ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра комп‘ютерних наук

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА WEB-ДОДАТКУ ПОШУКУ ОПТИМАЛЬНОЇ КОНФІГУРАЦІЇ МЕРЕЖЕВОГО ОБЛАДНАННЯ МОВОЮ JAVA**»

Виконав: студент 4 курсу, групи КНД-42

спеціальності

122 Комп’ютерні науки

(шифр і назва спеціальності)

Гончаренко О.І.

(прізвище та ініціали)

Керівник _____

(прізвище та ініціали)

Рецензент _____

(прізвище та ініціали)

Київ – 2021

ВСТУП

В наш час науково-технічного прогресу в основі вирішення багатьох проблем лежить обробка інформації, яка в свою чергу, залежить від рівня інформатизації суспільства. Забезпечення вільного доступу до інформаційних ресурсів формується розробкою надійної та добре розвинутої мережної інфраструктури. Пропускна спроможність та надійність в її роботі напряму залежить від вибору обладнання і воно стає все більш затребуваним і необхідним. В зв'язку з тим, що швидкий розвиток технологій провокує виробників на розробку та виготовлення все більш різноманітних моделей мережевого обладнання з різною конфігурацією та форм-фактором заплутатися в характеристиках моделей доволі легко – схожі пристрої можуть відрізнятися параметрами. Тому при виборі мережевого обладнання необхідно чітко і ясно розуміти задачі для рішення яких воно буде використовуватися.

Для удосконалення більш оптимального вибору мережевого обладнання, з урахуванням всіх його характеристик та виробників споживачі зацікавлені в постійному оновленні і апробаціях різних додатків, платформ, які б найбільш задовольняли їх поставленим цілям. Тому створення веб-додатку для пошуку оптимальної конфігурації мережного обладнання є досить актуальним.

Розробка веб-додатку передбачає під собою реалізацію наступних задач:

— вивчити архітектуру клієнт-серверних додатків та принцип роботи і методи протоколу HTTP;
— ознайомитися з актуальними технологіями розробки web-додатків;
— вибір платформи або мови програмування, яка буде використовуватися для реалізації бізнес логіки на сервері;
— вибір системи керування базами даних, що відповідає усім потребам для втілення необхідного функціоналу;
— розробка бази даних;

— проектування та розробка UI клієнтської частини додатку;
— розробка web-додатку з функціоналом пошуку обладнання по заданим критеріям.

Об’єктом дослідження є пошук оптимальної конфігурації мережного обладнання.

Предмет дослідження – розробка web-додатку пошуку оптимальної конфігурації мережного обладнання.

Мета – розробка веб-додатку для інформаційного забезпечення користувачів у підборі мережного обладнання потрібної конфігурації та закріплення і систематизація теоретичних знань набутих в процесі навчання на його прикладі.

Для розробки додатку будуть використані можливості платформи Java, в якості системи керування базами даних буде використовуватися реляційна система MySQL, для клієнтської частини будуть використовуватися типові технології такі, як: HTML, CSS, JavaScript.

Доступ до бази даних буде реалізований за допомогою фреймворку Spring Data та його модуля Spring Data JPA.

Також для створення динамічних сторінок використовуватиметься механізм шаблонів або шаблонізатор Thymeleaf.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Види мережного обладнання

Створення надійних мереж для передачі даних з ефективним розподілом мережевого трафіку передбачає використання високотехнологічного комутаційного та периферійного обладнання і асортимент провідних компаній, таки як CISCO, Huawei, Arista, H3C, Hewlett-Packard та інших, на сьогоднішній день представленні масою мережевих рішень.

В залежності від функціоналу мережеве обладнання буває двох видів: пасивне та активне.

Активне мережеве обладнання – це група пристроїв основні особливості якого полягають у тому, що воно містить електронні схеми, для роботи повинно підключатися до електричної мережі, приймає та підсилює, обробляє перетворює і передає інформацію відповідно до встановлених алгоритмів.

До активного мережного обладнання відносяться:

— Маршрутизатор – це апаратний пристрій, який підключено до декількох каналів для різних мереж через інтерфейс, розташований в кожній мережі. Він зазвичай розташований на рівнях мережі, які визначають шлях передачі даних, при цьому маршрутизатор виступає в якості блоку обробки інформаційних пакетів. Маршрутизатор дублює інформаційні пакети для використання при передачі з однієї мережі в іншу. Він використовує певний протокол або набір правил, щоб визначити, які інформаційні пакети повинні бути спрямовані на певні інтерфейси в мережі. Маршрутизатори різних типів виконують різні функції в залежності від вимог мережевої системи;

— Мережеві комутатори працюють аналогічно маршрутизаторам, оскільки вони обидва копіюють інформацію з однієї області мережі в іншу. Однак мережеві комутатори містять кілька портів для копіювання кадрів інформації з одного порту на інший. Як і маршрутизатори, комутатори працюють на рівнях мережі і оцінюють кожен кадр перед визначенням порту, в який кадр повинен бути скопійований. Мережеві комутатори більш складні, ніж їх попередник, мережевий концентратор, який копіював всі кадри на всі порти замість визначення окремих пунктів призначення. Це вимагало більшої пропускної здатності, ніж потрібно для мережевих комутаторів;

— Мережевий адаптер – вони використовуються для підключення кожного комп'ютера до мережі, щоб вони могли зв'язуватися з мережевим маршрутизатором для отримання інформаційних пакетів. Інтерфейсні карти визначають інфраструктуру локальної мережі (LAN) і дозволяють всім комп'ютерам підключатися до мережі. Існує безліч різних типів мережевих карт, які виконують різні функції в мережі, включаючи карти Ethernet і карти бездротового мережевого інтерфейсу.
— Повторювач - пристрій, що повторює сигнал з метою розширення діапазону мережі;
— Концентратор або Hub – тип мережевих пристроїв-повторювачів сигналу, основне призначення яких об'єднання декількох машин або пристроїв Ethernet в один загальний сектор однієї мережі.
— Міст - розділяє трафік в локальній мережі, розділяючи локальну мережу на кілька різних сегментів. Він також відповідає за фільтрацію даних шляхом визначення місця призначення даних або видалення непотрібних даних. Мережеві мости працюють на рівнях мережі, а також керують даними, які перетинають кордони однієї локальної мережі в іншу.
— Медіаконвертери - пристрій, здатний перетворити середу передачі даних;

— Мережевий трансивер - може змінити інтерфейс передачі даних;
— Міжмережеві екрани - використовуються для забезпечення безпеки мережі.

Пасивне мережне обладнання – це устаткування, яке не отримує живлення від електричної мережі або інших джерел живлення.

Пасивне обладнання здійснює свого роду обслуговування активного. Воно не має інтелектуальних функцій, тому не має можливості самостійно керувати потоками, розподіляти або переробляти сигнал, також, як і не спроможне обробляти інформацію про надходження даних.

Пасивне мережеве обладнання включає:

— Кабель (кручена пара) є найбільшим і найпростішим елементом пасивного обладнання. Основні з них виготовлені з міді та волокна.
— Кабельні органайзери з металу або пластику. Використовується для формування кабелів та захисту їх від зовнішніх пошкоджень. Деякі здатні захистити від електромагнітних перешкод.
— Кабель-канали - це коробки, які необхідні для захисту проводів, але в той же час вони мають більш стриманий зовнішній вигляд.
— Розетки. Розміщені на перегородках і надійно тримають прикріплені до них кабелі. Вони призначені для кріплення мережевих кабелів у приміщенні.
— Патч-корди - це комутаційні кабелі, які з'єднують активне мережеве обладнання між собою або з дротовою мережею.
— З'єднувач - роз'єм, який розташований на кінцях патч-корду.
— Патч-панелі з'єднують дроти, що надходять від споживача, а потім підключаються до активного обладнання.
— Серверні шафи - це металеві конструкції, які можна використовувати для компактного розміщення мережевого обладнання та захисту його від зовнішніх впливів.
— Серверні стійки - це шафи без стін, призначені для розміщення обладнання.

1.2 Визначення критеріїв вибору мережевого обладнання

Обирати мережеве обладнання в першу чергу потрібно з урахуванням того, які функції воно повинно буде виконувати. Для будинку з одним, максимум трьома підключеними до мережі пристроями цілком достатньо буде бездротових варіантів. Підійде таке мережеве обладнання для малого офісу або створення невеликих мереж. Чим більша кількість пристроїв буде підключено та надійніше і безпечніше зв'язок потрібно - тим більш складним і дорогим буде обладнання.

1.2.1 Вибір комутатора

Придбання комутатора в багатьох випадках зумовлене необхідністю розширення та модернізації мережі, а ефективність та безперебійна робота мережі безпосередньо залежить від правильного його вибору. До цього питання слід поставитись серйозно і взяти до уваги кілька важливих параметрів.

Мережевий комутатор - це пристрій з різним набором функцій та можливостей, залежно від конкретної моделі. Від цього залежить ціна, тому важливо заздалегідь визначити, які функції будуть затребувані для мережі певної організації, а які можна ігнорувати або повністю ігнорувати.

Всі комутатори можна приблизно розділити на дві великі категорії:

— Некерований. Такі пристрої здійснюють автоматичне управління передачею даних. У цих комутаторів зазвичай відсутня можливість ручного налаштування функцій та моніторингу мережі. Також неможливо оновити програмне забезпечення (іншими словами, неможливо буде "перепрошити" пристрій). Основна програма некерованих комутаторів полягає в об'єднанні декількох пристроїв у мережу та наданні доступу до Інтернету у випадках,

коли немає потреби обмежити швидкість на певних портах або визначити пріоритет трафіку деяких протоколів.

— Керований. Вони також мають можливість працювати в автоматичному режимі, але головною відмінністю від першої групи є можливість ручного налаштування функцій керування трафіком та ведення моніторингу. Це дає таким пристроям значну перевагу - організацію мережі та оперативні зміни в налаштуваннях з урахуванням конкретних завдань.

Основні характеристики комутаторів:

— Кількість портів. Цей параметр є числовим значенням для пристроїв, які можна підключити до комутатора. Зазвичай він варіюється від 5 до 48, але є моделі в яких цей параметр перевищує 54. Однак при виборі важливо надати "запас" для можливого розширення мережі в майбутньому. Так, наприклад, придбання 24-портового комутатора з меншою кількістю поточних мережевих пристроїв може забезпечити майбутнє розширення та уникнути витрат.

— Швидкість передачі даних. Ще одним важливим параметром комутатора є швидкість передачі даних. Важливо врахувати, скільки трафіку передається по мережі. Наприклад, у випадку, коли необхідно періодично створювати резервні копії декількох сотень гігабайт інформації з декількох носіїв, доцільно придбати комутатор з портами 1000 Мбіт/с. При порівняно менших обсягах трафіку буде достатньо 100 Мбіт/с. Далі слід розглянути, яку швидкість підтримують адаптери комп’ютерів та інших мережевих пристроїв. Якщо вони оснащені мережевими адаптерами Fast Ethernet швидкістю 100 Мбіт/с, то придбання гігабітного комутатора недоцільно. Тим не менш, важливо знати, що всі сучасні материнські плати оснащені інтерфейсом Gigabit Ethernet.

— Внутрішня пропускна здатність. Цей параметр призначений для демонстрації обсягу трафіку, який пристрій буде обробляти протягом періоду пікового завантаження на всіх портах. Ця характеристика не є

загальною ємністю всіх портів. Слід мати на увазі, що внутрішня пропускна здатність може бути меншою, особливо коли комутатор використовується з великою кількістю портів (24 або більше портів).

— Рівні комутаторів. Комутатори використовуються для побудови мереж, взаємозв'язку мережевих пристроїв та передачі даних з одного порту на інший на основі інформації, отриманої з переданих пакетів. Інформація організована відповідно до семирівневої моделі OSI, яку постачальники мереж використовують для забезпечення взаємодії між своїми продуктами. Звідси такі терміни, як рівень 2, рівень 3 тощо, в основному стосуються комутаторів, а також інших мережевих пристроїв.

1.2.2 Вибір маршрутизатора

Вибір найбільш задовільного маршрутизатора повинен ґрунтуватися на увазі до кількох параметрів. Основне завдання пристрою - створити локальну мережу між кількома комп’ютерами, отже, слід враховувати її структуру. Основні параметри на які слід звернути увагу це:

— Кількість портів. Цей параметр визначається завданнями і числом комп'ютерів, що підключаються до маршрутизатора, при цьому краще забезпечити певний «запас» на випадок виведення з ладу одного з портів.

— Брандмауер. Наявність вбудованого брандмауера надає ефективний захист мережі від атак з Інтернету. У багатьох випадках ця функція дійсно необхідна. Багато сучасних пристроїв оснащені цією функцією, оскільки захист даних - завдання не менш важливе, ніж забезпечення безперебійної роботи.

— Швидкість передачі даних. Швидкість обміну даними повинна відповідати параметрам інтернет-підключення, обсягами переданих даних, на підставі цього визначається і пропускна здатність.

— Швидкість WAN-порту. Цей параметр означає, яку максимальну швидкість підключення до мережі зможе підтримати пристрій. Вибір також ґрунтується на тому, наскільки висока швидкість інтернету в локальній мережі.
— Число WAN-портів. Кілька портів робить можливим не обмежуватися одним провайдером або єдиним підключенням. Це дозволяє мати резервне Інтернет-з'єднання або збільшити швидкість.
— Об'єм оперативної пам'яті. Велика кількість оброблюваних даних зобов'язує звернути вибір на обсяг оперативної пам'яті - чим він більший, тим ширше можливості пристрою.

1.2.3 Вибір пасивного обладнання

Вибір пасивного обладнання також має велике значення для роботи системи. Так неякісний кабель в кращому випадку може привести до зниження якості сигналу та швидкості передачі даних, а гіршому – стати причиною виходу обладнання з ладу. До таких наслідків може призвести безвідповідальний і невиправдано економний підхід до будь-якого іншого типу пасивного обладнання.

Саме від пасивного обладнання залежить якість зв’язку між маршрутизаторами, точками доступу, міжмережевими екранами та іншими видами активного мережного обладнання. Якщо всі ці елементи змонтовані коректно, то досягається висока надійність функціонування мережі, забезпечується якість його функціонування обладнання на всіх ділянках, економія електроживлення, а також значно спрощується процес підключення нового обладнання до мережі.

При виборі мережного устаткування можливо орієнтуватися тільки на високовиробниче обладнання, але не на всіх ділянках мережі воно буде повністю

затребувано. Тобто доцільніше робити вибір так, щоб забезпечити вимоги пропускної здатності і досягти ефективності використання з точки зору співвідношення продуктивності, надійності та ціни.

Провівши дослідження і з'ясувавши, що критеріями вибору є набір його ключових характеристик, то пошук оптимальної конфігурації потрібних пристроїв буде заснований саме на цьому.

Так як основні труднощі виникають при виборі активного мережного обладнання такого як комутатори та маршрутизатори то розробка веб-додатку буде націлена на пошук оптимальної конфігурації саме цього обладнання.

Для досягнення поставленої мети потрібно дослідити технології і принципи, що використовуються для розробки веб-додатків.

1.3 Поняття «Web-додаток»

Веб-додаток – це додаток, що розробляється з використанням веб-технологій. Зазвичай вони працюють в веб-браузері при переході по певній URL-адресі.

Кожен веб-додаток складається з двох частин: front-end та back-end

- Першою частиною є front-end – це клієнтська частина, яка відображається та працює у веб-браузері користувача. Він служить інтерфейсом для введення даних, запуску команд і перегляду даних. Всі елементи з якими може взаємодіяти користувач знаходяться в цій частині.
- Другою частиною є back-end – це серверна частина, яка виконується на віддаленій хост-машині та займається тим, що приймає запити від клієнтської. В серверній частині знаходиться основна логіка веб-додатку. В цій частині формується структура сторінок, встановлюється зв'язок з іншими службами, такими як сховища даних, сервери електронної пошти, зовнішні API та ін.

Всі веб-додатки засновані на архітектурі клієнт-сервер.

Розробники кодують веб-додатки на двох типах мов. Веб-додаток зазвичай використовує комбінацію сценарію на стороні сервера і сценарію на стороні клієнта для роботи. Сценарій на стороні сервера займається зберіганням і отриманням інформації та вимагає таких мов, як Python або Java. Розробники програмують на стороні сервера сценарії, які буде використовувати веб-додаток.

Для клієнтського скрипта потрібні такі мови, як JavaScript, каскадні таблиці стилів (CSS) і HTML5. Ці мови покладаються на браузер для виконання програми. Це мови, підтримувані браузером. Клієнтський скрипт займається представленням інформації користувачеві.

Технічно кожен веб-додаток також є веб-сайтом. Веб-сайти – це перш за все інформаційні ресурси, що означає, що основною його частиною є контент, і цей контент в основному статичний.

1.4 Виділення різниці в порівнянні Web-додатків та Web-сайтів

Різниця між поняттями веб-сайту і веб-додатком може бути неважлива для простого користувача. Однак вона буде мати значення при розробці інтернет-ресурсу для бізнесу з точки зору розуміння його вимог, планування на майбутнє, спілкування з потенційними розробниками.

Перше, з чого потрібно почати виділення різниці між веб-додатком і веб-сайтом, - це інтерактивність. Веб-сайт надає візуальний і текстовий контент, який користувач може бачити і читати, але ніяк не впливає на нього. У разі веб-додатку користувач може не тільки читати вміст сторінки, але і керувати даними на цій сторінці. Взаємодія приймає форму діалогу: користувач натискає кнопку або відправляє форму і отримує відповідь зі сторінки. Ця відповідь може приймати форму завантаження документа, онлайн-чату, електронного платежу і т. д..

Наочним прикладом інтерактивності веб-додатку є додаток онлайн-банкінгу, який виконує транзакції на основі введених даних клієнта. Аналогічні функції можна знайти в інтернет-магазині, що дозволяє відвідувачам шукати по каталогу і миттєво купувати товари. Соціальні мережі - ще один приклад. Вони підключають користувачів через чати і платформи блогів, генерують контент для каналів на основі переваг користувачів і забезпечують практично необмежений обмін контентом, не кажучи вже про їх вбудованих міні-додатках для розваги користувачів. Проблема в тому, що сьогодні рідко можна зустріти веб-сайт без натяку на інтерактивність. Сучасні веб-сайти зазвичай містять невеликі елементи веб-додатків. Наприклад, веб-сайт ресторану може містити віджет Google Maps, що показує маршрут до цього ресторану. Однак в разі веб-сайтів баланс між інформаційним змістом і інтерактивністю зміщений в сторону першого. Типовий веб-сайт містить набагато менше інтерактивних елементів, ніж інформаційний контент, і користувач зазвичай проводить більшу частину часу на веб-сайті, читаючи, переглядаючи або слухаючи. З веб-додатками ситуація протилежна, оскільки їх основна функціональність заснована на взаємодії.

Друге, що відрізняє веб-сайт від веб-додатку – інтеграція. Інтеграція означає об'єднання різних компонентів для побудови більш всеосяжної системи. І веб-сайти, і веб-додатки можуть бути інтегровані з іншим програмним забезпеченням. Проте, інтеграція більш типова для веб-додатків, оскільки їх складна функціональність часто вимагає взаємодії з додатковими системами. Візьмемо, наприклад, інтеграцію бізнес-веб-додатку (наприклад, інтернет-магазину) з системою CRM (Customer Relationship Management). CRM зберігає всі дані про клієнтів в одному місці, забезпечуючи легкий доступ до них для співробітників. Інтеграція дозволить автоматично збирати дані про користувачів веб-додатків і зберігати їх в CRM. Таким чином, ваша команда отримає доступ до повного набору даних про клієнтів, їх запитах, спілкуванні та відгуках. Це дозволяє вивчати поведінку і купівельні звички клієнтів, а також швидше врегулювати їх претензії. Більш того, будь-яка зміна даних про клієнтів буде миттєво

відображено в CRM. Завжди залишаючись в курсі переваг клієнтів, ви зменшите відтік клієнтів і збільшите продажу.

Веб-сайт також може бути інтегрований з CRM. Це дозволяє надавати користувачам більш персоналізований контент. Однак для веб-сайту це швидше функція, що рідко реалізується, ніж частина основних функцій.

Третя різниця – це аутентифікація. Аутентифікація - це процедура, яка включає в себе введення логіна і пароля користувача для доступу до системи. Це необхідно для веб-програмного забезпечення, яке вимагає будь-якої особистої інформації. Облікові записи користувачів повинні бути захищені від несанкціонованого доступу і витоку конфіденційних даних.

Веб-додатки в основному вимагають аутентифікації, так як вони пропонують набагато ширший набір можливостей, ніж веб-сайти. Розглянемо приклад соціальних мереж. При реєстрації ви створюєте обліковий запис і отримуєте унікальний ідентифікаційний номер. Система попередить вас, якщо ваш логін і пароль ненадійні. Якщо ви залишите їх без змін, хакери можуть отримати доступ до вашого профілю і вкрасти вашу інформацію.

Для інформаційних сайтів аутентифікація не обов'язкова. Користувачеві може бути запропоновано зареєструватися, щоб отримати доступ до додаткових функцій, недоступним для незареєстрованих відвідувачів сайту. Наприклад, ви можете переглядати новини і вибрані статті на новинному веб-сайті, не турбуючись про реєстрацію. Однак, якщо ви хочете залишити коментар, вам необхідно увійти в систему. Таким чином, користувачі підтверджують свою особистість, дозволяючи системі блокувати спамерів.

Як можна побачити, аутентифікація може знадобитися як для веб-сайтів, так і для веб-додатків. Однак для веб-додатків це обов'язково з міркувань безпеки.

1.5 Переваги та недоліки веб-додатків.

Першою перевагою є незалежність від платформи. На відміну від настільних та мобільних програм, веб-додатки спроможні запускатися на кожному пристрої, що має веб-браузер.

Друга перевага – це відсутність установки. Веб-додатки не потрібно встановлювати перш ніж ви зможете її використовувати. Єдине, що потрібно зробити, це перейти до веб-браузера за певною URL-адресою. Це досить вагома перевага для кінцевих користувачів оскільки позбавляє від необхідності великих завантажень, встановлення, конфігурації, вимог до місця на жорсткому диску.

По-третє – доступ звідусіль. Усі данні зберігаються на сервері (або серверах), що в сою чергу надає користувачеві можливість отримувати доступ до своїх даних звідусіль, без необхідності передавати файли.

Також не менш важливою перевагою є простіша підтримка зі сторони розробників. Розробники можуть обирати найбільш оптимальне середовище розгортання, і це єдине, що їм доведеться підтримувати. Це повністю усуває проблеми пов’язані з пристроєм кінцевого користувача або з різними операційними системами.

Не зважаючи на досить вагомі переваги веб-додатків вони мають і свої недоліки.

Одним з таких недоліків є «слабозв’язна» архітектура веб-мережі – відсутність підтримки стану сеансу роботи і затримка при перезавантаженні кожної сторінки. Кожне перезавантаження (або оновлення сторінки) викликає помітну затримку, викликану необхідністю встановити HTTP - з'єднання, обробити запит на сервері, передати по мережі у відповідь HTTP-повідомлення і перезавантажити сторінку браузером. Це створює скачки та переривчастий режим роботи користувача.

Також досить важливим недоліком є обмежений набір елементів управління для розробки форм додатків. Поточна версія мови HTML підтримує тільки обмежений набір елементів управління (текстові елементи, перемикачі, прапорці, списки, що розкриваються і командні кнопки). Мова не пропонує підтримки складних взаємодій, які часто використовуються в настільних додатках, таких як календарі, майстри, закладки, панелі інструментів, контекстні меню і т.п., які доступні навіть в найпростіших локальних додатках. Хоча такі елементи управління можуть бути розроблені з використанням JavaScript і CSS, відсутність підтримки вбудованого браузера призводить до безлічі реалізацій з несумісними уявленнями і способами роботи з ними. Як базова технологічна архітектура Інтернету, так і обмежений набір доступних елементів управління ускладнюють підтримку взаємодії з веб-додатками в порівнянні з локальними програмами. Крім того, взаємодія і зовнішній вигляд веб-додатки можуть не збігатися в різних операційних системах, оскільки більшість веб-додатків зроблені незалежними від конкретного браузера.

1.6 Аналіз сучасних підходів та проектних рішень щодо проектування і створення веб-додатків

На початку своєї історії всі програми будувалися без будь-яких архітектурних принципів, програма складалася з безлічі наступних один за одним рядків: без класових залежностей, наслідувань і інших сучасних можливостей мов програмування. Однак, прагнення до спрощення розробки, а також до структуризації як коду, так і всієї програми могло бути передумовою до появи такого поняття як «архітектура ПЗ».

В цілому можна виділити такі підходи до архітектурного проектування веб-додатків:

— Монолітний підхід;

— Модульний підхід;

— Сервіс-орієнтований підхід.

Монолітний підхід є найстарішою моделлю проектування ПЗ оскільки саме з неї і почалася розробка всього програмного забезпечення. В рамках даного підходу складної структури веб-додатки як такої може не бути: сервер зберігає всю бізнес-логіку, а база даних - дані необхідні сервера для роботи. Як правило подібні програми не відрізняються складністю в розробці і її великою вартістю на ранніх етапах, коли список необхідного функціоналу не відрізняється великою кількістю рядків, а помилкові дії досить просто виправляються на ранніх етапах.

Нова функціональність додається легко і швидко. Однак з плином часу або в поспіху за датою випуску цілком вірогідне зростання ризику припуститися помилки, яка перетворюється в «технічний борг», який необхідно буде виправити після випуску продукту. Однак з плином часу при монолітному підході, таких «помилок» може накопичитися дуже багато і таким чином, підтримувати монолітну систему в довгостроковій перспективі дуже складно і дорого, тому що команди розробників можуть змінюватися і виходить так, що простіше зробити «латочку», ніж розібратися як працює та чи інша частина програми.

Проблеми з підтримкою монолітного додатку можна пояснити і тим, що рано чи пізно невеликий список функціоналу в системі поповнюється новими вимогами та ідеями, які вимагають внесення змін до вже існуючого коду. Крім того, в рамках веб-додатку однією з проблем «моноліту» є масштабованість. Всі складові системи розташовані в одній точці, а тому потужність самої точки повинна бути відповідною для підтримки сервера і бази даних в робочому стані.

Приклад монолітної архітектури можна знайти в будь-якому додатку, навіть якщо він включає в себе різні класи. Варто відзначити, що в цей момент може здатися, що додаток стає модульним, проте це зовсім не так. Монолітний додаток по праву може включати модулі, однак, вони повністю або частково залежать один від одного.

Модульна архітектура на відміну від монолітної має на увазі розбиття всього функціоналу програми на окремі модулі, кожен з яких відповідає за певну частину функціоналу програми таким чином, що модульну архітектуру можна описати як сукупність безлічі монолітних модулів всередині однієї програми. Кожен модуль програми є функціонально незалежним від іншого, а тому його застосування в різних ділянках коду не викликатиме збоїв в працездатності, по суті це надає простоту рефакторінгу і перенесення між модулями. Варто відзначити, що при зміні одного модуля, інші модулі порушені не будуть, а тому спрощується завдання налагодження (debug).

Все, що потрібно, це дещо змінити налаштування інших модулів, що використовують функціонал зміненого під нові можливості і вимоги останнього, якщо зміни в ньому можуть призвести до збою в роботі програми. Крім того, всі модулі взаємодіють один з одним в синхронному порядку, оскільки знаходяться на одній апаратній частині, а при необхідності для модуля можна виділити окрему базу даних або ж частину даних, з якими цей модуль буде працювати.

Модульний підхід дозволив розробляти більш складні програми, що складаються з безлічі різних модулів. Розвиток даного підходу призвело до того, що деякі модулі почали виносити на окремі апаратні частини, що таким чином призвело до виникнення цілих окремих сервісів, а відповідно сервіс-орієнтованого підходу.

Сервіси – це окремі цілком самодостатні модулі, що володіють власною апаратною базою, а саме розташовуються на окремому сервері. Крім того, вони можуть володіти власною базою даних, а оскільки вони розташовуються на окремих апаратних пристроях, навіть якщо віртуально, то і взаємодія між сервісами здійснюється асинхронно, що може надати як певні переваги, так і деякі недоліки. Однак, одним з ключових переваг SOA (Service-oriented architecture) це те, що вона надає можливість незалежного масштабування компонентів, маючи на увазі збільшення потужності тільки того сервісу, який цього потребує, не зачіпаючи апаратну частину інших.

Можливість розробки цілісного додатку сервіс-орієнтованим підходом, надає можливість розробки сервісів на різних мовах програмування. Простий приклад: сервіс доставки повідомлень може бути написаний на NodeJS, де в якості мови програмування використовується JavaScript, в той час як основна програма, що використовує цей сервіс, може складатися з зв'язки React на клієнтській частині і Java на серверній. Для взаємодії цих двох частин необхідно тільки правильно використовувати програмний інтерфейс сервісу в основній частині програми.

Розробка сервісів відбувається окремо один від одного, а тому зміни в коді одного сервісу не будуть впливати на інші, якщо тільки не змінилася видача даних в програмному інтерфейсі сервісу, що відбувається вкрай рідко, оскільки з форматом виведення визначаються ще не перших етапах розробки сервісу.

1.7 Клієнт-серверна архітектура

Клієнт-серверна архітектура – це мережна архітектура, яка передбачає розділення за типом роботи комп’ютерів, що знаходяться в одній мережі, на сервери та клієнти. Сервери займаються тим, що розміщують, розподіляють і керують ресурсами та службами, які споживаються клієнтом. В такому типі архітектури мінімальна мережа складається з одного або декількох комп’ютерів-клієнтів підключених до комп’ютера-сервера локально або через Інтернет.

Клієнт-серверна архітектура також відома як модель мережевих обчислень або мережа клієнт-сервер, оскільки всі запити і послуги розподіляються по мережі. Сервери – це виділені процеси для управління комп'ютерами або дисковими накопичувачами (файлові сервери), принтерами (сервери друку) або мережевим транспортом (мережеві сервери). Клієнти – це ПК або робочі станції, на яких користувачі запускають додатки. Клієнти покладаються на ресурси серверів, таких як файли, пристрої та навіть обчислювальні потужності.

Архітектура клієнт-сервер в першу чергу розроблена для середовищ де кількість комп'ютерів досить велика. Наприклад, багато комп’ютерів пов'язані один з одним за допомогою мережевих технологій. Будь-який з них може перетворюються в робочу станцію. І файли всіх цих комп'ютерів зберігаються на сервері, така модель називається клієнт-серверної. У цій моделі є один або кілька комп'ютерних клієнтів і один сервер, де кожен клієнт відправляє запити на сервер через мережу, і сервер відповідає на цей запит. Така архітектура мережі допомагає використовувати ресурси в спільному режимі.

1.7.1 Клієнтський процес

Клієнт - це комп'ютерна система, яка виконує доступ до сервера на інших комп'ютерах через мережу. Клієнтські програми зазвичай керують частиною користувальницького інтерфейсу додатку, причому клієнтський процес є зовнішнім інтерфейсом програми, яку користувач бачить і з якою взаємодіє. Клієнтський процес також керує локальними ресурсами, якими він володіє, такими як монітор, клавіатура, центральний процесор робочої станції.

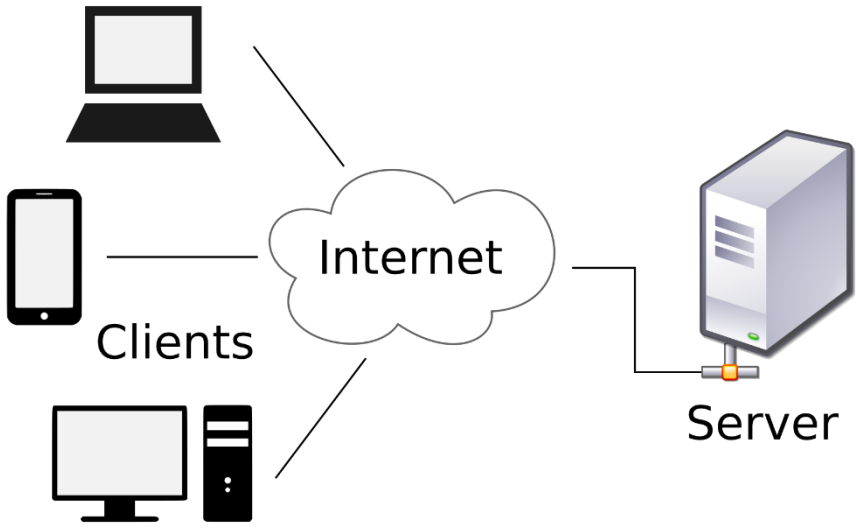

Рисунок 1. – Архітектура клієнт-сервер

Одним з ключових елементів клієнтської частини є графічний інтерфейс користувача (GUI).

1.7.2 Серверний процес

У клієнт-серверній архітектурі серверний процес - це програма, яка виконує заплановані завдання, у відповідь на запит клієнта. Сервером також можуть бути і інші машини, що знаходяться в мережі. Виступати сервером може як операційна система так і інші пристрої.

Серверний процес діє як програмний механізм, який керує загальними ресурсами, такими як бази даних, принтери, канали зв'язку або високоефективні процеси. Також він виконує внутрішні завдання, загальні для подібних додатків.

1.7.3 Дворівнева архітектура

У цьому типі клієнт-серверної архітектури інтерфейс користувача зберігається на клієнтській машині, а база даних зберігається на сервері. Логіка бази даних і бізнес-логіка зберігаються або на клієнті, або на сервері, але вони не повинні змінюватися. Якщо бізнес-логіка і логіка даних зберігаються на стороні клієнта, це називається архітектурою тонкого сервера товстого клієнта. Якщо бізнес-логіка і логіка даних зберігаються на сервері, це називається архітектурою товстого сервера з тонким клієнтом.

Дворівнева архітектура корисна, коли клієнт безпосередньо звертається до сервера. Зазвичай вона використовується в невеликих приміщеннях (менше 50 користувачів). Тут інтерфейс користувача розміщується в середовищі робочого столу користувача, а служби СУБД зазвичай розміщуються на сервері. Обробка

інформації розділена між середовищем інтерфейсу користувача системи і середовищем сервера управління базою даних.

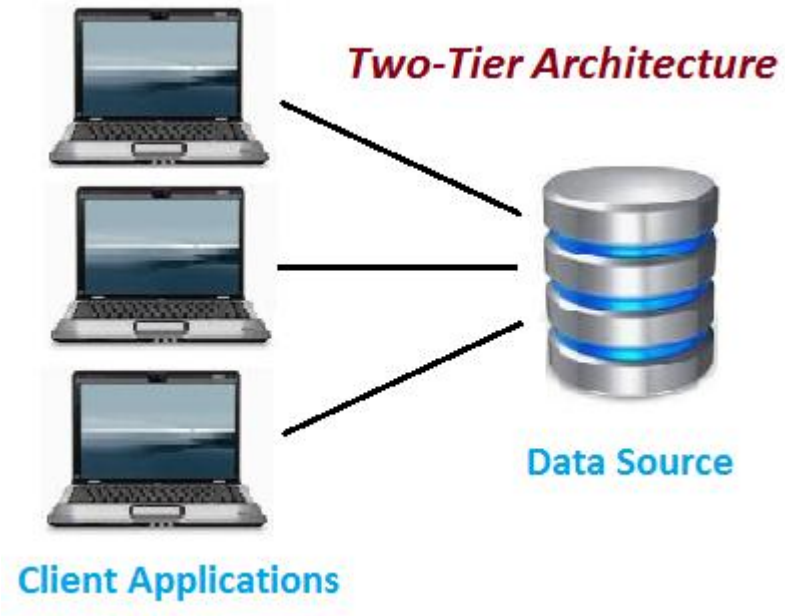

Рисунок 1. – Приклад дворівневої архітектури

1.7.4 Трьохшарова архітектура

В трьохрівневій архітектурі клієнт-серверу використовується додаткове проміжне програмне забезпечення, це означає, що запит клієнта проходить на сервер через цей проміжний рівень, а відповідь сервера спочатку приймається проміжним програмним забезпеченням, а потім клієнтом. Ця архітектура долає всі недоліки дворівневої архітектури та забезпечує кращу продуктивність. Проміжне ПО зберігає всю бізнес-логіку і логіку доступу до даних. Якщо існує декілька бізнес-логік і логік даних, це називається n-рівневої архітектурою. Метою проміжного програмного забезпечення є взаємодія з базою даних, організація черг, виконання додатків, планування і т.д. Проміжне програмне забезпечення може бути файловим сервером, сервером повідомлень, сервером додатків, монітором обробки транзакцій і т.д. Це підвищує гнучкість і забезпечує кращу продуктивність.

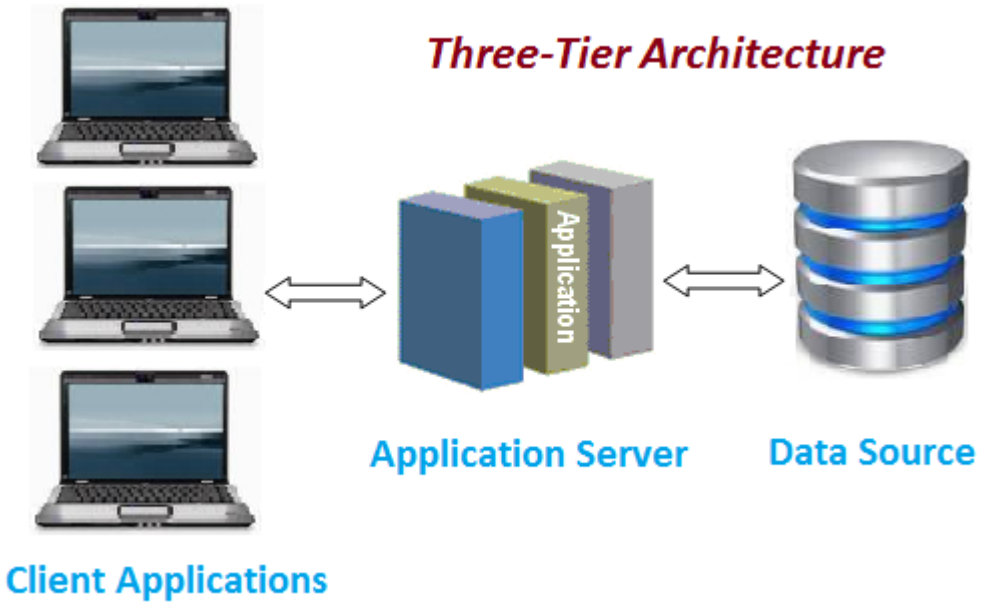

Рисунок 1. – приклад трьохрівневої архітектури

Сьогодні майже всі веб-додатки та просто клієнт-серверні додатки спроектовані у вигляді трьохрівневої архітектури або навіть n-рівневої. Це дозволяє розподілити навантаження між різними серверами та надає переваги в обслуговуванні і мобільності таких систем.

ВИСНОВКИ

В цьому розділі були розглянуті види мережного обладнання такі як: активне, пасивне. Була приділена увага до основних їх характеристик, які є водночас критеріями вибору.

Ознайомилися з визначенням веб-додатку та його основними частинами. Розглянули відмінності веб-додатку від веб-сайту та основні переваги і недоліки веб-додатків. Також приділили увагу підходам до проектування сучасного програмного забезпечення, які використовуються і в веб-додатках.

Визначили, що таке клієнт-серверна архітектура, які процеси до неї входять та з яких рівнів вона може складатися.

Проаналізувавши основні критеріях вибору мережевого обладнанням визначили параметри пошуку оптимальної конфігурації мережевого обладнанням, які будуть використовуватися в веб-додатку.

2 ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

При створенні веб-додатку були використані такі засоби реалізації:

— мова гіпертекстової розмітки HTML;
— каскадні таблиці стилів (CSS) для стилізації інтерфейсу користувача;
— мова програмування Java для написання серверної частини;
— фреймворк Spring Boot для спрощення та прискорення розробки програмного продукту;
— фреймворк Spring Data JPA для взаємодії серверної частини з базою даних;
— фреймворк Spring Security для налаштування аутентифікації та доступу;
— фреймворк Apache Maven для автоматичного завантаження залежностей проекту;
— система керування базами даних MySQL;
— шаблонизатор Thymeleaf;
— мова програмування JavaScript.

2.1 HTML

HTML – це скорочення, яке позначає мову гіпертекстової розмітки (англ. Hyper Text Markup Language). Він є базовою структурою всесвітньої павутини веб-сторінок в мережі Інтернет. В якості розмітки HTML виконує завдання опису докладної логічної структури веб-сайту. HTML виник з сімейства мов на основі SGML.

Мова гіпертекстової розмітки – це мова структурування текстів, але в ній також є можливість включати графіку і мультимедійний контент у вигляді посилань та інтегрувати їх в текст.

HTML дозволяє розділити веб-сторінки на такі розділи як: заголовок, вміст, нижній колонтитул, навігація і стаття. Він також дає можливість створювати:

— заголовки, абзаци тексту, списки і таблиці;
— інтерактивні посилання на будь-який інший веб-сайт або джерело даних в інтернеті;
— посилання на нетекстовий контент.

Не менш важливою особливістю HTML є наявність інтерфейсів для мов розширення, таких як: CSS або JavaScript, за допомогою яких з'являється спроможність створювати елементи за своїм бажанням або здійснювати взаємодію з користувачем.

Схема розмітки HTML заснована на ієрархічній структурі. Документи HTML початково складаються з теми, або просто заголовку, і контенту. Контент зазвичай складається із загальних елементів для областей веб-сайту, наприклад заголовків, стовпців вмісту, навігації. Стовпці вмісту складаються з заголовків 1-го та 2-го порядку, абзаців тексту, списків, таблиць і графіків. Більшість з цих елементів в свою чергу мають підгрупи елементів. Наприклад, абзац тексту може містити уривок позначений як виділений, напівжирний або курсив, маркований список складається з окремих елементів списку, а таблиця розділена на окремі рядки та стовпці.

HTML розроблений таким чином, що вся структура елементів веб-сайту відображена як деревоподібна структура з все більш тонкими гілками.

На рисунку 2.1 зображено типовий шаблон розмітки сторінки з використанням тегів HTML.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Заголовок</title>
</head>
<body>
    <header>
        <nav>
            <ul>
                <li>Меню</li>
            </ul>
        </nav>
    </header>
    <main>
        <article>
            <header>
                <h2>Заголовок</h2>
                <p>Абзац <a href="#">Автор</a> - <a href="#comments">6 коментарів</a></p>
            </header>
            <p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Assumenda facilis ipsam sint omnis eligendi voluptas? Ipsum tenetur eveniet animi accusantium
            pariatur cumque, quibusdam ducimus optio placeat voluptates error. Iure, inventore!</p>
        </article>
        <article>
            <header>
                <h2>Заголовок</h2>
                <p>Абзац  <a href="#">Автор</a> - <a href="#comments">6 коментарів</a></p>
            </header>
            <p>Lorem ipsum dolor sit amet consectetur, adipisicing elit. Dolor animi consequuntur quas eligendi. Mollitia labore magni iusto fuga natus. Velit non, earum
            dolores reprehenderit blanditiis voluptas iusto quae ullam fugit.</p>
    </article>
    </main>
    <footer>
        <p>Copyright</p>
     </footer>
</body>
</html>
```

Рисунок 2.1 – Приклад типової розмітки HTML

2.2 **CSS**

CSS розшифровується як каскадні таблиці стилів (анг. Cascading Style Sheets). Це окрема мова, що додатково використовується в основному тільки з HTML. Він легко підключається до гіпертекстової розмітки і виконує такі задачі: формування зовнішнього вигляду контенту та створення макетів веб-сайтів.

При відображенні HTML-документу без інформації CSS в браузері, основні структури відображаються таким чином, щоб їх можливо було легко візуально відрізнити (рис. 2.2). Заголовки відображаються з більшим кеглем шрифту та жирним обведенням, важливі помітки в тексті відображаються курсивом, підкресленням або непропорційним шрифтом. В цьому контексті говорять про внутрішні таблиці стилів, які використовують браузери за замовчуванням для відображення елементів HTML. В специфікації HTML описані рекомендації щодо представлень за замовчуванням.

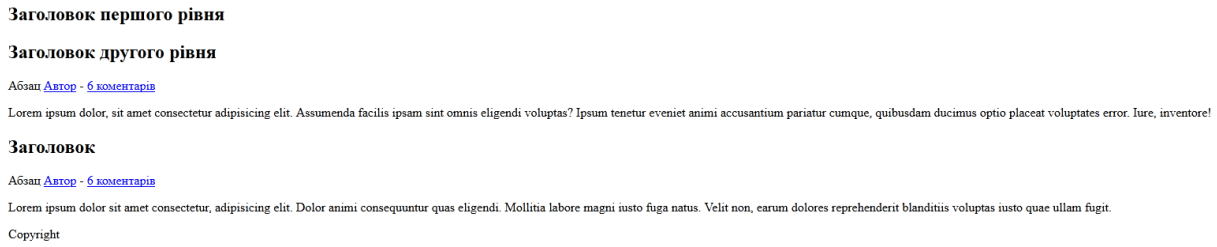

Рисунок 2.2 – Приклад сторінки без CSS

Тут на допомогу приходять каскадні таблиці стилів (CSS). З їх допомогою є можливість повністю змінювати зовнішній вигляд елементів, наприклад, вказати, що всі абзаци мають розмір шрифту 32 пікселя, зелений колір тексту, з інтервалом 5 пікселів та помаранчевим обведенням по периметру (рис 2.3 та 2.4). І так є можливість робити з кожним елементом розмітки HTML.

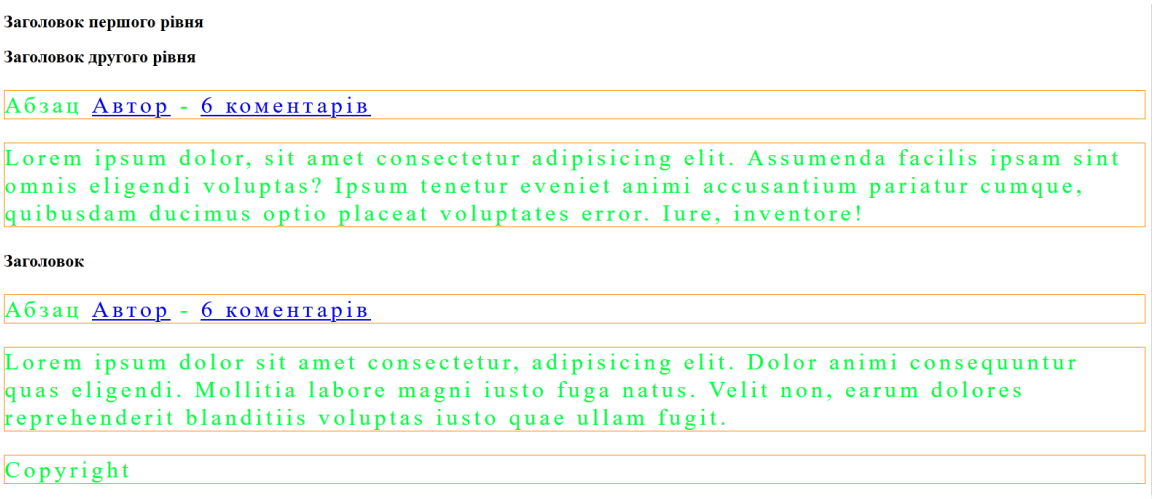

Рисунок 2.3 – Приклад сторінки з підключеним CSS

```
1  p {
2      font-size: 32px;
3      color: rgb(0, 255, 64);
4      letter-spacing: 5px;
5      border: 1px solid rgb(255, 145, 0);
6  }
```

Рисунок 2.4 – Приклад CSS коду

CSS дозволяє визначати центральні формати, наприклад, для усіх заголовків першого рядку чи для усіх блоків з визначеними іменем класу або для виділеного тексту, який знаходиться в таблиці. Опис стилів можна експортувати до

зовнішнього файлу, який в подальшому легко підключається до будь-якої кількості HTML-документів. CSS забезпечує крос-сторінкові, уніфіковані макети і форматування.

Як і HTML, CSS це:

— проста текстова мова. Не обов’язково мати спеціальне програмне забезпечення для CSS, досить текстового редактора.
— відкрито документована мова, стандартизована консорціумом W3, яку ви можете вільно використовувати без проблем з ліцензуванням. Процедура стандартизації CSS слідує таким ж правилам, що і для HTML.

Використання спеціальних властивостей формату CSS для розміщення елементів, дизайну фону і видимих фреймів дозволило створити сучасні багатоколонкові макети веб-сайтів на основі CSS. В принципі, такі макети працюють у всіх використовуваних сьогодні браузерах. Однак і тут не без сюрпризів, не всі браузери реалізують специфікацію CSS однаково, що може бути особливо помітно в макеті сторінки.

2.3 Java

Платформа Java - одна з платформ розробки і розгортання програмного забезпечення. Вона включає до себе пакет програмних компонентів, що забезпечують систему для розробки прикладного програмного забезпечення і його розгортання в будь-якому обчислювальному середовищі. На сьогоднішній день платформа Java широко використовується в широкому розмаїтті обчислювальних платформ, від пристроїв початкового рівня, таких як смартфони, вбудовані пристрої, до високопродуктивних пристроїв, як суперкомп'ютери та корпоративні сервери.

Складають платформу Java три компоненти: Java Runtime Environment (JRE), компілятор та бібліотеки. JRE надає віртуальну машину java (JVM), бібліотеки базової мови необхідні для запуску компонентів мови програмування Java. Щоб програми запускалися в JVM, спочатку вони повинні бути скомпільовані в байт-код Java, стандартний переносний двійковий формат, який зазвичай має розширення .class (файл класу Java)(рис 2.5). Програма Java може складатися з безлічі таких файлів. Платформа надає стандартну систему для упакування цих файлів класів в один архівний файл. Залежно від типу Java-програми розширення файлу архіву може бути .jar, .war, .ear або будь-яким іншим розширенням. Компілятор Java компілює вихідні файли і створює byte-код, незалежний від платформи. Потім байт-код запускається в JRE.

В рамках платформи Java існує кілька інших мов JVM, таких як Scala, Clojure, Groovy і Jython. Однак на сьогоднішній день мова програмування Java є найбільш часто використовуваним мовою програмування на платформі Java.

Достатньо важливою частиною Java є її бібліотеки. Крім основних бібліотек, що забезпечують роботу мови, Java надає декілька інших, необхідних для різних типів пристроїв та додатків. Також Java має наряду з офіційними бібліотеками велику базу бібліотек з відкритим вихідним кодом.

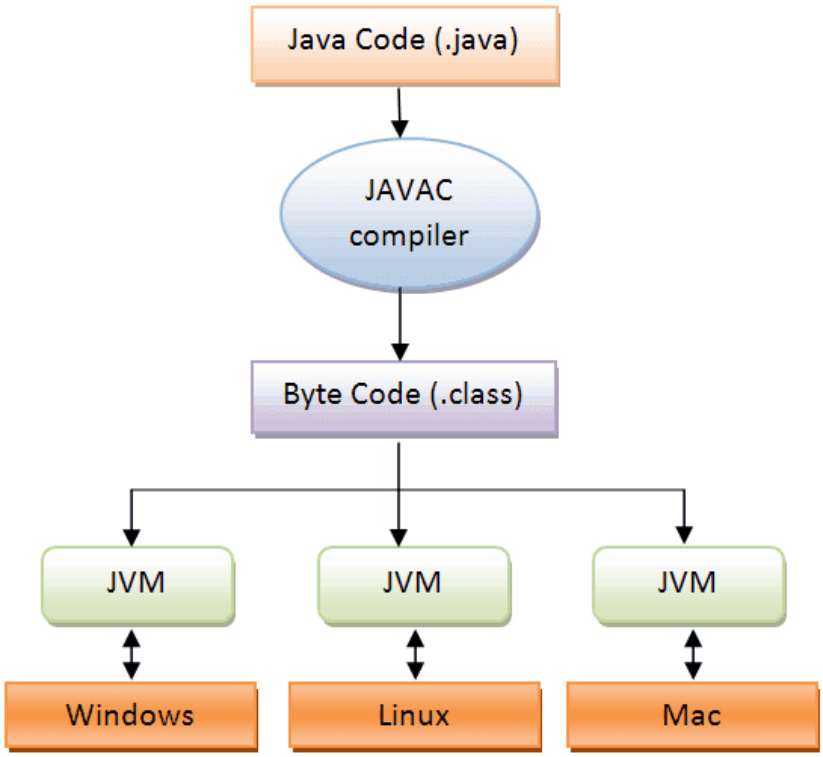

Рисунок 2.5 – Шлях від java коду до байт коду і JVM

Як мова програмування Java є об’єктно-орієнтованою загального призначення, заснованою на класах. Основна ціль розробки Java була в тому, щоб мати якомога меншу кількість залежностей реалізації. ЇЇ одним з основних принципів є – «Напиши один раз і працюй де завгодно». На сьогоднішній день Java вважається однією з найпопулярніших мов програмування світу. На початку Java розроблялась задля програмування побутових електричних пристроїв, але через декілька років вона була у використанні на всіх типах пристроїв, таких як корпоративні сервери, персональні комп’ютери, мобільні, та в інших вбудованих пристроях. Задля задоволення конкретних потреб різних типів пристроїв та додатків існують різні дистрибутиви Java, такі як:

— JavaFX – використовується для створення як настільних так і веб додатків з сучасним графічним інтерфейсом користувача.
— Java Mobile Edition (JME) – розроблена для використання на електронних пристроях з малою кількістю доступних обчислювальних ресурсів, таких як мобільні телефони, вбудовані пристрої.

— Java Standard Edition (Java SE) – використовується для розробки додатків для серверів і комп’ютерів загального призначення.
— Java Enterprise Edition (Java EE) – використовується для розробки додатків корпоративного рівня і описує в собі специфікації пов’язані з веб-додатками, обробкою XML, веб-службами та службами RESTful.
— Java Card – використовується для запуску Java-додатків на смарт-картах та інших пристроях з малим об’ємом внутрішньої пам’яті.

2.4 JDBC

API Java Database Connectivity (JDBC) є галузевим стандартом для незалежної від бази даних зв'язку між мовою програмування Java і широким спектром баз даних, базами даних SQL і іншими джерелами табличних даних, такими як електронні таблиці або плоскі файли. JDBC API надає API рівня викликів для доступу до бази даних на основі SQL.

Технологія JDBC дозволяє вам використовувати мову програмування Java для використання можливостей «Запис один раз, запускати де завгодно» для додатків, яким потрібен доступ до корпоративних даних. За допомогою драйвера з підтримкою технології JDBC ви можете підключити всі корпоративні дані навіть в гетерогенному середовищі.

2.5 Spring

Програмний каркас Spring створений Родом Джонсоном в 2003 році сьогодні є одним з найпопулярніших фреймворків для Java-платформи. На момент своєї появи він став відповіддю на складність розуміння специфікацій Jakarta EE який в той час мав назву J2EE. З часом ситуація зі складністю розуміння покращилась, але для роботи J2EE потрібна була ціла інфраструктура.

Spring на сьогодні в цілому можна вважати додатковою технологією до Jakarta EE, що дозволяє за допомогою впровадження залежностей швидше та з більшою зручністю створювати прикладні програми Java.

Фреймворк Spring розділений на модулі з яких є можливість обирати потрібні для додатку. В його основі лежать модулі основного контейнеру такі як (рис. 2.6): модель конфігурації і механізм впровадження залежностей. Несе він в собі такі технології, як servlet API (сервлети), WebSockets API (веб-сокети), засоби паралельної обробки, JSON Binding API, bean validation (перевірка достовірності даних), а також Java Persistence API (JPA, зберігання об’єктів в базі даних), Java Message Service (JMS, розсилка повідомлень), Java Transaction API та Java Connector Architecture.

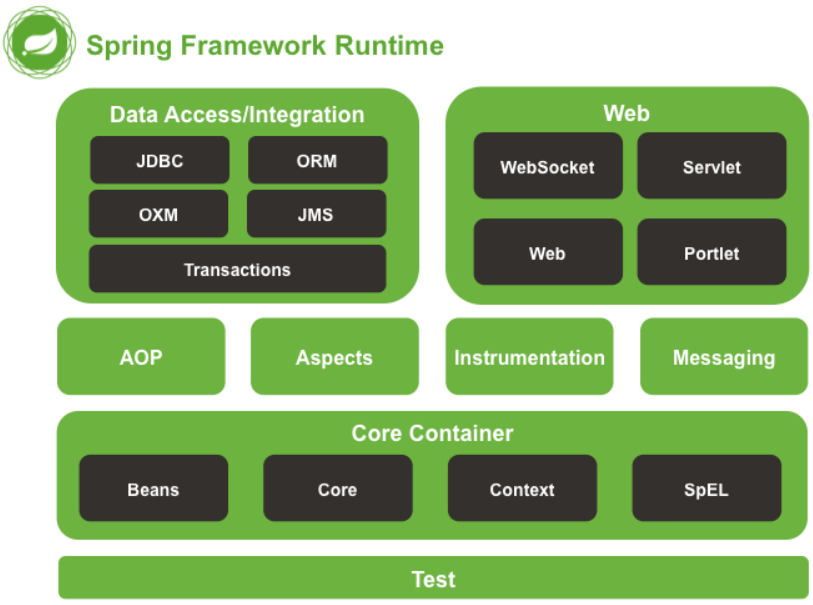

Рисунок 2.6 – Структура фреймворку Spring

Специфікація впровадження залежностей (DI, dependency injection), яку підтримує і для якої Spring виступає контейнером – це в першу чергу шаблон проектування, який призначений і використовується для створювання об’єктів поза класом, що їх використовує, і представлення йому цих об’єктів різними способами. При використанні DI увесь процес створення і прив’язки об’єктів, які потрібні класу для роботи виноситься за його межі. Для реалізації шаблону DI

зазвичай створюють класи-інжектори, що впроваджують об’єкти від яких залежить клас.

Також фреймворк Spring разом з впровадженням залежностей підтримує і стандартні анотації (common annotation), що разом дає значне спрощення розробки.

Наразі популярність Spring зумовлена не тільки тому, що зручніше і загалом працює, а і через те, що він постійно вводить нововведення, в числі яких є: Spring Boot, Spring Data, Spring Security.

Принципи фреймворку Spring:

— Можливість вибору на всіх рівнях. Spring дозволяє відкладати прийняття проектних рішень практично до самого кінця.
— Забезпечення строгої зворотньої сумісності. Контроль еволюції задля зменшення кількості критичних змін. Підтримка ретельно підібраного діапазону версій JDK і сторонніх бібліотек.
— Турбота про архітектуру API. Команда розробників вкладає багато зусиль і роздумів для створення інтуїтивно зрозумілих API.
— Високі стандарти якості коду. Spring Framework приділяє велику увагу змістовної, актуальною і точної документації javadoc. Це один з дуже небагатьох проектів, які можуть претендувати на чисту структуру коду без циклічних залежностей між пакетами.

2.6 Spring Boot

Spring Boot – це інструмент, що значно прискорює та спрощує розробку веб-додатків і мікросервісів та надає можливість їх запускати як звичайні програми написані мовою Java. З допомогою Spring Boot можливо створювати автономні додатки з вбудованим сервером, які повністю готові до запуску.

Завдяки суттєвому спрощенню створення готових до експлуатації додатків Spring Boot де-факто є основним способом створення Spring додатків.

Можливості Spring Boot:

- Дозволяє створювати додатки без XML-конфігурації не генеруючи ніякого коду.
- Надає додаткові способи налаштування подій та слухачів (listeners) додатків Spring.
- Надає продуману технологію, націлену на створення «правильного» типу додатків, як веб так і автономних.
- Має можливість за допомогою інтерфейсу ApplicationArguments отримувати доступ до будь-яких аргументів додатку. Це дозволяє передавати параметри в додаток під час запуску.
- Дозволяє виконувати код після запуску додатку. Для цього потрібно тільки реалізувати інтерфейс CommandLineRunner.
- Дозволяє використовувати зовнішні конфігурації з допомогою файлів application.properties та application.yml.
- Дозволяє використовувати функціональні можливості, зв’язані з адмініструванням за допомогою пункту spring.application.admin.enabled в файлі application.properties.
- Надає анотації @Enable<можливість>, які спрощують увімкнення, налаштування та використання таких технологій, як бази даних, хешування, планування виконання задач і обмін повідомленнями.
- Надає можливість використання профілів, завдяки яким додаток може працювати в різних середовищах.

Однією з найважливіших можливостей Spring Boot є автоматична конфігурація так, як саме вона відповідає за всі налаштування додатку відповідно з шляхом до класів, анотацій і усіх інших оголошених налаштувань.

2.7 Spring Data JPA

Spring Data JPA вважається одним з головних проектів у наборі інструментів Spring. Він базується і розширює JPA (Java Persistence API).

Однією з головних цілей Spring Data JPA є зменшення обсягу коду та спрощення рівня доступу до даних, зберігаючи при цьому багатий і розширений набір функцій. Щоб зробити це можливим, Spring Data JPA дозволяє створювати інтелектуальні, стереотипні інтерфейси Spring Repository. Ці сховища є інтерфейсами Java, які дозволяють вам, як розробнику, визначати контракт доступу до даних. Структура Spring Data JPA може потім перевірити цей контракт і автоматично створити для вас реалізацію інтерфейсу.

Для інтелектуальної генерації інтерфейсу репозиторію Spring Data JPA необхідний Query DSL. DSL розшифровується як Domain Specific Language. Спеціальна доменна мова запитів дозволяє створювати методи інтерфейсу Java, які використовують конкретні ключові слова разом із атрибутами сутності JPA для виконання роботи, необхідної для правильної реалізації ваших запитів, без необхідності надавати багато способів фактичного кодування.

2.8 Spring Security

Spring Security – це потужна та настроювана система аутентифікації та контролю доступу. Це фактичний стандарт захисту програм на основі Spring.

Spring Security – це структура, яка зосереджена на забезпеченні аутентифікації та авторизації програм Java. Як і у всіх проектах Spring, справжня сила Spring

Security полягає в тому, що його легко можна розширити для задоволення індивідуальних вимог.

Основні Функції:

— Комплексна та розширювана підтримка як аутентифікації, так і авторизації
— Захист від таких атак, як фіксація сеансу, розбивка кліків, підробка міжсайтових запитів тощо.
— Інтеграція API Servlet
— Додаткова інтеграція з Spring Web MVC

За своєю суттю Spring Security – це просто набір фільтрів сервлетів, які допомогають додати аутентифікацію і авторизацію в веб-додаток. Він також добре інтегрується з такими фреймворками, як Spring Web MVC (або Spring Boot) і зі стандартами OAuth2 або SAML. Spring Security автоматично генерує сторінки входу і виходу та захищає від поширених експлойтів таких як CSRF.

2.9 Maven

Maven – це безкоштовний інструмент для підтримки процесу збірки проекту розроблений Apache Software Foundation. Основою Maven є декларативний підхід до реалізації процесу збірки, який заснований на моделі. Він не реалізує індивідуальні цілі, а використовує так звану модель проекту, яка описує тільки метадані проекту і зазвичай знаходиться в файлі pom.xml. Метадані включають інформацію про структуру проекту і специфікації зовнішніх бібліотек.

Центральною точкою для визначення процесу складання проекту є його модель (pom.xml). Якщо потрібні бібліотеки існують в локальному репозиторії Maven буде використовувати їх, в іншому випадку, якщо це можливо від завантажить їх автоматично з віддаленого сховища.

Структура моделі проекту.

Елементи, що знаходяться в файлі-моделі pom.xml можна розділити на 4 частини:

— Базові елементи – це елементи, які зазвичай завжди присутні;
— Додаткова інформація про проект – елементи, що включають до себе повний опис проекту;
— Параметри побудови – це елементи в яких вказується структура каталогів проекту та плагіни;
— Налаштування середи.

Базові елементи включають до себе:

— modelVersion – в залежності від використаної версії Maven вказується версія моделі проекту;
— groupId – ідентифікатор користувача. Зазвичай використовують доменне ім’я в зворотному напрямку;
— artifactId – ім’я того, що буде на виході;
— version – версія проекту;
— packaging – вказується розширення файлу;
— dependencies – містить список залежностей (бібліотек) проекту;
— parent – використовується для наслідування проектів в Maven;
— dependencyManagement – використовується для допомоги в керуванні інформацією про залежності для усіх дочірніх елементів;
— modules – містить список модулей. Модулі - це проекти, перераховані в цьому файлі, що виконуються, як група. Представляють собою відносні шляхи до каталогів або файлів POM-файлів цих проектів;
— properties – містить список змінних які можна в подальшому використовувати в файлі pom за допомогою позначення $(X) де X це змінна.

До додаткової інформації про проект відносяться такі елементи:

— name – ім’я проекту;
— description – опис проекту;

— url – посилання на веб-сайт проекту;
— inceptionYear – рік зародження проекту;
— licenses – це список ліцензій проекту, що визначають, як і коли проект або його частини можуть бути використані;
— organization - включає до себе інформацію про організацію, що працює над проектом;
— developers – перераховуються розробники (програмісти) проекту. Зазвичай вказують тільки тих хто безпосередньо відповідає за код;
— contributors – вказуються співавтори проекту. Співавтори відіграють допоміжну роль у життєвому циклі проекту. Співавтором може вважатися наприклад той хто робив виправлення якої небудь помилки або додав відсутню інформацію в документацію. В проектах з відкритим кодом співавторів зазвичай більше ніж розробників.

До частини «Налаштування середи» відносяться такі елементи:

— issueManagement – визначає використовувану систему відстеження дефектів (Bugzilla, TestTrack, ClearQuest і т.д.). В основному ця інформація використовується для створення проектної документації;
— ciManagement - continuous Integration Management зазвичай використовується в системах, що запускають збірку проекту і використовуючи тригери або таймери повідомляє про статус збірки;
— mailingLists – містить список розсилок для підтримки зв’язку з людьми.

До частини «Параметри побудови» відносяться такі елементи:

— build;
— reporting - використовується для генерації сайту на якому можна переглядати звіти.

2.10 MySQL

MySQL - це система керування реляційними базами даних із відкритим кодом (СУБД), яку можна легко впровадити та керувати як локально, так і через хостинг-провайдера. Вона підтримує багато одночасних записів та масштабування з реплікацією (хоча це може бути складно). З цієї причини та через відносно низькі витрати на технічне обслуговування і масштабованість, вона в основному використовується як виробнича база даних.

MySQL була побудована, як рішення з відкритим кодом, тому її можна розгортати на готовому обладнанні та масштабувати за передбачуваною вартістю. Широка сумісність MySQL також означає, що її можна будь-коли замінити більш індивідуальним рішенням (багато конкуруючих баз даних сумісні з MySQL саме для цієї мети).

Розробники MySQL прийняли рішення надати пріоритет швидкості та продуктивності функціям, зробивши MySQL більш швидкою, хоча і обмеженою базою даних, ніж інші постачальники в категорії транзакційних баз даних.

MySQL з самого початку був розроблений для веб-додатків. Це особливо корисно для структурованих та добре спланованих веб-додатків.

Ця система часто є першим вибором для невеликих компаній або стартапів через свою надійність, найбільшу поширеність та ефективність.

Архітектура MySQL складається з декількох шарів:

— Шар додатків. Це те, як різні клієнти підключаються до MySQL та подають запити. Такі програми, як MySQL Workbench і Looker, підключаються до рівня додатків MySQL для запиту та контролю доступу користувачів до бази даних.
— Обробник запитів. Як тільки на рівні програми видається запит на читання або запис даних із базового сховища, процесор запитів перетворює запит у план запиту, який база даних може виконати.
— Управління транзакціями. MySQL відповідає ACID, що означає, що набір запитів може бути інкапсульований у транзакції, і всі вони досягають успіху

або не вдаються. Менеджер транзакцій контролює це, видаючи команду COMMIT для виконання кожної транзакції. Якщо транзакція не вдається, менеджер транзакцій видасть команду ROLLBACK, щоб скасувати будь-які зміни в базі даних, які раніше були зроблені в невдалій транзакції.

— Управління відновленням. MySQL відрізняється високою стійкістю, і менеджер відновлення відповідає за повернення бази даних до останнього стабільного стану у разі несправності. Він реєструє кожну операцію, виконану в базі даних (з моменту її створення), і в разі відмови виконує кожну команду в журналі, тим самим ефективно повертаючи базу даних до останнього стабільного стану.

— Управління сховищем. Менеджер зберігання відповідає за розподіл ресурсів пам'яті, необхідних для отримання даних з фізичного диска та доставки результатів клієнту.

2.11 Thymeleaf

Thymeleaf - сучасний серверний механізм Java-шаблонів для веб і автономних середовищ, здатний обробляти HTML, XML, JavaScript, CSS і навіть простий текст. Основною метою Thymeleaf є створення елегантного і зручного способу шаблонізації. Для досягнення цього, Thymeleaf ґрунтується на концепції Natural Templates, для впровадження своєї логіки в файли шаблонів таким чином, щоб це не впливало на відображення прототипу дизайну. Це покращує комунікацію в команді і зменшує розрив між дизайнерами та програмістами. Thymeleaf від самого початку розроблявся з урахуванням стандартів Web, особливо HTML5, що дозволяє створювати шаблони які повністю відповідають стандарту.

Thymeleaf може обробляти шість типів шаблонів, кожен з яких називається режимом шаблону (два режими шаблону макета (HTML і XML), три режими текстових шаблонів (TEXT, JAVASCRIPT і CSS), режим без використання шаблонів (RAW)):

— Режим шаблону HTML дозволяє вводити будь-який тип HTML, включаючи HTML5, HTML 4 та XHTML.
— Режим шаблону XML дозволяє вводити XML.
— Режим шаблону TEXT дозволяє спеціальний синтаксис для шаблонів без розмітки. Прикладами таких шаблонів є текстові повідомлення електронної пошти або шаблонна документація.
— Режим шаблону JAVASCRIPT дозволить обробляти файли JavaScript
— Режим шаблону CSS дозволить візуалізувати файли CSS
— Режим RAW-шаблону просто не обробляє шаблони. Він призначений для вставки незмінних ресурсів (файлів, відповідей URL-адрес тощо) в оброблені шаблони.

У веб-додатках Spring MVC контролер повертає представлення (шаблон Thymeleaf), які в результаті відображатимуться у браузері.

2.12 JavaScript

Javascript - це мова, заснована на стандарті ECMAScript. Javascript - мова зі слабкою типізацією, це означає, що вам не потрібно визначати типи виводу, коли ви пишете на ній код. Javascript визначить, які типи ви намагаєтесь використовувати, залежно від контексту, в якому вони використовуються.

Javascript має дві основні реалізації - він вбудований у ваш браузер, але різні браузери по-різному підтримують нові функції Javascript. Тому деякі нові функції Javascript підтримуються не всіма браузерами. Друга реалізація - через Node.JS. Знову ж таки, Node.JS може не підтримувати всі нові функції Javascript відразу, але зазвичай він встановлюється на серверах, а не в браузерах.

Javascript дозволяє робити багато речей. Для інтерфейсу деякі приклади цього включають:

— додавати або змінювати CSS в документі HTML;

— створювати нові елементи HTML програмно;

— відстежувати, що робить користувач, і реагувати;

— змінювати або видаляти теги HTML, тобто змінювати класи, які має елемент;

— зберігати дані в об’єктах та масивах.

На сервері все інакше. Ви можете зробити щось на зразок:

— створювати цілі веб-сайти та кодувати те, що відбувається за певними URL-адресами.

— створювати сервери, до яких користувачі можуть підключатися.

— створювати реальні API та веб-сокети, до яких користувачі можуть підключатися.

— обробляти стиснення файлів, щоб пришвидшити роботу.

— керувати даними для зберігання в базах даних.

ВИСНОВКИ

В цьому розділі демонструються технології за допомогою яких буде розроблено Web-додаток. Було розглянуто засоби реалізації, які використовуються для розробки:

— клієнтської частини: HTML, CSS, JavaScript;
— серверної частини: Java, Spring Boot, Spring Data JPA, Spring Security, MySQL, Thymeleaf.

Було обрано такі засоби розробки програмних продуктів, які б полегшили роботу програміста, надавши всі необхідні інструменти для реалізації поставленого завдання.

3 ОПИС РОЗРОБЛЕНОГО ВЕБ-ДОДАТКУ

Для розробки серверної частини веб-додатку використовується Spring Framework, його складові Spring Boot, Spring Data JPA та Spring Security для спрощення та прискорення процесу розробки.

На початку роботи проведена конфігурація проекту, яка торкнулася налаштування підключення до бази даних MySQL, в якій будуть зберігатися усі сутності, що використовуватимуться.

Контейнером в якому запускатиметься створений додаток буде Apache Tomcat, який розгортається фреймворком Spring Boot в процесі запуску.

Клієнтська частина розроблена використовуючи чистий CSS3 та мову розмітки HTML5 з невеликим використанням JavaScript.

Структура додатку побудована з використанням схеми розділення даних – MVC, яка передбачає створення окремих компонентів такі як: модель, представлення та контролер. Як зображено на рисунку 3.1 структура каталогів логічно розділена на контролери (каталог controller), сутності (каталог entity) та представлення (каталоги static та templates за шляхом src/main/resources). Каталог static включає до себе елементи, що не змінюються в процесі роботи додатку, загалом це каскадні таблиці стилей (CSS) та зображення. В каталозі templates зберігаються шаблони сторінок розбиті на окремі фрагменти, які в процесі роботи додатку за допомогою шаблонізатора thymeleaf формуються в потрібні сторінки. Таке розділення на фрагменти дозволяє повторно використовувати елементи сторінок без зайвого копіювання.

Також в представленому зображенні слід звернути увагу на каталог repository. В цьому каталозі зберігаються інтерфейси репозиторіїв, що інкапсулюють набори об’єктів, збережених в базі даних і операції, які можна виконувати з цими об’єктами. Приклад такого репозиторію зображений на рисунку 3.2.

Інтерфейси репозиторіїв наслідують основні свої методи від інтерфейсу CrudRepository. Це такі методи як:

— save – зберігає переданий об’єкт в базі;

— delete – видаляє знайдений по ID об’єкт з бази;

— deleteAll – видаляє усі об’єкти з бази;

— count – повертає кількість об’єктів;

— findAll – знаходить та повертає усі об’єкти в базі;

— exists – перевіряє за ID чи наявний відповідний об’єкт в базі;

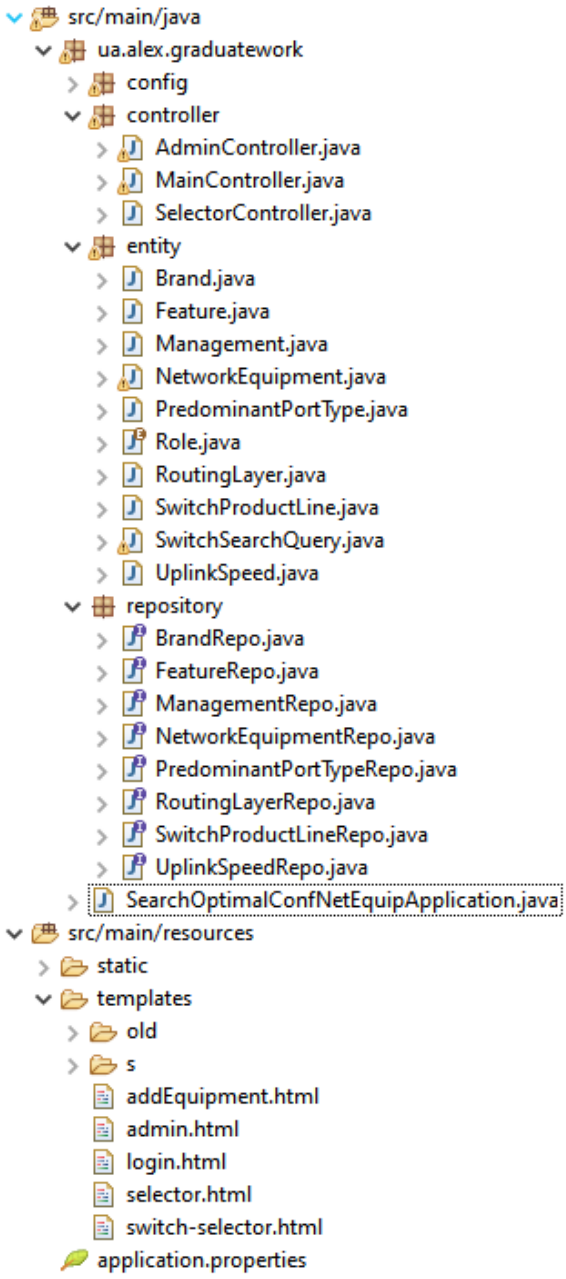

Рисунок 3.1 – Структура каталогів проекту

```java
package ua.alex.graduatework.repository;

import java.util.List;

public interface NetworkEquipmentRepo extends CrudRepository<NetworkEquipment, Integer>{

    List<NetworkEquipment> findByBrand(String brand);

    List<NetworkEquipment> findByProductLine(String productLine);

    List<NetworkEquipment> findByManagement(String management);

    List<NetworkEquipment> findByRouting(String routing);

    List<NetworkEquipment> findByPortType(String portType);

    List<NetworkEquipment> findByName(String name);

    List<NetworkEquipment> findByDescription(String description);

    List<NetworkEquipment> findByPortCount(int portCount);

    List<NetworkEquipment> findByFeaturesIn(List<String> features);

    List<NetworkEquipment> findByUplinkTypeIn(List<String> uplinkType);

}
```

Рисунок 3.2 – Приклад репозиторію

3.1 Сутності

Сутності бази даних створюються перш за все і описують стовпці таблиць та типи зв’язків між цими таблицями. Вони описуються як звичайний POJO-клас з тою відмінністю, шо поля в них позначені анотаціями наданими фреймворком Spring. Анотації дозволяються зв’язати поля об’єктів з таблицями бази даних та їх атрибутами. На рисунку 3.3 відображено лістинг класу NetworkEquipment в якому можна побачити ці анотації. Більшість з них описує зв’язок типу один-до-одного (анотація @OneToOne) між таблицями.

Таким же або схожим способом імплементовані всі сутності додатку, що формують модель даних.

```java
@Entity
@Table(appliesTo = "networkEquipments")
public class NetworkEquipment {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;

    @OneToOne(fetch = FetchType.EAGER)
    private Brand brand;

    @OneToOne(fetch = FetchType.EAGER)
    private SwitchProductLine productLine;

    @OneToOne(fetch = FetchType.EAGER)
    private Management management;

    @OneToOne(fetch = FetchType.EAGER)
    private RoutingLayer routing;

    @OneToOne(fetch = FetchType.EAGER)
    private PredominantPortType portType;

    private String name;

    private String description;

    private int portCount;

    @ElementCollection
    private List<String> features;

    @ElementCollection
    private List<String> uplinkType;

    private String linkToDetails;

    private String fileName;

    public NetworkEquipment(String name, String description,
            List<String> features, List<String> uplinkType, String linkToDetails, String fileName) {
        this.name = name;
        this.description = description;
        this.setFeatures(features);
        this.setUplinkType(uplinkType);
        this.linkToDetails = linkToDetails;
        this.setFileName(fileName);
    }
```

Рисунок 3.3 – Лістинг класу NetworkEquipment

Основні сутності додатку:

— NetworkEquipment, яка представляє комутатори

— RouterNetworkEquipment – маршрутизатори

Допоміжні сутності:

— Brand

— Feature

— Management

— PredominantPortType

— RoutingLayer

— UplinkSpeed

3.2 Контролери

Контролери, як і сутності створюються в першу чергу. Вони описують поведінку додатку при переході користувача по певній URL-адресі. Саме в контролерах визначається на які посилання користувач може переходити та що він отримає від сервера при переході. В контролері обробляються запити від клієнтської частини такі як: GET, POST, DELETE, PATCH, . Приклад контролера зображено на рисунку 3.4.

```
27
28  @Controller
29  @RequestMapping("/admin")
30  public class AdminController {
31      @Autowired
32      private BrandRepo brandRepo;
33
34      @Autowired
35      private FeatureRepo featureRepo;
36
37      @Autowired
38      private ManagementRepo managementRepo;
39
40      @Autowired
41      private NetworkEquipmentRepo networkEquipmentRepo;
42
43      @Value("${upload.path}")
44      private String uploadPath;
45
46      @GetMapping
47      public String admin(Model model) {
48
49          Iterable<Brand> brands = brandRepo.findAll();
50          Iterable<Feature> features = featureRepo.findAll();
51          Iterable<Management> managements = managementRepo.findAll();
52
53          model.addAttribute("brandList", brands);
54          model.addAttribute("features", features);
55          model.addAttribute("managements", managements);
56
57          return "admin";
58      }
59
60      @PostMapping("/add")
61      public String addEquipment(
62              @RequestParam String name,
63              @RequestParam("file") MultipartFile file,
64              @RequestParam String brand,
65              @RequestParam String productLine,
66              @RequestParam String management,
67              @RequestParam String routing,
68              @RequestParam String portType,
69              @RequestParam List<String> uplinkType,
70              @RequestParam List<String> features,
71              @RequestParam String portCount,
72              @RequestParam String text,
73              Model model
74              ) throws IllegalStateException, IOException {
```

Рисунок 3.4 - Лістинг контролера адмін панелі

Запити поступають до контролера після того як їх обробить DispatcherServlet.

Після обробки прийнятого HTTP запиту DispatcherServlet знаходить необхідний контролер в якому створені методи, що обробляють прийнятий запит. Такі методи

позначаються анотацією @GetMapping або @PostMapping та реалізують логіку реакції серверу відповідно на GET та POST запити.

3.3 Опис структури бази даних

Модель бази даних створюється автоматично за допомогою фреймворку Spring Data JPA. Він використовуючи об’єкти сутностей та репозиторії, описаних мовою Java створює в базі даних таблиці та запити для цих таблиць. Таким чином без великого обсягу маніпуляцій та написання SQL коду отримуємо в підсумку повноцінну структуру бази даних (рис. 3.5)

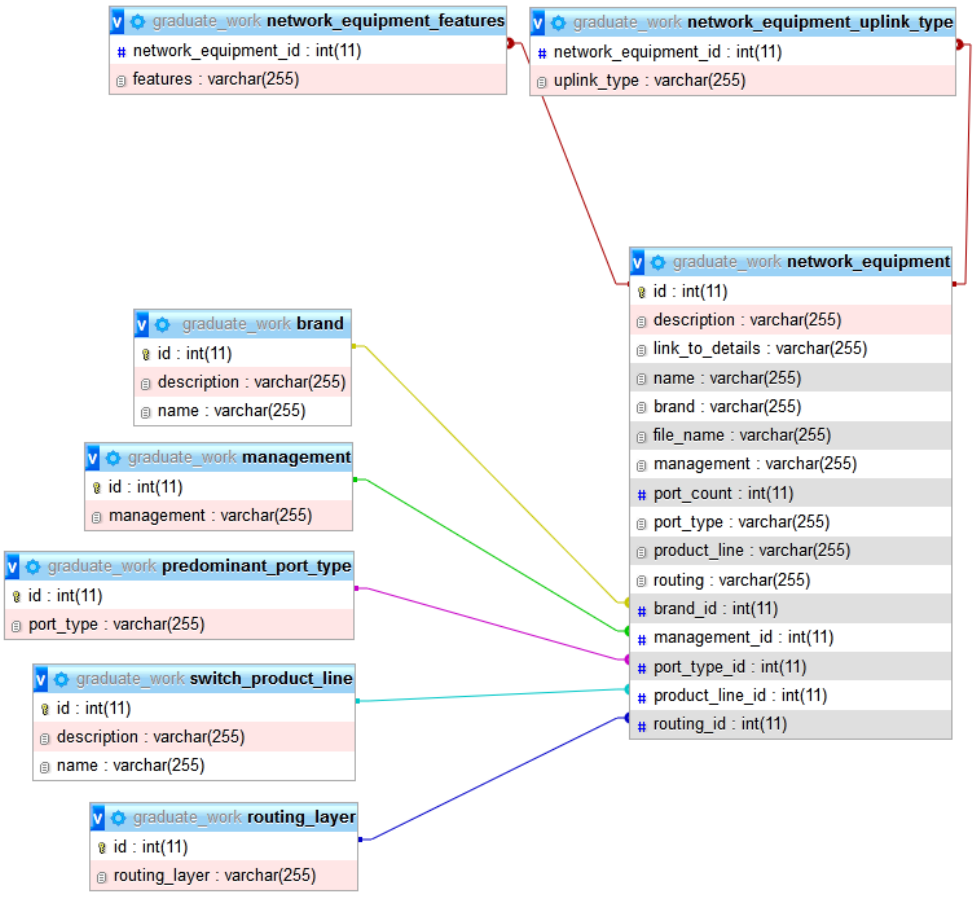

Рисунок 3.5 – Структура бази даних

3.4 Функціонал клієнтської частини

3.4.1 Панель керування

В панелі керування знаходяться три основні вкладки за допомогою яких адміністратор може додавати нове обладнання до бази, додавати критерії пошуку, переглядати список обладнання, що вже наявне в базі.

На рис. 3.6 зображено сторінку панелі керування з обраною вкладкою «Додати обладнання». В цій вкладці знаходяться поля вводу інформації про обладнання. Є можливість ввести назву обладнання, обрати зображення, виробника, лінійку продуктів до якої належить обладнання, спосіб керування, шар маршрутизації, кількість портів, переважаючий тип порту, особливості, швидкість каналу зв’язку та додати опис.

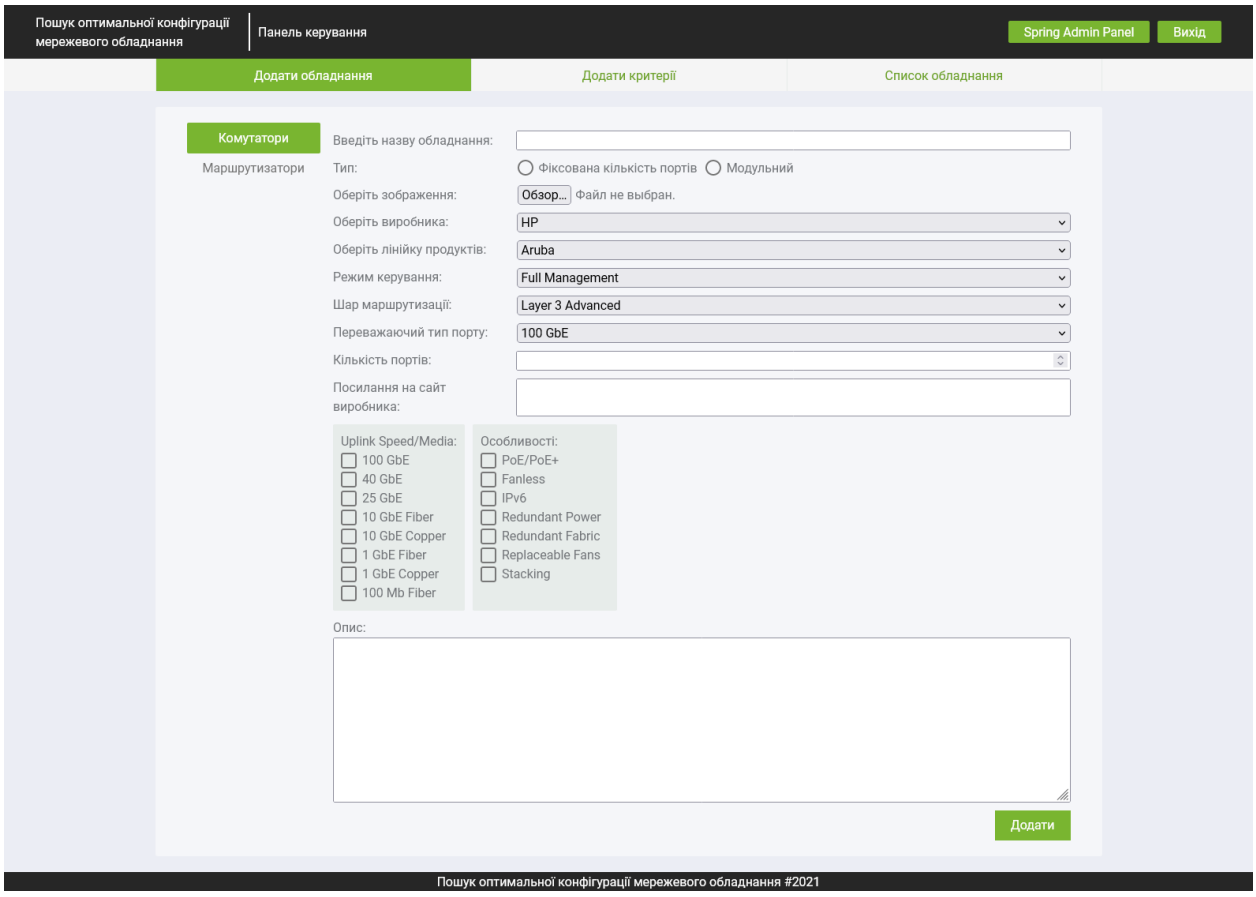

Рисунок 3.6 – Панель керування вкладка «Додати обладнання».

У вкладці «Додати критерії» (рис. 3.7) панелі керування адміністратор має можливість додавати до бази нові лінії продуктів, типи порту, швидкості каналу зв’язку та особливості, які в подальшому будуть відображатися в відповідних списках. Усі вони мають однакові поля для вводу (назва та опис).

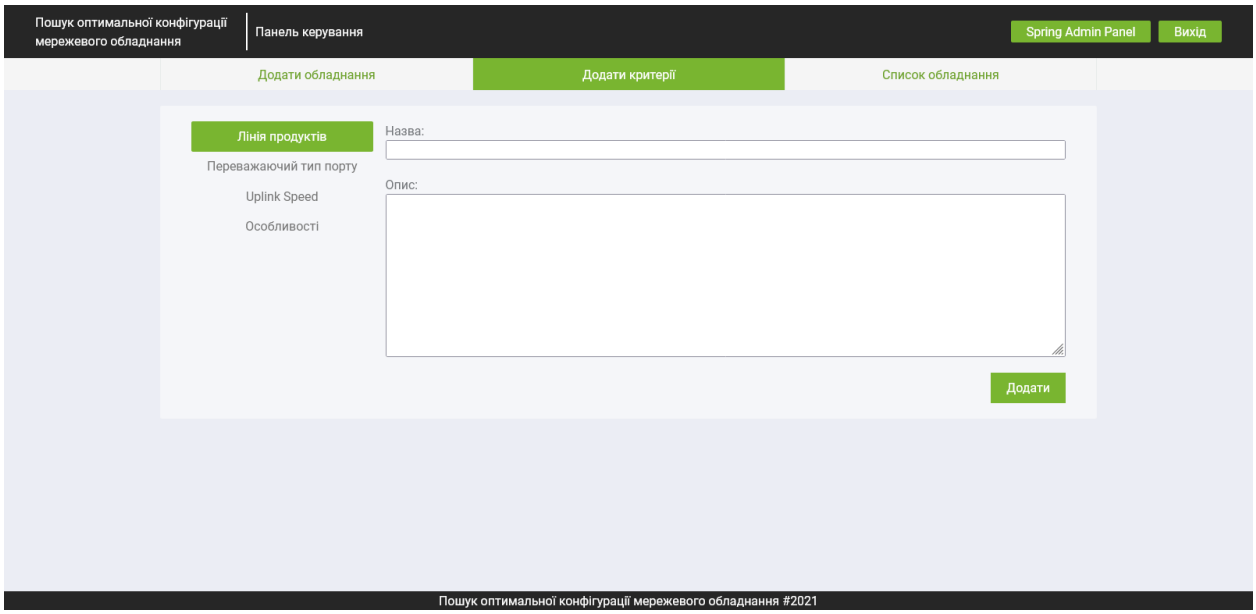

Рисунок 3.7 – Вкладка «Додати критерії»

В вкладці «Список обладнання» (рис. 3.8) відображається все обладнання, що знаходиться в базі даних. Список виконаний у вигляді таблиці. Таблиця має стовпці: ідентифікатор, назва, та взаємодія. В стовпці взаємодія знаходяться кнопка перегляду детальної інформації про обладнання, кнопка редагування та видалення.

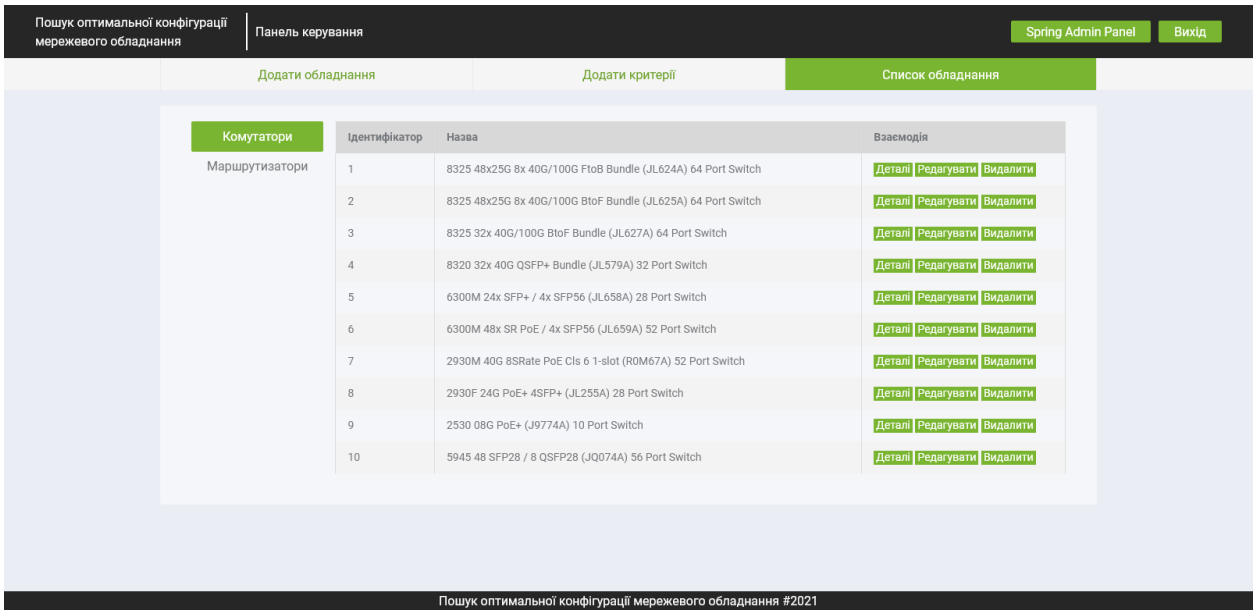

Рисунок 3.8 – Панель керування вкладка «Список обладнання»

3.5 Сторінка пошуку обладнання

На сторінці пошуку обладнання (рис. 3.9) основним блоками виступають фільтр за параметрами якого повинен виконуватися пошук та список знайденого обладнання. В списку відображається назва та зображення. За допомогою кнопки «Порівняти» можна перейти на сторінку порівняння (рис. 3.10) і серед обраних елементів списку обрати найбільш підходящій. На сторінці також є інформація про обладнання в окремому блоці який можна розгорнути.

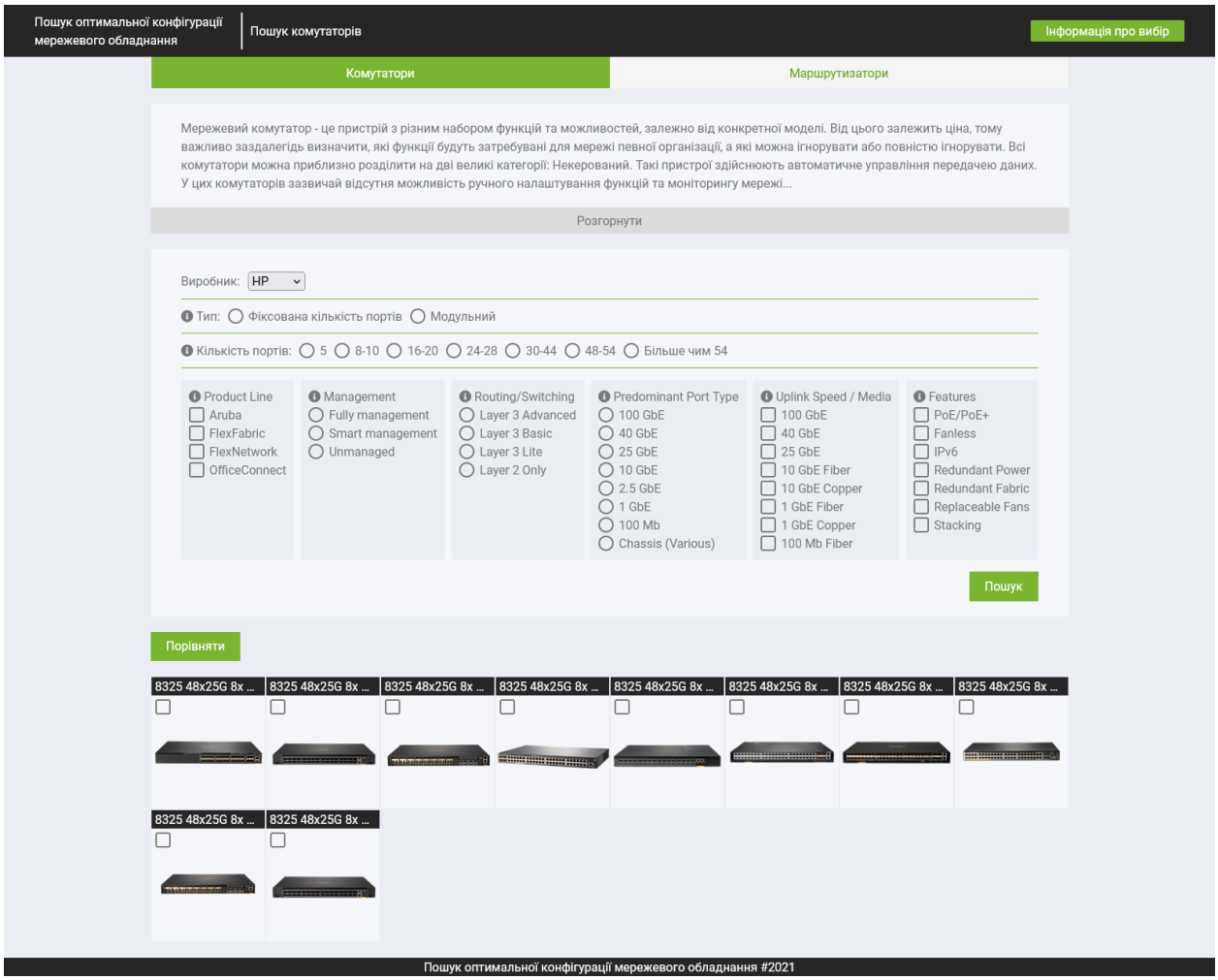

Рисунок 3.9 – Сторінка пошуку обладнання

При натисканні на обладнання із списку відкривається вікно з більш детальною інформацією (рис. 3.11) де можна ознайомитися з описом та характеристиками.

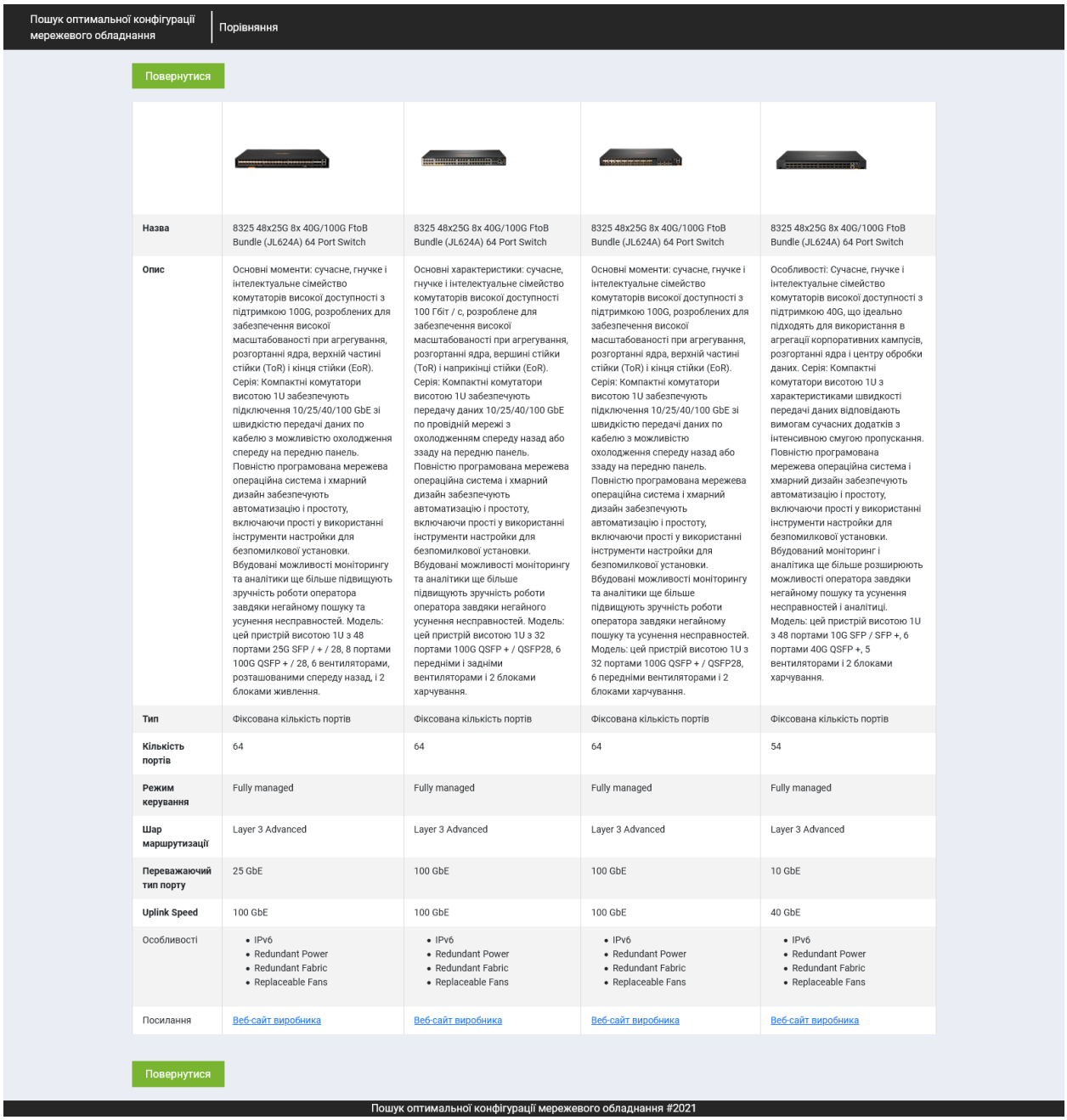

Рисунок 3.10 – Сторінка порівняння

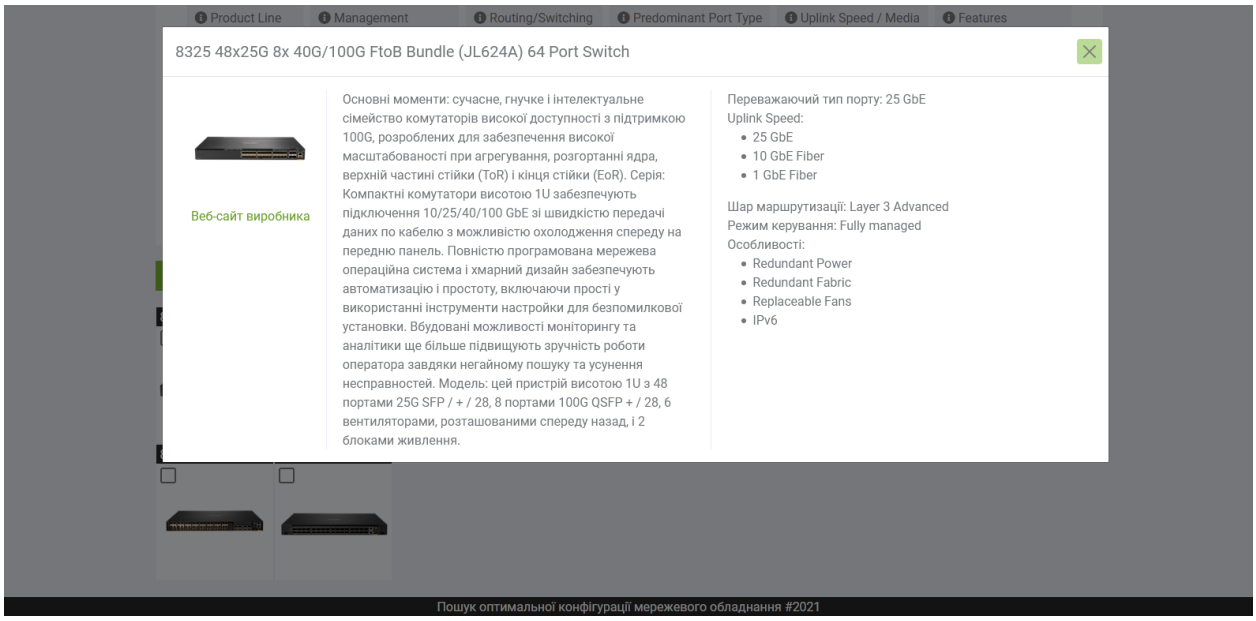

Рисунок 3.11 – Вікно з описом обладнання

ВИСНОВКИ

Головною метою дипломної роботи є досягнення поставленої мети та виконання завдань проекту, а саме розробка веб-додатку по пошуку оптимальної конфігурації мережного обладнання та закріплення і систематизація теоретичних знань набутих в процесі навчання на його прикладі.

В ході виконання даної роботи були проведені огляд та аналіз області мережевих рішень і було визначено критерії вибору оптимальної конфігурації мережного обладнання, на основі яких було розроблено функціонал веб-додатку. В роботі розглянуто основні теоретичні відомості, знання яких необхідні для розуміння роботи веб-додатків, проведено аналіз сучасних підходів та проектних рішень щодо проектування і створення веб-додатків. Проведено огляд клієнт-серверної архітектури та основні шари, на які розділені сучасні системи, що створені на основі цієї архітектури.

Для реалізації серверної частини додатку були застосовані такі засоби, як фреймворк Spring Boot, Spring Data JPA, Maven, Spring Security та платформа Java. Для клієнтської частини використовувалися HTML, CSS та JavaScript. Для зберігання даних внесених до додатку була налаштована та підключена реляційна система контролю базами даних MySQL.

У третьому розділі продемонстровано функціонал розробленого додатку та основні моменти його роботи.

Розроблений веб-додаток задовольняє потреби користувачів у пошуку оптимальної конфігурації мережного обладнання і таким чином зумовлює вірний вибір необхідного устаткування, який дозволить не тільки оптимізувати співвідношення якості та вартості, але зможе вберегти від значних проблем в ході експлуатації.

Завдяки модульній архітектурі розроблений додаток можливо в подальшому розширити, додаючи нові типи обладнання та характеристик.