НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Індивідуальний дослідницький проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційне забезпечення робототехнічних систем »
спеціальності 126 «Інформаційні системи та технології»
на тему: «Пошукова база даних промислових роботів для АРМ проєктувальника робототехнічних систем»

Виконав:
студент IV курсу, групи ІК-81
Мальцев Нікіта Сергійович _____

Керівник:
доцент
Поліщук Михайло Миколайович _____

Засвідчую, що у цьому проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційне забезпечення робототехнічних систем »

ЗАВДАННЯ

на індивідуальний дослідницький проєкт студенту

Мальцеву Нікіти Сергійовичу

1. Тема проєкту **«Пошукова база даних промислових роботів для АРМ проєктувальника робототехнічних систем».**

Керівник проєкту: доцент, Поліщук Михайло Миколайович.

2. Термін подання студентом проєкту: 15 червня 2022 року

3. Вихідні дані до проєкту:
3.1Технічні характеристики промислових роботів (ПР).
3.2Вихідні данні для побудови бази даних.

4. Зміст розрахунково-пояснювальної записки :
4.1. Введення (обґрунтування теми роботи, перелік задач, що підлягають вирішенню).
4.2. Розділ 1. Огляд типів інформаційних баз даних.
4.3. Розділ 2. Розробка алгоритмічного забезпечення пошукової системи бази даних ПР
4.4. Розділ 3. Розробка програмного забезпечення бази даних роботів.

5. Перелік графічного матеріалу:
5.1. Електронна та паперова версії (планшети) діалогових вікон для користувача автоматизованого робочого місця проєктувальника.

6. Вимоги до рівня стандартизації:
6.1. ПЗ розробляється з урахуванням реалізації в середовищі WINDOW
6.2. Використовуються стандартні програмні засоби.

7. Дата видачі завдання « 27 » вересня 2021 р.

Календарний план

№ з/п	Назва етапів виконання проєкту	Термін виконання етапів проєкту	Примітка
1	Розділ 1. Огляд типів інформаційних баз даних.	25.11.21 р.	
2	Розділ 2. Розробка алгоритмічного забезпечення пошукової системи бази даних ПР.	15.02.22 р.	
3	Розділ 3. Розробка програмного забезпечення бази даних роботів.	30.04.22 р.	
4	Апробація, оформлення та підготовка до захисту роботи	25.05.22 р.	

Студент Нікіта МАЛЬЦЕВ

Керівник Михайло ПОЛІЩУК

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІК81.180БАК.003 ПЗ	Пояснювальна записка	54		
6	А3	ІК81.180БАК.003 Е1	Пошукова база даних	1		
7			Схема структурна			
8	А3	ІК81.180БАК.003 Е2	Пошукова база даних	1		
9			Схема функціональна			
10	А3	ІК81.180БАК.003 Е2	Пошукова база даних	1		
11			Схема функціональна			
12	А3	ІК81.180БАК.003 Е2	Пошукова база даних	1		
13			Схема функціональна			
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						

Зм.	Аркуш	№ докум.	Підпис	Дата	ІК81.180БАК.003 ТП		
Розроб.		Мальцев Н.С.			Пошукова база даних Відомість проєкту	Літ.	Аркуш 1 / Аркушів 1
Керівн.		Поліщук М.М.				КПІ ім. Ігоря Сікорського Група ІК-81	
Затв.							

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту містить 54 сторінок, 35 рисунків, 4 додатки, 10 джерел.

Об`єкт дослідження: фреймворк Sprig, як інструмент для створення пошукової база даних промислових роботів для АРМ проєктувальника робототехнічних систем.

Мета проєкту**:** створити пошукову базу даних промислових роботів для АРМ проєктувальника робототехнічних систем, використовуючи фреймворк Sprig та набір його інструментів.

У першому розділі зробили огляд типів інформаційних баз даних та створення інформаційних баз даних. Завдяки цьому здобули теоретичні знання для написання проєкту

У другому розділі детально розібралися у програмному забезпеченні для редагування коду, а також дізналися що таке фреймворк Spring. Завдяки цьому здобули знання як прискорити та модернізувати написання коду для проєкту

У третьому розділі почали розробку нашого проєкту у редакторі коду Eclipse та на основі фреймворку Spring. Завдяки минулим розділам зміг створити цей проєкт з можливістю його редагувати або модифікувати

КЛЮЧОВІ СЛОВА: Spring, Eclipse, база даних, функція, контролер.

ANNOTATION

The explanatory note of the diplom project contains 54 pages, 35 pictures, 4 appendices, 10 sources.

Object of research: Sprig framework, as a tool for creating a search database of industrial robots for the workstation of the designer of robotic systems.

The purpose of the project: to create a search database of industrial robots for the workstation of the designer of robotic systems, using the Sprig framework and a set of its tools.

The first section reviews the types of information databases and the creation of information databases. Due to this, they gained theoretical knowledge for writing a project

In the second section, we looked in detail at the code editing software, as well as learned what the Spring framework is. Thanks to this, they gained knowledge on how to speed up and modernize the writing of code for the project

In the third section, we started developing our project in the Eclipse code editor and based on the Spring framework. Thanks to the previous sections I was able to create this project with the ability to edit or modify it

KEY WORDS: Spring, Eclipse, database, function, controller.

Пояснювальна записка

до дипломного проєкту

на тему:

«Пошукова база даних промислових роботів для АРМ проєктувальника робототехнічних систем»

Київ – 2022 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Spring (Spring Framework) - є одним із найпопулярніших фреймворків для створення веб-додатків Java.

БД - база даних

VSCode (Visual Studio Code) - редактор відкритого коду

Фреймворк - бібліотека об'єктів класу

Eclipse - редактор відкритого коду

Figma -програма для редагування графіки

АРМ - Автоматизоване робоче місце

IoC (Inversion of Control) - інверсія контролю.

ВСТУП

Сьогодні людство переживає інформаційний вибух. Кількість інформації, що надходить до людини через усі інформаційні носії, постійно зростає. Тому для кожної людини, яка живе в інформаційному суспільстві, дуже важливо володіти засобами оптимального вирішення проблеми накопичення, упорядкування та раціонального використання інформації.

Саме тому потрібні такі бази даних, завдяки них розробники АРМ можуть прискорити процес створення своїх проєктів. Людина має багато потужних інструментів для модифікації пошукових систем для баз даних.

Spring Framework, або просто Spring, є одним із найпопулярніших фреймворків для створення веб-додатків Java. Інформація яка тут знаходиться, обов'язково розробникам АРМ знадобиться для кращого та прискореного розуміння між типами та моделями видів існуючих баз даних, також це допоможе прискорити процес створення нових баз даних.

1. ОГЛЯД ТИПІВ ІНФОРМАЦІЙНИХ БАЗ ДАНИХ

1.1. Огляд типів інформаційних баз даних.

1.1.1. Інформаційні системи

Сьогодні людство переживає інформаційний вибух. Кількість інформації, що надходить до людини через усі інформаційні носії, постійно зростає. Тому для кожної людини, яка живе в інформаційному суспільстві, дуже важливо володіти засобами оптимального вирішення проблеми накопичення, упорядкування та раціонального використання інформації.

Можливості людини в обробці інформації різко зросли з використанням комп’ютерів. У використанні ЕОМ для вирішення завдань інформаційного обслуговування можна виділити два періоди:

Початковий період, коли до вирішення проблем обробки інформації та організації даних було залучено вузьке коло людей, системних програмістів. Цей період характеризується тим, що програмні засоби були створені для вирішення конкретної задачі обробки даних. У той же час для вирішення іншої задачі, яка використовувала ті самі дані, необхідно було створити нові програми;

Період системного використання ЕОМ. Для вирішення комплексу завдань на комп’ютері створюються програмні засоби, які оперують одними і тими ж даними, використовуючи єдину інформаційну модель об’єкта. Ці засоби не залежать від природи об'єкта, саме тому що, його моделі, їх можна використовувати для інформаційного обслуговування різноманітних завдань у різних системах. Людство прийшло до організації інформації в інформаційних системах.

Інформаційні системи (ІС) — це великі масиви даних разом із програмним та апаратним забезпеченням для їх обробки. Розрізняють такі види ІС: фактологічні, документальні та експертні системи.

Фактографічна IS — це масив фактів — конкретних значень даних про об’єкти реального світу.

Інформація у фактографічній ІС зберігається в чітко структурованому вигляді, тому вона здатна дати однозначні відповіді на поставлені запитання, наприклад:

–«Хто переможець Чемпіонату зі спортивної гімнастики 1999 року?»;

–«Кому належить AUDI 8 автомобіль з реєстраційним номером АА8997АР?»;

–«Який номер телефону в бухгалтерії ЗСУ?»;

–«Хто став Президентом на виборах у березні 2002 року?» тощо.

Фактографічні інформаційні системи використовуються буквально в усіх сферах людської діяльності - у науці, матеріальному виробництві, транспорті, медицині, державному та громадському житті, торгівлі, криміналістиці, мистецтві, спорті.

Документальні інформаційні системи виконують принципово інший клас завдань, не передбачатиме однозначні відповіді на запитання. Бд систем формує сукупність неструктурованих текстових документів (статей, книг, рефератів, текстів законів) та графічних об’єктів, оснащених тим чи іншим формалізованим пошуковим апаратом. Призначення системи, як правило, полягає у видачі у відповідь на запит користувача переліку документів або об'єктів, які певною мірою задовольняють умовам, сформульованим у запиті. Наприклад: наведіть список усіх статей, у яких зустрічається слово «Пушкін». Особливістю документної системи є здатність, видавати документи, непотрібні користувачеві (наприклад, де слово «Пушкін» вживається в іншому значенні, ніж очікувалося), а з іншого боку, не видати необхідні (наприклад, якщо автор використав якийсь синонім або неправильно написано). Документальна система повинна мати можливість визначати значення певного терміна за контекстом, наприклад, розрізняти «ромашка» (рослина), «ромашка» (тип друкуючої головки принтера).

Експертні системи (ЕС) — це інтелектуальні системи, призначені для виконання ролі «радника», побудовані на основі формалізованого досвіду та знань експерта. Ядром ЕС є бази знань, які містять знання експертів (фахівців) певної галузі, на основі яких ЕС дозволяє моделювати міркування фахівців з

даної предметної області. Зазначена класифікація та віднесення ІС до того чи іншого типу є застарілими, оснащеними структурованими дескрипторами.

1.1.2. Текстові документи та бази даних

Значна частина користувачів, купуючи комп'ютер або отримуючи до нього доступ, в першу чергу освоює операції з текстовими файлами. На першому етапі зазвичай комп’ютер використовується як зручна і «розумна» друкарська машинка (для підготовки, зберігання, модифікації та друку всіляких листів, рефератів, тез, оголошень, статей тощо).

Навряд чи багато хто думає, що вже на цьому етапі вони використовують примітивну інформаційну систему, яка в даному випадку складається з таких елементів:

1) текстовий редактор як інструмент для маніпулювання текстами;
2) групи текстових файлів (баз даних) як об’єкт обробки.

На наступному етапі багатьом спадає на думку використовувати текстовий файл як своєрідну житницю, куди можна легко ввести різноманітну інформацію «список», наприклад, рецепти, номери телефонів ваших друзів, довідники відеотеки, музична бібліотека, адреси та назви організацій тощо. Спосіб подання та розміщення інформації в таких «амбарних» книгах зазвичай вигадує сам користувач. Наприклад, адвокат може помістити в текстовий файл картки своїх клієнтів із зазначенням прізвища, імені та по батькові, адреси проживання, теми юридичної консультації та інших даних, наприклад: «Іванов П.І., вул. Сафонова, д. 12, спадщина”, “Сидоров П.Т., вул. Трефська, 34, кв. 25, ДТП тощо.

Які недоліки такого підходу? Створюючи бази даних, прагнемо надати собі можливість, по-перше, систематизувати інформацію за різними ознаками (наприклад, за темою консультації), а по-друге, швидко витягувати зразки з довільною комбінацією характеристик (наприклад, клієнти, які звернулися за консультацією щодо отримання спадщини). Однак описана вище організація даних теж не дозволить, оскільки організувати інформацію в текстовому файлі

набагато складніше, ніж навіть у картонній коробці. Крім того, комп’ютер навіть не зможе вибрати клієнтів з однаковою темою консультації, якщо одна й та сама тема буде по-різному записана в записі про різних клієнтів (наприклад, «спадок», «Nael.» тощо).

Для того, щоб комп’ютер міг точно шукати й впорядковувати дані, спочатку необхідно розробити й дотримуватись певних правил (угоди) щодо того, як інформація подається під час запису даних. Стосовно вищеописаної інформаційної системи адвоката, це означає, що тема консультації має бути точно зазначена у всіх випадках вступу. Усі записи про клієнтів мають бути однакової довжини (наприклад, два рядки на клієнта), положення опису певних атрибутів даних у кожному записі має бути однаковим (наприклад, запис починається з прізвища, записується тема юридичної консультації з початку другого рядка). Цей процес адаптації форматів даних і значень до можливостей комп’ютера, тобто усунення довільності у поданні довжини і (або) значень, називається структуруванням інформації. Іншими словами, структурування – це введення умов щодо того, як дані представлені. Звідси випливає, що інформаційна система — це сукупність у тій чи іншій мірі структурованих даних (баз даних) і комплекс апаратно-програмних засобів для зберігання даних та маніпулювання ними.

1.1.3. Типи моделей даних

Основою інформаційної системи, об'єктом її обробки є база даних (БД). База даних — це сукупність інформації про конкретні об’єкти реального світу в будь-якій предметній області або розділі предметної області. Наприклад, база даних про університети (вища освіта), база даних про ліки (медицина), база даних про автомобілі (автомагазин), база даних про будівельні матеріали (склад) тощо. Синонімом терміну «база даних» є « банк даних».

Центром бд є модель даних, яка є структурою даних, умовами щодо того, як вони представлені, та операціями для маніпулювання ними. Іншими словами, це формалізований опис між ними.

Існує три основних типи моделей даних:

-ієрархічна;

-мережева;

-реляційна.

Ієрархічна структура — це сукупність елементів, у якій дані одного рівня підпорядковані даним іншого рівня, а зв’язки між елементами утворюють деревовидну структуру. У такій структурі початкові елементи породжують інші елементи, а ці елементи, у свою чергу, породжують наступні елементи тощо. Головне, що кожен дочірній елемент має лише одного «батька». Зауважте, що в ієрархічній структурі батьківським елементом може бути не сам об’єкт, а лише конкретний екземпляр об’єкта. Прикладом ієрархічної бази даних може бути ваше сімейне дерево.

Існують і більш складні - мережеві структури, в яких кожен згенерований елемент може мати більше одного батьківського елемента. Модель мережевих даних відрізняється від ієрархічної тим, що кожен елемент структури мережевих даних асоціюється з елементом. Складна мережева структура бази даних, що містить інформацію про студентів, які займаються різними гуртами. При цьому можливі заняття одного учня в різних гуртках, а також відвідування кількома учнями занять в одному гуртку. Мережні та ієрархічні структури можна звести до простих двовимірних таблиць.

1.1.4. Реляційні бази даних

Найзручнішим як для користувача, так і для комп’ютера є подання даних у вигляді двовимірної таблиці – з такими таблицями працює більшість сучасних інформаційних систем. Бази даних, які складаються з двовимірних таблиць, називаються реляційними (по-англійськи «relation» — відношення). Основна

ідея реляційного підходу полягає в представленні довільної структури даних у вигляді простої двовимірної таблиці.(Рис.1.1)

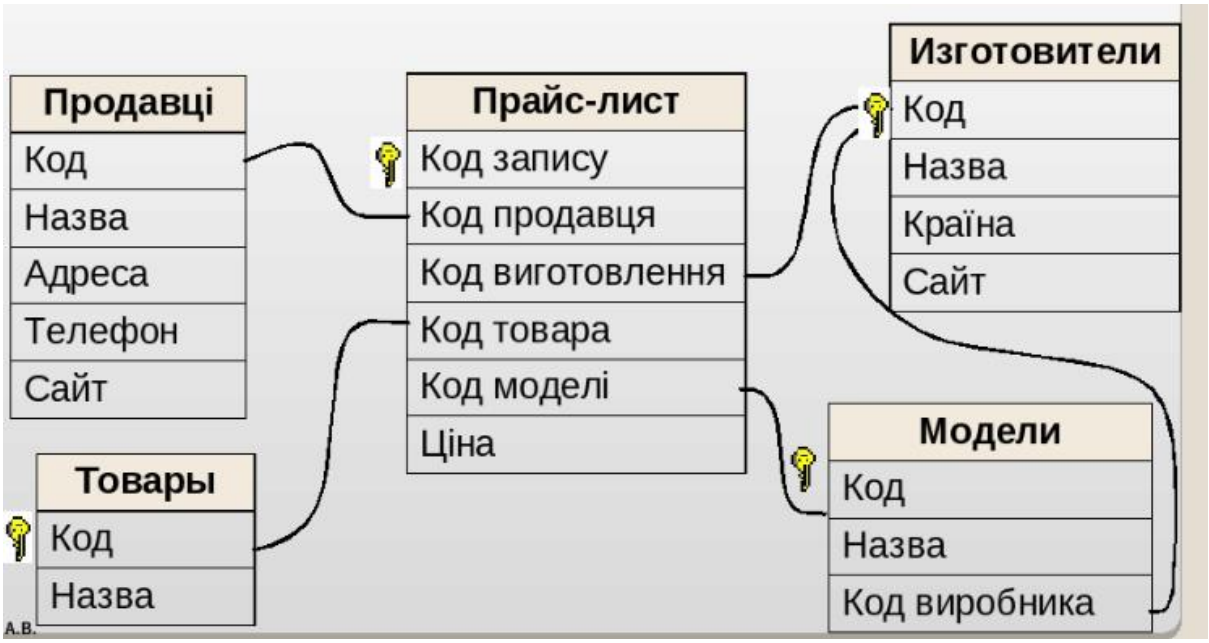

Рисунок 1.1 – Реляційні бази даних

Прикладом реалізації реляційної моделі даних може бути інформаційна таблиця студента.

Як видно з наведеного вище прикладу, реляційна таблиця має такі властивості:

1) кожен рядок таблиці є одним елементом даних (інформація про одного учня):
    a) усі стовпці таблиці є однорідними, тобто всі елементи в стовпці мають однаковий тип і довжину (наприклад, у стовпці «Ім’я» відображаються імена студентів типу символів з максимальною довжиною 17 символів);
    b) кожен стовпець має унікальну назву (наприклад, у таблиці немає двох стовпців Name).
2) ідентичні рядки в таблиці не допускаються (кожний учень записується лише один раз):

а) порядок рядків і стовпців у таблиці може бути довільним (запис про учня в таблиці робиться при вступі до школи, при цьому порядок стовпців не має значення).

1.1.5. Структурні елементи реляційної бази даних

На прикладі реляційної таблиці розглянемо основні структурні елементи бази даних:

1) у реляційних базах даних будь-які набори даних представлені у вигляді двовимірних таблиць (відношень), подібно до списку студентів, описаних вище. Кожна таблиця складається з фіксованої кількості стовпців і певної (змінної) кількості рядків. Опис стовпців називається компоновкою таблиці;
2) кожен стовпець таблиці представляє поле — елементарну одиницю логічної організації даних, якій відповідає неподільна одиниця інформації — реквізити об’єкта даних (наприклад, прізвище, адреса учня);
3) для опису поля використовуються наступні характеристики:
    a) назва поля (наприклад, номер особової справи, Прізвище);
    b) тип поля (наприклад, символ, дата);
    c) додаткові характеристики (довжина поля, формат, точність).

Наприклад, поле «Дата народження» може мати тип «дата» і довжину 8 (6 цифр і 2 крапки, що розділяють день, місяць і рік у записі дати).

Кожен рядок таблиці називається записом. Запис логічно об’єднує всі поля, що описують один об’єкт даних, наприклад, усі поля в першому рядку наведеної вище таблиці описують дані про студента Петрова Івана Васильовича 12.03.1989 р.н., який проживає за адресою вул. Горького, 12-34, навчається в 4А класі, номер особової справи - П-69. Система нумерує записи в порядку: 1,2, ..., n, де n – загальна кількість записів (рядків) у таблиці на даний момент. На відміну від кількості полів (стовпців) у таблиці, кількість записів під час роботи БД може змінюватися будь-яким чином (від нуля до мільйонів). Кількість полів, їх назви та типи також можна змінити, але це вже спеціальна операція, яка називається зміною макета таблиці.

Структура запису файлу визначає поля, значення яких є простим ключем, що ідентифікує екземпляр запису. Прикладом такого простого ключа в таблиці

Students є поле номер файлу, значення якого однозначно ідентифікує один об’єкт таблиці – одного учня, оскільки в таблиці немає двох учнів з однаковим номером файлу.

Кожне поле може бути включено в кілька таблиць (наприклад, поле Прізвище можна включити в таблицю Список тих, хто займається театральною групою).

1.1.6. Системи управління базами даних та їх функції

У сучасній технології баз даних для створення баз даних, підтримки та підтримки їх використовується спеціалізоване програмне забезпечення – системи управління базами даних. СУБД — це набір програмних і мовних засобів, необхідних для створення та функціонування баз даних.

На етапі розробки БД СУБД служить для опису структури БД:

-визначення таблиць;

-визначення кількості полів;

-тип відображаються в них даних;

-розміри поля;

-визначення зв'язків між таблицями.

Крім таблиць більшість СУБД передбачає створення спеціальних інструментів для роботи з даними - форм, запитів. Під час роботи баз даних СУБД забезпечує редагування структури бази даних, наповнення її даними, пошук, сортування, вибір даних за заданими критеріями та формування звітів.

В інформаційних системах, які працюють на IBM-сумісних персональних комп’ютерах, набули поширення так звані dBASE-подібні системи управління базами даних, такі як dBASE, FoxPro і Clipper. Для користувачів важливо, щоб, відрізняючись один від одного командними мовами та форматом індексних файлів, усі ці СУБД використовували однакові файли бази даних з розширенням .DBF, формат яких на деякий час став своєрідною базою даних. стандартний.

У базах даних, подібних до dBASE, фактично використовується реляційний підхід до організації даних, тобто кожен файл .DBF є двовимірною

таблицею, яка складається з фіксованої кількості стовпців і змінної кількості рядків (записів). У термінах, прийнятих у технічній документації, кожен стовпець відповідає полю одного з п'яти типів (N - числовий, C - символьний, D - дата, L - логічний, M - примітка), а кожному рядку - запис фіксованого довжини, що складається з полів фіксованого числа. За допомогою командних мов цих СУБД створюються та коригуються макети файлів .DBF (описи таблиць), створюються файли індексів, описуються процедури роботи з базами даних (читання, пошук, зміна даних, звітування тощо). більше).

Характерною особливістю файлу .DBF є простота і наочність: фізичне представлення даних на диску точно відповідає представленню таблиці на папері. Однак загалом системи на основі файлів .DBF слід вважати застарілими.

Також великою популярністю користуються інші СУБД (з іншим форматом файлів) – Paradox, Clarion та ін. Слід підкреслити, що перераховані системи походять від MS-DOS, але зараз майже всі вони вдосконалені та мають версії для Windows.

Серед сучасних реляційних систем найпопулярнішими СУБД для Windows є Access від Microsoft, Approach від Lotus, Paradox від Borland. Багато з цих систем підтримують технологію OLE і можуть маніпулювати не тільки числовою та текстовою інформацією, а й графічними зображеннями (кресленнями, фотографіями) і навіть звуковими та відеокліпами.

Ці СУБД часто називають настільними СУБД, що означає відносно невеликий обсяг даних, що обслуговуються цими системами. Однак часто з ними працюють не лише окремі користувачі, а й цілі команди (особливо в локальних мережах).

У той же час більш потужні реляційні СУБД з так званим доступом SQL поступово переміщаються в центр сучасних інформаційних технологій. Ці СУБД засновані на технології клієнт-сервер. Серед провідних виробників таких систем – Oracle, Centura (Gupta), Sybase, Informix, Microsoft та інші.

1.2 Створення інформаційних баз даних

1.2.1 Створення таблиць

Кожна таблиця містить інформацію про об'єкти одного типу. Кожен рядок – це запис, набір даних про певний об’єкт. Кожен стовпець - це поле, однорідні дані про всі об'єкти. Кожна окрема таблиця не може мати однакові записи. Загальна кількість заявок не обмежена. При розробці бази даних користувач, відповідно до аналізу програмного забезпечення, повинен визначити:

1) загальна кількість столів;
2) назва та кількість полів у кожній таблиці.

Типи та розміри даних для кожного з полів таблиці (відповідно до типу інформації, що зберігається, - текст, цифри, малюнки, посилання, дати):

1) ключові поля, що забезпечують контроль над унікальністю записів;
2) можуть бути інші необхідні параметри;
3) назви таблиць і склад їх полів повністю визначаються структурами реляційних таблиць, розробленими в процесі інформаційно-логічного проєктування;
4) типи та розміри даних для кожного з полів визначаються описами деталей, отриманих під час дослідження програмного забезпечення.

1.2.2 Режими створення таблиці

Якщо відкрито вікно бази даних: активуйте вкладку Таблиця і натисніть Створити:

1) access запропонує список режимів створення таблиці:
2) режим таблиці;
3) конструктор;
4) майстер;
5) імпорт;
6) посилання на таблицю.

Тестуємо конструктор і майстер. При виборі режиму конструктора відкривається вікно, що складається з двох частин.

У верхній частині вікна необхідно вказати:

1) ім'я поля;
2) тип даних;
3) опис (коментар). (не обов'язково).

Внизу - властивості поля:

1) розмір поля - відповідно до типу даних і розміру, обраного при розробці структури;
2) формат поля;
3) підпис (при відображенні на екрані має вищий пріоритет, ніж ім'я поля);
4) значення за замовчуванням;
5) повідомлення про помилку;
6) обов’язкове поле (значення, зазначене в полі);
7) індексація полів тощо.

Після визначення властивостей потрібно:

1) вкажіть ключове поле, для якого;
2) виберіть це поле (або групу полів);
3) натисніть на кнопку «Ключ».

Після цього потрібно:

1) натисніть кнопку [x], щоб закрити вікно «Конструктор» таблиці;
2) у відкритому діалоговому вікні;
3) вкажіть ім’я створюваної таблиці;
4) натисніть кнопку [Ok].

1.2.3 Створення таблиць у режимі майстра

При створенні таблиці в режимі «Майстер» потрібно вибирати найбільш підходящу таблицю з бібліотеки Access і редагувати (завершити) її в режимі проєктування. Порядок визначення ключових полів.

Кожна таблиця реляційної бази даних повинна мати ключовий елемент, який дозволяє ідентифікувати кожен конкретний запис і виключає повторення ідентичних записів.

Правильний вибір ключового елемента дає змогу задовольнити один із принципів нормалізації: кожен рядок таблиці (запис) містить інформацію про конкретний об’єкт. Немає дублікатів рядків.

Типи ключових елементів:

1) простий ключ складається з одного поля, яке має лише унікальні значення в межах даної таблиці;
2) складений ключ - складається з кількох полів. Встановлюють у тих таблицях, в яких неможливо знайти одне поле з неповторюваними значеннями.

Приклад вибору ключового елемента:

1) поле лічильника - вибирається, якщо в таблиці неможливо знайти компактне поле (або групу полів) з неповторюваними значеннями. Access автоматично збільшує значення в цьому полі на 1 для кожного нового запису;
2) поле лічильника може вибрати користувач, але воно також буде встановлено Access за замовчуванням, якщо користувач забуде вказати ключове поле під час створення таблиці.

Щоб визначити ключове поле, користувач:

1) вибирає потрібне поле або групу полів (за допомогою клавіші Ctrl);
2) натискає кнопку «Ключове поле»;
3) створення схеми даних.

Дані різних таблиць об’єднуються в один банк за допомогою об’єкта Access, який називається «схема даних» (схема даних). Столи з'єднані лініями зв'язку в один блок.

Існують такі типи зв'язків між таблицями реляційної бази даних:

1) 1:M - один до багатьох; один запис таблиці «А» відповідає кільком записам у таблиці «В». Основний вид комунікації в реляційних базах даних;
2) M:H - багато до багатьох. Слова …….. Відношення типу M:N зазвичай не вказуються на схемі даних, але встановлюються під час створення запитів;
3) 1:1 - один до одного, - організовується у разі розбиття окремої таблиці, що містить велику (можна сказати, безмежну) кількість полів на 2 або кілька з видимою кількістю полів.

Формування схеми даних. Щоб сформувати схему даних, необхідно: Натисніть кнопку «Схема даних» (або меню Інструменти рядок Схема даних)

1) відкривається «Схема даних»;
2) спочатку всі пов’язані таблиці повинні бути розміщені в робочій області вікна, що відкривається, для цього;
3) виконайте по черзі подвійне клацання на назві кожної з доданих таблиць;
4) або з виділеною назвою таблиці натисніть кнопку «Додати».

Примітка: якщо список таблиць не відображається на екрані під час виклику схеми даних. Треба виконати такі дії:

1) директива меню
2) інструменти
3) додати таблицю.

Завершивши підбір таблиць, необхідно встановити зв'язки між ними, для цього.

Зв'язки між таблицями встановлюються таким чином:

1) перетягніть ключове поле основної таблиці за допомогою миші в область однойменного поля (не обов'язково входить до ключа) підлеглого;
2) відпустіть там кнопку миші (опустіть поле).

Після цього на екрані ТО відкривається Access, в якому пропонує визначити необхідність контролю цілісності даних (розміщує у вікні опцію з такою назвою).

Користувач встановлює у вікні, що відкрилося, опцію «Цілісність даних».

Клацає [Ок]. Якщо Access позначає зв’язок як один до багатьох, зв’язок встановлюється. Якщо Access відмовляється визначити зв’язок, слід шукати помилку.

Цілісність — це унікальність, послідовність і правильність даних, розподілених по окремих таблицях.

Щоб забезпечити цілісність користувач встановлює параметр у діалоговому вікні під час створення кожного з посилань під час створення цілісної схеми даних.

Доступ - стежить за виконанням умов цілісності і з цієї точки зору не дозволяє:

1) введіть дані в підтаблицю, яких немає в основній таблиці;
2) видалити з основної таблиці дані, на які посилаються в підпорядкованих таблицях.

1.3. Висновок

Був виконан огляд існуючих типів інформаційних баз даних. Дана інформація обов'язково знадобиться для кращого розуміння різниці між типами та моделями видів існуючих баз даних, також це допоможе прискорити процес створення нових баз даних або редагування вже існуючих баз даних для проєкту пошукової бази даних промислових роботів для АРМ проєктувальника робототехнічних систем.

2. РОЗРОБКА АЛГОРИТМІЧНОГО ЗАБЕЗПЕЧЕННЯ ПОШУКОВОЇ СИСТЕМИ БАЗИ ДАНИХ ПР

2.1. Опис Spring

2.1.1.Spring Framework

Spring Framework, або просто Spring, є одним із найпопулярніших фреймворків для створення веб-додатків Java.

Фреймворк — це бібліотека, але є один момент. Використовуючи бібліотеку, розробник просто створюєте об’єкти класів, які в ній є, викликаєте потрібні методи і таким чином отримує потрібний результат. Тобто є більш імперативний підхід. Людина чітко вказує у програмі, в який конкретний момент який об’єкт необхідно створити, в який момент викликати конкретний метод тощо.

З фреймворками справи йдуть трохи інакше. Людина просто пише деякі свої класи, записує туди частину логіки, а сам фреймворк створює об’єкти ваших класів і викликає для вас методи. Найчастіше класи реалізують деякі інтерфейси з фреймворка або успадковують деякі класи від нього, отримуючи таким чином частину функціональних можливостей, уже написаних для користувача. Але це не повинно бути таким чином.

У Spring, наприклад, намагаються максимально відійти від такої жорсткої зв’язки (коли класи безпосередньо залежать від деяких класів/інтерфейсів з цього фреймворку) і використовують для цього анотації. Повернемося до цього моменту пізніше. Але важливо розуміти, що spring - це лише набір деяких класів та інтерфейсів, які вже написані для вас. Також хочу відразу зазначити, що spring можна використовувати не тільки для веб-додатків, але і для найпоширеніших консольних програм, які так звичні всім нам. А сьогодні навіть напишемо щось подібне.

2.1.2.Структура

Але spring — це не одна конкретна рамка. Це загальна назва для ряду невеликих фреймворків, кожен з яких виконує певну роботу. (рис. 2.1) та (рис. 2.2)

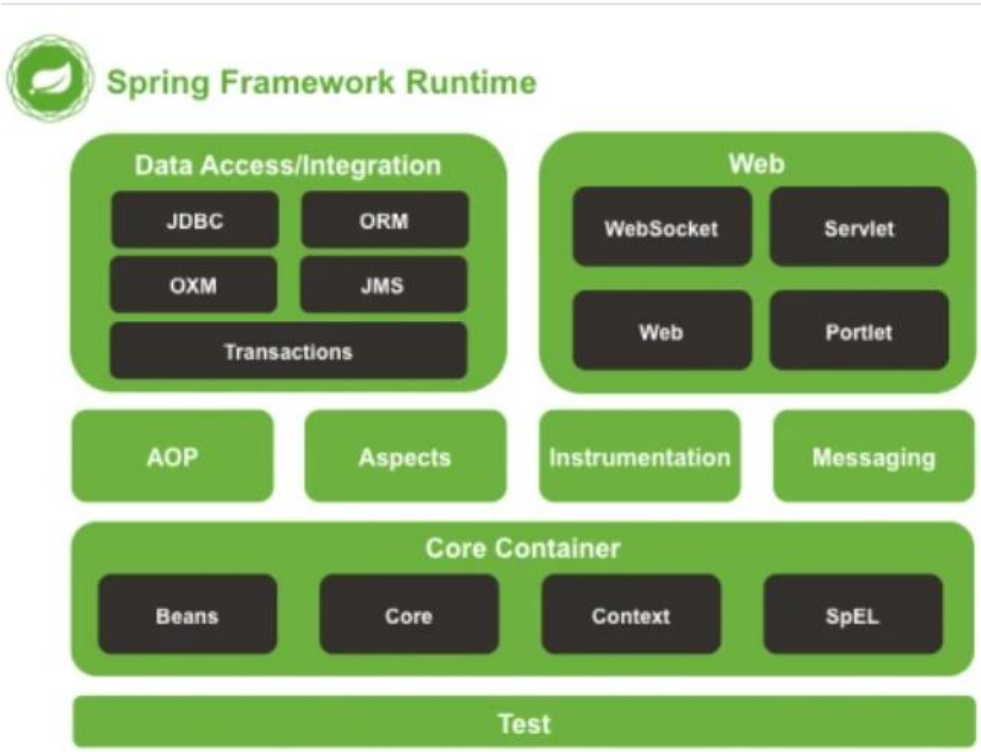

Рисунок 2.1– Структура Spring Framework

Рисунок 2.2– Структура Spring Framework

Як бачимо, пружина має модульну конструкцію. Це дозволяє підключати лише ті модулі, які потрібні для нашої програми, а не ті, які програміст, очевидно, не буде використовувати. Відомо, саме такий підхід допоміг Spring обійти свого тодішнього конкурента (EJB) і зайняти лідерство. Тому що програми, що використовують EJB, потягнули за собою багато залежностей, і загалом вони виявилися повільними і незграбними.

На зображенні видно, що пружинний каркас складається з кількох модулів:

-доступ до даних;

-мережа;

-ядро;

-і інші.

Познайомимося з деякими поняттями основного модуля, такими як: beans, context та інші.

Модуль доступу до даних містить інструменти для роботи з даними (переважно базами даних), web — для роботи в мережі (в тому числі для створення веб-додатків, про які мова піде далі). Крім того, існує також так звана інфраструктура цілої пружини: багато інших проєктів, які офіційно не включені в сам фреймворк, але безперешкодно інтегровані у ваш spring-проєкт.

2.1.3.Spring в Java

Як тільки хоч трохи програміст освоїть Spring, то зрозуміє, скільки різної роботи зараз не треба робити, а скільки Spring бере на себе.

Розробник може написати пару десятків рядків конфігів, написати пару класів - і отримує робочий проєкт. Але як тільки починає думати про те, скільки всього «під капотом», скільки роботи виконується і скільки коду потрібно було б написати, якби робили той самий проєкт на голих сервлетах або на сокетах і чистій Java. Є навіть такий вираз, як «магія» spring. Це коли бачиш, що все працює, але приблизно оцінюєш, скільки всього має статися, щоб все запрацювало і як там все працює - здається, що все це відбувається завдяки

якійсь магії. Легше називати це все магією, ніж намагатися пояснити, як це все там взаємопов’язано.

Другий аргумент «за» вивчення spring полягає в тому, що приблизно в 90% вакансій для молодшого - або знання, або хоча б загальне уявлення про джентльменський набір sping з data , web-mvc і security.

IoC (Inversion of Control) - інверсія контролю.

При використанні бібліотеки потрібно записати у коді, який метод якого об'єкта викликати, а у випадку з фреймворками, найчастіше фреймворк буде викликати код, який розробники написали праворуч. Тобто тут вже не контролюємий процес виконання коду/програми, а фреймворк робить це сам. Коли дали йому контроль (інверсія контролю).

Під DI мають на увазі або інверсію залежностей (інверсія залежностей, тобто намагання не створювати жорсткі зв’язки між вашими модулями / класами, де один клас безпосередньо прив’язаний до іншого), потім ін’єкцію залежності (ін’єкція залежності, це коли програміст не створюєте cat об’єкти в main, а потім ви передаєте їх своїм методам, і пружина створює їх, і просто кажете йому щось на кшталт «Хочу завести тут кота», і він передає це вашому методу).

2.1.4.Біни і контекст

Одне з ключових понять spring – квасоля. Це просто об'єкт якогось класу. Наприклад, для нашої програми потрібно використовувати 3 об’єкти: кішку, собаку і папуг.

І є купа класів з купою методів, де іноді потрібен кіт для методу і собака для іншого методу, а іноді є методи, де потрібні кіт і папуга (наприклад, метод годування кішки), а в деяких методах знадобляться всі три об'єкти.

Так, спочатку потрібно створити ці три об’єкти в основному, а потім передати їх створеним класам, а всередині класів – у потрібних методах... І так протягом всієї програми. І якщо уявляємо, що періодично потрібно змінювати

список прийнятих параметрів для створених методів (треба щось переписати або додати функціональність), то доведеться вносити багато змін у код, якщо потрібно щось змінити. А тепер якщо припустити, що таких об’єктів не 3, а 300?

Як варіант, це зібрати всі існуючі об'єкти в якийсь загальний список об'єктів (List <Object>) і всі методи для його передачі, а зсередини вже методів отримати той чи інший об'єкт, який потрібен. Але що, якщо уявимо, що під час програми можемо додати об’єкт до цього списку або (що гірше) відійти?

Тоді у всіх методах, де отримуємо об’єкти зі списку за їх індексом, все може зламатися.

Тоді вирішуємо зберегти не список, а карту, де ключем буде назва потрібного об'єкта, а значенням - сам об'єкт, і тоді можемо отримати з нього потрібні об'єкти просто за їх ім'ям: get ("папуга") і натомість отримати об'єкт папуга.

Або, наприклад, ключ - це клас об'єкта, а значення - сам об'єкт, тоді вже можемо вказати не назву об'єкта, а просто клас потрібного об'єкта, теж зручно. Або навіть написати над картою якусь обгортку, де зробити методи отримання об'єктів в одних випадках за їх іменами, а в інших випадках - за класами. Це те, що буде робити контекст програми з spring.

Контекст — це набір компонентів (об’єктів). Посилаючись на контекст, можемо отримати потрібний bean (об’єкт) за його назвою, наприклад, або за його типом, або за чимось іншим.

Крім того, можемо попросити саму пружину шукати в її контексті потрібний бін і передати його методу. Коли пружина викликала цей метод, вона передавала йому об’єкт нашого кота зі свого контексту. Тепер вирішили, що окрім кота нашому методу потрібен ще й папуга. Користуючись пружиною - немає нічого простішого, і spring, коли він називає цей метод, він зрозуміє, що тут потрібно передати кота і папугу, він заходить у його контекст, отримує ці два об'єкти і передає їх методу.

Передавши управління новою програмою Spring, можливо також перекласти відповідальність за створення об’єктів і передачу їх нашим методам, які вона буде викликати.

2.1.5.Методи налаштування програми

Існує три основні способи налаштування програми (тобто сказати Spring, з якими об’єктами потрібно працювати):

1) використання файлів/конфігурацій xml;
2) використання конфігурацій Java;
3) автоматичне налаштування.

Розробники Spring ранжують їх у такому порядку пріоритету, найбільш пріоритетним методом, якому слід віддати перевагу, є автоматичне налаштування. Якщо за допомогою автоматичного налаштування неможливо правильно налаштувати всі можливі bean-компоненти, використовуйте конфігурацію java (створення об'єктів за допомогою java-коду).

Найнижчий спосіб – це старомодний спосіб із використанням конфігурацій xml.

Крім того, пружина дозволяє поєднувати ці методи. Наприклад, все, що можна налаштувати автоматично - нехай це зробить сама spring, де потрібно вказати якісь спеціальні параметри - зробіть це за допомогою конфігурацій java, а крім того, можна підключити деякі застарілі конфіги у форматі xml. Загалом, все це можна зробити досить гнучко. Але все ж, якщо все можна зробити за допомогою автоматичної конфігурації, скористайтеся нею.

Розглядатимемо лише авто конфігурацію та конфігурації Java. Конфігурації xml вже використовуються майже в кожному spring прикладі в Інтернеті, і, зрозумівши, як працює конфігурація java, не повинно виникнути проблем, щоб «прочитати» файл xml, який робить те ж саме.

Автоматичне налаштування використовується, коли об’єкти, які потрібні для роботи, є об’єктами класів, які написали.

Якщо для створення об’єкта нашого класу потрібна якась дуже специфічна логіка, або якщо не маємо можливості позначити якийсь клас необхідною анотацією, яка буде підібрана автоматичною конфігурацією, це можна зробити в конфігураціях java.

2.2. Програмне забезпечення для створення коду

2.2.1. Visual Studio Code

Visual Studio Code це редактор відкритого коду, який був розроблений у Microsoft для операційних систем Windows, Linux а також macOS. Редактор позиціонується як простий для редагування коду для кросплатформної розробки веб- додатка. Включає налагодження процесу, інструменти для Git, IntelliSense та інструменти створених для рефакторингу. Він має широкі можливості для налаштування власних тем, комбінації швидких клавіш і файли для конфігурації. Редактор є безкоштовним, програмне забезпечення з відкритим кодом.

Visual Studio Code це редактор відкритого коду. У редактора є багатомовний інтерфейс користувача(Рис.2.3), підсвічування синтаксису для швидкого поняття коду, IntelliSense, налагодження процесу компіляції, зручну навігацію по коду, підтримку Git.

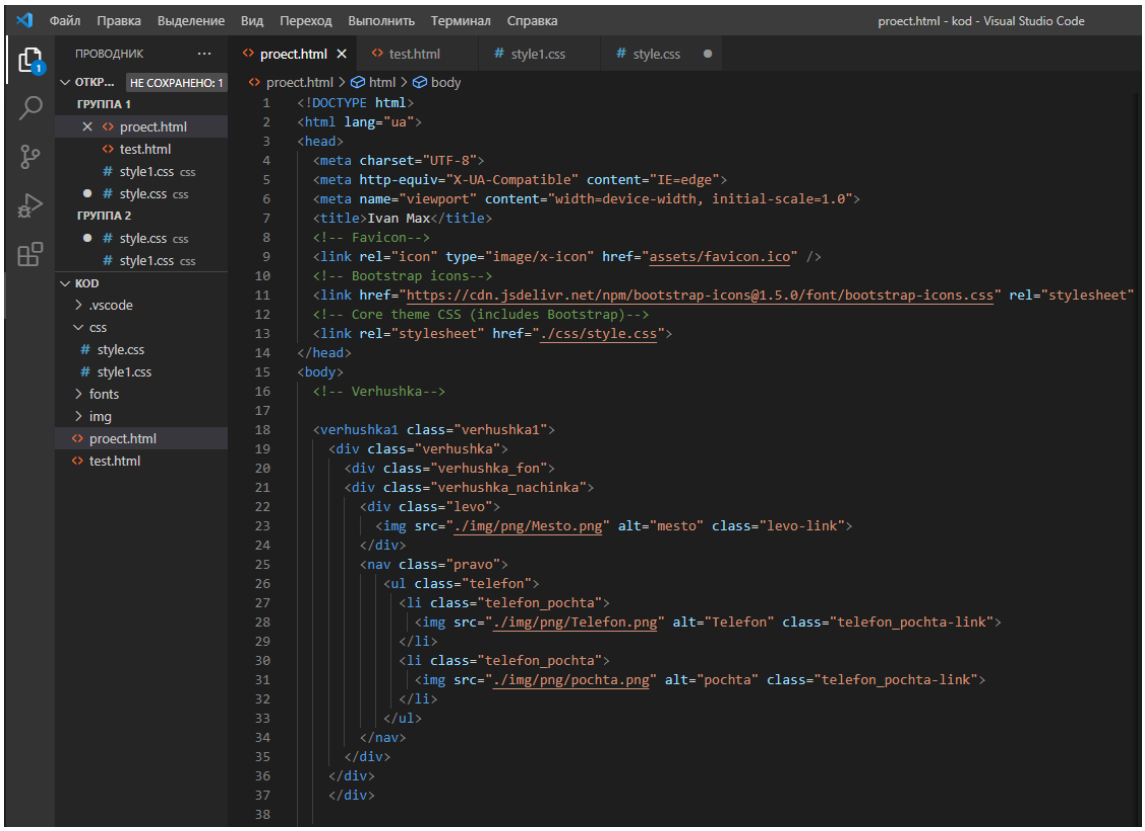

Рисунок 2.2– Інтерфейс Visual Studio Code

Багато функцій VSCode через графічний інтерфейс можуть бути не показаними, часто через палітру команд. Палітра команд походить на рядок з виводом команд, який викликається через комбінацію швидких клавіш.

VSCode дозволяє змінити сторінку кода в момент завантаження документа на носій, так само як і створений рядок і мову програмування документа.

У 2018 з'явилося спеціальне розширення з відкритим легким кодом для VSCode під назвою Python. Розширення надає велику можливість редагування а також тестування з налагодженням коду.

2.2.1.Eclipse

Проєкт Eclipse був спочатку створений IBM у листопаді 2001 року і підтримується консорціумом постачальників програмного забезпечення. Проєктом Eclipse продовжують користуватися мільйони розробників.

Eclipse Foundation був створений у січні 2004 року як незалежна некомерційна корпорація, щоб діяти в якості розпорядника спільноти Eclipse. Незалежна некомерційна корпорація була створена для створення незалежної, відкритої та прозорої спільноти навколо Eclipse.

Спільнота Eclipse складається з окремих розробників та організацій, що працюють у багатьох галузях. Eclipse Foundation фінансується за рахунок щорічних внесків наших членів і керується Радою директорів. Стратегічні розробники та стратегічні споживачі мають місця в цій раді, а також представники, обрані постачальниками додаткових компонентів та комітерами з відкритим кодом. Комітери Eclipse, як правило, працюють в організаціях або є незалежними розробниками, які зголосилися працювати над проєктами Eclipse.

Послуги. Eclipse Foundation надає чотири ключові послуги для спільноти Eclipse:

-управління IP;

-розробка та маркетинг екосистеми;

-процес розробки;

-IT-інфраструктура.

Співробітники підключені до кожної з цих областей і працюють із спільнотою Eclipse, щоб задовольнити потреби зацікавлених сторін.

Унікальна модель розробки з відкритим кодом. Eclipse Foundation був створений для обслуговування проєктів Eclipse з відкритим кодом і спільноти Eclipse. Як незалежна некомерційна корпорація, Eclipse Foundation та модель управління гарантує, що жодна організація не зможе контролювати стратегію, політику або діяльність спільноти Eclipse.

Фонд Eclipse прагне створити середовище для успішних проєктів з відкритим кодом і сприяти впровадженню технології Eclipse у комерційних і відкритих рішеннях. Завдяки таким послугам, як IP Due Diligence, щорічні випуски, підтримка спільноти розробників та розвиток екосистеми, модель розробки з відкритим кодом Eclipse є унікальною та перевіреною моделлю розробки з відкритим кодом.

2.2.3 Figma

Figma одна з програм для редагування графіки. Одна з найголовніших переваг, так це те, що його можна безкоштовно використовувати.

У цьому проєкті саме у Figma будемо створювати початковий дизайн сайту. Будемо використовувати її щоб не витрачаючи години на створення дизайну з нуля в редакторі коду.

Figma — це веб-додаток для редагування графіки та дизайну інтерфейсу користувача. Його можна використовувати для виконання будь-яких видів графічного дизайну, від макетів веб-сайтів, дизайну інтерфейсу мобільних додатків, створення прототипів, публікацій у соціальних мережах і всього, що між ними.

Figma відрізняється від інших інструментів для редагування графіки. В основному тому, що він працює прямо у браузері. Це означає, що можна отримати доступ до своїх проєктів і почати проєктування з будь-якого

комп’ютера або платформи без необхідності купувати декілька ліцензій або встановлювати програмне забезпечення.

Ще одна причина, чому дизайнери використовують цю програму, полягає в тому, що Figma пропонує безкоштовний план, де є можливість створювати та зберігати 3 активних дизайну одночасно. Цього більш ніж достатньо для навчання, експериментів і роботи над невеликими проєктами.

Все, що потрібно, щоб почати використовувати програму, це настільний комп’ютер або ноутбук із хорошим браузером і підключенням до Інтернету.

Потім можна відвідати веб-сайт Figma, щоб зареєструвати безкоштовний обліковий запис. І можна відразу почати працювати над дизайном.

Figma має дуже зручний редактор, де можливо створювати дизайни з нуля або використовуючи готові шаблони. Існує багато ресурсів, щоб навчитися користуватися Figma.

2.3. Висновок

Провівши дослідження та попрацювавши у програмному забезпеченні для написання коду, для створення проєктку використовуємо безкоштовний редактор Eclipse, також буде використано редактор для створення дизайну сайтів та об'єктів Figma. Завдяки цим програмам буде створено початковий дизайн, а потім розпочнеця написання коду.

3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ БАЗИ ДАНИХ РОБОТІВ

3.1. Інформація про інформаційну базу даних та його функціонал

Даний проєкт може підключитися до будь якої бази даних та надати користувачу виводити на екран роботів які знаходяться у самій базі даних. Також користувач має можливість фільтрувати дані які він хоче побачити, головною особливістю цього сайту є адаптивність до бази даних. Тобто підєднавши нову бд або редагувати стару бд, цей проєкт завдяки функціям сам буде будувати вигляд сайту.

3.2. Розробка інформаційної бази даних

3.2.1. Створення дизайна у Figma

Для початку треба створити початковий зовнішній вигляд сайта у Figma, який потім буде змінюватися у процесі створення проєкту. (рис.3.1)

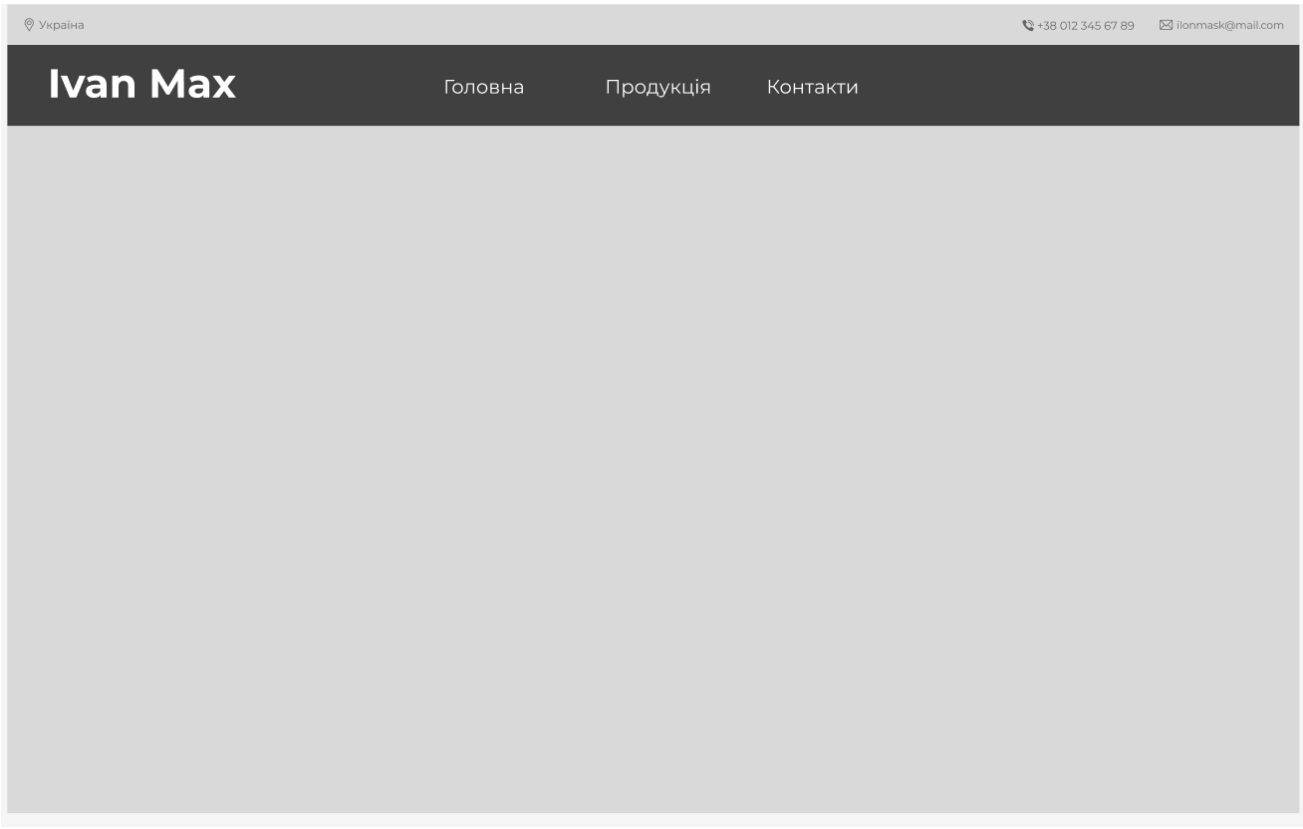

Рисунок 3.1–Початковий дизайн сайту

Таким чином буде прискорено та полегшення написання HTML коду, а також написання css файлів для зовнішнього вигляду та положення компонента на сайті.

Після цього буде дуже просто коригувати код щоб змінився зовнішній вигляд, дивлячись на дизайн сайта та вихідний варіант. Так само, Figma може демонструвати користувачу дані про будь який елемент створений у нему. Тобто можна дуже швидко дізнатися про розмір, стиль, шрифт, колір тексту і тд. (рис.3.2)

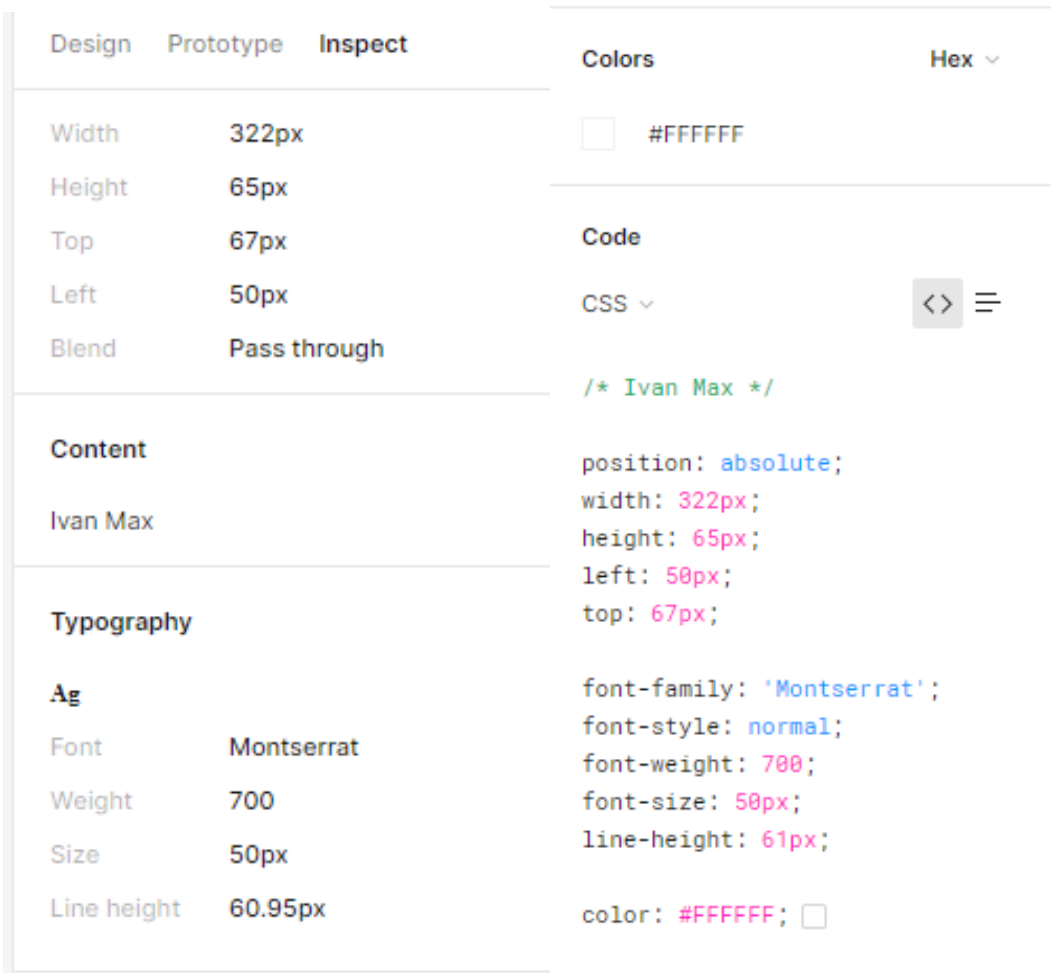

Рисунок 3.2– CSS данні сайту

Після створення дизайну та отримання всіх даних які потрібні для написання css файли, потіс розпочинаємо написання HTML коду використовуючи дані які взяли у редакторі.

3.2.2 Написання HTML коду

Написання та редагування коду робимо у кодовому редакторі Eclipse. Для початку створимо папку для проєкту, призначемо йому назву Store1, після цього треба створити ще декілька папок для того щоб не плутати де и яки файли знаходяться, але їх будемо створювати протягом усього проєкту.(рис.3.3)

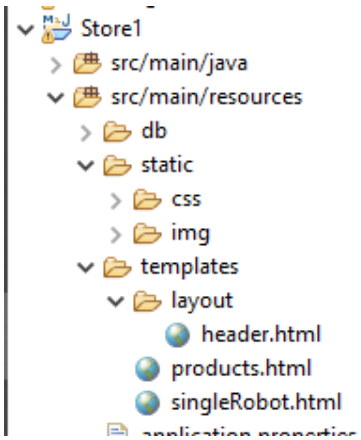

Рисунок 3.3–Стовені папки

Після створення папок додаємо файли з HTML кодом під назвою:

1) header.html;
2) product.html;
3) singleRobot.html.

Та починаємо написання коду.

Спочатку у пустий файл header.html, він являється навігацією по сайту, додаємо основу частину коду, саме head і body: в head підключаємо усі залежності(рис.3.4):

1) bootstrap;
2) JQuery(але потім у процесі розробки від нього відмовимося);
3) css файли.

Далі у body приписуємо клас у якому будуть знаходитися діви у яких тим часом буде написані усі компоненти верхнього навігаційного меню нашого сайту.

Жодної логіки та скриптів у цьому файлі немає. Потім заповнюємо меню, враховуючи назви bootstrap та властивості для подальшої роботи з стилем(рис.3.5)

```
<html xmlns:th="http://www.thymeleaf.org">
<head lang="en">
<script src="/webjars/jquery/3.6.0/jquery.min.js"></script>
<script src="/webjars/bootstrap/4.6.0/js/bootstrap.min.js"></script>
<link rel="stylesheet"
    href="/webjars/bootstrap/4.6.0/css/bootstrap.min.css" />
<script th:src="@{/scrypts/scrypts.js}"></script>
<link th:href="@{/css/style.css}" href="../../static/css/style.css"
    rel="stylesheet" />
```

Рисунок 3.4–Підключення бібліотек

```
  <verhushka1 class="verhushka1">
  <div class="verhushka">
      <div class="verhushka_fon">
          <div class="verhushka_nachinka">
              <div class="levo">
                  <img src="../img/png/Mesto.png" alt="mesto" class="levo-link">
              </div>
              <nav class="pravo">
                  <ul class="telefon">
                      <li class="telefon_pochta"><img src="../img/png/Telefon.png"
                          alt="Telefon" class="telefon_pochta-link"></li>
                      <li class="telefon_pochta"><img src="../img/png/pochta.png"
                          alt="pochta" class="telefon_pochta-link"></li>
                  </ul>
              </nav>
          </div>
      </div>
  </div>

  </verhushka1>

  <!-- Glav_okno-->

  <glav_okno class="glav_okno">
<div class="glav_okno_fon">
  <div class="glav_okno_nachinka">
    <div class="glav_okno_pic">

        <div class="ivan_maks">
          <a href="/" class="ivan_maks">
            <img src="./img/png/Ivan Max.png" alt="ivan" class="ivan_maks-link">
          </a>
        </div>
        <div class="kontakt">
          <a href="#!" class="glava_link">
            <img src="./img/png/Контакти.png" alt="golova" class="glava_link">
          </a>
        </div>

    </div>
  </div>
</div>
</glav_okno>
```

Рисунок 3.5–Заповнення сайту

Потім у файлі product.html використовуємо thymeleaf.

За допомогою thymeleaf підключемо попередній файл як блок у head, це якраз додасть усі підключені скрипти і стилі які написані у пропередньому файли.(рис.3.6)

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
  <head>
    <th:block th:include="layout/header"></th:block>
  </head>
  <body>
    <section class="py-5">
      <div class="row">
        <div class="col-md-4">
          <form id="filter_form" class="row" role="group">
          <div class="btn-group-vertical" role="group">
              <span>Weight</span>
              <input type="radio" class="btn-check" name="Weight" id="min3"
              <label class="btn btn-secondary" for="min3">10-100</label>
              <input type="radio" class="btn-check" name="Weight" id="mid3"
              <label class="btn btn-secondary" for="mid3">100-200</label>
              <input type="radio" class="btn-check" name="Weight" id="max3"
              <label class="btn btn-secondary" for="max3">200+</label>
            </div>
```

Рисунок 3.6–Підключення Thymeleaf

Тут буде два блоки: фильтр та список роботів. Фільтр був створений повністю за допомогою bootstrap і він включає в себе логіку виклику функції(про неї нижче)(рис.3.7) та (рис.3.8), а список роботів буде створений динамічно за допомогою Thymeleaf, якому лише задаємо логіку циклу та порядок виведення полів роботів з бази даних на головний екран. Завдяки цьому при підключені іншої бази даних або після редагування вже підключеної бази даних, не потрібно знову створювати нові HTML та CSS файли, бо завдяки Thymeleaf та функцій буде динамічно створюватися нові списки роботів автоматично.

```
 class="py-5">
class="row">
 class="col-md-4">
orm id="filter_form" class="row" role="group">
iv class="btn-group-vertical" role="group">
  <span>Weight</span>
  <input type="radio" class="btn-check" name="Weight" id="min3" autocomplete="off" value="min">
  <label class="btn btn-secondary" for="min3">10-100</label>
  <input type="radio" class="btn-check" name="Weight" id="mid3" autocomplete="off" value="mid">
  <label class="btn btn-secondary" for="mid3">100-200</label>
  <input type="radio" class="btn-check" name="Weight" id="max3" autocomplete="off" value="max">
  <label class="btn btn-secondary" for="max3">200+</label>
</div>
<div class="btn-group-vertical">
  <span>Capacity</span>
  <input type="radio" class="btn-check" name="Capacity" id="min1" autocomplete="off" value="min">
  <label class="btn btn-secondary" for="min1">3-5</label>
  <input type="radio" class="btn-check" name="Capacity" id="mid1" autocomplete="off" value="mid">
  <label class="btn btn-secondary" for="mid1">6-7</label>
  <input type="radio" class="btn-check" name="Capacity" id="max1" autocomplete="off" value="max">
  <label class="btn btn-secondary" for="max1">20-25</label>
</div>
<div class="btn-group-vertical" role="group">
  <span>Hand Length</span>
  <input type="radio" class="btn-check" name="HandL" id="min2" autocomplete="off" value="min">
  <label class="btn btn-secondary" for="min2">500-1000</label>
  <input type="radio" class="btn-check" name="HandL" id="mid2" autocomplete="off" value="mid">
  <label class="btn btn-secondary" for="mid2">1000-1500</label>
  <input type="radio" class="btn-check" name="HandL" id="max2" autocomplete="off" value="max">
  <label class="btn btn-secondary" for="max2">1500+</label>
</div>
<div class="column">
  <button class="btn btn-primary" type="button" onclick="myFunction()">Submit</button>
</div>
form>
v>
```

Рисунок 3.7– Фукція фільтрації

```
=" container px-4 px-lg-5 mt-5">
ss="row gx-4 gx-lg-5 row-cols-2 row-cols-md-3 row-cols-xl-4 justify-content-center">
h:each="robot : ${robots}" class="col mb-5">
 class="card h-100">
-- Product image-->
mg class="card-img-top" th:src="${robot.image}" alt="..." />
-- Product details-->
iv class="card-body p-4">
<div class="text-center">
  <h4 class="fw-bolder" th:text="${robot.firm}">Fancy Product</h4>
  <h5 class="fw-bolder" th:text="${robot.name}">Fancy Product</h5>
  <div>
    <span th:text="'Load Capacity:'+${robot.loadCapacity}">loadCapacity </span>
  </div>
  <div>
    <span th:text="'Hand Length:'+${robot.handHength}">loadCapacity </span>
  </div>
  <div>
    <span th:text="'Accuracy:'+${robot.accuracy}">loadCapacity </span>
  </div>
  <div>
    <span th:if="${robot.weight!=null}" th:text="'Weight:' + ${robot.weight}">weight </span>
  </div>
  <div>
    <span th:if="${robot.weight==null}">Weight: no info </span>
  </div>
</div>
<div class="card-footer p-4 pt-0 border-top-0 bg-transparent">
  <div class="text-center">
    <a class="btn btn-outline-dark mt-auto" th:href="@{/products/{id}(id=${robot.id})}">View options</a>
  </div>
```

Рисунок 3.8– Фукція фільтрації

Сюди також додаємо деякі функції які зчитують натиснути кнопки для фільтрації даних, тобто натиснувши на кнопку з параметрами робота які потрібно витягнути з бази даних, та шоб вони відображалися на екрані.(рис.3.9)

```
  <!-- Core theme JS-->
  <script src="js/scripts.js"></script>
</body>
<script>
    function myFunction() {
        var options1 = document.getElementsByName('Weight');
        var options2 = document.getElementsByName('Capacity');
        var options3 = document.getElementsByName('HandL');
        var selectedWeight;
        var selectedCapacity;
        var selectedHand;

        for(var i = 0; i < options1.length; i++) {
           if(options1[i].checked)
               selectedWeight = options1[i].value;
         }
        for(var i = 0; i < options2.length; i++) {
           if(options2[i].checked)
              selectedCapacity = options2[i].value;
         }
        for(var i = 0; i < options3.length; i++) {
           if(options3[i].checked)
              selectedHand = options3[i].value;
         }

        var urlParams = new URLSearchParams(window.location.search);

        urlParams.delete('minWeight');
        urlParams.delete('maxWeight');
        urlParams.delete('minHand');
        urlParams.delete('maxHand');
        urlParams.delete('minCapacity');
        urlParams.delete('maxCapacity');
```

Рисунок 3.9– Компонент фільтрації

У файлі singleRobot.html буде знаходитися лише один блок зі списком одного робота.(рис.3.10)

```
ainer-fluid">
row">
ss="col-md-2"></div>
ss="col-md-8">
 class="text-center">
<div>
    <h4 class="fw-bolder" th:text="${robot.firm}">Fancy Product</h4>
    <h5 class="fw-bolder" th:text="${robot.name}">Fancy Product</h5>
</div>
<table class="table table-hover table-bordered">

    <thead class="thead-dark">
        <tr>
            <th>Load Capacity:</th>
            <th>Hand Length:</th>
            <th>Accuracy:</th>
            <th>Weight:</th>
            <th>Img</th>
            <th>Angles</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td class="propertyValue" th:text="${robot.loadCapacity}">
                Value</td>
            <td class="propertyValue" th:text="${robot.handHength}">
                Value</td>
            <td class="propertyValue" th:text="${robot.accuracy}">
                Value</td>
            <td class="propertyValue" th:text="${robot.weight}">Value</td>
            <td><img th:src="${robot.image}" class="image-in-table" /></td>
            <td><table class="table table-hover table-bordered">
                    <thead class="thead-dark">
                        <tr>
                            <th>Shaft</th>
                            <th>Degree</th>
                            <th>Speed</th>
                        </tr>
                    </thead>
                    <tbody>
                        <tr th:each="shaft : ${robot.shafts}">
                            <th class="globalProperty" th:text="${shaft.shaftName}">Name<
                            <td class="propertyValue" th:text="${shaft.shaftDegree}+'°'">
                                Value</td>
                            <td class="propertyValue"
                                th:text="${shaft.shaftSpeed}+' mm/s'">Value</td>
                        </tr>
                    </tbody>
                </table></td>
```

Рисунок 3.10– Блок сайту

3.2.3. Створення контролерів

Проєкт написан за допомогою Spring, тому зв’язок між сайтом та бек-ендом майже не конфігуруємо, це робить ModelWebControl- одна з частин Sping.(рис.3.11)

```
25          <dependency>
26              <groupId>org.springframework.boot</groupId>
27              <artifactId>spring-boot-starter-thymeleaf</artifactId>
28          </dependency>
29          <dependency>
30              <groupId>org.springframework.boot</groupId>
31              <artifactId>spring-boot-starter-validation</artifactId>
32          </dependency>
33          <dependency>
34              <groupId>org.springframework.boot</groupId>
35              <artifactId>spring-boot-starter-web</artifactId>
36          </dependency>
```

Рисунок 3.11– Зв'язок між бек-ендом

Контроллер це частина проєкту, яка забезпечує обробку реквестів та підготування сторінки. У Model через атрибути передаємо бек-енді структури(Список роботів для основної сторінки та одного робота для сторінки робота), якимі потім обрисовує Thymeleaf, (рис.3.12)

```
@RequestMapping(value = { "/products", "/" }, method = RequestMethod.GET)
public String getAllCategories(Model model, @RequestParam(name = "minWeight", required =
        @RequestParam(name = "maxWeight", required = false) Integer maxWeight,
        @RequestParam(name = "minHand", required = false) Integer minHand,
        @RequestParam(name = "maxHand", required = false) Integer maxHand,
        @RequestParam(name = "minCapacity", required = false) Integer minAbuility,
        @RequestParam(name = "maxCapacity", required = false) Integer maxAbuility) {
    logger.info("Html page= /categories");
    List<Robot> robots = productService.findAll();// Находит в бд существующие локации
    robots = filterWeight(robots, minWeight, maxWeight);
    robots = filterHand(robots, minHand, maxHand);
    robots = filterAbuility(robots, minAbuility, maxAbuility);
    model.addAttribute("robots", robots);// добавляет их на страницу
    return "products";
```

Рисунок 3.12–Атрибути

Через скрипти на сайті формуємо фільтрацію, яка буде виводити потрібних роботів на головний екран, яка також відбувається на шарі (зараз маємо тришаровий проєкт+інтерфейс) контроллеру. Сам класс контроллеру за допомогою анотації @Controller є компонентом спрінгу, тому якихось інших маніпуляцій крім завдання логіки не робимо.

Інші анотації відповідають за контроль реквестів на сторінки і визначають шлях, який контролює той чи інший метод контроллеру та його вхідні дані:

1) @RequestMapping(value=”someValue”, method=RequestMethod.GET)- відповідають за шлях, який контролює метод, де @RequestMapping- анотація, яка визначає роль у Спрінг-проєкті, value- шлях, який контролюється методом, а method- HTTP метод запиту(реквесту)(рис.3.13);
2) @RequestParam та @PathVariable є частинами шляху, які зчитуються з реквесту та передаються в метод як параметри.

```
}

@RequestMapping(value = "/products/{id}", method = RequestMethod.GET)
public String getModelById(Model model, @PathVariable(name = "id") Integer id) {
    logger.info("Html page= /categories");
    Robot robot = productService.find(id).get();// Находит в бд существующего робота
    model.addAttribute("robot", robot);// добавляет его на страницу
    return "singleRobot";
}

private List<Robot> filterWeight(List<Robot> robots, Integer min, Integer max) {
    if (min != null)
```

Рисунок 3.13–Функція Request

Головна різниця між ними це @RequestParam є невід’ємною частиною шляху, який задається в value анотації @RequestMapping, коли @PathVariable це частина аргументів, які задаються як аргументи у шляху.(рис.3.14)

```
@RequestMapping(value = { "/products", "/" }, method = RequestMethod.GET)
public String getAllCategories(Model model, @RequestParam(name = "minWeight", required =
        @RequestParam(name = "maxWeight", required = false) Integer maxWeight,
        @RequestParam(name = "minHand", required = false) Integer minHand,
        @RequestParam(name = "maxHand", required = false) Integer maxHand,
        @RequestParam(name = "minCapacity", required = false) Integer minAbuility,
        @RequestParam(name = "maxCapacity", required = false) Integer maxAbuility) {
    logger.info("Html page= /categories");
    List<Robot> robots = productService.findAll();// Находит в бд существующие локации
    robots = filterWeight(robots, minWeight, maxWeight);
    robots = filterHand(robots, minHand, maxHand);
    robots = filterAbuility(robots, minAbuility, maxAbuility);
    model.addAttribute("robots", robots);// добавляет их на страницу
    return "products";
```

Рисунок 3.14–RequestParam

Такі приклади ви бачили на будь-яких сайтах з фільтрацією, де у шляху змінюються лише аргументи після “?”:

Наприклад:…someStore/categories/someCategory?filter=weight&maxWeight=100,де filter та maxWeight є аргументами

Окрім цих аргументів, метод контроллеру приймає Model- об’єкт, за допомогою якого додаємо об’єкти з бек-енду на сторінку Model.addAttribute (name, object) додає на сторінку об’єкт object і дає йому назву name, щоб потім за допомогою Thymeleaf показав цей об’єкт користувачу.(рис.3.15)

```
private List<Robot> filterWeight(List<Robot> robots, Integer min, Integer max) {
    if (min != null)
        robots = robots.stream().filter(robot -> robot.getWeight() != null).filter(robot -> robot.getWeight()
                .collect(Collectors.toList());
    if (max != null)
        robots = robots.stream().filter(robot -> robot.getWeight() != null)
                .filter(robot -> robot.getWeight() != null || robot.getWeight() < max).collect(Collectors.toL
    return robots;
}

private List<Robot> filterHand(List<Robot> robots, Integer min, Integer max) {
    if (min != null)
        robots = robots.stream().filter(robot -> robot.getHandHength() > min).collect(Collectors.toList());
    if (max != null)
        robots = robots.stream().filter(robot -> robot.getHandHength() < max).collect(Collectors.toList());
    return robots;
}

private List<Robot> filterAbuility(List<Robot> robots, Integer min, Integer max) {
    if (min != null)
        robots = robots.stream().filter(robot -> robot.getLoadCapacity() > min).collect(Collectors.toList());
    if (max != null)
        robots = robots.stream().filter(robot -> robot.getLoadCapacity() < max).collect(Collectors.toList());
    return robots;
}
```

Рисунок 3.15–Thymeleaf

Функція контроллеру має повертати назву файлу сторінки коли будемо повертатися на головний екран, або вибирати інші фільтри після попереднього вибору фільтрів. Повертатися будемо у файл який створили раніше в resources/templates.(Рис.3.15.1)

```
        return "products";
    }
```

Рисунок.3.15.1-Команда повернення

3.2.4 Написання сервісів

Другим шаром проєкту є сервіс- шар, у якому виконується основна логіка всього проєкту: саме тут мають проводитись всі операції з сухими даними з репозиторію.

Тому що репозиторій не фільтрує та не приводить у гарний вигляд дани які були витягнути з бази даних.

Спочатку створимо простий інтерфейс ModelService з 4 методами:

-save;

-read;

-readAll;

-delete.

Основними операціями, якими має користуватись юзер(адмін), від якого потім і будемо відштовхуватись при створенні сервісу виключно для роботів(рис.3.16) та (рис.3.17)

Взагалі, існує архітектура CRUD для роботи з базами даних, але у цій версії проєкту не будемо створювати або видаляти дані з бази.

Також на додатку 3 можливо побачити як саме дани дистаються з бази даних та фільтруються у сервісах.

```java
    @Override
    @Transactional
    public Robot save(Robot product) {
        Robot saved = null;
        try {
            saved = repository.save(product);
            logger.info("Product {} saved", saved.toString());
            return saved;
        } catch (Exception e) {
            logger.error(e.toString());
        }
        return saved;
    }

    @Override
    @Transactional
    public Optional<Robot> find(int id) {
        Optional<Robot> founded = null;
        if (logger.isDebugEnabled())
            logger.debug("Searching product with id {}", id);
        founded = repository.findById(id);
        if (founded.isPresent())
            logger.info("Product with id {} was found successfull", id);
        else
            logger.warn("Product with id {} doesnt exsist", id);
        return founded;
    }

    @Override
    @Transactional
    public List<Robot> findAll() {
        List<Robot> list = null;
        if (logger.isDebugEnabled())
            logger.debug("Searching all products");
        list = repository.findAll();
        logger.info("Product list was found");
        return list;
    }

    @Override
    @Transactional
    public boolean delete(int id) {
        if (logger.isDebugEnabled())
            logger.debug("Deleting product with id {}", id);
        try {
            repository.deleteById(id);
            logger.info("Product with id={} was deleted", id);
            return true;
        } catch (Exception e) {
            logger.warn(e.toString());
            return false;
```

Рисунок 3.16–Сервіс роботів

```java

    @Override
    @Transactional
    public Shafts save(Shafts property) {
        Shafts saved = null;
        try {
            saved = shaftRepository.save(property);
            logger.info("Property {} saved", saved.toString());
            return saved;
        } catch (Exception e) {
            logger.error(e.toString());
        }
        return saved;
    }

    @Override
    @Transactional
    public Optional<Shafts> find(int id) {
        Optional<Shafts> founded = null;
        if (logger.isDebugEnabled())
            logger.debug("Searching property with id {}", id);
        founded = shaftRepository.findById(id);
        if (founded.isPresent())
            logger.info("Property with id {} was found successfull", id);
        else
            logger.warn("Property with id {} doesnt exsist", id);
        return founded;
    }

    @Override
    @Transactional
    public List<Shafts> findAll() {
        List<Shafts> list = null;
        if (logger.isDebugEnabled())
            logger.debug("Searching all products");
        list = shaftRepository.findAll();
        logger.info("Property list was found");
        return list;
    }

    @Override
    @Transactional
    public boolean delete(int id) {
        if (logger.isDebugEnabled())
            logger.debug("Deleting property with id {}", id);
        try {
            shaftRepository.deleteById(id);
            logger.info("Property with id={} was deleted", id);
            return true;
        } catch (Exception e) {
            logger.warn(e.toString());
            return false;
        }
    }
```

Рисунок 3.17–Сервіс роботів

@Service- анотація спрінгу.(рис.3.18) та (рис.3.19)

```
@Service
public class RobotService implements ModelService<Robot> {

    private final RobotRepository repository;

    private static final Logger logger = LoggerFactory.getLogger(RobotService.class.getName());

    public RobotService(RobotRepository repository) {
        this.repository = repository;
    }
```

Рисунок 3.18–Анотація спрінгу

```
@Service
public class ShaftsService implements ModelService<Shafts> {

    private final ShaftRepository shaftRepository;

    private static final Logger logger = LoggerFactory.getLogger(ShaftRepository.class.getName());

    public ShaftsService(ShaftRepository repository) {
        this.shaftRepository = repository;
    }
```

Рисунок 3.19–Анотація спрінгу

@Transactional- анотація, яка відповідає за створення транзакцій: Кращим прикладом правильної транзакції є банк: коли одна людина переводить інший гроші- операція не буде успішною, поки обидва кроки не будуть виповнені- операція не буде успішною.

Відкрити транзакцію:

1) зняти гроші у відправника;
2) додати гроші до отримувача;
3) закрити транзакцію.

Ця анотація відповідає за правильність виконання операцій і відкат змін, якщо хоча б один крок не був успішним, у проєкті вона використовується без параметрів і спеціалізацій, але при виконанні складних операцій вона буде корисною.

Складної логіки тут немає, тому методи зводяться до логування операції та виклику репозиторія.

3.2.5. Створення репозиторію

Проєкт зроблений за допомогою Sprig, тому зв’язок програми та бази даних не потрібно писати за допомогою JDBC та інших винаходів колес, це зробить Hibernate та JPA, де JPA- правила поведінки з базою, а Hibernate- реалізація цих правил і додаткові можливості.(рис.3.20) та (рис.3.21)

```
@Repository
public interface RobotRepository extends JpaRepository<Robot, Integer> {

    public Optional<Robot> findByName(String name);
```

Рисунок 3.20–реалізація правил

```
package com.jabascript.warehouse.repositorys;

import org.springframework.data.jpa.repository.JpaRepository;

@Repository
public interface ShaftRepository extends JpaRepository<Shafts, Integer> {
```

Рисунок 3.21–реалізація правил

Почнемо з об’єктів в базі. Робот та його кути- це прості POJO(Pure Old Java Objects), які мають лише філди, конструктор та Геттери-Сеттери. Java є повністю об’єктно-орієнтованою мовою, тому зв’язок між цима об’єктами є основою їх відносин.

Почнемо з простих анотацій:

1) @Entity, @Table- анотації які визначають те, що цей клас є об'єктом в базі даних і знаходиться в якійсь таблиці;
2) @Id, @GeneratedValue- анотації, які відповідають за визначення поля ідентифікатора та стратегії його генерації;
3) @Column-відповідає за відповідність поля в джава-об’єкті та колонки в базі.

Відношення між об’єктами та їх частинами є декілька типів відносин, і які б толерантні не були розробники, тут маємо використовувати саме @OneToMany в роботі, бо в робота є декілька кутів, а в одного окремого плеча є лише один робот, тому кажемо Hibernate про те, що при записі/видаленні робота, його плечі не можуть існувати без нього, і при цьому у кожного плеча є лише один робот, і при витягуванні його з бази даних, з ним йдуть і його плечі

Зі сторони плечей це робиться за допомогою @ManyToOne та @JoinColumn, остання з яких і каже про відповідність між полем відношення Shaft.Robot_Id та ідентифікатором робота Robots.Id. Сам репозиторій буде екстендити інтерфейс JpaRepository<T, ID>, де T- Об’єкт Entity, а ID-тип його ідентифікатора в базі. Сам інтерфейс дозволяє використовувати готові методи типу:

1) CRUD;
2) findAll;
3) findById.

Навіть за допомогою полів Entity створювати методи лише за назвою, і Hibernate сам, базуючись лише на назві методу, створить SQL запити, на прикладі findByName(String name) це можна побачити, створиться запит типу “Select * from Robots …join… where Robots.name=name”

Репозиторій це третій і останній шар, який такоє є компонентом Spring

3.2.6 Написання бази даних

Ані Java, ані Spring з коробки не підтримують SqLite бази даних, тому спочатку підключимо файл з Діалектом(відношенням між типами бази та типами Java Також у Додатку 1 можно побачити характеристики роботів які є в цій базі даних. Підключимо залежності в мавен.

Потім, у файлі application.properties-головному конфігураційному файлі програми, в якому лише 20 строк(рис.3.22)

У строки потрібно прописати:

-діалект;

-тип бази даних;

-юзернейм;

-пароль.

```
spring.h2.console.enabled=true
spring.jpa.show-sql=false

server.port=8081
server.error.include-stacktrace=always


spring.jpa.database-platform=com.springboot.sqlite.SQLDialect
spring.datasource.url = jdbc:sqlite:sqlite.db
spring.datasource.driver-class-name = org.sqlite.JDBC
spring.datasource.username=user
spring.datasource.password=password

logging.level.root=info;
logging.pattern.console=%d{HH:mm:ss.SSS} [%-5p][%t][%-32.32c] - %m%n

logging.file.name=logging.log

spring.jpa.hibernate.ddl-auto=none
```

Рисунок 3.22–Конфігурацяи

На цьому все: Spring сам запустить і підв’яже єдину базу даних.

Flyway- інструмент міграції бази данних, який і відповідає у за створення та заповнення бази даних. Він працює на міграціях, в яких у Schema задали схему таблиць бази, а в Data- заповнили таблиці значеннями(рис.3.23) та (рис.3.24)

```
spring.flyway.skip-default-callbacks=true
spring.flyway.baseline-on-migrate=true
spring.flyway.baselineVersion=0
```

Рисунок 3.23–Налаштування Flyway

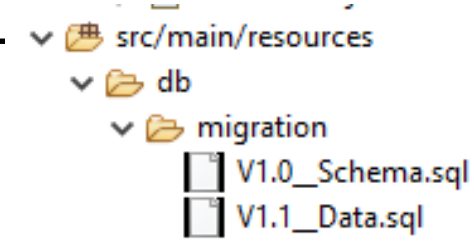

Рисунок 3.24–База даних

В цій базі даних знаходяться характеристики 12 роботів які потрібно буде витягнути з неї та направити ці дані до модулів з функціями для фільтрування. Також ці дані просдуть с початку через репозиторій, а потім через сервіси, тому що Spring та Java не працюють на пряму з базами даних.

З бази даних репозиторій витягує сирі дані, а саме:

-вантажопідйомність;

-число осей;

-довжина руки;

-точність позиціону-вання;

-кут повороту;

-швидкість повороту;

-вага.

Репозиторій їх витягує з бази даних, еле вони ніяк не сформовані та наразі це просто купа даних. Тому ці дані далі направляємо у сервіси де вони формують таблиці та фільтрують дані по таблицям.

Так само сервіси виділяють ті дані які вказано у файли з функціями контролерів:

-вага;

-вантажопідйомність;

-довжина руки;

-кут повороту.

3.2.7 Підключення усіх компонентів

Оскільки проєкт написан на Spring, та має в собі Мавен тому, задля запуску і зв'язування всіх компонентів потрібно лише три речі:

1) JRE- віртуальна машина джави і девелоперський кіт, щоб заранити це все;
2) доступ до інтернету, щоб Мавен зміг скачати і підключити потрібні версії залежностей;
3) хоча-б один палець, щоб запустити програму і зайти на порт 8080, де готовий сайт і буде працювати.

3.3 Перевірка роботи інформаційної бази даних

Спочатку у редакторі Eclipse треба запустити проєкт та відкрити його у будь якому браузері за посиланням http://localhost:8081/products

Таким чином потрапляемо на головну сторинку де демонструються усі види роботів які є в базі даних(рис.3.25)

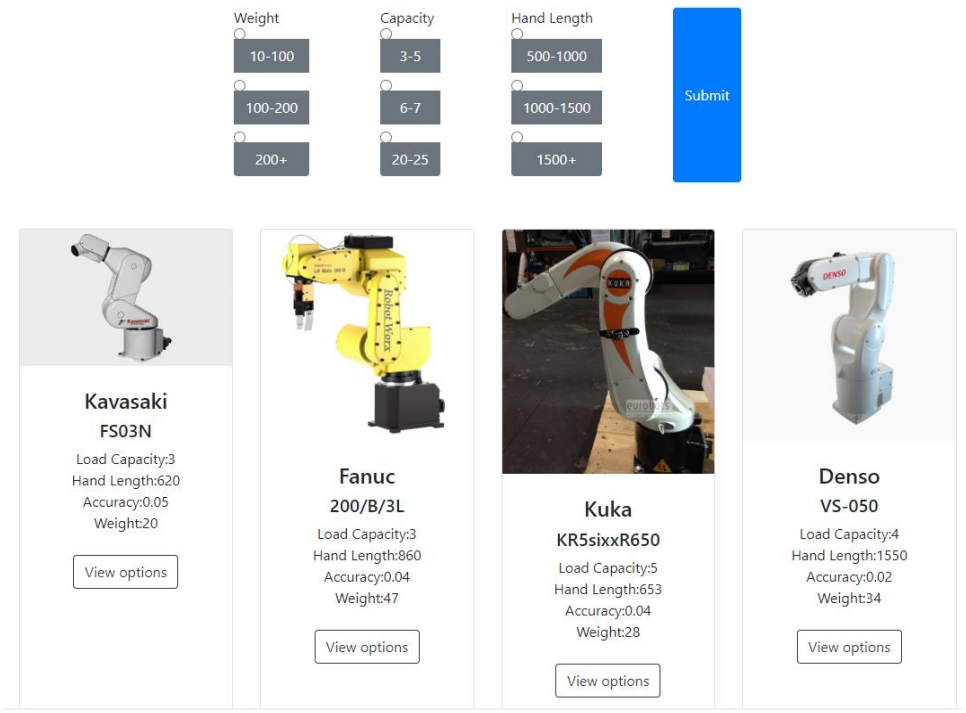

Рисунок 3.25–Головна сторінка

Якщо проскролить вниз то побачимо інші моделі які присутні (рис.3.26)

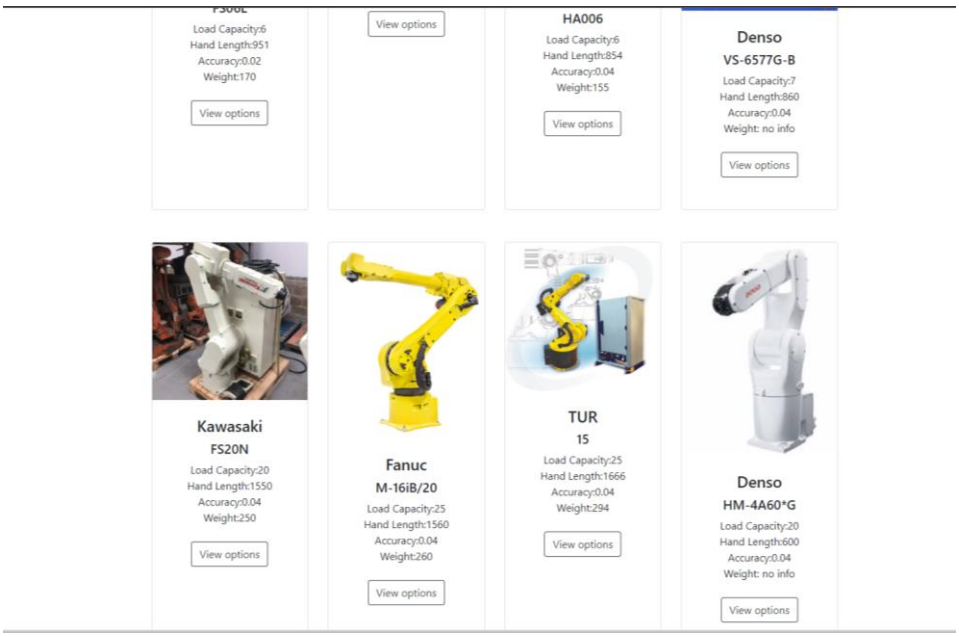

Рисунок 3.26–Моделі роботів

Спробуємо відкрити сторінку з роботом де буде демонструватися його фотографія та повна інформація про нього(рис.3.27) Після цього потрібно перевірити яка працює фільтрація, тому треба повернутися на головну сторінку натиснувши на кнопку Ivan Max. Знову знаходимося на головній сторінці, почнемо перевіряти фільтри.

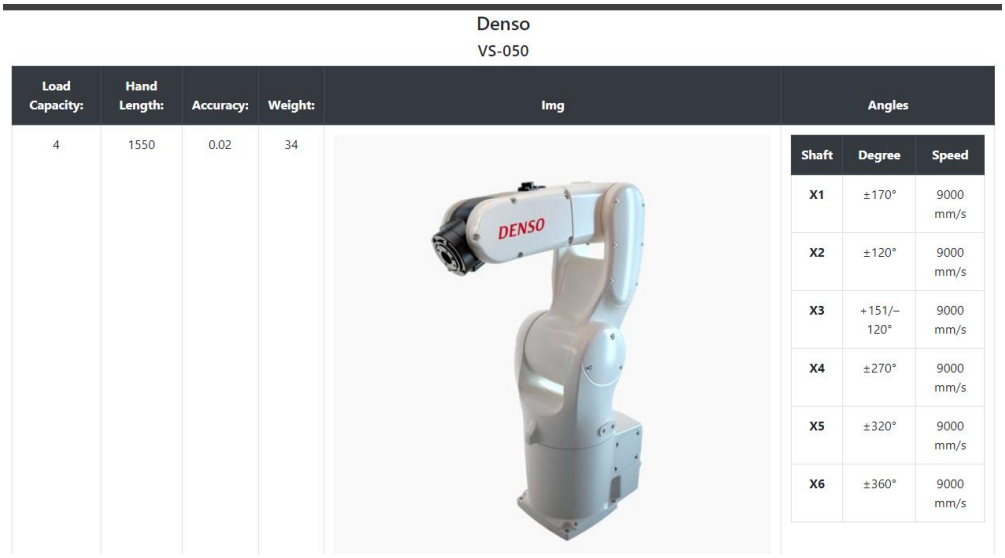

Рисунок 3.27–Модель робота

Наприклад потрібен робот який важить більше двох сотень кілограм, може піднімати двадцять три кілограма та довжина його руки була більша за півтора метра(рис.3.28)

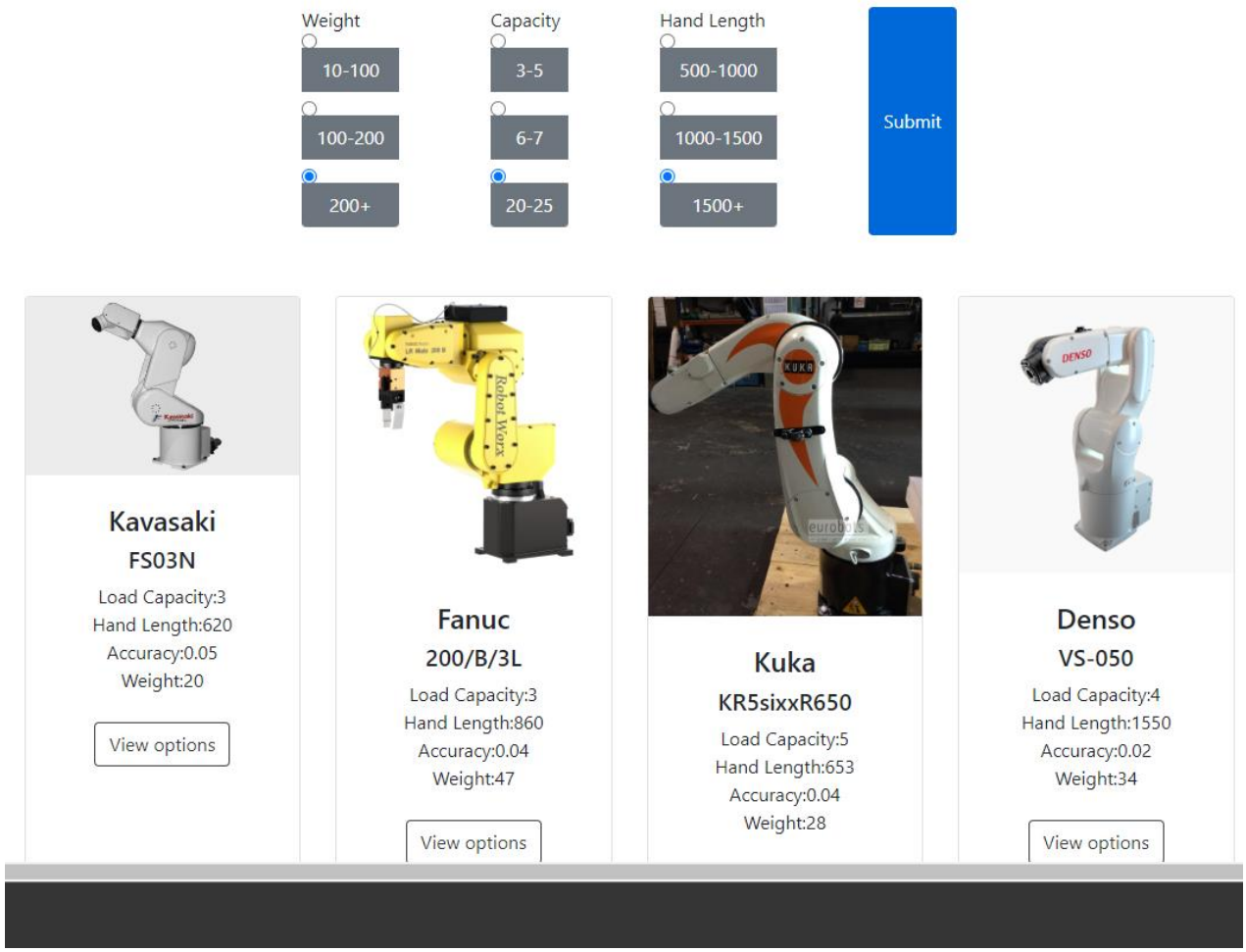

Рисунок 3.28–Фильтрація роботів за параметрами

Ось такий результат отримали після того як вибрали усі потрібні характеристики та натиснули на кнопку субміт(рис.3.29)

Так само перевірив інші фільтри, також якщо задати у фильтрі критерії робота якого немає в базі даних отримаємо просто пусту сторінку з можливістю повернутися на головну сторінку або знову завдатися до фільтра(рис.3.30)

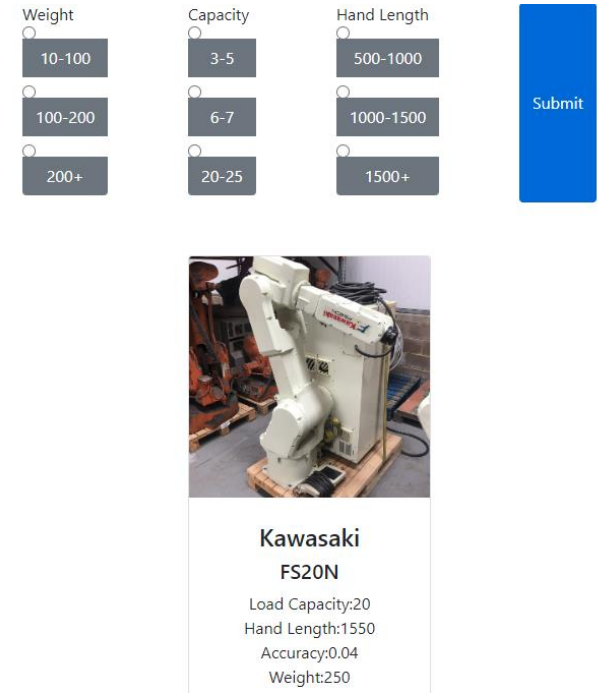

Рисунок 3.29–Вивід робота після фільтрації

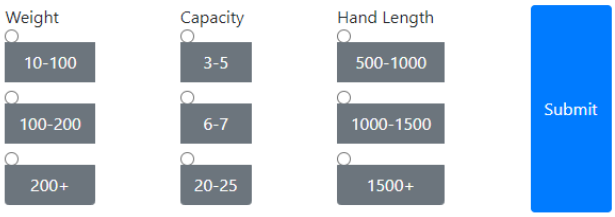

Рисунок 3.30–Роботів не було знайдено

3.4. Висновок

Використовуючи редактор Eclipse та фреймворк Spring, створили проєкт пошукову базу даних промислових роботів у якій користувач може фільтрувати роботів за їхніми можливостями та бачити характеристики кожгого робота окремо. Таким чином була виконана основна задача у створені пошукової бази даних промислових роботів для АРМ проєктувальника робототехнічних систем яку можливо дуже легко редагувати або модифікувати підєднавши будь яку базу даних роботів з їхніми характеристики. На функціональному додатку 2 та 3 можна побачити структуру роботи проєкту.

ВИСНОВКИ

Було створено пошукову базу даних промислових роботів для АРМ проєктувальника робототехнічних систем, використовуючи фреймворк Spring.

У першому розділі зробили огляд типів інформаційних баз даних та створення інформаційних баз даних. Завдяки цьому здобули теоретичні знання для написання проєкту.

У другому розділі детально розібралися у програмному забезпеченні для редагування коду, а також дізналися що таке фреймворк Spring. Завдяки цьому здобули знання як прискорити та модернізувати написання коду для проєкту.

У третьому розділі почали розробку проєкту у редакторі коду Eclipse та на основі фреймворку Spring. Завдяки минулим розділам створити проєкт пошукову базу даних з можливістю його редагувати або модифікувати.

ПЕРЕЛИК ПОСИЛАНЬ

1) Офіційний сайт Eclipse // Eclipse. URL: https://www.eclipse.org/
2) Скальський В.Р. Становлення методу акустичної емісії в установах Західного наукового центру. Теорія і практика раціонального проєктування, виготовлення і експлуатації машинобудівельних конструкцій: праці 2 міжнар. наук.-техн. конф. (Львів, 11–13 листопада 2010 р.). Львів, 2010. С. 9–10.
3) Офіційний сайт Spring // Spring. URL: https://spring.io/
4) Ресурсосбережение и энергоэффективность инженерной инфраструктуры урбанизированных территорий и промышленных предприятий: материалы II Международной научно-технической интернет-конференции (2–27февраля 2016 г., Харьков). Харьков, 2016. 150 с
5) Офіційний сайт Visual Studio Code // Visual Studio Code. URL: https://code.visualstudio.com/
6) Наукові публікації і видавнича діяльність НАН України. Київ, 2007. URL: http://www.nas.gov.ua/publications (дата звернення: 19.03.2014).
7) Офіційний сайт Bootstrap // Bootstrap. URL: https://getbootstrap.com/
8) Alex d. Development agent system of the neural web-network diagnostics or remoting monitoring all of patients. European Journal of Enterprise system Technologies. 2018.
9) Офіційний сайт Figma // Figma. URL: https://www.figma.com/
10) Офіційний сайт Java // Java. URL: https://www.java.com/