МІНІСТЕРСТОВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп‘ютерної та програмної інженерії
Кафедра комп’ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ А.С. Савченко
«____» _____ 20___р

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СПУПЕНЯ «БАКАЛАВР»

Тема: «Інформаційна система ведення ремонтного процесу автомобільних двигунів»

Виконавець: студент УС-412 Швець Владислав Сергійович
(студент, група, призвіще, ім'я, по батькові)

Керівник: к. т. н., доцент Савченко Аліна Станіславівна
(науковій ступень, вчене звання, призвіще, ім'я, по батькові)

Нормоконтролер: ст. викл. Шевченко О.П. _____
(П.І.Б.) (підпис)

КИЇВ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп‘ютерної та програмної інженерії

Кафедра комп’ютерних інформаційних технологій

Освітній ступінь **Бакалавр**

Галузь знань, спеціальність, спеціалізація: 12 “Інформаційні технології”, 122 “Комп’ютерні науки ”, “Інформаційні управляючі системи та технології (за галузями)”

ЗАТВЕРДЖУЮ

Завідувач кафедри

А.С. Савченко

“_____”_____ 2021 р.

ЗАВДАННЯ

на виконання дипломного проекту студента

Швеця Владислава Сергійовича

(прізвище, ім’я, по батькові)

1. Тема проекту: «Інформаційна система ведення ремонтного процесу автомобільних двигунів» затверджена наказом ректора № 636/ст. від 22.04.2021р.

2. Термін виконання роботи: з 10.05.2021 по 14.06.2021р.

3. Вихідні дані до роботи: організація інформаційної системи ведення ремонтного процесу автомобільних двигунів для компанії ProMotor.

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): вступ, аналітичний огляд і постановка задачі, огляд технологій та інструментів для розробки системи, розробка функціональності системи та інтерфейсу для користувачів, висновки.

5. Перелік обов’язкового графічного матеріалу: схема рівнів архітектури системи, інформаційна діаграма бази даних системи.

КАЛЕНДАРНИЙ ПЛАН

	Етапи виконання дипломної роботи	Термін виконання етапів	Примітк
1	Аналіз літератури та джерел за темою дипломного проекту.	10.05.2021р. – 11.05.2021р.	
2	Розробка та затвердження плану дипломного проекту.	12.05.2021р.	
3	Проведення консультації з науковим керівником щодо створення першого розділ.	13.05.2021р.	
4	Аналітичний огляд і постановка задачі.	14.05.2021р. – 16.05.2021р.	
5	Порівняльний аналіз існуючих систем управління документами.	17.05.2021р. – 19.05.2021р.	
6	Огляд технологій для розробки системи.	20.06.2021р. – 21.05.2021р.	
7	Розробка компонентів системи.	22.05.2021р. – 29.05.2021р.	
8	Висновки та оформлення пояснювальної записки дипломного проекту.	30.05.2021р. – 31.05.2021р.	
9	Підписання необхідних документів у встановленому порядку.	1.06.2021р. – 2.06.2021р.	
10	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломного проекту. Підготовка необхідних документів	03.06.2021р. – 14.06.2021р.	

7. Дата видачі завдання: 10.05.2021 р.

Керівник дипломного проекту _____ Савченко А.С.

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ _____

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Інформаційна система ведення ремонтного процесу автомобільних двигунів для компанії ProMototr»: 16 літературних джерел.

Об’єкт дослідження: інформаційне супроводження робочого процесу автомобільних двигунів.

Предмет дослідження: інформаційна система ведення ремонтного процесу автомобільних двигунів для компанії ProMotor.

Мета роботи: розробка інформаційної системи ведення ремонтного процесу автомобільних двигунів для прискорення та полегшення роботи співробітників компанії ProMotor.

Методи дослідження, технічні та програмні засоби: розробка бази даних за допомогою MariaDB, порівняльний аналіз, компоненти на основі Hibernate, засоби фреймворків VueJs та Spring, обробка літературних джерел.

Отримані результати та їх новизна: розроблено інформаційну систему ведення ремонтного процесу автомобільних двигунів з централізованою базою даних та веб-інтерфейсом для майстрів компанії ProMotor.

Ключові слова: СИСТЕМА УПРАВЛІННЯ, БАЗА ДАНИХ, ПРОЦЕС РЕМОНТУ, SPRING, VUEJS, JAVA, JAVASCRIPT, MARIADB.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ТЕРМІНІВ ТА СКОРОЧЕНЬ

СУБД – система управління базами даних

БД – база даних

HTML – HyperText Markup Language – мова гіпертекстової розмітки

CSS – Cascading Style Sheets – каскадні таблиці стилів

MVC – Model – View – Controller – модель представлення контролера

JSP - Java Server page – сторінки серверу Java

SQL – Structured Query Language – мова структурованих запитів

DAO – Data Access Object – объект доступа к данным

ВСТУП

Актуальність теми. На сьогоднішній день багато компаній переводять процес ведення документації з паперового варіанту на оптимізовані інформаційні системи. Такі процеси притаманні економічним, бухгалтерським та рекламним компаніям, тобто там, де немає потреби у ручній праці. Серед компаній технічного спрямування, наприклад, з ремонту автомобільних двигунів, така супровідна документація до процесів досі залишається у паперовому варіанті. Одна з причин обумовлена тим що, в багатьох випадках технічні фахівці (майстри з ремонту двигунів) в таких компаніях не мають досвіду роботи з веб додатками. Саме це спричиняє уповільнення роботи та незахищеність даних.

Одним з основних методів оптимізації робочого процесу є створення web-додатку. Це забезпечує швидкий доступ до потрібних даних, на відмінну від ПК-додатків. Існують величезні хмарні сховища даних, на основі яких створюються інформаційні системи, які заміняють великі склади де зберігалися документи з даними. Отже, розробка інформаційної системи для супроводження процесу ведення ремонту автомобільних двигунів зі зручним та інтуїтивно зрозумілим інтерфейсом є актуальною здачею.

Метою дипломного проекту є розробка інформаційної система ведення ремонтного процесу автомобільних двигунів для прискорення та полегшення роботи співробітників компанії ProMotor.

Для досягнення поставленої мети слід вирішити наступні завдання:

- розглянути теоретичну основу питання: поняття інформаційна система;
- дослідити існуючі інформаційні системи із захистом персональних даних;
- ознайомитися із засобами розробки всіх частин інформаційної системи та програмного забезпечення захисту даних;

- реалізувати інформаційну систему ведення процесу ремонту автомобільних двигунів, що забезпечує захист персональних даних за допомогою програмних і апаратних засобів.

.

РОЗДІЛ 1
ОБЛАСТЬ ЗАСТОСУВАННЯ СИСТЕМИ ТА ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ІСНУЮЧИХ СИСТЕМ

1.1. Предметна область

На сьогоднішній день масу заводів, мережа СТО та майстерень витрачають багато часу, для знаходження потрібних ремонтних параметрів, оскільки знаходження потрібної деталі потрібного двигуна в інтернеті, представляє дуже складну проблему (іноді не вирішувану). Будова двигуна має безліч нюансів, через що створення бази інформації по двигунах стає необхідністю для багатьох робітників. Найчастіше вони ведуть ручний запис даних в excel, який має масу недоліків, так як це десктопна програма не призначена для цієї області. Отже розробка сучасної інформаційної системи, яка враховує специфіку області застосування та має зручний інтерфейс дозволить вирішити вищезгадані проблеми, оскільки загально доступний веб-додаток, що будь-який робочий може з будь-якого комп'ютера моментально запустити і працювати з величезною базою даних параметрів.

Одна з головних можливостей системи - це швидкий пошук потрібних даних. Так як автомобіль складається з багатьох частин, які інколи не співпадають між собою, то в даній системі створено дерево параметрів. Воно відповідає за створення залежності між автомобілем та ремонтними параметрами цього автомобіля. Однією з ключових особливостей цього дерева є можливість

динамічно змінюватись, при цьому не треба наново збирати бекенд та фронтенд, це може зробити звичайниймайстер. Також крім ремонтних параметрів існують і параметри автомобіля наприклад: пальне, виробник, потужність двигуна, хід поршнів.

1.2. Порівняння існуючих систем

Основне завдання – це зручна і зрозуміла інформаційна система ведення ремонтного процесу автомобільних двигунів. В дипломному проекті використано різні способи вводу даних. Крім полів вводу є випадаючі списки для зручного обрання потрібного фільтру, які в свою чергу обновлюють дані у всіх фільтрах, та процентний пошук, який забезпечує більш легкий пошук. За допомогою цього пошуку можна знайти більше даних ніж зазвичай, тому що коли майстер вписує дані, то точний пошук може не видати результату, а процентний створює похибку, яка дозволяє розширити пошук. Існують кілька різних рішень описаної вище задачі від інших розробників, але в основному для заводів використовують десктопні програми, тому буде проведено огляд як конкурента wsmengine (http://app.wsmengine.com.ua/). Wsmengine - це прямий конкурент.

Для порівняння систем, треба визначити характеристики за якими будемо порівнювати ці системи, а саме:

1. зручність
2. швидкість
3. гнучкість
4. зрозумілість
5. доступність

Порівняння представлено в наступній таблиці 1.2. (для зручності проектовану систему будем називати engprocess).

Таблиця 1.2.

Порівняльна характеристика

Характеристика	Wsmengine	Проектована система Engprocess
Зручність	Старий не практичний дизайн, складний для не експерта в даній області	Сучасний дизайн, не переватажений зайвими деталями
Швидкість	Задовільна швидкість, але через необхідність повного початкового завантаження всієї інформації повільніше ніж проектована система engprocess	Висока швидкість за рахунок часткового завантаження даних у фоновому режимі
Гнучкість	Параметри представлені у вигляді таблиці з текстовими даними, що ускладнює створення нового функціонала	Високий ступінь розширюваності системи. Продумана структура бази даних, дозволяє в подальшому нарощувати функціонал інформаційної системи.
Зрозумілість	Для не професіонала складно орієнтуватися на сайті	Інтуїтивно зрозумілий інтерфейс для пересічного користувача66
Доступність	Доступна за місячну плату	Безкоштовно

1.3. Процес ремонту автомобільних двигунів

Однією з найважливіших частин автомобіля як і раніше залишається двигун. З метою економії коштів власник автомобіля стикається з вибором, відремонтувати або купити новий двигун. Дуже важливо знайти такий автомобільний сервіс, де ремонт і

обслуговування двигуна проводять якісно і швидко. Часто буває так, що господар автомобіля надовго розлучається зі своїм автомобілем через не професіоналізм працівників СТО.

Найрізноманітніші роботи з капітального ремонту двигуна автомобіля починаються з діагностики двигуна. Але для правильного огляду та знаходження рішення потрібні дані пов'язані безпосередньо з конкретним двигуном, так як їх багато, то ремонтних даних теж. Головне завдання дипломного проекту полегшити і прискорити вирішення проблеми.

Капітальний ремонт двигунів автомобілів повинен здійснюватися на спеціалізованих станціях технічного обслуговування. Досвідчені фахівці з ремонту двигунів внутрішнього згоряння зможуть взяти на ремонт і обслуговування практично будь-який автомобіль.

Розглянемо ремонтні роботи, які на багато прискорить впровадження інформаційної системи ведення ремонтного процесу:

1. Заміна гільз і деталей шатунно-поршневої групи двигуна;
2. Заміна гільз і деталей шатунно-поршневої групи двигуна;
3. Розбірно-складальні роботи деталей циліндро-поршневої групи;
4. Комплектування деталей гільзо-поршневої групи,
5. Ремонт і установка шатунно-поршневої групи двигуна;
6. Заміна прокладки масляного піддону двигуна;
7. Заміна головки блоку циліндрів або ремонт головки блоку;
8. Заміна прокладки впускного і випускного колектора;
9. Заміна сальника колінчастого вала;
10. Заміна поршневих кілець;
11. Встановлення нових поршнів;
12. Шліфування колінчастого валу.

Без зняття двигуна можна провести наступні ремонти елементів двигуна:

1. Ремонт водяного насоса;
2. Ремонт стартера;

3. Ремонт розподільника запалювання;
4. Ремонт генератора;
5. Ремонт бензонасоса;
6. Ремонт карбюратора;
7. Ремонт головки блоку циліндрів;
8. Ремонт клапанного механізму.
9. Заміна поршневих кілець;
10. Заміна шатунно-поршневої групи.

Висновки до розділу 1

Отже, в даній сфері немає великої кількості рішень які багато майстерні могли б використовувати, а аналоги мають свої недоліки. На основі вивчення конкурентів і аналізу їхніх помилок, визначено, що система повинна бути зручною, швидкою і зрозумілою навіть для звичайної людини, який захотів дізнатися ремонтні розміри деталей свого двигуна.

РОЗДІЛ 2
ОПИС АРХІТЕКТУРИ СИСТЕМИ ТА ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ

На сьогоднішній день для створення складної системи є багато шляхів реалізації. Дивлячись на сферу діяльності та спосіб застосування, обирають потрібні технології та мови програмування. Для інформаційної системи ведення ремонтного процесу автомобільних двигунів було обрано створити веб-додаток, так як це передбачає переваги наприклад: доступність, швидкість, масштабованість, захищеність даних, адаптивність і зрозумілість інтерфейсу. Складна і багаторівнева структура сучасних веб-додатків вимагає ієрархічного поділу процесу їх розробки. Історично цей процес поділяється на дві частини: front-end (клієнтська) і back-end (серверна). Для розробки дипломного проекту потрібно розібрати в кожен етап створення клієнтська та серверна частини, поговоримо про їх відмінностях, точки дотику і зонах відповідальності.

2.1. Складові інформаційно-управляючої системи

Для створення складної інформаційної системи, як зазначено вище, потрібно розробити фроненд (клієнтська) і бекенд (серверна), також для зберігання даних потрібно використовувати базу даних. При розробці сучасних систем використовуються фреймворки (програмна структура). Для фроненда була обрана мова програмування JavaScript, також був узятий фреймворк-Vue.js. В якості мови розробки бекенда була взята Java, так як це мова створена для розробки складних систем, а фреймворк був обраний Spring Framework.

Також щоб забезпечити управління створенням і використанням баз даних потрібно вибрати СУБД (Система Управління Базами Даних), так як в дипломному проекті потрібна вільна система управління і високу сумісність з MySQL, то кращім варіантом буде MariaDB.

Для розуміння взаємодії між цими частинами ми розглянемо full-stack архітектуру (рис. 1.1).

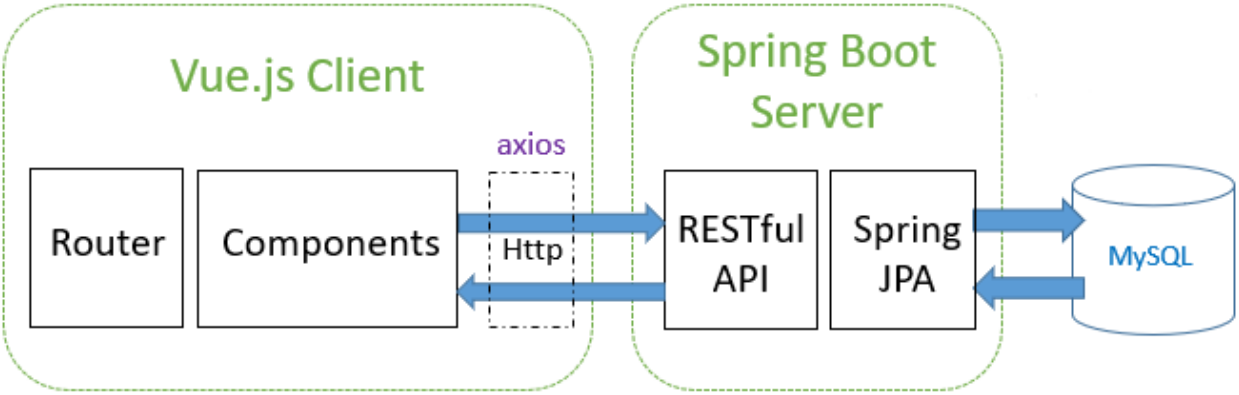

Рис. 1.1. Full-stack архітектуру дипломного проекта

2.1.1. Взаємодія між backend і frontend

Якщо говорити простою мовою, то взаємодія між бекенд і фронтенд відбувається по колу:

1. фронтенд відправляє призначену для користувача інформацію в бекенд;
2. бекенд обробляє дані;
3. інформація повертається назад, приймаючи зрозумілу для користувача форму або відповідному форматі.

Варіанти взаємодії бекенд і фронтенд:

1. HTTP-запит. Безпосередньо відправляється на сервер, сервер шукає дані, вбудовує їх в шаблон, а потім повертає у вигляді HTML-сторінки. Між отриманням запиту і відповіддю на нього сервер зазвичай шукає по сформованому запиту

інформацію в БД. За допомогою HTML визначається, що буде показано, а CSS - як все буде виглядати. JS необхідний для особливих взаємодій;

2. використання інструментарію AJAX (Asynchronous JavaScript and XML). Запит відправляється за допомогою JavaScript, завантаженим в браузер. Відповідь приходить у форматі JSON або XML;

3. односторінкові додатки. Вони завантажують дані без оновлення веб-сторінки. Це також здійснюється за допомогою AJAX або за допомогою фреймворків Ember і Angular;

4. Ember або бібліотека React. Вони допомагають використовувати додаток і в клієнті, і на сервері. Backend і frontend взаємодіють за допомогою AJAX і HTML-коду, оброблюваного на сервері.

5. Для дипломного проекту обрано інструментарій AJAX, тому що це простий надійний спосіб для взаємодії бекенда і фронтенда, який для системи дозволить принести наступні переваги:

6. зменшити трафік (через скорочення розміру даних переданих між сервером і браузером);

7. знизити навантаження на сервер (тому що не потрібно генерувати різні елементи, що повторюються на сторінці, а тільки той контент, який необхідно передати);

8. прискорити чуйність інтерфейсу сторінки і його мерехтіння (через те що немає необхідності в повній перезавантаження сторінки);

9. поліпшити інтерактивність (тобто більш зручно вести діалог (здійснювати взаємодію) з користувачем)

2.1.2. Бекенд

Бекенд - це серверна логіка веб-додатки. Тут реалізується робота зі сховищем даних, тому людина, що займається розробкою серверної логіки, повинен мати навички роботи з базами даних, а також досвід використання механізмів ORM. Саме від бекенд-розробника залежить продуктивність серверного коду, його масштабованість, безпеку і раціональність.

Крім серверної логіки в сферу відповідальності бекенда входить налагодження та прототипування з використанням клієнтської частини програми. Це тягне за собою необхідність розуміння роботи стека протоколів TCP / IP, HTTP, REST / SOAP, принципів взаємодії браузера з веб-додатком.

Незважаючи на те, що сфера фронтенда традиційно вважається найбагатшою і різноманітною в плані технологій, бекенд також має широкий спектр інструментів розробки. Крім канонічного PHP, міцні ніші зайняли Python з фреймворком Django, Java і Node.js, Ruby, а також набирає популярність Go. Як вказав вище в дипломному проекті була використана Java.

2.1.3. Фронтенд

Frontend - це публічна частина web-додатків (вебсайтів), з якої користувач може взаємодіяти і контактувати напряму. У Frontend входить відображення функціональних завдань, призначеного для користувача інтерфейсу, що виконуються на стороні клієнта, а також обробка запитів користувачів. По суті, фронтенд - це все те, що бачить користувач при відкритті web-сторінки.

У свою чергу, web-додаток - клієнт-серверний додаток, в якому клієнтом виступає в основному браузер, а сервером - web-сервер. Логіка web-додатки розподілена між сервером і клієнтом, зберігання даних здійснюється переважно на сервері, обмін інформацією відбувається по мережі. Простіше кажучи, це те, що

бачить користувач і які дії виконує кожен раз, коли підключається до мережі інтернет і відкриває будь-який браузер.

Frontend-розробка - це робота зі створення публічної частини web-додатки, з якої безпосередньо контактує користувач, і функціоналу, який зазвичай виконується на стороні клієнта. Тобто, фронтенд розробник працює над тим, щоб на сайті кожна кнопочка, іконка, текст і вікно не тільки стояли на своєму місці, не перекривали один одного і виглядали цілісно (це веб-верстка), але і щоб вони виконували своє пряме призначення - виробляли якісь дію (наприклад, щоб кнопка "купити" відкривала кошик, а "play" - запускала відтворення фільму або музики).

З метою створення затребуваного і доступного продукту (веб-додатки) фронтенд-розробнику необхідно взаємодіяти з іншими програмістами, дизайнерами, маркетологами, аналітиками та іншими фахівцями.

Компоненти фронтенд розробки:

1. HTML (HyperText Markup Language) кажучи простими словами - це мова розмітки всіх елементів і документів на сторінці, і їх взаємодію в структурі сторінки.

2. CSS (Cascading Style Sheets) - це мова характеристики і стилізації зовнішнього вигляду документа. За допомогою CSS-коду браузер розуміє, як саме необхідно відображати елементи. CSS створює шрифти, кольори, визначає розташування блоків сайту, і інше. Також адаптує один і той же документ в різних стилях, виводить передачу на екран або для читання голосом.

JavaScript - мова, створена оживити веб-сторінки. Завдання JavaScript - відгукуватися на дії користувача, обробляти натискання клавіш, переміщення курсора, кліки мишкою. JavaScript також дає можливість вводити повідомлення, посилати запити на сервер, а також завантажує дані без перезавантаження сторінки, і так далі.

2.2. Опис реляційної бази даних та СУБД MariaDB

Найголовніше для інформаційної системи є робота з даними. Щоб працювати з даними їх потрібно зберігати. Для цього я використовую реляційну базу даних

Поняття бази даних (БД) абстрактне. Конкретними реалізаціями є бази даних чогось. Наприклад, база даних бібліотеки, сайту або база даних магазину, в якій зберігаються відомості про співробітників, товари, постачальників і покупців.

2.2.1. Реляційна база даних

Реляційна база даних організовує дані в таблиці, які можна пов’язати на основі ідентифікаторів або даних, загальних для кожної. Ця можливість дозволяє отримати абсолютно нову таблицю з даних в одній або декількох таблицях за допомогою одного запиту. Це також дозволяє краще зрозуміти взаємозв'язок між усіма наявними даними та отримати нові уявлення для прийняття кращих рішень або виявлення нових можливостей.

Наприклад, компанія веде таблицю клієнтів, яка містить дані компанії про кожен рахунок клієнта та одну або кілька таблиць транзакцій, що містять дані, що описують окремі транзакції.

Стовпцями (або полями) для таблиці клієнтів можуть бути ідентифікатор клієнта, назва компанії, адреса компанії тощо; стовпцями таблиці транзакцій можуть бути дата транзакції, ідентифікатор клієнта, сума транзакції, спосіб оплати тощо. Таблиці можуть бути пов’язані на основі загального поля ідентифікатора клієнта. Таким чином, ви можете запитати таблицю для отримання цінних звітів, таких як зведена заява клієнта.

Всі стовпчики в одній таблиці повинні мати унікальні імена, однак дозволяється присвоювати однакові імена стовпцями, розташованим в різних таблицях. На практиці такі імена стовпців, як NAME, ADDRESS, QTY, PRICE і SALES, часто зустрічаються в різних таблицях однієї бази даних.

Стовпці таблиці впорядковані зліва направо, і їх порядок визначається при створенні таблиці. У будь-якій таблиці завжди є як мінімум один стовпець. У стандарті ANSI / ISO не вказується максимально допустиму кількість стовпців в таблиці, проте майже у всіх комерційних СУБД ця межа існує і зазвичай становить

приблизно 255 стовпців. В відміну від стовпців, рядки таблиці не мають певного порядку. Це значить, що якщо послідовно виконати два однакових запиту для відображення вмісту таблиці, немає гарантії, що обидва рази рядки будуть перераховані в одному і тому ж порядку. У таблиці може міститися будь-яку кількість рядків. Цілком припустимо існування таблиці з нульовою кількістю рядків. Така таблиця називається порожній. Порожня таблиця зберігає структуру, визначену її стовпцями, просто в ній не міститься дані. Стандарт ANSI / ISO не накладаються обмежень на кількість рядків у таблиці, і в багатьох СУБД розмір таблиць обмежений лише вільним дисковим простором комп'ютера. В інших СУБД є максимальний межа, однак він досить високий - близько двох мільярдів рядків, а іноді і більше.

Особливістю реляційної бази даних є використання в ній реляційної моделі даних і що випливають з цього наслідки:

Модель даних в реляційних БД визначена заздалегідь. Є строго типізований, містить обмеження і відносини для забезпечення цілісності даних.

Модель даних заснована на природному поданні містяться даних, а не на функціональності додатку.

Модель даних піддається нормалізації, щоб уникнути дублювання даних. Нормалізація породжує відносини між таблицями. Відносини пов'язують дані різних таблиць.

2.2.2. MariaDB

Сервер MariaDB - одна з найпопулярніших реляційних баз даних з відкритим кодом. Він зроблений оригінальними розробниками MySQL і гарантовано залишатиметься відкритим. Це частина більшості хмарних пропозицій та за замовчуванням у більшості дистрибутивів Linux.

Він побудований на ефективності, стабільності та відкритості, а Фонд MariaDB забезпечує прийняття внесків з технічної точки зору. Остання нова функціональність включає розширену кластеризацію з Galera Cluster 4, функції

сумісності з Oracle Database та Temporal Data Tables, що дозволяє запитувати дані, як вони стояли в будь-якій точці на минулого.

Як і у всіх open-source проектів, у MySQL трапився вдалий форк, який отримав назву MariaDB. І материнська СУБД, і її відгалуження, носять імена дочок творця: Мю і Марія. Цю систему звикли називати альтернативою MySQL, однак це в корені невірна заяву.

Метою розробників «Марії» було створити продукт, повністю сумісний з MySQL, але значно покращений. Наприклад, движком для зберігання даних в MySQL була MyISAM. У Марії - це Aria, яка подарувала СУБД велику продуктивність, в порівнянні з основним проектом. І, хоча MariaDB заснована на MySQL, останні версії містять не більше ніж 25% оригінального коду.

Марія може похвалитися більш високою продуктивністю в цілому. Особливо це стосується перекодування символів. На високих обсягах інформації коефіцієнт досягає більш ніж 2%. Налагоджувальний код теж оптимізований, в порівнянні з MySQL. В цілому, розробники відзначають високу швидкість розробки, ніж міг видати «батько». Спільнота, яка працює над MariaDB обіцяє ще більші поліпшення.

Крім того, сам користувач може покращувати і оптимізувати роботу Марії. Що відрізняє цю СУБД від всіх інших, так це повноцінний open-source: ніяких закритих елементів або модулів, все в доступі. Грати з кодом можна необмежено, як і робити пропозиції щодо зміни спільноті, яке і розробляє MariaDB.

2.3. Розробка бекенда та опис фреймворків

Для розробки інформаційної системи обрано мову програмування Java. В якості фреймворка обрав Spring - це універсальний фреймворк, який добре підходить для створення бекенда. Щоб була можливість зберігати в зручному вигляді Java-об'єкти в базі даних треба реалізувати JPA(Java Persistence API). Для реалізації використовувалось ORM(Object-Relational Mapping) Hibernate. Знизу представлена візуальна інтерпретація зв'язку компонентів в бекенді (рис. 2.3.)

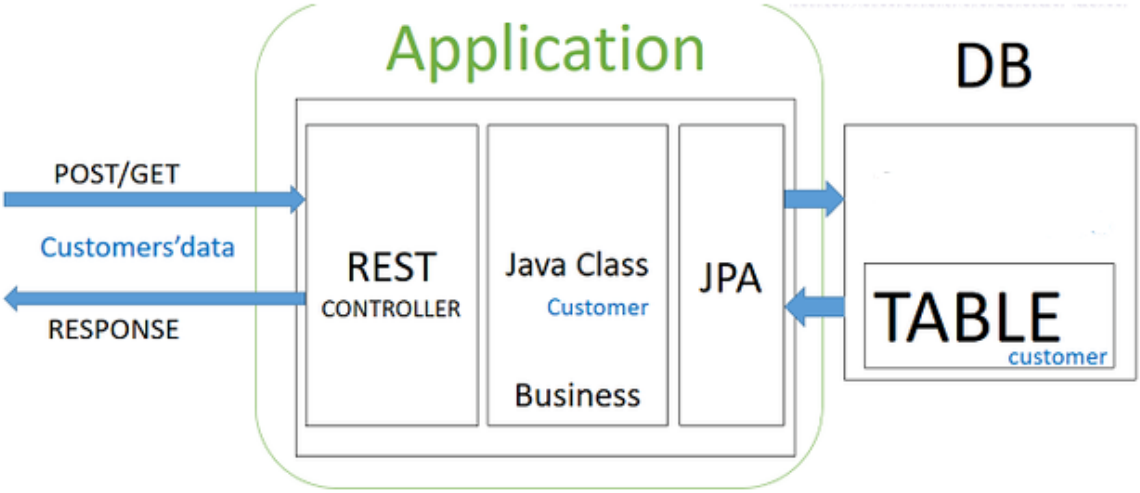

Рис. 2.3. Зв'язок компонентів в бекенді

2.3.1. Мова програмування Java

Java - це мова програмування та обчислювальна платформа, вперше випущена Sun Microsystems в 1995 році. Java - це мова програмування, створена для епохи Інтернету. Існує безліч програм та веб-сайтів, які не працюватимуть, якщо у вас не встановлена Java, і щодня створюються нові. Java швидка, безпечна та надійна. Від ноутбуків до центрів обробки даних, ігрових консолей до наукових суперкомп'ютерів, стільникових телефонів до Інтернету.

Якщо використовуєте мову програмування Java, то можете гарантовано очікувати наступне:

1. мова програмування Java є об'єктно-орієнтованою, проте вона все ще проста;
2. цикл розробки набагато швидший, оскільки технологія Java інтерпретується. Цикл compile-link-load-test-crash-debug застарілий - тепер ви просто компілюєте та запускаєте;
3. програми є портативними на декількох платформах. Напишіть свої програми один раз, і вам ніколи не потрібно буде їх переносити - вони працюватимуть без змін у декількох операційних системах та апаратних архітектурах;

4. програми надійні, оскільки середовище виконання Java керує пам’яттю для вас;

5. інтерактивні графічні програми мають високу продуктивність, оскільки у вашій програмі підтримуються багатонитковість, вбудованою в мову програмування Java та платформу виконання;

6. програми пристосовані до змінних середовищ, оскільки ви можете динамічно завантажувати модулі коду з будь-якої точки мережі;

7. кінцеві користувачі можуть довіряти безпеці ваших програм, навіть якщо вони завантажують код з усього Інтернету; середовище виконання Java має вбудований захист від вірусів та фальсифікацій.

8. Java - проста, загальноприйнята, об'єктно-орієнтована, інтерпретована, надійна, безпечна, нейтральна до архітектури, портативна, високопродуктивна, багатопоточна комп'ютерна мова. Він призначений для того, щоб дозволити розробникам додатків "писати один раз, запускати де завгодно" (WORA), що означає, що код, який працює на одній платформі, не потрібно перекомпілювати для роботи на іншій.

Java - це високорівнева, надійна, захищена та об'єктно-орієнтована мова програмування. А будь-яке апаратне або програмне середовище, в якому працює програма, відоме як платформа. Оскільки Java має власне середовище виконання (JRE) та API, це називається платформою.

У мові програмування Java весь вихідний код спочатку пишеться у звичайних текстових файлах, що закінчуються розширенням .java. Потім ці вихідні файли компілюються у файли .class компілятором javac. Файл .class не містить коду, який є рідним для вашого процесора; натомість він містить байт-коди - машинну мову віртуальної машини Java1 (Java VM). Потім засіб запуску Java запускає вашу програму з екземпляром віртуальної машини Java(рис. 1.3.1.).

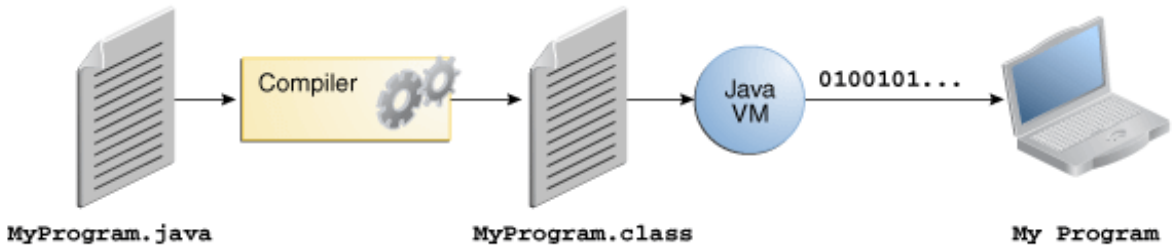

Рис. 1.3.1. Приклад роботи віртуальної машини

Платформа - це апаратне чи програмне середовище, в якому працює програма. Одними з найпопулярніших платформ є Microsoft Windows, Linux, Solaris OS та Mac OS. Більшість платформ можна описати як поєднання операційної системи та базового обладнання. Платформа Java відрізняється від більшості інших платформ тим, що це лише програмна платформа, яка працює поверх інших апаратних платформ.

Платформа Java складається з двох компонентів:

1. Віртуальна машина Java;
2. Інтерфейс програмування програм Java (API).

На даний момент існує безліч пристроїв, на яких використовується Java. Деякі з них такі:

1. Настільні програми;
2. Веб-додатки;
3. Мобільний;
4. Вбудована система;
5. Робототехніка;
6. Ігри тощо;

Типи програм Java:

1. Прикладні програми - це окремі програми, які написані для виконання певних завдань на локальному комп’ютері, таких як розв’язування рівнянь, читання та запис файлів тощо.

2. Аплети - це невеликі програми Java, розроблені для Інтернет-програм. Аплет, розташований на віддаленому комп'ютері, можна завантажити через Інтернет і виконати на локальному комп'ютері за допомогою браузера, що підтримує Java.

2.3.2. Патерни проєктування

У дипломному проекті були використані патерни проєктування. Для початку треба описати що це таке та для чого вони потрібні. Патерни проектування - це готові до використання рішення часто виникають в програмуванні задач. Патерни проектування, що підходить під завдання, реалізується в кожному конкретному випадку. Це не клас і не бібліотека, яку можна підключити до проекту, це щось більше. Правильно застосований патерн допоможе вирішити задачу легко і просто.

Типи патернів:

1. Породжують;
2. структурні;
3. поведінкові.

Породжувальні патерни надають механізми ініціалізації, дозволяючи створювати об'єкти зручним способом.

Структурні патерни визначають відносини між класами і об'єктами, дозволяючи їм працювати спільно.

Поведінкові патерни використовуються для того, щоб спростити взаємодію між сутностями.

У дипломному проекті використовувалися наступні патерни:

1. Породжувальні:

1.1. Singleton - обмежує створення одного примірника класу, забезпечує доступ до його єдиного об'єкту.

1.2. Abstract Factory - використовуємо супер фабрику для створення фабрики, потім використовуємо створену фабрику для створення об'єктів.

1.3. Builder - використовується для створення складного об'єкта з використанням простих об'єктів. Поступово він створює більший об'єкт від малого і простого об'єкта

1.4. Prototype - допомагає створити дубльований об'єкт з кращою продуктивністю, замість нового створюється повертається клон існуючого об'єкта.

2. Структурні:

2.1. Proxy - являє функціональність іншого класу.

2.2. Facade - забезепечує простий інтерфейс для клієнта, і клієнт використовує інтерфейс для взаємодії з системою.

3. поведінкові:

3.1. Observer - дозволяє одним об’єктам стежити і реагувати на події, що відбуваються в інших об'єктах.

2.3.3 Фреймворк Spring

Spring Boot забезпечує швидкий спосіб створити готовий до виробництва додаток на основі Spring. Схематичний приклад зв’язку Spring, Spring Boot та User на рис. 2.3.3.2. Він заснований на Spring Framework, надає перевагу конвенції над конфігурацією і призначений для того, щоб якнайшвидше почати працювати.

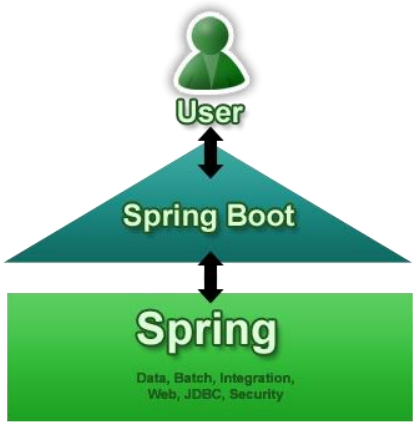

Рис. 3.3.3.1. Схематичне зображення зв’язку Spting Boot

Spring Framework - це платформа Java, яка забезпечує всебічну підтримку інфраструктури для розробки програм Java. Spring обробляє інфраструктуру, щоб ви могли зосередитись на своєму додатку.

Spring дозволяє створювати додатки з "простих старих об'єктів Java" (POJO) і застосовувати корпоративні служби неінвазивно до POJO. Ця можливість застосовується до моделі програмування Java SE та до повної та часткової Java EE.

Приклади того, як ви, як розробник додатків, можете отримати вигоду від платформи Spring:

1. Зробіть метод Java виконаним у транзакції бази даних, не маючи справу з API транзакцій.
2. Зробіть локальний метод Java кінцевою точкою HTTP, не маючи справу з API сервлету.
3. Зробіть локальний метод Java обробником повідомлень, не маючи справу з API JMS.
4. Зробіть локальний метод Java операцією управління без необхідності мати справу з API JMX.

Додаток Java - вільний термін, який запускає діапазон від обмежених, вбудованих програм до n-рівня серверних корпоративних додатків - як правило, складається з об'єктів, які спільно формують належну програму. Таким чином, об'єкти в додатку мають залежності один від одного.

Хоча платформа Java забезпечує безліч функціональних можливостей для розробки програм, їй бракує засобів для організації основних будівельних блоків у цілісне ціле, залишаючи це завдання архітекторам та розробникам. Хоча ви можете використовувати шаблони дизайну, такі як Factory, Abstract Factory, Builder, Decorator та Service Locator, щоб складати різні класи та екземпляри об'єктів, що складають додаток, ці шаблони є просто такими: найкращі практики, що отримують назву, з описом що робить шаблон, де його застосовувати, проблеми, які він вирішує, тощо. Шаблони - це формалізовані найкращі практики, які ви повинні впровадити самостійно у своєму додатку.

Компонент Spring Framework Inversion of Control (IoC) вирішує цю проблему, надаючи формалізовані засоби складання різнорідних компонентів у повністю робочий додаток, готовий до використання. Spring Framework кодифікує формалізовані шаблони дизайну як першокласні об’єкти, які ви можете інтегрувати у свої власні програми. Численні організації та установи використовують Spring Framework таким чином для створення надійних, ремонтопридатних програм.

Spring Framework складається з функцій, організованих приблизно в 20 модулів. Ці модулі згруповані в Основний контейнер, Доступ до даних / Інтеграція, Веб, AOP (Аспектно-орієнтоване програмування), Інструментарій, Обмін повідомленнями та Тест, як показано на наступній схемі(рис. 2.3.3.2.).

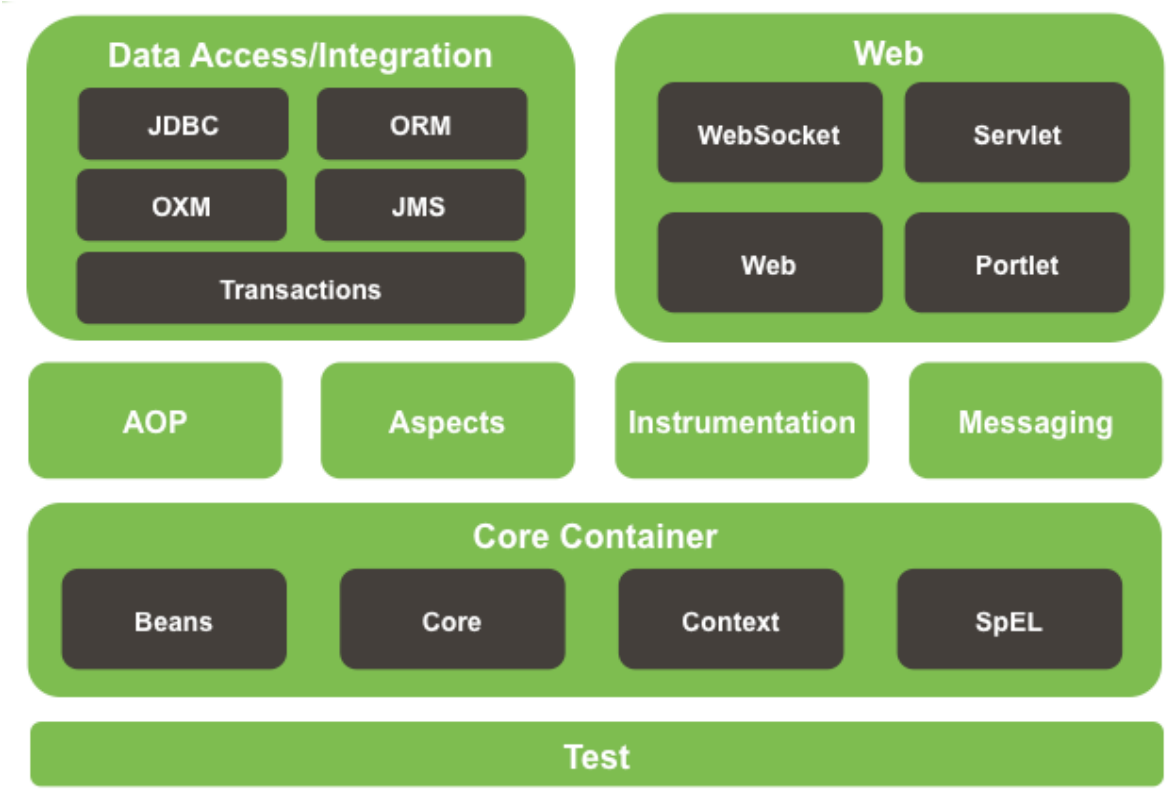

Рис. 2.3.3.2. Модулі Spring Framework

Основний контейнер складається з модулів spring-core, spring-beans, spring-context, spring-context-support та spring-expression (Spring Expression Language).

Модулі spring-core та spring-beans надають основні частини фреймворку, включаючи функції IoC та інжекції залежності. BeanFactory - це вишукана реалізація заводського зразка. Це позбавляє від необхідності програмних одиночних програм і

дозволяє відокремити конфігурацію та специфікацію залежностей від вашої фактичної логіки програми.

Модуль Context (spring-context) базується на надійній основі, що забезпечується модулями Core і Beans: це засіб доступу до об'єктів у стилі фреймворку, подібне до реєстру JNDI. Модуль Context успадковує свої функції від модуля Beans і додає підтримку інтернаціоналізації (використовуючи, наприклад, пакети ресурсів), розповсюдження подій, завантаження ресурсів та прозоре створення контекстів, наприклад, контейнером Servlet. Модуль Context також підтримує такі функції Java EE, як EJB, JMX та базове віддалення. Інтерфейс ApplicationContext є фокусною точкою модуля Context. spring-context-support забезпечує підтримку інтеграції спільних сторонніх бібліотек у контекст програми Spring, зокрема для кешування (EhCache, JCache) та планування (CommonJ, Quartz).

Модуль spring-expression забезпечує потужну мову виразів для запитів та маніпулювання графіком об’єктів під час виконання. Це розширення уніфікованої мови виразів (уніфікованої EL), як зазначено в специфікації JSP 2.1. Мова підтримує встановлення та отримання значень властивостей, присвоєння властивостей, виклик методу, доступ до вмісту масивів, колекцій та індексаторів, логічних та арифметичних операторів, іменованих змінних та отримання об’єктів за іменами із IoC-контейнера Spring. Він також підтримує проектування та виділення списків, а також загальні агрегації списків.

Модуль spring-aop забезпечує реалізацію аспектно-орієнтованого програмування, сумісну з AOP Alliance, що дозволяє вам визначити, наприклад, перехоплювачі методів і точки для чистого роз'єднання коду, що реалізує функціональні можливості, які слід розділяти. Використовуючи функціональність метаданих на рівні джерела, ви також можете включити інформацію про поведінку у свій код, подібним чином до атрибутів .NET.

Окремий модуль spring-аспекти забезпечує інтеграцію з AspectJ.

Модуль spring-instrument забезпечує підтримку інструментарію класу та реалізації навантажувача класів, які будуть використовуватися на певних серверах

додатків. Модуль spring-instrument-tomcat містить інструментальний агент Spring для Tomcat.

Spring Framework 4 включає модуль Spring-Messaging з ключовими абстракціями проекту Spring Integration, такими як Message, MessageChannel, MessageHandler та інші, які служать основою для програм на основі обміну повідомленнями. Модуль також включає набір анотацій для зіставлення повідомлень із методами, подібними до моделі програмування на основі анотацій Spring MVC.

Управління залежністю та введення залежності - це різні речі. Щоб отримати ці приємні особливості Spring у вашому додатку (наприклад, введення залежностей), вам потрібно зібрати всі необхідні бібліотеки (jar-файли) і перенести їх у свою шлях до класу під час виконання, і, можливо, під час компіляції. Ці залежності не вводяться віртуальними компонентами, а фізичними ресурсами у файловій системі (як правило). Процес управління залежностями передбачає пошук цих ресурсів, їх зберігання та додавання до шляхів занять. Залежності можуть бути прямими (наприклад, моя заявка залежить від Spring під час виконання) або непрямою (наприклад, моя заявка залежить від commons-dbcp, яка залежить від commons-pool). Непрямі залежності також відомі як "транзитивні", і саме ті залежності найважче визначити та керувати ними.

Якщо ви збираєтеся використовувати Spring, вам потрібно отримати копію бібліотек jar, які складаються з частин Spring, які вам потрібні. Щоб полегшити це, Spring упакований у набір модулів, які максимально відокремлюють залежності, тому, наприклад, якщо ви не хочете писати веб-програму, вам не потрібні модулі spring-web. Для посилання на модулі бібліотеки Spring у цьому посібнику ми використовуємо скорочену назву імені spring- * або spring - *. Jar, де * представляє коротку назву модуля (наприклад, spring-core, spring-webmvc, spring-jms тощо).). Фактичне ім'я файлу jar, яке ви використовуєте, як правило, ім'я модуля, об'єднане з номером версії (наприклад, spring-core-4.3.11.RELEASE.jar).

Кожен випуск Spring Framework буде публікувати артефакти в таких місцях:

Maven Central, що є сховищем за замовчуванням, яке Maven запитує, і не вимагає спеціальної конфігурації для використання. Багато загальних бібліотек, від яких залежить Spring, також доступні в Maven Central, і значна частина спільноти Spring використовує Maven для управління залежностями, тому це зручно для них. Назви банок тут мають форму spring - * - <version> .jar, а група MavenId - org.springframework.

У загальнодоступному сховищі Maven, розміщеному спеціально для Spring. На додаток до остаточних випусків GA, у цьому сховищі також розміщені моментальні знімки та етапи розробки. Назви jar-файлів мають таку ж форму, що і Maven Central, тому це корисне місце, щоб отримати версії для розробки Spring, які можна використовувати з іншими бібліотеками, розгорнутими в Maven Central. Цей репозиторій також містить zip-файл розподілу пакетів, який містить усі банки Spring, зв'язані разом для зручного завантаження.

Отже, перше, що вам потрібно вирішити, це як керувати своїми залежностями: ми, як правило, рекомендуємо використовувати таку автоматизовану систему, як Maven, Gradle або Ivy, але ви також можете зробити це вручну, завантаживши всі банки самостійно.

Ведення журналу є дуже важливою залежністю для Spring, оскільки:

a) це єдина обов'язкова зовнішня залежність;

b) кожному подобається бачити певний результат від інструментів;

c) Spring інтегрується з безліччю інших інструментів.

Однією з цілей розробника додатків часто є встановлення уніфікованого журналювання в центральному місці для всієї програми, включаючи всі зовнішні компоненти. Це складніше, ніж це могло б бути, оскільки існує так багато варіантів систем реєстрації.

Приємна річ загальних журналів полягає в тому, що вам не потрібно нічого іншого, щоб ваша програма працювала. Він має алгоритм виявлення середовища виконання, який шукає інші фреймворки журналювання у відомих місцях на шляху до класу та використовує той, який вважає за потрібний (або ви можете вказати, який

саме, якщо вам потрібно). Якщо нічого іншого не доступно, ви отримуєте досить приємні журнали просто з JDK (java.util.logging або JUL коротше). Ви повинні виявити, що ваш додаток Spring працює та входить у консоль із більшістю випадків, і це важливо.

Spring програми можуть працювати на контейнері, який сам забезпечує реалізацію JCL, наприклад Сервер додатків IBM WebSphere (WAS). Це не викликає проблем як таких, але призводить до двох різних сценаріїв, які потрібно зрозуміти:

У "батьківській першій" моделі делегування ClassLoader (за замовчуванням для WAS) програми завжди вибирають серверну версію Commons Logging, делегуючи підсистемі ведення журналу WAS (яка фактично базується на JUL). Варіант JCL, що надається додатком, будь то стандартний протокол Commons Logging або мост JCL-over-SLF4J, буде фактично ігноруватися разом із будь-яким місцевим постачальником послуг журналів.

З моделлю делегування "батьківського останнього" (за замовчуванням у звичайному контейнері сервлетів, але явна опція конфігурації на WAS), буде обрано запропонований програмою варіант спільного журналу, що дозволить вам налаштувати постачальника журналів, включеного локально, наприклад Log4j або Logback у вашій програмі. У разі відсутності локального постачальника журналів, звичайне ведення журналу Commons за замовчуванням делегуватиме JUL, ефективно входячи до підсистеми ведення журналу WebSphere, як у сценарії "спочатку батьківський".

Загалом, ми рекомендуємо розгортати програми Spring у "батьківській останній" моделі, оскільки це, природно, дозволяє місцевим провайдерам, а також підсистемі журналу сервера.

2.3.4. Опис Spring Security

Spring Security - це структура, яка зосереджена на забезпеченні механізмів автентифікації та авторизації додатків Spring. Він був розпочатий у 2003 році як

проект з відкритим кодом під назвою "Acegi Security", перш ніж був офіційно включений у Spring проекти. На додаток до автентифікації та авторизації, Spring Security можна налаштувати для захисту вашого додатка від багатьох поширених атак, включаючи, але не обмежуючись ними, CSRF, XSS, Brute Forcing та MITM (Man in the Middle), якщо ви хочете дізнатись більше про них атаки, ви можете поглянути на цю статтю Холлі Ллойд. На рис. 2.3.4. зображено візуальне представлення роботи Spring Security.

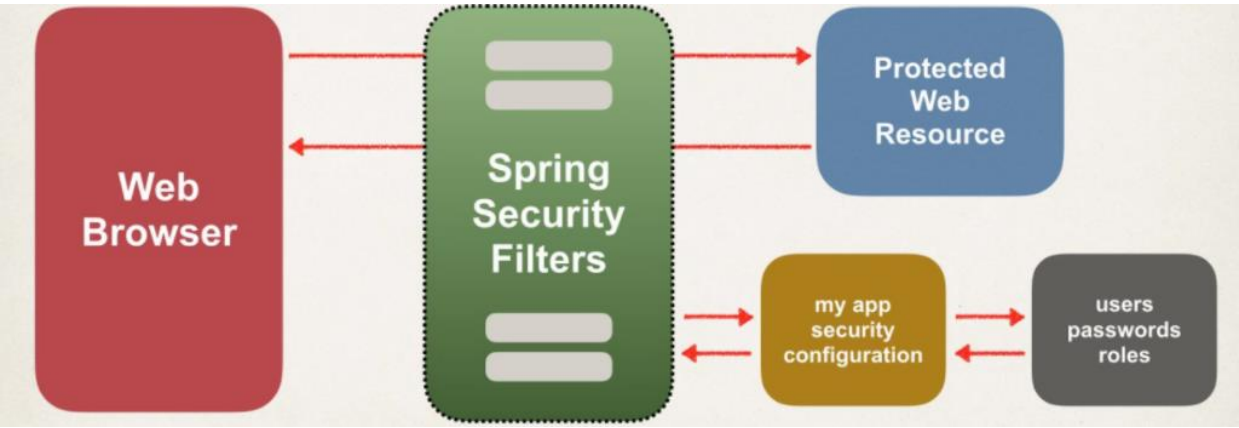

рис. 2.3.4.1 візуальне представлення роботи Spring Security

Одне з найкращих дій, яке ви можете зробити, якщо у вас немає досвіду захисту програми, - це з’ясувати, чи має мова / платформа, яку ви використовуєте, систему захисту. Використовуючи структуру безпеки, на яку можна покластися, ми передаємо відповідальність за визначення архітектури та реалізацію основних функцій безпеки групі експертів у цій галузі, які працюють над цією структурою.

Переваги Spring Security не обмежуються лише тим, що ми допомагаємо нам з автентифікацією та авторизацією. Це також може допомогти нам застосовувати найкращі практики з порятунку користувачів, зокрема, створення функції реєстрації.

У процесі створення функції реєстрації є кілька типових помилок, які можуть призвести до серйозних проблем із безпекою.

Однією з таких помилок є збереження паролів користувачів у відкритому тексті без будь-якого хешування. Це не тільки дозволяє хакерам, які можуть отримати доступ до вашого сервера, безпосередньо переглядати паролі, але також відкриває для

користувачів загрозу злому інших облікових записів, оскільки більшість людей використовують однакові паролі для всіх своїх облікових записів.

Ще однією поширеною помилкою є спроба винайти колесо та створити власний алгоритм шифрування / хешування замість того, щоб використовувати добре відомі, перевірені, перевірені алгоритми.

Spring Security дозволяє нам призначити захищений кодер пароля нашому об’єкту UserDetails, щоб запобігти цим помилкам. За замовчуванням він використовує BCrypt для шифрування паролів, що вважається досконалим алгоритмом кодування паролів. Також можна встановити кількість раундів хешування (або потужність, як підказує назва параметра) та безпечну реалізацію випадкового алгоритму, які будуть використовуватися в процесі.

Аутентифікація в пам’яті означає використання бази даних, яка залишається в пам’яті додатків / оперативній пам’яті (одним із прикладів є база даних h2), щоб зберегти користувачів та виконати автентифікацію, не зберігаючи їх у постійній базі даних. Такі бази даних можуть бути корисними, коли ви створюєте додаток, що підтверджує концепцію, або створює прототип свого додатка, і вам потрібно почати роботу або протестувати код автентифікації, не турбуючи себе базою даних. База даних в пам'яті також корисна, якщо ви запускаєте тести без макетів, щоб ваша реальна база даних залишалася недоторканою, а зміни, викликані тестами, перебували у тимчасовій пам'яті.

У додатках, де бекенд і інтерфейс тісно пов'язані, наприклад, програми Spring MVC або будь-яка інша структура MVC: Щоразу, коли користувач входить в систему, сеанс зберігається на сервері, що містить інформацію про цього користувача. Такі операції, як отримання, кешування, знищення цього сеансу, називаються управління сесіями. Spring Security забезпечує механізми управління цим об’єктом сеансу. Він створить сеанс, коли користувач увійде в систему, знищить його, коли вийде з системи, і дозволить нам встановити значення часу очікування.

Spring Security також вживає додаткових заходів для забезпечення безпечного використання сеансів:

- Це можна налаштувати для вимкнення перезапису URL-адрес, щоб уникнути атаки відстеження сеансів.
- Він автоматично мігрує сеанс, коли користувач входить знову, щоб уникнути фіксації сеансу.
- Це дозволяє нам використовувати httpOnly та захищати прапори на файлах cookie сеансу для захисту наших файлів cookie.

Оскільки REST є безсесійним протоколом, радимо не зберігати сеанс у RESTful API. Натомість можна використати інший популярний підхід, щоб перевірити, чи уповноважений користувач вжити певних дій.

У програмах, що використовують JWT, після успішного входу користувача в систему маркер доступу генерується із секретним ключем на сервері, який зазвичай містить інформацію про ідентичність користувача та позначку часу генерування маркера. У окремих архітектурах інтерфейсу-сервера, інтерфейс зберігає цей маркер у файлах cookie.

Можна застосувати авторизацію JWT для вашого додатку, використовуючи бібліотеку Auth0 JWT для кодування / декодування токенів. На додаток до цього, Spring Security може використовуватися для фільтрації запитів та перевірки ролей користувача, дозволяючи лише авторизованим користувачам проходити через фільтри.

Ланцюг відповідальності - це шаблон поведінкового дизайну GoF, який спрямований на зменшення тісного зв’язку між обробниками запитів та відправником. На рис 2.3.4.2. зображено візуальне представлення ланцюга відповідальності. "Ланцюжок" відноситься до списку впорядкованих обробників запитів, як і структура LinkedList, а обробник запитів може бути методом або класом, що реалізує якийсь інтерфейс обробника. Відповідальність - це просто приємне слово відповіді на запит, кожен обробник ланцюга може або повернути відповідь (взявши на себе відповідальність), або передати запит наступному обробнику на ланцюжку (делегуючи відповідальність).

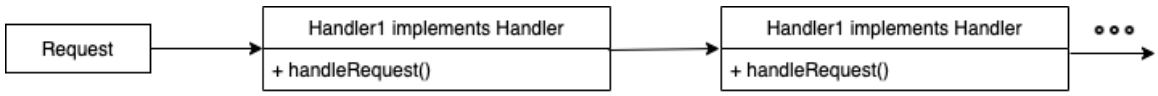

рис. 2.3.4.2 ланцюг відповідальності

2.3.5 Опис ORM Hibernate

Насамперед ми повинні описати JDBC та навіщо він потрібен. JDBC розшифровується як Java Database Connectivity. Він надає набір Java API для доступу до реляційних баз даних із програми Java. Ці Java API дозволяють програмам Java виконувати оператори SQL та взаємодіяти з будь-якою базою даних, сумісною з SQL.

JDBC забезпечує гнучку архітектуру для написання програми, незалежної від бази даних, яка може працювати на різних платформах та взаємодіяти з різними СУБД без будь-яких змін.

Коли працюємо з об’єктно-орієнтованою системою, виникає невідповідність об’єктної моделі та реляційної бази даних. СУБД представляють дані у табличному форматі, тоді як об'єктно-орієнтовані мови, такі як Java або C #, представляють їх як взаємопов'язаний графік об'єктів.

Перша проблема, а що, якщо нам потрібно змінити дизайн нашої бази даних після розробки декількох сторінок або нашого додатку? По-друге, завантаження та збереження об’єктів у реляційній базі даних піддає нас наступним п’яти невідповідностям:

1. Гранулярність. Іноді у вас буде об'єктна модель, яка має більше класів, ніж кількість відповідних таблиць у базі даних.

2. Спадщина. СУБД не визначають нічого подібного до "Спадкування", що є природною парадигмою в об'єктно-орієнтованих мовах програмування.

3. Особистість. СУБД визначає точно одне поняття "однаковості": первинний ключ. Однак Java визначає ідентичність об'єкта (a == b) і рівність об'єкта (a.equals (b)).

4. Асоціації. Об'єктно-орієнтовані мови представляють асоціації з використанням посилань на об'єкти, тоді як СУБД представляє асоціацію як стовпець зовнішнього ключа.

5. Навігація. Шляхи доступу до об’єктів у Java та RDBMS принципово відрізняються.

Об'єктно-реляційне відображення (ORM) - це рішення для обробки всіх вищезазначених невідповідностей імпедансу.

Hibernate - це об’єктно-реляційне відображення (ORM) для JAVA. Це стійкий фреймворк з відкритим кодом, створений Гевіном Кінгом у 2001 році. Це потужний високоефективний об’єктно-реляційний сервіс стійкості та запитів для будь-яких додатків Java.

Hibernate пов'язує класи Java із таблицями баз даних та від типів даних Java до типів даних SQL і звільняє розробника від 95% загальних завдань програмування, пов'язаних із збереженням даних.

Hibernate знаходиться між традиційними об’єктами Java та сервером баз даних, щоб виконувати всі роботи щодо збереження цих об’єктів на основі відповідних механізмів та шаблонів O / R.(рис. 2.3.5.)

Рис. 2.3.5. Схема роботи Hibernate

Переваги Hibernate:

1. Hibernate дбає про відображення класів Java до таблиць баз даних за допомогою XML-файлів і без написання будь-якого рядка коду.

2. Забезпечує прості API для зберігання та отримання об’єктів Java безпосередньо до бази даних та з неї.

3. Якщо в базі даних або будь-якій таблиці є зміни, вам потрібно змінити лише властивості файлу XML.

4. Абстрагує незнайомі типи SQL і забезпечує спосіб обійти знайомі об'єкти Java.

5. Hibernate не вимагає роботи сервера додатків.

6. Маніпулює складними асоціаціями об’єктів вашої бази даних.

7. Мінімізує доступ до бази даних за допомогою розумних стратегій отримання.

8. Забезпечує простий запит даних.

Hibernate підтримує майже всі основні СУБД. Нижче наведено перелік декількох механізмів баз даних, що підтримуються Hibernate :

1. Механізм баз даних HSQL;
2. DB2 / NT;
3. MySQL;
4. PostgreSQL;
5. FrontBase;
6. Oracle;
7. База даних Microsoft SQL Server;
8. Sybase SQL Server;
9. Динамічний сервер Informix.

Підтримувані технології:

1. XDoclet Spring;
2. J2EE;
3. Eclipse plug-ins;
4. Maven.

2.4. Опис фронтенда та фреймворк VueJS

Для початку опишемо навіщо потрібен VueJS. Фреймворки розробляються для того, щоб спростити нам життя і звільнити від написання однотипного коду. Але, у міру того як кодова база деяких фреймворків сильно розростається, вони починають привносити свою частку якої складності в дипломний проект. Через це при плануванні розробки ми повинні враховувати два фактори:

1. складність нашого застосування
2. складність фреймворка, який ми використовуємо

Один за найкращих фреймворків, який задовольняє вище вказані фактори це VueJS. З ним дуже просто почати працювати, навіть якщо ви ніколи не працювали з JavaScript фреймворками. Це ідеальне поєднання зручності і потужності.

VueJS - це прогресивна платформа JavaScript з відкритим кодом, що використовується для розробки інтерактивних веб-інтерфейсів. Це один із відомих фреймворків, що використовується для спрощення веб-розробки. VueJS фокусується на рівні подання. Його можна легко інтегрувати у великі проекти для інтерфейсної розробки без будь-яких проблем.

В основі Vue.js лежить реактивна система прив'язки даних, яка дозволяє надзвичайно просто синхронізувати ваші дані та DOM. Використовуючи jQuery для ручного маніпулювання DOM, код, який ми пишемо, часто є імперативним, повторюваним та схильним до помилок. Vue.js охоплює концепцію подання, керованого даними. Простими словами, це означає, що ми використовуємо спеціальний синтаксис у наших звичайних шаблонах HTML, щоб “прив’язати” DOM до базових даних. Після створення прив’язок DOM буде синхронізовано з даними. Щоразу, коли ви змінюєте дані, DOM оновлюється відповідно. Як результат, більша частина нашої логіки додатків зараз безпосередньо обробляє дані, а не возиться з оновленнями DOM. Це робить наш код простішим для написання, легшим для міркувань та простішим у обслуговуванні.

Нижче наведені функції, доступні у VueJS:

1. Віртуальний DOM. VueJS використовує віртуальний DOM, який також використовується іншими фреймворками, такими як React, Ember тощо. Зміни не вносяться в DOM, натомість створюється копія DOM, яка присутня у вигляді структур даних JavaScript . Щоразу, коли потрібно внести будь-які зміни, вони вносяться до структур даних JavaScript, і останні порівнюються з початковою структурою даних. Потім остаточні зміни оновлюються до реального DOM, який користувач побачить змінюватися. Це добре з точки зору оптимізації, це дешевше, і зміни можна вносити швидше.

2. Прив'язка даних. Функція прив'язки даних допомагає маніпулювати або призначати значення атрибутам HTML, змінювати стиль, призначати класи за допомогою директиви прив'язки, яка називається v-bind, доступна у VueJS.

3. Компоненти. Компоненти - одна з важливих особливостей VueJS, яка допомагає створювати власні елементи, які можна використовувати повторно в HTML.

4. Обробка подій. v-on - це атрибут, доданий до елементів DOM для прослуховування подій у VueJS.

5. Анімація / Перехід. VueJS пропонує різні способи застосувати перехід до елементів HTML, коли вони додаються / оновлюються або видаляються з DOM. VueJS має вбудований компонент переходу, який потрібно обернути навколо елемента для ефекту переходу. Ми можемо легко додати сторонні бібліотеки анімації, а також додати більше інтерактивності інтерфейсу.

6. Обчислювані властивості. Це одна з важливих особливостей VueJS. Це допомагає прослуховувати зміни, внесені до елементів інтерфейсу, та виконує необхідні обчислення. Для цього не потрібно додаткове кодування.

7. Шаблони. VueJS надає шаблони на основі HTML, які пов'язують DOM з даними екземпляра Vue. Vue компілює шаблони у функції віртуального DOM Render. Ми можемо використовувати шаблон функцій візуалізації, і для цього нам доведеться замінити шаблон функцією візуалізації.

8. Директиви. VueJS має вбудовані директиви, такі як v-if, v-else, v-show, v-on, v-bind та v-model, які використовуються для виконання різних дій на інтерфейсі.

9. Сторожки. Спостерігачі застосовуються до даних, які змінюються. Наприклад, елементи введення форми. Тут нам не потрібно додавати додаткові події. Watcher піклується про будь-які зміни даних, роблячи код простим і швидким.

10. Маршрутизація.Навігація між сторінками здійснюється за допомогою vue-router.

11. Сценарій VueJS дуже легкий, і продуктивність також дуже швидка.

12. Vue-CLI. VueJS можна встановити в командному рядку за допомогою інтерфейсу командного рядка vue-cli. Це допомагає легко створювати та компілювати проект за допомогою vue-cli.

2.5. Середовища розробки інформаційно-управляючої системи

Середовище розробки - це набір процедур і інструментів для розробки, тестування і налагодження програми або програми.

Зазвичай середовище розробки має три рівні серверів: розробка, виробництво і виробництво. Всі три рівня разом зазвичай називаються DSP.

Сервер розробки: тут розробник тестує код і перевіряє, чи успішно програма працює з цим кодом. Після того, як додаток протестовано і розробник відчує, що код працює правильно, програма переходить на проміжний сервер.

Проміжний сервер: це середовище спроектована так, щоб виглядати в точності як операційне середовище сервера. Програма тестується на проміжному сервері, щоб перевірити надійність і переконатися, що вона не дає збоїв на реальному виробничому сервері. Цей тип тестування на проміжному сервері є останнім кроком перед розгортанням програми на робочому сервері. Додаток має бути схвалено для розгортання на робочому сервері.

Виробничий сервер. Після затвердження додаток стає частиною цього сервера.

2.5.1. Intellij idea

Для розробки бекенду було обране програмне забезпечення JetBrains IntelliJ IDEA, яке є провідним середовищем швидкого розвитку мови Java. IntelliJ IDEA - це високотехнологічний комплекс тісно інтегрованих інструментів програмування, що включає інтелектуальний редактор джерел із вдосконаленими засобами автоматизації, потужні засоби рефакторингу коду, вбудовану підтримку технологій J2EE, механізми інтеграції із середовищем тестування Ant / JUnit та системами контролю версій., унікальний інструмент оптимізації та перегляду коду Code Inspection, а також інноваційний візуальний конструктор графічного інтерфейсу(рис. 2.5.1.).

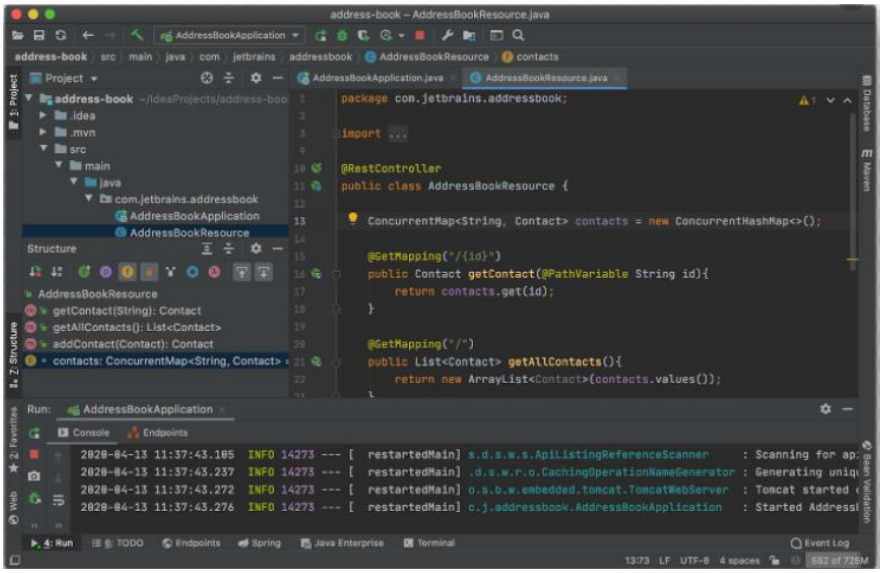

Рис. 2.5.1. Графічний інтерфейс

Унікальні можливості JetBrains IntelliJ IDEA знімають тягар рутинної роботи з програміста, допомагають своєчасно виправляти помилки та покращувати якість коду, підвищуючи продуктивність розробників на нові висоти.

Серед безліч переваг IntelliJ IDEA особливо виділяється наступні:

1. Multi-platform. IntelliJ IDEA - це міжплатформна IDE, яка забезпечує стабільний досвід роботи в Windows, macOS та Linux.

2. Supported languages. Розробка сучасних додатків передбачає використання багатьох мов, інструментів, фреймворків та технологій. IntelliJ IDEA розроблений як IDE для мов JVM, але численні плагіни можуть розширити його, щоб забезпечити поліглот.

3. JVM languages. Використовуйте IntelliJ IDEA для розробки програм наступними мовами, які можна скомпілювати в байт-код JVM, а саме: Java, Kotllin, Scala, Groovy.

4. User Interface. IntelliJ IDEA забезпечує середовище, орієнтоване на редактора. Він слідкує за вашим контекстом і автоматично пропонує необхідні інструменти, які допоможуть мінімізувати ризик переривання потоку розробника.

5. Coding assistance. IntelliJ IDEA допомагає прискорити процес кодування, забезпечуючи завершення контекстного коду.

6. Refactorings. IntelliJ IDEA пропонує повний набір автоматизованих рефакторингів коду, що призводить до значного підвищення продуктивності. Наприклад, коли ви перейменовуєте клас, IDE оновить усі посилання на цей клас, який є в проекті.

7. Static code analysis. IntelliJ IDEA забезпечує набір перевірок, які є вбудованими засобами статичного аналізу коду. Вони допомагають знаходити потенційні помилки, знаходити мертвий код, виявляти проблеми з продуктивністю та покращувати загальну структуру коду.

8. Debugger. IntelliJ IDEA забезпечує вбудований налагоджувач JVM. Це дозволяє отримувати та аналізувати інформацію про час роботи, що корисно для діагностики проблем та глибшого розуміння того, як працює програма.

2.5.2 Webstorm

WebStorm - це інтегроване середовище розробки для кодування в JavaScript та пов'язаних з ним технологіях, включаючи TypeScript, React, Vue, Angular, Node.js, HTML та таблиці стилів. Подібно до IntelliJ IDEA та інших середовищ розробки

середовищ JetBrains, WebStorm робить ваш досвід розробки приємнішим, автоматизуючи рутинні роботи та допомагаючи вам легко виконувати складні завдання. WebStorm має приємний та зрозумілий інтерфейс(рис. 2.5.2.)

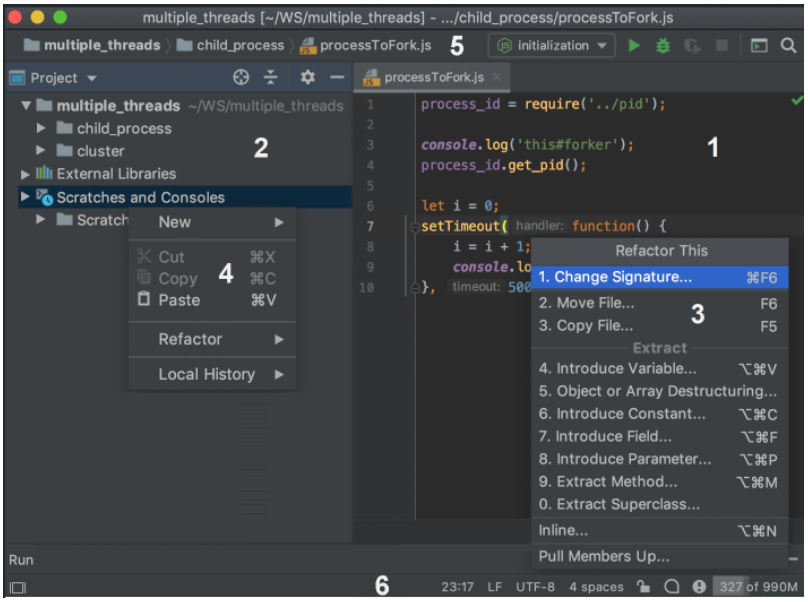

Рис. 2.5.2. Приклад інтерфейсу

Дипломний проект у веб-папці WebStorm з вихідним кодом редагують, бібліотеки та інструменти, які використовуються (наприклад, у вкладених модулях), а також різні додатки конфігурації (наприклад, package.json або .eslintrc).

З головних особливостей середовища відмічаються:

1. Modern frameworks. WebStorm надає розширену допомогу в кодуванні Angular, React, Vue.js та Meteor. Насолоджуйтесь підтримкою React Native, PhoneGap, Cordova та Ionic для мобільної розробки та розробки на стороні сервера за допомогою Node.js. Все в одному IDE!

2. Smart editor. IDE аналізує проект, щоб забезпечити найкращі результати завершення коду для всіх підтримуваних мов. Сотні вбудованих перевірок повідомляють про можливі проблеми прямо під час набору тексту та пропонують варіанти швидкого виправлення.

3. Navigation & Search. WebStorm допомагає ефективніше обійти свій код та заощадити час при роботі з великими проектами. Перейдіть до методу, функції або визначення змінної лише одним клацанням миші або знайдіть звички.

4. Testing. Виконуйте тестування з легкістю, оскільки WebStorm інтегрується з бігунком тесту Karma, Mocha, Jest та Protractor. Запустіть та налагодьте тести безпосередньо в IDE, перегляньте результати у приємному та наочному форматі та перейдіть до тестового коду.

5. Tracing and profiling. WebStorm має вбудований інструмент spy-js, який допомагає простежити код JavaScript. Дослідить, як файли пов’язані із викликами функцій, та ефективно виявіть можливі вузькі місця.

6. Project templates. Починайте нові проекти з екрану привітання за допомогою популярних шаблонів проектів, таких як Express або Web starter kit, і отримуйте доступ до ще більшої кількості генераторів проектів завдяки інтеграції з Yeoman.

7. VCS. WebStorm надає уніфікований інтерфейс для роботи з багатьма популярними системами керування версіями, забезпечуючи незмінну взаємодію користувачів із git, GitHub, SVN, Mercurial та Perforce.

8. Local history. Незалежно від того, використовуєте ви VCS чи ні, «Місцева історія» може справді заощадити код. Ви можете будь-коли перевірити історію певного файлу або каталогу та повернутися до будь-якої з попередніх версій.

9. Customization. WebStorm надзвичайно настроюється. Налаштуйте його відповідно до вашого стилю кодування - від ярликів, шрифтів та візуальних тем до вікон інструментів та макета редактора.

2.5.3. Datagrip

DataGrip - це середовище управління базами даних для розробників. Він призначений для запитів, створення та управління базами даних. Бази даних можуть працювати локально, на сервері або в хмарі. Підтримує MySQL, PostgreSQL, Microsoft

SQL Server, Oracle та багато іншого. Якщо у вас є драйвер JDBC, додайте його до DataGrip, підключіться до СУБД і починайте працювати.

DataGrip підтримує багато різних СУБД. Якщо у вас є база драйвера JDBC, DataGrip може з нею працювати. Також має один із найкращих інтерфейсів (Рис. 2.5.3.)

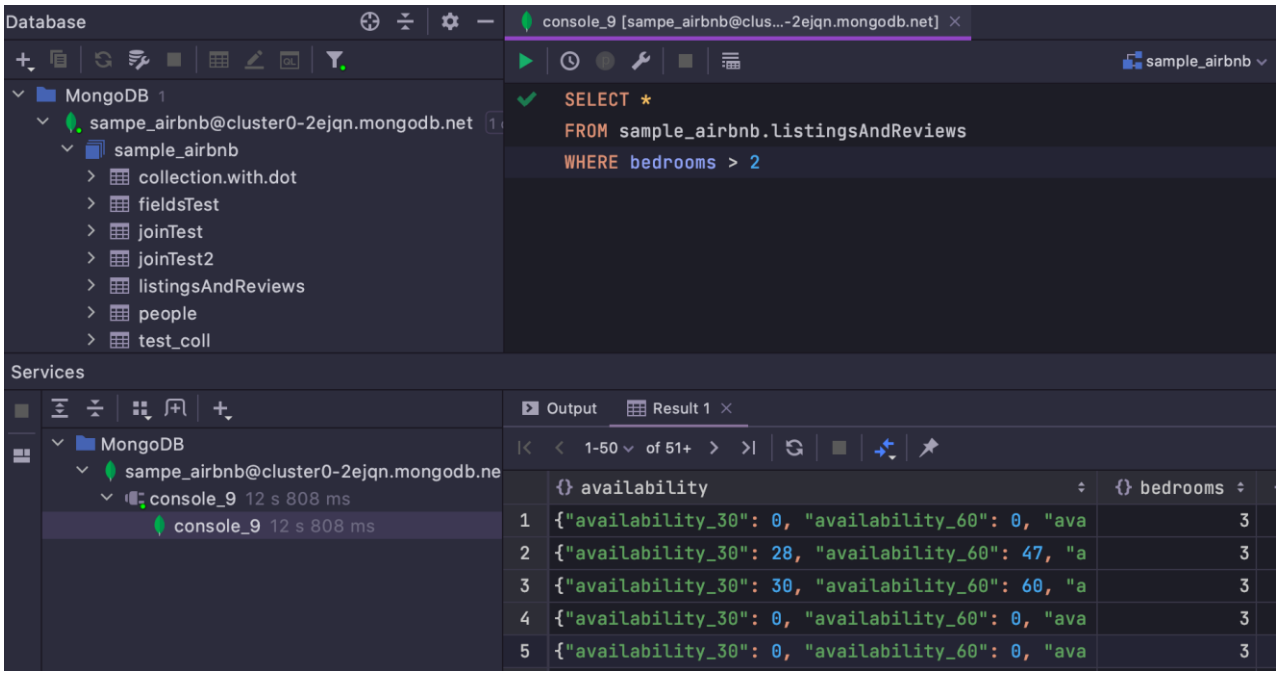

Рис. 2.5.3. Приклад інтерфейсів

основних можливості DataGrip:

1. Database objects. DataGrip відкине інформацію про всі важливі речі про вашу базу, які відображаються біля дерева. Є Інтерфейс для створення таблиць, таблиць, покажчиків, взаємозв’язку та інших об’єктів.

2. Data editor. Додавайте, видаляйте, міняйте дані в табличному редакторі. Переходьте в пов'язані таблиці із зовнішніх ключів, використовуйте текстовий пошук, щоб знайти дані в будь-якому стовпці.

3. Navigation. Переходьте до вихідного об'єкту коду з використання цього об'єкта в запиті. Знаходьте будь-який об'єкт на ім'я або навіть по частині імені.

4. Writing queries. Як і належить середовище розробки, в DataGrip є все необхідне, щоб писати SQL: підсвічування, автодоповнення, раннє визначення помилок і підказки. Ви будете витрачати менше часу на створення запитів.\

5. Smart text editor. DataGrip використовує текстовий редактор платформи IntelliJ, який допомагає вам працювати з текстом більш ефективно. Переміщайте блоки тексту, виділяйте код на основі синтаксису, використовуйте мультикурсор, форматуйте код.

6. Autocompletion. Автодоповнення розуміє контекст: пропонує ключові слова або об'єкти в залежності від того, що підходить в конкретному місці коду. Воно враховує зовнішні ключі, структуру об'єктів і навіть об'єкти, створені в цьому ж скрипти.

7. Visual file comparison. Порівнюйте результати запитів або дані таблиць. DataGrip підсвічує різницю між наборами даних.

8. Importing CSV files. У DataGrip є інтерфейс для імпорту CSV-файлів з будь-яким роздільником. Імпортувати можна в нову таблицю або в існуючу. Якщо число колонок не збігається, вкажіть відповідності явно.

9. Export to SQL. Набір даних може бути представлений, як набір UPDATE або INSERT запитів, які потім можна виконати в поточному або іншій основі.

10. Diagrams. Отримаєте повне уявлення про свою базі за допомогою діаграми зв'язків.

Висновки до розділу 2

В арсеналі як фронтенда, так і бекенд є безліч мов і технологій. Про їх актуальність і зручність ведуться запеклі суперечки. Але істина, як завжди, десь посередині. На вибір технології повинна впливати не мода, а те, які завдання розробник перед собою ставить. Кожна мова і фреймворк доречний тоді, коли він відповідає специфіці вирішуваної проблеми. У даному випадку найкраще буде використовувати технології які були зазначені вище та описані чому саме вони найкраще підходять.

Нижче описані технологічні рішення основних задач дипломного проекту:

- реалізувати систему захисту даних за допомогою із застосуванням Spring Security
- розробити структуру бази даних для інформаційної системи ведення процесу ремонту автомобільних двигунів за допомогою СУБД MariaDb;
- розробити бекенд на мові Java та фреймфорка Spring;
- реалізувати методи для взаємодії з базою даних за допомогою фреймворку Hibernate;
- розробити фронтенд із застосуванням феймворка VueJs.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ

3.1. Розробка бекенда на мові Java та фремворка Spring

Бекенд інформаційної системи ведення ремонтного процесу автомобільних двигунів має файлову структуру, яка зображена на рис. 3.1.1., складається з директорії і піддиректорії, кожна з яких відповідає за окремий функціональний компонент системи відповідно до архітектури MVC:

1. компонент Model – директорія Service;
2. компонент Controller – директорія Controller;
3. компонент View – директорія Payload.Response.

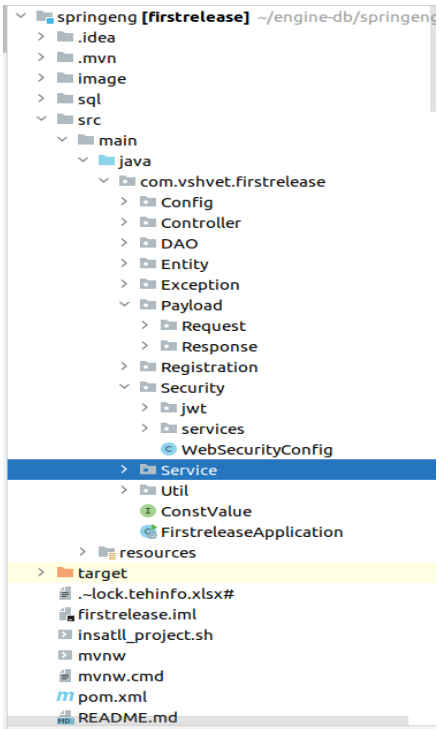

Рис. 3.1.1. Файлова структура інформаційної системи

Директорія DAO складається з файлів, які використовується для збереження об'єктів бізнес-області в базі даних. У цій директорії містяться класи мають CRUD методи для конкретної сутності. Запити створюються на мові HQL (об'єктів-орієнтована мова запитів).У цій директорії містяться класи мають CRUD методи для конкретної сутності. Запити створюються на мові HQL (об'єктів-орієнтована мова запитів). Один із прикладів складання запиту можете бачити на рис. 3.1.2.

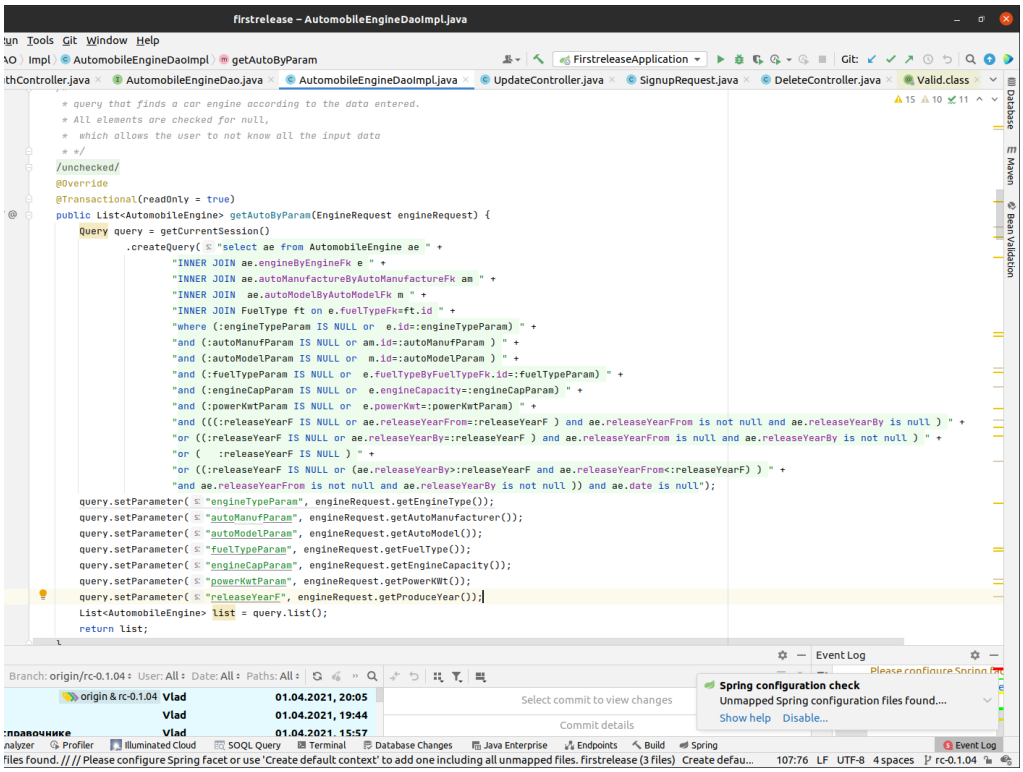

Рис. 3.1.2. Приклад складання запиту

Entity в JPA - це не що інше, як POJO, що представляють дані, які можуть зберігатися в базі даних. Сутність представляє таблицю, що зберігається в базі даних. Кожен екземпляр сутності представляє рядок у таблиці.

Директорія Payload складається з класів потрібних для подання тіла запиту і відповіді, яке отримує бекенд.

Директорія Config складається з класів конфігурації помічені анотацією відповідної анотацією. Анотація Spring @Configuration є частиною основи Spring Core. Анотація Spring Configuration вказує, що клас має методи визначення @Bean.

Тож контейнер Spring може обробляти клас і генерувати Spring Beans, який буде використовуватися в додатку. На рис. 3.1.3. наведено приклад конфігурації транзакцій.

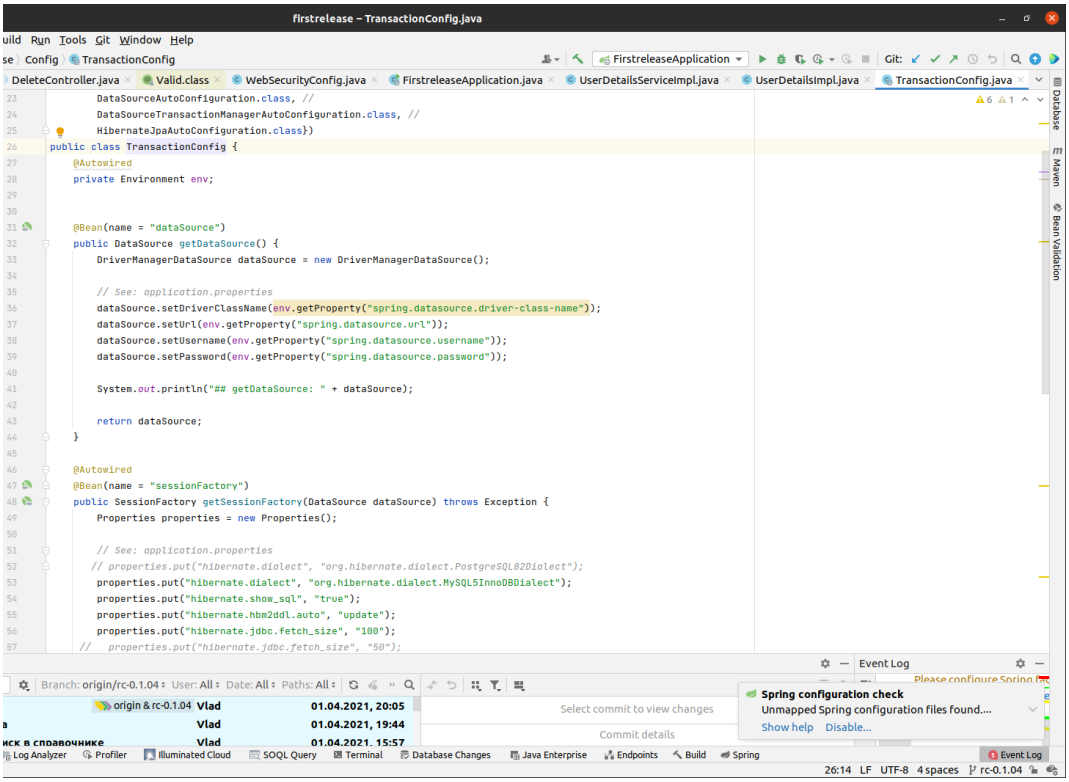

Рис. 3.1.3. Клас конфігурації транзвкцій

Директорія Security містить конфігурацію і реалізацію авторизації і аутентифікації з використанням специфікацій Spring Security. Це потужне і настроюване середовище перевірки автентичності та контролю доступу. Це де-факто стандарт захисту додатків на основі Spring. Приклад реалізації UserDetails зображено на рис. 3.1.4.

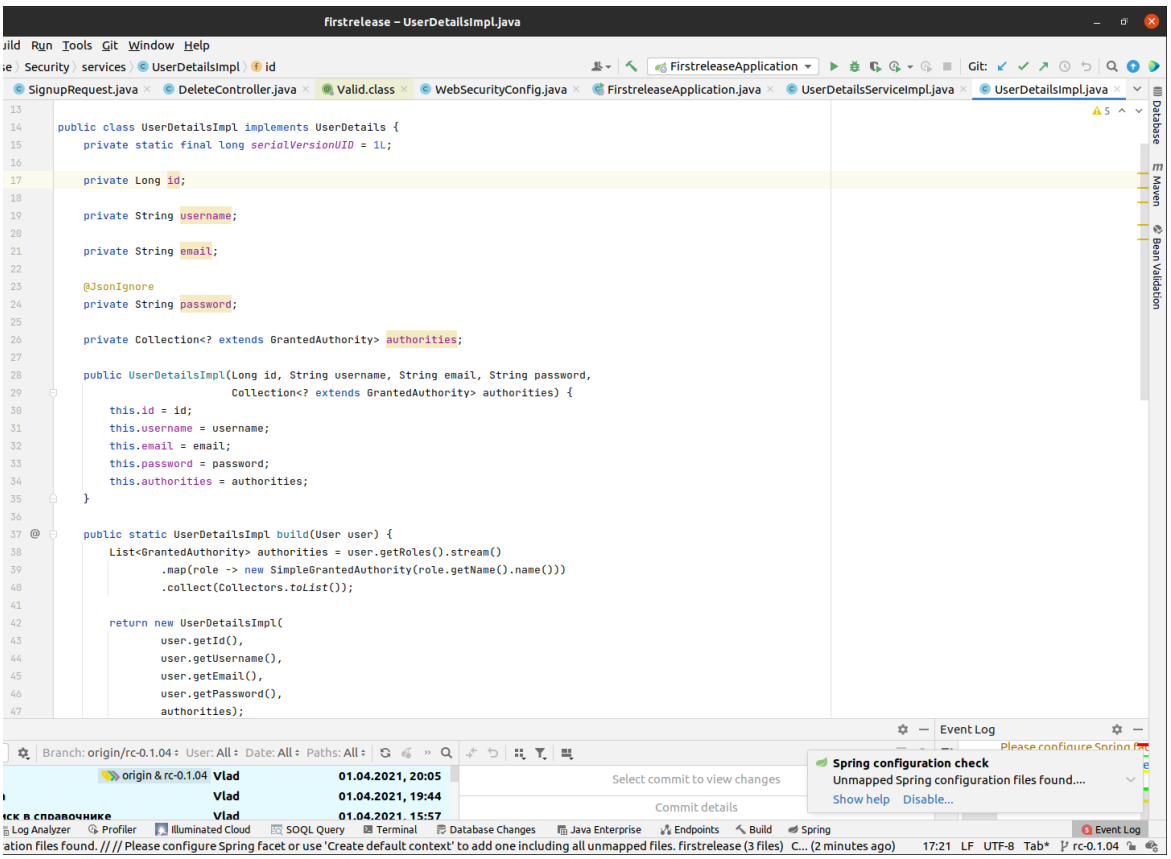

Рис. 3.1.4. Приклад реалізації UserDetails

Використання цього фреймворку гарантує проектованій інформаційній системі:

1. Комплексну і розширювану підтримка як для автентифікації, так і для авторизації;
2. Захист від атак, таких як фіксація сеансу, розбивка кліків, підробка міжсайтових запитів тощо;
3. Інтеграція API сервлетів;
4. Необов’язкова інтеграція з Spring Web MVC.

3.2. Розробка фронтенда на мові JavaScript та фремворка VueJS

Фронтенд має файлову структуру, яка зображена на рис. 3.2.1.

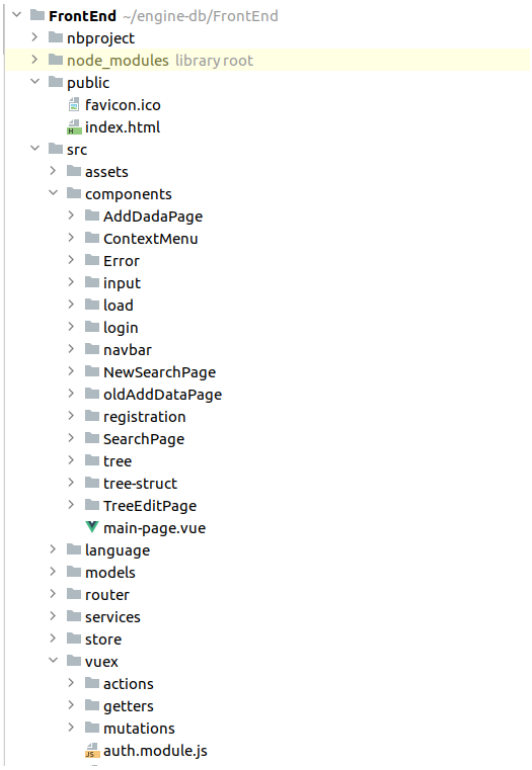

Рис. 3.2.1. Файлова структура інформаційної системи

В директорії Components містяться компоненти Vue, з яких складається видима частина фронтенда. Розберемо головні елементи на одній з робочих сторінок майстра. Розглянемо основний інтерфейс для роботи майстрів в компоненті NewSearch, який зображений на рис. 3.2.2.

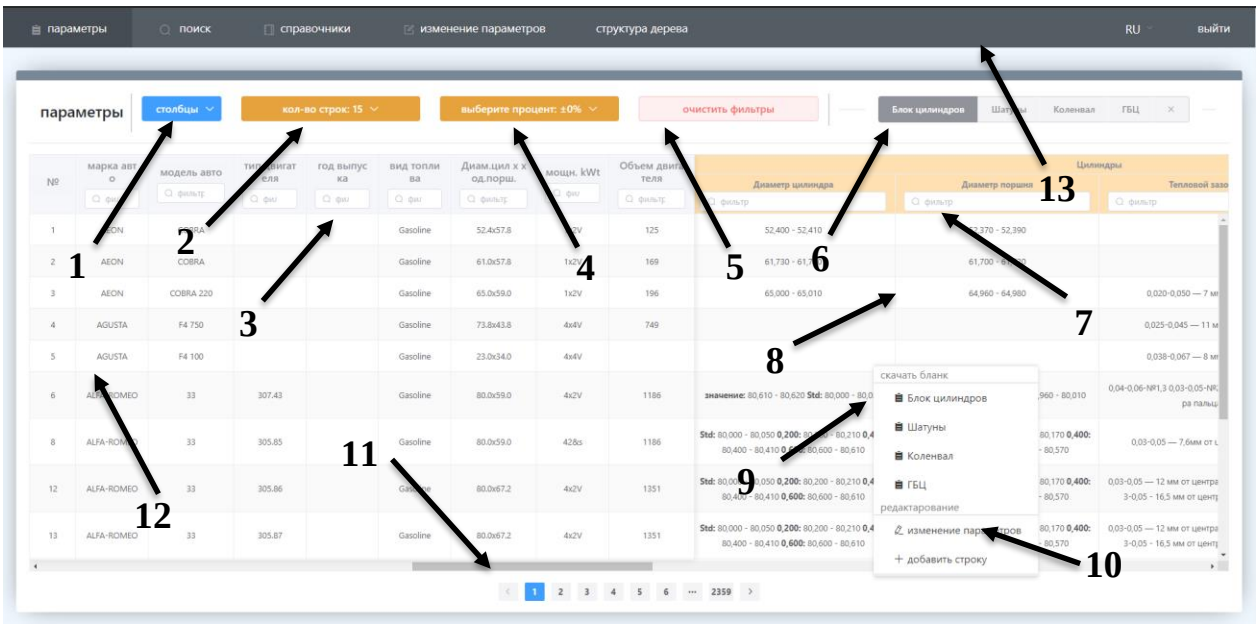

Рис. 3.2.2. Інтефейс пошуку в інформаційній системі

Опис основних елементів інтерфейсу:

1. Один з елементів конфігурації таблиці даних, з допомогою якого можна керувати, які стовпці відображати.

2. Елементів конфігурації, який відповідає кількість відображуваних рядків

3. Поле пошуку по основним даними авто двигуна

4. Елемент корекції пошуку. За допомогою цього елемента збільшуємо кількість пошукових даних. Якщо вводимо числове значення, пошук працює за допомогою формули, яка зображена на наступному рисунку.

$$(знач. * проц.) + знач. \geq дані \geq знач - (знач. * проц.)$$

Рис. 3.2.3. Формула корекції пошуку

де знач. – це значення введене для пошуку, дані – це шукані дані, проц. – це процент обраний в копоненті

5. Елемент який дозволяє очистити всі фільтри пошуку, та повернути початкові дані.

6. Меню для вибору відображення параметрів певного блоку даних. За допомогою цього елемента також можна відключити відображення ремонтних параметрів.

7. Поле пошуку основним параметрам авто двигуна. Враховує як і числові так і текстові дані.

8. Поле відображення параметрів

9. Власне контекстне меню, яке дозволяє завантажити бланк параметрів.

10. Частина контекстного меню, яке дозволяє швидко корегувати дані в обраному полі.

11. Елемент нумерації сторінок, який дозволяє змінювати певну сторінку.

12. Поле відображення даних авто двигуна.

13. Меню сайта, за допомогою якого переходимо по сторінкам.

Для аутентифікації використовувався компонент Login. На рис. 3.2.4. показаний інтерфейс входу в додаток.

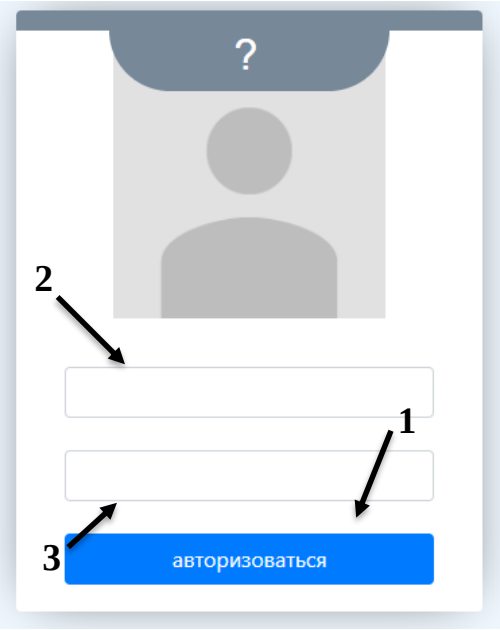

Рис. 3.2.4. Інтерфейс входу

Опис основних елементів інтерфейсу входу:

1. Кнопка для здійснення входу.
2. Поле вводу логіну користувача.
3. Поле вводу паролю користувача.

Для аутентифікації на стороні клієнта використовується JWT. Це, абревіатура від JSON Web Token, представляє собою відкритий стандарт, який дозволяє розробникам перевіряти підключеність даних інформації, що називається затвердженнями, за допомогою підписів. Ця підпис може бути секретною або парою відкритого / закритого ключів. Разом з заголовком і корисним навантаженням, вони можуть бути використані для генерації або побудови JWT.

Кожен наступний запит буде включати JWT як заголовок авторизації, дозволяючи доступ до захищених маршрутами і ресурсів. Крім того, як тільки внутрішній сервер перевіряє правильність підпису, він при необхідності витягує призначені для користувача дані з токена. Зверніть увагу, що для того, щоб гарантувати, що JWT дійсний, тільки сторона, що володіє ключами або секретом, несе відповідальність за підписання інформації.

У системах на основі аутентифікації JWT, коли користувач успішно входить в систему, використовуючи свої облікові дані, веб-токен JSON буде повернутий зухвалому клієнту. Всякий раз, коли користувач хоче отримати доступ до захищеного маршруту або ресурсу, призначений для користувача агент відправляє той же самий JWT, зазвичай в Authorizationзаголовке з використанням Bearer схеми. На рис. 3.2.4. проілюстровано схема роботи сторони клієнта с JWT.

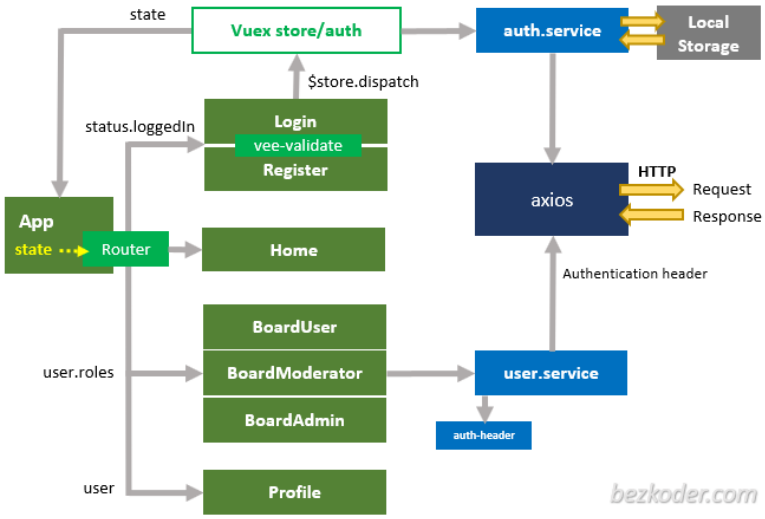

Рис. 3.2.4. Схема роботи JWT

JWT пропонує повністю механізм аутентифікації без збереження стану, оскільки стан користувача ніколи не зберігається в пам'яті сервера або базі даних. Захищені маршрути сервера перевірятимуть дійсний JWT в заголовку авторизації, і якщо він є, користувачеві буде надано доступ.

Зберігання токенов localStorageсопряжено зі своїм набором ризиків, оскільки вони уразливі для атак міжсайтового скриптинга (XSS). Дуже важливо ознайомитися з подробицями про запобігання проблем безпеки з JWT і про те, коли вам слід використовувати щось ще.

Для внесення та редагування всіх назв та значень та назв використовується сторінка зображена на рис. 3.2.5. Для отримання доступу на цю сторінку потрібно мати роль ADMIN, тому що ці дані може вносити тільки перевірений користувач.

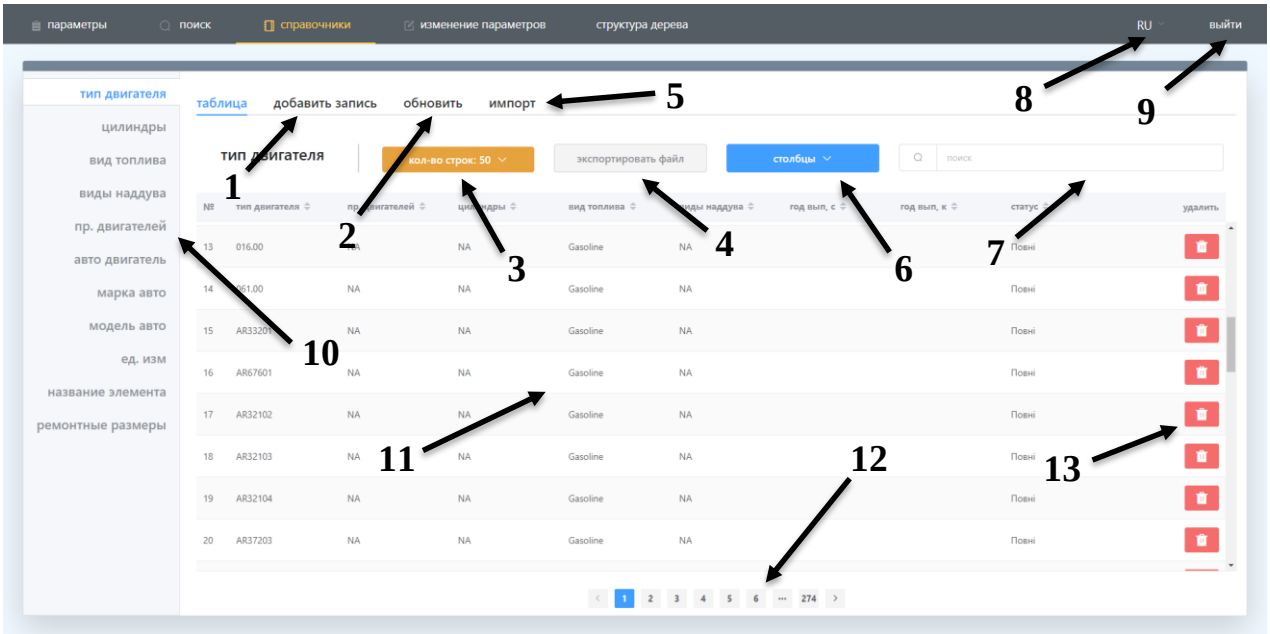

Рис. 3.2.5. Інтерфейс довідника

Опис основних елементів інтерфейсу довідника:

1. Вкладка додавання даних з перевіркою на унікальність.
2. Вкладка оновлення даних з перевіркою на унікальність.
3. Елементів конфігурації, який відповідає кількість відображуваних рядків.
4. експорт даних в Excel
5. Імпорт даних за допомогою excel файла
6. Один з елементів конфігурації таблиці даних, з допомогою якого можна керувати, які стовпці відображати, а які ні.
7. Рядок пошуку по введених даних. Пошук проводитися не тільки за назвою, а також по всіх текстових полях.
8. Вибір мови на якому зручно розуміти інтерфейс. Система підтримує 4 мови: українська, російська, польська, англійська
9. Кнопка виходу з системи
10. Меню вибору довідника для редагування
11. Поле відображення даних обраного довідника
12. Елемент нумерації сторінок, який дозволяє змінювати певну сторінку.

13. Кнопка видалення записів. У цьому дії задієте перевірка залежностей, так що не можна видалити запис від якої покладається інші записи

3.3. Розробка бази даних

На початку треба налаштувати доступ до бази даних, задавши значення користувача user, пароля password та назви бази даних name.

Команда для створення бази даних: CREATE DATABASE databaseName;. На рис. 3.3.1. зображено приклад підключення до локальної бази даних. За допомогою вбудованого інструментарію Datagrip, який дозволяє швидко налаштувати доступ до бази даних. Таблиці в цій системі представлені у вигляді дерева таблиць. За допомогою цих інструментів можна оперувати існуючою в базі інформацією.

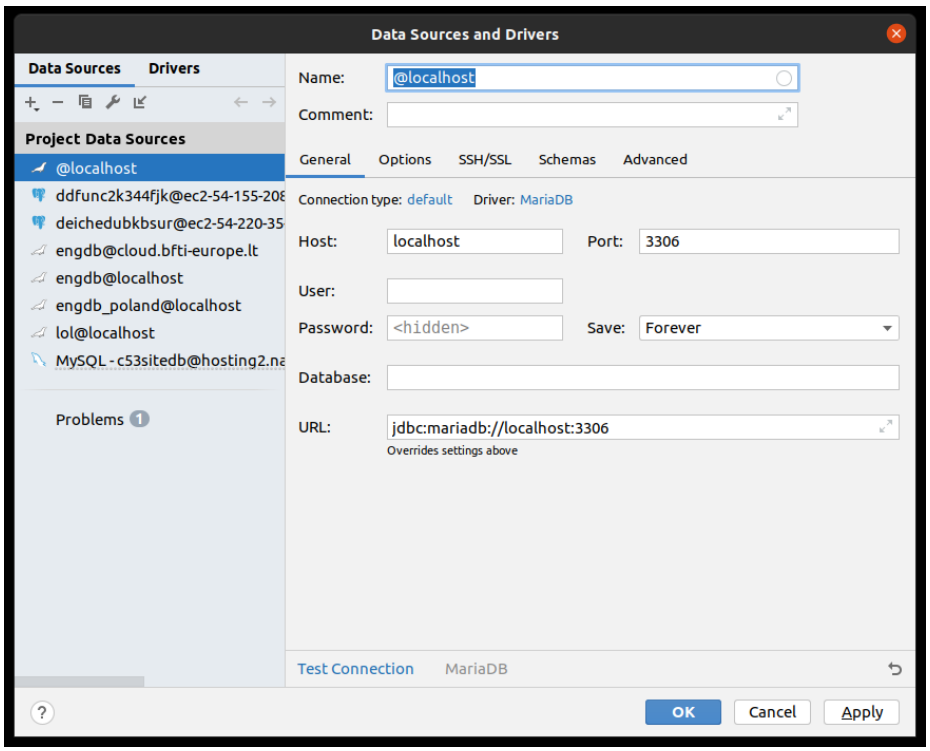

Рис. 3.2.1. приклад підключення до локальної бази даних

Після створення бази даних необхідно додати сутності та їх атрибути відповідно до розробленої інформаційної моделі. Для створення таблиці виконуємо

команду зображену на рис. 3.3.2. Обов’язково треба вказати первинні та вторинні ключі, які забезпечують зв'язок мж таблицями.

```
create table users
(
    user_id  bigint       not null
        primary key,
    email    varchar(50)  not null,
    password varchar(120) not null,
    username varchar(64)  not null,
    enabled  bit          null,
    constraint UK6dotkott2kjsp8vw4d0m25fb7
        unique (email),
    constraint UKr43af9ap4edm43mmtq01oddj6
        unique (username)
);

```

рис. 3.2.2. приклад створення таблиці

В результаті виконання описаних команд в локальній базі формуються таблиці, які відповідають об’єктам в інформаційній моделі. Схема розробленої бази даних представлена на рис. 3.3.3.

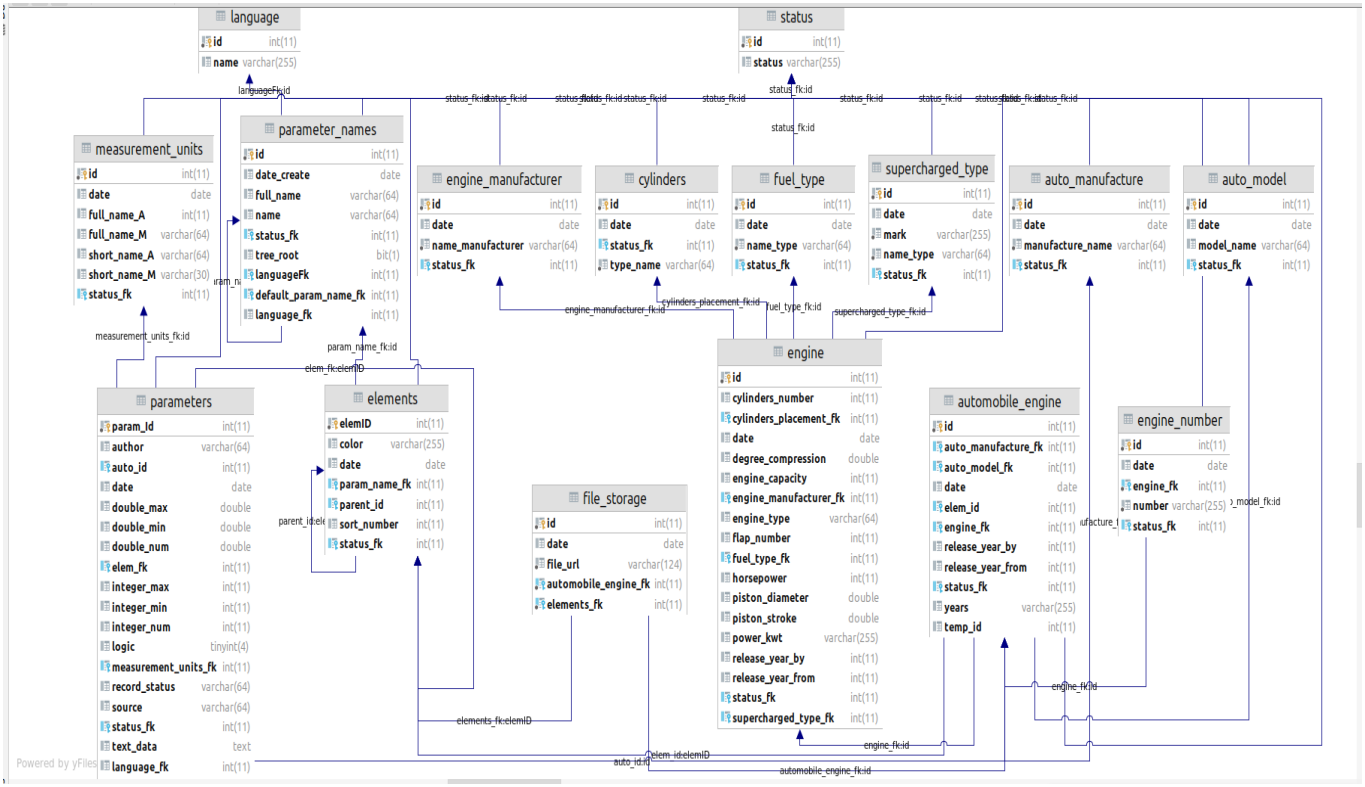

Рис. 3.3.3. Схема бази даних

Висновки до розділу 3

Основні етапи створення складної інформаційно-управляючої системи є розробка front-end (клієнтська) і back-end (серверна), також для зберігання даних потрібно було створити базу даних.

Бекед був створений на мові Java. В якості фреймворка був обраний Spring. Для створення структури системи використовувався MVC. Spring Security використовувався для реалізації авторизації і аутентифікації.

Для розробки фронтенда використовували прогресивну платформу JavaScript з відкритим кодом, що використовується для розробки інтерактивних веб-інтерфейсів, який іменується VueJS. Це один із відомих фреймворків, що використовується для спрощення веб-розробки. VueJS фокусується на рівні подання. Його можна легко інтегрувати у великі проекти для інтерфейсної розробки без будь-

яких проблем. Сторінки для виводу форм та інформації про об’єкти і документи створено за допомогою структур HTML і стилів CSS.

Найголовніше для інформаційно-керуючої системи є робота з даними. Щоб працювати з даними їх потрібно зберігати. Для цього я використовую реляційну базу даних MariaDb.

В результаті виконання описаних етапів була створена інформаційна система, якою можуть користуватися майстри з ремонту автомобільних двигунів.

Головна можливість розробленої системи полягає в можливості спростити ведення документації та підвищити ефективність майстрів.

ВИСНОВКИ

На сьогодні більшість підприємств намагається автоматизувати робочий процес шляхом відказу від не потрібних процесів, що вимагають витрату часу та роботи з документацією. З допомогою новітніх технологій для оптимізації робочого процесу співробітників, робочій процес стає легким та не відволікає від основної роботи, що у свою чергу збільшує ефективність підприємства.

У наш час більшість інформації, якою оперують компанії, існує в електронному форматі, програмним забезпеченням для підвищення ефективності роботи співробітників є автоматизована система управління робочими процесами.

Система була створена для компанії ProMototr, співробітники якої кожного дня звертаються по допомогу у ремонті двигунів, які в свою чергу користуються документацією з даними по ремонту, переважна більшість яких зберігається текстових файлах.

Проведення аналізу існуючих інформаційних систем ведення ремонтного процесу автомобільних двигунів дозволило виявити багато недоліків, як старий не практичний дизайн, мала швидкість у порівнянні з даною системою, висока вартість, гнучкість.

В процесі виконання роботи для розробки системи було обрано наступні технології та інструменти:

- MySQL – система управління реляційними базами даних, яка надає можливість використовувати структуровану мову запитів SQL для управління даними в базі;
- Spring – це платформа Java, яка забезпечує всебічну підтримку інфраструктури для розробки програм Java. Spring обробляє інфраструктуру, щоб ви могли зосередитись на своєму додатку.
- HTML, CSS – технології для створення інтерфейсу для користувачів;

- JavaScript– це легка, інтерпретована, об’єктно-орієнтована мова з першокласними функціями і найбільш відома як мова сценаріїв веб-сторінок, але вона використовується і в багатьох середовищах, що не належать до браузера.

- VueJs-- це прогресивна платформа JavaScript з відкритим кодом, що використовується для розробки інтерактивних веб-інтерфейсів.

- Java- це високорівнева, надійна, захищена та об'єктно-орієнтована мова програмування.

Оскільки інформаційна система ведення ремонтного процесу автомобільних двигунів містить веб-інтерфейс, вона не потребує додаткового завантаження та налаштування програмного забезпечення на пристрої користувачів. Безпосередній доступ до системи відбувається через веб-браузер.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. IntelliJ IDEA – IntelliJ IDEA overview [Електронний ресурс]. Режим доступу: https://www.jetbrains.com/help/idea/discover-intellij-idea.html#developer-tools
2. WebStorm IDEA – WebStorm Features [Електронний ресурс]. Режим доступу: https://www.jetbrains.com/webstorm/features/
3. Datagrip IDEA – Datagrip Introduction [Електронний ресурс]. Режим доступу: https://www.jetbrains.com/help/datagrip/meet-the-product.html
4. Spring Framework- Spring Framework Overview [Електронний ресурс]. Режим доступу: https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html
5. Герберт Шилдт – Java. Полное руководство 10-е издание, 2020. – с. 1488. – ISBN 9785604004364.
6. MariaDB – MariaDB Server: The open source relational database [Електронний ресурс]. Режим доступу: https://mariadb.org/
7. Relation Database – Relational Database Overview database [Електронний ресурс]. Режим доступу: https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html
8. JavaScript – About JavaScript [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript
9. HTML, CSS – Web Design 101: How HTML, CSS, and JavaScript Work [Електронний ресурс]. Режим доступу: https://blog.hubspot.com/marketing/web-design-html-css-javascript
10. Java back-end– The Beginner’s Guide to Backend Development [Електронний ресурс]. Режим доступу: https://learntocodewith.me/posts/backend-development/
11. JavaScript front-end – Front-end Developer Handbook [Електронний ресурс]. Режим доступу: https://frontendmasters.com/books/front-end-

handbook/2019/#:~:text=Front%2Dend%20web%20development%2C%20also,and%20interact%20with%20them%20directly

12. TCP/IP – TCP/IP Overview [Електронний ресурс]. Режим доступу: https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13769-5.html

13. RESP API – RESTful Web Services [Електронний ресурс]. Режим доступу: https://www.tutorialspoint.com/restful/restful_introduction.html

14. Spring Security – Spring Security Overview [Електронний ресурс]. Режим доступу https://auth0.com/blog/spring-security-overview/

15. Hibernate – Hibernate - Overview [Електронний ресурс]. Режим доступу https://www.h2kinfosys.com/blog/hibernate-overview/

16. VueJs – Reactive Data Binding [Електронний ресурс]. Режим доступу https://v1.vuejs.org/guide/overview.html#Reactive-Data-Binding

17. Spring Security – Spring Security Overview [Електронний ресурс]. Режим доступу https://auth0.com/blog/spring-security-overview/

ДОДАТКИ

Додаток А

Програмна реалізація класу DeleteController

```
import com.vshvet.firstrelease.Service.*;
import com.vshvet.firstrelease.Payload.Request.IdRequest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.Map;

@RestController
@RequestMapping("/delete")
public class DeleteController {


 private final ElementsService elementsService;

 private final AutomobileEngineService automobileEngineService;

 private final ParametrsService parametrsService;

 private final EngineNumberService engineNumberService;

 private final EngineService engineService;

 private final FuelTypeService fuelTypeService;

 private final AutoModelService autoModelService;

 private final EngineManufactureService engineManufactureService;

 private final ParameterNameService parameterNameService;

 private final MeasurementUnitsService measurementUnitsService;

 private final AutoManufactureService autoManufactureService;

 private final CylindersService cylindersService;
```

```
        public final SuperchargedTypeService superchargedTypeService;
public final FileStorageService fileStorageService;



 @RequestMapping(value = "/elements", //
    method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
      MediaType.APPLICATION_XML_VALUE})
 @ResponseBody
 public Map<String, Object> elements(@RequestBody IdRequest elementsIdRequest) {
  return new HashMap<String, Object>() {{
   put("listDependencyElements", elementsService.delete(elementsIdRequest.getId()));
  }};
 }

 @RequestMapping(value = "/paramName", //
    method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
      MediaType.APPLICATION_XML_VALUE})
 @ResponseBody
 public Map<String, Object> paramName(@RequestBody IdRequest elementsIdRequest) {
  return new HashMap<String, Object>() {{
   put("listDependencyElements", parameterNameService.delete(elementsIdRequest.getId()));
  }};
 }

 @RequestMapping(value = "/parameters", //
    method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
      MediaType.APPLICATION_XML_VALUE})
 @ResponseBody
 public Map<String, Object> parameters(@RequestBody IdRequest elementsIdRequest) {
  return new HashMap<String, Object>() {{
   put("listDependencyElements", parametrsService.delete(elementsIdRequest.getId()));
  }};
 }

 @RequestMapping(value = "/autoModel", //
    method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
      MediaType.APPLICATION_XML_VALUE})
 @ResponseBody
 public Map<String, Object> autoModel(@RequestBody IdRequest elementsIdRequest) {
  return new HashMap<String, Object>() {{
   put("listDependencyElements", autoModelService.delete(elementsIdRequest.getId()));
  }};
 }
```

```
            @RequestMapping(value = "/engine", //
    method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
        MediaType.APPLICATION_XML_VALUE})
@ResponseBody
public Map<String, Object> engine(@RequestBody IdRequest elementsIdRequest) {
  return new HashMap<String, Object>() {{
    put("listDependencyElements", engineService.delete(elementsIdRequest.getId()));
  }};
}


@RequestMapping(value = "/autoEngine", //
    method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
        MediaType.APPLICATION_XML_VALUE})
@ResponseBody
public Map<String, Object> autoEngine(@RequestBody IdRequest elementsIdRequest) {
  return new HashMap<String, Object>() {{
    put("listDependencyElements", automobileEngineService.delete(elementsIdRequest.getId()));
  }};
}


@RequestMapping(value = "/measurementUnits", //
    method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
        MediaType.APPLICATION_XML_VALUE})
@ResponseBody
public Map<String, Object> measurementUnits(@RequestBody IdRequest elementsIdRequest) {
  return new HashMap<String, Object>() {{
    put("listDependencyElements", measurementUnitsService.delete(elementsIdRequest.getId()));
  }};
}



@RequestMapping(value = "/cylinders", //
    method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
        MediaType.APPLICATION_XML_VALUE})
@ResponseBody
public Map<String, Object> cylinders(@RequestBody IdRequest elementsIdRequest) {
  return new HashMap<String, Object>() {{
    put("listDependencyElements", cylindersService.delete(elementsIdRequest.getId()));
  }};
}



@RequestMapping(value = "/fuelType", //
```

```
                method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
        MediaType.APPLICATION_XML_VALUE})
@ResponseBody
public Map<String, Object> fuelType(@RequestBody IdRequest elementsIdRequest) {
  return new HashMap<String, Object>() {{
    put("listDependencyElements", fuelTypeService.delete(elementsIdRequest.getId()));
  }};
}

@RequestMapping(value = "/engineManufacture", //
    method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
        MediaType.APPLICATION_XML_VALUE})
@ResponseBody
public Map<String, Object> engineManufacture(@RequestBody IdRequest elementsIdRequest) {
  return new HashMap<String, Object>() {{
    put("listDependencyElements", engineManufactureService.delete(elementsIdRequest.getId()));
  }};
}

@RequestMapping(value = "/autoManufacture", //
    method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
        MediaType.APPLICATION_XML_VALUE})
@ResponseBody
public Map<String, Object> autoManufacture(@RequestBody IdRequest elementsIdRequest) {
  return new HashMap<String, Object>() {{
    put("listDependencyElements", autoManufactureService.delete(elementsIdRequest.getId()));
  }};
}



@RequestMapping(value = "/fileStorage", //
    method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
        MediaType.APPLICATION_XML_VALUE})
@ResponseBody
public Map<String, Object> fileStorage(@RequestBody IdRequest elementsIdRequest) {
  return new HashMap<String, Object>() {{
    put("listDependencyElements", fileStorageService.delete(elementsIdRequest.getId()));
  }};
}



@RequestMapping(value = "/supercharged", //
    method = RequestMethod.POST, //
    produces = {MediaType.APPLICATION_JSON_VALUE, //
```

```
                MediaType.APPLICATION_XML_VALUE})
  @ResponseBody
  public Map<String, Object> supercharged(@RequestBody IdRequest elementsIdRequest) {
    return new HashMap<String, Object>() {{
      put("listDependencyElements", superchargedTypeService.delete(elementsIdRequest.getId()));
    }};
  }


  @Autowired
  public DeleteController(AutomobileEngineService automobileEngineService,
          ElementsService elementsService,
          ParametrsService parametrsService,
          EngineNumberService engineNumberService,
          EngineManufactureService engineManufactureService,
          MeasurementUnitsService measurementUnitsService,
          EngineService engineService,
          FuelTypeService fuelTypeService,
          AutoModelService autoModelService,
          ParameterNameService parameterNameService,
          FileStorageService fileStorageService,
          CylindersService cylindersService,
          AutoManufactureService autoManufactureService,
          SuperchargedTypeService superchargedTypeService) {
    this.fileStorageService = fileStorageService;
    this.superchargedTypeService = superchargedTypeService;
    this.cylindersService = cylindersService;
    this.autoManufactureService = autoManufactureService;
    this.engineManufactureService = engineManufactureService;
    this.engineNumberService = engineNumberService;
    this.parametrsService = parametrsService;
    this.automobileEngineService = automobileEngineService;
    this.elementsService = elementsService;
    this.engineService = engineService;
    this.fuelTypeService = fuelTypeService;
    this.autoModelService = autoModelService;
    this.parameterNameService = parameterNameService;
    this.measurementUnitsService = measurementUnitsService;


  }
}
```

Програмна реалізація класу UserDetailsService

```
package com.vshvet.firstrelease.Security.services;
import com.vshvet.firstrelease.DAO.UserDao;
import com.vshvet.firstrelease.Entity.Role;
import com.vshvet.firstrelease.Entity.User;
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.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {
  @Autowired
  UserDao userDao;



// @Override
// @Transactional
// public UserDetails loadUserByEmail(String username) throws UsernameNotFoundException {
//   //userDao.openCurrentSessionwithTransaction();
//
//   User user = userDao.findByEmail(username)
//       .orElseThrow(() -> new UsernameNotFoundException("User Not Found with username: " + username));
//   System.out.println(user.getRoles().toString());
//   //userDao.closeCurrentSessionwithTransaction();
//   return UserDetailsImpl.build(user);
// }


  @Override
  @Transactional
  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    //userDao.openCurrentSessionwithTransaction();


    User user = userDao.findByUsername(username)
        .orElseThrow(() -> new UsernameNotFoundException("User Not Found with username: " + username));


    System.out.println(user.getRoles().toString());
    //userDao.closeCurrentSessionwithTransaction();
    user.setEnabled(true);
```

```
        return UserDetailsImpl.build(user);
  }


}


    package com.vshvet.firstrelease.Security;

import com.vshvet.firstrelease.Entity.ERole;
import com.vshvet.firstrelease.Security.jwt.AuthEntryPointJwt;
import com.vshvet.firstrelease.Security.jwt.AuthTokenFilter;
import com.vshvet.firstrelease.Security.services.UserDetailsServiceImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(
    // securedEnabled = true,
    // jsr250Enabled = true,
    prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
  @Autowired
  UserDetailsServiceImpl userDetailsService;

  @Autowired
  private AuthEntryPointJwt unauthorizedHandler;

  @Bean
  public AuthTokenFilter authenticationJwtTokenFilter() {
    return new AuthTokenFilter();
  }

  @Override
  public void configure(AuthenticationManagerBuilder authenticationManagerBuilder) throws Exception {
```

```
            authenticationManagerBuilder.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
      return super.authenticationManagerBean();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
      return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
      http.cors().and().csrf().disable()
          .exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()
          .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
          .authorizeRequests().antMatchers("/**").permitAll()
          .antMatchers("/start").permitAll()
          .anyRequest().authenticated();

      http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);
    }
}
```