Cipő Webáruház Backend – Fejlesztői dokumentáció

Tartalom

[Végpontok: 1](#_Toc133784592)

[AppModule: 17](#_Toc133784593)

[Entitások: 18](#_Toc133784594)

[Dto-k: 29](#_Toc133784595)

CTRL+kattintással lehet navigálni a tartalmak között.

# Végpontok:

A képen szöveg, válaszfal, képernyőkép, ezüst látható

Automatikusan generált leírás

Ez egy POST HTTP kéréskezelő metódus egy /register végponttal, amely a RegisterDto típusú adatokat várja a testben. Amikor a metódus meghívódik, a felhasználói adatokat kinyeri a registerDto objektumból, majd létrehoz egy User példányt, amelyet elment az adatbázisban. A jelszó hash-elése előtt a metódus az await bcrypt.hash segítségével hash-elni fogja a jelszót a biztonságos tárolás érdekében. A metódus visszatér egy JSON objektummal, amely az elmentett felhasználó adatait tartalmazza.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód a bejelentkezést kezeli egy API-n keresztül. A @Post('/login') annotáció jelzi, hogy ez egy HTTP POST kérésre reagál. A loginDto egy RegisterDto típusú objektum, amely tartalmazza a felhasználói azonosítót (email vagy felhasználónév) és a jelszót.

A kód megnézi, hogy van-e olyan felhasználó az adatbázisban, akinek az azonosítója (email vagy felhasználónév) megegyezik a loginDto-ban kapott azonosítóval. Ha nincs ilyen felhasználó, akkor hibát dob. Ha van ilyen felhasználó, akkor ellenőrzi a jelszót. Ha a jelszó helytelen, akkor hibát dob.

Ha a bejelentkezés sikeres, akkor a kód JSON objektumot ad vissza, amely tartalmazza az üzenetet ('success'), az access token-t (JSON Web Token), és a felhasználó adatait (id, felhasználónév, email). Az access token-t a kód a JWTService segítségével generálja, majd beállítja a HTTP cookie-ban.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy GET HTTP kérést vár a '/user' elérési útra. Az endpoint célja, hogy ellenőrizze, hogy a bejelentkezett felhasználó érvényes hozzáférést rendelkezik-e a védett erőforrásokhoz. A kérésben a kliens által küldött JSON Web Token (JWT) sütiként (cookie) van átadva. Az endpoint megpróbálja ellenőrizni a token érvényességét, és ha az érvényes, akkor a visszatérési érték null lesz. Ha a token érvénytelen, akkor az endpoint egy UnauthorizedException kivételt dob.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy GET kérést végrehajtó végpontot definiál a '/shoes' elérési útvonalra. Az aszinkron függvény a cipők adatainak listáját adja vissza. Az adatokat a Product entitásból és a Category és Stock entitásokból szerez be, amelyek a relációs táblákra utalnak. A lekérdezéshez az adatbázis-kezelő keretrendszer (TypeORM) API-ját használja.

A lekérdezés olyan adatokat tartalmaz, mint a cipő neve, leírása, ára, képei és a hozzájuk rendelt készletméretek. A keresés kizárólag az inaktív cipőket kizárja. A visszaadott adatokat egy Promise-ként adja vissza, amelynek értéke egy tömb a talált cipőkkel.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy GET kérést definiál, amely az '/releases' elérési útvonalon hallgatózik. Az üzenetküldő aszinkron, és egy adatbázis-lekérdezést hajt végre, hogy visszaadja az összes kiadást a 'Release' táblából. A visszaadott adat JSON formátumban lesz.  
  
A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy GET kéréskezelőt hoz létre a '/shoes/popular' útvonalra. A kérésre adott válasz a termékek listája, amelyek népszerűek és aktívak. A termékek listáját a Product adatbázis táblából állítják össze, csatlakoztatva a Category táblához a bal oldali JOIN segítségével. A feltételekkel a népszerű és az inaktív termékeket szűrik. A válasz tartalmazza a termék nevét, leírását, képeit, árát, státuszát és kategóriáját.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód a "/shoes/:id" útvonalra definiál egy GET kérést, ami az adott azonosítójú cipő adatait kérdezi le az adatbázisból. Az ":id" egy dinamikus útvonalparaméter, amely az adott cipő azonosítójára utal.

A kód először lekéri a Product entitásra vonatkozó repository-t, majd a query builder használatával lekérdezi az adott azonosítójú cipőt az adatbázisból. A lekérdezés LEFT JOIN-el összekapcsolja a Category és a Stocks táblákkal is, és a visszatérő eredmény egyetlen objektum lesz, amely tartalmazza a cipő adatait és azokat az információkat, amelyeket a JOIN-ok során összegyűjtött. A visszatérő eredmény tehát egyetlen Product objektum lesz.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy GET HTTP kérésre válaszol a /shoes/name/:name elérési úton. A name paraméter a keresett cipő nevét tartalmazza.

A függvény először lekéri a Product entitásokat a Product táblából az adatforrásából. Ezután a lekérdezést az adatbázis lekérdezési építőjével állítja össze. A lekérdezés WHERE feltétele tartalmaz egy keresési mintát, amely megtalálja a cipőket, amelyeknek a neve tartalmazza a megadott nevet. Az inaktív cipőket kizárják a lekérdezésből.

Végül a függvény visszaadja a talált cipőket egy tömbben.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód az általunk megadott id-vel rendelkező cipők lekérdezését végzi a megfelelő kategória alapján. Az első lépésben létrehozunk egy productRepo változót, amely az adatbázis Product táblájára mutat. Ezután elkészítjük az adatbáziskérdezést a createQueryBuilder függvénnyel, amely a product táblán dolgozik. A leftJoinAndSelect függvénnyel az category táblát is bekapcsoljuk, hogy azonosítsuk a megfelelő kategóriát. Az id-vel megegyező category.id szerinti szűrés végrehajtása után az eredményt visszaadjuk a products változóban. Végül visszatérünk a products-ban található cipőkkel.



Ez a kód egy GET HTTP kérésre válaszol a /shoes/category/:name útvonalon. A kód lekérdezi a Product entitásokat a megadott kategória neve alapján. Az adatbázisból a TypeORM segítségével kérdez le adatokat. A lekérdezés során az alábbi lépéseket hajtja végre:

* Létrehoz egy Product entitásokat lekérdező objektumot.
* Kapcsolódik a Category entitáshoz bal oldali csatlakozással (leftJoinAndSelect).
* A Category entitás neve alapján szűri a lekérdezést.
* Az eredményül kapott Product entitásokat adja vissza egy tömbben.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy GET kérést végrehajtó API végpontot definiál, amely az összes felhasználót lekéri az adatbázisból és visszatér az eredményekkel. A kód használja az @Get() dekorátort a '/users' útvonalhoz, és az async kulcsszó jelzi, hogy a metódus aszinkron módon fut. Az adatok lekérdezése az adatbázisból a User entitás rétegén keresztül történik a getRepository() metódus meghívásával. A find() metódus lekéri az összes felhasználót az adatbázisból és visszaadja az eredményeket.

A képen szöveg látható

Automatikusan generált leírás

Ez a kódrészlet egy GET kérés kezelő funkció, amely lekéri az adatbázisból a megadott felhasználó azonosítójú felhasználó rekordját. Az Express keretrendszerben a @Get() dekorátorral jelölt útvonalhoz tartozik.

A funkció első paramétere az @Param() dekorátorral van ellátva, ami azt jelzi, hogy a következő paraméter az útvonalban található dinamikus :id paraméter értéke. A paraméter neve itt 'id'.

Az adatok lekérdezése az adatbázisból a TypeORM által biztosított adatbázis objektumon (Repository) keresztül történik. Ebben a példában az adatbázis objektum a User entitásra van konfigurálva. Az adatbázisból való lekérdezéshez az adatbázis objektum findOneBy() metódusát használjuk, és az id értéket a funkció első paramétereként átadjuk.

A funkció visszatérési értéke az adatbázisból lekért felhasználó rekordja. Ha az adatbázis nem tartalmaz ilyen felhasználót, akkor a visszatérési érték null.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód a felhasználókat adja vissza az adatbázisból azon felhasználó neve alapján, amelyet a name paraméterként megkapott értékkel kapcsolatban talál. Az async getUsersByName() metódus lekéri a User modellekhez tartozó repositoryt, majd visszaadja az adatbázisban a username mező értékével egyező User modelleket.  
  
A képen szöveg látható

Automatikusan generált leírás

Ez a kód a felhasználók között keresi meg azt a felhasználót, akinek az email címe megegyezik a kapott paraméterrel. A kód lekér egy User típusú adatbázis repo-t, majd visszatér az adatbázisban megtalált felhasználóval, aki rendelkezik a paraméterként kapott email címmel.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy POST végpontot definiál, amely lehetővé teszi az új tételek hozzáadását a felhasználó kosarához. A kosár tételét a felhasználói által kiválasztott termékre, méretre és mennyiségre vonatkozó adatok alapján hozzuk létre.

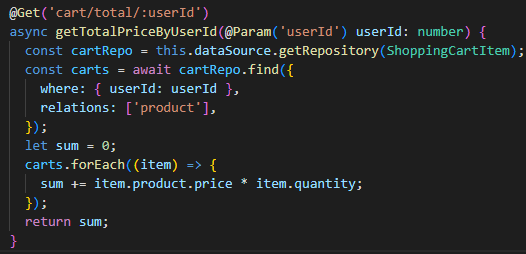
A kód egy új ShoppingCartItem példány létrehozásával kezdődik, majd az adatbázisból lekéri a megadott terméket, méretet és felhasználót. Ezután beállítja ezeket a kapcsolódó tulajdonságokra az új ShoppingCartItem példányban. Végül elmenti a kosár tételét az adatbázisban és visszaadja azt a válaszként.

A pontos működéshez szükségesek a CartDto DTO-ban definiált mezők, amelyek tartalmazzák a kiválasztott termék azonosítóját (productId), a méretet (sizeId) és a felhasználó azonosítóját (userId), valamint a kívánt mennyiséget (quantity).

A képen szöveg látható

Automatikusan generált leírás

Ez a kód a felhasználó kosarában található elemek lekérdezésére szolgál. A @Get dekorátor hatására a funkció elérhető az API-n keresztül az /cart/:userId végponton keresztül, ahol a userId egy dinamikus paraméter. A kód az adatbázisból elkéri a ShoppingCartItem entitásokat a userId alapján, majd visszatér ezekkel az elemekkel egy tömbben.



Ez a kód a felhasználó által kosárba helyezett termékek árának kiszámítását végzi el. Az @Get('cart/total/:userId') végpontot lekérdezéskor a felhasználó által helyezett összes terméket lekéri az adatbázisból a ShoppingCartItem táblából, majd összeszámolja az árat, amelyet a felhasználó fizetne az összes termékért. Az összegzéshez kapcsolja a Product táblához tartozó árakat és a quantity mezőben tárolt darabszámokat. A kiszámított végösszeget adja vissza a lekérdezés eredményeként.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy DELETE HTTP kérést definiál az /cart/delete/:id végponton. A :id paraméter dinamikus, és az adott kosár elem azonosítóját tartalmazza. Az async deleteCartItem() függvény törli a megadott id-val rendelkező kosár elemet az adatbázisból. Először lekérdezi a ShoppingCartItem entitást reprezentáló repository-t, majd az adott id-val rendelkező elemet keresi az adatbázisban. Ezután a cartRepo.delete(cart) segítségével törli a megtalált elemet, majd visszatér az eredménnyel.

A képen szöveg látható

Automatikusan generált leírás

Ez a DELETE HTTP kérés kezelőfüggvénye a /cart/delete/user/:userId végponton. A userId egy azonosító, amelyet a kérés paraméterként kap.

A függvény a ShoppingCartItem adatbázis-tábla entitásainak Repository-ját használja a dataSource adatforrásból. A cartRepo.delete() metódusát hívja meg, hogy törölje az összes ShoppingCartItem objektumot, amelyeknek a userId mezője megegyezik a userId paraméter értékével. A függvény nem tér vissza semmilyen értékkel, ha a törlés sikeres volt.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy POST kérésre válaszol, ami az "/like" végponton érkezik a szerverre. Az @HttpCode(200) dekorátor beállítja a válasz státusz kódját 200-ra, ami azt jelenti, hogy a kérés sikeres volt.

Az @Body() dekorátorral az adatokat a kérés testében várja, amit a LikeDto típusú változóba tölt be.

Ezután a kód az adatbázisból először lekéri a Product és User modellek egy-egy példányát a megadott id alapján, majd létrehoz egy új Like példányt, amelyhez hozzárendeli ezeket a modelleket. Ezután elmenti a likeRepo adatbázis objektumba, majd a like példányt adja vissza a kérésre válaszként.

A képen szöveg látható

Automatikusan generált leírás.

Ez a kód egy "Like" törlési műveletet valósít meg a megadott "productId" és "userId" paraméterek alapján.

A @Delete dekorátor beállítja a HTTP DELETE kérést, amely lehetővé teszi az erőforrások törlését a szerveren. A dekorátoron belül az elérési útvonalat /like/:productId/:userId-re állítják be, ami azt jelenti, hogy a kliensnek a szerver ezen az útvonalon kell kérnie a törlési műveletet.

Az @Param dekorátor használatával a megadott "productId" és "userId" paramétereket kinyerik az URL-ből. Ezeket az értékeket azután a műveletben használják.

A kódban meghívják az adatforrás Like entitásra vonatkozó tárolóját, majd az adatbázisból lekérdezik azt a Like entitást, amelynek az "productId" és "userId" mezői megegyeznek a megadott értékekkel. Az eredményt egy "like" változóba mentik, majd a delete metódust hívják meg a "like" objektum törléséhez az adatbázisból. A művelet végül visszatér a törölt "like" objektummal.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy GET HTTP kérést definiál a /like/product/:id útvonalra. Az id egy szám típusú paraméter, ami az adott termék azonosítóját jelöli. A kód lekérdezi a Like entitásokat a megadott termék azonosítóval rendelkező elemek között az adatbázisból, majd visszatér ezekkel a Like entitásokkal egy tömbben.

Ez magyarul azt jelenti, hogy amikor egy kérés érkezik a /like/product/:id útvonalra, a szerver visszaadja az adott id-vel rendelkező termékhez kapcsolódó Like entitásokat az adatbázisból.

A képen szöveg, válaszfal, képernyőkép látható

Automatikusan generált leírás

Ez a kód a megadott felhasználóhoz tartozó összes kedvelt terméket kéri le az adatbázisból, majd azokat egy tömbként adja vissza. A visszaadott tömb minden eleme egy objektum, amely a termék és a hozzá tartozó adatokat tartalmazza.

A kód lekéri az adatbázisból az összes kedvelt terméket, amelyeket a megadott felhasználó kedvelt (liked). A kapcsolatokat az adatbázisban definiált összekapcsolási logikák alapján állítja össze. A visszaadott tömb minden eleme egy objektum, amely az adott kedvelt termék és azok adatokat tartalmazza, mint például az azonosító, a neve és az első kép url címe.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy POST kérésre ad választ az /orderitem végponton. Az OrderItemDto objektumot tartalmazza a testben, ami az új rendelés tételeket írja le. Az OrderItem entitáshoz tartozó repository-t kéri le az adatforrásból, majd létrehoz egy új OrderItem objektumot, amit feltölt a kérés testéből kapott adatokkal, mint a termék (Product), mennyiség (quantity), méret (size), és a rendelés (order). Az adatok mentése után az új OrderItem objektummal tér vissza a kérésre adott válaszban.

A képen szöveg látható

Automatikusan generált leírás

Ez a kódrészlet egy GET HTTP kérést reprezentál, amely az /orderitem/:id végpontot célozza meg. Az id paraméter a megrendelés azonosítóját jelöli.

A getOrderItemsById függvény lekéri az adatbázisból azokat az OrderItem objektumokat, amelyeknek az orderId mezője megegyezik az id paraméter értékével. Az adatbáziskapcsolatot a dataSource szolgáltatáson keresztül éri el a getRepository metódus segítségével, majd az adatbázisból lekért adatokat visszaadja a hívónak.

A képen szöveg látható

Automatikusan generált leírás

Ez a kódrészlet egy GET kérésre reagál az `/shippingmethod` útvonalon. A visszatérési értéke a szállítási módok listája a `ShippingMethod` adatbázis-táblából. A `getRepository()` metódus meghívásával a `dataSource` adatforrásból lekérjük a `ShippingMethod` adatbázis-tábla repository objektumát, majd a `find()` metódussal visszatérünk az összes szállítási móddal.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy GET kérést hajt végre az `/shippingmethod/:id` útvonalon, ahol az `id` paraméter adott értékre van beállítva. Az adatbázisból azonosító alapján keresi meg a megfelelő szállítási módot a `ShippingMethod` entitásból, majd visszatér a talált szállítási mód adataival.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy GET kérést indít a "/paymentmethod" végpontra, amely visszaadja az összes elérhető fizetési módot az adatbázisból. Az adatbázis hozzáférést a PaymentMethod entitás Repository-jának létrehozásával éri el.

A képen szöveg látható

Automatikusan generált leírás

Ez a kódrészlet egy GET végpontot definiál a /paymentmethod/:id útvonalon, amely lekér egy fizetési módszert az azonosítója alapján. Az id egy dinamikus útválasztó paraméter, amely az URL része, és amelynek értékét a felhasználó adja meg a kérés során.

A kód először meghívja a getRepository() metódust, hogy lekérje a PaymentMethod entitásra vonatkozó repository-t, majd meghívja a findOneBy() metódust, hogy megtalálja a megfelelő fizetési módszert az adott azonosító alapján. A találat eredményeként visszatér a talált fizetési mód entitásával.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy új címadatot ad hozzá az adatbázishoz. A `@Post('address')` dekorátor azt jelzi, hogy a hálózaton érkező HTTP POST kéréseket kezeli, amelyek az `/address` elérési útvonalon érkeznek. Az `@Body()` dekorátor azt jelzi, hogy a beérkező kérés teste egy `AddressDto` típusú objektum lesz, amelyet a függvény belsejében használnak fel.

A függvény először létrehoz egy `Address` típusú objektumot, majd hozzáadja a kérésben kapott város, irányítószám, állam és utcaadatokat az objektumhoz. Ezután hozzáadja az új címet az adatbázishoz a `addressRepo.save()` metódus használatával, majd visszaküldi az elmentett cím adatait a kliensnek, amelynek azonosítója is bekerül az adatok közé a `{ ...savedAddress, id: savedAddress.id }` kifejezéssel.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy GET kérési útvonalat hoz létre az /address/:id címen, amely a megadott ID-val rendelkező címet keresi meg az adatbázisban. Az ID-t az útvonal paraméterében kapja meg, majd az adatbázisból az Address entitás repo-jának segítségével lekérdezi a megfelelő címet. A visszaadott eredmény egy Promise, amely tartalmazza az adatbázisból lekért címet.



Ez a kód egy új megrendelés hozzáadását teszi lehetővé az "/order" végponton keresztül. A HTTP POST kérésben az OrderDto objektumot várja testként, amely tartalmazza az új megrendelés részleteit, például a felhasználó azonosítóját, a szállítási címet, a fizetési és szállítási módokat stb.

Az új megrendelés létrehozásához a kód lekérdezi a kapcsolódó adatbázis-táblákból (Address, PaymentMethod, ShippingMethod és User) a megfelelő rekordokat a megadott azonosítók alapján, majd azokat hozzárendeli az új megrendeléshez. A kód az új megrendelés dátumát is beállítja, majd elmenti az adatbázisban.

A függvény visszatérési értéke az elmentett Order objektum a kiegészített id tulajdonsággal.

A képen szöveg látható

Automatikusan generált leírás

Ez a kódrészlet egy GET API végpontot definiál a '/stock/:productId' útvonalon, amely a termék raktárkészletének lekérdezésére szolgál a megadott termékazonosító alapján. A végpont megkapja a termék azonosítóját a dinamikus paraméterként megadott ':productId' változóból, majd ezt használja a megfelelő raktárkészlet rekordok lekérésére az adatbázisból. Az eredményként visszaadott adatok a termékhez tartozó összes raktárkészletet tartalmazzák.

A képen szöveg látható

Automatikusan generált leírás

Ez a PUT kérés csökkenti a készletet a megadott productId-hoz és sizeId-hoz tartozó elemen, a megadott number mennyiséggel. Először lekéri az adott productId és sizeId alapján az adott raktári tételt a Stock adatbázis táblából, majd csökkenti az inStock értékét a megadott number mennyiséggel. Ezután az adatbázisban elmenti a módosított raktári tételt, majd visszatér a módosított tétellel. Ha az adott productId és sizeId alapján nincs raktári tétel a Stock adatbázisban, akkor hibaüzenetet dob a NotFoundException segítségével.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy API végpontot definiál, amely lehetővé teszi az üzleti készlet módosítását a megadott termékhez és mérethez.

A PUT /stock/add/:productId/:sizeId/:number végpont a következő paraméterekkel rendelkezik:

* productId: a termék azonosítója
* sizeId: a méret azonosítója
* number: a készlethez hozzáadott darabszám

A kód először lekéri a megfelelő készletet a megadott termékhez és mérethez az adatbázisból. Ha a készlet nem található, akkor egy NotFoundException kivételt dob. Ha a készlet megtalálható, akkor hozzáadja a megadott darabszámot a jelenlegi készlethez, majd elmenti az adatbázisban. A végpont a módosított készletet adja vissza.

# AppModule:

A képen szöveg látható

Automatikusan generált leírás

Ez a kód az AppModule nevű modult definiálja, amely az alkalmazás fő modulja. Az import szekcióban megadja azokat az összetevőket, amelyeket a modul használ. A TypeOrmModule modul segítségével definiálja az adatbáziskapcsolatot, valamint az adatbázis entitásait, például a User, Address, Product, stb. A synchronize opció segítségével a modul szinkronizálja az adatbázis sémáját az entitásokkal.

A JwtModule modul használatával hozzáférést biztosít a JSON Web Token-hez, amely lehetővé teszi az autentikáció és az autorizáció kezelését az alkalmazásban. A signOptions lehetővé teszi a tokennél használt titkos kulcs és lejárati idő beállítását.

A controllers tömbben az alkalmazás vezérlői (controller) találhatóak, ahol az AppController van megadva.

A providers tömbben az alkalmazás szolgáltatásait definiálja, jelen esetben csak az AppService van megadva.

# Entitások:

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy TypeORM entitás definíciója a ShoppingCartItem táblához, amely egy felhasználó kosárjában található tételt reprezentál. A kód az alábbi mezőket és kapcsolatokat tartalmazza:

* @Entity(): Az osztályt entitásként definiálja a TypeORM számára.
* @PrimaryGeneratedColumn(): Egy automatikusan generált, egyedi azonosítót hoz létre a ShoppingCartItem táblában.
* @ManyToOne(() => User, (user) => user.cartItems): Egy User entitással való Many-to-One kapcsolatot hoz létre, amely a felhasználót köti össze a kosár tétellel. A user.cartItems egy olyan mező a User entitásban, amely az adott felhasználó kosarának összes elemét tartalmazza.
* @Column() userId: number;: Egy szám típusú mező, amely az adott ShoppingCartItem elemhez tartozó felhasználó azonosítóját tartalmazza.
* @Column() quantity: number;: Egy szám típusú mező, amely a kosárban található termék darabszámát tartalmazza.
* @ManyToOne(() => Product, (product) => product.cartItems): Egy Product entitással való Many-to-One kapcsolatot hoz létre, amely a terméket köti össze a kosár tétellel. A product.cartItems egy olyan mező a Product entitásban, amely az adott termékhez tartozó kosár elemeket tartalmazza.
* @Column() productId: number;: Egy szám típusú mező, amely az adott ShoppingCartItem elemhez tartozó termék azonosítóját tartalmazza.
* @ManyToOne(() => Size): Egy Size entitással való Many-to-One kapcsolatot hoz létre, amely a termék méretét köti össze a kosár tétellel.
* @Column() sizeId: number;: Egy szám típusú mező, amely az adott ShoppingCartItem elemhez tartozó termék méretének azonosítóját tartalmazza.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy TypeORM entitás definíciót tartalmaz az Order nevű adattáblához. Az @Entity() dekorátorral jelöli az osztályt, hogy az egy adattábla entitás.

Az adattábla mezőit a dekorátorokkal jelölt osztályváltozók határozzák meg. Az @PrimaryGeneratedColumn() dekorátor a táblához tartozó elsődleges kulcsot definiálja, ami automatikusan generált értékkel lesz feltöltve. Az @Column() dekorátorokkal jelölt változók az adattábla oszlopait határozzák meg.

Az osztályváltozók közül az orderDate egy dátum típusú változó, ami az adott rendelés dátumát tartalmazza. A user változó egy User típusú entitással áll kapcsolatban, ami a felhasználót reprezentálja, aki a rendelést leadta. A shippingMethod és paymentMethod változók a szállítási és fizetési mód entitásával vannak kapcsolatban.

Az @ManyToOne() dekorátor jelzi, hogy az adott osztályváltozó egy másik entitáshoz tartozó adattáblával áll kapcsolatban. Az @JoinColumn() dekorátor az adattáblák közötti kapcsolatot határozza meg. Az orderItems változó egy OrderItem típusú entitással áll kapcsolatban, ami a rendelésben szereplő termékeket és azok mennyiségét tartalmazza.

Az Order entitás definíciója azt is megmondja, hogy az address változó egy másik entitással áll kapcsolatban, amelyet az @ManyToOne() dekorátor jelöl. Az Address entitásban definiált azonosítóhoz tartozó adatbázis oszlopot az @JoinColumn() dekorátor segítségével lehet beállítani.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy TypeORM által használt entitás (modell) definícióját tartalmazza, ami az "OrderItem" nevet viseli. Az "OrderItem" reprezentál egy adott termék egy adott megrendelésen belüli pozícióját.

A modellnek a következő mezői vannak:

* "id": Az "OrderItem" egyedi azonosítója, automatikusan generált szám.
* "quantity": A megrendelt termékek mennyisége.
* "product": Az "OrderItem"-hez tartozó termék, mely egy másik entitásra hivatkozik, amelynek típusa "Product".
* "productId": Az "OrderItem"-hez tartozó termék azonosítója, mely az adatbázisban található "Product" entitás egyedi azonosítója.
* "order": Az "OrderItem"-hez tartozó megrendelés, mely egy másik entitásra hivatkozik, amelynek típusa "Order".
* "orderId": Az "OrderItem"-hez tartozó megrendelés azonosítója, mely az adatbázisban található "Order" entitás egyedi azonosítója.
* "size": Az "OrderItem"-hez tartozó méret, mely egy másik entitásra hivatkozik, amelynek típusa "Size".
* "sizeId": Az "OrderItem"-hez tartozó méret azonosítója, mely az adatbázisban található "Size" entitás egyedi azonosítója.

Az entitás definíciójának használata lehetővé teszi az alkalmazás számára, hogy adatbázisból kérjen le és mentjen el "OrderItem" objektumokat, valamint lehetővé teszi az entitások közötti relációk kezelését és lekérdezését a TypeORM segítségével.

A képen szöveg látható

Automatikusan generált leírás

Ez az osztály egy PaymentMethod entitást definiál, amely a fizetési módszerek adatait tárolja a webshop alkalmazásban. Az @Entity() dekorátorral jelöljük az osztályt entitásként, amelyet az adatbázisban tárolunk.

A PaymentMethod entitás tartalmaz két mezőt: az első a fizetési mód azonosítóját, a második pedig a fizetési mód nevét tartalmazza. A @PrimaryGeneratedColumn() dekorátorral jelöljük az elsődleges kulcsot, amely automatikusan növekszik minden új rekord hozzáadásakor.

A PaymentMethod entitásban definiálunk egy kapcsolatot az Order entitással, amely a @OneToMany() dekorátorral van jelölve. Ez azt jelenti, hogy egy fizetési módhoz több rendelés is kapcsolódhat. Az Order entitásban definiáltuk ezt a kapcsolatot a @ManyToOne() dekorátorral, amely azt jelöli, hogy egy rendeléshez csak egy fizetési mód tartozik. A két entitás közötti kapcsolat az Order entitásban a paymentMethod mezőn keresztül van definiálva.

A képen szöveg látható

Automatikusan generált leírás

Ez az osztály egy ShippingMethod entitás, amely egy adatbázis táblát reprezentál. Az @Entity() dekorátor jelzi, hogy ez az osztály egy TypeORM entitás.

Az osztály attribútumai (id, name és price) megfelelnek az adatbázis tábla oszlopainak, amelyeket az @Column() dekorátorral jelöltek meg. Az @PrimaryGeneratedColumn() dekorátor pedig azt jelzi, hogy ez az oszlop az elsődleges kulcs, amely automatikusan generálódik.

Az orders attribútum egy reláció, amely az Order entitással van definiálva, és az @OneToMany() dekorátor jelzi, hogy egy adott szállítási módhoz több rendelés tartozhat. Az orders attribútummal kapcsolatos lekérdezések segítségével elérhetjük a ShippingMethod entitáshoz tartozó összes rendelést.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy "Category" nevű entitást definiál, ami egy adatbázis táblát reprezentál. Az adatbázisban a "Category" tábla az alábbi oszlopokkal rendelkezik:

* "id": egyedi azonosító, amely az entitásokat különbözteti meg egymástól.
* "name": kategória neve.

Az entitás TypeScriptben írt osztályként van megvalósítva, amelynek az osztály neve megegyezik az adatbázis tábla nevével. Az osztály az "typeorm" modul által biztosított "BaseEntity" osztályból származik, amely tartalmazza azokat az alapvető metódusokat és dekorátorokat, amelyekkel az entitásokat az adatbázishoz lehet kapcsolni. Az osztály további dekorátorokat is tartalmaz, például a "@PrimaryGeneratedColumn()" dekorátort az "id" oszlop számára, és az "@OneToMany()" dekorátort a "Product" entitásokra vonatkozó kapcsolathoz.

A képen szöveg látható

Automatikusan generált leírás  
  
Ez az osztály egy termék reprezentációjáért felelős az adatbázisban egy TypeORM alapú Node.js alkalmazásban. A @Entity() dekorátor jelzi, hogy ez egy adatbázis entitás.

A Product osztály tartalmazza a termék adatait, mint például a name (név), desc (leírás), imageUrl (kép URL-ek), price (ár), inactive (inaktív termék-e) és popular (népszerű termék-e). Az osztály tartalmaz továbbá relációkat a Category (kategória), Stock (készlet), ShoppingCartItem (kosár tétel), OrderItem (rendelés tétel) és Like (lájk) entitásokkal.

Az @OneToMany dekorátorok jelzik, hogy az adott entitás több kapcsolódó entitással rendelkezik, például egy terméknek lehet több raktári készlete (stocks), több kosárban is lehet (cartItems), vagy több rendelési tételben is szerepelhet (orderItems).

Az @ManyToOne dekorátorok pedig a másik irányú kapcsolatot jelölik, például egy termék csak egy kategóriához tartozik (category).

Az @Column dekorátorok a termék adatainak adatbázisbeli tárolását jelölik, mint például a name, desc, imageUrl vagy price.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy Release nevű entitás létrehozására szolgál, amely tárolja egy kiadás (például termékkiadás) adatait. Az entitás az @Entity() dekorátorral van ellátva, ami jelzi, hogy az adatbázisban egy táblához tartozik.

Az entitásnak vannak mezői, amelyek az adatbázis táblájának oszlopai lesznek. Az @PrimaryGeneratedColumn() dekorátor az elsődleges kulcsot jelzi, és azt jelzi, hogy az adatbázis létrehozza az id mezőt és automatikusan generálja az értékeit.

A mezők közé tartozik a kiadás neve (name), a kiadás dátuma (releaseDate), a leírás (desc) és az első négy kép URL-je (imageUrl1, imageUrl2, imageUrl3, imageUrl4). A nullable opcióval lehetővé teszi, hogy az URL-ek üresek legyenek.

A Release entitás nem tartalmaz kapcsolatokat más entitásokkal, de az @Entity() dekorátor miatt könnyen csatlakoztatható más entitásokhoz az adatbázisban.

A képen szöveg látható

Automatikusan generált leírás

Ez az osztály az áru méretét reprezentálja. A @Entity() dekorátor jelzi, hogy ez az osztály egy adatbázis entitás, és a neve "Size" lesz. A @PrimaryGeneratedColumn() dekorátor jelzi, hogy az "id" oszlop az elsődleges kulcs, amely automatikusan generált értékeket tartalmaz. Az @Column() dekorátor használata után a következő mező egy oszlop a táblában, és annak típusa a dekorátor által meghatározott. A @OneToMany() dekorátor jelzi, hogy az osztály egy kapcsolatot tartalmaz egy másik táblával, és a kapcsolat típusa 1-N (egy méretnek több raktárkészlete és rendelése lehet).

A képen szöveg látható

Automatikusan generált leírás

Ez az osztály egy raktárkészletet reprezentál, amely a termékekhez és méretekhez van hozzárendelve. Az osztályhoz tartoznak a következő tulajdonságok:

* id: Az adatbázisban tárolt raktárkészlet egyedi azonosítója.
* inStock: A raktárkészlet mennyisége a megadott méretű termékből. Ez egy getter/setter, amely a inStock értékének beállításakor ellenőrzi, hogy a raktárkészlet sosem lehet negatív értékű.
* size: A méret, amelyhez a raktárkészlet hozzá van rendelve. Ez egy Size típusú objektum.
* sizeId: Az adatbázisban tárolt méret egyedi azonosítója.
* product: A termék, amelyhez a raktárkészlet hozzá van rendelve. Ez egy Product típusú objektum.
* productId: Az adatbázisban tárolt termék egyedi azonosítója.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy "Address" nevű entitást definiál, amely egy cím leírását tartalmazza. Az entitás mezői a következők:

* "id": az entitás azonosítója
* "streetAddress": az utcanevek és házszám leírása
* "city": a város neve
* "state": az állam neve vagy rövidítése
* "postalCode": az irányítószám

Az @Entity() dekorátorral az entitás definíciója jelölhető ki, és a @PrimaryGeneratedColumn() dekorátorral az entitás azonosító mezője állítható be. Az @Column() dekorátorral az entitás mezői határozhatók meg.

A képen szöveg látható

Automatikusan generált leírás

A kódrészlet egy TypeORM entitást definiál Like néven. A @Entity() dekorátorral jelölt osztály egy adatbázis táblát reprezentál, amely az adatbázisban tárolja a felhasználók által kedvelt termékeket.

Az entitás a következő mezőket tartalmazza:

* id: Az entitás egyedi azonosítója, ami egy automatikusan generált elsődleges kulcs.
* user: Az a felhasználó, aki kedvelte a terméket. Egy felhasználóhoz több kedvelt termék is tartozhat.
* userId: Az a felhasználó azonosítója, aki kedvelte a terméket.
* product: A kedvelt termék, amelyet a felhasználó jelölt meg. Egy termékhez több felhasználó is kedvelheti.
* productId: A kedvelt termék azonosítója.

A @ManyToOne() dekorátorral ellátott mezők kapcsolódnak más entitásokhoz, és az adatbázisban idegen kulcsként jelennek meg. Az entitás egyediségét a id mező biztosítja, amely elsődleges kulcsként szolgál az adatbázisban.

A képen szöveg látható

Automatikusan generált leírás

A fenti kód egy TypeORM által generált "User" entitást definiál TypeScript-ben. Az "@Entity()" dekorátor jelzi a TypeORM számára, hogy ez az osztály egy adatbázis entitás, és társítja azt a "users" táblával az adatbázisban.

Az entitás négy mezőt tartalmaz: "id", "username", "email" és "password". Az "@PrimaryGeneratedColumn()" dekorátor jelzi, hogy az "id" mező az elsődleges kulcs, amelyet az adatbázis automatikusan generál.

Az "orders" mező egy olyan kapcsolatot definiál, amely az "Order" entitással van, és a kapcsolat típusa egy az többhöz. Az "@OneToMany()" dekorátor jelzi a TypeORM számára, hogy ez egy egy az többhöz kapcsolat, és az "Order" entitás az a kapcsolat másik vége. A kapcsolatot az adatbázisban az "orders" táblával valósítjuk meg.

Az "cartItems" mező egy olyan kapcsolatot definiál, amely a "ShoppingCartItem" entitással van, és a kapcsolat típusa egy az többhöz. Az "@OneToMany()" dekorátor jelzi a TypeORM számára, hogy ez egy egy az többhöz kapcsolat, és a "ShoppingCartItem" entitás az a kapcsolat másik vége. A kapcsolatot az adatbázisban az "shopping\_cart\_items" táblával valósítjuk meg.

# Dto-k:

A képen szöveg látható

Automatikusan generált leírás

Ez egy típusdefiníció, amely egy Address nevű osztályhoz kapcsolódó adattípusokat határoz meg. Az AddressDto osztály tagjai (streetAddress, city, state, postalCode) megfelelnek egy cím adatainak, és azt jelzik, hogy egy címet lehet tárolni és kezelni ebben az adattípusban. Az "export default" kulcsszó azt jelzi, hogy az osztály publikus és más fájlokból importálható.

A képen szöveg látható

Automatikusan generált leírás

Ez a CartDto (Kosár adattranszfer objektum) nevű osztály egy adott termék kosárba helyezésére vonatkozó információkat tartalmazza. Az osztály három szám típusú adattagot tartalmaz: quantity (mennyiség), productId (termék azonosítója) és userId (felhasználó azonosítója). Emellett van még egy sizeId adattag is, ami az adott méret azonosítóját tartalmazza. Ezek az adatok általában egy felhasználó által kosárba helyezett termék adatait tartalmazzák, amelyeket az alkalmazás backendjének kliensei átadhatnak az adatok továbbítása céljából.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy LikeDto nevű osztályt exportál, amely az alábbi két adattagot tartalmazza:

* userId: A felhasználó azonosítója, aki kedvelt egy terméket.
* productId: A termék azonosítója, amelyet a felhasználó kedvelt.

A kódban szereplő nevek angolul vannak írva, ezért a magyar fordításban is az angol neveket használjuk.

A képen szöveg látható

Automatikusan generált leírás

Ez az OrderDto egy adatátviteli objektum (DTO), amely az Order entitás létrehozásához vagy módosításához szükséges adatokat tartalmazza. A DTO-k általában azért használatosak, hogy könnyebben átvihető formátumot biztosítsanak az entitásokkal kapcsolatos adatokhoz, például akkor, amikor a kliensoldali és szerveroldali alkalmazások között adatokat kell küldeni.

A OrderDto tartalmaz néhány adatot az rendeléssel kapcsolatban, beleértve a felhasználó azonosítóját (userId), a szállítási cím azonosítóját (addressId), valamint a szállítási és fizetési módot (shippingMethod és paymentMethod). Ezek az adatok azt mutatják, hogy milyen rendelést hajt végre a felhasználó.

A képen szöveg látható

Automatikusan generált leírás

Ez az OrderItemDto osztály egy adatátviteli objektum (DTO), amely az OrderItem entitásból származó adatokat tartalmazza. Az adatokat különböző rétegek közötti kommunikációra használják, például az adatbázisból a felhasználói felületre vagy a felhasználói felületről az üzleti logikába. Az OrderItemDto tartalmazza az alábbi adatokat:

* quantity: A termékek mennyisége az adott rendeléshez.
* productId: Azonosító, amely az adott termékre mutat, amelyet a felhasználó rendelt.
* userId: Azonosító, amely az adott felhasználóhoz kapcsolódik, aki a rendelést feladta.
* sizeId: Azonosító, amely az adott termék méretéhez kapcsolódik, amelyet a felhasználó rendelt.
* orderId: Azonosító, amely az adott rendeléshez kapcsolódik.

A képen szöveg látható

Automatikusan generált leírás

Ez a kód egy RegisterDto osztályt exportál, amelynek tagváltozói a felhasználói regisztráció során szükséges adatokat tartalmazzák.

* A username tagváltozó egy string típusú felhasználói nevet tartalmaz, amelynek hossza legalább 6 karakter kell legyen.
* Az email tagváltozó egy string típusú e-mail címet tartalmaz, amelynek nincsenek további validálási követelményei.
* A password tagváltozó egy string típusú jelszót tartalmaz, amelynek hosszának legalább 4, de legfeljebb 20 karakternek kell lennie. A jelszónak tartalmaznia kell legalább egy kisbetűt, egy nagybetűt, egy számot vagy speciális karaktert. A jelszó erősségét a Matches dekorátorral ellenőrizzük.
* Az identifier tagváltozó egy tetszőleges típusú azonosítót tartalmaz, amely lehetővé teszi azonosítóként való használatot a kódban.

A dekorátorok (@MinLength, @MaxLength, @Matches) segítségével validáljuk az adatokat. Az IsString dekorátor azt jelenti, hogy a password tagváltozó csak string típusú lehet. Az adatok validálása azt jelenti, hogy ellenőrizzük, hogy megfelelnek-e az előre definiált követelményeknek, és ha nem, akkor hibát dobunk vissza.