById (int id) на вхід приймає індитифікатор та знаходить відповідний запис в БД і повертає об`єкт Post, якщо запис не знайдено повертає посилання на null

Інтерфейс EngienSizeRepository буде описувати CRUD операції для сутності EngienSize. Методи інтерфейсу:

Collection <EngienSize> findById(int userId) на вхід приймає ід та знаходить відповідний запис в БД і повертає об`єкт користувача, якщо запис не знайдено повертає посилання на null.

Optional < EngienSize> findById (int id) на вхід приймає індитифікатор та знаходить відповідний запис в БД і повертає об`єкт Post, якщо запис не знайдено повертає посилання на null

Опис контроллерів для web-додатку

Контроллер – це такий клас який приймає HTTP запит користувача, опрацьовує його і повертає сторінку представлення в браузер. Аннотація @Controller є обов`язковою для класу, при запуску додатка Spring-контейнер стоворить об`єкт цього класу, і потім буде опрацьовувати запити зі сторони клієнта.

HTTP — протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від Hyper Text Transfer Protocol, протокол передачі гіпер-текстових документів.

Контроллер UserController має такі методи:

- метод loginUser(@ModelAttribute User user, Model model, HttpServletRequest request) повертає сторінку аунтифікації;

- метод register(Model model, @ModelAttribute User user) приймає запит на реєстрацію нового користувача, превіряє правильність введенних данних. Повертає сторінку аутентифікаціїї.

Контроллер PostsController має такі методи:

- метод String getPostList примає запит з індетифікаторм завдання. Виконує пошук відповідного завдання та прикріпленнь які належать завданню. Генерує сторінку і повертає клієнту

метод getMyPostList запит з індетифікаторм завдання. Виконує пошук відповідного завдання та прикріпленнь які належать завданню. Генерує сторінку і повертає клієнту

Контроллер PostController має такі методи:

- метод addPost приймає запит коли користувач зберігає новий пост. Метод звертається до сессії в якій користувач зареєструвався та зберігає новий пост. Далі генерує нові сторінку і повертає клієнту.
- метод getEditPostPage приймає запит на редагування поста. Викононує оновлення запису в БД і повертає оновленну сторінку
- метод deletePost приймає запит на видалення поста. Видаляє відповідний запис в БД і повертає оновленну сторінку.

3.3 Тестування та аналфз результатів

Web-додаток складається з 6 сторінок: головної сторінки, сторінки для реєстрації, сторінка для авторизації, сторінка додавання оголошення. Розглянемо детально усі сторінки окремо:

• Головна сторінка

Головна сторінка уявляє собою список автомобілів, на якій користувач має змогу переглянути список усіх машин які є в продажі та їх характеристики , а також перейти на інші сторінки.(рис. 3.5)

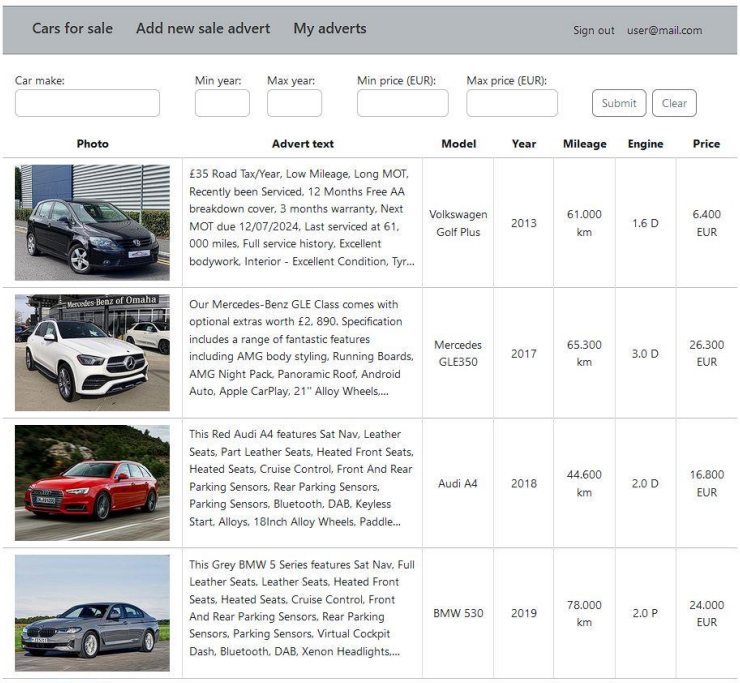

Рис. 3.5 головна сторінки

На сторінці реєстрації(рис. 3.6) користувач має змогу створити власний аккаунт в системі. Користувач повинен ввести данні про електронну пошту, пароль, та телефон. Якщо усі данні введено у вірному форматі, його аккаунт буде створено і його переадресує на головну сторінку.

Рис. 3.6 Сторінка реєстрації

На ній ж ми можемо зробити пошук машиш по певним характеристикам. А також перейти на сторіку детального перегляду авто.

На сторінці перегляду авто(рис. 3.7) можна переглянути авто і детальний опис. А також для власників додатково є редагування інформації про авто.

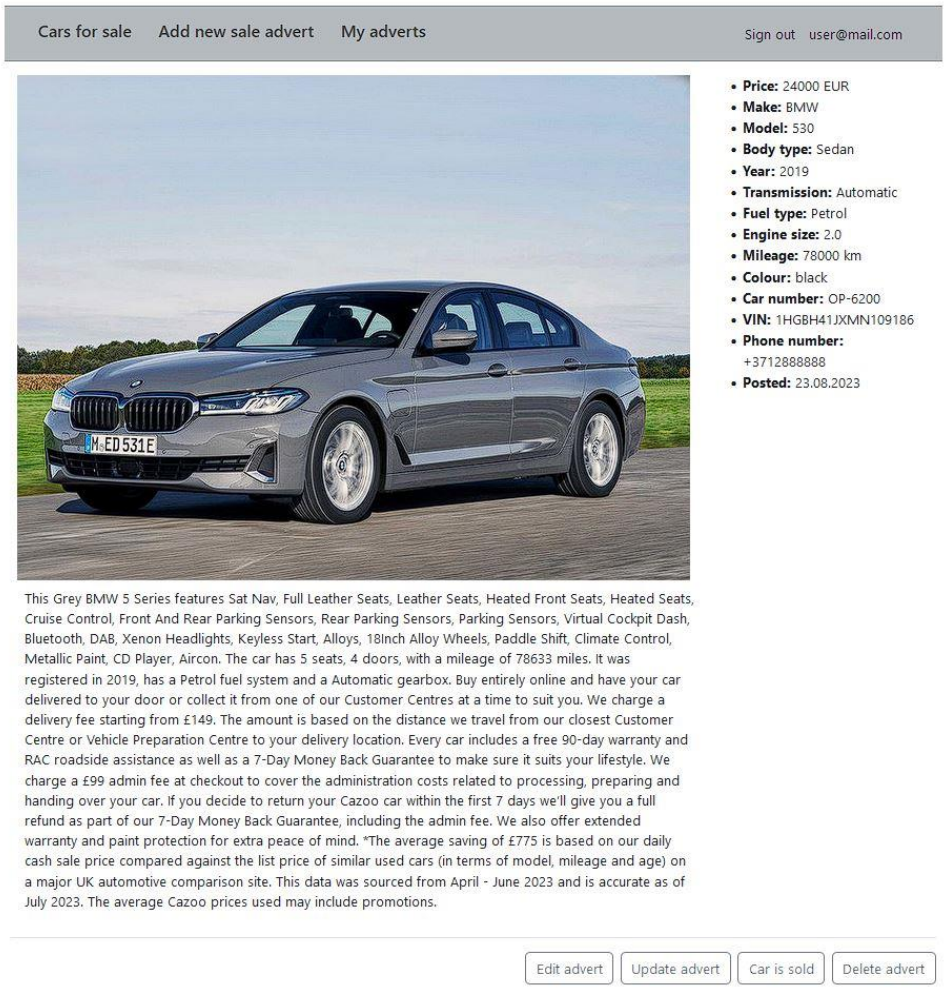

Рис. 3.7 Сторінка перегляду авто

Сторінка перегляду власних авто на рис 3.8 тут користувач може редагувати свої оголошення, а також додавати нові автомобілі для продажу.

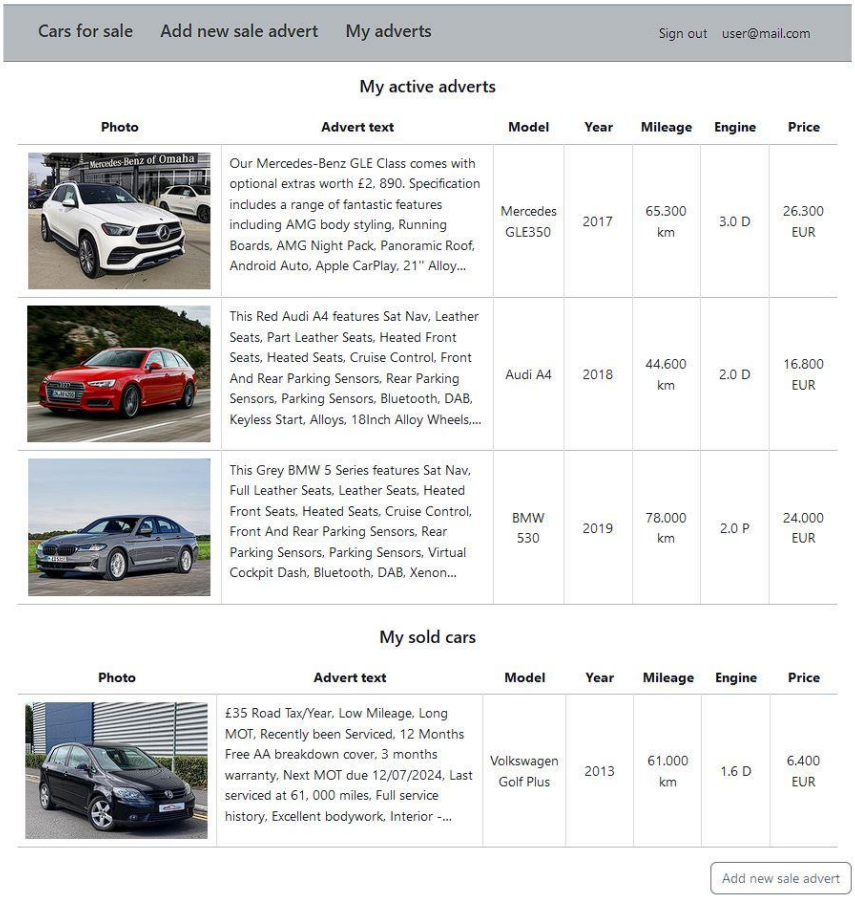

Рис. 3.8 Сторінка перегляду власних авто

ВИСНОВКИ

В ході виконання першої частини роботи – з обґрунтування розробки Web базовної інформаційної системи електронної комерції – були розглянуті основні принци роботи web-додатка, основні ознаки та складові частини. Після проведення аналізу архітектури web-додатка, було виявлено, що саме паттерн MVC є найбільш гнучким та слабозв'язаною системою для web-додатку. Також був проведенний аналіз предметної області. Розглянуто вимоги до web-додатку які відповідають сучасним вимогам та стандартам.

У другій частині роботи розглядаються особливості технології Spring Framework. А також інструменти які використовувалися для розробки системи. Такі як мов програмування Java, база даних PostgreesSql та середовище розробки IntelliJ IDEA.

Завданням даного курсового проєкту було розробка Web базовної інформаційної системи електронної комерції, основна функція якого – зберігання та обробка інформації про хід процесу реалізації проекту. Для імплементації цього завдання були виконані наступні пункти:

1) аналіз та перегляд літературних джерел по темі курсового проекту;

2) аналіз проектування архітектури web-додатка;

3) проектування логічної моделі бази даних;

4) проектування фізичної моделі бази даних;

5) створення БД;

6) створення web- додатку.

7) налаштування середовища для web- додатку.

Для проектування логічної та фізичної моделей бази даних використовувався CASE-засіб ERwinDataModele, який надає можливість автоматично згенерувати скрипт з усіма необхідними SQL-запитами для створення реальної БД. Для створення додатку використовувалося інтегроване середовище розробки Intellij IDEA, мовою програмування була обрана Java.

Візуально додаток виглядає інтуїтивно-зрозуміло для будь-якого користувача, тому для його використання не потрібно проходити спеціальної підготовки. Користовач може шукати автомобілі як заключовими словами так і за детальним пошуком, сортувати найдені авто за ціною за роком випуску, також може додавати редагувати свої оголошення.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Web-додаток [Інтернет-ресурс] / Режим доступу: https://uk.wikipedia.org/wiki/

2. Что такое веб-приложения и динамические веб-страницы [Інтернетресурс] / Режим доступу: https://helpx.adobe.com/ru/dreamweaver/using/webapplications.html

3. Model–view–controller [Інтернет-ресурс] / Режим доступу: https://en.wikipedia.org/wiki/Model-view-controller

4. Spring Framework [Інтернет-ресурс] / Режим доступу: https://spring.io/

5. Як працює оцінювання задач у Kanban на практиці [Інтернет-ресурс] / Режимдоступу: https://www.scrum.ua/post/yak-pratsyue-otsinyuvannya-zadach-u-kanban-na-praktitsi

6. Базы данных: модели, разработки, реализация / Т.С. Карпова. - СПб.: Питер, 2002. – 240 с.

ДОДАТОК А

Файл “Prom.xml”

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

<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>ru.job4j</groupId>
  <artifactId>job4j_cars</artifactId>
  <version>1.0-SNAPSHOT</version>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.6</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.thymeleaf.extras</groupId>
      <artifactId>thymeleaf-extras-java8time</artifactId>
      <version>3.0.4.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>com.google.code.findbugs</groupId>
      <artifactId>jsr305</artifactId>
      <version>3.0.2</version>
    </dependency>
    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <version>42.5.1</version>
    </dependency>
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <version>2.1.214</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-test</artifactId>
  </dependency>
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-dbcp2</artifactId>
    <version>2.9.0</version>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.26</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.6.11.Final</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
      <version>3.1.2</version>
      <dependencies>
        <dependency>
          <groupId>com.puppycrawl.tools</groupId>
          <artifactId>checkstyle</artifactId>
          <version>10.3.1</version>
        </dependency>
      </dependencies>
      <executions>
        <execution>
          <id>validate</id>
          <phase>validate</phase>
          <configuration>
            <configLocation>checkstyle.xml</configLocation>
            <encoding>UTF-8</encoding>
            <consoleOutput>true</consoleOutput>
            <failsOnError>true</failsOnError>
            <includeTestSourceDirectory>true</includeTestSourceDirectory>
          </configuration>
          <goals>
            <goal>check</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <groupId>org.liquibase</groupId>
      <artifactId>liquibase-maven-plugin</artifactId>
      <version>4.15.0</version>
      <configuration>
        <propertyFile>${liquibase.config}</propertyFile>
      </configuration>
```

```
      <executions>
        <execution>
          <phase>process-resources</phase>
          <goals>
            <goal>clearCheckSums</goal>
            <goal>update</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
 </build>

 <profiles>
   <profile>
     <id>test</id>
     <properties>
       <liquibase.config>db/liquibase_test.properties</liquibase.config>
     </properties>
     <activation>
       <activeByDefault>true</activeByDefault>
     </activation>
   </profile>
   <profile>
     <id>production</id>
     <properties>
       <liquibase.config>db/liquibase.properties</liquibase.config>
     </properties>
   </profile>
 </profiles>

</project>
```

ДОДАТОК Б

Файл “PostController. java”

```
package ru.job4j.cars.controller;

import lombok.AllArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import ru.job4j.cars.dto.ImageFileDto;
import ru.job4j.cars.model.Car;
import ru.job4j.cars.model.Post;
import ru.job4j.cars.model.User;
import ru.job4j.cars.service.*;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import java.util.Optional;

@Controller
@AllArgsConstructor
@RequestMapping("/post")
public class PostController {

    private final PostService postService;

    private final CarModelService carModelService;

    private final BodyTypeService bodyTypeService;

    private final FuelTypeService fuelTypeService;

    private final EngineSizeService engineSizeService;

    private final TransmissionService transmissionService;

    private final CarColourService carColourService;

    private final CarService carService;

    @GetMapping("/{postId}")
    public String getOnePostPage(@PathVariable int postId, Model model) {
        Optional<Post> optionalPost = postService.findById(postId);
        model.addAttribute("post", optionalPost.get());
        return "post/one";
    }

    @GetMapping("/create")
    public String getPostCreatePage(Model model) {
        model.addAttribute("carModels", carModelService.findAll());
        model.addAttribute("allBodyTypes", bodyTypeService.findAll());
        model.addAttribute("allFuelTypes", fuelTypeService.findAll());
        model.addAttribute("allEngineSizes", engineSizeService.findAll());
        model.addAttribute("allTransmissions", transmissionService.findAll());
        model.addAttribute("allCarColours", carColourService.findAll());
```

```
        return "post/create";
    }

    @PostMapping("/create")
    public String addPost(Model model,
                          @ModelAttribute Car car,
                          @RequestParam String description,
                          @RequestParam Integer price,
                          @RequestParam MultipartFile imageFileDto,
                          HttpServletRequest request) throws IOException {
        carService.save(car);
        Post post = new Post();
        post.setCar(car);
        post.setDescription(description);
        post.setPrice(price);
        HttpSession session = request.getSession();
        User user = (User) session.getAttribute("user");
        post.setUser(user);
        boolean ifSaved = postService.save(post, new ImageFileDto(imageFileDto.getOriginalFilename(),
imageFileDto.getBytes()));
        if (!ifSaved) {
            model.addAttribute("message", "Error! Advert is not added");
            return "post/create";
        }
        return "redirect:/";
    }

    @GetMapping("/change/{postId}")
    public String getEditPostPage(@PathVariable int postId, Model model) {
        Optional<Post> optionalPost = postService.findById(postId);
        model.addAttribute("post", optionalPost.get());
        return "post/change";
    }

    @PostMapping("/change/{postId}")
    public String editPost(@ModelAttribute Post post) {
        postService.update(post);
        return "redirect:/posts/mylist";
    }

    @GetMapping("/sold/{postId}")
    public String movePostToSold(@PathVariable int postId) {
        postService.movePostToSold(postId);
        return "redirect:/posts/mylist";
    }

    @GetMapping("/delete/{postId}")
    public String deletePost(@PathVariable int postId) {
        postService.delete(postId);
        return "redirect:/posts/mylist";
    }

    @GetMapping("/updateDate/{postId}")
    public String updatePost(@PathVariable int postId) {
        postService.updateDate(postId);
        return "redirect:/posts/mylist";
    }

}
```

ДОДАТОК В
Файл “PostRepository.java”

```
package ru.job4j.cars.repository;

import lombok.AllArgsConstructor;
import org.springframework.stereotype.Repository;
import ru.job4j.cars.model.Post;

import java.time.LocalDateTime;
import java.util.Map;
import java.util.Optional;

@Repository
@AllArgsConstructor
public class HibernatePostRepository implements PostRepository {

    private final CrudRepository crudRepository;

    @Override
    public boolean save(Post post) {
        return crudRepository.ifSaved(post);
    }

    @Override
    public boolean update(Post post) {
        return crudRepository.ifChanged(post);
    }

    @Override
    public boolean delete(int postId) {
        return crudRepository.ifChanged(
                "DELETE Post WHERE id = :pId",
                Map.of("pId", postId)
        );
    }

    @Override
    public Optional<Post> findById(int postId) {
        return crudRepository.optional(
                "FROM Post p JOIN FETCH p.priceHistories WHERE p.id = :pId", Post.class,
                Map.of("pId", postId)
        );
    }

    @Override
    public boolean movePostToSold(int id) {
        return crudRepository.ifChanged("UPDATE Post SET isSold = true WHERE id = :pId",
                Map.of("pId", id));
    }

    @Override
    public boolean updateDate(int id) {
        return crudRepository.ifChanged("UPDATE Post SET created = :pDate WHERE id = :pId",
                Map.of("pId", id, "pDate", LocalDateTime.now()));
    }
}
```

ДОДАТОК Г
Файл “PostService. java”

```
package ru.job4j.cars.service;

import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import ru.job4j.cars.dto.ImageFileDto;
import ru.job4j.cars.model.Post;
import ru.job4j.cars.model.PriceHistory;
import ru.job4j.cars.repository.PostRepository;
import ru.job4j.cars.repository.PriceHistoryRepository;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;

@Service
@AllArgsConstructor
public class SimplePostService implements PostService {

    private final PostRepository postRepository;
    private final PriceHistoryRepository priceHistoryRepository;
    private final ImageFileService imageFileService;
    private final CarService carService;

    @Override
    public boolean save(Post post, ImageFileDto imageFileDto) {
        post.setCreated(LocalDateTime.now());
        post.setPriceHistories(List.of(addNewPriceHistory(post)));
        if (imageFileDto.getName().isEmpty()) {
            post.setImageFile(imageFileService.getDefault());
        } else {
            post.setImageFile(imageFileService.save(imageFileDto));
        }
        return postRepository.save(post);
    }

    private PriceHistory addNewPriceHistory(Post post) {
        PriceHistory priceHistory = new PriceHistory();
        priceHistory.setPrice(post.getPrice());
        priceHistory.setCreated(post.getCreated());
        return priceHistory;
    }

    @Override
    public boolean update(Post post) {
        boolean ifUpdated = false;
        post.setCreated(LocalDateTime.now());
        Optional<Post> opt = postRepository.findById(post.getId());
        if (opt.isPresent()) {
            Post postOld = opt.get();
            if (postOld.getPrice() != post.getPrice()) {
                postOld.getPriceHistories().add(addNewPriceHistory(post));
            }
            post.setPriceHistories(postOld.getPriceHistories());
            post.setUser(postOld.getUser());
            post.setCar(postOld.getCar());
            post.setImageFile(postOld.getImageFile());
            ifUpdated = postRepository.update(post);
```

```
        }
        return ifUpdated;
    }

    @Override
    public boolean delete(int postId) {
        Optional<Post> optPost = findById(postId);
        if (optPost.isEmpty()) {
            return false;
        }
        Post post = optPost.get();
        if (!post.getPriceHistories().isEmpty()) {
            post.getPriceHistories().forEach(p -> priceHistoryRepository.delete(p.getId()));
        }
        boolean ifDeleted = postRepository.delete(postId);
        if (ifDeleted) {
            carService.delete(post.getCar().getId());
            imageFileService.delete(post.getImageFile().getId());
        }
        return ifDeleted;
    }

    @Override
    public Optional<Post> findById(int postId) {
        return postRepository.findById(postId);
    }

    @Override
    public boolean movePostToSold(int id) {
      return postRepository.movePostToSold(id);
    }

    @Override
    public boolean updateDate(int id) {
      return postRepository.updateDate(id);
    }
}
```