Ecommerce application

Fejlesztői dokumentáció – backend

Kód struktúra

A Java backend alkalmazásokban gyakori a különböző rétegek és komponensek szétválasztása funkcionális egységek szerint. Ez a struktúra segít a kód karbantarthatóságának és skálázhatóságának javításában. A lentebb említett mappák mindegyike egy-egy specifikus réteget vagy funkciót képvisel a rendszerben:

1. config

Ez a mappa tartalmazza az alkalmazás konfigurációs osztályait, amelyek a különböző beállításokat és konfigurációkat definiálják. Ide tartozhatnak:

Spring Boot konfigurációk, például biztonsági beállítások, adatbázis kapcsolatok, CORS beállítások.
Bean definíciók, amelyek meghatározzák az alkalmazásban használt komponensek életciklusát.
Profile-specifikus konfigurációk, amelyek különböző környezetekhez (fejlesztési, tesztelési, produkciós) kínálnak eltérő konfigurációkat.

2. controller

A controller mappa az MVC (Model-View-Controller) architektúra "Controller" rétegét implementálja. Itt találhatók azok az osztályok, amelyek a felhasználói interakciókat kezelik

HTTP kérések fogadása és a megfelelő válaszok küldése.
Endpointok definiálása, amelyek meghatározzák az elérhető API útvonalakat.
Kérés adatok validálása és továbbítása a megfelelő szolgáltatások felé.

3. dao (Data Access Object)

A DAO réteg felelős az adatbázis műveletek absztrahálásáért és a Entity osztályok által képviselt adatmodellek kezeléséért. Ez magában foglalja:

CRUD műveletek (Create, Read, Update, Delete).
Adatlekérdezések megfogalmazása, tipikusan SQL vagy JPA (Java Persistence API) használatával.
Adatbázis tranzakciók kezelése.

4. dto (Data Transfer Object)

A DTO mappa azokat az osztályokat tartalmazza, amelyek az adatátviteli objektumokat definiálják. Ezek az objektumok a rétegek közötti adatcserét hivatottak egyszerűsíteni:

Egyszerűsített objektumok, amelyek nem tartalmaznak üzleti logikát, csak adatokat.
Adatok szerializálása és deszerializálása az alkalmazás és a külvilág (pl. frontend vagy más backend szolgáltatások) között.

5. entity

Az entity mappa az adatbázisban tárolt adatokat képviselő Java osztályokat tartalmazza. Ezek az osztályok általában egy-egy adatbázis táblának felelnek meg:

Adatbázis séma definíciói, mint például táblák, oszlopok, kapcsolatok (one-to-many, many-to-one, stb.).
JPA annotációk, amelyek leírják az osztályok és az adatbázis közötti kapcsolatot.
6. service
A service mappa az üzleti logikát tartalmazó osztályokat foglalja magában. Ezek az osztályok felelősek az alkalmazás specifikus műveleteinek végrehajtásáért:

Logikai műveletek implementálása, amelyek az alkalmazás specifikus funkcióit támogatják.
Interakció a DAO réteggel az adatok eléréséhez.
Transzakciókezelés, biztosítva, hogy az adatbázisműveletek konzisztensek maradjanak.
Ezek a mappák együttesen képviselik az alkalmazás architekturális felépítését, segítve ezzel a kód strukturált és jól karbantartható maradását.

Osztálystruktúra, komponensek

config/MyDataRestConfig.java

Komponensek

EntityManager

- **Leírás**: Az EntityManager az JPA entitások kezelésére szolgál. Ez felelős az entitásokkal kapcsolatos műveletekért, mint például lekérdezések és tranzakciók kezelése.
- **Injektálás**: Az EntityManager példány automatikusan injektálódik az osztályba az @Autowired annotáció segítségével, biztosítva, hogy az entitás metaadatai elérhetők legyenek a konfiguráció során.

Konfigurációs Metódusok

configureRepositoryRestConfiguration

- **Cél**: Ez a metódus a REST API alapvető viselkedését konfigurálja, különösen az entitás típusokhoz kapcsolódó HTTP műveletek engedélyezését vagy tiltását.
- **Műveletek Letiltása**: Az osztály letiltja a PUT, POST, és DELETE műveleteket a Product, ProductCategory, Country, és State entitásokra, hogy megakadályozza a nem kívánt adatmanipulációt.
- **Implementáció**: A tiltás végrehajtásához a disableHttpMethods segédmetódus kerül meghívásra minden releváns entitásra.

disableHttpMethods

- **Paraméterek**:
- theClass: Az entitás osztály, amire a műveletek tiltása vonatkozik.
- config: A RepositoryRestConfiguration objektum, amelyen keresztül a konfiguráció megtörténik.
- theUnsupportedActions: A tiltandó HTTP műveletek tömbje.
- **Funkció**: Letiltja az adott HTTP műveleteket az adott entitás típus minden példányára és gyűjteményére.

exposeIds

- **Cél**: Az entitások egyedi azonosítóinak (ID) exponálása a REST válaszokban, amely alapértelmezés szerint nem történik meg.
- **Működés**: A metódus az EntityManager segítségével lekérdezi az összes entitás típust, majd azok Java típusait egy listába gyűjti. Ezt követően a listában szereplő típusok ID-jait exponálja.

Összegzés

A MyDataRestConfig osztály kulcsfontosságú szerepet játszik a REST API biztonságának és funkcionalitásának növelésében. A műveletek letiltásával csökkenti az adatok véletlenszerű vagy szándékos manipulációjának kockázatát, míg az ID-k exponálásával javítja az API használhatóságát. Ez az osztály biztosítja, hogy az alkalmazás adatkezelése kontrollált és biztonságos maradjon.

dao/CountryRepository.java

A CountryRepository interface a com.luv2code.ecommerce.dao csomagban található, és az org.springframework.data.jpa.repository.JpaRepository interface-t

bővíti ki. Ez az interface specifikusan a Country entitás kezelésére szolgál az adatbázisban. A Spring Data JPA által nyújtott standard CRUD (Create, Read, Update, Delete) műveleteken túl lehetőséget biztosít a Country entitások egyszerű lekérdezésére is.

Annotációk és Konfiguráció

@CrossOrigin

- **Cél**: Engedélyezi a CORS (Cross-Origin Resource Sharing) kéréseket a megadott forrásból, ebben az esetben a http://localhost:4200 címről.
- **Használat**: Ez az annotáció lehetővé teszi a frontend alkalmazások számára, amelyek a localhost:4200 címről futnak, hogy hozzáférjenek a CountryRepository által nyújtott adatokhoz. Tipikusan fejlesztési környezetben használatos, ahol a frontend és a backend külön szervereken fut.

@RepositoryRestResource

- **collectionResourceRel**: Megadja a kapcsolódó erőforrások gyűjteményének nevét, ami ebben az esetben "countries". Ez határozza meg, hogy az entitások gyűjteménye hogyan jelenik meg a HAL (Hypertext Application Language) válaszokban.
- **path**: Az API endpoint útvonalát határozza meg, ami "countries". Ez az útvonal lesz használva a Country entitások elérésére az API-n keresztül.

Funkciók és Használat

- **CRUD Műveletek**: A JpaRepository interface kiterjesztésével a CountryRepository automatikusan rendelkezésre álló CRUD műveleteket biztosít. Ez magában foglalja a Country entitások létrehozását, olvasását, frissítését, és törlését.
- **Lekérdezések Egyszerűsítése**: A Spring Data JPA lehetőséget biztosít egyszerű metódus nevek definiálására, amelyek automatikusan lekérdezéseket generálnak. Például, egy metódus, mint findByName(String name), automatikusan generál egy lekérdezést, ami visszaadja az összes Country entitást, amelynek neve megegyezik a megadott paraméterrel.

Összegzés

A CountryRepository interface kritikus szerepet tölt be az e-kereskedelmi platformon belül, mivel kezeli a Country entitások adatbázis-műveleteit, és lehetővé teszi az adatok egyszerű és hatékony elérését a REST API-n keresztül. A @CrossOrigin és @RepositoryRestResource annotációk használatával ez az interface nem csak hogy megkönnyíti a fejlesztők munkáját a CORS problémák kezelésében, hanem egyértelmű és könnyen használható API végpontot is biztosít a Country entitások számára.

dto/Purchase.java

A Purchase osztály a com.luv2code.ecommerce.dto csomagban található, és egy Data Transfer Object (DTO), amely összefoglalja az e-kereskedelmi tranzakció adatait egyetlen objektumban. Ez az osztály az ecommerce platform vásárlási folyamatának különböző entitásait (pl. vásárló, címek, rendelés, rendelési tételek) aggregálja, így egyszerűsíti az adatok kezelését és továbbítását az alkalmazás rétegei között.

Osztály Struktúra és Annotációk

@Data (Lombok)

- **Cél**: Automatikusan generálja a gettereket, settereket, equals(), hashCode() és toString() metódusokat a Purchase osztály számára. Ez a Lombok könyvtár annotációja, amely jelentősen csökkenti a boilerplate kód mennyiségét és javítja az osztály olvashatóságát.

Attribútumok

- **Customer customer**: A vásárlást végző ügyfél entitása. Ez az attribútum a vásárló alapvető információit tartalmazza, mint például név és kapcsolattartási adatok.
- **Address shippingAddress**: A szállítási cím entitása, amely az áru kézbesítési helyét jelöli.
- **Address billingAddress**: A számlázási cím entitása, amely az áru számlázási címét jelöli. Gyakran megegyezik a szállítási címmel, de lehetőség van eltérő cím megadására is.
- **Order order**: Az aktuális rendelés entitása, amely a rendelés összefoglaló adatait tartalmazza, beleértve a rendelés státuszát és az összesített végösszeget.
- **Set<OrderItem> orderItems**: A rendeléshez tartozó rendelési tételek gyűjteménye. Egy Set gyűjteményben vannak tárolva, amely biztosítja, hogy minden tétel egyedi legyen. Az OrderItem entitások az egyes termékek adatait tartalmazzák, mint például a termék azonosítót, nevét, mennyiségét és egységárát.

Funkciók és Használat

- **Adatok Csoportosítása**: A Purchase DTO a rendelési folyamat során keletkező összes releváns adatot egy objektumban csoportosítja, így egyszerűsítve az adatok kezelését, validálását és továbbítását az alkalmazás különböző rétegei között.
- **Adatátvitel**: Ez az osztály ideális választás az adatátvitelre a frontend és a backend, vagy a különböző backend szolgáltatások között, mivel minden releváns információt tartalmaz a vásárlással kapcsolatban.

Összegzés

A Purchase DTO kulcsfontosságú szerepet játszik az e-kereskedelmi tranzakciók kezelésében, mivel összefogja és szabványosítja az adatokat, amelyek szükségesek a vásárlási folyamat végrehajtásához. Ezáltal hozzájárul az alkalmazás modularitásának és karbantarthatóságának javításához. Az osztály használatával javul az adatkezelés hatékonysága és a fejlesztési folyamat átláthatósága.

entity/Address, Country, Customer

Az ecommerce csomagban található entitás osztályok (Address, Country, Customer) az adatmodell alapvető elemeit képezik. Ezek az osztályok az adatbázis táblákkal vannak összekapcsolva, és a JPA (Java Persistence API) annotációkat használják az adatbázis műveletek kezelésére.

Address **Entitás**

Leírás

Az Address entitás az ügyfelek címadatait tárolja. Ez magában foglalja az utca, város, állam, ország, és irányítószám adatokat, valamint egy egyedi kapcsolatot a Order entitással.

Mezők és Kapcsolatok

- **id** (Long): Az egyedi azonosító.
- **street** (String): Az utca neve.
- **city** (String): A város neve.
- **state** (String): Az állam neve.
- **country** (String): Az ország neve.
- **zipCode** (String): Az irányítószám.
- **order** (Order): Egy-egy kapcsolat a rendeléssel, amely a címet használja.

Annotációk

- @Entity: Jelzi, hogy az osztály egy entitás.
- @Table(name="address"): Megadja az adatbázis tábla nevét.
- @Id és @GeneratedValue: Azonosító generálása.
- @OneToOne és @PrimaryKeyJoinColumn: Kapcsolat a Order entitással.

Country **Entitás**

Leírás

A Country entitás országokat reprezentál, mindegyik országnak van egy egyedi kódja, neve, és államainak listája.

Mezők és Kapcsolatok

- **id** (int): Az egyedi azonosító.
- **code** (String): Az országkód.
- **name** (String): Az ország neve.
- **states** (List<State>): Kapcsolódó államok listája.

Annotációk

- @OneToMany(mappedBy = "country"): Egy-egy kapcsolat a State entitásokkal.
- @JsonIgnore: Megakadályozza a JSON serializálás során a states mező feldolgozását.

Customer **Entitás**

Leírás

A Customer entitás a vásárlókat jelöli, tartalmazza azok alapvető információit, mint például név és email, valamint a hozzájuk kapcsolódó rendelések listáját.

Mezők és Kapcsolatok

- **id** (Long): Az egyedi azonosító.
- **firstName** (String): Vásárló keresztneve.
- **lastName** (String): Vásárló vezetékneve.
- **email** (String): Vásárló email címe.
- **orders** (Set<Order>): A vásárlóhoz tartozó rendelések halmaza.

Annotációk

- @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL, fetch = FetchType.EAGER): Egy-egy kapcsolat a Order entitásokkal, az EAGER fetch típussal a kapcsolódó rendelések azonnal betöltődnek.

Összegzés

Ezek az entitások kritikus szerepet játszanak az ecommerce alkalmazás adatmodelljében, mivel a vásárlási folyamat alapvető elemeit képezik. A kapcsolatok és a mezők jól definiáltak, biztosítva az adatok integritását és a könnyű elérhetőséget az alkalmazás többi része számára. A kapcsolódó entitások és az adatbázis műveletek kezelése hatékonyan történik a JPA annotációk használatával.

Service-ek

CheckoutService.java

A CheckoutService interface a com.luv2code.ecommerce.service csomagban helyezkedik el, és definiál egy kulcsfontosságú szolgáltatási szerződést az e-kereskedelmi platformon belül. Ez az interface felelős a vásárlási folyamat kezeléséért, különösen a vásárlások megerősítéséért és a rendelések feldolgozásáért.

Funkciók

placeOrder

- **Leírás**: A placeOrder metódus egy vásárlást dolgoz fel, létrehozva a szükséges rendelést az adatbázisban, és visszaadva a vásárlási tranzakció eredményét egy PurchaseResponse objektumban.
- **Paraméterek**:
  - purchase (Purchase): A Purchase DTO, amely tartalmazza az összes szükséges információt a rendelés elkészítéséhez, beleértve a vásárló adatokat, szállítási és számlázási címeket, valamint a rendelt termékek részleteit.
- **Visszatérési Érték**:
  - PurchaseResponse: Egy objektum, amely visszaadja a rendelési tranzakció eredményét, tipikusan tartalmazza a rendelés azonosítóját vagy egy egyedi követési számot, amely lehetővé teszi a vásárló számára a rendelés nyomon követését.
- **Folyamat**:
  - A metódus először validálja a purchase objektumot a benne foglalt információk alapján.
  - Létrehozza a rendelést az adatbázisban, amely magában foglalja a vásárló adatainak, címének és a rendelési tételeknek a mentését.
  - Visszaad egy PurchaseResponse objektumot, amely jelzi a feldolgozás sikerességét és tartalmazza a rendelési azonosítót.

Használati Példák

java

Copy code

// Példa a CheckoutService használatára public class CheckoutProcess { private CheckoutService checkoutService; public CheckoutProcess(CheckoutService service) { this.checkoutService = service; } public void completeOrder() { Purchase purchase = createPurchase(); // Feltételezzük, hogy ez a metódus létrehozza a szükséges Purchase objektumot PurchaseResponse response = checkoutService.placeOrder(purchase); System.out.println("Order placed with tracking number: " + response.getOrderTrackingNumber()); } }

Összefoglalás

A CheckoutService interface esszenciális a com.luv2code.ecommerce alkalmazás vásárlási folyamatának megvalósításában, mivel központi szerepet játszik a rendelések kezelésében és a vásárlási tranzakciók megerősítésében. A placeOrder metódusának implementációja biztosítja, hogy a vásárlások hatékonyan és biztonságosan legyenek feldolgozva, növelve ezzel az ügyfél elégedettséget és hozzájárulva az üzleti folyamatok zavartalan működéséhez.

CheckoutServiceImpl.java

A CheckoutServiceImpl osztály a CheckoutService interfész implementációja, amely a com.luv2code.ecommerce.service csomagban helyezkedik el. Ez a szolgáltatás felelős a vásárlási folyamatok kezeléséért, beleértve a rendelések létrehozását, azok azonosítójának generálását, és az adatok adatbázisba való mentését.

Függőségek
CustomerRepository: Az ügyfelekkel kapcsolatos adatbázis-műveleteket kezeli. A repository-t a Spring konténere automatikusan injektálja a konstruktoron keresztül.
Fő Metódusok
placeOrder
Paraméterek:
purchase (Purchase): A vásárlás adatait tartalmazó DTO, amely magában foglalja a vásárló információit, a szállítási és számlázási címeket, valamint a rendelés tételeit.
Folyamat:
Rendelés Információinak Lekérése: Az Order objektumot kinyeri a Purchase DTO-ból.
Nyomkövetési Szám Generálása: Egyedi rendelési azonosítót generál (orderTrackingNumber), melyet az Order objektumhoz rendel.
Rendelési Tételek Hozzáadása: A rendeléshez tartozó tételeket (OrderItem) hozzáadja az Order objektumhoz.
Címek Hozzárendelése: A szállítási és számlázási címeket beállítja az Order objektumban.
Ügyfél és Rendelés Összekapcsolása: A vásárló objektumot (Customer) frissíti az új rendeléssel.
Adatok Mentése: Az ügyfél objektumot, beleértve a hozzá kapcsolódó rendeléseket és címeket, elmenti az adatbázisba a CustomerRepository segítségével.
Visszatérés:
PurchaseResponse: Objektum, amely tartalmazza a generált rendelési nyomkövetési számot.
generateOrderTrackingNumber

Leírás: Véletlenszerű UUID sztring generálása, amelyet mint rendelési nyomkövetési szám használnak.
Visszatérés: Egy véletlenszerűen generált UUID sztring, amely egyedileg azonosítja a rendelést.
Példa Kód
java
Copy code

```
public class CheckoutProcessDemo {
    private CheckoutService checkoutService;

    public CheckoutProcessDemo(CheckoutService service) {
        this.checkoutService = service;
    }

    public void processOrder() {
        Purchase purchase = createDummyPurchase();  // Egy példa Purchase
objektum létrehozása
        PurchaseResponse response = checkoutService.placeOrder(purchase);
        System.out.println("Order placed with tracking number: " +
response.getOrderTrackingNumber());
    }

    private Purchase createDummyPurchase() {
        // Példa adatokkal töltsük fel a Purchase objektumot
        return new Purchase();
    }
}
```

Összefoglalás
A CheckoutServiceImpl osztály alapvetően fontos a vásárlási tranzakciók kezelésében, ahol biztosítja, hogy a rendelések hatékonyan és következetesen kerüljenek feldolgozásra. A szolgáltatás integrálja a rendelési adatokat, a vásárlói adatokat és a szállítási információkat, majd biztonságosan tárolja azokat az adatbázisban. A generált nyomkövetési szám további biztonságot és nyomon követhetőséget kínál a vásárló számára.