Інститут комп'ютерних систем
Кафедра інформаційних систем

Національний університет «Одеська політехніка»
Інститут комп'ютерних систем
Кафедра інформаційних систем

КУРСОВА РОБОТА

з дисципліни «Об'єктно-орієнтоване програмування»
Тема «Програмування »

Студентки 2 курсу AI-222 групи
Спеціальності 122 – «Комп’ютерні науки»
Якуша Дар’я Олександрівна
Керівник доцент, Годовіченко М. А.

Національна шкала _____
Кількість балів: _____
Оцінка: ECTS _____

Члени комісії

_____
(підпис) (прізвище та ініціали)

_____
(підпис) (прізвище та ініціали)

_____
(підпис) (прізвище та ініціали)

м. Одеса – 2024 рік
Національний університет «Одеська політехніка»

АНОТАЦІЯ

Розглянуто створення системи логістики на платформі Java Spring. Основною метою є створення REST API для управління ланцюгами постачання, що дозволяє ефективно контролювати процеси доставки та обробки вантажів і замовлень. Для оптимізації логістичних процесів використовується Spring AOP (Аспектно-орієнтоване програмування), що забезпечує підвищення продуктивності та зниження операційних витрат.

Інтеграція з системою управління базами даних реалізована для трекінгу вантажів і замовлень у режимі реального часу, що гарантує точність і актуальність інформації. Забезпечення безпеки логістичних операцій досягається за допомогою Spring Security, який захищає дані від несанкціонованого доступу та зловживань.

ABSTRACT

The creation of a logistics system on the Java Spring platform is considered. The main goal is to create a REST API for supply chain management, which allows you to effectively control the processes of delivery and processing of goods and orders. To optimize logistics processes, Spring AOP (Aspect-Oriented Programming) is used, which ensures increased productivity and reduced operating costs.

Integration with the database management system is implemented for tracking cargo and orders in real time, which guarantees the accuracy and relevance of information. Ensuring the security of logistics operations is achieved with Spring Security, which protects data from unauthorized access and abuse.

ЗМІСТ

ВСТУП

У сучасному світі ефективне управління логістичними процесами відіграє ключову роль у забезпеченні конкурентоспроможності підприємств. Ланцюги постачання стають все більш складними, що вимагає застосування інноваційних підходів та сучасних технологій для їх оптимізації. Одним з таких підходів є використання платформи Java Spring, яка надає потужний інструментарій для розробки масштабованих і гнучких систем.

Java Spring є одним з найбільш популярних фреймворків для розробки корпоративних додатків, що забезпечує високий рівень продуктивності, безпеки та гнучкості. Використання Spring дозволяє спростити конфігурацію та управління залежностями, забезпечуючи при цьому модульність та розширюваність додатків. Завдяки широкому набору бібліотек та інструментів, Spring підтримує створення різноманітних компонентів систем, включаючи REST API, безпеку, управління даними та оптимізацію процесів.

Метою цієї роботи є розробка системи логістики з використанням Java Spring, що включає створення REST API для управління ланцюгами постачання, інтеграцію з системами управління базами даних для трекінгу вантажів і замовлень, а також забезпечення безпеки логістичних операцій за допомогою Spring Security.

REST API забезпечує зручний інтерфейс для взаємодії з системою, дозволяючи виконувати операції додавання, оновлення, видалення та перегляду даних про вантажі, замовлення та інші логістичні процеси. Spring AOP дозволяє впроваджувати крос-концернні аспекти, такі як логування, моніторинг продуктивності та обробка помилок, що допомагає

підвищити ефективність та надійність системи. Spring Data забезпечує спрощене управління доступом до даних та їх обробку, що дозволяє зберігати та аналізувати великі обсяги інформації. Spring Security надає потужний набір інструментів для аутентифікації та авторизації користувачів, захисту даних та запобігання несанкціонованому доступу, що є критично важливим для будь-якої логістичної системи.

Ця система спрямована на підвищення ефективності управління логістичними процесами, забезпечуючи високу надійність, безпеку та оперативність у сфері постачання. Робота детально розглядає підходи до розробки та впровадження системи логістики на базі Java Spring, що включає технічні аспекти розробки, методи інтеграції та забезпечення безпеки, а також приклади використання сучасних технологій для досягнення поставлених цілей. Результатом дослідження стане розроблена система, яка сприятиме підвищенню ефективності управління ланцюгами постачання, зниженню витрат і покращенню загальної продуктивності підприємств у сфері логістики.

ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО JAVA SPRING

Java Spring є одним з найпотужніших і найгнучкіших фреймворків для розробки Java-додатків. Його значення в сучасній розробці важко переоцінити. З моменту свого створення Spring постійно розвивається, пропонуючи розробникам інструменти для спрощення, вдосконалення процесу створення корпоративних програмних рішень.

Основним принципом, на якому будується Spring, є Inversion of Control (IoC). Цей принцип полягає у передачі контролю над управлінням залежностями від об'єкта до зовнішнього контейнера. Це означає, що об'єкти не створюють свої залежності самостійно, а отримують їх ззовні, що дозволяє уникнути жорсткого зв'язування між компонентами системи. Впровадження залежностей (Dependency Injection, DI) є конкретною реалізацією IoC, яка дозволяє автоматично інжектувати необхідні залежності під час створення об'єктів. DI у Spring може бути реалізована за допомогою аннотацій, таких як @Autowired, або за допомогою XML-конфігурацій.

Основні компоненти Spring

Spring AOP (Aspect-Oriented Programming) дозволяє розділяти крос-функціональну логіку, таку як логування, безпека, транзакції, від бізнес-логіки. Це досягається шляхом впровадження аспектів, які можуть бути застосовані до певних точок виконання програми (наприклад, до методів).

Spring Data Access надає підтримку для роботи з базами даних через JDBC, ORM (Hibernate, JPA) та NoSQL (MongoDB, Cassandra). Він забезпечує декларативне управління транзакціями, що значно спрощує роботу з базами даних.

Spring MVC (Model-View-Controller) є модулем для створення веб-додатків на основі архітектури MVC. Він підтримує аннотації для конфігурації контролерів, моделей та представлень, що спрощує розробку веб-інтерфейсів.

Spring Boot — це підмножина фреймворку Spring, яка спрощує створення та розгортання автономних, готових до виробництва Spring-додатків. Завдяки підходу "конвенція над конфігурацією", Spring Boot значно зменшує кількість необхідних налаштувань, що дозволяє швидше розпочати розробку.

Spring Security забезпечує аутентифікацію та авторизацію для додатків Spring. Він підтримує широкий спектр механізмів безпеки, включаючи базову аутентифікацію, OAuth, LDAP та інші.

Spring Cloud — це набір інструментів для розробки розподілених систем та мікросервісів. Він включає підтримку конфігурації, балансування навантаження, виявлення сервісів та багато іншого.

1.1 Поняття Spring Boot

Spring Boot — це розширення фреймворку Spring, яке спрямоване на спрощення процесу створення виробничих, готових до використання, самодостатніх додатків на базі Java. Він усуває багато складнощів та монотонної роботи, пов'язаної з конфігурацією Spring-додатків, що робить його ідеальним вибором для швидкої та ефективної розробки.

Spring Boot значно спрощує процес конфігурації додатків, автоматично налаштовуючи потрібні компоненти на основі залежностей, які є у проєкті. Це дозволяє розробникам зосередитись на логіці додатка, не витрачаючи час на налаштування інфраструктури.

Spring Boot надає вбудовані контейнери для розгортання додатків, такі як Tomcat, Jetty і Undertow. Це дозволяє запускати додатки як звичайні Java-програми без необхідності налаштовувати зовнішні сервлет-контейнери.

Spring Boot пропонує широкий спектр стартерів, які представляють собою набір заздалегідь налаштованих залежностей для певних функціональних можливостей, таких як безпека, робота з базами даних, тестування та інше. Це дозволяє легко додавати потрібні функціональні можливості до додатка.

Spring Initializr — це інструмент для швидкого створення нових проектів на базі Spring Boot. Він дозволяє генерувати проекти з необхідними залежностями та конфігураціями через веб-інтерфейс або за допомогою командного рядка.

Spring Boot Actuator надає набір інструментів для моніторингу та управління додатками, таких як точки доступу для перевірки стану додатка, збору метрик, перегляду інформації про налаштування та інше. Це дозволяє легко інтегрувати додаток з інструментами моніторингу та діагностики.

Реалізацією Spring Boot є створення простого RESTful веб-сервісу за допомогою Spring Boot.

Переваги Використання Spring Boot

1. Швидкість Розробки: завдяки автоматичній конфігурації та готовим стартер-пакетам, Spring Boot значно прискорює процес розробки, дозволяючи швидко створювати готові до використання додатки.

2. Легкість Розгортання: Spring Boot додатки можна запускати як звичайні Java-програми, що спрощує процес розгортання та тестування. Це особливо корисно для мікросервісної архітектури, де кожен сервіс може бути самодостатнім.

3. Гнучкість: незважаючи на автоматизацію багатьох аспектів, Spring Boot дозволяє легко налаштовувати та перевизначати конфігурації відповідно до специфічних потреб проекту.

4. Інтеграція з Іншими Технологіями: Spring Boot легко інтегрується з іншими компонентами екосистеми Spring та сторонніми бібліотеками, що робить його універсальним інструментом для розробки різноманітних додатків.

Отже, Spring Boot є потужним інструментом для прискорення розробки Java-додатків, пропонуючи широкий спектр можливостей для спрощення конфігурації, розгортання та управління додатками. Завдяки своїй гнучкості, інтеграційним можливостям та підтримці спільноти, Spring Boot став невід'ємною частиною сучасного процесу розробки, допомагаючи розробникам створювати ефективні та надійні програмні рішення.

1.2 Spring REST Docs: Документування RESTful API

Spring REST Docs — це інструмент для створення документації RESTful API, який інтегрується з Spring Framework. Його основна мета — забезпечити автоматичне створення точної та детальної документації, яка завжди відповідає поточному стану вашого API. Це досягається за рахунок генерації документації на основі інтеграційних тестів.

Spring REST Docs генерує документацію, використовуючи результати інтеграційних тестів. Це означає, що документація завжди відображає фактичну поведінку API, що робить її особливо корисною для підтримки актуальності та точності документації.

Інструмент підтримує різні формати виводу, включаючи Asciidoctor, Markdown та HTML. Це дозволяє інтегрувати документацію у будь-які існуючі системи управління документацією або веб-сайти.

Оскільки документація генерується на основі тестів, вона автоматично оновлюється при зміні API. Це усуває ризик застарілої документації і гарантує, що розробники завжди мають доступ до актуальної інформації про API.

Переваги Використання Spring REST

1. Підвищення Точності Документації: оскільки документація базується на інтеграційних тестах, вона точно відображає фактичну поведінку API. Це знижує ризик помилок і невідповідностей між документацією та реалізацією.

2. Зменшення Зусиль на Підтримку Документації: автоматизація процесу генерації документації зменшує ручну роботу, необхідну для її підтримки, що дозволяє розробникам зосередитись на розробці функціоналу, а не на оновленні документації.

3. Підвищення Якості API: Інтеграційні тести, що використовуються для генерації документації, також підвищують загальну якість API, оскільки вони забезпечують додатковий рівень перевірки і валідації функціональності.

Spring REST Docs — це потужний інструмент для документування RESTful API, який забезпечує автоматичну актуальність і точність документації. Завдяки інтеграції з Spring та генерації документації на основі інтеграційних тестів, Spring REST Docs допомагає розробникам створювати і підтримувати високоякісну документацію з мінімальними зусиллями.

1.3 Spring AOP (Aspect-Oriented Programming)

Aspect-Oriented Programming (AOP) — це парадигма програмування, яка дозволяє модульно впроваджувати поведінку, яка перетинає основні функціональні можливості програми, тобто крос-функціональні аспекти. У контексті Spring, AOP надає механізм для відокремлення таких крос-функціональних проблем, як логування, управління транзакціями, безпека та інше, від бізнес-логіки.

Основні Концепції AOP

Аспект (Aspect): В основі AOP лежить поняття аспекту — модуля, який інкапсулює крос-функціональну поведінку. Наприклад, такі завдання, як логування, управління транзакціями, безпека та обробка виключень, можуть бути реалізовані як аспекти. Це дозволяє розробникам відокремити ці загальні завдання від бізнес-логіки, що робить код більш зрозумілим і легшим для підтримки.

Точка з'єднання (Join Point): Точка з'єднання — це конкретний момент виконання програми, де аспект може бути впроваджений. У контексті Spring AOP, точками з'єднання зазвичай є виклики методів. Це дозволяє точково впливати на поведінку додатку, додаючи або змінюючи функціонал в певних місцях.

Рада (Advice): Рада — це частина коду, яка виконується на певній точці з'єднання.

Пойнткат (Pointcut): Пойнткат визначає набір точок з'єднання, де аспект може бути впроваджений. Він використовується для визначення умов, за яких буде виконуватися рада. Це дозволяє чітко окреслити, в яких саме частинах програми застосовуватиметься аспект.

Впровадження (Weaving): Впровадження — це процес пов'язування рад з конкретними точками з'єднання. У Spring AOP впровадження зазвичай відбувається під час виконання програми, що забезпечує гнучкість і динамічність.

Переваги Використання Spring AOP

1. Відокремлення Крос-Функціональної Логіки: AOP дозволяє ізолювати крос-функціональні аспекти, такі як логування та управління транзакціями, від бізнес-логіки, що робить код чистішим і легшим для підтримки.

2. Повторне Використання: Аспекти можна використовувати повторно у різних частинах програми, що зменшує дублювання коду.

3. Гнучкість і Масштабованість: AOP забезпечує можливість динамічно додавати нові аспекти, що робить систему більш гнучкою і легко масштабованою.

Отже, Spring AOP є потужним інструментом для розробки Java-додатків, який дозволяє ефективно управляти крос-функціональними аспектами. Використання AOP спрощує структуру програмного забезпечення, роблячи його більш модульним, чистим та підтримуваним. Це особливо важливо в контексті сучасних вимог до якості, продуктивності та гнучкості

програмних систем. Spring AOP, завдяки своїй потужності та простоті використання, продовжує залишатися важливою частиною екосистеми Spring, забезпечуючи розробникам інструменти для створення високоякісних програмних рішень.

1.4 Spring Security: Захист Java-Додатків

Spring Security є одним з ключових модулів фреймворку Spring, який забезпечує надійний механізм для аутентифікації та авторизації Java-додатків. Його потужність та гнучкість роблять його популярним вибором для захисту веб-додатків та корпоративних систем.

Основні Концепції Spring Security

Аутентифікація (Authentication): процес перевірки особи користувача. Користувач надає свої облікові дані (логін і пароль), і система перевіряє їх на відповідність даним, що зберігаються у базі даних або іншому сховищі. У Spring Security аутентифікація може бути реалізована через різні механізми, такі як базова аутентифікація (Basic Authentication), формова аутентифікація (Form-Based Authentication), LDAP, OAuth2 та інші.

Авторизація (Authorization): процес визначення прав доступу користувача до ресурсів додатка. Після успішної аутентифікації система перевіряє, чи має користувач відповідні права на виконання певних дій або доступ до певних ресурсів. Spring Security використовує ролі та дозволи для контролю доступу до ресурсів. Це може бути реалізовано через конфігурації на рівні URL, методів або через аннотації.

Контекст безпеки (Security Context): Spring Security зберігає інформацію про аутентифікованого користувача у SecurityContext. Цей контекст

доступний у будь-якій точці програми і дозволяє отримати інформацію про поточного користувача та його права.

Spring Security пропонує прості способи конфігурування безпеки додатків. Конфігурацію можна здійснити через XML-файли або Java-конфігурацію.

Spring Security дозволяє легко інтегруватися з базою даних для зберігання облікових даних користувачів та ролей. Це можна зробити за допомогою JDBC або JPA.

Переваги Використання Spring Security

1. Гнучкість та Модульність: Розробники можуть легко налаштовувати і розширювати механізми аутентифікації та авторизації, використовуючи аннотації або конфігураційні класи.

2. Широкий спектр можливостей: Підтримка різних механізмів аутентифікації (формова аутентифікація, базова аутентифікація, OAuth2, JWT та інші) дозволяє адаптуватися до різних вимог проекту.

3. Безпека: Використання сучасних алгоритмів шифрування та надійних методів аутентифікації забезпечує високий рівень безпеки додатків.

Отже, Spring Security є незамінним інструментом для забезпечення безпеки Java-додатків. Його потужність, гнучкість та простота використання дозволяють розробникам створювати безпечні та надійні програми, які відповідають сучасним вимогам до безпеки. Завдяки широкому спектру можливостей і легкій інтеграції з іншими компонентами фреймворку Spring, Spring Security продовжує залишатися вибором номер один для багатьох розробників, які прагнуть створити високоякісні та безпечні програмні рішення.

ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАВДАННЯ

Опис програми

Розроблено програму для створення інтегрованої системи логістики, яка дозволяє ефективно управляти ланцюгами постачання, оптимізувати логістичні процеси, відстежувати вантажі та замовлення, а також забезпечувати безпеку операцій. Для досягнення цих цілей буде використовуватися стек технологій Java Spring. Інтеграція REST API, оптимізація процесів за допомогою Spring AOP, надійне зберігання даних та забезпечення безпеки створюють комплексне рішення, яке відповідає сучасним вимогам бізнесу та сприяє підвищенню продуктивності логістичних операцій.

Опис класів з пакету model

1) Опис Класу Client

Клас Client у системі логістики на основі Java Spring є моделлю даних, яка представляє клієнтів. Це JPA-сутність, що дозволяє зберігати та керувати клієнтськими даними в реляційній базі даних. Клас містить кілька полів, які описують основні характеристики клієнта, такі як ідентифікатор, ім'я, прізвище, контактний номер і електронна адреса.

Поле id, яке є первинним ключем, анотоване за допомогою @Id та @GeneratedValue(strategy = GenerationType.IDENTITY), що вказує на автоматичне генерування унікальних значень. Це забезпечує унікальність кожного запису клієнта в базі даних.

Крім основних атрибутів, клас містить поле orders, яке представляє список замовлень, пов'язаних з конкретним клієнтом. Це поле анотоване

@OneToMany(mappedBy = "client"), що означає зв'язок типу "один до багатьох" між клієнтом і замовленнями. Анотація @JsonIgnore використовується для запобігання серіалізації цього поля у форматі JSON, щоб уникнути потенційних рекурсивних залежностей при обміні даними між клієнтом і сервером.

Завдяки анотаціям JPA, клас Client інтегрується з базою даних, що дозволяє зберігати екземпляри цього класу у реляційній базі даних. Відношення з класом Order визначає, що кожен клієнт може мати багато замовлень, що спрощує управління даними про замовлення, пов'язані з конкретним клієнтом.

2) Опис Класу Order

Клас Order у системі логістики на основі Java Spring є сутністю JPA, що представляє замовлення і дозволяє зберігати цю інформацію в реляційній базі даних. Цей клас містить кілька полів, які описують основні характеристики замовлення, такі як ідентифікатор, дата замовлення, клієнт і посилки, пов'язані з цим замовленням.

Клас анотований за допомогою @Entity, що вказує на те, що це є JPA-сутність. Анотація @Table(name = "order") задає ім'я таблиці в базі даних, де будуть зберігатися записи про замовлення. Ім'я таблиці оточене лапками, щоб уникнути конфліктів із зарезервованими словами SQL, такими як order.

Поле id є унікальним ідентифікатором замовлення. Поле orderDate зберігає дату та час замовлення. Анотація @JsonFormat(pattern = "yyyy-MM-dd'T'HH:mm:ss") забезпечує правильне серіалізування дати у форматі JSON, що важливо для передачі даних через API.

Поле client зв'язує замовлення з клієнтом. Воно має анотації @ManyToOne, @JoinColumn(name = "id_client") та @JsonBackReference. Відношення @ManyToOne вказує на те, що кожне замовлення може бути пов'язане з одним клієнтом. Анотація @JoinColumn(name = "id_client") визначає зовнішній ключ у таблиці замовлень, який посилається на таблицю клієнтів. Анотація @JsonBackReference використовується для запобігання рекурсивної серіалізації JSON при обміні даними між клієнтом і сервером.

Поле parcels зв'язує замовлення з посилками. Воно також має анотації @ManyToOne, @JoinColumn(name = "id_parcels") та @JsonBackReference.

3) Опис Класу Parcel

Клас Parcel у системі логістики на основі Java Spring є сутністю JPA, що представляє посилки. Цей клас дозволяє зберігати інформацію про посилки в реляційній базі даних і містить поля, які описують основні характеристики посилки, такі як ідентифікатор, вага, дата відправлення, пункт призначення та опис.

Поле id є унікальним ідентифікатором посилки. Поле weight зберігає вагу посилки. Це поле типу double забезпечує можливість зберігання значень з плаваючою комою, що дозволяє точно вказувати вагу посилки. Поле shipmentDate зберігає дату та час відправлення посилки. Поле destination зберігає пункт призначення посилки. Це поле типу String, що дозволяє зберігати текстові дані про місце, куди відправляється посилка. Поле description зберігає опис посилки. Це поле також типу String, що дозволяє зберігати додаткову інформацію про посилку, яка може бути корисною для її ідентифікації або обробки. Поле orders представляє список замовлень, які включають цю посилку. Поле orders представляє список замовлень, які включають цю посилку. Це відношення типу "один до багатьох" (@OneToMany(mappedBy = "parcels")), що вказує на те, що одна посилка може бути частиною багатьох замовлень.

Опис методів

ClientController

1) @PostMapping("/clients")

```
public ResponseEntity<?> createClients(@RequestBody List<Client> clients) {
        /* тіло методу */
}
```

Цей метод обробляє запити POST на URL /clients, створюючи список клієнтів з тіла запиту і повертаючи їх з HTTP-статусом 201 (Created).

2) @GetMapping ("/clients/surname")

```
   public List<Client> getClientsBySurname(@RequestParam String surname){
      return clientService.getAllClientsBySurname(surname);
   }
```

Цей метод обробляє GET-запити на URL /clients/surname, повертаючи список клієнтів із зазначеним прізвищем, отриманим із параметра запиту.

3) @PutMapping("/{clientId}")

```
public ResponseEntity<String> updateClient(@PathVariable Long clientId,
@RequestBody Client newClientData) {
        /* тіло методу */
}
```

Цей метод обробляє PUT-запити на URL /{clientId}, оновлюючи дані клієнта із зазначеним ідентифікатором на основі даних із тіла запиту, та повертає HTTP-відповідь з повідомленням про успіх чи помилку.

4) @DeleteMapping("/clients/delete/{id}")

```
public ResponseEntity<?> deleteClientById(@PathVariable Long id) {
        /* тіло методу */
}
```

Цей метод обробляє DELETE-запити на URL /clients/delete/{id}, видаляючи клієнта із зазначеним ідентифікатором та повертаючи HTTP-відповідь зі статусом 200 (OK) у разі успішного видалення або 404 (Not Found) у випадку, якщо клієнт не знайдений.

5) @GetMapping("/clients/orders/period")
public ResponseEntity<?> getClientsByOrderPeriod(
    @RequestParam("startDate") String startDateStr,
    @RequestParam("endDate") String endDateStr) {
/* тіло методу */
}
Цей метод обробляє GET-запити на URL /clients/orders/period, повертаючи список клієнтів, у яких були замовлення в зазначений період, і HTTP-відповідь зі статусом 200 (OK) або 404 (Not Found), або 400 (Bad Request) при неправильному форматі дати.

OrderController
1) public ResponseEntity<?> createOrders(@RequestBody List<OrderRequest> orderRequestDTOs) {
    /* тіло методу */
}
Цей метод обробляє запити POST, створюючи замовлення зі списку запитів, переданих у тілі запиту, і повертає створені замовлення з HTTP-статусом 201 (Created).

2) @GetMapping("/details")
public ResponseEntity<?> getAllOrdersWithDetails() {
    /* тіло методу */
}

Цей метод обробляє GET-запити на URL /details, повертаючи всі замовлення з детальною інформацією про клієнтів та посилки у форматі DTO з HTTP-статусом 200 (OK).

3) @DeleteMapping("/delete/{id}")

```
public ResponseEntity<?> deleteOrderById(@PathVariable Long id) {
    /* тіло методу */
}
```

Цей метод обробляє DELETE-запити на URL /delete/{id}, видаляючи замовлення із зазначеним ідентифікатором і повертаючи HTTP-відповідь зі статусом 200 (OK) у разі успішного видалення або 404 (Not Found) у разі, якщо замовлення не знайдено.

4) @GetMapping("/ByClientsId/{clientId}")

```
public ResponseEntity<?> getOrdersByClientId(@PathVariable Long clientId){
  /* тіло методу */
}
```

Цей метод опрацьовує GET-запити на URL /ByClientsId/{clientId}, повертаючи всі замовлення для клієнта із зазначеним ідентифікатором разом із докладною інформацією про клієнта та надсилання у форматі DTO з HTTP-статусом 200 (OK).

5) @GetMapping("/withParcelWeightLessThan5kg")

```
public List<OrderParcelDetails> getOrdersWithParcelWeightLessThan5kg() {
    /* тіло методу */
}
```

Цей метод обробляє GET-запити на URL /withParcelWeightLessThan5kg, повертаючи список замовлень, у яких вага посилки менше 5 кг, з інформацією про замовлення та посилку у форматі DTO.

6) @GetMapping("/ordersByClient/{clientId}")
public ResponseEntity<?> getOrdersByClient(@PathVariable Long clientId) {
    /* тіло методу */
}
Цей метод обробляє GET-запити на URL /ordersByClient/{clientId}, повертаючи всі замовлення для клієнта із зазначеним ідентифікатором разом з детальною інформацією про кожне замовлення та клієнта у форматі DTO, з HTTP-статусом 200 (OK) або 404 (Not Found) , якщо замовлення не знайдено.

ParcelController
1) @PostMapping("/parcel")
public ResponseEntity<?> createParcel(@RequestBody List<Parcel> parcels) {
    /* тіло методу */
}
Цей метод обробляє запити POST на URL /parcel, створюючи посилки на основі переданих даних та повертаючи створені посилки з HTTP-статусом 201 (Created).

Опис класів з пакету aspect

1) клас LoggingAspect
Цей клас реалізує аспект логування для методів контролерів у Spring-додатку. Аспект дозволяє автоматично записувати логові повідомлення перед викликом методу та після його успішного

виконання.Логування викликів методів здійснюється централізовано, що дозволяє легко відстежувати виконання методів без необхідності додавати логування до кожного методу окремо. Цей аспект логування допомагає стежити за тим, які методи викликаються та які результати вони повертають, що є критичним для ефективного налагодження та моніторингу роботи додатку.

2) клас PerformanceAspect

Клас PerformanceAspect призначений для логування часу виконання методів у сервісному шарі додатку. Він реалізований за допомогою AOP, що дозволяє автоматично вимірювати та логувати час виконання методів без впливу на основну бізнес-логіку. Метод logExecutionTime анотований @Around, що означає, що цей метод буде виконуватися перед і після виконання цільового методу. Ця анотація дозволяє не тільки виконувати додаткові дії до і після методу, але й контролювати сам процес виконання методу. Аспект дозволяє централізовано відстежувати продуктивність методів, що полегшує аналіз і оптимізацію системи. Це значно знижує потребу в додаванні вручну логування у кожен метод.

Опис класів з пакету service

1) клас ClientService

Клас ClientService є сервісом у Spring Framework, який містить бізнес-логіку для роботи з клієнтами. Він використовує два репозиторії: ClientRepository для операцій з клієнтами та OrderRepository для роботи із замовленнями. Анотація @Service вказує на те, що цей клас є сервісним компонентом Spring, який містить бізнес-логіку. Конструктор з анотацією @Autowired впроваджує залежність ClientRepository через конструктор.

1. Метод createClients зберігає список клієнтів у базі даних і повертає збережені сутності.
2. Метод getAllClientsBySurname знаходить і повертає список клієнтів за їх прізвищем.
3. Метод getClientById знаходить і повертає клієнта за його ідентифікатором. Якщо клієнт не знайдений, повертає null.
4. Метод updateClient знаходить існуючого клієнта за ідентифікатором і оновлює його дані новими значеннями. Якщо клієнт не знайдений, викидає виняток.
5. Метод deleteClientById видаляє клієнта і всі його замовлення з бази даних, якщо клієнт існує. Повертає true, якщо видалення успішне, і false, якщо клієнт не знайдений.
6. Метод getClientsByOrderPeriodWithDetails виконує користувацький запит для отримання клієнтів, які мають замовлення у визначеному періоді, і повертає їх разом з деталями замовлень.

Цей клас забезпечує основні операції для управління клієнтами, включаючи створення, читання, оновлення та видалення (CRUD), а також спеціалізовані методи для пошуку клієнтів за певними критеріями.

2) клас OrderService

Цей клас OrderService є сервісним компонентом у Spring Framework, який містить бізнес-логіку для роботи із замовленнями. Він використовує три репозиторії: OrderRepository для операцій із замовленнями, ClientRepository для роботи з клієнтами та ParcelRepository для роботи з посилками.

Метод createOrders зберігає список замовлень у базі даних і повертає збережені сутності. Метод getAllOrdersWithDetails повертає всі замовлення з бази даних. Метод getOrderById знаходить і повертає замовлення за його ідентифікатором. Якщо замовлення не знайдено, повертає null. Метод

deleteOrderById видаляє замовлення за його ідентифікатором, якщо воно існує. Повертає true, якщо видалення успішне, і false, якщо замовлення не знайдено. Метод getOrdersByClientId повертає всі замовлення певного клієнта за його ідентифікатором.
Метод getOrdersWithParcelWeightLessThan5kg знаходить всі замовлення і фільтрує їх, повертаючи ті, у яких вага посилки менше 5 кг. Метод getOrdersByClient повертає всі замовлення певного клієнта.

Цей клас забезпечує основні операції для управління замовленнями, включаючи створення, читання, оновлення та видалення, а також спеціалізовані методи для пошуку замовлень за певними критеріями, такими як клієнт чи вага посилки.

3) клас ParselService

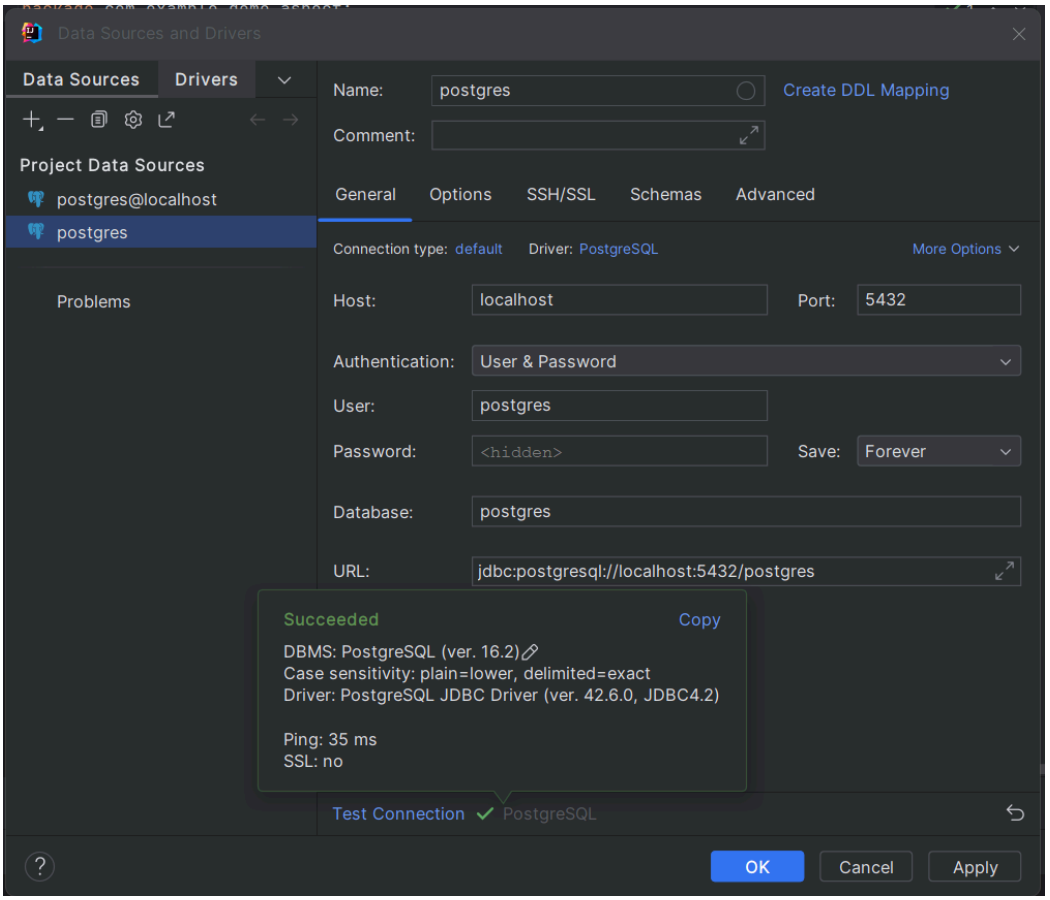
Цей клас ParcelService є сервісним компонентом у Spring Framework, який містить бізнес-логіку для роботи з посилками. Він використовує ParcelRepository для операцій з посилками.

Анотація @Service вказує на те, що цей клас є сервісним компонентом Spring, який містить бізнес-логіку. Залежність ParcelRepository: Репозиторій для роботи з посилками. Конструктор з анотацією @Autowired впроваджує залежність ParcelRepository через конструктор.

Метод createParcels приймає список посилок та зберігає їх у базі даних за допомогою репозиторію. Метод getParcelById знаходить і повертає посилку за її ідентифікатором, якщо посилка не знайдена, повертає null.

Цей клас забезпечує основні операції для управління посилками, включаючи створення нових посилок і отримання інформації про конкретну посилку за її ідентифікатором.

Інтеграція з СУБД



Ці налаштування використовуються для підключення до бази даних PostgreSQL, що працює на локальному комп'ютері (localhost) на стандартному порту (5432). Вхід здійснюється під користувачем postgres з відповідним паролем. Всі ці параметри забезпечують успішне з'єднання, про що свідчить повідомлення про успішне тестове підключення.

Клас SecurityConfig

Цей клас SecurityConfig є конфігураційним класом у Spring Security, який налаштовує безпеку веб-додатка. Він використовує анотації та методи для налаштування аутентифікації та авторизації користувачів.

Анотація @Configuration вказує на те, що цей клас є конфігураційним класом Spring. Анотація @EnableWebSecurity включає підтримку веб-безпеки у Spring.

Метод securityFilterChain налаштовує фільтри безпеки для HTTP-запитів.

http.authorizeHttpRequests визначає правила авторизації для HTTP-запитів, встановлюючи, що будь-який запит повинен бути аутентифікований. http.formLogin налаштовує форму входу, вказуючи на користувацьку сторінку входу (/login) і дозволяючи всім користувачам доступ до неї. http.logout дозволяє всім користувачам доступ до сторінки виходу. http.httpBasic(withDefaults()) включає базову HTTP-аутентифікацію.

Метод userDetailsService налаштовує сервіс для управління даними користувачів. User.withDefaultPasswordEncoder() створює користувача з іменем user, паролем 12345 та роллю USER. InMemoryUserDetailsManager використовується для зберігання даних користувачів в пам'яті.

Цей клас забезпечує базову конфігурацію безпеки, включаючи аутентифікацію користувачів через форму входу та HTTP Basic Authentication, а також зберігання даних користувачів у пам'яті для тестових цілей.

ВИСНОВОК

В даній роботі ми розглянули процес створення інтегрованої системи логістики, використовуючи стек технологій Java Spring. Основна мета полягала в розробці ефективного інструменту для управління ланцюгами постачання, що включає оптимізацію логістичних процесів, відстеження вантажів та замовлень, а також забезпечення безпеки операцій.

Розроблено REST API, що дозволяє інтерактивно працювати з логістичними даними, забезпечуючи доступність основних функцій системи через стандартні HTTP-запити.

Застосування аспектно-орієнтованого програмування (AOP) дозволило відокремити допоміжні функції, такі як логування, контроль доступу і управління транзакціями, від основної бізнес-логіки, що підвищило читабельність та підтримуваність коду.

Завдяки використанню Spring Data JPA забезпечено надійне зберігання та ефективний доступ до даних про вантажі та замовлення, що є критичним для функціонування системи.

Впровадження Spring Security дозволило створити надійну систему аутентифікації та авторизації, що захищає дані від несанкціонованого доступу та забезпечує безпеку операцій.

На прикладі аспекта логування показано, як можна централізовано і ефективно управляти логами викликів методів у контролерах, що є важливою частиною для моніторингу та налагодження системи. Підключення до бази даних PostgreSQL продемонстровано з використанням стандартних інструментів та налаштувань, що забезпечують стабільну роботу системи.

Завдяки використанню архітектури Spring, система легко адаптується до нових вимог і розширюється новими функціями. Застосування Spring AOP та інтеграція з реляційною базою даних забезпечують високу продуктивність та надійність роботи. Використання Spring Security гарантує рівень безпеки для користувачів та даних системи.

Розробка системи логістики на базі Java Spring є потужним рішенням для сучасних бізнес-задач, що дозволяє ефективно управляти ланцюгами постачання. Інтеграція REST API, оптимізація логістичних процесів, надійне зберігання даних та забезпечення безпеки створюють комплексну систему, яка сприяє підвищенню продуктивності та ефективності логістичних операцій. Завдяки використанню передових технологій і підходів, таких як Spring AOP і Spring Security, було створено систему, яка відповідає вимогам сучасного ринку і забезпечує високий рівень якості та надійності.

Перелік використаних джерел

1) Spring in Action" by Craig Walls

2) "Spring Boot in Action" by Craig Walls

3) https://spring.io/projects/spring-boot

4) https://javarush.com/quests/lectures/questspringboot.level01.lecture00

5) https://itproger.com/course/java-spring