**Adatbázis bemutatása**

**JPA**

A Java Persistence API (JPA) a perszisztencia és az object-relational mappinghez (ORM) biztosít modszereket.

Hibernate: Objection-Relational(ORM)Framework amely implementálja a JPA-at.

ORM: Az object-relational mapper objektumorientált réteget biztosít a relációs adatbázisok és az objektumorientált programozási nyelvek között anélkül, hogy SQL-lekérdezéseket kellene írni.

**@Entity**: A perszisztens objektumokat entitásoknak is nevezik. A @Entity csatolása egy olyan osztályhoz, arról tájékoztatja a JPA-t, hogy ezt az osztályt és annak objektumait perzisztensnek kell tekinteni.

Mindegyik Entity @Entityvel van meg annotálva és a BaseEntityről van leszármazva.

**BaseEntity:** A BaseEntity egy @MappedSuperClass annotációval ellátott Entity. Az egyetlen fieldje az “id” nevű field, aminek be van allítva a „Sequence Generator” annotációval. Így adhat meg egy olyan Sequence-t, amely létezik az adatbázisban. A Sequence egy olyan adatbázis-objektum, amely lehetővé teszi értékek automatikus generálását és a szekvenciák optimálisan alkalmasak az egyedi kulcsértékek generálására.

Text

Description automatically generated

A JPA Annotációk amelyeket használtam egy entitásomban a következők: @Entity, @Id, @Table, @Column, @ManyToMany, @JoinTable, @ManyToOne. A következő ábra a Product osztályomat mutatja, amely több ilyen annotációt használ.

Text

Description automatically generated

A megfelelő adatbázis kiválasztása a projekthez nagyon fontos, néhány szempont, amit a keresés során figyelembe vettem, a modernitás, az adatintegritás és a biztonság. Egy kis utána nézés után leszűkítettem a lehetőségeimet kettő valásztásra, a Postgresql és Mysql-re.

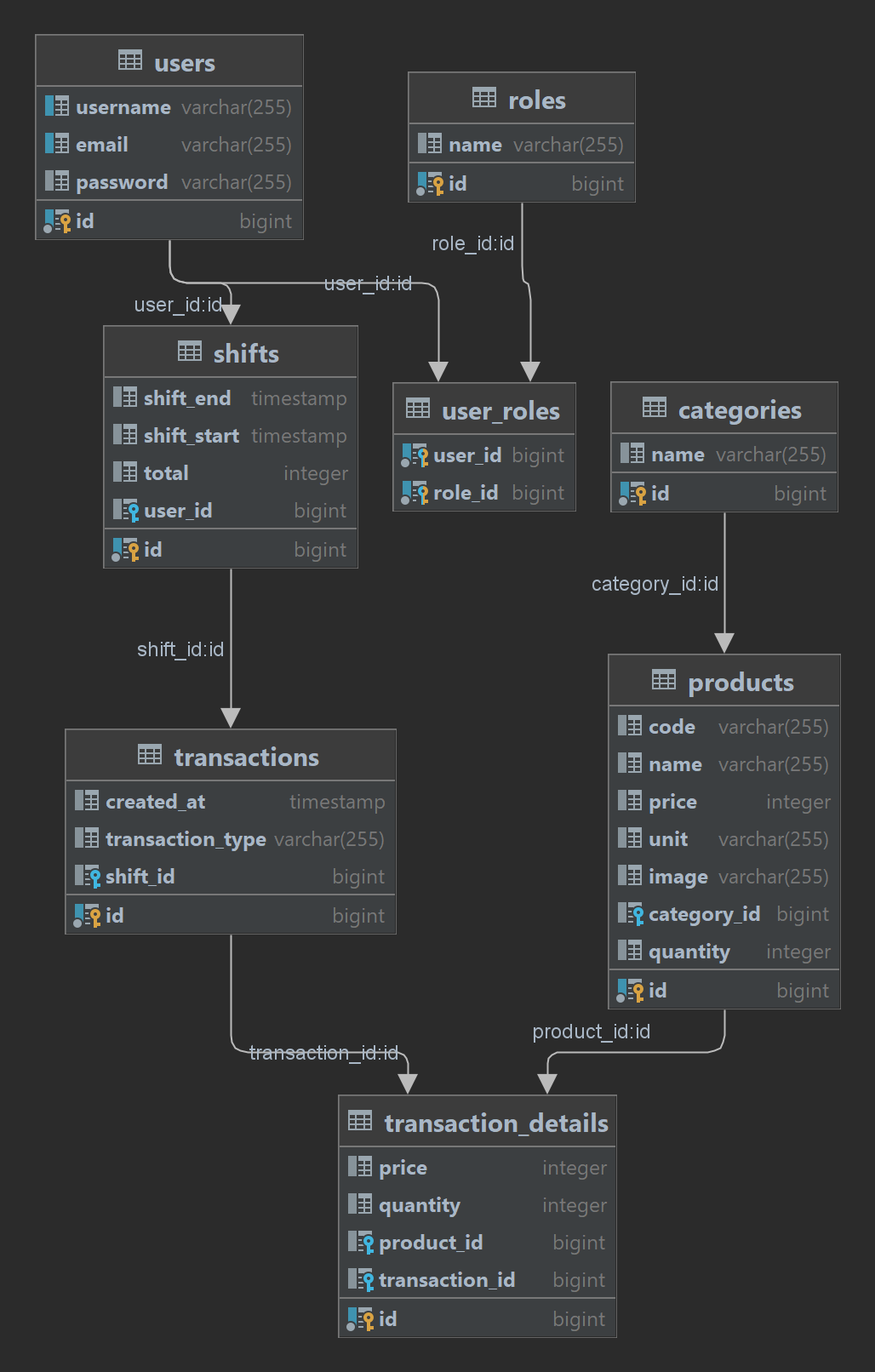
Végül úgy döntöttem, hogy a Postgresql-t használom adatbázisként, mindkét adatbázis rendkívül hasonló minden tulajdonságában, de a Postgresql sokkal jobb volt az konkurrencia kezelésében.

**Postgresql**

a PostgreSQL egy ingyenes és open-source relációs adatbázis-kezelő rendszer, ami jelenleg második legnépszerűbb adatbázis-kezelő rendszer. Az elmúlt néhány évben a népszerűségének ugrásszerű növekedése azt sejteti, hogy a Postgresql valószínű megelőzi a Mysql-t.

Ami számomra előnyok Postgresről:

* Adattípusok: A PostgreSQL támogatja az összes számomra szükséges adattípust.
* Adatintegritás: Postgres biztosítja az adatok integritását a „constraint”-ek bevezetésével
* Teljesítmény: A PostgreSQL közismerten gyorsabb a hatalmas adathalmazok, bonyolult lekérdezések és írási-olvasási műveletek kezelésében.

****

Az ábrán látható relációs diagramot elkészítéséhez a DataGrip programot használtam. Ahol „one to many, many to many” kapcsolat van, ott az id-n keresztül kapcsolódnak egymáshoz a táblák.

**Az adatbázis tábláinak bemutatása**

* categories: A termékkategóriák tábla, a termékek típusai itt vannak tárolva.
* products: A Termékek tábla tárolja a leltárban lévő termékeket és azok mennyiségét.
* roles: a roles tábla tárolja azokat a szerepeket, amelyekkel egy felhasználó rendelkezhet.
* shifts: A műszakok tábla tárolja az egyes felhasználók műszakjait.
* transaction\_details: A tranzakció details tábla egy tranzakción belüli bejegyzést tárol. tartalmazza a terméket, annak mennyiségét, valamint az eladási/bevételi árát. Tárolja annak a tranzakciónak az azonosítóját is, amelyhez tartozik.
* transactions: Tranzakció típusa, valamint a létrehozás dátuma.
* user: felhasználónév, e-mail cím és titkosított jelszó.
* user\_roles: a felhasználó és a szerepek tábla közötti “join table” amik “many to many” típusúak.

Graphical user interface

Description automatically generated with low confidence

**Bevezetés**

**A dolgozatban bemutatott program**

A szakdolgozatom célja egy webes alkalmazás létrehozása volt vállalkozások számára. Miután 2 évig dolgoztam az customer-service területén különböző munkakörökben. Azt tapasztaltam, hogy minden vállalkozásnak valamilyen módon el kellett végeznie ezt a kettő tevékenységet:

1. A pénztár gépekben lévő pénz megszámolása a nap végén.

2. leltárellenőrzést végezni a termékeken idő közönként.

Ez volt az oka annak, hogy kitaláltam egy olyan webapplikációt, amelyen nyomon követhető az összes tranzakció ami megtörtént. Végül is fontos egy vállalat számára, hogy megbizonyosodjon arról, hogy nem veszít pénzt a pénztárban, vagy nem hiányoznak termékek. Azért is fontos, hogy tisztában legyenek a készletekkel, hogy tudják, mikor kell többet rendelniük egy bizonyos termékből, amely hamarosan elfogy.

Ebben az alkalmazásban két fő role (szerep) van, van az adminisztrátori (ADMIN) role, amelyet az üzletfelügyelőknek/vezetőknek szántam és az általános (user) role amit az alkalmazottaknak szántam. Először is a két role teljesen eltérő nézeteket lát a bejelentkezés után. A dolgozó egy olyan oldalra kerül, ahol csak tranzakciókat tud hozzá adni a saját műszakjukhoz, valamint befejezheti a műszakját, amikor végez a munkával. Az Admin role viszont sokkal több oldalhoz és funkcióhoz is van hozzáférése. Néhány ezek közül a leltár megjelenítése, a leltár szerkesztése és még sok más, amiről részletesen beszámolok egy későbbi részben a szakdolgoztban.

**Cél közönség**

Ezt a webes alkalmazást bárki használhatja, beleértve a kis- és nagyvállalatokat is. A weboldal megnyitható egy böngészővel és reszponzív, ami azt jelenti, hogy a méretváltozások a weboldal megjelenését is megváltoztatja. Az reszponszivitásnak köszönhetően telefonon is jól mutat az oldal.

Ahhoz, hogy bármit elérjen a weboldalon, a felhasználónak be kell jelentkeznie, ha az URL-en keresztül bármit megpróbál elérni, miközben nincs bejelentkezve, a felhasználót a frontendbe kódolt AuthGuard miatt visszairányítja a bejelentkezési oldalra.

**II. Felhasználói dokumentáció**

A weboldal három részre van osztva. A bejelentkezés/regisztrációs oldalak, a USER role által elérhető oldal és az ADMIN role által elérhető oldalak. A Bejelenkezés és regisztrációs oldalak akkor jelennek meg ha nincsen bejelentkezve a felhasználó. A Cashier oldal vagyis a pénztáros oldal egy olyan oldal amit egy dolgozó tud elérni. Az ADMIN role felhasználóknak elérhetőek az admin oldalak amin képes szerkeszteni minden adatot a rendszeren belül.

**II. 1. Bejelentkezés & regisztráció oldalak**

**Login:** Amikor a felhasználó először nyitja meg az oldalt, ide irányítja át a rendszer. Ezen kívül három módja van annak, hogy a felhasználó átirányításra kerüljön ide:

1. Kijelentkezés miután bejelentkezet egy felhasználó.

2. A műszak befejezése alkalmazott felhasználóként.

3. Sikeres regisztráció után.

A helyes felhasználónév és jelszó megadása után a beküldés gombra való kattintás után a felhasználó a rolejának (USER, ADMIN) megfelelően átirányításra kerül az adminisztrációs vagy a pénztár oldalra. Graphical user interface, application, Teams

Description automatically generated

**Register:**

Ha a felhasználó még nem regisztrált, akkor a bejelentkezési oldalon a regisztráció gombra kattintva juthat el ide. A sikeres regisztrációhoz a felhasználónak meg kell adnia egy egyedi felhasználónevet és e-mail címet, valamint egy jelszót, majd a regisztráció gombra kell kattintania. Ezt követően a felhasználó a bejelentkezés oldalra kerül és bejelentkezhet a létrehozott felhasználónevével és jelszavával.

Graphical user interface, application, Teams

Description automatically generated

**II. 2. User oldal**

**Cashier:**

Ezt az oldalt látja a dolgozó a bejelentkezés utan. A munkavállaló műszakja automatikusan elindul az oldal megnyitása után, és a munkavállaló itt adhatja hozzá a műszakja során végzett összes eladást. Miután végzett, befejezheti a műszakot, ami után megnézheti a műszak alatt elért összértékét.

Graphical user interface, application

Description automatically generated

A fenti ábrán egy tranzakció példája látható, ahhoz, hogy termékeket adjon hozzá a tranzakcióhoz, az alkalmazottnak először be kell írnia egy érvényes terméket, amely létezik a rendszerben (az admin felhasználók képesek új termékeket hozzáadni, ha egy adott termék még nem létezik a rendszerben), majd hozzá kell adnia az összeget, végül hozzáadás gombra kell kattintania. Ezután a tranzakció megjelenik a táblázatban a kiszámított részösszeggel és tranzakcióösszeggel együtt. Azt a logikát, amellyel csak a terméktáblában szereplő termékeket ismeri fel, a dolgozat üzleti logikai részében fogom megemlíteni. Röviden az ok, amiért ez nem egy legördülő lista, az az, hogy lehetnek olyan esetek, amikor egy vállalkozás több ezer termék közül választhat, és az összes termék betöltése és a köztük való keresés nem hatékony megoldás.

Graphical user interface, application, Teams

Description automatically generated

A "Checkout" gombra kattintva a dolgozót megkérdezi, hogy biztosan befejezte-e az aktuális tranzakciót. Ha igennel válaszol, az aktuális tranzakciót elmenti a műszakjába (Az elmentett tranzakciót az admin megtekintheti a Transactions oldalon), a táblázat törlődik, és az oldal készen áll egy új tranzakció naplózására.

Graphical user interface, text, application

Description automatically generated

A "End Shift" gombbal a munkavállaló befejezheti a műszakját, és tájékoztatást kap a műszakja alatt eladott mennyiségről. A végösszeg 1900-ra jött ki, ami azt jelenti, hogy a műszakjuk alatt 200 forint értékű korábbi tranzakciót bejegyeztek a legutóbbi 1700 tranzakción felül.

**II. 3 Admin oldalak**

**Products**

**Graphical user interface, text, application

Description automatically generated**

Itt adhat hozzá, törölhet és frissíthet termékeket a leltárban az adminisztrátor. Valamint manuálisan megváltoztathatja a termék mennyiségét a leltárban, ami általában nem lehetséges mert az összeget csak az eladásokból vagy megrendelésekből származó tranzakciók (Sale, Receive) módosítják. Lehetőség van a termékek név szerinti keresésére a jobb felső sarokban a „Search”-nél.

**Product Dialogs**

**Graphical user interface, text, application

Description automatically generated**

Ezen az oldalon mind az új hozzáadása, mind a szerkesztés gombok dialógusablakkot hívnak elő. A fenti képen egy példa a termék létrehozása dialógusablakára, ahol minden mező ki van töltve. A bejegyzés elmentéséhez minden mezőt ki kell tölteni, az egyetlen opcionális mező a képfájl.

A termék létrehozása és a termék szerkesztése dialógusablakok valóban ugyanúgy néz ki. Csak kettő különbség van az, hogy a szerkesztés során a mezők a termék korábbi értékeivel lesznek előrefeltöltve és fent a dialog címe "Edit Product" lesz.

**Product Categories**

Termékkategóriák hozzáadása, szerkesztése, törlése. Az újonnan hozzáadott kategóriák, új termék létrehozásakor megjelennek kereséskor és választhatók lesznek. Lehetőség van a kategóriák név szerinti keresésére is.

Graphical user interface, text, application, email

Description automatically generated

**Product Category Dialogs**

**Graphical user interface, text, application, chat or text message

Description automatically generated**

A termékkategória mentéséhez vagy szerkesztéséhez kotelezo a Name mezőt kitölteni. A módosítások a beküldés után láthatóvá válnak a táblázatban.

**Transactions**

Ezen az oldalon lehetősége van új tranzakció létrehozására, amely egy új Transaction details oldalra vezet. A szerkesztés gomb viszont az adott tranzakcióhoz tartozó, Transaction details oldalára vezet. E két funkció mellett az adminisztrátor itt törölheti a tranzakciókat. Graphical user interface, text

Description automatically generated

**Transaction Details**

A Transaction Details oldal a tranzakcióban szereplő összes termék felsorolására szolgál. Ezen az oldalon hozzáadhat, szerkeszthet és törölhet termékeket egy tranzakcióból.

Graphical user interface, text, application

Description automatically generated

A fenti ábra azt az oldalt mutatja, amikor az adminisztrátor új tranzakciót hoz létre. A dátum és a tranzakció típusának (eladás, átvétel) kiválasztása után elkezdhetik a termékek hozzáadását a tranzakcióhoz.

**Transaction Details Dialog:**

Graphical user interface, application, Teams

Description automatically generated

A következő példa egy termék hozzáadására a tranzakcióhoz. A termék kiválasztása után az ár mező automatikusan kitöltődik, és a felhasználónak csak az eladott/átvett összeget kell hozzáadnia a folyamat befejezéséhez.

A screenshot of a computer

Description automatically generated

Amikor egy tranzakció szerkesztésére kerül sor, a létrehozás dátuma és a tranzakció típusa ki van kapcsolva, itt csak termékeket adhat hozzá, szerkeszthet vagy távolíthat el ebben a tranzakcióban.

**III. Fejlesztői Dokumentáció**

**A tervezett szoftver megvalósítása**

**III.2.1 A tervezett szoftver architektúrája**

**Liquibase:**

A Liquibase egy open-source projekt, amely segít az adatbázis-sémák változásainak nyomon követésében, verzió kontrollba és deployolásában. Azért kezdtem el használni a Liquibase-t, mert a szakmai gyakorlatom alatt a projekt, amelyen dolgoztam, szintén liquibase-t tartalmazott. A changeSets segítségével nagyon kényelmes volt nyomon követni az adatbázis tábláin végzett változtatásokat, valamint azt, hogy a projekt melyik fejlesztője melyik részt írta.

Emellett több betekintést adott a táblázatok és oszlopok manuális kézileg való létrehozására, valamint az idegenkulcsos kapcsolatok meghatározására, ahelyett hogy a JPA minden munkát elvégezne helyettem. Mivel minden táblázatomhoz van changelog fájlom, az összes táblázat megjelenítése túl hosszú lenne. Ezért csak két adatbázis changelog fájlt fogok megmutatni, a products táblázatot, amely "Many to One" kapcsolatot tartalmaz a product category-hoz. A második egy "Many to Many” kapcsolat join table a user tábla és és a roles tábla között, a user-roles table.

**Changelog-master**

A master changelog fájl konfigurációs fájlként működik, amely tartalmazza az összes többi changelogra való hivatkozást. Az alábbi kép a changelog-master.xml fájlom tartalma.

Text

Description automatically generated

**Products Table**

A create-products-table.xml a change-log file, amely létrehozza a “products” table-t. Itt két changeSet látható, az első "create\_products\_table" létrehozza a táblázat összes oszlopát. A második changeset beállítja az idegen kulcsot. Bár én vagyok az egyetlen, aki ezen a projekten dolgozik, a Szerző mező kötelező a ChangeSet-ben, emiatt mindkét szerzői mezőt a nevemmel töltöttem ki.

Text

Description automatically generated

**User-Roles Table**

Ez a táblázat csak két oszlopot tartalmaz, a user\_id és a role\_id, amelyeket az első changeSet-ben hoztam létre. A második changeSet-ben a foreign key constraintjeiket a megfelelő tábláikra állítottam be. A következő attribútumokat kell beállítani az foreign key constraint-ben: az eredeti tábla neve, az eredeti tábla oszlopneve, a hivatkozott tábla, és a hivatkozott oszlopneve. A korlátozás nevére általában a következőt használom: FK\_EREDETITABLA\_TO\_HIVATKOZOTTTABLA, ahol az FK a ForeignKey rövidítése. Text

Description automatically generated

**Spring Security**

Ebben a projektben a Spring Security használtam, a Spring Security egy olyan keretrendszer, amely különböző biztonsági funkciókat biztosít, mint például: hitelesítés, engedélyezés. . Az authentikációhoz két végpontja van az egyik a: „api/auth/signup” regisztrációhoz és „api/auth/signin” a bejelentkezéshez.

**JWT**

A JWT népszerű a hitelesítéshez és az információcseréhez. A JWT 3 részből áll: a Header, Payload és az Signature. A szerver az adatokat egy JSON webes tokenbe (JWT) kódolja és elküldi a clientnek, a client elmenti a JWT-t és onnantól a client minden kéréshez a headerben csatolva küldi a JWT-t.

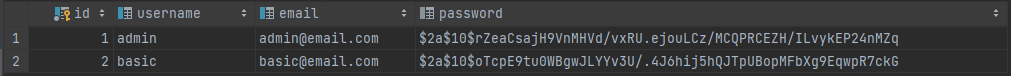
**A Spring Security Struktúrája.**

**Text

Description automatically generated with medium confidence**

**WebSecurityConfig:**

HttpSecurity konfigurációkat biztosít a cors, csrf, session management konfigurálásához. Szintén itt állítom be a jelszó kódolót, a userDetailsService-t, az AuthTokenFilter-t és sok más konfigurációt, mindezeket sorrendben fogom elmagyarázni.

* Userdetailsservice: A UserDetailsService a Spring Security keretrendszer egyik alapvető interfésze, amely a felhasználó autentikációra és jogosultságai lekérdezésére szolgál. Ennek az interfésznek csak egy loadUserByUsername() nevű metódusa van, amelyet a UserDetailsServiceImpl osztályál implementáltam.
* Password Encoder: a jelszó kódolóhoz a BcryptPasswordEncoder-t állítottam be. A bcrypt egy biztonságos jelszó-hashing funkció. Az alábbi képen példák láthatók a users adatbázisában tárolt titkosított jelszavakra. 
* AuthTokenFilter: Az AuthTokenFilter az Authorization header-ből a JWT kinyerésére és érvényesítésére szolgál, a "Bearer" előtag eltávolításával. Ezután megtalálja a felhasználót a JWT-ből és beállítja az aktuális userDetails-t a securityContextbe.
* AuthEntryPointJwt: Elkapja a unauthorized error-t és 401 hibakódot küld vissza.

**JwtUtils:**

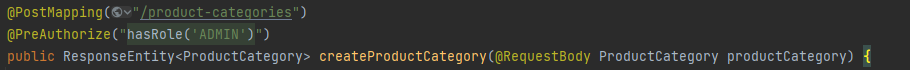
A jwtUtils három metódussal rendelkezik: JWT generálása, felhasználónév lekérdezése a JWT-ből és a JWT validálása. Ezek közül a getUsernameFromJwtToken és a ValidateJwtToken metódusokat használja az AuthTokenFilter.

**CorsConfiguration:**

A corsConfigurationSource segítségével globálisan be tudtam állítani a CORS-t az alkalmazásom számára. Amint az alábbi kódban látható, beállítottam a domain-t, ahonnan a kérések engedélyezve vannak, az engedélyezett HTTP-módszereket, valamint engedélyeztem a Authorization Header-t. Text

Description automatically generated

**@PreAuthorize:** A controller metódusokon használt preAuthorize annotation-t arra használom, hogy biztosítom arról, hogy a kéréseket küldő felhasználók a megfelelő Role-al rendelkeznek. Mivel a projektemben csak két Role van, a fő célja annak biztosítása, hogy a User Role felhasználók ne férhessenek hozzá az Admin Role felhasználói kérelmekhez. Természetesen a regisztrációs és bejelentkezési kérelmekre nincsenek ilyen korlátozások.

****

**Üzleti logika (Business Logic)**

**Java Spring**

A Spring Framework egy alkalmazási keretrendszer amely egy container-t biztosít ezt nevezzük Spring Application Contextnek. Ez hozza létre és kezeli az alkalmazáskomponenseket, vagyis a (beanseket), amelyek összekapcsolódnak, hogy a teljes alkalmazást alkossák.

Dependency Injection: A dependency injektálás Spring-el a Spring Beanek összekapcsolásából áll. Egy dependency injection alkalmazás egy különálló konténerre támaszkodik, ez a konténer hozza létre és karban tartja az összes komponenst. Ez a konténer injektálja a komponenseket az őket igénylő beanekbe.  
A következő részben a használt Spring Bean Annotation-öket mutatom be.

* @Repository: A @Repository annotációval megjegyzett osztályok az alkalmazás adatbázis-hozzáférési rétegét képviselik.
* @Service: Az alkalmazás üzleti logikája általában tartalmazó osztályok.
* @Controller: Ez az annotáció arra szolgál, hogy meg mondja a Spring-nek, hogy ez az osztály Controllerként szólgál.
* @Configuration: A @Configuration annotáció azt jelzi, hogy egy osztály egy vagy több @Bean metódust deklarál, és azt a Spring konténer feldolgozhatja.

**Lombok**

A projektemben a Lombok-ot használom a getterek, setterek és konstruktorok generálására. A @Data annotáció az összes adattagra generál getter, és settereket. A @RequiredArgsConstructor-t használom a Spring constructor injektáláshoz. Az alábbi kép egy példát mutat arra, hogy a TransactionServiceImpl, RequiredArgsConstructor annotációval injektálja a TransactionRepository-t és a TransactionDetailsRepository-t.

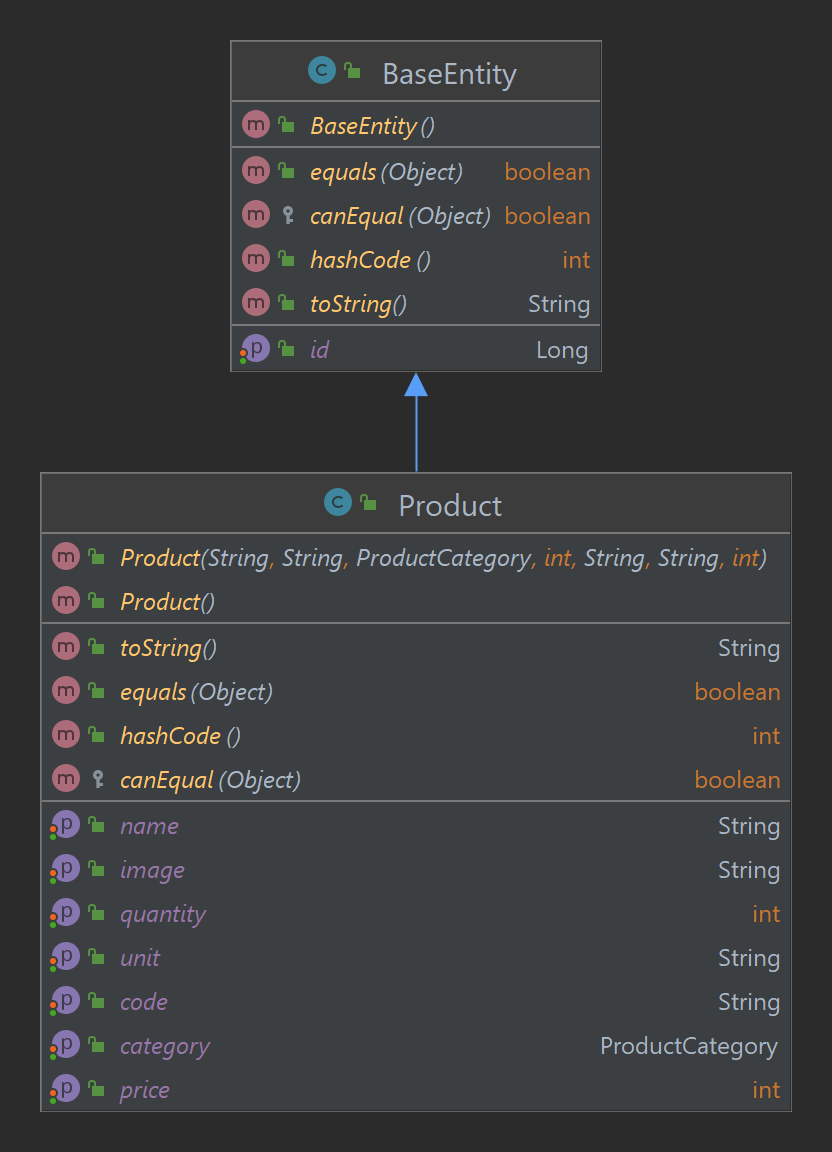
Text

Description automatically generated

**Product Üzleti Logikája**

Ebben a részben bemutatom a Termékek üzleti logikáját a kezdettől (Modell entitás) a befejezésig (Controller). Ez 4 részre lesz bontva: Model, Repository, Service és Controller.

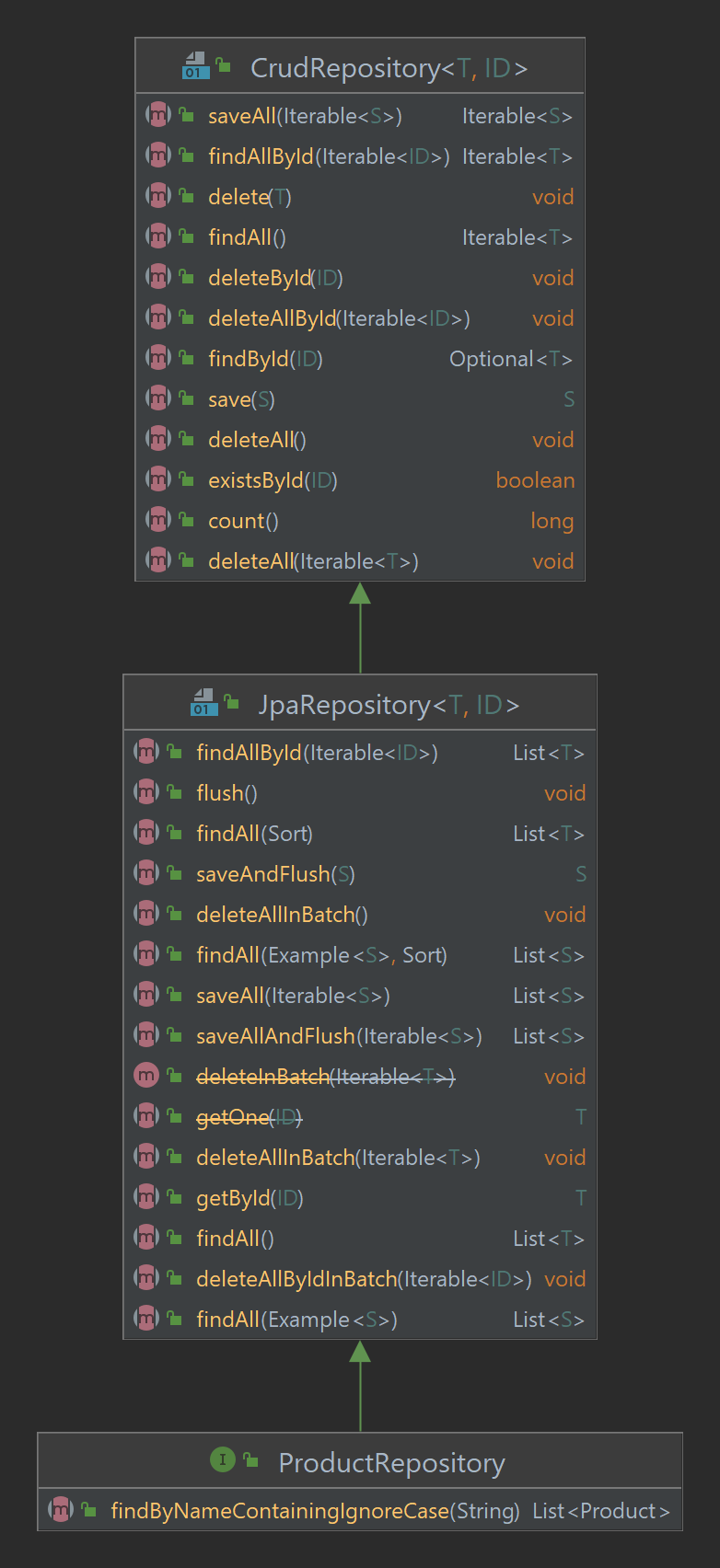
**Model/Entity:** A termék entitás tartalmazza a kód, név, kategória, ár, egység, kép és mennyiség propertyket.



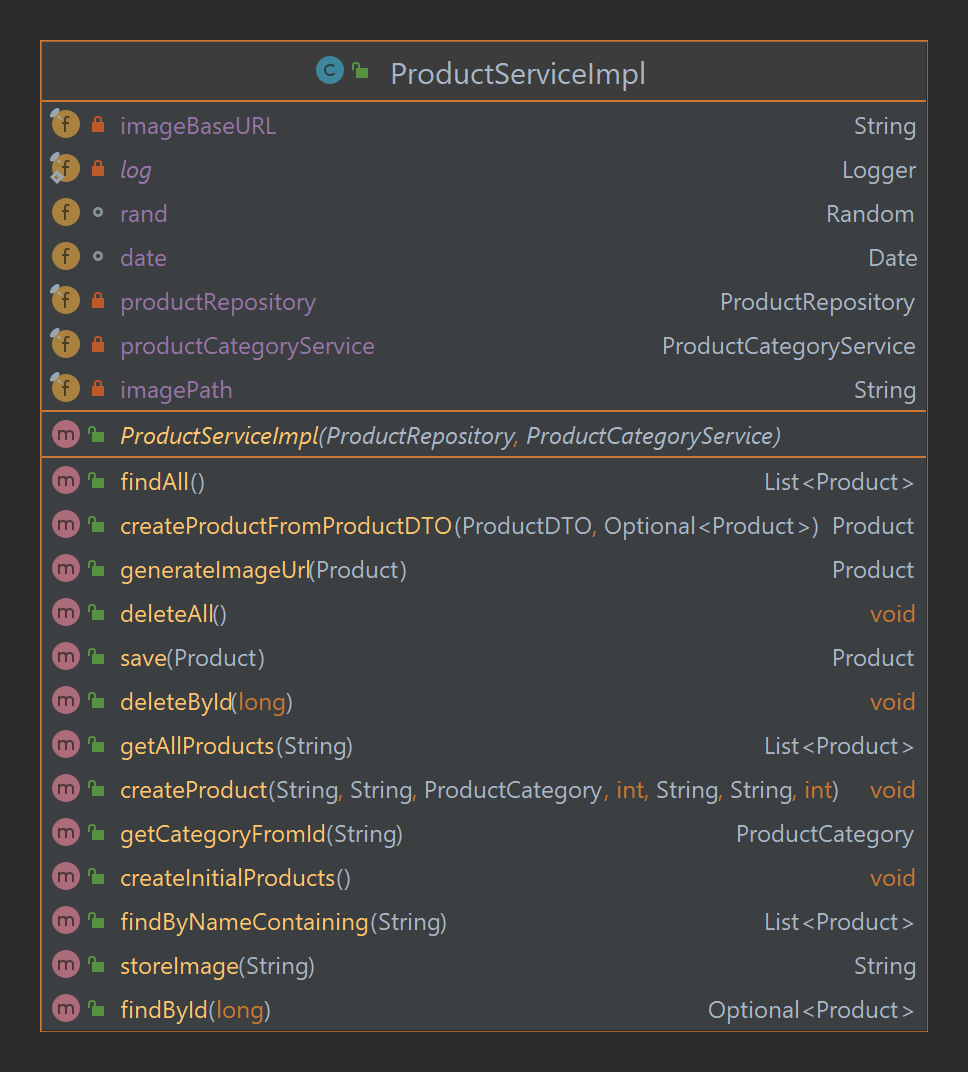
Amint látható, a képet egy String formájában tárolom, erről részletesebben a dolgozat egy későbbi részében fogok beszélni.

Létezik a kategória tulajdonság is, amelynek egy ManyToOne kapcsolata van a ProductCategory entitáshoz. A FetchType.EAGER-t használom, ami azt jelenti, hogy ez a mező az összes többi mezővel együtt töltődik be. A FetchType.EAGER-t használom, ami azt jelenti, hogy ez a mező az összes többi mezővel együtt töltődik be. A @JoinColumn annotáció alatt megadom az oszlop nevét az aktuális táblázatban, valamint a hivatkozott oszlop nevét a ProductCategory táblában, amely az id oszlop.

**Repository:** A Spring Repository egy olyan interfész, amely lehetővé teszi a fejlesztő számára, hogy metódusokat hozzon létre a querykhez, a Spring ezután a metódus neve és paraméterei alapján JPQL lekérdezéseket generál a futás során. Ez azt jelenti, hogy nem kell implementációkat írni a lekérdezésekhez.A ProductRepository a JpaRepositoryból származik le, a JpaRepository pedig a Crud Repository-t extendeli. Ez az osztály tartalmazza az összes alapvető tárolási metódust amire szükségem volt a parent osztályokon keresztül, mint például a save, deleteAll, deleteById, findById és egyéb metódusok. Az egyetlen egyéni módszer, amelyre szükségem volt, a findByNameContainingIgnoreCase() módszer volt, amelyet a termék keresésére használok az űrlapjaimban.



**Service:** A Service réteg a köztes szoftver a Controller és a repository között. A Service réteg célja, hogy összegyűjtse az adatokat a Controllertől, elvégezze az érvényesítést, az üzleti logikát, valamint használja a korábban említett Repository-t az adatmanipulációhoz. Egy interfészből (ProductService) és egy implementációból (ProductServiceImpl) osztályból áll. Az implementáló osztály @Service annotációval van ellátva, ezzel be tudom injektálni az interfész osztályt ahol szügségem van rá (pl. ProductController).



Ez az osztály tartalmazza az összes Repository metódust. Ezen felül itt található a createInitialProducts metódus, amely létrehoz néhány terméket, amikor az alkalmazás elindul, valamint a képtárolás logikáját.

**Képek:**

**StoreImage(String):** Először a frontend küld egy base64 stringet a backendnek, majd ebben a Service metódusban generálok egy egyedi képnevet. Az egyedi képnév a date.getTime() (1970. január 1., 00:00:00 GTM óta eltelt milliszekundumok számát adja vissza) és egy 0 és 10000 közötti véletlen egész szám kombinációja. Végül dekódolom a base64-et egy bájt tömbre és tárolom a fájlrendszerben a most generált a névvel, az adatbázisban viszont csak a kép nevét tárolom.

Graphical user interface, text, chat or text message

Description automatically generated

A fenti kép véletlenszerűen generált képneveket mutat.

A képek lekérése az Image controlleren keresztül történik, egy get mappingen keresztül, amelynek az elérési útvonala "/images/{imageName}". Tehát amikor elküldöm a termékeket a frontendnek, a **GenerateURL()** metódus a képmezőt átírja a következőre: (baseURL) "//localhost:8080/images/" + (adatbázisban tárolt képneve) 1649159079971449.jpg (teljes: "//localhost:8080/images/1649159079971449.jpg”), így a frontend a Controlleren keresztül kapja meg a képet.

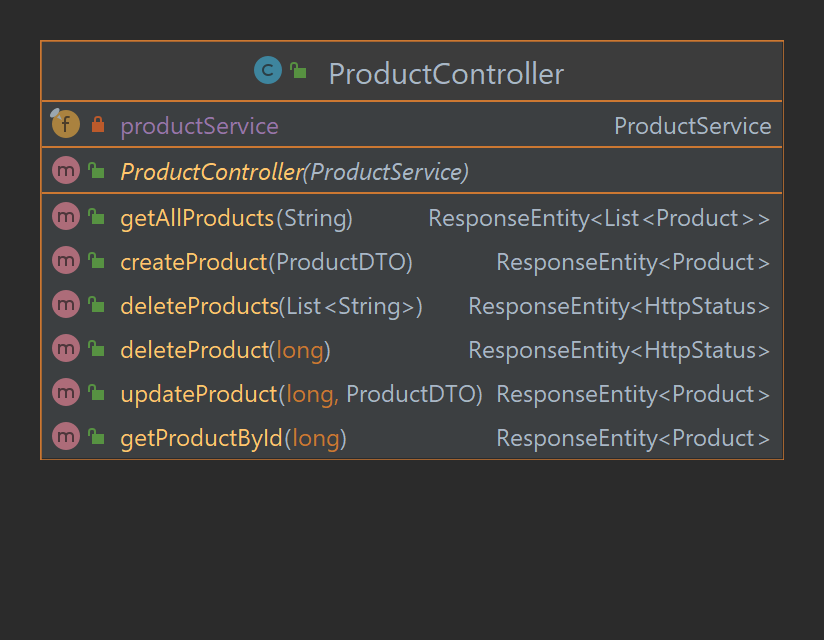
Text

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

a fenti képen az url egy elmentett fejhallgató képet jelenít meg.

**Controller: **

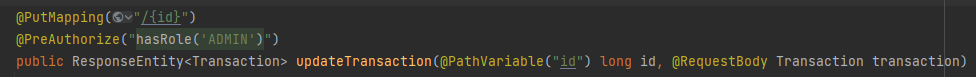
A ProductController a @RestController annotációval van annotálva. A @RestController annotáció a restful web services keszítésére szolgál, és lehetővé teszi, hogy az osztály kezelje a client által küldött kéréseket. A @RequestMapping annotációval jelölöm azt, hogy ez az osztály kezeli a /productshoz irányuló kéréseket. Az 5 HTTP kérés metódusok is annotációval vannak megjelölve: @GetMapping, @PostMapping, @PutMapping, @DeleteMapping and @PatchMapping.

Text

Description automatically generated

**@RequestBody:** A @RequestBody annotáció azt jelzi, hogy a Springnek a kérés body-t egy objektummá kell szerializálnia.

**@PathVariable:** A @PathVariable annotáció azt jelzi, hogy a metódus paraméterét egy URI template variable-hez kell kötni.

A payloadban találhatóak a Request és Response objektumok osztályai (DTO, Response).

**Felhasználói felület(User Interface)**

**Angular:** Az Angular egy TypeScript-re épülő fejlesztési platform. Az Angular alkalmazások single page applicationök és modulárisak, az Angularnak saját moduláris rendszere van, az: NgModules. Ez a keretrendszer számos hasznos eszközt nyújt egy webes alkalmazás létrehozásához. A következő részben bemutatom az összes általam használt eszközt.

**App Module:** Az AppModule a gyökérmodul ami az app.module.ts fájlban van. Ez a fajl importálja a más NgModulokból exportált funkciókat és exportálják ezeket más NgModulokhoz, hogy tudják használni.

**AuthInterceptor:** Az AuthInterceptor fájl egy HttpInterceptor. A fő funkciója az, hogy lekérje a token-t a Sessionstorage-ból, és hozzácsatolja a headerhez az összes kérésnél.

**AppRouting:** Az AppRoutinglehetővé teszi a navigációt azáltal, hogy a böngésző URL-címét a nézetváltásra vonatkozó utasításként értelmezi.

**TokenStorageService:** Ez a szolgáltatás a JWT token mentésére és a sessionStorage-ból való lekérdezésére szolgál. A felhasználói információk karbantartására is segít.