Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Батальського Микити Аркадійовича*
(ПІБ)

академічної групи *121-18-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка інтернет-магазину одноразових електронних сигарет*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Реута О. В.*			
розділів:				
спеціальний	*доц. Реута О. В.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-18-1* (група) *Батальського Микити Аркадійовича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка інтернет-магазину одноразових електронних сигарет*

затверджена наказом ректора НТУ «ДП» від *18.05.2022р.* № *268-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПО й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *доц. Реута О.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Батальський М. А.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 91 с., 32 рис., 5 табл., 3 дод., 20 джерел.

Об'єкт розробки: інтернет-магазин одноразових електронних сигарет (ВЕ).

Мета кваліфікаційної роботи: розробка ВЕ для інтернет-магазину, користувач може зареєструватись та увійти на сайті, переглядати список товарів, які магазин може запропонувати, додати у кошик і зробити замовлення. Адміністратори та менеджери можуть переглядати список усіх клієнтів, що зареєстровані, та власноруч реєструвати нових, якщо це потрібно, а також мають змогу додавати новий товар.

У вступі виконується аналіз сучасного стану проблеми, уточняється постановка завдання, мета кваліфікаційної роботи та галузь її застосування, обґрунтовується актуальність теми.

У першому розділі проводиться дослідження предметної області та існуючих рішень, визначається актуальність завдання та призначення розробки, розроблюється постановка завдання.

У другому розділі обирається платформа для розробки, виконується проектування програми і її розробка, наводиться опис структури функціонування системи, описується робота програми.

В економічному розділі визначається трудомісткість розробленого програмного продукту, проводиться підрахунок вартості роботи по створенню застосунку та розраховується час на його створення.

Практичне значення полягає в розробці серверної частини веб-додатку інтернет-магазину, що дозволить клієнтам купувати товар, а бізнесу - заробляти.

Актуальність програмного продукту визначається великим інтересом до електронних сигарет у людей. Кожен магазин мусить мати свій сайт для продажу у двадцять першому столітті. Розроблюваний додаток допоможе клієнтам замовляти їх улюблений продукт онлайн й отримувати його там, де йому зручно, навіть якщо знаходиться в іншому місті, порівняно з фізичним розташуванням магазину.

Список ключових слів: ПРОГРАМА, ІНТЕРНЕТ-МАГАЗИН, САЙТ, БРАУЗЕР, СЕРВЕР.

ABSTRACT

Explanatory note: 91 p., 32 figs., 5 tabl., 3 appx., 20 sources.

Object of development: online store of disposable electronic cigarettes (BE).

The purpose of the qualification work: development of BE for the online store, the user can register and log in to the site, view the list of products that the store can offer, add to cart and place an order. Administrators and managers can view a list of all registered customers and manually register new ones if needed, and have the ability to add new products.

The introduction analyzes the current state of the problem, clarifies the problem, the purpose of the qualification work and the scope of its application, substantiates the relevance of the topic.

In the first section the research of the subject area and existing decisions is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed.

In the second section, the platform for development is selected, the program is designed and developed, a description of the structure of the system, describes the operation of the program.

In the economic section, the complexity of the developed software product is determined, the cost of work on creating the application is calculated and the time for its creation is calculated.

Of practical importance is the development of the server part of the web application of the online store, which will allow customers to buy goods and businesses - to earn.

The relevance of the software product is determined by the great interest in e-cigarettes in humans. Every store should have its own site for sale in the twenty-first century. The developed application will help customers to order their favorite product online and get it where it is convenient, even if it is in another city, compared to the physical location of the store.

Keywords: PROGRAM, ONLINE-SHOP, SITE, BROWSER, SERVER.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

API – інтерфейс для програмування застосунків;

БД – база даних;

ООП – об'єктно-орієнтоване програмування;

ОС – операційна система;

ПК – персональний комп’ютер;

ПЗ – програмне забезпечення;

ІТ – інформаційні технології;

BE – Back-End;

MVC – ModelViewController.

ВСТУП

Продаж був присутній усе життя, з першу це були просто люди, які пропонували «обмін» (твої гроші на його товар), на разі йде двадцять перше століття й майже у кожного магазину є свій сайт, де користувач може безпосередньо замовити той товар, що є у продавця. Для клієнтів це дуже зручно, бо для цього їм не потрібно кудись йти та шукати, де цей товар продається, вони можуть просто знайти в інтернеті сайт, обрати товар, який їм потрібно, додати його до кошику, написати адресу відправлення у форму та, в решті решт, підтвердити своє замовлення. Для створення замовлення, зазвичай, клієнтам потрібно зареєструватись на сайті та увійти в свій аккаунт. За винятком створення замовлення та перегляду товарів, у них є можливість змінювати свій профіль, наприклад, пароль.

Таким чином, завданням даного дипломного проекту є створення BE-частини інтернет-магазину для клієнтів, що має свою БД, де й зберігаються усі дані про користувачів, товари, кошики користувачів та їх замовлення.

Завдання даної кваліфікаційної роботи та об'єкт його діяльності безпосередньо пов'язані з напрямом підготовки «Інженерія програмного забезпечення» та відповідає узагальненій тематиці кваліфікаційних робіт і переліку зазначених виробничих функцій, типових задач діяльності, умінню та компетенціям, якими повинні володіти бакалаври напряму 121 «Інженерія програмного забезпечення» галузі знань «Інформаційні технології».

Завдання даного дипломного проекту та об'єкт його діяльності безпосередньо пов'язані з напрямом підготовки та відповідає узагальненій тематиці кваліфікаційних робіт і переліку зазначених виробничих функцій, типових задач діяльності, умінню та компетенціям, якими повинні володіти бакалаври напряму 121 «Інженерія програмного забезпечення».

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості з предметної галузі

У даному етапі розвитку нашої цивілізації набуває нового та дуже важливого значення мережа Інтернет і все, що з нею пов‘язано. Ми з вами у реальному часі можемо спостерігати, як Інтернет стає основним генератором світових тенденцій. Інтернет у собі зберігає безліч інформації, настільки багато, що людський мозок не здатний усю її зберігати. Зараз дуже важко собі уявити наше життя без Інтернету, бо саме завдяки йому ми можемо отримати інформацію, яка нам потрібна, просто пошукавши у браузері, використовуючи, наприклад, телефон або ПК. Кожна людина активно користується Інтернетом вдома, на роботі, загалом – будь де, де є зв’язок, що дало нам можливість практично весь час перебувати онлайн. Поява Інтернету також внесла зміни й у магазини. Інтернет розвивався, з'явилася можливість використовувати всі його досягнення так, як заманеться. Таким чином, магазини почали розробляти свої сайти, мета яких – розширити базу клієнтів, бо більше клієнтів – більше замовлень. До появи Інтернету була лише одна можливість щось придбати – прийти до магазину й купити. Завдяки Інтернету люди можуть без всіляких проблем дистанційно замовити свій товар. На разі, у вік інформаційних технологій, розробка інтернет-магазину є дуже актуальною, як для бізнесу, так і для клієнтів.

Перший веб-сервер і браузер був створений Тім Бернерс-Лі у 1990 році, що у 1991 році вже був відкритий для комерційного використання. У 1994 році відбулися інші досягнення, наприклад, онлайн-банкінг та відкриття інтернет-магазину піци «Pizza Hut». Netscape представила SSL-шифрування даних, переданих в мережі у тому ж році, яке стало критично необхідним для безпеки інтернет-магазинів. Крім того, у 1994 році німецька компанія Intershop представила свою першу систему інтернет-магазинів. Amazon запустила свій перший інтернет-магазин вже у 1995 році, а в 1996 році з'явився eBay. З кожним

роком ця тема поширювалась дедалі більше й так само швидко вдосконалювалась, що дозволяло створювати нові сайти інтернет-магазинів, робити їх більш зручними, як для клієнтів, так і для співробітників магазину, що проводили моніторинг залишків товару та обробку замовлень клієнтів.

Авжеж, це було б дуже важко, потрібно було б завжди перераховувати залишки товару й оновлювати їх кількість на сайті, саме тому БД є невід'ємною частиною інтернет-магазину. БД зберігає у собі усі дані: дані клієнтів, дані щодо товарів, дані кошиків, дані замовлень.

1.2 Призначення розробки та галузь її застосування

Призначенням розробки є створення інтернет-магазину одноразових електронних сигарет.

Даний продукт має використовуватись клієнтами, тобто споживачами, та безпосередньо співробітниками магазину, для забезпечення швидкого створення замовлення й подальшого його відправлення.

Призначення розробки полягає у наданні потенціальним користувачам наступних переваг автоматизованої системи:

– безперервна робота розроблюваної мережі;

– дає можливість зареєструватися та використовувати сайт для замовлення незалежно від місця дислокації;

– дозволяє оглядати список товарів та їх додавання у кошик;

– дає можливість менеджерам магазину додавати новий продукт на сайт;

– дозволяє клієнтам змінювати дані для входу на сайт у свій профіль;

– дає можливість менеджерам створювати власноруч клієнтів, наприклад, аккаунт нового менеджера магазину.

1.3 Підстави для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

– освітня програма 121 «Інженерія програмного забезпечення»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № 268-с від 18.05.2022 р;

завдання на кваліфікаційну роботу на тему «Розробка інтернет-магазину одноразових електронних сигарет.».

1.4. Постановка завдання

Завданням є розробка інтернет-магазину одноразових електронних сигарет.

Кінцевий продукт передбачає наступний функціонал:

– сторінки входу та реєстрації аккаунту споживача;

– перегляд списку усіх товарів, що можна купити;

– можливість додавання товару до кошика, при умові, що користувач увійшов у аккаунт;

– можливість редагування кошика;

– можливість створення замовлення;

Для розробки було обрано мову програмування JAVA, стек технологій: Spring та Hibernate.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Кінцевий продукт має дотримуватися наступних функціональними вимог:

– якщо користувач не увійшов, то замість «Додати у кошик» у нього повинно бути «Увійти в аккаунт»;

– користувач мусить мати змогу зареєструватись на сайті;

– реєстрація неможлива, якщо вже існує користувач з такою самою поштою;

– якщо користувач переходить на сторінку кошику, коли він пустий, то друкується повідомлення про це;

– якщо користувач переходить на сторінку кошику, у якому є товар, то він бачить таблицю з товарами, їх кількістю, ціною, а також друкується сума до сплати, з’являється місце для введення адреси відправлення й кнопка для створення замовлення;

– користувач має можливість редагувати власні дані;

– менеджери можуть передивлятись список усіх користувачів та редагувати деякі їх дані, ця сторінка доступна тільки для менеджерів;

– менеджери повинні мати змогу створювати власноруч користувачів;

– менеджери можуть додавати на сайт нові товари, ця функція доступна тільки для них;

1.5.2.Вимоги до інформаційної безпеки

Для забезпечення надійного функціонування системи необхідно реалізувати наступні вимоги:

– перевірка введеної пошти: пошта повинна бути унікальна для реєстрації нового аккаунту;

– захист від несанкціонованого доступу до функціоналу за допомогою Spring Security;

– обробка виняткових ситуацій;

– виведення повідомлень у разі виникнення помилок;

1.5.3. Вимоги до складу та параметрів технічних засобів

Для коректного функціонування кінцевого продукту необхідними технічними умовами є:

– операційна система Windows 10, Linux, MacOS, Android або IOS;

– наявність будь-якого сучасного браузера, наприклад, Google Chrome або Opera;

– підтримка високошвидкісного Інтернету;

– обсяг оперативної пам’яті (ОЗУ) не менш 2 ГБ.

1.5.4. Вимоги до інформаційної та програмної сумісності

Java – мова програмування, що була обрана для розробки BE-частини інтернет-магазину. Код повинен бути написаний за всіма стандартами та згідно з договором про стиль коду мови Java. Для розробки програмного застосунку було використано середу розробки IntellijIDEA та фреймворк для автоматизації збору проекту на основі опису їх структури у файлі pom.xml.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми.

Результатом виконання цієї кваліфікаційної роботи має бути програмний додаток, який у активній фазі дає доступ користувачам до сайту з товаром, який продається. Додаток має приймати запити на контролери та повертати необхідну відповідь на фронт-частину.

2.2. Опис застосованих математичних методів.

У розробленій системі використовуються стандартні математичні методи(складання) для суми до сплати замовлення та кількості товару.

2.3. Опис використаної архітектури та шаблонів проектування.

У даному розробленому проекті було застосовано один патерн проектування – Builder. Це породжувальний патерн проектування, що дає змогу створювати складні об’єкти крок за кроком. Будівельник дає можливість використовувати один і той самий код будівництва для отримання різних відображень об’єктів. Був застосований завдяки Lombok, анотації @Builder над класом сутності та DTO.

Спочатку було задумано використати @RestController, проте, через необхідність показати у браузері роботи програми, довелось використовувати @Controller (MVC – ModelViewController). MVC - схема поділу даних програми та керуючої логіки на три окремі компоненти: модель, уявлення та контролер — таким чином, що модифікація кожного компонента може здійснюватися незалежно. Модель надає дані та реагує на команди контролера, змінюючи свій стан.

Для виконання мети проекту використовувались фреймворки Spring (Security, Web, Thymeleaf, Data), H2, Flyway, PostgreSql, Lombok.

Завдяки фреймворку Spring розробник може використовувати вже існуючий функціонал, таким чином – економиться час та сили на розробку більш унікальних функцій.

H2, Flyway, PostgreSql – це вже фреймворки для роботи з БД, Flyway, наприклад, дозволяє використовувати скрипти для створення таблиць й взагалі для операцій з ними. Фреймворк PostgreSql повинен бути по тій причині, що використовував саме цю БД.

Так як було потрібно продемонструвати роботу застосунку, було вирішено використовувати Thymeleaf - шаблонізатор Java XML/XHTML/HTML5, який може працювати як у веб-середовищі, так і не в веб-середовищі. Він краще підходить для обслуговування XHTML/HTML5 на рівні представлення веб-додатків на основі MVC(ModelViewController). Завдяки ньому був написаний мінімальний фронт, який відображає роботу BE інтернет-магазину

2.4. Опис використаних технологій та мов програмування.

Дана інформаційна система була розроблена з використанням таких технологій:

- Java
- Maven
- Thymeleaf
- Spring Boot
- Lombok
- Flyway
- JQuery
- Stomp
- SockJs

Java

Java — це об'єктно-орієнтована мова програмування. У офіційній реалізації Java-програми компілюються у byte-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Використовування даного язика не залежить від платформи, Java є усюди.

Ключовою особливістю мови Java є те, що його код спочатку транслюється в спеціальний byte-код, сумісний із різними платформами. А потім цей byte-код виконується віртуальною машиною JVM (Java Virtual Machine). В цьому плані Java відрізняється від різних стандартних мов як Python або Ruby, бо їх код відразу ж виконується інтерпретатором. У той же час Java не являє собою й чисто компільовані мовою, як C або C ++.

Подібна архітектура забезпечує крос-платформенність і апаратну переносимість програм на Java, завдяки цьому подібні програми можуть виконуватися на різних платформах – Linux, Windows, Mac і т.д., без компіляції. Для кожної з платформ може бути своя реалізація віртуальної машини JVM, але кожна з них, без винятків, може виконувати один і той самий код.

Особливістю Java є те, що вона підтримує автоматичну збірку сміття(GarbageCollector). А це означає, що не треба власноруч звільняти пам'ять від об'єктів, що вже втратили посилання, тобто більше не будуть використовуватись, як в C ++, так як збирач сміття це зробить автоматично за вас, але все ж, згідно зі своєю логікою обробки.

Рис. 2.1. Логотип Java

Плюси:

– ООП – принцип проектування ПЗ;

– Простий синтаксис, більше схожий на людську мову, аніж на машинний код;

– Надійність, завдяки суворій типізації;

– Крос-платформенність, написав один раз – працює усюди;

– Мультифункціональність;

– Автоматична збірка сміття(видалення із пам’яті неактивних об’єктів, або об’єктів, на які немає посилань);

– Суспільство, що постійно розвивається;

– Потужний інструментарій для розробки Android-застосунків.

Мінуси:

– Використовує багато ОЗУ;

– Платне комерційне використання;

– Швидкість порівняно менша, ніж у C та C++.

Maven

Засіб автоматизації роботи з програмними проектами, який спочатку використовувався для Java проектів. Використовується для управління (management) та складання (build) програм. За принципами роботи кардинально відрізняється від Apache Ant, та має простіший вигляд щодо build-налаштувань, яке надається в форматі XML. XML-файл описує проект, його зв'язки з зовнішніми модулями і компонентами, порядок будування (build), папки та необхідні плагіни. Сервер із додатковими модулями та додатковими бібліотеками розміщується на серверах.

Рис. 2.2. Логотип Maven

Структура проекту:

– src/main/java – директорія Java класів;

– src/main/resources – директорія конфігураційних файлів та template-ів;

– src/test/java – директорія тестів.

Thymeleaf

Так як було потрібно продемонструвати роботу застосунку, було вирішено використовувати Thymeleaf - шаблонізатор Java XML/XHTML/HTML5, який може працювати як у веб-середовищі, так і не в веб-середовищі. Він краще підходить для обслуговування XHTML/HTML5 на рівні представлення веб-додатків на основі MVC(ModelViewController). Завдяки ньому був написаний мінімальний фронт, який відображає роботу BE інтернет-магазину, хоча й у дуже примітивному вигляді, проте клієнт може заповнити форми на сайті, додати товар у кошик, створити замовлення.

Рис. 2.3. Логотип Thymeleaf

Spring Boot

Spring Boot — це веб-фреймворк, що спрощує написання клієнт-серверних веб-застосунків за допомогою потужного вбудованого функціоналу.

Даний фреймворк може розглядатись як сукупність взаємно незалежних фреймворків, що можуть працювати як окремо один від одного, так і разом, забезпечуючи максимальну ефективність роботи. Найважливіші складові Spring:

– Inversion of Control: конфігурування компонентів застосунку та керування життєвим циклом об’єктів;

– Spring Security - фреймворк, що забезпечує безпеку застосунку: інструментарій конфігурації процесів аутентифікації та авторизації;

– Фреймворк доступу даних: у Spring Boot таку роль виконує ORM Spring Data Jpa, за допомогою якої можна будувати інтуїтивно зрозумілі та прості запроси до серверу БД;

– Фреймворк для роботи з транзакціями: інструментарій налаштування транзакцій;

– Фреймворк MVC: інструментарій для налаштування HTTP – запитів;

– Тестування: Spring Boot надає широкі можливості для застосування різноманітних фреймворків тестування, найпопулярніший - Junit.

Рис. 2.4. Логотип Spring Boot

Lombok

Lombok – це бібліотека для скорочення коду в класах та розширення функціональності мови Java. Підключається до середовища розробки (IDE) або інструменту складання програм Maven, Gradle як плагін.

Бібліотека отримала назву на честь індонезійського острова Ломбок, розташованого неподалік від острова Ява. У перекладі з індонезійського Lombok означає «перець чилі»: за аналогією з приправою, бібліотека покликана підвищити якість Java-коду.

Рис. 2.5. Логотип Lombok

Flyway

Flyway оновлює версії бази даних за допомогою міграцій. Міграції можна писати на SQL (з синтаксисом, специфічним для конкретної СУБД) або Java. У даному ПЗ використовувався тільки для початкового створення таблиць й залежностей.

Рис. 2.6. Логотип Flyway

JQuery

jQuery — набір функцій JavaScript, що фокусується на взаємодії JavaScript та HTML. Бібліотека jQuery допомагає легко отримувати доступ до будь-якого елемента DOM, звертатися до атрибутів та вмісту елементів DOM, маніпулювати ними. Також бібліотека jQuery надає зручний API для роботи з AJAX.

Рис. 2.7. Логотип JQuery

Stomp

Simple Text Oriented Message Protocol, раніше відомий як TTMP, є простим текстовим протоколом, розроблений для роботи з проміжним програмним забезпеченням, орієнтованим на повідомлення.

Рис. 2.8. Логотип Stomp

SockJs

SockJs - це JavaScript бібліотека, яка забезпечує двосторонній між доменний канал зв'язку між клієнтом та сервером. Тобто SockJs імітує WebSocket API, під капотом він спочатку намагається використати нативну реалізацію WebSocket API.

Рис. 2.9. Логотип SockJs

2.5. Опис структури програми та алгоритмів її функціонування.

Рис. 2.10. Структура проекту

Дана програма побудована на основі моно-модульної архітектури, що означає, що класи розподілені по папкам, згідно з їхнім функціоналом. Таким чином, у папці config – конфігурація проекту, у папці controllers – усі контролери, що використовувались, у папці dao – репозиторії сутностей(CRUD операції), у папці domain – сутності(таблиці БД), у папці dto – об’єкти, що передаються по MVC, у папці service – усі сервіси, що використовувались у об’єднанні контролерів та репозиторіїв. Клас, що запускає програму знаходиться у корені.

Далі, у папці resources знаходяться усі ресурси, що необхідні ПЗ, такі як html сторінки, скрипти для БД та js файл для додавання продукту з сайту.

```
public interface UserRepository extends JpaRepository<User, Long> {
    User findFirstByName(String name);
}

public interface OrderRepository extends JpaRepository<Order, Long> {
}

public interface CigaretteRepository extends JpaRepository<Cigarette, Long> {
}

public interface CartRepository extends JpaRepository<Cart, Long> {
}
```

Рис 2.11. Репозиторії

Як можна бачити на цих скріншотах, усі репозиторії успадковують інтерфейс JpaRepository, який включає в себе найнеобхідніші методи для роботи з базою даних, такі як:

findAll(): знаходить усі записи в таблиці, еквівалентний запису «SELECT * FROM `*ENTITY*`»

deleteAll(): видаляє усі записи з таблиці, еквівалентний запису «DELETE FROM `*ENTITY*`»

deleteById(ID): видаляє з таблиці запис, що відповідає вказаному параметру, еквівалентний запису «DELETE FROM `*ENTITY*` WHERE ENTITY_ID = `*ID*`»

findById(ID): знаходить запис в таблиці, що відповідає вказаному параметру, еквівалентний запису «SELECT * FROM `*ENTITY*` WHERE ENTITY_ID = `*ID*`»

findAlllById(Collection<ID>): знаходить в таблиці всі записи, що відповідають критеріям пошуку, еквівалентний запису «SELECT * FROM `*ENTITY*` WHERE ENTITY_ID IN (Collection<ID>)»

saveAll(Collection<Entity>): зберігає або оновлює існуючі записи на основі вказаних параметрів.

Проте, можна додати й свої методи, якщо це потрібно, таким чином мені знадобився лише один метод - User findFirstByName(String name), так як імена я встановив унікальні, то пошук я зробив по ним, даний метод знаходить першого користувача з даним ім’ям і повертає його об’єкт.

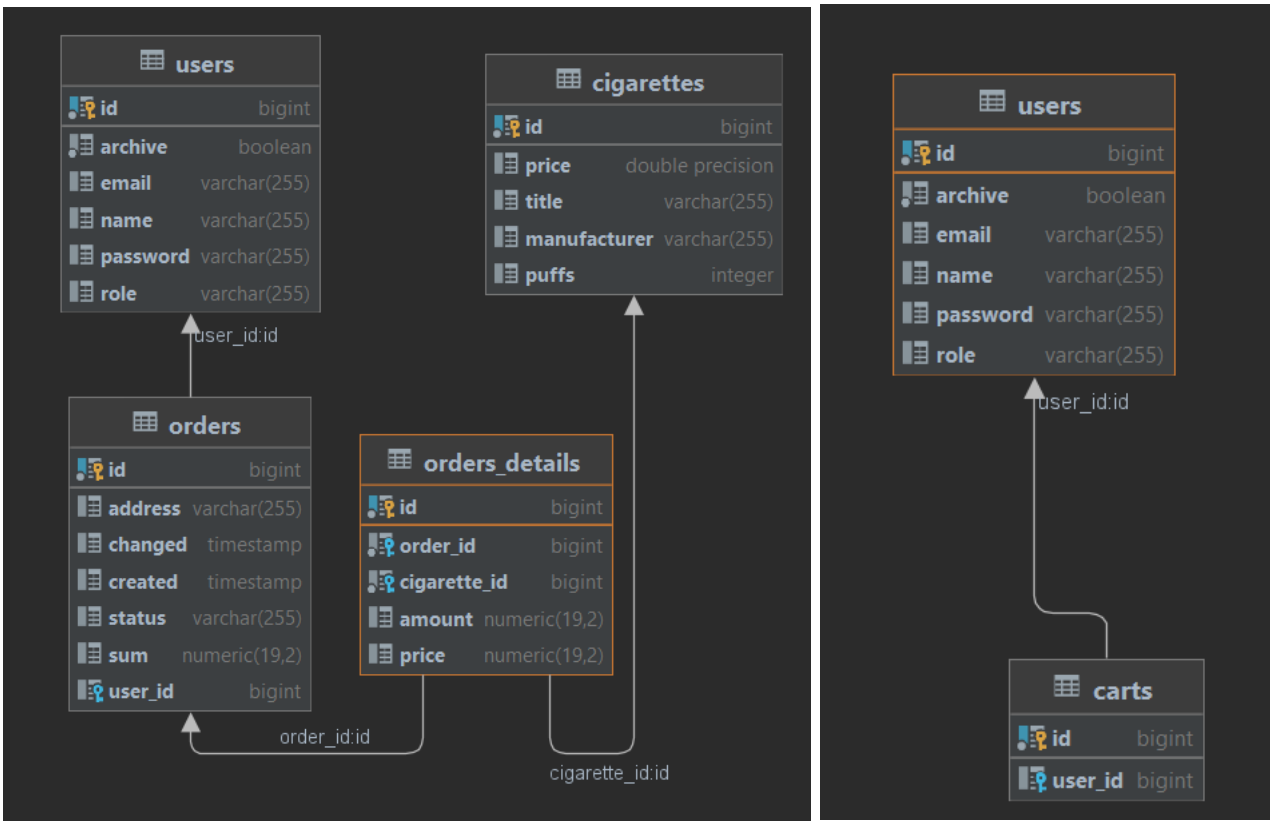

Рис. 2.12. Схеми таблиць та їх зв’язків у БД

Таблиця 2.1

Структура об’єкту users

Назва колонки	Тип	Призначення
id	Bigint	Унікальний ідентифікатор користувача
archive	Varchar	Статус користувача
email	Varchar	Пошта користувача, має бути унікальною
name	Varchar	Ім’я користувача, має бути унікальною
password	Varchar	Пароль користувача у зашифрованому вигляді

role	Varchar	Роль користувача

Таблиця 2.2

Структура об’єкту carts

Назва колонки	Тип	Призначення
id	bigint	Унікальний ідентифікатор кошику
user_id	bigint	Унікальний ідентифікатор користувача, що створив цей кошик. Зовнішній ключ.

Таблиця 2.3

Структура об’єкту cigarettes

Назва колонки	Тип	Призначення
id	bigint	Унікальний ідентифікатор сигарети
price	double precision	Ціна сигарети
title	varchar	Назва сигарети
manufacturer	varchar	Виробник сигарети
puffs	integer	Кількість затяжок

Таблиця 2.4

Структура об’єкту orders

Назва колонки	Тип	Призначення
id	bigint	Унікальний ідентифікатор замовлення
address	varchar	Адреса для відправки замовлення

changed	timestamp	Дата та час зміни статусу замовлення
created	timestamp	Дата та час створення замовлення
status	varchar	Статус замовлення
sum	numeric	Сума замовлення
user_id	bigint	Унікальний ідентифікатор користувача, що створив це замовлення. Зовнішній ключ.

Таблиця 2.5

Структура об’єкту orders_details

Назва колонки	Тип	Призначення
id	bigint	Унікальний ідентифікатор замовлення
order_id	bigint	Унікальний ідентифікатор замовлення. Зовнішній ключ.
cigarette_id	bigint	Унікальний ідентифікатор сигарети, що потрапила у замовлення. Зовнішній ключ.
amount	numeric	Кількість сигарет
price	numeric	Ціна сигарети

Migrations

Дана папка відповідає за маніпуляцію базою даних. Тут зберігаються файли, що мають назву changelog. В таких файлах можна створювати таблиці, наповнювати їх даними, також у цих файлах можна створювати різноманітні об’єкти БД, такі як: індекси, тригери тощо. Формати запису: .xml, .sql. У моєму випадку, там знаходяться скрипти зі створенням таблиць та sequences, додавання

адміністратора з паролем “pass” у зашифрованому вигляді та скрипт на додавання перших позицій товару на продаж.

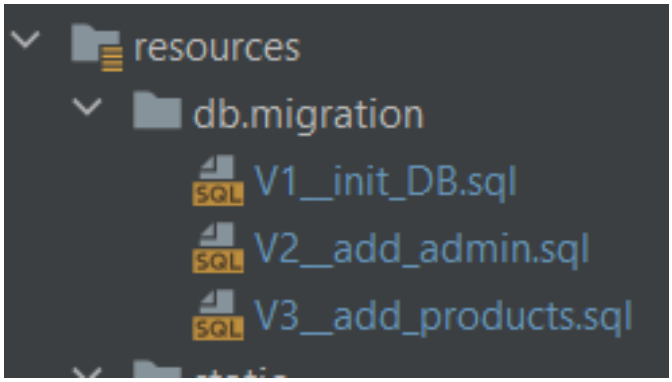

Рис. 2.13. Migrations

Service

Папка Service відповідає за бізнес-логіку застосунку.

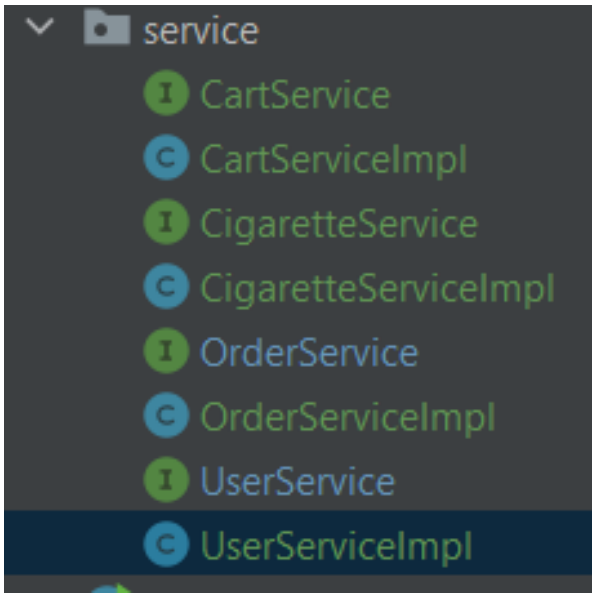

Рис. 2.14. Сервіси ПЗ

UserServiceImpl використовується для керування користувачами. Основні функції:

- Список усіх користувачів;
- Збереження та оновлення даних користувача у БД;
- Перевірка на унікальне ім’я користувача(перед реєстрацією нового);
- Перевірка на унікальну пошту користувача(перед реєстрацією нового);
- Пошук користувача за його ім’ям у БД;

- Оновлення профілю користувача(його може викликати лише той користувач, який увійшов у систему);
- Оновлення даних користувача(доступне тільки менеджерам та адміністратору);
- Метод конвертації сутності у DTO.

OrderServiceImpl використовується для керування замовленнями. Основні функції:

- Збереження замовлення у БД;
- Пошук замовлення по id.

CigaretteServiceImpl використовується для керування сигаретами. Основні функції:

- Список усіх сигарет;
- Додавання до кошику користувача, тільки якщо він увійшов у систему;
- Додати сигарету до списку, тільки для менеджерів та адміністратору сайту;
- Пошук сигарети по id;
- Конвертація у DTO та навпаки.

CartServiceImpl використовується для керування кошику. Основні функції:

- Створити кошик для користувача й додати обраний товар;
- Пошук кошику по імені користувача;
- Прибрати деталь замовлення з кошику;
- Підтвердити замовлення.

Також кожен з цих сервісів має такі спільні методи, як save(), update(), getById(). getAll(), delete() – що забезпечують увесь необхідний CRUD-функціонал.

Dto

DTO (Data transfer object) – об’єкти передачі даних, не несуть у собі ніякої логіки, використовуються лише у якості об’єктів для взаємодії із фронт-частиною.

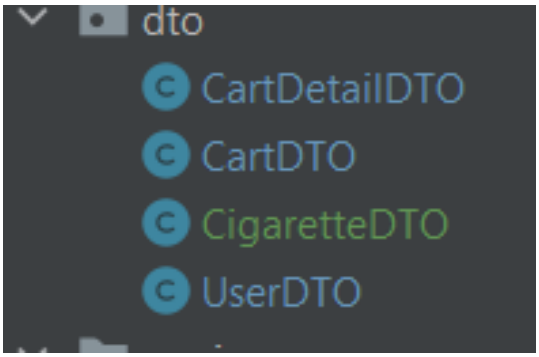

Рис. 2.15. DTO

Controllers

На разі час для контролерів, що безпосередньо оброблюють запит з фронт-частини й звертаються до БД через сервіси, а потім передають відповідь BE клієнтові.

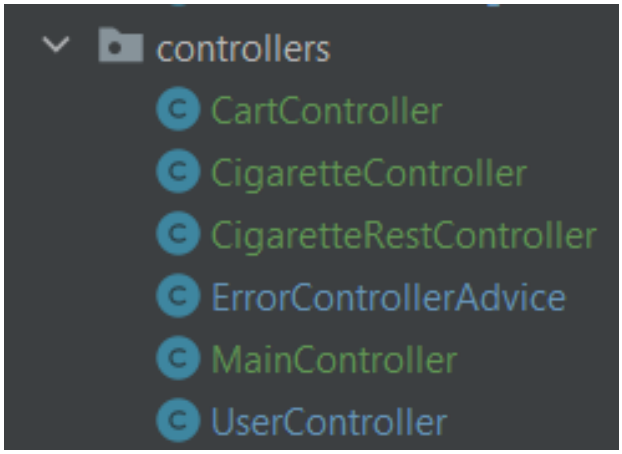

Рис. 2.16. Controllers

MainController – контролер для обробки запитів головної сторінки. Основні api-методи:

- GET:
  - “”, “/” – повертає сторінку index(початкова);
  - “/login” – повертає сторінку login з формою для входу у систему;

- “/login-error” – повертає сторінку login-error, у разі виняткових сценаріїв;
- “/register” – повертає сторінку register з формою для реєстрації.

UserController – контролер для обробки запитів пов’язаних з користувачем. Основні api-методи:

- GET:
  - “/users” – повертає увесь список користувачів, що вже зареєстровані;
  - “/users/new” – повертає сторінку для створення нового користувача;
  - “/users/update/{username}” – у запиті передається унікальне ім’я користувача, знаходиться у БД, конвертується у DTO та повертає сторінку для редагування ролі користувача, доступ є тільки у менеджерів;
  - “/users/profile” – повертає сторінку з даними користувача, що увійшов у систему.
- POST:
  - “/users/new” – отримує об’єкт з фронт-частини, та передає його у метод save, там проходить перевірку на унікальність полів і зберігається, відкриваючи сторінку зі списком усіх користувачів, або повертає помилку на сторінку створення нового користувача;
  - “/users/update” – отримує об’єкт з фронт-частини, знаходить такого користувача у БД, потім йде перевірка і якщо щось змінено – передає цей об’єкт у метод для оновлення інформації щодо користувача, а потім повертається на сторінку “/users”, або нічого не відбувається, якщо нічого не було змінено;
  - “/users/register” – отримує об’єкт з фронт-частини, та передає його у метод save, там проходить перевірку на унікальність

полів і зберігається, відкриваючи сторінку входу, або повертає помилку на сторінку реєстрації;

- “/users/profile” – отримує об’єкт з фронт-частини, перевіряє, чи змінений пароль та його валідацію(password і mathcingPassword мусять бути однаковими), якщо валідація пройдена, то дані у БД оновлюються, якщо ні – повертається така сама сторінка і нічого не оновлюється.

CigaretteController – контролер для обробки запитів пов’язаних з сигаретами. Основні api-методи:

- GET:
  - "/cigarettes" – повертає список усіх сигарет у магазині;
  - "/cigarettes/{id}/cart" – додає сигарету до кошику користувача, якщо він увійшов у систему;
  - "/cigarettes/{id}” – повертає сигарету, знайдену по id.
- POST:
  - “/cigarettes” – додає нову сигарету до списку;
- Message:
  - “/cigarettes” – Message для динамічного додавання сигарету до списку та виводу на екран.

CigaretteRestController – контролер для обробки запитів пов’язаних з сигаретами. Основні api-методи:

- GET:
  - “/api/v1/cigarettes/{id}” – пошук сигарети за її id.
- POST:
  - “/api/v1/cigarettes” – додавання сигарета у список.

CartController – контролер для обробки запитів пов’язаних з кошиками. Основні api-методи:

- GET:
  - "/cart" – повертає кошик користувача;

- "/cart/{id}" – прибирання деталі замовлення.

– POST:

- "/cart" – підтвердження замовлення.

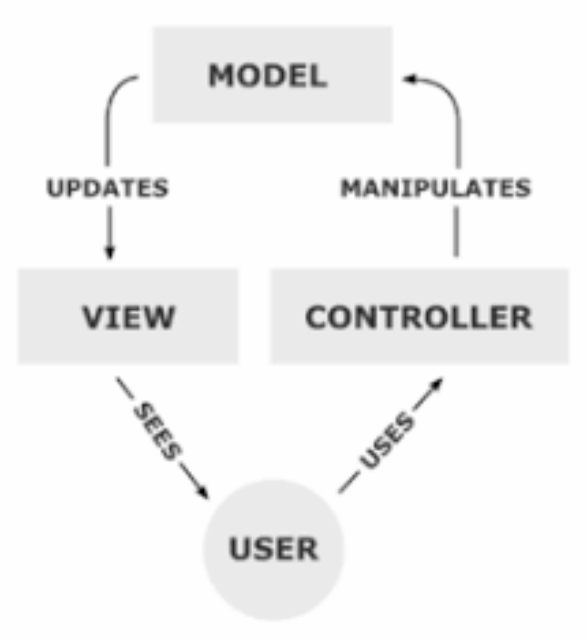

Рис. 2.17. Схема роботи системи

Отже, за схемою, розроблювана система має працювати наступним чином:

1. Клієнтський запит, що був сформований на фронт-частині у форматі джсон надходить на шар контролерів. За допомогою класу DispatcherServlet, Spring направляє запит на відповідний метод відповідного класу-контролеру.

2. Після надходження на відповідний метод, керування приймає відповідний сервіс, куди з контролеру перенаправляються тіло запиту та допоміжні змінні для обробки.

3. В свою чергу сервіси взаємодіють із репозиторіями, формуючи різноманітні SQL-запити відповідно до бізнес логіки.

4. Тим часом, репозиторії містять класи-моделі, що є проекціями таблиць у БД, та власне репозиторії, за допомогою яких отримується доступ до БД та взаємодія з нею.

5. Також варто не забувати про конфігурації, що забезпечує безпеку несанкціонованих доступів до ендпоінтів.

2.6. Обґрунтування та організація вхідних та вихідних даних програми.

Вхідні дані надаються системі у форматі DTO-об’єкту, що формуються на фронт-частині застосунку в залежності від дій користувача. А також в залежності від необхідної інформації, фронт-частина надсилає відповідні запити до розроблюваної системи, котра, в свою чергу, оброблює ці запити та відповідає вихідними даними у форматі DTO-об’єкту або JSON.

2.7. Опис розробленого програмного продукту.

2.7.1. Використані технічні засоби.

В процесі тестування та розробки були використані наступні технічні засоби:

- оперативна пам’ять обсягом 20 гб;
- процесор Intel Core i5 8th Gen;
- накопичувач на жорсткому диску обсягом 150 гб;
- відеоадаптер NVIDIA GeForce MX 150;
- клавіатура та мишка.

Вказані технічні засоби не мають бути обов’язково ідентичних характеристик для безперешкодної роботи системи.

2.7.2. Використані програмні засоби

Під час розробки даного застосунку були використані такі програмні засоби:

– IntelliJ IDEA;

– Git, GitHub;

– PostgreSQL.

IntelliJ IDEA

IntelliJ IDEA – інтегроване середовище розробки програмного забезпечення для багатьох мов програмування, зокрема Java, JavaScript, Python, розроблене компанією JetBrains. Community версія середовища підтримує інструменти для проведення тестування TestNG і JUnit, системи контролю версій CVS, Subversion, Mercurial і Git, засоби складання Maven, Ant, Gradle, мови програмування Java, Scala, Clojure, Groovy, Kotlin і Dart. Підтримується розробка застосунків для мобільних платформ Android. До складу входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, редактор регулярних виразів, XML-редактор, система перевірки коректності коду, система контролю за виконанням завдань і доповнення для імпорту та експорту проектів з Eclipse. Доступні засоби інтеграції з системами відстеження помилок JIRA, Trac, Redmine, Pivotal Tracker, GitHub, YouTrack, Lighthouse.

Ultimate Edition - комерційна версія, що відрізняється наявністю підтримки додаткових мов програмування (наприклад, PHP, Ruby, Python, JavaScript, CoffeeScript, HTML, CSS, SQL), підтримкою технологій Java EE, UML-діаграм, підрахунок покриття коду, можливістю роботи з фреймворками (Rails, Grails, Google Web Toolkit, Spring, Play Framework і Hibernate), засобами інтеграції з Perforce, Microsoft Team Foundation Server і Rational ClearCase.

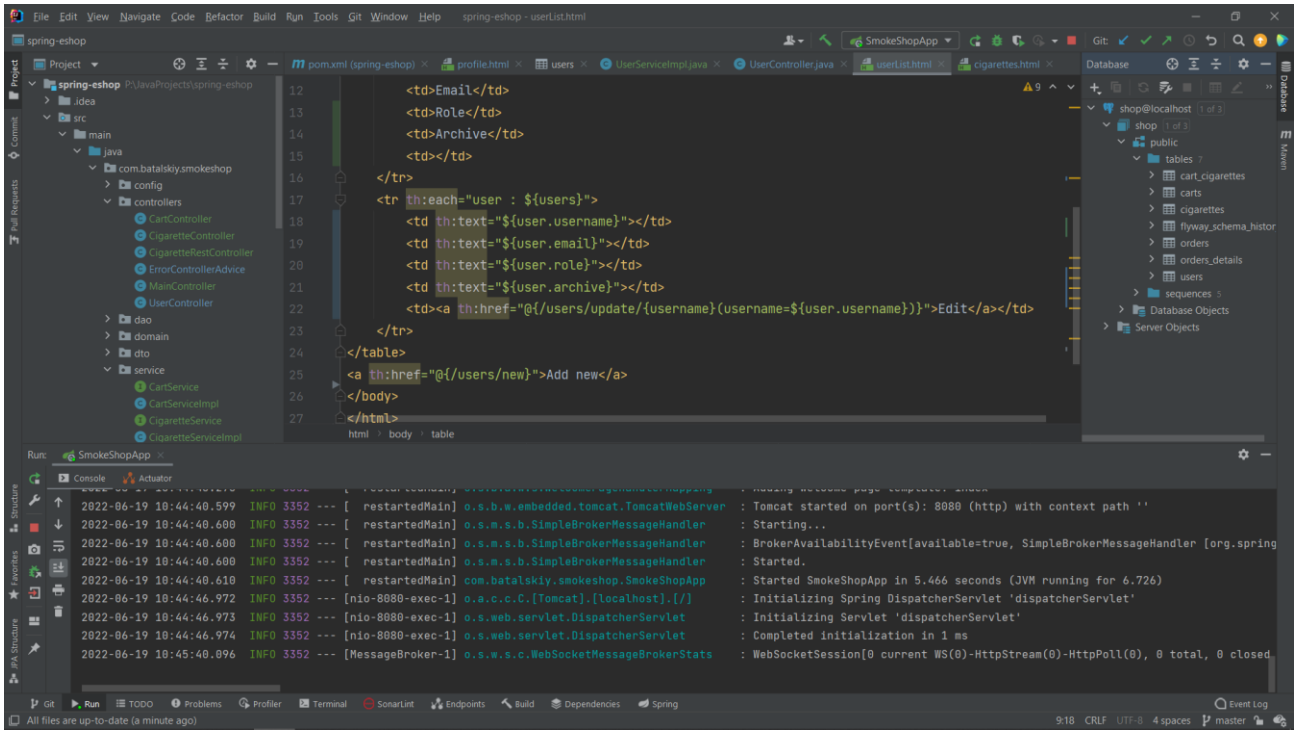

Рис. 2.18. Інтерфейс IntelliJ IDEA

GIT

Git – це система управління розподіленими версіями (DVCS), яка зазвичай використовується для розробки відкритого комерційного та некомерційного ПЗ. DVCS дозволяють отримати повний доступ до кожного файлу, гілки та ітерації проекту та дозволяє кожному користувачеві отримати доступ до повної та самодостатньої історії всіх змін. На відміну від популярних колись централізованих систем управління версіями, «DVCS типу Git не потребує постійного підключення до центрального сховища.

GitHub – це платформа для розміщення коду для контролю версій та співпраці. Він дозволяє спільно працювати над проектами з будь-якого місця та зберігати версії проекту на віддаленому сервері.

PostgreSQL

PostgreSQL – це СУБД(вільна об'єктно-реляційна система управління базами даних).

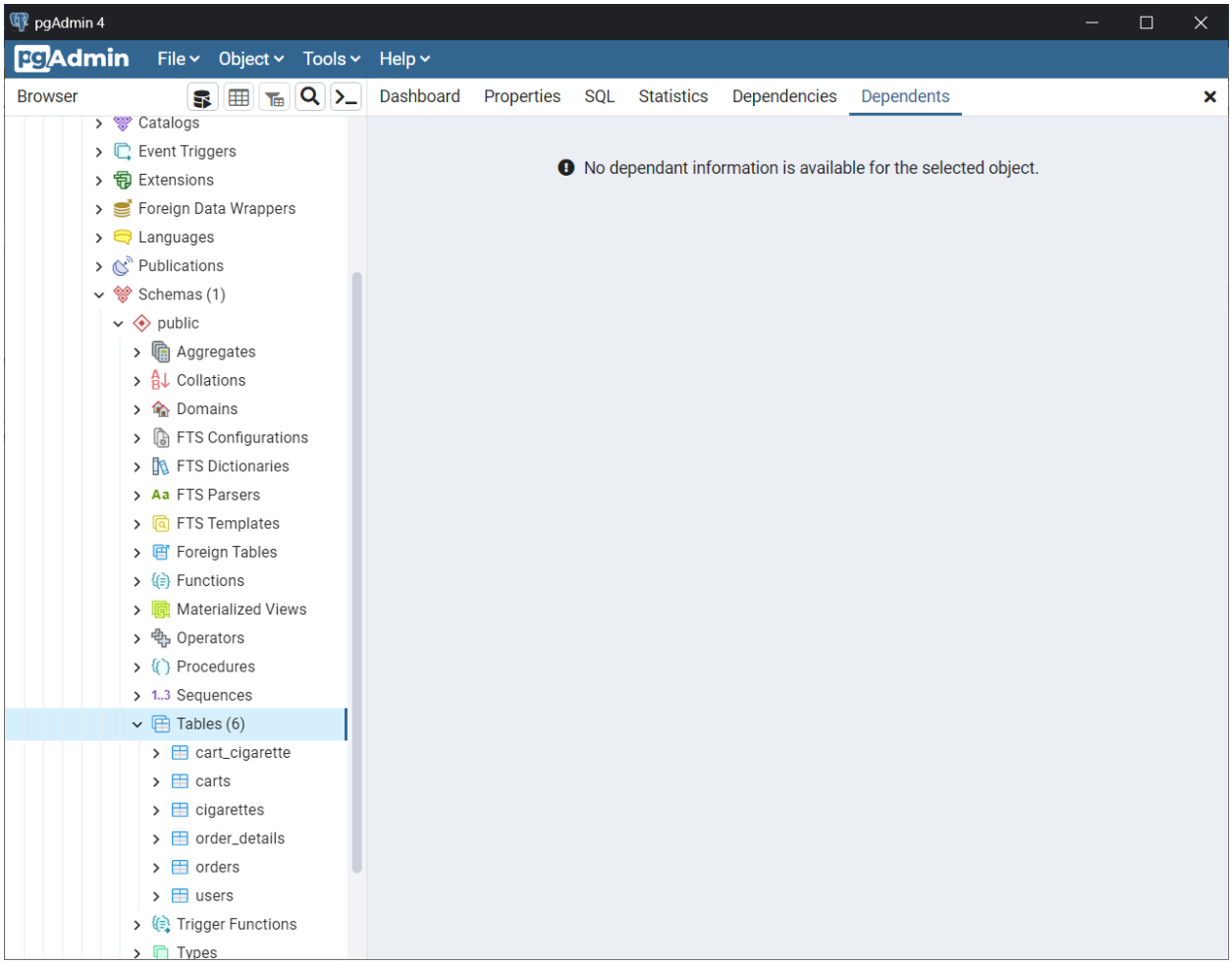

Рис. 2.19. Інтерфейс PostgreSQL

2.7.3. Виклик та завантаження програми

Для роботи з розробленим веб-додатком локально потрібно запустити клас SmokeShopApp.java та перейти у браузері за адресою localhost:8080, на разі є тільки такий спосіб, далі це можливо реалізувати на віртуальному сервері, для цього буде потрібно зібрати проект завдяки команді mvn clean install та задеплоїти його на сервер, а також додати скрипт, що буде постійно тримати зібраний проект у запущеному стані. Авжеж, у такому випадку потрібно буде використовувати ip-адресу замість localhost, а також новий порт.

2.7.4. Опис інтерфейсу користувача

Рис. 2.20. Стартова сторінка

Зверху ліворуч стандартне меню, праворуч зверху - посилання на вхід та реєстрацію на сайті. Меню скорочене, бо у не авторизованого користувача не може бути кошику для замовлення.

Title	Price	Manufacturer	Puffs	
ElfBar Watermelon	300.0	ElfBar	1500	Log in
ElfBar Pink Lemonade	300.0	ElfBar	1500	Log in
ElfBar Siberian Forest	300.0	ElfBar	1500	Log in
ElfBar Energy Grape	300.0	ElfBar	1500	Log in
ElfBar Energy Strawberry	300.0	ElfBar	1500	Log in
ElfBar Blue Raz Lemonad	300.0	ElfBar	1500	Log in
ElfBar Coconut Melon	300.0	ElfBar	1500	Log in
ElfBar Speer Mint	300.0	ElfBar	1500	Log in
ElfBar Strawberry Banana	300.0	ElfBar	1500	Log in
ElfBar Banana Milk	300.0	ElfBar	1500	Log in
ElfBar Apple Peach	300.0	ElfBar	1500	Log in
ElfBar Blueberry	300.0	ElfBar	1500	Log in

Рис. 2.21. Сторінка товарів

На даній сторінці ми бачимо список товарів, що є у продажі, на разі у таблиці ми бачимо посилання на вхід, бо користувач на авторизований, він не може додавати товар у кошик.

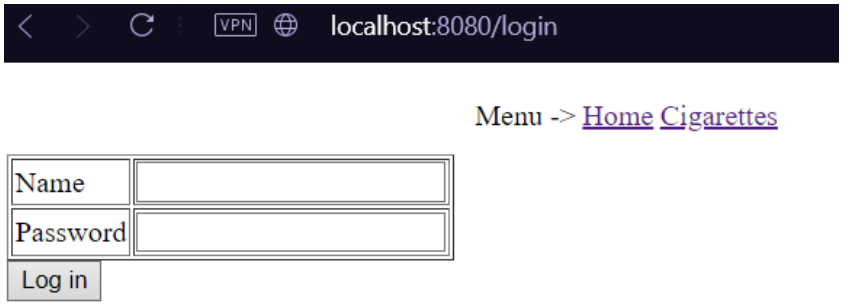

Рис. 2.22. Сторінка входу

Тут ми бачимо форму для входу, користувач повинен ввести дані (ім’я та пароль) для входу у свій аккаунт, запит відправляється на BE та завдяки Spring Security дає дозвіл на вхід, якщо дані співпадати.

Рис. 2.23. Авторизувались під аккаунтом адміністратора

Увійшли у аккаунт адміністратора, потрапили на головну сторінку і вже бачимо зміни: з’явились нові опції меню, праворуч зверху - ім’я користувача з посиланням на його профіль та посилання для виходу з аккаунту.

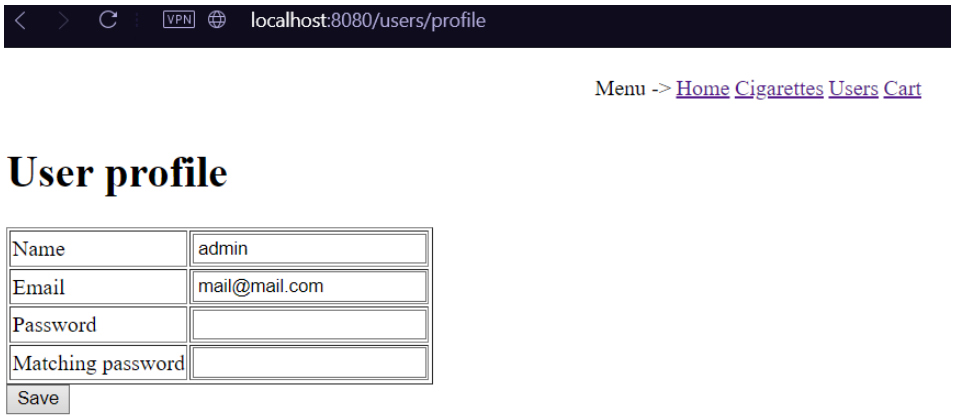

Рис. 2.24. Сторінка профілю

На даній сторінці ми бачимо профіль адміністратору, тут можна змінити пароль для входу, якщо вони не будуть співпадати, то буде оновлена сторінка без зміни паролю, у противному випадку – пароль буде змінений.

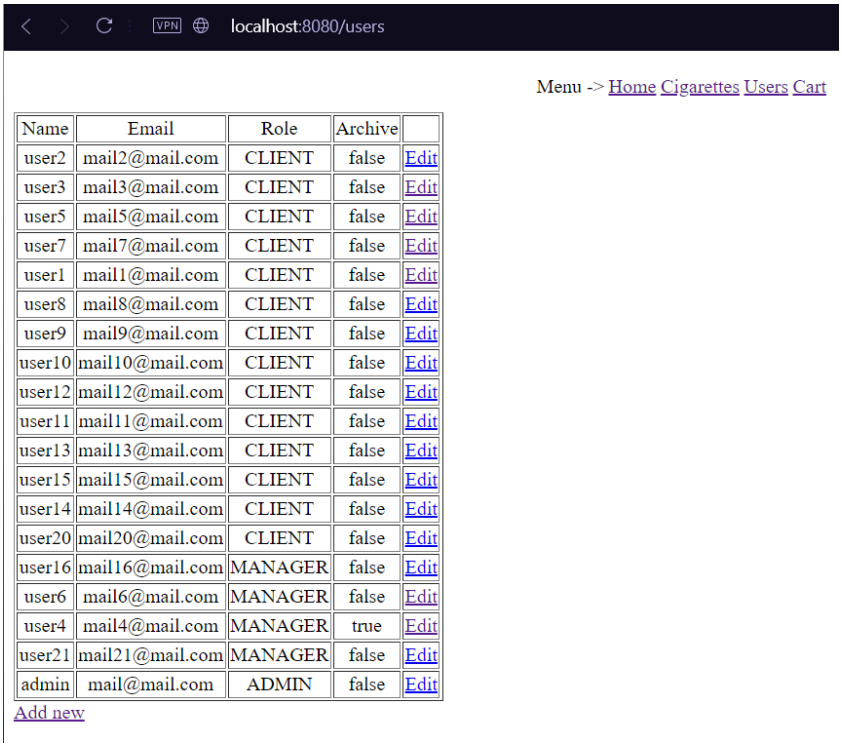

Рис. 2.25. Сторінка списку користувачів

Це сторінка зі списком усіх користувачів, зареєстрованих на сайті. До цієї сторінки мають доступи лише аккаунти менеджерів та адміністратору. Тут є посилання для створення нового користувача та окреме посилання редакції тих, що є у списку.

Рис. 2.26. Сторінка створення нового користувача

Тут уся влада у менеджера, на сам перед це для створення нових аккаунтів менеджерів.

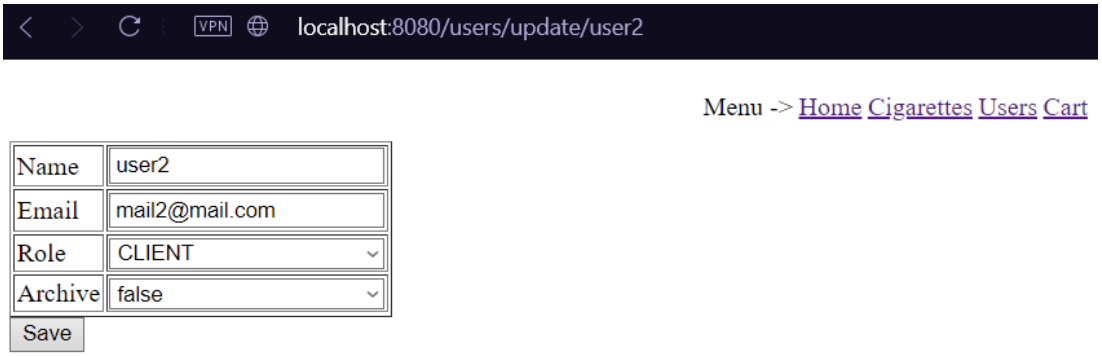

Рис. 2.27. Сторінка зміни користувача

На цій сторінці менеджер може змінити Роль користувача й статус його аккаунту. При зміні даних повертається попередня сторінка з уже зміненими даними.

Рис. 2.28. Сторінка кошику

Так виглядає сторінка кошику, якщо вона порожня.

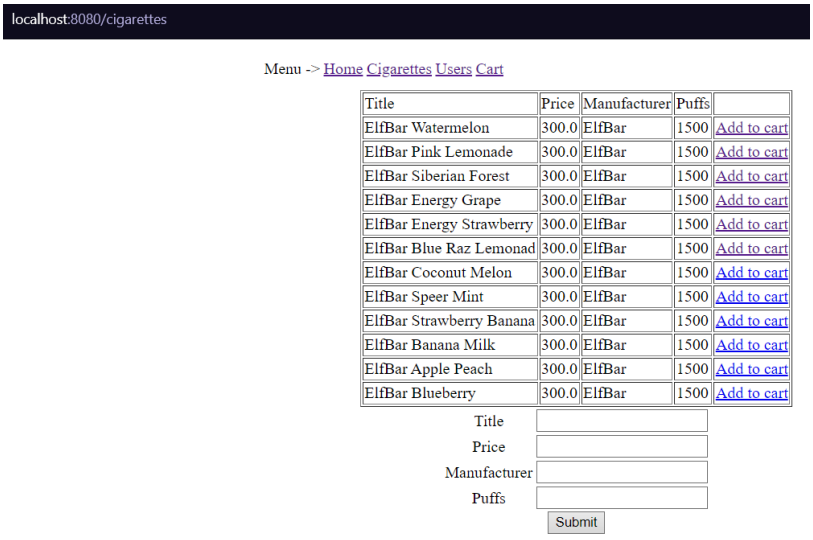

Рис. 2.29. Сторінка товарів (авторизований менеджер)

Як ми бачимо, якщо авторизований менеджер заходить на сайт, то під таблицею зі списком товарів з’являється форма для додавання нового товару.

При відправленні даних сторінка динамічно змінюється і до списку додається новий товар. А також, коли авторизований користувач переходить на цю сторінку, то в нього з’являється можливість додати товар до кошику.

Рис. 2.30. Сторінка кошику після додавання товару

На цій сторінці ми бачимо додані товари, суму замовлення у такому вигляді, строку для вводу адреси відправлення, а також є можливість зменшити кількість товару або зовсім його прибрати.

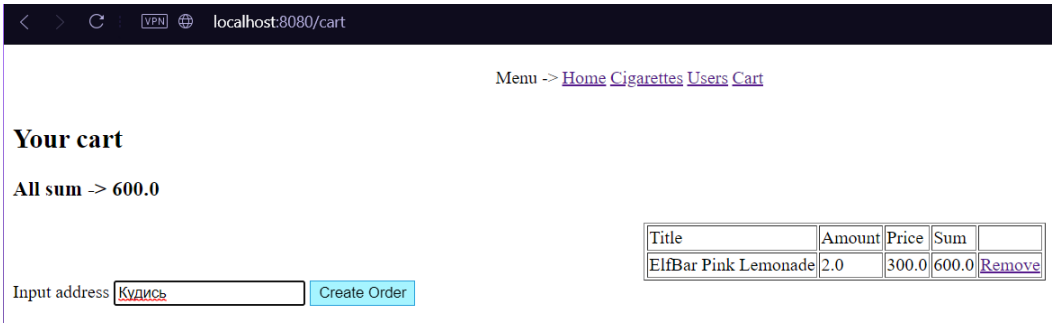

Рис. 2.31. Прибрали першу позицію

Із кошика прибрали першу позицію, сума перерахована, кошик оновлений. Після натискання кнопки біля адреси, зберігається у БД дані замовлення, а кошик стає порожнім.

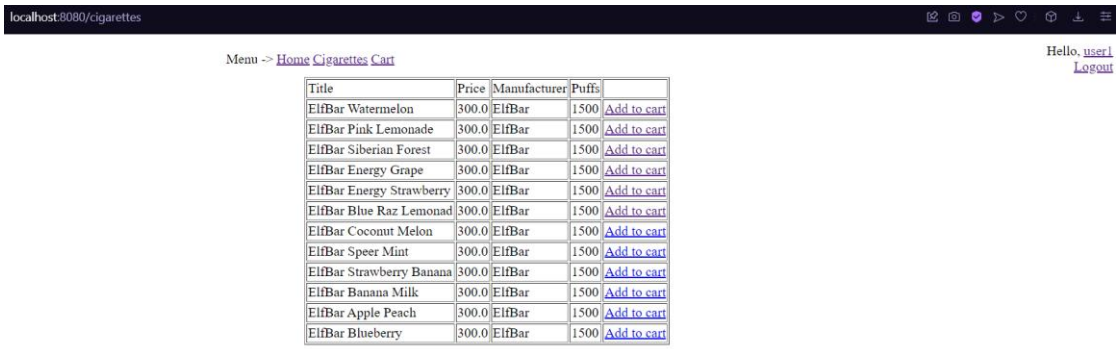

Рис. 2.32. Сторінка товарів, увійшов у аккаунт звичайного користувача

Як ми бачимо, авторизувавшись у аккаунт користувача, нема полей для створення нового товару, а також у меню нема заборонених для користувача endpoint-ів.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 2057;

2. коефіцієнт складності програми – 1.4;

3. коефіцієнт корекції програми в ході її розробки – 0.07;

4. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1.4;

5. годинна заробітна плата програміста – 173 грн/год;

Згідно зі статистикою сайту «Української спільноти програмістів (DOU)», дізнаємося середню зарплату програміста у годину. Середня українська заробітна плата програміста, що пише програми на Java та з досвідом роботи близько року дорівнює 950 американських доларів в місяць, таким чином, очікувана заробітна плата має бути в цих рамках. При поточному курсі валют НБУ станом на початок червня 2022 року один американський долар дорівнює 29.25 грн, тому середня зарплата в гривнях дорівнює 27 787.5 грн (950 доларів США).

6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0.8;

7. вартість машино-години ЕОМ – 20.20 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_d, \text{ людино-годин,} \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається як 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

t_d – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \quad (3.2)$$

де q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт кореляції програми в ході її розробки.

Підрахунок за формулою (3.2):

$$Q = 2057 \cdot 1{,}4 \cdot (1 + 0{,}07) = 3081.39;$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot K}, \text{ людино-годин}, \quad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

Витрати праці на розробку алгоритму рішення задачі за формулою(3.3):

$$t_u = \frac{3081.39 \cdot 1{,}4}{80 \cdot 0.8} = 67.41, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25) \cdot K}; \quad (3.4)$$

де Q - умовне число операторів програми;

k - коефіцієнт кваліфікації програміста.

Підставивши відповідні значення змінних у формулу(3.4), отримаємо:

$$t_a = \frac{3081.39}{20 \cdot 0.8} = 192.59, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25) \cdot K}; \quad (3.5)$$

Відповідно до формули(3.5) отримаємо:

$$t_n = \frac{3081.39}{25 \cdot 0.8} = 154.07, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5) \cdot K}; \quad (3.6)$$

Відповідно до формули(3.6) отримаємо:

$$t_{отл} = \frac{3081.39}{5 \cdot 0,8} = 770.35, \text{ людино-годин,}$$

– за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1,5 \cdot t_{отл}; \quad (3.7)$$

Відповідно до формули(3.7) отримаємо:

$$t_{отл}^{к} = 1,5 \cdot 770.35 = 1155.53, \text{ людино-годин}$$

Витрати праці на підготовку документації:

$$t_d = t_{dp} + t_{do}, \quad (3.8)$$

де t_{dp} – трудомісткість підготовки матеріалів і рукопису;

t_{do}– трудомісткість редагування, печатки й оформлення документації.

$$t_d = \frac{Q}{(15...20) \cdot K}; \quad (3.9)$$

Відповідно до формули(3.9) отримаємо:

$$t_d = \frac{3081.39}{20 \cdot 0{,}8} = 192.59, \text{ людино-годин.}$$

$$t_{do} = 0{,}75 \cdot t_{dp} \quad (3.10)$$

$$t_{do} = 0{,}75 \cdot 192.59 = 144.44, \text{ людино-годин.}$$

$$t_d = 192.59 + 144.44 = 337.03, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

$t = 50 + 67.41 + 192.59 + 154.07 + 770.35 + 337.03 = 1571.45,$ людино-годин.

У результаті ми розрахували, що в загалом необхідно 1571.45 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{ПО}=З_{ЗП}+З_{МВ}, \text{ грн,} \quad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП}=t \cdot С_{ПР}, \text{ грн,} \quad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

Відповідно до формули(3.12) отримаємо:

$$З_{ЗП} = 1571.45 \cdot 173 = 271860.85, \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ}=t_{отл} \cdot С_{МЧ}, \text{ грн,} \quad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$С_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

Відповідно до формули(3.13) отримаємо:

$$З_{МВ} = 770.35 \cdot 20.20 = 15561.07 \text{ грн.}$$

Витрати на створення ПЗ знайдемо за формулою (3.11):

$$К_{ПО} = 271860.85 + 15561.07 = 287421.92 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T=\frac{t}{B_k \cdot F_p}, \text{ мес.} \quad (3.14)$$

де B_k- число виконавців;

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні Fp=160 годин).

$$T = \frac{1571.45}{1 \cdot 160} = 9{,}8 \text{ міс.}$$

Висновки. На розробку даного програмного забезпечення піде 1571.45 людино-годин. Тобто, ймовірна очікувана тривалість розробки складатиме 9.8 місяці при стандартному 40-годинному робочому тижні і 160-годинному робочому місяці. Очікувані витрати на створення програмного забезпечення складатимуть 287421.92 грн.

ВИСНОВКИ

Метою кваліфікаційної роботи було створення ВЕ-частини інтернет-магазину електронних сигарет. Робота є актуальною за технологіями, фреймворками та бібліотеками.

Призначення розробленого застосунку полягає у полегшенні клієнтам можливості замовлення їх улюблених електронних сигарет, без необхідності виходити й шукати цей товар у фізичних магазинах.

Для розробки додатку використовувались сучасні підходи для створення програм з можливістю подальшого впровадження до інших систем. За необхідності застосунок може бути розширено новим функціоналом, якщо такий буде потрібен для покращення роботи додатку.

У реальному житті цей додаток може потенційно допомогти людям замовити товар, який знаходиться далеко від їхньої дислокації і важко, або неможливо знайти десь поряд.

В результаті розробки створено додаток, який можна буде вдосконалювати новим функціоналом та використовувати як основу для подальших розробок нових інтернет-магазинів, а в наш час їх створюється дуже багато, бо це дуже зручно як для клієнтів, так і для бізнесу.

Виконуючи цю кваліфікаційну роботу були досягнуті наступні етапи:

– проаналізована предметна область задачі;

– встановленні вимоги розробки та підстави для розробки інтернет-магазину;

– визначена технології та структура для створення додатку;

– розробка коду ПЗ.

Також під час виконання кваліфікаційної роботи було визначено трудомісткість розробленого програмного продукту (1571.45 людино-годин), підраховано вартості роботи по створенню програми (287421.92 грн) а також підраховано час на його створення (9.8 міс).

Список використаних джерел

1. Кішорі Шаран. Особливості мови програмування Java. 2018 р. 236 с.
2. Мартін Калін. Java Web Services: Up and Running, 2009. 320 с.
3. Крейг Уолс. Спрінг у Дії, Шосте Видання, 2021. 520 с.
4. Юліана Косміна. Pivotal Certified Professional Core Spring 5 Developer Exam: A Study Guide Using, 2020, 609 с.
5. Flyway документація: https://flywaydb.org/documentation/
6. Веб-сайт «Українська спільнота програмістів»: https://jobs.dou.ua/salaries/#period=dec2020&city=all&title=Junior%20Software%20Engineer&language=Java&spec=&exp1=0&exp2=2?period=2021-12&position=Junior%20SE&technology=Java&experience=0-1
7. Веб-сайт НБУ: https://minfin.com.ua/currency/nbu/#:~:text=Курс%20валют%20НБУ&text=Курс%20доллара%20США%20(USD)%20не,30%2C5962%20грн%20за%20евро.
8. Maven документація: https://maven.apache.org/guides/index.html
9. Thymeleaf документація: https://www.thymeleaf.org/documentation.html
10. Lombok приклад: https://www.baeldung.com/intro-to-project-lombok
11. JQuery документація: https://api.jquery.com
12. Stomp документація: https://www.cloudamqp.com/docs/stomp.html
13. SockJs документація: https://www.rubydoc.info/gems/sockjs
14. Spring MVC tutorial: https://www.baeldung.com/spring-mvc-tutorial
15. Spring MVC Forms:

https://www.baeldung.com/spring-mvc-form-tutorial

16. Spring MVC Thymeleaf:

https://www.baeldung.com/thymeleaf-in-spring-mvc

17. Basic entity annotations:

https://zetcode.com/springboot/annotations/#:~:text=The%20%40Entity%20annotation%20specifies%20that,to%20be%20used%20for%20mapping.

18. DTO pattern:

https://www.baeldung.com/java-dto-pattern

19. Spring Security in Java:

https://spring.io/guides/gs/securing-web/

20. WebSockets with Spring:

https://www.baeldung.com/websockets-spring

ДОДАТОК А

КОД ПРОГРАМИ

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.4.5</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.batalskiy</groupId>
   <artifactId>spring-eshop</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>spring-eshop</name>
   <description>Smoke Shop</description>
   <properties>
      <java.version>1.8</java.version>
   </properties>
   <dependencies>
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
   </dependency>
   <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <version>1.4.200</version>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web-services</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
   </dependency>
   <dependency>
```

```
        <groupId>org.flywaydb</groupId>
        <artifactId>flyway-core</artifactId>
    </dependency>
    <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity5</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-websocket</artifactId>
    </dependency>
    <dependency>
        <groupId>org.webjars</groupId>
        <artifactId>sockjs-client</artifactId>
        <version>RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.webjars</groupId>
        <artifactId>stomp-websocket</artifactId>
        <version>RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.webjars</groupId>
        <artifactId>jquery</artifactId>
        <version>RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
      </plugins>
   </build>
</project>
```

SmokeShopApp.java

```
@SpringBootApplication
public class SmokeShopApp {

	public static void main(String[] args) {
		SpringApplication.run(SmokeShopApp.class, args);
	}

}
```

SecurityConfig.java

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true,
jsr250Enabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

	private UserService userService;

	@Autowired
	public void setUserService(UserService userService) {
		this.userService = userService;
	}

	@Override
	protected void configure(AuthenticationManagerBuilder auth) throws Exception {
		auth.authenticationProvider(authenticationProvider());
	}

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

	@Bean
	public DaoAuthenticationProvider authenticationProvider() {
		DaoAuthenticationProvider auth = new DaoAuthenticationProvider();
		auth.setUserDetailsService(userService);
		auth.setPasswordEncoder(passwordEncoder());
		return auth;
	}

	@Override
	protected void configure(HttpSecurity http) throws Exception {
```

```
                http.authorizeRequests()
                                .antMatchers("/users").hasAnyAuthority(Role.ADMIN.name(),
Role.MANAGER.name())
                                .anyRequest().permitAll()
                                .and()
                                .formLogin()
                                .loginPage("/login")
                                .failureUrl("/login-error")
                                .loginProcessingUrl("/auth")
                                .permitAll()
                                .and()
                                .logout().logoutRequestMatcher(new
AntPathRequestMatcher("/logout"))
                                .logoutSuccessUrl("/").deleteCookies("JSESSIONID")
                                .invalidateHttpSession(true)
                                .and()
                                .csrf().disable();
        }
}
```

WebSocketConfig.java

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

        @Override
        public void registerStompEndpoints(StompEndpointRegistry registry) {
                registry.addEndpoint("/socket").withSockJS();
        }

        @Override
        public void configureMessageBroker(MessageBrokerRegistry registry) {
                registry.enableSimpleBroker("/topic");
                registry.setApplicationDestinationPrefixes("/app");
        }
}
```

CartController.java

```
@Controller
@RequestMapping("/cart")
public class CartController {

        private final CartService cartService;
        private final CigaretteService cigaretteService;

        public CartController(CartService cartService, CigaretteService cigaretteService) {
                this.cartService = cartService;
                this.cigaretteService = cigaretteService;
        }

        @GetMapping
        public String aboutCart(Model model, Principal principal){
```

```
            if(principal == null){
                model.addAttribute("cart", new CartDTO());
            }
            else {
                CartDTO cartDto = cartService.getCartByUser(principal.getName());
                model.addAttribute("cart", cartDto);
            }

            return "cart";
        }

        @PostMapping
        public String commitCart(Principal principal, @RequestParam String address){
            if(principal != null){
                cartService.commitCartToOrder(principal.getName(), address);
            }
            return "redirect:/cart";
        }

        @GetMapping("/{id}")
        public String removeFromCart(@PathVariable Long id, Model model, Principal
principal){
            CigaretteDTO cigaretteToRemove = cigaretteService.getById(id);
            model.addAttribute("cart",
cartService.removeCartDetail(cigaretteToRemove, principal.getName()));
            return "cart";
        }
}
```

CigaretteController.java

```
@Controller
@RequestMapping("/cigarettes")
public class CigaretteController {

        private final CigaretteService cigaretteService;

        public CigaretteController(CigaretteService cigaretteService) {
            this.cigaretteService = cigaretteService;
        }

        @GetMapping
        public String list(Model model){
            List<CigaretteDTO> list = cigaretteService.getAll();
            model.addAttribute("cigarettes", list);
            return "cigarettes";
        }

        @GetMapping("/{id}/cart")
        public String addToCart(@PathVariable Long id, Principal principal){
            if(principal == null){
                return "redirect:/cigarettes";
            }
```

```
            cigaretteService.addToUserCart(id, principal.getName());
            return "redirect:/cigarettes";
        }

        @PostMapping
        public ResponseEntity<Void> addCigarette(CigaretteDTO dto){
            cigaretteService.addCigarette(dto);
            return ResponseEntity.status(HttpStatus.CREATED).build();
        }

        @MessageMapping("/cigarettes")
        public void messageAddCigarette(CigaretteDTO dto){
            cigaretteService.addCigarette(dto);
        }

        @GetMapping("/{id}")
        @ResponseBody
        public CigaretteDTO getById(@PathVariable Long id){
            return cigaretteService.getById(id);
        }

}
```

CigaretteRestController.java

```
@RestController
@RequestMapping("/api/v1/cigarettes")
public class CigaretteRestController {

   private final CigaretteService cigaretteService;


   public CigaretteRestController(CigaretteService cigaretteService) {
      this.cigaretteService = cigaretteService;
   }

   @GetMapping("/{id}")
   public CigaretteDTO getById(@PathVariable Long id){
      return cigaretteService.getById(id);
   }

   @PostMapping
   public void addCigarette(@RequestBody CigaretteDTO dto){
      cigaretteService.addCigarette(dto);
   }

}
```

ErrorControllerAdvice.java

```
@ControllerAdvice
public class ErrorControllerAdvice {

        @ExceptionHandler(Exception.class)
```

```
	@ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
	public String exception(Exception exception, Model model){
		String errorMessage = (exception != null ? exception.getMessage() :
"Unknown error");
		model.addAttribute("errorMessage", errorMessage);
		return "error";
	}
}
```

MainController.java

```
@Controller
public class MainController {

	@GetMapping({"","/"})
	public String index(){
		return "index";
	}

	@GetMapping("/login")
	public String login(){
		return "login";
	}

	@GetMapping("/login-error")
	public String loginError(Model model){
		model.addAttribute("loginError", true);
		return "login";
	}

	@GetMapping("/register")
	public String register(Model model, UserDTO userDTO){
		model.addAttribute("user", userDTO);
		return "register";
	}
}
```

UserController.java

```
@Controller
@RequestMapping("/users")
public class UserController {

	private final UserService userService;

	public UserController(UserService userService) {
		this.userService = userService;
	}

	@GetMapping
	public String userList(Model model){
		model.addAttribute("users", userService.getAll());
		return "userList";
	}
```

```
@PreAuthorize("hasAnyAuthority('ADMIN','MANAGER')")
@GetMapping("/new")
public String newUser(Model model){
	model.addAttribute("user", new UserDTO());
	return "user";
}

@PreAuthorize("hasAnyAuthority('ADMIN','MANAGER')")
@PostMapping("/new")
public String saveUser(UserDTO dto, Model model){
	if(userService.save(dto)){
		return "redirect:/users";
	}
	else {
		model.addAttribute("user", dto);
		return "user";
	}
}

@PreAuthorize("hasAnyAuthority('ADMIN','MANAGER')")
@GetMapping("/update/{username}")
public String updateUserPage(Model model, @PathVariable String username){
	User user = userService.findByName(username);
	UserDTO dto = UserDTO.builder()
			.username(user.getName())
			.email(user.getEmail())
			.role(user.getRole())
			.archive(user.isArchive())
			.build();
	model.addAttribute("user", dto);
	return "update";
}

@PreAuthorize("hasAnyAuthority('ADMIN','MANAGER')")
@PostMapping("/update")
public String updateUser(Model model, UserDTO userDTO){
	User user = userService.findByName(userDTO.getUsername());
	boolean changed = false;
	if (!Objects.equals(user.getRole(), userDTO.getRole())){
		changed = true;
	}
	if(!user.isArchive() == userDTO.isArchive()) {
		changed = true;
	}
	if (changed) {
		userService.updateUser(userDTO);
		return "redirect:/users";
	}
	model.addAttribute("user", userDTO);
	return "update";
}
```

```
	@PostMapping("/register")
	public String registerUser(UserDTO dto, Model model){
		if(userService.save(dto)){
			return "redirect:/login";
		}
		else {
			model.addAttribute("user", dto);
			return "user";
		}
	}

	@PreAuthorize("isAuthenticated()")
	@GetMapping("/profile")
	public String profileUser(Model model, Principal principal){
		if(principal == null){
			throw new RuntimeException("You are not authorized");
		}
		User user = userService.findByName(principal.getName());
		UserDTO dto = UserDTO.builder()
				.username(user.getName())
				.email(user.getEmail())
				.build();
		model.addAttribute("user", dto);
		return "profile";
	}

	@PreAuthorize("isAuthenticated()")
	@PostMapping("/profile")
	public String updateProfileUser(UserDTO dto, Model model, Principal principal){
		if(principal == null
				|| !Objects.equals(principal.getName(), dto.getUsername())){
			throw new RuntimeException("You are not authorized");
		}
		if(dto.getPassword() != null
				&& !dto.getPassword().isEmpty()
				&&!Objects.equals(dto.getPassword(),
dto.getMatchingPassword())){
			model.addAttribute("user", dto);
			return "profile";
		}
		userService.updateProfile(dto);
		return "redirect:profile";
	}

}
```

CartRepository.java

```
public interface CartRepository extends JpaRepository<Cart, Long> {
}
```

CigaretteRepository.java

```
public interface CigaretteRepository extends JpaRepository<Cigarette, Long> {
```

```
}
```

OrderRepository.java

```
public interface OrderRepository extends JpaRepository<Order, Long> {
}
```

UserRepository.java

```
public interface UserRepository extends JpaRepository<User, Long> {
	User findFirstByName(String name);
}
```

Cart.java

```
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "carts")
public class Cart {
	private static final String SEQ_NAME = "cart_seq";

	@Id
	@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQ_NAME)
	@SequenceGenerator(name = SEQ_NAME, sequenceName = SEQ_NAME, allocationSize = 1)
	private Long id;
	@OneToOne
	@JoinColumn(name = "user_id")
	private User user;
	@ManyToMany
	@JoinTable(name = "cart_cigarettes",
			joinColumns = @JoinColumn(name = "cart_id"),
			inverseJoinColumns = @JoinColumn(name = "cigarette_id"))
	private List<Cigarette> cigarettes;

	public Long getId() {
		return id;
	}

	public User getUser() {
		return user;
	}

	public void setUser(User user) {
		this.user = user;
	}

	public List<Cigarette> getCigarettes() {
		return cigarettes;
	}

	public void setCigarettes(List<Cigarette> cigarettes) {
		this.cigarettes = cigarettes;
	}
```

```
	@Override
	public boolean equals(Object o) {
		if (this == o) return true;
		if (!(o instanceof Cart)) return false;
		Cart cart = (Cart) o;
		return Objects.equals(id, cart.id) && Objects.equals(user, cart.user)
				&& Objects.equals(cigarettes, cart.cigarettes);
	}

	@Override
	public int hashCode() {
		return Objects.hash(id, user, cigarettes);
	}
}
```

Cigarette.java

```
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@Table(name = "cigarettes")
public class Cigarette {
	private static final String SEQ_NAME = "cigarette_seq";
	@Id
	@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQ_NAME)
	@SequenceGenerator(name = SEQ_NAME, sequenceName = SEQ_NAME, allocationSize = 1)
	private Long id;
	private String title;
	private String manufacturer;
	private int puffs;
	private Double price;

	public Long getId() {
		return id;
	}

	public String getTitle() {
		return title;
	}

	public void setTitle(String title) {
		this.title = title;
	}

	public String getManufacturer() {
		return manufacturer;
	}

	public void setManufacturer(String manufacturer) {
```

```
        this.manufacturer = manufacturer;
    }

    public int getPuffs() {
        return puffs;
    }

    public void setPuffs(int puffs) {
        this.puffs = puffs;
    }

    public Double getPrice() {
        return price;
    }

    public void setPrice(Double price) {
        this.price = price;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Cigarette)) return false;
        Cigarette cigarette = (Cigarette) o;
        return puffs == cigarette.puffs && Objects.equals(id, cigarette.id)
                && Objects.equals(title, cigarette.title)
                && Objects.equals(manufacturer, cigarette.manufacturer)
                && Objects.equals(price, cigarette.price);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, title, manufacturer, puffs, price);
    }
}
```

Order.java

```
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "orders")
public class Order {
   private static final String SEQ_NAME = "order_seq";

   @Id
   @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQ_NAME)
   @SequenceGenerator(name = SEQ_NAME, sequenceName = SEQ_NAME,
allocationSize = 1)
   private Long id;
   @CreationTimestamp
   private LocalDateTime created;
```

```
@UpdateTimestamp
private LocalDateTime changed;
@ManyToOne
@JoinColumn(name = "user_id")
private User user;
private BigDecimal sum;
private String address;
@OneToMany(mappedBy = "order", cascade = CascadeType.ALL)
private List<OrderDetails> details;
@Enumerated(EnumType.STRING)
private OrderStatus status;

public Long getId() {
   return id;
}

public LocalDateTime getCreated() {
   return created;
}

public void setCreated(LocalDateTime created) {
   this.created = created;
}

public LocalDateTime getChanged() {
   return changed;
}

public void setChanged(LocalDateTime changed) {
   this.changed = changed;
}

public User getUser() {
   return user;
}

public void setUser(User user) {
   this.user = user;
}

public BigDecimal getSum() {
   return sum;
}

public void setSum(BigDecimal sum) {
   this.sum = sum;
}

public String getAddress() {
   return address;
}
```

```
    public void setAddress(String address) {
        this.address = address;
    }

    public List<OrderDetails> getDetails() {
        return details;
    }

    public void setDetails(List<OrderDetails> details) {
        this.details = details;
    }

    public OrderStatus getStatus() {
        return status;
    }

    public void setStatus(OrderStatus status) {
        this.status = status;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Order)) return false;
        Order order = (Order) o;
        return Objects.equals(id, order.id) && Objects.equals(created, order.created)
                && Objects.equals(changed, order.changed) &&
Objects.equals(user, order.user)
                && Objects.equals(sum, order.sum) &&
Objects.equals(address, order.address)
                && Objects.equals(details, order.details) && status ==
order.status;
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, created, changed, user, sum, address, details, status);
    }
}
```

OrderDetails.java

```
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "orders_details")
public class OrderDetails {
        private static final String SEQ_NAME = "order_details_seq";

        @Id
        @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
SEQ_NAME)
```

```
	@SequenceGenerator(name = SEQ_NAME, sequenceName = SEQ_NAME,
allocationSize = 1)
	private Long id;
	@ManyToOne
	@JoinColumn(name = "order_id")
	private Order order;
	@ManyToOne
	@JoinColumn(name = "cigarette_id")
	private Cigarette cigarette;
	private BigDecimal amount;
	private BigDecimal price;

	public OrderDetails(Order order, Cigarette cigarette, Long amount) {
		this.order = order;
		this.cigarette = cigarette;
		this.amount = new BigDecimal(amount);
		this.price = BigDecimal.valueOf(cigarette.getPrice());
	}

	public Long getId() {
		return id;
	}

	public Order getOrder() {
		return order;
	}

	public void setOrder(Order order) {
		this.order = order;
	}

	public Cigarette getCigarette() {
		return cigarette;
	}

	public void setCigarette(Cigarette cigarette) {
		this.cigarette = cigarette;
	}

	public BigDecimal getAmount() {
		return amount;
	}

	public void setAmount(BigDecimal amount) {
		this.amount = amount;
	}

	public BigDecimal getPrice() {
		return price;
	}

	public void setPrice(BigDecimal price) {
```

```
        this.price = price;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof OrderDetails)) return false;
        OrderDetails that = (OrderDetails) o;
        return Objects.equals(id, that.id) && Objects.equals(order, that.order) &&
Objects.equals(cigarette, that.cigarette) && Objects.equals(amount, that.amount) &&
Objects.equals(price, that.price);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, order, cigarette, amount, price);
    }
}
```

OrderStatus.java

```
public enum OrderStatus {
    NEW,APPROVED,CANCELED,PAID,CLOSED
}
```

Role.java

```
public enum Role {
    CLIENT,MANAGER,ADMIN
}
```

User.java

```
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "users")
public class User {

    private static final String SEQ_NAME = "user_seq";

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
SEQ_NAME)
    @SequenceGenerator(name = SEQ_NAME, sequenceName = SEQ_NAME,
allocationSize = 1)
    private Long id;
    private String name;
    private String password;
    private String email;
    private boolean archive;
    @Enumerated(EnumType.STRING)
    private Role role;
    @OneToOne(mappedBy = "user", cascade = CascadeType.REMOVE)
```

```
private Cart cart;

public Long getId() {
	return id;
}

public String getName() {
	return name;
}

public void setName(String name) {
	this.name = name;
}

public String getPassword() {
	return password;
}

public void setPassword(String password) {
	this.password = password;
}

public String getEmail() {
	return email;
}

public void setEmail(String email) {
	this.email = email;
}

public boolean isArchive() {
	return archive;
}

public void setArchive(boolean archive) {
	this.archive = archive;
}

public Role getRole() {
	return role;
}

public void setRole(Role role) {
	this.role = role;
}

public Cart getCart() {
	return cart;
}

public void setCart(Cart cart) {
	this.cart = cart;
```

```
        }

        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (!(o instanceof User)) return false;
            User user = (User) o;
            return archive == user.archive && Objects.equals(id, user.id) &&
    Objects.equals(name, user.name) && Objects.equals(password, user.password) &&
    Objects.equals(email, user.email) && role == user.role && Objects.equals(cart, user.cart);
        }

        @Override
        public int hashCode() {
            return Objects.hash(id, name, password, email, archive, role, cart);
        }
    }
```

CartDetailDTO.java

```
@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class CartDetailDTO {

    private String title;

    private Long cigaretteId;

    private Double price;

    private String manufacturer;

    private int puffs;

    private Double amount;

    private Double sum;


    public CartDetailDTO(Cigarette cigarette){

        this.title = cigarette.getTitle();

        this.cigaretteId = cigarette.getId();

        this.price = cigarette.getPrice();

        this.manufacturer = cigarette.getManufacturer();

        this.puffs = cigarette.getPuffs();

        this.amount = 1.0;
```

```
            this.sum = cigarette.getPrice();
        }
}
```

CartDTO.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class CartDTO {
        private long amountCigarettes;
        private double sum;
        private List<CartDetailDTO> cartDetails = new ArrayList<>();

        public void aggregate(){
                this.amountCigarettes = cartDetails.size();
                this.sum = cartDetails.stream()
                                .map(CartDetailDTO::getSum)
                                .mapToDouble(Double::doubleValue)
                                .sum();
        }
}
```

CigaretteDTO.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class CigaretteDTO {
   private Long id;
   private String title;
        private String manufacturer;
        private int puffs;
   private Double price;

}
```

UserDTO.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class UserDTO {
        private String username;
        private String password;
        private String matchingPassword;
        private String email;
```

```
        private Role role;
        private boolean archive;
}
```

CartService.java

```
public interface CartService {
        Cart createCart(User user, List<Long> cigaretteIds);
        void addCigarettes(Cart cart, List<Long> cigaretteIds);
        CartDTO getCartByUser(String name);
        CartDTO removeCartDetail(CigaretteDTO cigaretteDTO, String name);
        void commitCartToOrder(String username, String address);
}
```

CartServiceImpl.java

```
@Service
public class CartServiceImpl implements CartService {

   private final CartRepository cartRepository;
   private final CigaretteRepository cigaretteRepository;
   private final UserService userService;
   private final OrderService orderService;

   public CartServiceImpl(CartRepository cartRepository,
                          CigaretteRepository cigaretteRepository,
                          UserService userService,
                          OrderService orderService) {
      this.cartRepository = cartRepository;
      this.cigaretteRepository = cigaretteRepository;
      this.userService = userService;
      this.orderService = orderService;
   }

   @Override
   @Transactional
   public Cart createCart(User user, List<Long> cigaretteIds) {
      Cart cart = new Cart();
      cart.setUser(user);
      List<Cigarette> cigaretteList = getCollectRefCigarettesByIds(cigaretteIds);
      cart.setCigarettes(cigaretteList);
      return cartRepository.save(cart);
   }

   private List<Cigarette> getCollectRefCigarettesByIds(List<Long> cigaretteIds) {
      return cigaretteIds.stream()
            .map(cigaretteRepository::getOne)
            .collect(Collectors.toList());
   }

   @Override
```

```
    @Transactional
    public void addCigarettes(Cart cart, List<Long> cigaretteIds) {
        List<Cigarette> cigarettes = cart.getCigarettes();
        List<Cigarette> newCigarettesList = cigarettes == null ? new ArrayList<>() : new
ArrayList<>(cigarettes);
        newCigarettesList.addAll(getCollectRefCigarettesByIds(cigaretteIds));
        cart.setCigarettes(newCigarettesList);
        cartRepository.save(cart);
    }

    @Override
    public CartDTO getCartByUser(String name) {
        User user = userService.findByName(name);
        if (user == null || user.getCart() == null) {
            return new CartDTO();
        }
        return getCartDTO(user);
    }

    private CartDTO getCartDTO(User user) {
        CartDTO cartDTO = new CartDTO();
        Map<Long, CartDetailDTO> mapByCigaretteId = new HashMap<>();
        List<Cigarette> cigarettes = user.getCart().getCigarettes();

        for (Cigarette cigarette : cigarettes) {
            CartDetailDTO detail = mapByCigaretteId.get(cigarette.getId());
            if (detail == null) {
                mapByCigaretteId.put(cigarette.getId(), new CartDetailDTO(cigarette));
            } else {
                detail.setAmount(detail.getAmount() + 1.0);
                detail.setSum(detail.getSum() + cigarette.getPrice());
            }
        }

        cartDTO.setCartDetails(new ArrayList<>(mapByCigaretteId.values()));
        cartDTO.aggregate();
        return cartDTO;
    }

    @Override
    public CartDTO removeCartDetail(CigaretteDTO cigaretteDTO, String name) {
        User user = userService.findByName(name);
        CartDTO dto = getCartDTO(user);
        List<CartDetailDTO> cartDetails = dto.getCartDetails();
        AtomicBoolean toRemove = new AtomicBoolean(false);

        AtomicReference<CartDetailDTO> cartDetailToRemove = new AtomicReference<>(new
CartDetailDTO());
```

```
    cartDetails.stream().filter(cartDetailDTO ->
                Objects.equals(cartDetailDTO.getCigaretteId(), cigaretteDTO.getId()))
            .forEach(cartDetailDTO -> {
                if (cartDetailDTO.getAmount() > 1) {
                    cartDetailDTO.setAmount(cartDetailDTO.getAmount() - 1);
                    cartDetailDTO.setSum(cartDetailDTO.getSum() - cigaretteDTO.getPrice());
                } else {
                    cartDetailToRemove.set(cartDetailDTO);
                    toRemove.set(true);
                }
            });

    if (toRemove.get()) cartDetails.remove(cartDetailToRemove.get());

    dto.setCartDetails(cartDetails);
    List<Cigarette> editedCigarettes = new ArrayList<>();

    for (int i = 0; i < cartDetails.size(); i++) {
        for (int j = 0; j < cartDetails.get(i).getAmount(); j++) {
            editedCigarettes.add(Cigarette.builder()
                    .id(cartDetails.get(i).getCigaretteId())
                    .title(cartDetails.get(i).getTitle())
                    .manufacturer(cartDetails.get(i).getManufacturer())
                    .puffs(cartDetails.get(i).getPuffs())
                    .price(cartDetails.get(i).getPrice())
                    .build());
        }
    }

    Cart cart = user.getCart();
    cart.setCigarettes(editedCigarettes);
    cartRepository.save(cart);
    return dto;
}

@Override
@Transactional
public void commitCartToOrder(String username, String address) {
    User user = userService.findByName(username);
    if (user == null) {
        throw new RuntimeException("User is not found");
    }
    Cart cart = user.getCart();
    if (cart == null || cart.getCigarettes().isEmpty()) {
        return;
    }

    Order order = new Order();
```

```
        order.setStatus(OrderStatus.NEW);
        order.setUser(user);

        Map<Cigarette, Long> cigaretteWithAmount = cart.getCigarettes().stream()
                .collect(Collectors.groupingBy(cigarette -> cigarette, Collectors.counting()));

        List<OrderDetails> orderDetails = cigaretteWithAmount.entrySet().stream()
                .map(pair -> new OrderDetails(order, pair.getKey(), pair.getValue()))
                .collect(Collectors.toList());

        BigDecimal total = BigDecimal.valueOf(orderDetails.stream()
                .map(detail -> detail.getPrice().multiply(detail.getAmount()))
                .mapToDouble(BigDecimal::doubleValue).sum());

        order.setDetails(orderDetails);
        order.setSum(total);
        order.setAddress(address);

        orderService.saveOrder(order);
        cart.getCigarettes().clear();
        cartRepository.save(cart);
    }
}
```

CigaretteService.java

```
public interface CigaretteService {
        List<CigaretteDTO> getAll();
        void addToUserCart(Long productId, String username);
        void addCigarette(CigaretteDTO dto);
        CigaretteDTO getById(Long id);
}
```

CigaretteServiceImpl.java

```
@Service
public class CigaretteServiceImpl implements CigaretteService {

        private final CigaretteRepository cigaretteRepository;
        private final UserService userService;
        private final CartService cartService;
        private final SimpMessagingTemplate template;

        public CigaretteServiceImpl(CigaretteRepository cigaretteRepository,
                                    UserService userService,
                                    CartService cartService,
                                    SimpMessagingTemplate template) {
                this.cigaretteRepository = cigaretteRepository;
                this.userService = userService;
                this.cartService = cartService;
                this.template = template;
```

```
	}

	@Override
	public List<CigaretteDTO> getAll() {
		return cigaretteRepository.findAll().stream()
				.map(this::toDTO).collect(Collectors.toList());
	}

	@Override
	@Transactional
	public void addToUserCart(Long cigaretteId, String username) {
		User user = userService.findByName(username);
		if(user == null){
			throw new RuntimeException("User not found. " + username);
		}
		Cart cart = user.getCart();
		if(cart == null){
			Cart newCart = cartService.createCart(user,
Collections.singletonList(cigaretteId));
			user.setCart(newCart);
			userService.save(user);
		}
		else {
			cartService.addCigarettes(cart, Collections.singletonList(cigaretteId));
		}
	}

	@Override
	@Transactional
	public void addCigarette(CigaretteDTO dto) {
		Cigarette cigarette = fromDTO(dto);
		Cigarette savedCigarette = cigaretteRepository.save(cigarette);

		template.convertAndSend("/topic/cigarettes",
				toDTO(savedCigarette));
	}

	@Override
	public CigaretteDTO getById(Long id) {
		Cigarette cigarette = cigaretteRepository.findById(id).orElse(new Cigarette());
		return toDTO(cigarette);
	}

	private CigaretteDTO toDTO(Cigarette cigarette) {
		return CigaretteDTO.builder()
				.id(cigarette.getId())
				.title(cigarette.getTitle())
				.manufacturer(cigarette.getManufacturer())
```

```
                .puffs(cigarette.getPuffs())
                .price(cigarette.getPrice())
                .build();
    }

    private Cigarette fromDTO(CigaretteDTO cigaretteDTO) {
        return Cigarette.builder()
                .id(cigaretteDTO.getId())
                .title(cigaretteDTO.getTitle())
                .manufacturer(cigaretteDTO.getManufacturer())
                .puffs(cigaretteDTO.getPuffs())
                .price(cigaretteDTO.getPrice())
                .build();
    }
}
```

OrderService.java

```
public interface OrderService {
   void saveOrder(Order order);
   Order getOrder(Long id);
}
```

OrderServiceImpl.java

```
@Service
public class OrderServiceImpl implements OrderService {
   private final OrderRepository orderRepository;

   public OrderServiceImpl(OrderRepository orderRepository) {
      this.orderRepository = orderRepository;
   }

   @Override
   @Transactional
   public void saveOrder(Order order) {
      orderRepository.save(order);
   }

   @Override
   public Order getOrder(Long id) {
      return orderRepository.findById(id).orElse(null);
   }

}
```

UserService.java

```
public interface UserService extends UserDetailsService {
        boolean save(UserDTO userDTO);
        void save(User user);
        List<UserDTO> getAll();
```

```
    User findByName(String name);
    void updateProfile(UserDTO userDTO);
    void updateUser(UserDTO userDTO);
}
```

UserServiceImpl.java

```
@Service
public class UserServiceImpl implements UserService {

   private final UserRepository userRepository;
   private final PasswordEncoder passwordEncoder;

   public UserServiceImpl(UserRepository userRepository, PasswordEncoder passwordEncoder) {
      this.userRepository = userRepository;
      this.passwordEncoder = passwordEncoder;
   }

   @Override
   public List<UserDTO> getAll() {
      return userRepository.findAll().stream()
            .map(this::toDto)
            .collect(Collectors.toList());
   }

   @Override
   @Transactional
   public boolean save(UserDTO userDto) {
      if (!Objects.equals(userDto.getPassword(), userDto.getMatchingPassword())) {
         throw new RuntimeException("Password is not equal");
      } else if (!uniqueMail(userDto.getEmail())) {
         throw new RuntimeException("This mail is already in use");
      } else if (!uniqueName(userDto.getUsername())){
         throw new RuntimeException("This username is already in use");
      }
      else {
         User user = User.builder()
               .name(userDto.getUsername())
               .password(passwordEncoder.encode(userDto.getPassword()))
               .email(userDto.getEmail())
               .role(userDto.getRole())
               .build();
         userRepository.save(user);
      }
      return true;
   }

   private boolean uniqueMail(String mail) {
      List<String> mails = getAll().stream().map(UserDTO::getEmail).collect(Collectors.toList());
```

```
        return !mails.contains(mail);
    }

    private boolean uniqueName(String username) {
        List<String> usernames =
getAll().stream().map(UserDTO::getUsername).collect(Collectors.toList());
        return !usernames.contains(username);
    }

    @Override
    public User findByName(String name) {
        return userRepository.findFirstByName(name);
    }

    @Override
    @Transactional
    public void updateProfile(UserDTO dto) {
        User user = userRepository.findFirstByName(dto.getUsername());
        if (user == null) {
            throw new RuntimeException("User not found by name " + dto.getUsername());
        }

        boolean changed = false;
        if (dto.getPassword() != null && !dto.getPassword().isEmpty()) {
            user.setPassword(passwordEncoder.encode(dto.getPassword()));
            changed = true;
        }
        if (!Objects.equals(dto.getEmail(), user.getEmail())) {
            user.setEmail(dto.getEmail());
            changed = true;
        }
        if (changed) {
            userRepository.save(user);
        }
    }

    @Override
    @Transactional
    public void updateUser(UserDTO userDTO) {
        User user = userRepository.findFirstByName(userDTO.getUsername());
        if (user == null) {
            throw new UsernameNotFoundException("User not found with name: " +
userDTO.getUsername());
        }
        boolean changed = false;
        if (!Objects.equals(userDTO.isArchive(), user.isArchive())){
            user.setArchive(userDTO.isArchive());
            changed = true;
```

```
        }
        if (!Objects.equals(userDTO.getRole(), user.getRole())) {
            user.setRole(userDTO.getRole());
            changed = true;
        }
        if (changed){
            userRepository.save(user);
        }
    }

    @Override
    public void save(User user) {
        userRepository.save(user);
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
{
        User user = userRepository.findFirstByName(username);
        if (user == null) {
            throw new UsernameNotFoundException("User not found with name: " + username);
        }

        List<GrantedAuthority> roles = new ArrayList<>();
        roles.add(new SimpleGrantedAuthority(user.getRole().name()));

        return new org.springframework.security.core.userdetails.User(
                user.getName(),
                user.getPassword(),
                roles);
    }

    private UserDTO toDto(User user) {
        return UserDTO.builder()
                .username(user.getName())
                .email(user.getEmail())
                .role(user.getRole())
                .archive(user.isArchive())
                .build();
    }
}
```

application.yaml

```
spring:
  jpa:
    hibernate:
      ddl-auto: validate
    show-sql: true
```

```
  datasource:
    url: jdbc:postgresql://localhost:5432/shop
    username: postgres
    password: thisisnotpostgres
  flyway:
    baseline-on-migrate: true
```

V1__init_DB.sql

```
create sequence user_seq start 1 increment 1;

create table users (
                    id int8 not null,
                    archive boolean not null,
                    email varchar(255),
                    name varchar(255),
                    password varchar(255),
                    role varchar(255),
                    primary key (id)
);
create sequence cart_seq start 1 increment 1;

create table carts (
                     id int8 not null,
                     user_id int8,
                     primary key (id)
);

alter table if exists carts
    add constraint carts_fk_user
        foreign key (user_id) references users;

create sequence cigarette_seq start 1 increment 1;

create table cigarettes (
                      id int8 not null,
                      price float8,
                      title varchar(255),
                      manufacturer varchar(255),
                      puffs int,
                      primary key (id)
);

create table cart_cigarettes (
                           cart_id int8 not null,
                           cigarette_id int8 not null
);

alter table if exists cart_cigarettes
```

```
    add constraint cart_cigarettes_fk_cigarette
        foreign key (cigarette_id) references cigarettes;

alter table if exists cart_cigarettes
    add constraint cart_cigarettes_fk_cart
        foreign key (cart_id) references carts;

create sequence order_seq start 1 increment 1;

create table orders (
                        id int8 not null,
                        address varchar(255),
                        changed timestamp,
                        created timestamp,
                        status varchar(255),
                        sum numeric(19, 2),
                        user_id int8,
                        primary key (id)
);

alter table if exists orders
    add constraint orders_fk_user
        foreign key (user_id) references users;

create sequence order_details_seq start 1 increment 1;

create table orders_details (
                            id int8 not null,
                            order_id int8 not null,
                            cigarette_id int8 not null,
                            amount numeric(19, 2),
                            price numeric(19, 2),
                            primary key (id)
);

alter table if exists orders_details
    add constraint orders_details_fk_order
        foreign key (order_id) references orders;

alter table if exists orders_details
    add constraint orders_details_fk_cigarette
        foreign key (cigarette_id) references cigarettes;
```

V2__add_admin.sql

```
INSERT INTO users (id, archive, email, name, password, role)
VALUES (1, false, 'mail@mail.com', 'admin',
'$2a$10$QRWAbXveneW1d.EKZPnxV.D7hjqPfw9Ex4nJuJpLgiFCDFopflogC', 'ADMIN');
ALTER SEQUENCE user_seq RESTART WITH 2;
```

V3__add_products.sql

```
INSERT INTO cigarettes (id, price, title, manufacturer, puffs)
values (1, 300.0, 'ElfBar Watermelon', 'ElfBar', 1500);

INSERT INTO cigarettes (id, price, title, manufacturer, puffs)
values (2, 300.0, 'ElfBar Pink Lemonade', 'ElfBar', 1500);

INSERT INTO cigarettes (id, price, title, manufacturer, puffs)
values (3, 300.0, 'ElfBar Siberian Forest', 'ElfBar', 1500);

INSERT INTO cigarettes (id, price, title, manufacturer, puffs)
values (4, 300.0, 'ElfBar Energy Grape', 'ElfBar', 1500);

INSERT INTO cigarettes (id, price, title, manufacturer, puffs)
values (5, 300.0, 'ElfBar Energy Strawberry', 'ElfBar', 1500);
ALTER SEQUENCE cigarette_seq RESTART WITH 6;
```

cigarette.js

```
var stomp = null;
// подключаемся к серверу по окончании загрузки страницы
window.onload = function() {
    connect();
};
function connect() {
    var socket = new SockJS('/socket');
    stomp = Stomp.over(socket);
    stomp.connect({}, function (frame) {
        console.log('Connected: ' + frame);
        stomp.subscribe('/topic/cigarettes', function (cigarette) {
            renderItem(cigarette);
        });
    });
}
// хук на интерфейс
$(function () {
    $("form").on('submit', function (e) {
        e.preventDefault();
    });
    $( "#send" ).click(function() { sendContent(); });
});
// отправка сообщения на сервер
function sendContent() {
    stomp.send("/app/cigarettes", {}, JSON.stringify({
        'title': $("#title").val(),
        'price': $("#price").val(),
        'manufacturer': $("#manufacturer").val(),
        'puffs': $("#puffs").val()
    }));
```

```
}
// рендер сообщения, полученного от сервера
function renderItem(cigaretteJson) {
   var cigarette = JSON.parse(cigaretteJson.body);
   $("#table").append("<tr>" +
      "<td>" +cigarette.title +"</td>" +
      "<td>" +cigarette.price +"</td>" +
      "<td>" +cigarette.manufacturer +"</td>" +
      "<td>" +cigarette.puffs +"</td>" +
      "<td><a href='/cigarettes/" + cigarette.id +"/cart'>Add to cart</a></td>" +
      "</tr>");
}
```

menu.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml"
xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
   <meta charset="UTF-8">
   <title>Menu</title>
</head>
<body>
<table id="nav-menu" width="100%">
   <tr>
      <td align="center">
         <table>
            <tr>
               <td>Menu -></td>
               <td><a th:href="@{/}">Home</a></td>
               <td><a th:href="@{/cigarettes}">Cigarettes</a></td>
               <td sec:authorize="hasAnyAuthority('ADMIN', 'MANAGER')"><a
th:href="@{/users}">Users</a></td>
               <td sec:authorize="isAuthenticated()"><a th:href="@{/cart}">Cart</a></td>
            </tr>
         </table>
      </td>
      <td align="right">
         <div sec:authorize="isAuthenticated()">
            <div>
               Hello, <a th:href="@{/users/profile}"><span
sec:authentication="principal.username"></span></a>
            </div>
            <div>
               <a th:href="@{/logout}">Logout</a>
            </div>
         </div>
         <div sec:authorize="!isAuthenticated()">
            <div>
```

```
            Hello, Unknown. Please <a th:href="@{/login}">Log in</a>
          </div>
          <div>
            No account yet? <a th:href="@{/register}">Sing up</a>
          </div>
        </div>
      </td>
    </tr>
</table>
</body>
</html>
```

cart.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>Cart</title>
</head>
<body>
<div th:insert="~{fragments/menu :: #nav-menu}"></div>
<h2>Your cart</h2>
<h3>All sum -> <b th:text="${cart.sum}"></b></h3>
<h2 th:if="${#lists.isEmpty(cart.cartDetails)}" style="color: red;">Add something to your cart
first</h2>
<table border="1" align="center" th:unless="${#lists.isEmpty(cart.cartDetails)}">
    <tr>
        <td>Title</td>
        <td>Amount</td>
        <td>Price</td>
        <td>Sum</td>
        <td></td>
    </tr>
    <tr th:each="detail : ${cart.cartDetails}">
        <td th:text="${detail.title}"></td>
        <td th:text="${detail.amount}"></td>
        <td th:text="${detail.price}"></td>
        <td th:text="${detail.sum}"></td>
        <td><a th:href="@{/cart/{id}(id=${detail.cigaretteId})}">Remove</a></td>
    </tr>
</table>
<form th:action="@{/cart(address=${address})}" method="post"
th:unless="${#lists.isEmpty(cart.cartDetails)}">
    <label>Input address</label>
    <input type="text" th:value="${address}" name="address">
    <button type="submit">Create Order</button>
</form>
</body>
```

```
</html>
```

Cigarettes.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml"
xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
   <meta charset="UTF-8">
   <title>Cigarettes</title>
   <script src="/webjars/sockjs-client/sockjs.min.js"></script>
   <script src="/webjars/stomp-websocket/stomp.min.js"></script>
   <script src="/webjars/jquery/jquery.min.js"></script>
   <script src="/cigarette.js"></script>
</head>
<body>
<div th:insert="~{fragments/menu :: #nav-menu}"></div>

<table border="1" align="center" id="table">
   <tr font>
      <td>Title</td>
      <td>Price</td>
      <td>Manufacturer</td>
      <td>Puffs</td>
      <td></td>
   </tr>
   <tr th:each="cigarette : ${cigarettes}">
      <td th:text="${cigarette.title}"></td>
      <td th:text="${cigarette.price}"></td>
      <td th:text="${cigarette.manufacturer}"></td>
      <td th:text="${cigarette.puffs}"></td>
      <td sec:authorize="isAuthenticated()"><a
th:href="@{/cigarettes/{id}/cart(id=${cigarette.id})}">Add to cart</a></td>
      <td sec:authorize="!isAuthenticated()"><a th:href="@{/login}">Log in</a></td>
   </tr>
</table>
<form style="text-align: center" sec:authorize="hasAnyAuthority('ADMIN', 'MANAGER')">
   <table align="center">
      <tr>
         <td>Title</td>
         <td><input type="text" id="title"></td>
      </tr>
      <tr>
         <td>Price</td>
         <td><input type="number" id="price"></td>
      </tr>
      <tr>
         <td>Manufacturer</td>
         <td><input type="text" id="manufacturer"></td>
```

```
        </tr>
        <tr>
            <td>Puffs</td>
            <td><input type="text" id="puffs"></td>
        </tr>
    </table>
    <button id="send" type="submit">Submit</button>
</form>
</body>
</html>
```

error.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml"
xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="UTF-8">
  <title>Error 500</title>
</head>
<body>
<div th:insert="~{fragments/menu :: #nav-menu}"></div>
<p th:utext="${errorMessage}">Error java.lang.NullPointerException</p>
</body>
</html>
```

index.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml"
xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>Smoke Shop</title>
</head>
<body>
<div th:insert="~{fragments/menu :: #nav-menu}"></div>
</body>
</html>
```

login.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml"
xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>Login page</title>
</head>
<body>
<div th:insert="~{fragments/menu :: #nav-menu}"></div>
<p th:if="${loginError}" class="error">Wrong username or password</p>
```

```
<form th:action="@{/auth}" method="post">
   <table border="1">
      <tr>
         <td>Name</td>
         <td><input type="text" name="username"></td>
      </tr>
      <tr>
         <td>Password</td>
         <td><input type="password" name="password"></td>
      </tr>
   </table>
   <button type="submit">Log in</button>
</form>
</body>
</html>
```

profile.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml"
xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
   <meta charset="UTF-8">
   <title>Profile</title>
</head>
<body>
<div th:insert="~{fragments/menu :: #nav-menu}"></div>
<h1>User profile</h1>
<form th:action="@{/users/profile}" th:object="${user}" method="post">
   <table border="1">
      <tr>
         <td>Name</td>
         <td><input type="text" th:field="*{username}"/></td>
      </tr>
      <tr>
         <td>Email</td>
         <td><input type="email" th:field="*{email}"/></td>
      </tr>
      <tr>
         <td>Password</td>
         <td><input type="password" th:field="*{password}"/></td>
      </tr>
      <tr>
         <td>Matching password</td>
         <td><input type="password" th:field="*{matchingPassword}"/></td>
      </tr>
   </table>
   <button type="submit">Save</button>
</form>
```

```
</body>
</html>
```

register.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml"
xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
   <meta charset="UTF-8">
   <title>Register</title>
</head>
<body>
<div th:insert="~{fragments/menu :: #nav-menu}"></div>
<form th:action="@{/users/register}" th:object="${user}" method="post">
   <table border="1">
      <tr>
         <td>Name</td>
         <td><input type="text" th:field="*{username}"></td>
      </tr>
      <tr>
         <td>Password</td>
         <td><input type="password" th:field="*{password}"></td>
      </tr>
      <tr>
         <td>Matching password</td>
         <td><input type="password" th:field="*{matchingPassword}"></td>
      </tr>
      <tr>
         <td>Email</td>
         <td><input type="email" th:field="*{email}"></td>
      </tr>
   </table>
   <button type="submit">Save</button>
</form>
</body>
</html>
```

update.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml"
xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
   <meta charset="UTF-8">
   <title>Update User</title>
</head>
<body>
<div th:insert="~{fragments/menu :: #nav-menu}"></div>
<form th:action="@{/users/update}" th:object="${user}" method="post">
   <table border="1">
```

```
        <tr>
            <td>Name</td>
            <td><input type="text" th:field="*{username}" readonly/></td>
        </tr>
        <tr>
            <td>Email</td>
            <td><input type="email" th:field="*{email}" readonly/></td>
        </tr>
        <tr>
            <td>Role</td>
            <td>
                <select th:field="*{role}" style='width:100%;'>
                    <option th:value="'CLIENT'" th:text="CLIENT"></option>
                    <option th:value="'MANAGER'" th:text="MANAGER"></option>
                </select>
            </td>
        </tr>
        <tr>
            <td>Archive</td>
            <td>
                <select th:field="*{archive}" style='width:100%;'>
                    <option th:value="'false'" th:text="false"></option>
                    <option th:value="'true'" th:text="true"></option>
                </select>
            </td>
        </tr>
    </table>
    <button type="submit">Save</button>
</form>
</body>
</html>
```

user.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml"
xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>Add user</title>
</head>
<body>
<div th:insert="~{fragments/menu :: #nav-menu}"></div>
<form th:action="@{/users/new}" th:object="${user}" method="post">
    <table border="1">
        <tr>
            <td>Name</td>
            <td><input type="text" th:field="*{username}"></td>
        </tr>
```

```
        <tr>
            <td>Password</td>
            <td><input type="password" th:field="*{password}"></td>
        </tr>
        <tr>
            <td>Matching password</td>
            <td><input type="password" th:field="*{matchingPassword}"></td>
        </tr>
        <tr>
            <td>Email</td>
            <td><input type="email" th:field="*{email}"></td>
        </tr>
        <tr>
            <td>Role</td>
            <td>
                <select th:field="*{role}" style='width:100%;'>
                    <option th:value="'CLIENT'" th:text="CLIENT"></option>
                    <option th:value="'MANAGER'" th:text="MANAGER"></option>
                </select>
            </td>
        </tr>
    </table>
    <button type="submit">Save</button>
</form>
</body>
</html>
```

userList.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml"
xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>User list</title>
</head>
<body>
<div th:insert="~{fragments/menu :: #nav-menu}"></div>
<table border="1" style="text-align: center">
    <tr>
        <td>Name</td>
        <td>Email</td>
        <td>Role</td>
        <td>Archive</td>
        <td></td>
    </tr>
    <tr th:each="user : ${users}">
        <td th:text="${user.username}"></td>
        <td th:text="${user.email}"></td>
```

```
        <td th:text="${user.role}"></td>
        <td th:text="${user.archive}"></td>
        <td><a th:href="@{/users/update/{username}(username=${user.username})}">Edit</a></td>
    </tr>
</table>
<a th:href="@{/users/new}">Add new</a>
</body>
</html>
```

ДОДАТОК Б

Відгук керівника економічного розділу

ДОДАТОК В

Перелік файлів на диску

ПЕРЕЛІК ДОКУМЕНТІВ НА МАГНІТНОМУ НОСІЇ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Батальський.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Батальський.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Батальський.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Батальський.ppt	Презентація кваліфікаційної роботи