МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____Аліна Савченко

«_____»_____2022 р.

ДИПЛОМНИЙ ПРОЕКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСНИКА ОСВІТНЬОГО СТУПЕНЯ "БАКАЛАВР"

ЗА СПЕЦІАЛЬНІСТЮ 122 «КОМП'ЮТЕРНІ НАУКИ»

Тема: «Web-сайт інтернет-магазин»

Виконавець: Гречка Кирило Сергійович

Керівник: к.т.н. доц. Толстікова Олена Володимирівна

Нормоконтролер: к.т.н. доц. Боровик Володимир Миколайович

КИЇВ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *кібербезпеки, комп'ютерної та програмної інженерії*
Кафедра *комп'ютерних інформаційних технологій*
Освітній ступінь *Бакалавр*
Спеціальність *122 «Комп'ютерні науки »*
Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Аліна Савченко
«_____»_____2022 р.

ЗАВДАННЯ
на виконання дипломного проекту

студента: *Гречка Кирила Сергійовича*

1. Тема дипломного проекту *«Web-сайт інтернет-магазин»*
затверджена наказом ректора від « 29 » квітня 2022 р. № 454/ст
2. Термін виконання проекту: з 09.05.2022 р. по 19.06.2022 р.
3. Вихідні дані до проекту: ознайомлення з програмним забезпеченням для виконання роботи. Аналіз сьогоденнішніх актуальних технологій для розробки вебсайту. Проектування та розробка проекту відповідно до обраної теми.
4. Зміст пояснювальної записки: підготовка теоретичного матеріалу для виконання роботи.

Розділ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.

Розділ 2. ПРОЕКТУВАННЯ СТРУКТУРИ WEB-САЙТУ.

Розділ 3. РОЗРОБКА WEB-САЙТУ.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу: слайди презентації засобами MS Power Point.

6. Календарний план-графік

№ п/п	Етапи виконання дипломного проекту	Термін виконання етапів	Примітки
1	Визначення теми дипломного проекту.	09.05.2022-10.05.2022	
2	Формування предметної області.	10.05.2022 - 15.05.2022	
3	Аналіз та розробка структури дипломного проекту.	15.05.2022 - 19.05.2022	
4	Налаштування серверів для обробки даних.	19.05.2022 - 23.05.2022	
5	Розробка front-частини сайту.	23.05.2022 - 31.05.2022	
6	Розробка back-частини сайту.	31.05.2022 - 10.06.2022	
7	Розробка “шлюзу” для взаємодії мікросервісів.	10.06.2022 - 13.06.2022	
8	Оформлення пояснювальної записки	13.06.2022 - 17.06.2022	
9	Підготовка та захист дипломного проекту.	17.06.2022 - 19.06.2022	

7. Дата видачі завдання: 9 травня 2022 р.

Керівник дипломного проекту _____ Толстікова Олена Володимирівна

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Гречка Кирило Сергійович

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Web-сайт інтернет-магазин» 46 сторінок, 23 рисунків, 14 використаних джерела.

JAVA, POSTGRESQL, MONGODB, VUE, SPRING BOOT, LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL, ACTIVE DIRECTORY, KEYCLOAK, JAVASCRIPT, CSS, HTML, MAVEN, GRADLE, NODE MODULES, GIT, GITHUB, MICROSERVICES, IDEA.

Об’єкт дослідження – інтернет-магазини та його мікросервісна структура.

Мета дипломного проекту – отримати кінцевий продукт у вигляді робочого інтернет-магазину для різної тематики.

Основна мета поділена на допоміжні завдання: підготовка теоретичного матеріалу по роботі з використанням різних фреймворків та інших інструментів для свторення сайту.

Для цього, необхідно дослідження та розробка мікросервісної архітектури, аналіз та використання наявних актуальних технологій для створення вебсайтів та забеспечення ефективної роботи інтернет-магазину.

В результаті виконання дипломного проекту був розроблений вебсайт на основі мікросервісної архітетури з використанням актуальних технологій.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

LDAP	-	Lightweight directory access protocol
AD	-	Access directory
HTTP	-	Hypertext transfer protocol
DNS	-	Domain name system
API	-	Application programming interface
IDE	-	Integrated development environment
CTA	-	Call to click

ВСТУП

Вебсайт – сукупність логічно зв’язаної гіпртекстової інформації, оформленої у виголяді окремих сторінок і лоступної в мережі інтернет. Сайт на основі мікросервісної архітектури. Якщо коротко, то архітектурний стиль мікросервісів - це підхід, при якому єдиний додаток будується як набір невеликих сервісів, кожен з яких працює у своєму процесі та комунікує з іншими використовуючи легковажні механізми, як правило, HTTP. Ці послуги побудовані навколо бізнес-потреб і розгортаються незалежно з використанням повністю автоматизованого середовища.

Архітектура мікросервісів використовує бібліотеки, але їхній основний спосіб розбиття програми - шляхом розподілу його на послуги. Ми визначаємо бібліотеки як компоненти, які підключаються до програми та викликаються нею в тому ж процесі, у той час як сервіси – це компоненти, що виконуються в окремому процесі та комунікують між собою через веб-запити або remote procedure call (RPC).

Таким чином мікросервісний підхід до розбиття передбачає розбиття на послуги відповідно до потреб бізнесу. Такі сервіси включають повний набір технологій, необхідних для цієї бізнес-потреби, в тому числі інтерфейс користувача, сховищі даних і будь-які зовнішні взаємодії.

Вирішення проблем за рахунок фреймворків. Сучасні сайти, додатки, сервіси стають все більш складними, динамічними, багатофункціональними. Вони повинні активно взаємодіяти з користувачем, бути здатними підлаштовуватися під мінливе середовище, що обумовлює постійний розвиток методів їх розробки, підтримки. Прагнучи відповідати зростаючим потребам ринку IT-технологій, фахівці розробили новий інструмент – web framework. Він являє собою каркас, платформу для створення веб-продуктів нового покоління, їх ефективної підтримки. Він призначений для складних, масштабних проектів, дозволяє реалізувати нестандартні рішення.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Мікросервісна архітектура

Архітектура мікросервісів складається з невеликого набору автономних сервісів. Кожна послуга має реалізувати єдину бізнес-можливість в автономному та обмеженому контексті. Обмежений контекст забезпечує відкриті межі з природним поділом бізнесу та предметною моделлю (рис. 1.1).

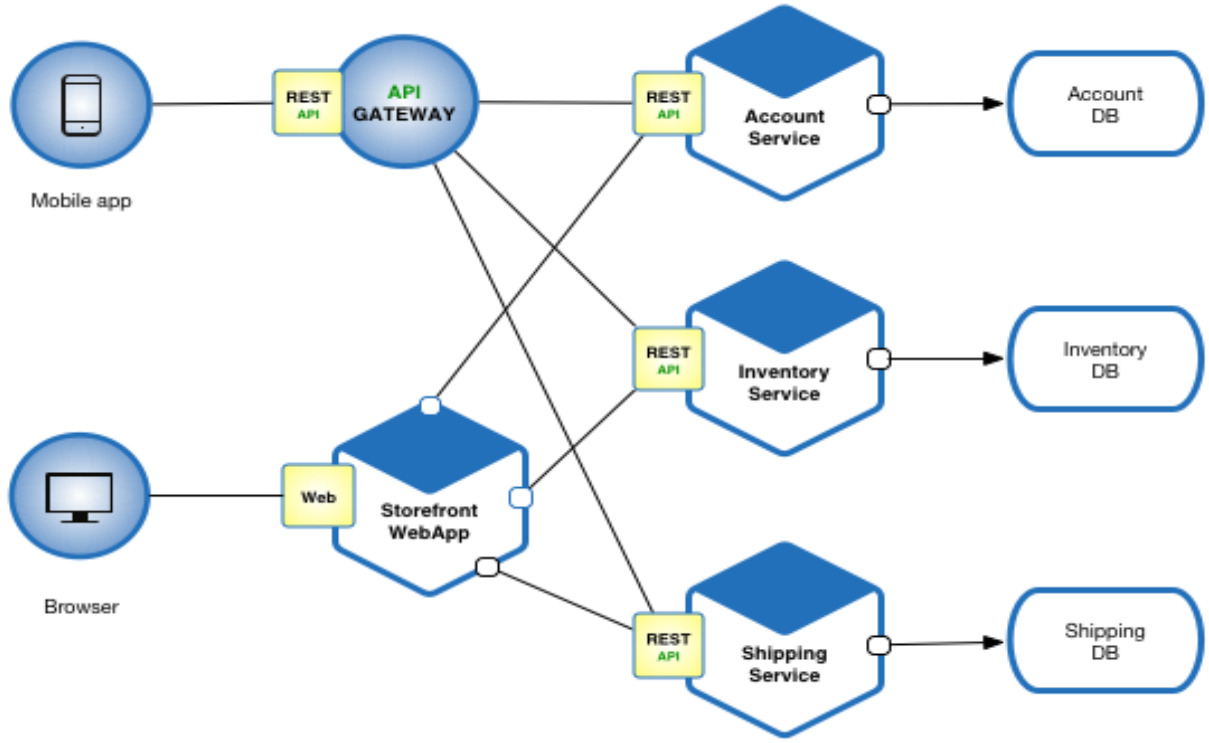

Рис. 1.1. Вигляд мікросервісної архітектури

Що таке мікросервіси?

1. Мікросервіси являють собою незалежні, невеликі та слабо пов'язані компоненти. Невелика команда розробників може писати й підтримувати сервіс.
2. Кожен сервіс має свою базу коду, яку підтримують та розробляють невелика команда розробнкиів.

3. Послуги можна надавати самостійно. Команда може оновлювати існуючу службу без необхідності перебудовувати й перерозгортати всю програму.
4. Служби відповідають за збереження власних даних або зовнішнього статусу. Це відрізняється від традиційної моделі, коли один рівень даних обробляє а інший зберігає дані.
5. Служби взаємодіють один з одним за допомогою чітко визначених API. Внутрішні деталі реалізації кожної служби приховані від інших служб.
6. Підтримує програмування поліглотів. Наприклад, службам не потрібно використовувати один і той самий стек технологій, бібліотеки чи фреймворки.

Переваги використання мікросервісної архітектури:

1. Покращена масштабованість.
2. Краща ізоляція несправностей і більш стійкі додатки.
3. Агностик мови програмування та технологій.
4. Кращий захист даних і відповідність вимогам.
5. Швидший час виходу на ринок і «забезпечення майбутнього».
6. Більша гнучкість бізнесу та підтримка DevOps.
7. Підтримка команд розробників Two-Pizza.

Архітектура програми мікросервісів призводить до поділу монолітної програми на її складові сервісні функції. Після визначення окремих сервісів розробники перебудовують моноліт так, щоб кожен сервіс функціонував автономно як окремий «мікросервіс». Потім вони вільно з’єднують ці мікросервіси через API, щоб створити більшу програму на основі мікросервісів.

Отримана програма базується на мікросервісах, які пропонують використання архітектурного стилю. Цей стиль пропонує швидкі та недорогі оновлення. Недорого і легко зменшити частину програми. Ви можете впливати на решту програми.

Розробники в створенні мікросервісів у контейнері на хмарному сервері, наприклад Amazon AWS використовують інструменти оркестрування контейнерів, такі як Kubernetes, щоб ефективно розподілити потужність обробки та інші ресурси для контейнерів за потреби.

1.2. Інструменти для розробки

Так як в основі вебсайту лежить мікросервісна архітектура, то доцільно використовувати різні технології для різних частин проетку. Для розробки даного проекту було обрано два основні фреймворки для front-end – Vue.js, як швидкий та багаторазове використання компонентів, та для back-end – Spring Boot, як потужний фреймворк для швидкої конфігурації з коробки та найбільш простішим у розробці.

1.2.1. Фреймворки

1.2.1.1. Vue

Vue — це прогресивна структура для створення інтерфейсів користувача. Він розроблений з нуля, щоб його можна було поступово засвоювати, і може легко масштабуватися між бібліотекою та фреймворком залежно від різних випадків використання. Він складається з доступної основної бібліотеки, яка зосереджена лише на рівні перегляду, та екосистеми допоміжних бібліотек, яка допомагає вам вирішувати складність у великих односторінкових програмах.

Основні переваги використання цього фреймворка:

1. Простота.
2. Архітектура на базі компонентів.
3. Можливість багаторазового використання.
4. Висока ефективність.
5. Документація.
6. Кроссплатформенність.

7. Фреймворки та бібліотеки.
8. Virtual-DOM.
9. Реактивне двостороннє прив’язування даних.
10. Декларативне оформлення.

1.2.1.2. Spring Boot

Spring Boot дозволяє легко створювати автономні додатки на основі Spring, які можна «просто запускати».

Розробники дотримуються упевненого погляду на платформу Spring та сторонні бібліотеки, щоб ми могли розпочати роботу з мінімальною затратою. Більшість програм Spring Boot потребують мінімальної конфігурації Spring.

Особливості:

1. Створюйте окремі програми Spring.
2. Вставте Tomcat, Jetty або Undertow безпосередньо (не потрібно розгортати файли WAR).
3. Надайте упевнені залежності «початківця», щоб спростити конфігурацію збірки.
4. За можливості автоматично налаштовуйте бібліотеки Spring та сторонніх розробників.
5. Надайте готові для виробництва функції, такі як показники, перевірки справності та зовнішні конфігурації.
6. Абсолютно без генерації коду та без вимог до конфігурації XML.

1.2.2. База даних

PostgreSQL, використовується як реляційна база даних для зберігання товарів, їх опис, відношення користувачів до товарів, в той час MongoDB зберігає картинки, які відносяться до певного типу товару.

1.2.2.1. PostgreSQL

PostgreSQL — це потужна об'єктно-реляційна система баз даних з відкритим вихідним кодом з більш ніж 30-річною активною розробкою, завдяки якій вона заслужила міцну репутацію за надійність, стійкість функцій і продуктивність.

В офіційній документації можна знайти велику кількість інформації, яка описує, як встановити та використовувати PostgreSQL. Спільнота PostgreSQL пропонує багато корисних місць, щоб ознайомитися з технологією, дізнатися, як вона працює, і знайти можливості для кар'єрного зростання.

Переваги:

1. PostgreSQL може запускати динамічні вебсайти та веб-програми як варіант стека LAMP.
2. Попереднє ведення журналу PostgreSQL робить його високовідмовостійкою базою даних.
3. Вихідний код PostgreSQL вільно доступний під ліцензією з відкритим вихідним кодом. Це дає вам свободу використовувати, змінювати та впроваджувати його відповідно до потреб вашого бізнесу.
4. PostgreSQL підтримує географічні об'єкти, тому ви можете використовувати його для служб на основі розташування та географічних інформаційних систем.
5. PostgreSQL підтримує географічні об'єкти, тому його можна використовувати як сховище геопросторових даних для служб, що базуються на місцезнаходження, і геоінформаційних систем.
6. Щоб вивчити Postgres, вам не потрібно багато навчання, оскільки його легко використовувати.
7. Низький рівень обслуговування та адміністрування як для вбудованого, так і для корпоративного використання PostgreSQL.

1.2.2.2. MongoDB

MongoDB — програма керування базами даних NoSQL з відкритим вихідним кодом. NoSQL використовується як альтернатива традиційним реляційним базам даних. Бази даних NoSQL досить корисні для роботи з великими наборами розподілених даних. MongoDB — це інструмент, який може керувати документально-орієнтованою інформацією, зберігати або отримувати інформацію.

MongoDB пропонує багато переваг перед традиційними реляційними базами даних:

1. Повна хмарна платформа даних додатків
2. Гнучкі схеми документів
3. Широко підтримуваний доступ до даних із власним кодом
4. Зручний для змін дизайн
5. Потужні запити та аналітика
6. Легке горизонтальне масштабування з фрагментуванням
7. Проста установка
8. Економічно ефективним
9. Повна технічна підтримка та документація

MongoDB зберігає дані в гнучких документах, подібних до JSON, тобто поля можуть відрізнятися від документа до документа, а структура даних може змінюватися з часом.

Модель документа зіставляється з об’єктами в коді програми, що полегшує роботу з даними.

Спеціальні запити, індексація та агрегація в реальному часі надають потужні способи доступу та аналізу ваших даних.

MongoDB по суті є розподіленою базою даних, тому висока доступність, горизонтальне масштабування та географічний розподіл вбудовані та прості у використанні.

1.2.3. Активний каталог (AD)

Active directory підійде для зберігання користувачів.

Active Directory — це реалізація служб каталогів, що надає всі види функцій, як-от аутентифікація, керування групами та користувачами, адміністрування політики тощо.

AD базується на ієрархічній структурі. Завдяки використанню різних організаційних компонентів (або об’єктів) компанія може створити інфраструктуру мережевого управління та структуру каталогів, що віддзеркалює бізнес-організацію. Це означає, що якщо компанія має 10 основних підрозділів, кожен з яких має кілька відділів (наприклад, відділи продажів і людських ресурсів), модель служб каталогів може відображати цю структуру за допомогою використання різних об’єктів всередині каталогу. Ця структура може ефективно вмістити фізичні та логічні аспекти інформаційних ресурсів, таких як інші бази даних, користувачі та комп’ютери. На додаток до ієрархічної організації об'єктів всередині, Active Directory також інтегрується зі службою імен мережі, системою доменних імен (DNS). DNS забезпечує ієрархічне найменування та розташування ресурсів по всій компанії та в загальнодоступному Інтернеті.

1.2.3.1. Полегшений протокол доступу до директорій (LDAP)

LDAP – це спосіб спілкування з Active Directory.

LDAP — це протокол, який можуть зрозуміти багато різних служб каталогів і рішень керування доступом.

Відносини між AD і LDAP дуже схожі на відносини між Apache і HTTP:

HTTP — це веб-протокол.

Apache — це веб-сервер, який використовує протокол HTTP.

LDAP — це протокол служб каталогів.

Active Directory — це сервер каталогів, який використовує протокол LDAP.

Іноді ви почуєте, як хтось каже: «У нас немає Active Directory, але у нас є LDAP». Ймовірно, вони мають на увазі, що у них є інший продукт, наприклад OpenLDAP, який є сервером LDAP.

Це схоже на те, що хтось сказав: «У нас є HTTP», коли насправді мав на увазі «У нас є веб-сервер Apache».

Проста авторизація передбачає три можливі механізми аутентифікації:

1. Анонімна авторизація: надає LDAP анонімний статус клієнта.
2. Неаутентифіеована: лише для цілей реєстрації, не слід надавати доступ клієнту.
3. Аутентифікація імені/паролю: надає доступ до сервера на основі наданих облікових даних – проста аутентифікація користувача/паролю не є безпечною і не підходить для аутентифікації без захисту конфіденційності.

1.2.4. Keycloak

Keycloak — це рішення з відкритим вихідним кодом для управління ідентифікацією та доступом для сучасних програм і служб.

Особливості:

1. Підтримка кількох протоколів - на даний момент Keycloak підтримує три різні протоколи, а саме - OpenID Connect, OAuth 2.0 і SAML 2.0.
2. SSO - keycloak має повну підтримку єдиного входу та єдиного виходу.
3. Консоль адміністратора - keycloak пропонує графічний інтерфейс на основі вебсайту, де ви можете «натиснути» всі конфігурації, необхідні вашому екземпляру, щоб працювати так, як ви бажаєте.
4. Ідентичність користувача та доступи - keycloak можна використовувати як окремий менеджер ідентифікації користувача та доступу, дозволяючи

нам створювати базу даних користувачів із користувацькими ролями та групами. Цю інформацію можна використовувати для автентифікації користувачів у нашій програмі та захисту її частин на основі попередньо визначених ролей.

5. Синхронізація джерела зовнішньої ідентифікації - якщо ваш клієнт наразі має певний тип бази даних користувачів, Keycloak дозволяє нам синхронізуватися з такою базою даних. За замовчуванням він підтримує LDAP і Active Directory, але ви можете створювати власні розширення для будь-якої бази даних користувачів за допомогою Keycloak User Storage API. Майте на увазі, що таке рішення може містити не всі дані, необхідні для повноцінної роботи Keycloak, тому не забудьте перевірити, чи працює ваша бажана функціональність.
6. Ідентифікаційний посередник - keycloak також може працювати як проксі-сервер між вашими користувачами та деякими зовнішніми постачальниками ідентифікаційних даних. Їх список можна редагувати з панелі адміністратора Keycloak.
7. Постачальники соціальної ідентифікації – крім того, Keycloak дозволяє нам використовувати постачальників соціальної ідентифікації. Він має вбудовану підтримку Google, Twitter, Facebook, Stack Overflow, але, зрештою, вам доведеться налаштувати всі їх вручну з панелі адміністратора. Повний список підтримуваних провайдерів соціальної ідентифікації та посібник з їх налаштування можна знайти в документації Keycloak.
8. Налаштування сторінок - keycloak дозволяє налаштувати всі сторінки, які він відображає вашим користувачам. Ці сторінки у форматі .ftl, тож ви можете використовувати класичні HTML-розмітки та стилі CSS, щоб сторінка відповідала вашому стилю програми та бренду вашої компанії. Ви навіть можете розмістити власні сценарії JS як частину налаштування сторінок, щоб можливості були безмежними.

1.2.5. Інтегроване середовище розробки (IDE)

Інтегроване середовище розробки (IDE) — це набір програмного забезпечення, який об’єднує основні інструменти, необхідні для написання та тестування програмного забезпечення.

Розробники використовують різні інструменти під час створення, створення та тестування програмного коду. Інструменти розробки часто включають текстові редактори, бібліотеки коду, компілятори та тестові платформи. Без IDE розробник повинен вибирати, розгортати, інтегрувати всі ці інструменти та керувати ними окремо. IDE об’єднує багато інструментів розробки як єдину платформу, додаток або службу. Вбудований набір інструментів призначений для спрощення розробки програмного забезпечення та може виявляти та мінімізувати помилки кодування та друкарські помилки.

1.2.5.1. IDEA

IntelliJ IDEA. Кожен аспект IntelliJ IDEA був розроблений для максимальної продуктивності розробників. Разом розумна допомога в кодуванні та ергономічний дизайн роблять розробку не тільки продуктивною, але й приємною.

Після того, як IntelliJ IDEA проіндексував ваш вихідний код, він пропонує надзвичайно швидкий і розумний досвід, надаючи відповідні пропозиції в

кожному контексті: миттєве й розумне завершення коду, аналіз коду на льоту та надійні інструменти рефакторингу. Вигляд середовища (рис. 1.2).

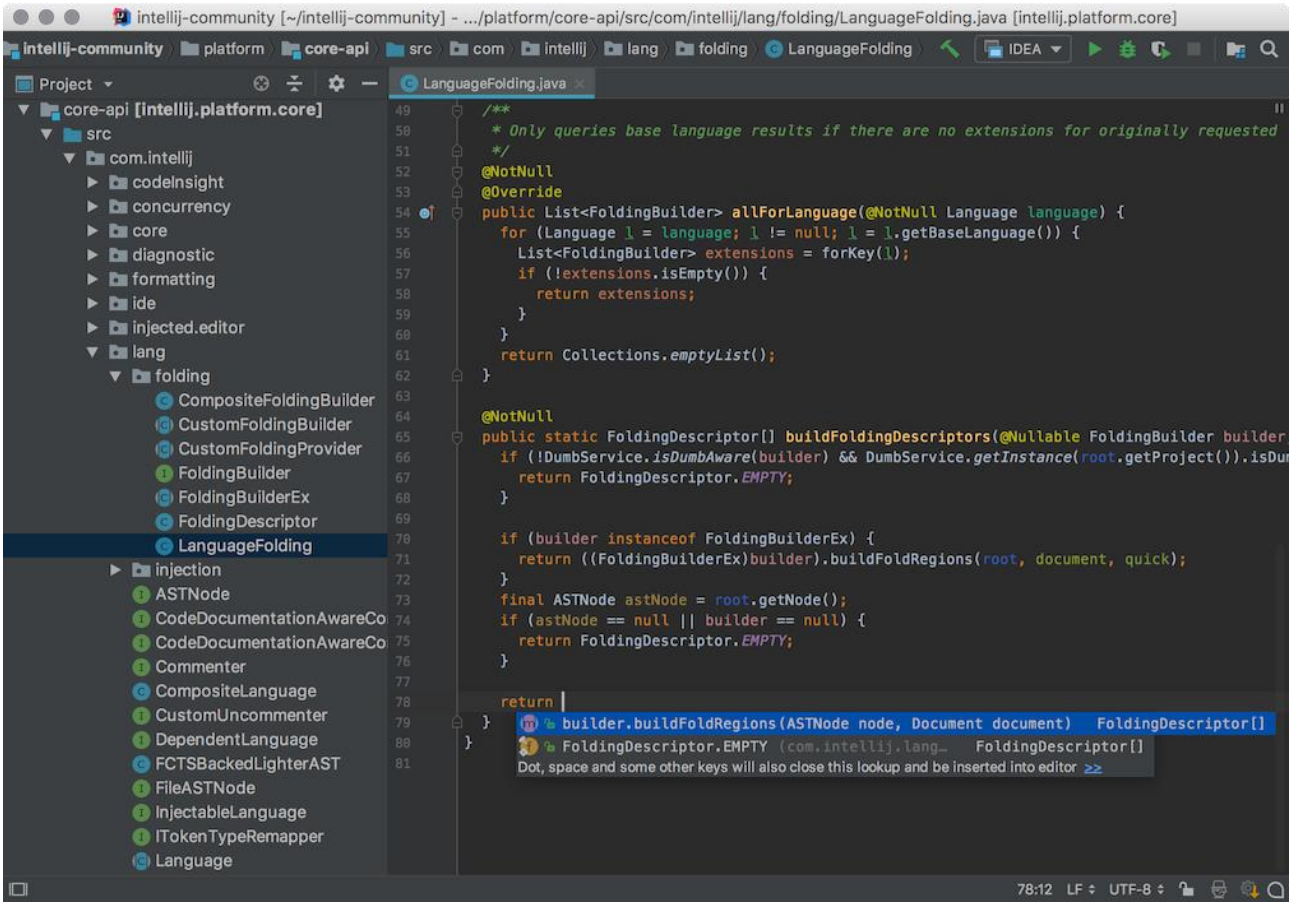

Рис. 1.2. Середовище розробки IDEA

1.2.5.2. DataGrip

IDE бази даних, яка відповідає конкретним потребам професійних розробників SQL.

Критично важливі інструменти, такі як інтегровані системи контролю версій і широкий спектр підтримуваних мов і фреймворків, є під рукою — без додаткових проблем із плагіном.

Дозволяє виконувати запити в різних режимах і надає локальну історію, яка відстежує всю вашу активність і захищає вас від втрати роботи.

Дозволяє перейти до будь-якої таблиці, представлення чи процедури за її назвою за допомогою відповідної дії або безпосередньо з її використання в коді SQL.

DataGrip забезпечує контекстно-залежне завершення коду, допомагаючи писати код SQL швидше. Завершення знає структуру таблиць, зовнішні ключі і навіть об’єкти бази даних, створені в коді, який ви редагуєте. Вигляд середовища (рис. 1.3).

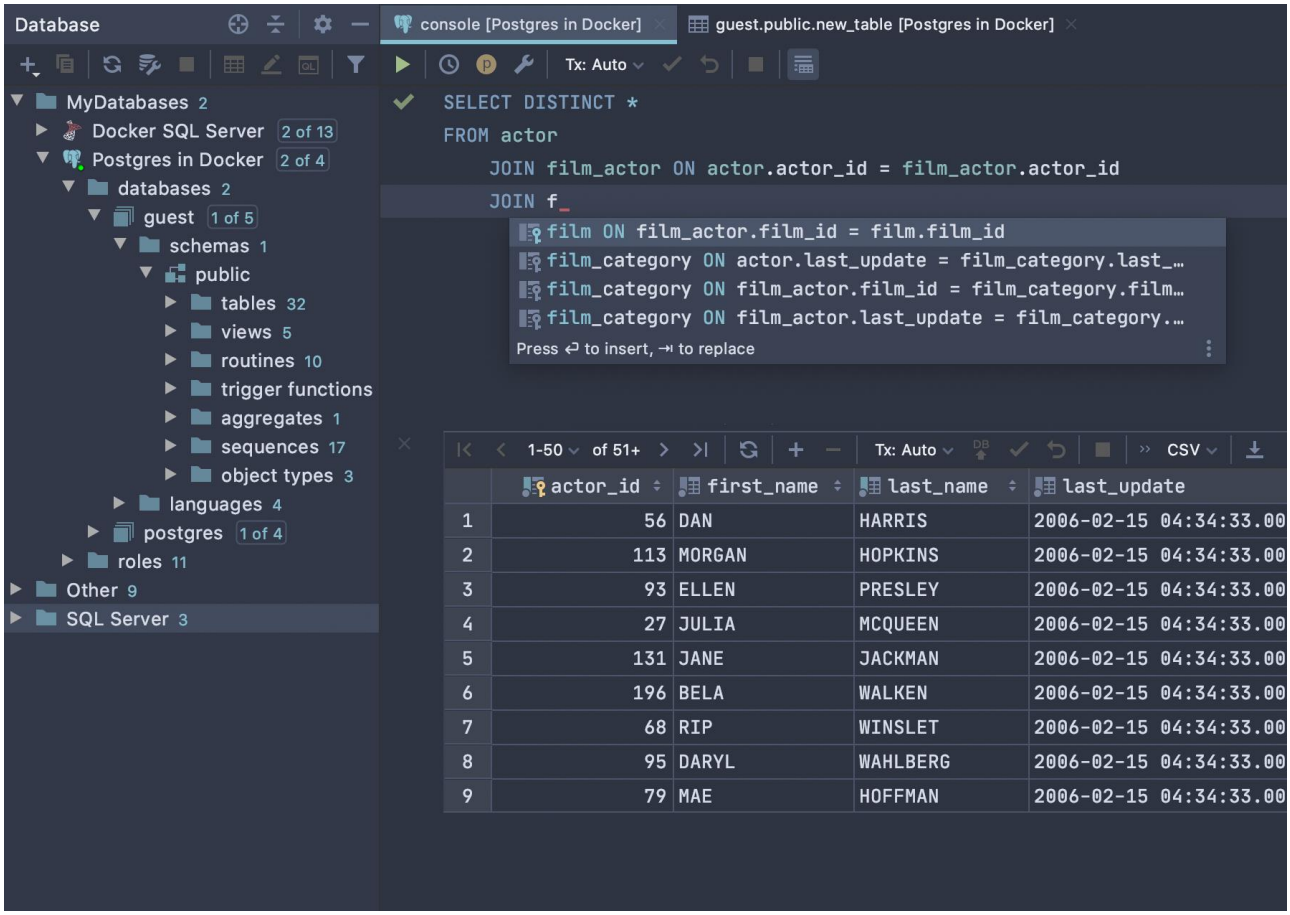

Рис. 1.3. Середовище розробки DataGrip

DataGrip виявляє ймовірні помилки у вашому коді та пропонує найкращі варіанти їх виправлення на льоту. Він миттєво повідомить вам про невирішені об’єкти, використовуючи ключові слова як ідентифікатори, і завжди пропонує спосіб вирішити проблеми.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБСАЙТУ

За основу архітектури було обрано мікросервіси. Це означає, що кожен сервіс працює окремо на своєму порті і опрацьовує запити.

Проект складається з 6 мікросервісів, 2 баз даних.

Мікросервіси:

- Frontend
- Backend
- Embedded-ldap
- Gateway
- Pictures
- Keycloak

Вигляд архітектури додатку (рис. 2.1).

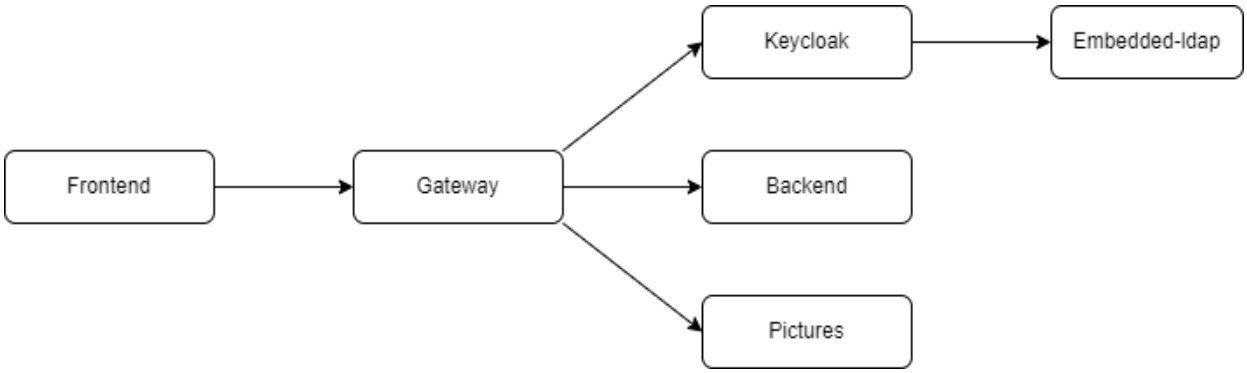

Рис. 2.1. Архітектура додатку

2.1. Frontend

Структура сайту - поняття більш вузьке, це логічна побудова сторінок, розташування розділів та їх взаємозв'язок. Робота зі структурою є одним із прийомів SEO. Це впливає на роботу користувачів із ресурсом і сприйняття його пошуковими системами.

Структура веб-сайту є однією з найважливіших складових успіху продукту. Це логічне поєднання всіх кінцевих ресурсів. На жаль, незрозуміло, як має виглядати структура існуючого сайту, оскільки багато компонентів можуть динамічно з’являтися при розробці або доповненні проекту. Від правильної конструкції конструкції залежить зручність для користувача. Якщо структура спроектована неправильно, навігація незручна для споживачів, щоб знайти потрібну інформацію, товар, категорію, то цей користувач довго не затримається.

Структура семантики ґрунтується на трьох складових:

- Наскільки часто використовується
- Потреби користувача, чи потрібна йому ця сторінка та навіщо
- Кластеризація зібраної семантики по топу

Зупинимося на потребах користувачів. Працюючи з навігацією по сайту та внутрішніми посиланнями, задайте собі три основні запитання:

- Що потрібно користувачам?
- Що їм можна запропонувати?
- Що їм ще може бути цікаво, тобто які сторінки пов'язати один з одним?

Налаштуйте сторінки відповідно до ваших потреб. Сторінка не буде ранжуватися, якщо вона не містить вмісту, який хочуть бачити користувачі.

Чи відповідає сторінка потребам користувача, визначається поведінковими факторами: ступенем клікабельності у виводі, часом переходу на сторінку, кількістю помилок тощо.

Такого ж принципу необхідно керуватися при складанні навігації, виборі та посиланні сторінок, які цікавлять користувача, і він повинен закрити всі питання і зробити замовлення.

Якщо вебсайт створений успішно, користувач зможе вирішити власну проблему і знайти те, що йому потрібно, як можна швидше.

Усі рішення щодо створення структури повинні прийматися на основі даних відвідувань веб-сайту та аналізу його функцій, щоб зробити веб-сайт

більш зручним та зрозумілим для пошукових систем. Проведіть аналітику та визначте:

- які сторінки мають високий трафік;
- які відповідають запитам користувачів;
- з якими довше взаємодіють;
- на яких виконують конверсійні дії.

Сторінки, які мають високу ефективність за цими параметрами, мають бути більш помітними на сайті. Сторінки з нижчими оцінками, ймовірно, будуть менш важливими.

Наприклад, якщо сторінка «Про нас» має низький рівень конверсій, відображається в заголовку та важлива для навігації по сайту, вона, ймовірно, не буде такою цікавою для клієнтів, і її можна перемістити, розмістивши її на своєму місці на панелі навігації. найважливіші сторінки.

Інша справа, якщо сторінка дійсно важлива для конверсії, то вона веде до CTA – кнопок перетворення з інших сторінок і кількох конверсій. Можливо, ви вибрали поганий варіант CTA. Тестуйте різні варіанти ключів з різним текстом, змінюйте макет, вигляд або формулювання.

Чим менше кліків потрібно для переходу на сторінку, тим легше її знайти, і чим ближче вона до головної сторінки, тим вона важливіша.

Багато оптимізаторів застосовують «правило трьох кліків»: жодна важлива сторінка на вебсайті не повинна знаходитися більше ніж на три кліки від головної сторінки чи будь-якої іншої сторінки високого рівня.

Пошуковий бот перетинає сторінки, ніби багатошарові: спочатку головну сторінку, потім сторінки другого рівня вкладення, потім третій і так далі. Чим ближче сторінка до головної, тим більша ймовірність, що він її відвідає. Посилання на новий пост з важливої сторінки сигналізує боту, що його також потрібно перевірити.

Тут допомагає правило трьох клацань. Але це не правило, а рекомендація. Можуть бути ситуації, коли це безглуздо, але краще намагатися не розміщувати важливі сторінки далеко від головної сторінки.

У вас може виникнути питання: чому б не пов'язувати всі сторінки з головною, щоб вони виявилися важливими? Є щонайменше дві причини так не робити:

- Багато посилань мають свої недоліки, наприклад, розмивання довіри, яку можна передати на різні важливі сторінки.
- Однорівнева структура виключає можливість організації контекстної ієрархії в контенті, що важливо для пошукових систем.

Під час організації вмісту часто краще використовувати URL-адреси, які відображають структуру, тобто сторінку центру та підкатегорію, де розташована вихідна сторінка.

Навіщо потрібна ієрархічна структура URL:

- Користувачі можуть по URL зрозуміти, де знаходяться.
- Ключові слова в URL-адресі можуть допомогти в ранжируванні та CTR.
- Плюс до ранжування: Google використовує показники рівня вкладеності сторінки, щоб якийсь час визначати важливість і релевантність нових URL-адрес.

Деякі веб-майстри «підробляють» плоску структуру каталогів, роблячи всі URL-адреси постійними або обмежуючи папки. Хоча цей метод може мати свої переваги, Google більше піклується про те, скільки кліків потрібно, щоб перейти на сторінку вмісту, ніж про те, скільки інформації міститься між косими рисками в його URL-адресі. Розміщення сторінок у різних категоріях дає Google додатковий контекст про кожну сторінку цієї категорії.

Для сторінок категорій зі списками зі 100 або 1000 позицій є три методи, які допоможуть згладити великі масиви пунктів:

- пагінація, тобто розподіл однорідного контенту на сторінках;
- перегляд усіх позицій;
- нескінченне прокручування.

Найпростішим і популярним рішенням є розділення списків сторінок. Правильне впровадження допоможе повідомити Google, що всі записи на сторінках є частиною однієї спільноти. Більшість оптимізаторів вибирають саме цей метод.

З іншого боку, перегляд усіх сторінок також може спростити архітектуру, пов’язуючи всі записи сторінки. Деякі SEO-спеціалісти використовують цей метод, оскільки вважають, що Google краще перевіряє записи на одній сторінці. Це добре працює, якщо у вас є лише кілька позицій. Якщо категорій більше 100, сторінка може завантажуватися повільно або бути незручною для перегляду через інтерфейс користувача.

Гібридний підхід — «Нескінченна прокрутка». Результати постійно завантажуються в браузер користувача, але позначаються на сторінках, які легко сприймаються.

Розумні фільтри пов’язані з навігацією, вони дозволяють сортувати та звужувати результати за багатьма критеріями.

Фільтрація корисна для відвідувачів вебсайту, але вона створює мільйони комбінацій URL-адрес для пошукових систем. Багато з них дублюють вміст, що заважає сканувати та індексувати сайт.

Ось як вирішити проблему: вам потрібно спрямовувати пошукові системи на унікальні сторінки з високою відвідуваністю та забороняти сканування менш важливих сторінок.

Наприклад, Google може проіндексувати сторінку "Кросівки для чоловіків", але має сенс виключити розмір сторінки "Кросівки для чоловіків..." з індексації.

Після того, як ви зрозумієте, які URL-адреси ви хочете індексувати, а які ні, у вашому розпорядженні багато інструментів для управління розумною фільтрацією, у тому числі:

- мета-роботи;
- robots.txt;
- rel = canonical;
- параметри Search Console;

- атрибути nofollow;
- схеми JavaScript

2.2. Gateway

Шлюз API забезпечує єдину точку входу API для одного або кількох внутрішніх API. Вони також схильні обмежувати швидкість і безпеку. Рівень керування API, як-от Tyk.io, додає додаткові функції, такі як аналітика, монетизація та керування життєвим циклом.

Архітектура мікросервісів може мати багато різних служб. Шлюз API може допомогти забезпечити єдину точку входу для зовнішніх споживачів, незалежно від кількості та складу внутрішніх мікросервісів.'

Запобігає показ внутрішніх проблем зовнішнім клієнтам. Шлюз API відокремлює зовнішні загальнодоступні API від внутрішніх API мікросервісів, дозволяючи додавати мікросервіси та змінювати пороги. Результатом є можливість належної реорганізації та масштабування мікросервісів з часом без впливу на зовнішніх клієнтів. Він також приховує інформацію про службу та версію клієнта, забезпечуючи єдину точку входу для всіх ваших мікросервісів.

Впроваджу додатковий рівень безпеки вашим мікросервісам. Шлюзи API допомагають запобігти зловмисним атакам, забезпечуючи додатковий захист від векторів атак, таких як ін’єкція SQL, експлойти аналізатора XML та атаки відмови в обслуговуванні (DoS).

Включає підтримку змішування протоколів зв'язку. Хоча зовнішні API часто пропонують API на основі HTTP або REST, внутрішні мікросервіси можуть отримати вигоду від використання різних протоколів зв’язку. Протоколи можуть включати ProtoBuf, AMQP або, можливо, системну інтеграцію з SOAP, JSON-RPC або XML-RPC. API Gateway може надати уніфікований зовнішній API на основі REST для цих різних протоколів, дозволяючи командам вибирати те, що найкраще відповідає внутрішній архітектурі.

Зменшена складність мікросервісу. Мікросервіси мають загальні проблеми, такі як автентифікація маркерів API, контроль доступу та обмеження швидкості. Кожна з цих проблем може додати більше часу для розробки мікросервісів, вимагаючи від кожної служби їх впровадження. Шлюз API усуне ці проблеми з вашого коду, дозволяючи вашим мікросервісам зосередитися на поставленому завданні.

Мокінг і віртуалізація мікросервісів. Відокремлюючи API мікросервісів від зовнішнього API, ви можете імітувати або віртуалізувати свої послуги, щоб перевірити вимоги до дизайну або допомогти в тестуванні інтеграції.

Spring Cloud Gateway має на меті надати простий, але ефективний спосіб маршрутизації до API та забезпечувати їх наскрізні проблеми, такі як безпека, моніторинг/метрики та стійкість.

Як працює Spring Cloud Gateway: клієнти роблять запити до Spring Cloud Gateway. Якщо зіставлення обробника шлюзу визначає, що запит відповідає маршруту, він надсилається до веб-обробника шлюзу. Цей обробник запускає запит через ланцюжок фільтрів, який є специфічним для запиту. Причина, по якій фільтри розділені пунктирною лінією, полягає в тому, що фільтри можуть виконувати логіку як до, так і після надсилання запиту проксі. Вся логіка попереднього фільтра виконується. Потім робиться запит на проксі. Після виконання запиту проксі запускається логіка фільтра “post”. (рис. 2.2).

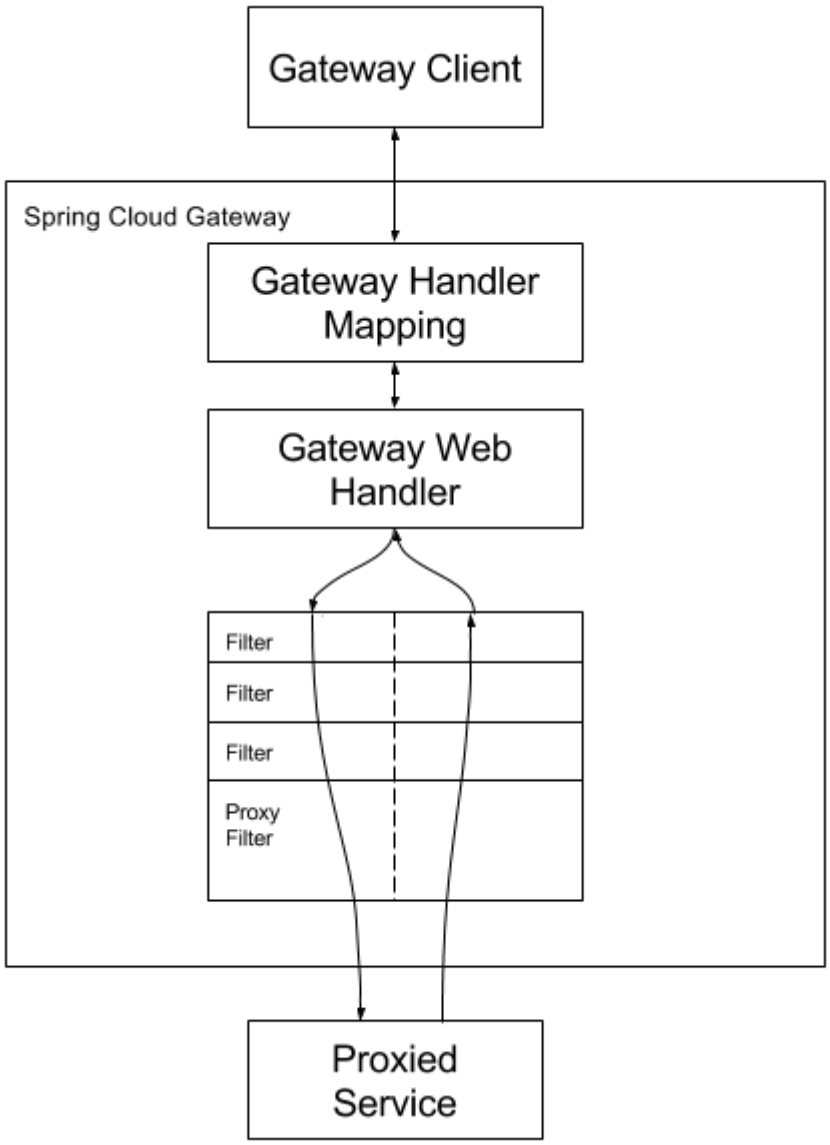

Рис. 2.2. Огляд високого рівня роботи Spring Cloud Gateway.

Spring Cloud Gateway відповідає маршрутам як частина інфраструктури Spring WebFlux HandlerMapping. Spring Cloud Gateway містить багато вбудованих фабрик предикатів маршрутів. Усі ці предикати збігаються з різними атрибутами запиту HTTP. Ви можете комбінувати декілька фабрик предикатів маршруту з логічними операторами та операторами.

2.3. Backend

Веб-архітектура серверної частини дозволяє дизайнерам вебсайтів мати більше контролю над своїм середовищем, дозволяючи логіці програми працювати з користувачем, який повинен бачити кожну деталь коду. Це

створює більш зручне середовище, гарантуючи, що користувач може зосередитися на насолоді функціями готової веб-сторінки.

Він також забезпечує додаткову безпеку, обмежуючи зовнішній доступ до важливих даних. Наприклад, вебсайт електронної комерції може приховувати логіку, яка визначає поточну вартість продуктів, які з'являються на вебсайті, і систему обробки платежів, яка захищає банківські реквізити компанії.

Веб-архітектура сервера дозволяє серверам виконувати обчислення замість того, щоб виконувати їх на стороні клієнта або на комп'ютері користувача. Це може дозволити вебсайту працювати ефективніше та дозволити більшій кількості користувачів отримати доступ до вебсайта незалежно від типів обладнання, операційних систем та веб-браузерів, які вони використовують.

Наприклад, вебсайт, який запускає гру, може розміщувати процеси, необхідні для роботи гри на власному сервері, щоб зменшити вимоги до користувачів під час запуску гри на власних комп'ютерах.

Сервер запускає програму, яка містить логіку про те, як відповідати на різні запити на основі дієслова HTTP та Uniform Resource Identifier (URI). Пара дієслова HTTP і URI називається маршрутом, а їх відповідність на основі запиту називається маршрутизацією.

Деякі з цих функцій обробника будуть проміжним програмним забезпеченням. У цьому контексті проміжне програмне забезпечення — це будь-який код, який виконується між отриманням сервером запиту та відправкою відповіді. Ці функції проміжного програмного забезпечення можуть змінювати об'єкт запиту, запитувати базу даних або іншим чином обробляти вхідний запит. Функції проміжного програмного забезпечення зазвичай закінчуються передачею керування наступній функції проміжного програмного забезпечення, а не надсиланням відповіді.

Часто програмісти використовують такі фреймворки, як Express або Ruby on Rails, щоб спростити логіку маршрутизації. Наразі просто подумайте, що кожен маршрут може мати одну або декілька функцій обробника, які

виконуються щоразу, коли збігається запит до цього маршруту (дієслово HTTP та URI).

Перш ніж зрозуміти архітектуру Spring Boot, ми повинні знати різні шари та класи, присутні в ній. У Spring Boot є чотири шари:

- Шар презентації
- Бізнес-рівень
- Шар стійкості
- Рівень бази даних

Рівень презентації: рівень презентації обробляє HTTP-запити, перетворює параметр JSON в об'єкт, аутентифікує запит і передає його на бізнес-рівень. Коротше кажучи, він складається з представлень, тобто інтерфейсної частини.

Бізнес-рівень: бізнес-рівень обробляє всю бізнес-логіку. Він складається з класів послуг і використовує послуги, що надаються рівнями доступу до даних. Він також виконує авторизацію та перевірку.

Рівень збереження: рівень збереження містить усю логіку зберігання та перекладає бізнес-об'єкти з і в рядки бази даних.

Рівень бази даних: на рівні бази даних виконуються операції CRUD (створення, отримання, оновлення, видалення).

Архітектура потоку spring завантаження. (рис. 2.3):

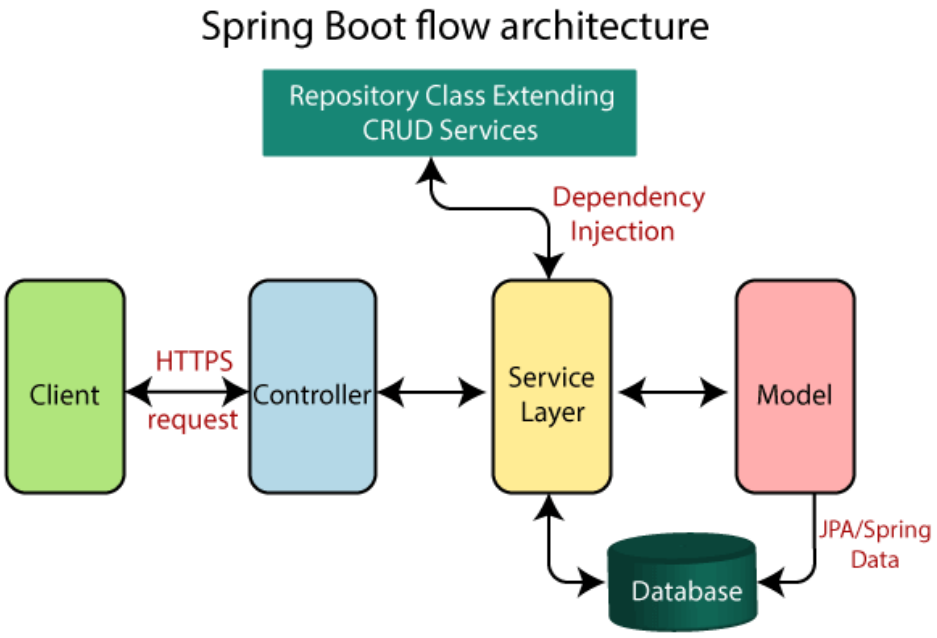

Рис. 2.3. Архітектура потоку spring завантаження.

- Тепер у нас є класи-валідатори, класи представлення даних і класи-служби.
- Spring Boot використовує всі модулі Spring-подібних Spring MVC, Spring Data тощо. Архітектура Spring Boot така ж, як і архітектура Spring MVC, за винятком одного: у завантаженні Spring немає потреби в класах DAO і DAOImpl.
- Створює рівень доступу до даних і виконує операцію CRUD.
- Клієнт робить HTTP-запити (PUT або GET).
- Запит надходить до контролера, а контролер відображає цей запит і обробляє його. Після цього він викликає логіку служби, якщо потрібно.
- На рівні сервісу виконується вся бізнес-логіка. Він виконує логіку над даними, які відображаються в JPA за допомогою класів моделі.
- Якщо помилка не сталася, користувачеві повертається сторінка JSP.

РОЗДІЛ 3. РОЗРОБКА ВЕБСАЙТУ.

3.1. Бази даних

Почнемо зі створення баз даних та їх сутностей. Перше, що нам знадобиться це завантажити сервери PostgreSQL та MongoDB. Так як ми завантажуємо бінарний файл Postgres'у то нам знадобиться додаткова конфігурація нашої бази. Зазвичай після розгортання бази створюють додаткового користувача і всю наступну діяльність виконують від його обличчя, але ми використаємо користувача за замовчуванням.

Перш ніж ви зможете щось зробити, ви повинні ініціалізувати область зберігання бази даних на диску. Ми називаємо це кластером бази даних. (У стандарті SQL використовується термін кластер каталогу.) Кластер баз даних — це набір баз даних, якими керує один екземпляр запущеного сервера баз даних. Після ініціалізації кластер бази даних міститиме базу даних з іменем postgres, яка призначена як база даних за замовчуванням для використання утилітами, користувачами та сторонніми програмами. Сам сервер бази даних не вимагає існування бази даних postgres, але багато зовнішніх допоміжних програм припускають, що вона існує. Інша база даних, створена в кожному кластері під час ініціалізації, називається template1. Як випливає з назви, це буде використовуватися як шаблон для згодом створених баз даних; його не слід використовувати для реальної роботи.

Далі створюємо свою базу даних для роботи з нашим проектом.

Для взаємодії між базами даних та backend частиною потрібно використовувати сутності та операції, які можна виконувати над ними.

Сутності PostgreSQL (рис. 3.1):

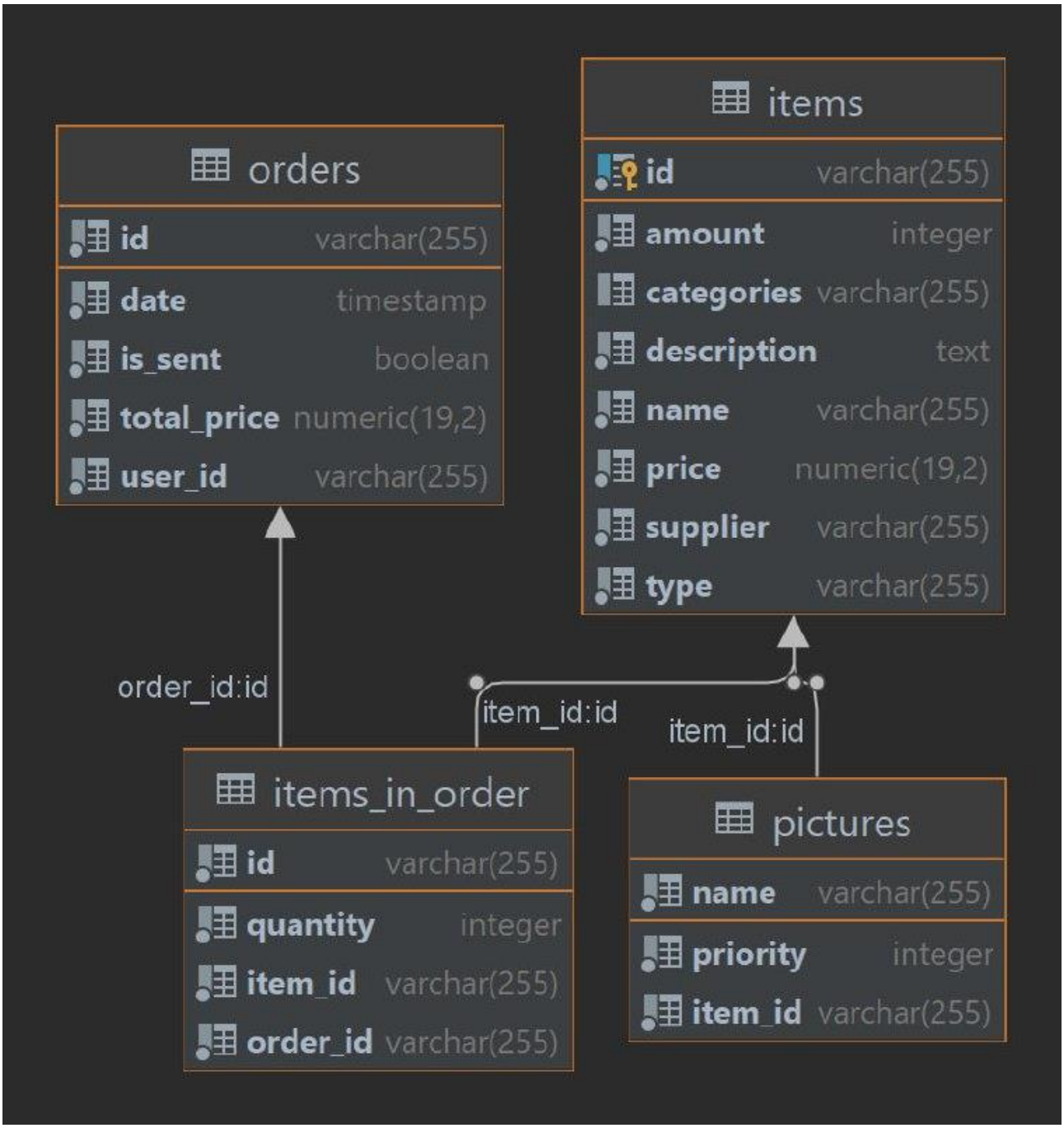

Рис. 3.1. Сутності PostgreSQL

Таблиця **items** зберігає інформацію про усі товари, які існують на сайті:

- **Id** – ідентифікатор
- **Amount** - к-сть товару
- **Categories** - категорії до яких належить товар
- **Description –** опис товару
- **Name** – назва товару
- **Price** – ціна

- **Supplier** – якому продавцю належить товар
- **Type** – тип товару

Таблиця **pictures** містить інформацію про картинки, які відображаються на сайті, хоча сама картинка перебуваю у іншій таблиці, адже MongoDB більше підходить для зберігання бінарних даних.

- **Name** – ідентифікатор
- **Priority –** пріорітетність, яка сама картинка буде відображатися для товару, адже одному товару відповідають декілька картинок
- **Item_id** – якому товару належить ця картинка

Таблиця **items_in_order** показує, які товари знаходяться у корзині для даного користувача

- **Id** – ілентифікатор
- **Quantity –** к-сть товару на замовлення
- **Item_id** – ідентифікатор товару
- **Order_id** – ідентифікатор замовлення

Таблиця **orders** вказує на замовлення користувачів

- **Id –** ідентифікатор замовлення
- **Date** – дата замовлення
- **Is_sent** – чи відправлено замовлення
- **Total_price –** ціна за всі товари
- **User_id** – ідентифікатор замовника

Для **MongoDB** нам знадобиться створення користувача, бази та сутність. Створення користувача відбувається за допомгою команди **db.createUser();** далі переключаємося на користувача якого створили та створюємо нову базу та колекцію.

Сутності MongoDB (рис. 3.2):

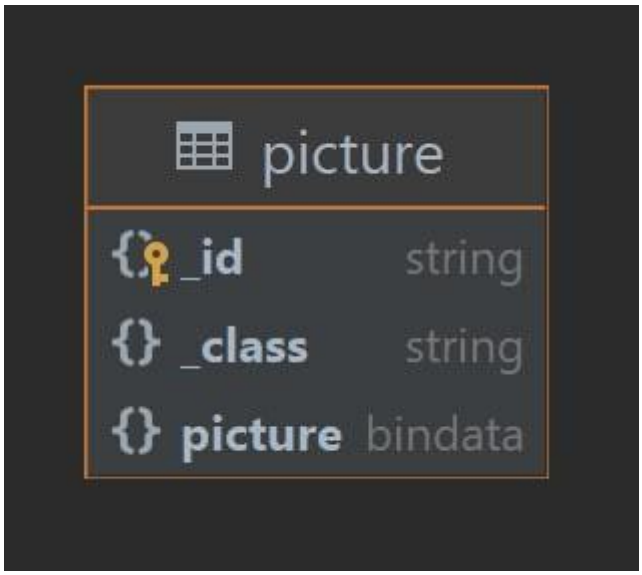

Рис. 3.2. Сутності в MongoDB

Таблиця **picture** зберігає в собі двійкове представлення картинки

- **Id** - ідентифікатор
- **Class -** представляє колекцію в базі даних і операції, які можна безпосередньо застосувати до них.
- **Picture –** картинка у двійковому представленні

3.2. Embedded-ldap

Перед тим як перейти до налаштування **Keycloak** потрібно розробити вигляд структури збереження користувачів та їх прав.

Для початка розробимо конфігурацію за допомогою якої наш сервіс може обробляти запити та по яким шляхам шукати певного користувача. Для цього ми повинні вказати base-dn – базовий DN є відправною точкою, яку використовує сервер LDAP під час пошуку автентифікації користувачів у вашому каталозі, credential – показує, хто може редагувати файли, ldif – початковий файл в якому знаходиться деяка конфігурація, що відноситься до **LDAP**, port – порт, на якому обслуговуэться служба.

В config.ldif записана будова активної директорії. Деяка частина функціональності (рис. 3.3):

```
## ____________Domain____________

dn: dc=embedded,dc=ldap
objectClass: top
objectClass: domain
objectClass: extensibleObject

## ____________First level of hierarchy: users and groups____________

## Groups
dn: ou=groups,dc=embedded,dc=ldap
objectClass: top
objectClass: organizationalUnit
ou: groups

## Users
dn: ou=users,dc=embedded,dc=ldap
objectClass: top
objectClass: organizationalUnit
ou: users

## ____________Records____________
```

Рис. 3.3. Частина конфігурацції ldif

На сайті усніє лише 5 привілегій корисутвачів:

- Анонімний - користувач, який уперше на сайті і йому потрібно зареєструватися.
- Звичайний користувач – користувач, який зараєструвався на сайті.
- Адміністратор – може виконувати різні зміни стосовно інших користувачів та сайту.
- Постачальник – додає, змінює, видаляє свою продукції на сайті.
- Комірник - – перевіряє надсилання товарів до замовлеників.

Конфігурація користувачів вигляда натсупним чином (рис. 3.4):

```
## 1
dn: login=robert,ou=users,dc=embedded,dc=ldap
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Robert
sn: Smith
uid: 9a5310f1-ecbd-4bc7-ba8a-37039a1b0063
login: robert
userPassword: WP1s>P%
phone: 323-543-6235
mail: r.smith@example.com
ou: Human Resources
```

Рис. 3.4. Вигляд конфігурації користувача в активній директорії

3.3. Keycloak

Keycloak виконує роль авторизаційного серверу. Початкову конфігурацію він бере з config.ldif. За допомогою цього сервісу можна легко керувати доступом до користувачів та груп до яких вони належать.

Як відбувається процес авторизації користувача (рис. 3.5)?

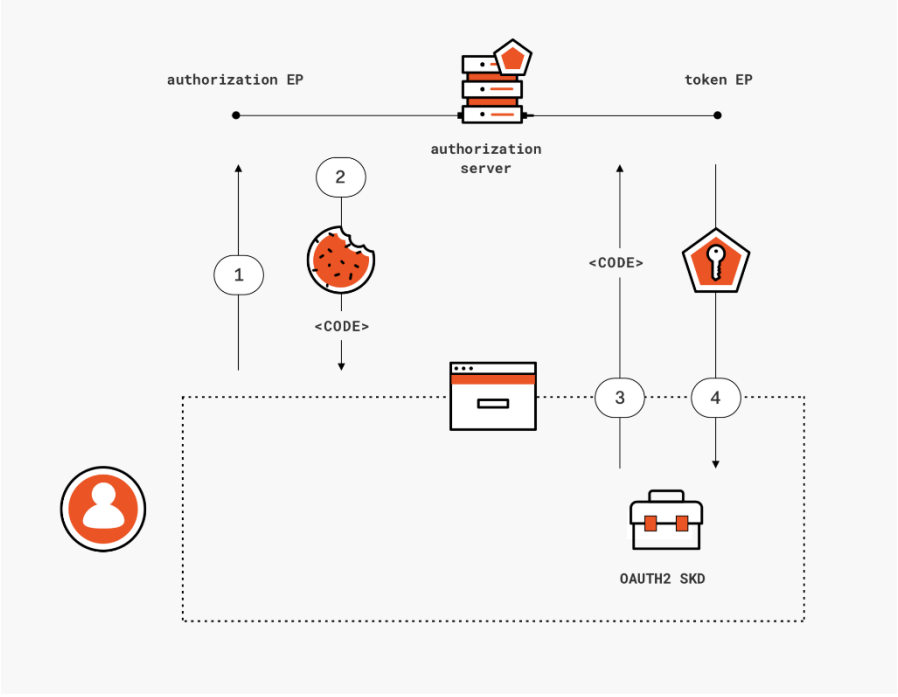

Рис. 3.5. Схема авторизації

Код авторизації — це тимчасовий код, який клієнт обмінює на маркер доступу. Сам код отримується від сервера авторизації, де користувач отримує можливість побачити, яку інформацію запитує клієнт, і схвалити або відхилити запит.

Потік коду авторизації пропонує кілька переваг перед іншими типами грантів. Коли користувач авторизує програму, він перенаправляється назад до програми з тимчасовим кодом в URL-адресі. Програма обмінює цей код на маркер доступу. Коли програма робить запит на маркер доступу, цей запит може бути аутентифікований за допомогою секрету клієнта, що зменшує ризик перехоплення зловмисником коду авторизації та його використання самостійно. Це також означає, що маркер доступу ніколи не бачить користувач або його браузер, тому це найбезпечніший спосіб передати маркер назад до програми, зменшуючи ризик витоку маркера комусь іншому.

Першим кроком веб-потоку є запит авторизації від користувача. Це досягається шляхом створення посилання на запит авторизації, на яке користувач може натиснути.

Після того, як користувач відвідує сторінку авторизації, служба показує користувачеві пояснення запиту, включаючи назву програми, область дії тощо Якщо користувач натискає кнопку «Схвалити», сервер переспрямує назад до програми з «кодом» і тим самим параметром «стан», які ви вказали в параметрі рядка запиту. Важливо зазначити, що це не маркер доступу. Єдине, що ви можете зробити з кодом авторизації, це зробити запит на отримання токена доступу.

3.4. Gateway

Будучи зосередженим на запитах маршрутизації, Spring Cloud Gateway пересилає запити до зіставлення обробника шлюзу, яке визначає, що слід робити із запитами, що відповідають певному маршруту.

Конфігурація шлюзу (рис. 3.6)

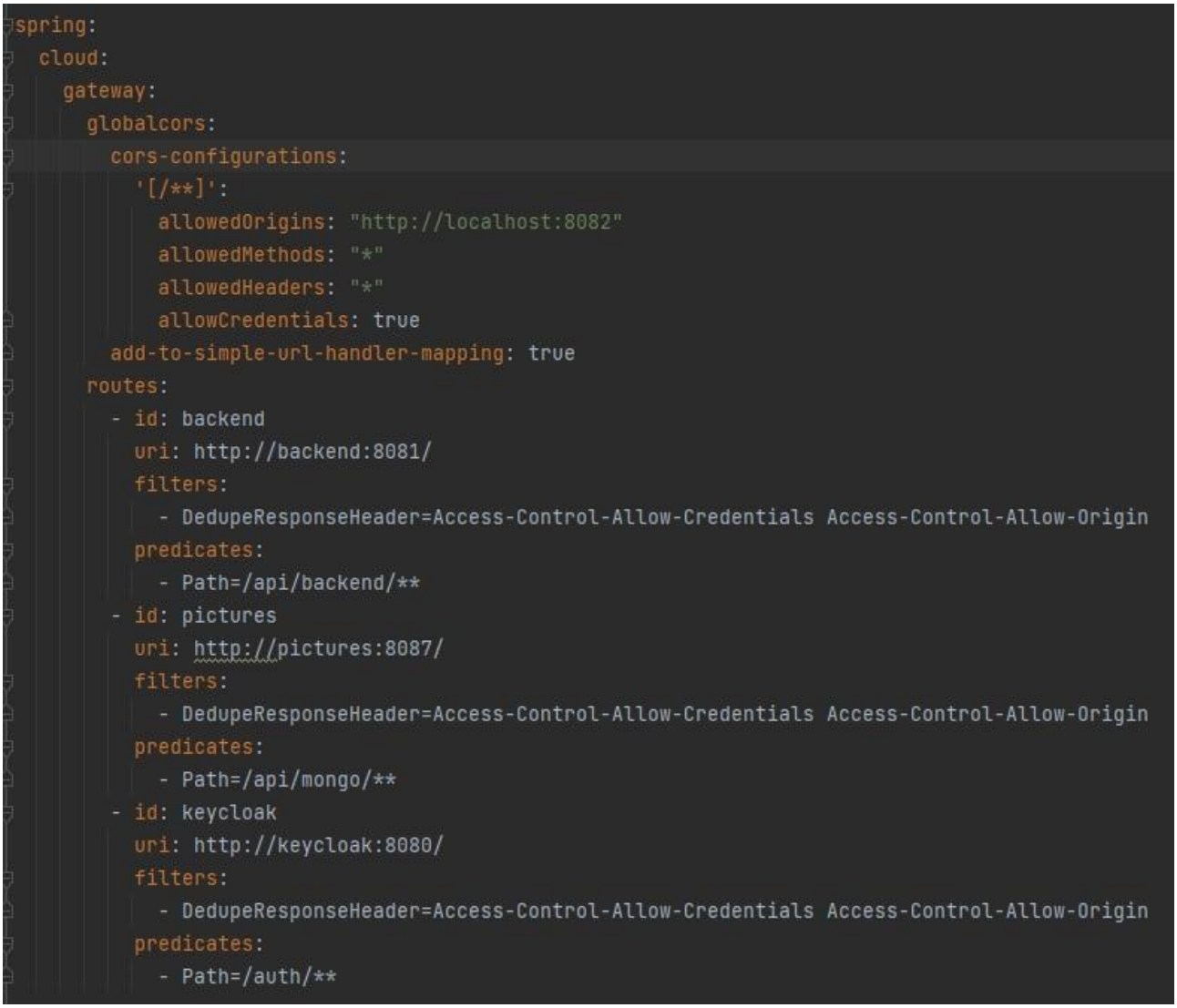

Рис. 3.6. Конфігурація шлюзу

3.5. **Pictures та Backend** (сервіси)

Обидва мікросервіси працюють параллельно, адже для товару потрібна повна інформація, як просто опис і дані, так і зображення товару. Тому на **frontend**'і робиться запит параллельно, як до першого так і для другого сервісу.

Кожен з мікросервісів має свої сутності, контроллери, репозиторії для взаємодії з базами даних, сервіси, винятки, конфігурації. В конфігурації задаються правила авторизації користувачів та які кінцеві ресурси їм доступні.

Простіше кажучи, сервер авторизації — це програма, яка видає токени для авторизації.

Раніше стек Spring Security OAuth пропонував можливість налаштування сервера авторизації як програми Spring. Але проект застарів, головним чином

тому, що OAuth є відкритим стандартом з багатьма добре зарекомендувалими постачальниками, такими як Okta, Keycloak і ForgeRock, і це лише деякі з них.

З них ми будемо використовувати Keycloak. Це сервер управління ідентифікацією та доступом з відкритим вихідним кодом, який керується Red Hat, розроблений на Java компанією JBoss. Він підтримує не тільки OAuth2, але й інші стандартні протоколи, такі як OpenID Connect і SAML.

3.6. Frontend

Вигляд навігації залежить від групи користувача. Так, для анонімного користувача з'являються 4 кнопки: home, shop, sign in, cart. (рис. 3.7).

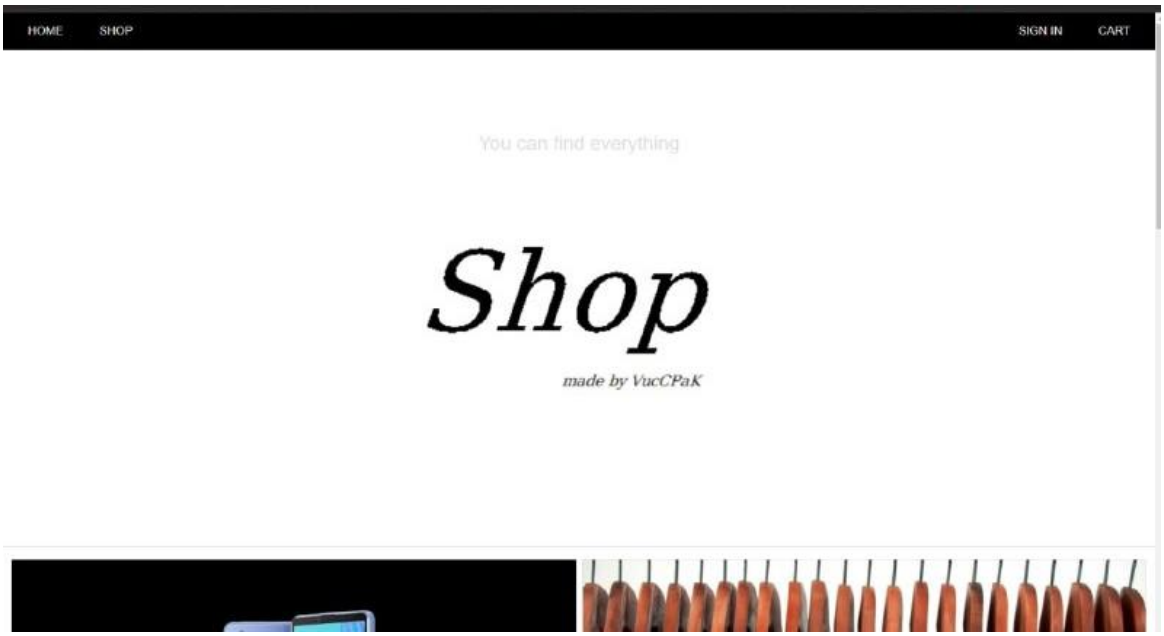

Рис. 3.7. Вигляд головної сторінки з навігацією для анонімного користувача

Щоб переключитися на іншого неанонімного користувача потріюно пройти авторизацію на сайті, так буде видан токен та права за допомогою якого сервер зможе розпізнавати від кого надходять запити. Так як ми підключили сервіс Keycloak, то при натиснені на кнопку sign in нас перекине на інший кінцевий ресурс для авторизації (рис. 3.8).

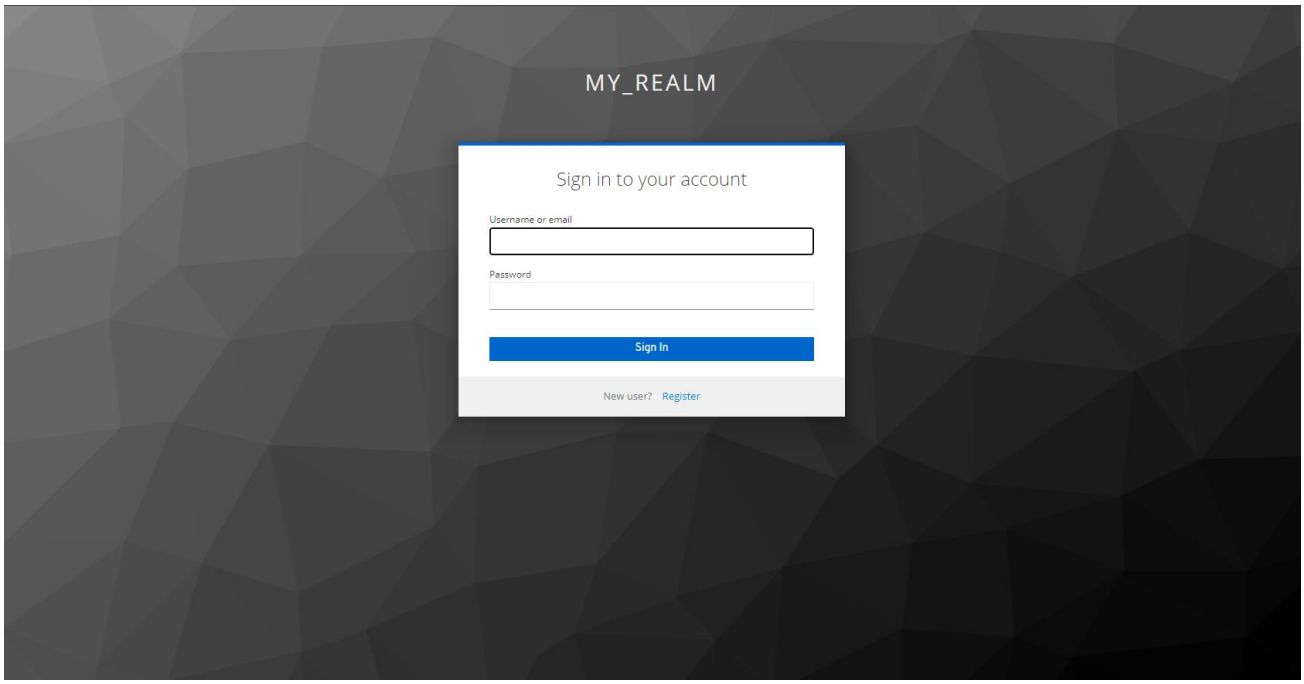

Рис. 3.8. Вигляд сторінки авторизації

Наприклад, зайдемо під правами адміністратора і отримаємо токен для підтримки наших запитів та розширений фунціонал навігації для адімінстарції. Це може бути як редагування товарів, підтвердження замовлення та інше (рис 3.9).

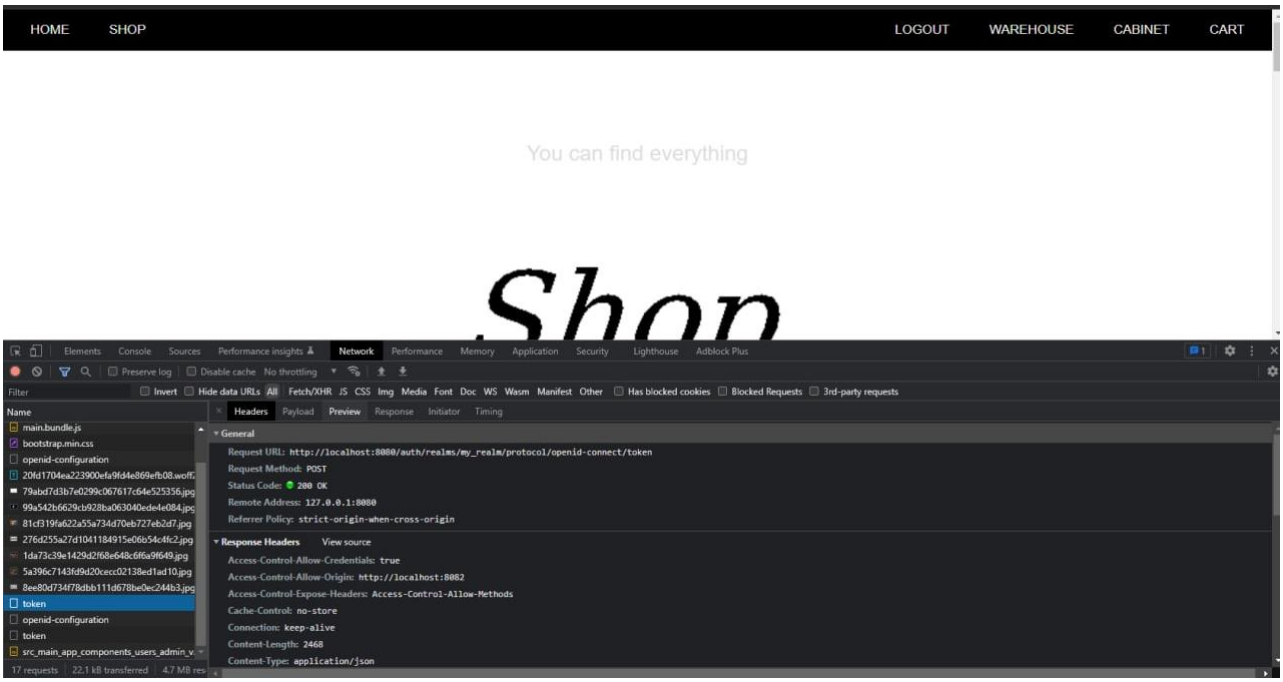

Рис. 3.9. Отримано значення токену та змінено панель навігації

Як бачимо в навігації з'явилися додаткові кнопки, такі як: cabinet, logout, warehouse. Де в cabinet ми можемо переглянути замовлені нами товари, warehouse використовується для комірників, щоб надсилати товару до пункту видачі, і logout для виходу з прав.

Також, на головній сторінці знаходиться показ товарів, які найпопулярніші та можна натиснути на них для швидкого переходу до магазина з цими товарами (рис. 3.10).

Рис. 3.10. Вигляд найпопулярніших товарів

Кожен сайт має нижній колонтитул, який має певну інформацію стосовно сайту. На ній можна розмістити популярні соціальні мережі, які є посиланнями на сайт. А також багато інших посилань на вимогу клієнта (рис. 3.11).

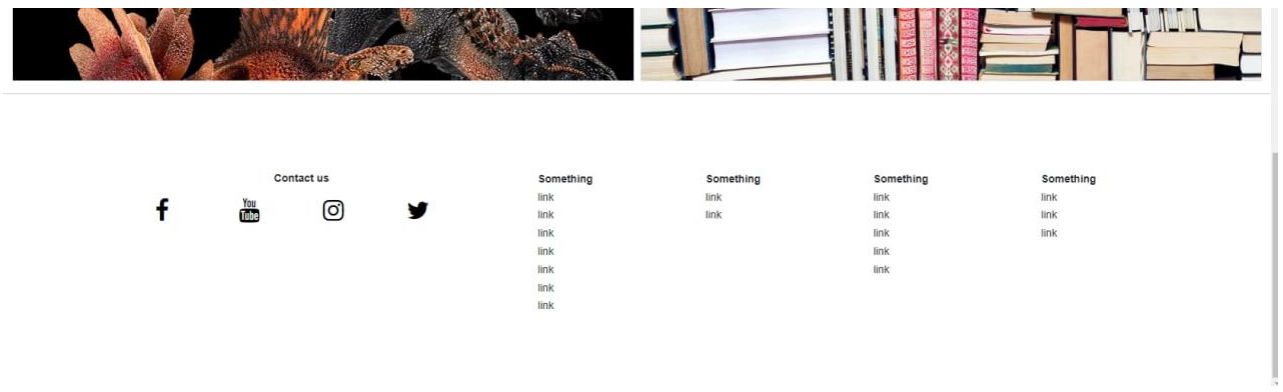

Рис. 3.11. Нижній колонтитул сайту

На вкладці shop знаходяться товари, які може придбати користувач або просто подивитися інформацію, або порівнювати їх з іншими товарами задля вибору найкращого. Так як наш вхід був під правами адміністратора, то в нас є додатковий функціонал, такий як, додавання нового товару, редагування вже існуючого товару. Також ми бачимо, що був виконаний запит на надання усіх товарі сайту, як і звичайний користувач, також може купувати. (рис. 3.12).

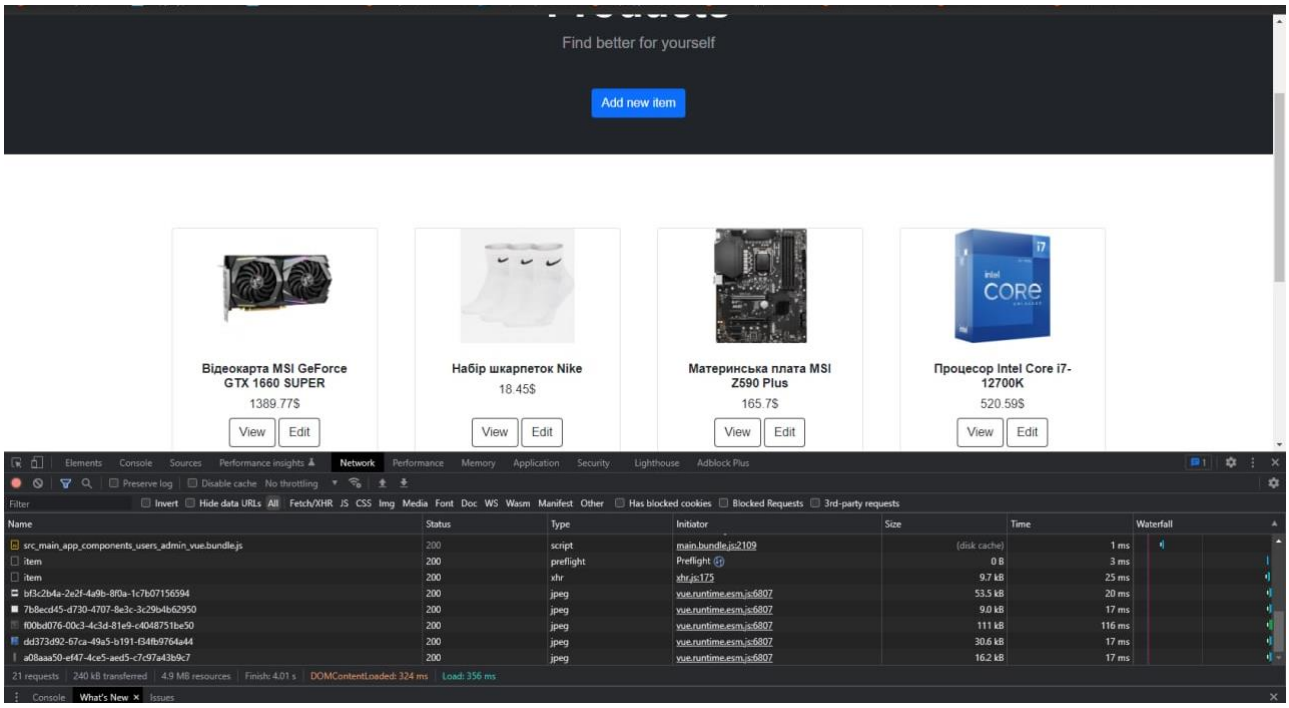

Рис. 3.12. Запит на всі товари та додатковий фунціонал адміністрації

Для редагування товару (це функція також доступна і для постачальника) потрібно натиснути кнопку edit на певному товарі, а для додавання нового кнопку add new item. Форма для редагування та створення нового – однакова. При додавані зображень потрібно враховувати пріорітетність зображень, яка саме буде відображатися у користувача на головному екрані магазину (рис. 3.13).

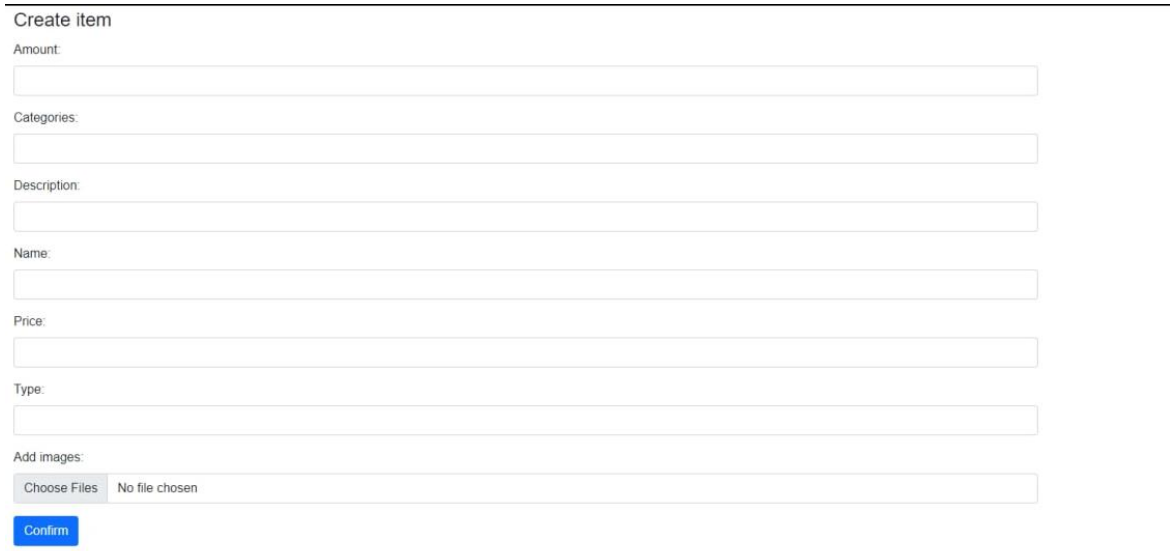

Рис. 3.13. Вигляд форми редагування та створення

Перегляному будь-який товар задля достовірності відображення апрвильності інформації (рис. 3.14).

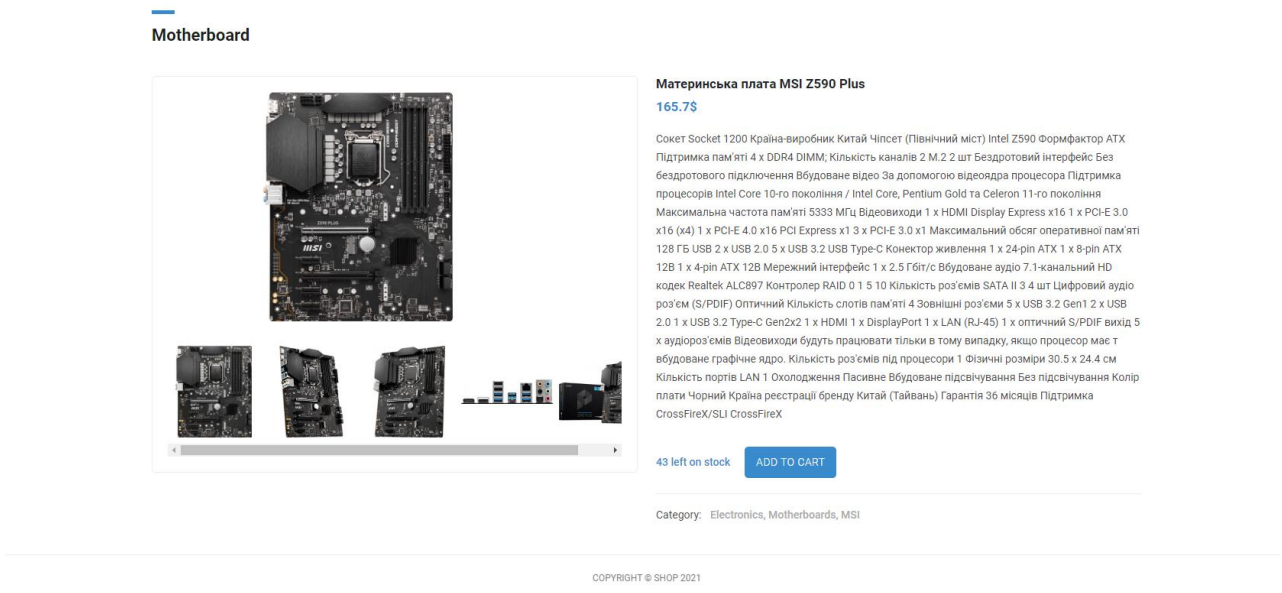

Рис. 3.14. Вигляд товару

Можна побачити, яка к-сть товару залишилась, додати до списку покупок, ціну, роздивитися різні ракурси даного товару та побачити до яких категорій належить товар.

Перейдемо до секції корзина, щоб побачити, що туди було додано новий товар та розрахована його ціна на к-сть товару (рис. 3.15).

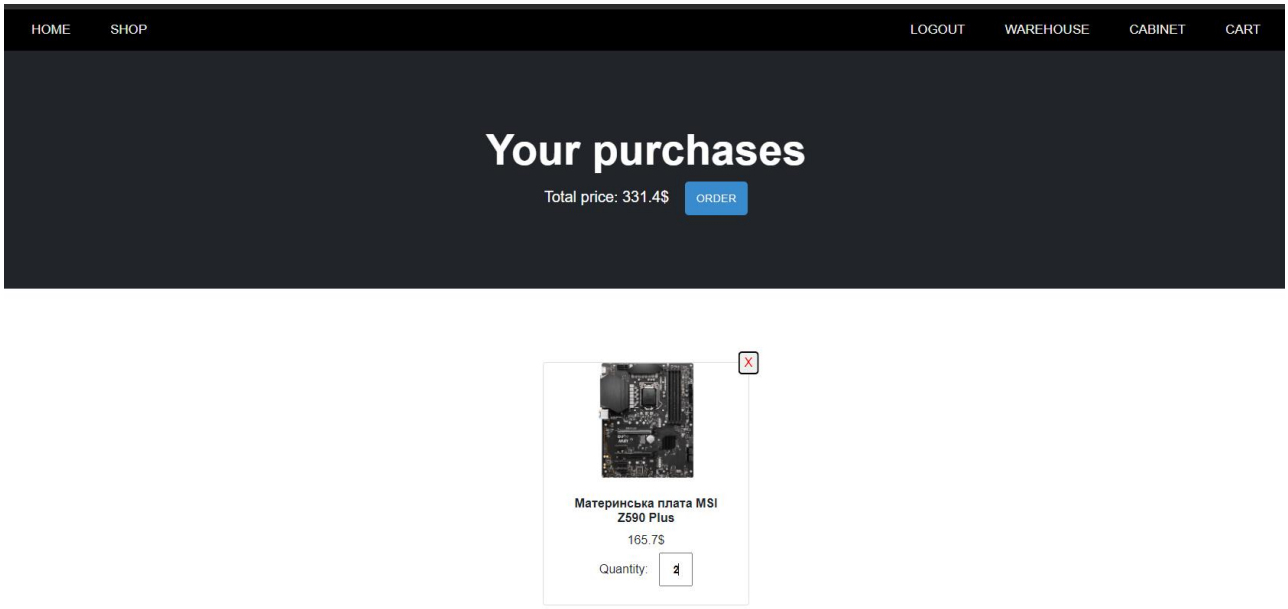

Рис. 3.15. Вигляд корзини в якій знаходяться товари

Натиснемо **order**, нам перекине головну сторінку. Щоб дізнатися чи дійсно було виконано наше замовлення перейдемо до cabinet, тут зберігаються наші покупки (рис. 3.16).

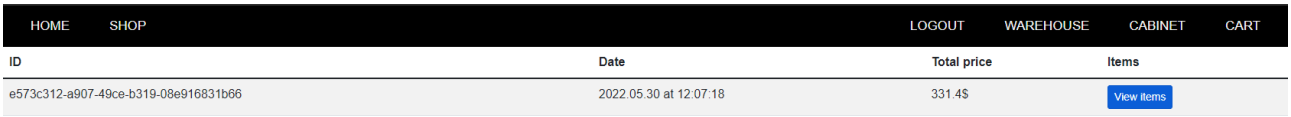

Рис. 3.16. Вигляд кабінету на якому зберігаються наші покупки

Подивитися, які товари були замовлені та їх опис можна через кнопку view items (рис. 3.17).

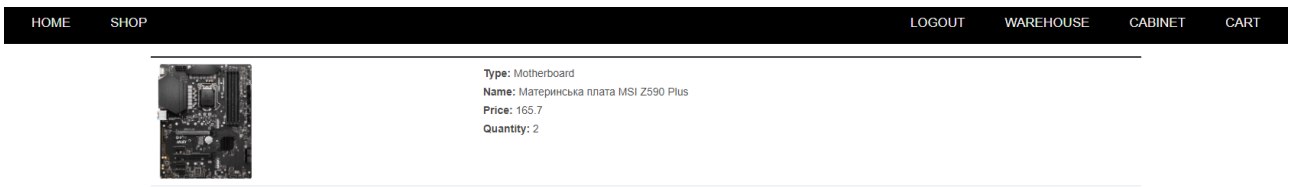

Рис. 3.17. Вигляд товарів, які були замовлені

ВИСНОВОК

В результаті виконання даної дипломного проекту було розроблено вебсайт інтернет-магазин. Був проведений аналіз для використання актуальних технологій, було приділено особливу увагу безпеці роботи сайту. Також був проведений аналіз предметної області та перегляд конкурентів для покращення даного застосунку.

Використання мікросервісної архітектури для більшої гнучкості, легшому розгортанню певного мікросервісу. Основною ідеєю мікросервісів є незалежне розгортання компонентів програми, що дає вам гнучкість вносити зміни лише до певних компонентів, не впливаючи на інші частини програми/системи, і якщо потрібно масштабувати лише певний компонент без масштабування всього . Використання statelessness REST задля більшої безпеки даних, які користувач надає до серверу. Statelessness означає, що кожен запит HTTP відбувається в повній ізоляції. Коли клієнт робить HTTP-запит, він включає всю інформацію, необхідну серверу для виконання запиту. Сервер ніколи не покладається на інформацію з попередніх запитів від клієнта. Якщо така інформація важлива, клієнт надішле її як частину поточного запиту.

Використано певну технологію авторизації. Ідея полягає в тому, що частина потоку перенаправлення, яка взаємодіє з кінцевою точкою авторизації, буде використовуватися лише для запиту та отримання коду авторизації, і що ця взаємодія буде захищена PKCE, щоб запобігти зловживанням зловмисників із кодом. Отримавши код авторизації, JavaScript продовжить викупити його кінцевою точкою токена – так само, як це роблять клієнти для мобільних пристроїв і комп’ютерів.

Побудовано систему, що відповідає поставленому завданню.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. GlobalLogic: Мікросервісна архітектура для початківців. URL: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/
2. Medium: Мікросервісна архітектура. URL: https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d
3. Microsoft: Microservice architecture style. URL: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices
4. Бородкін Г. О., Бородкіна І. Л. Інженерія програмного забезпечення: навч. посібник, 2018. С. 90-94.
5. DreamFactory: 7 Key Benefits of Microservices. URL: https://blog.dreamfactory.com/7-key-benefits-of-microservices/
6. Vuejs: Frequently Asked Questions. URL: https://vuejs.org/about/faq.html
7. Wikipedia: Vue.js. URL: https://en.wikipedia.org/wiki/Vue.js
8. Monocubed: What are the Advantages of Vue js Framework in Web Development. URL: http://monocubed.com/blog/advantages-of-vue-js/
9. Spring: Spring boot. URL: https://spring.io/projects/spring-boot
10. PostgreSQL: PostgreSQL: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/
11. Guru99: What is PostgreSQL? Introduction, Advantages & Disadvantages. URL: https://www.guru99.com/introduction-postgresql.html
12. MongoDB: Advantages of MongoDB. URL: https://www.mongodb.com/advantages-of-mongodb
13. Varonis: The Difference Between Active Directory and LDAP. URL: https://www.varonis.com/blog/the-difference-between-active-directory-and-ldap
14. Dzone: What Keycloak Is and What It Does? URL: https://dzone.com/articles/what-is-keycloak-and-when-it-may-help-you

ДОДАТОК А. ТЕКСТ ПРОГРАМИ

```
<template>
  <div>
   <!-- Header with image -->
   <div class="main-img container-fluid">
    <div class="fs-3 opacity-25 d-flex justify-content-center text-
secondary padding-top">You can find everything
    </div>
   </div>

   <hr>

   <!-- Images-->
   <div class="container-fluid">
    <div class="row g-2">
     <div class="col-6">
      <router-link to="/shop">
       <div class="spec-container border bg-light bg-img-
electronics"></div>
      </router-link>
     </div>
     <div class="col-6">
      <router-link to="/shop">
       <div class="spec-container border bg-light bg-img-
clothes"></div>
      </router-link>
     </div>

     <div class="col-6">
      <router-link to="/shop">
       <div class="spec-container border bg-light bg-img-
food"></div>
      </router-link>
     </div>
     <div class="col-6">
      <router-link to="/shop">
       <div class="spec-container border bg-light bg-img-
sport"></div>
      </router-link>
     </div>

     <div class="col-6">
      <router-link to="/shop">
       <div class="spec-container border bg-light bg-img-
toys"></div>
```

```
        </router-link>
       </div>
       <div class="col-6">
        <router-link to="/shop">
         <div class="spec-container border bg-light bg-img-
books"></div>
        </router-link>
       </div>
     </div>
   </div>

   <hr>

   <!-- footer -->
   <footer class="container-fluid margin-bottom">
    <div class="container-md">
     <div class="row">

      <div class="col-4">
       <div class="row">
        <div class="small d-flex justify-content-
center"><strong>Contact us</strong></div>
        <div class="col">
         <a href="https://www.facebook.com/">
          <i aria-hidden="true" class="fa fa-facebook fa-2x p-3"
style="color:black"></i>
         </a>
        </div>
        <div class="col">
         <a href="https://www.youtube.com/">
          <i aria-hidden="true" class="fa fa-youtube fa-2x p-3"
style="color:black"></i>
         </a>
        </div>
        <div class="col">
         <a href="https://www.instagram.com/">
          <i aria-hidden="true" class="fa fa-instagram fa-2x p-3"
style="color:black"></i>
         </a>
        </div>
        <div class="col">
         <a href="https://www.twitter.com/">
          <i aria-hidden="true" class="fa fa-twitter fa-2x p-3"
style="color:black"></i>
         </a>
```

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

<div class="col-8">
  <div class="row">
    <div class="col d-flex justify-content-center">
      <ul>
        <li class="small"><strong>Something</strong></li>
        <li class="small">link</li>
        <li class="small">link</li>
        <li class="small">link</li>
        <li class="small">link</li>
        <li class="small">link</li>
        <li class="small">link</li>
        <li class="small">link</li>
      </ul>
    </div>
    <div class="col d-flex justify-content-center">
      <ul>
        <li class="small"><strong>Something</strong></li>
        <li class="small">link</li>
        <li class="small">link</li>
      </ul>
    </div>
    <div class="col d-flex justify-content-center">
      <ul>
        <li class="small"><strong>Something</strong></li>
        <li class="small">link</li>
        <li class="small">link</li>
        <li class="small">link</li>
        <li class="small">link</li>
        <li class="small">link</li>
      </ul>
    </div>
    <div class="col d-flex justify-content-center">
      <ul>
        <li class="small"><strong>Something</strong></li>
        <li class="small">link</li>
        <li class="small">link</li>
        <li class="small">link</li>
      </ul>
    </div>
  </div>
</div>
```

```
        </div>
      </div>
    </footer>
  </div>
</template>

<style scoped>
@import "bootstrap/dist/css/bootstrap.min.css";
@import "font-awesome";
body, html {
  height: 100%
}
body, h1, h2, h3, h4, h5, h6 {
  font-family: sans-serif
}
.main-img {
  height: 700px;
  background-repeat: no-repeat;
  background-size: cover;
  background-image: url("../pictures/main.jpg");
  min-height: 100%;
}
.bg-img-electronics {
  background-image: url("../pictures/electronics.jpg");
  background-size: cover;
}
.bg-img-clothes {
  background-image: url("../pictures/clothes.jpg");
  background-size: cover;
}
.bg-img-sport {
  background-image: url("../pictures/sport.jpg");
  background-size: cover;
}
.bg-img-food {
  background-image: url("../pictures/food.jpg");
  background-size: cover;
}
.bg-img-books {
  background-image: url("../pictures/books.jpg");
  background-size: cover;
}
.bg-img-toys {
  background-image: url("../pictures/toys.jpg");
```

```
  background-size: cover;
}
.spec-container {
  width: 100%;
  height: 0;
  padding-bottom: 60%;
}
.padding-top {
  padding-top: 7%;
}
.margin-bottom {
  margin-bottom: 5%;
}
li {
  list-style-type: none;
  padding: .1em;
}
footer {
  padding-top: 5%;
}
</style>
```

Так як код дуже громіздкий, краще використати github для його перегляду.

https://github.com/VucCPaK/graduation_project