JUNIOR BACKEND PROGRAMOZÓI ZÁRÓVIZSGA

projekt specifikáció az Értékesítési és beszerzési rendszerről

Cél:

Ez a projekt egy egyszerű értékesítési rendszert valósít meg, amely a beszerzési és értékesítési folyamatokat kezeli.

A rendszer képes termékek beszerzésére, a raktárkészlet nyomon követésére, valamint a termékek értékesítésére.

Mottó:

A feladatcsoportban szereplő megoldási szükségletek 99%-át az elmúlt közel egy évben biztosan vettük egyszer, a 90% -át pedig inkább sokszor, semmint egyszer.

A feladat 100% -ban megoldható.

Ha valami nem működik akkor használd az "eszed" és lépd át úgy, ahogy a legkevesebb mértékig sérted a specikált tartalmat. Például a beszerzes.txt feltölthető a "gyartmanylista.adatfajl" tartalommal, stb...

Fontos:

1. Lehetőleg ne térj el a Specifikációtól.
   Ha mégis el kell térned (mert csak úgy megy) tedd azt! Inkább oldd meg a magad módján, semmint sehogyan sem!
2. A legfontosabb, hogy működjön, aminek működnie kell!
3. A hibák koncepciózusan lettek utólag elhelyezve a kódban!
   Azaz a kód tökéletesen működik (a Specifikáció ezen pillanatában legalábbis és egy telepített környezetben egészen biztosan!)
4. A teljes kódban végig dolgozik a **`ProjectAdmin`** osztály! Ezt ne bolygasd és ne babrálj vele, mert a vizsgád érvénytelenítését hozhatja. A projekt gyökérbe dolgozik és ha github-ra teszed fel viszi majd magával az admin fájlt is, ha más módon, akkor is becsomagolódik. Ez leegyszerűsíti a kód ellenőrzését, mert minden fontos működést kilogol csak össze kell számolnom megvan-e mindegyik és ha más telepített környezetben csináltad (és esetleg nálam nem fut, vagy csak átalakításokkal tenné) akkor is visszaigazolja a futásközbeni eseteket, mindent ami nekem az értéelés egyik részéhez kell.
5. Ha kérdésed van tedd fel a kérdést és amint tudunk válaszolunk. Az utolsó két szombati napon a vizsgával kapcsolatos kérdések mindegyikére választ adok, amennyiben az nem a megoldás iránti kérelem.

Eszközrendszerünk:

A vizsgafeladat egy JAVA Maven-projekt, Springboot-Starter 3.3.2 -es verzióval és JAVA-21 kódkészlettel!
Ettől - ha el szeretnél térni, ez nem probléma, de kérlek majd jelezd, mert az ellenőrzés ez esetben másik telepített környezetet igényel majd!

Részleteiben:

- Jdk-21.0.1 verzió (Java version: 21.0.1, vendor: Oracle Corporation, OpenJDK 64-Bit Server VM 21.0.1+12-29)
- Apache Maven 3.9.3
- Default locale: hu_HU, platform encoding: UTF-8
- **Product Version:** Apache NetBeans IDE 22
- **Runtime:** OpenJDK Runtime Environment 21.0.1+12-29
- **System:** Windows 10 version 10.0 running on amd64; UTF-8; hu_HU (nb)

  (a fenti – saját- adataidat a NetBeans.Help.About, illetve pl egy mappa, jobb klikk, "Open Git-Bash here" paranccsal nyitott shell ablakban kiadott: "mvn -v" tudod lekérdezni)
- Spring Data JDBC
- Tomcat (initialized with port 8080 http)
- H2 memória-adatbáziskezelő
- Hibernate ORM (objektum-db leképező mepper) core version 6.5.2.Final
- Spring-JPA EntityManagerFactory for persistence unit

A POM.XML adott és nem kell függőségeket sem bővíteni, sem szűkíteni., de amennyiben a Te környezetedben nem működne helyesen, akkor kérlek vizsgáld meg, esetleg ürítsd a lokális Maven tárolódat (C:\Users\User\.m2\repository), és próbáld újra buildelni a projekted. Ha ezt követően sem megy, akkor építsd újra a projekted!

Ha újra kell építened, akkor egyszerűen menj a "https://sping.io/" felületre és hozdd létre a projekted, a POM.XML -ben látható alapvetésekkel. Ha nem húzol be mindent a spring.io -nál, az sem gond. A tanultak szerint kicsomagolod, beállítod a projekt properties-t (compiler opciók, main-class, stb) és buildelsz, majd utána bekopizod az új POM.XML -ben még nem található egyéb függőségeket. Ilyenkor persze lehet, hogy verzió ütközések állnak elő, de kezeld rugalmasan. Előbb próbáld a verziókat törölni, majd ha nem hagyja, akkor igazítsd a SPRING fő verziódhoz (ami itt pl ez):

```
 <parent>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-parent</artifactId>
     <version>3.3.2</version>
```

<relativePath/>

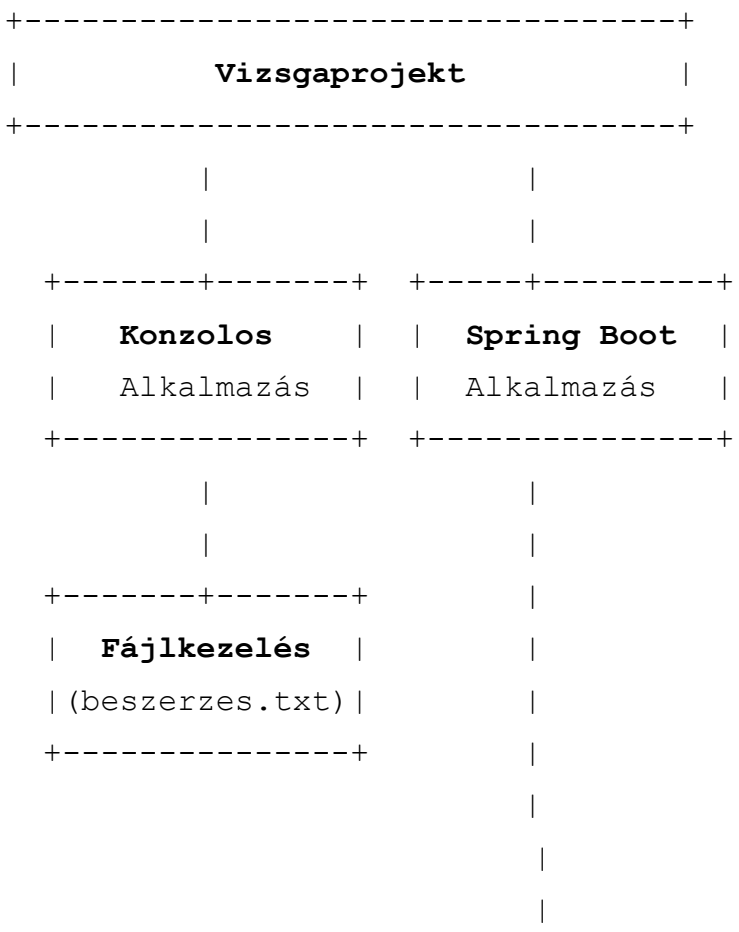
</parent>

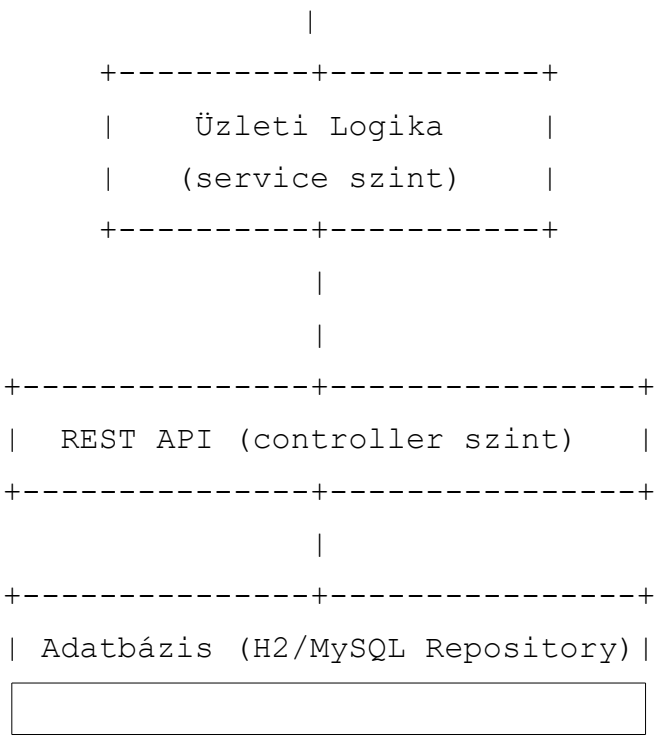
De mielőtt a maven -tárolódat kiüríted, törlöd, csinálj másolatot róla, mert könnyen időzavarba kerülhetsz, ha azt követően nem működne semmi sem..

A projekt két fő részből és egy script kiegészítőből áll:

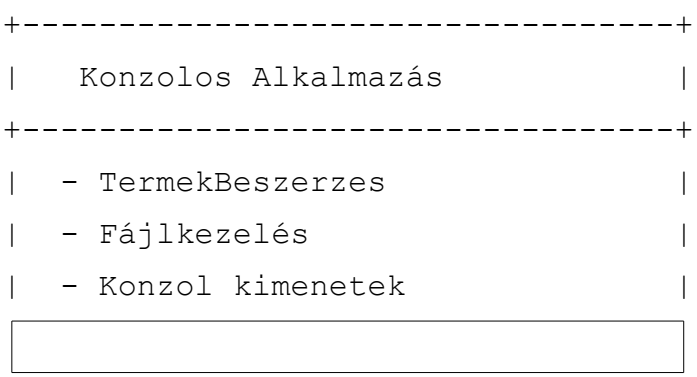
1. **Konzolos Alkalmazás**: Ez az alkalmazás a beszerzéseket kezeli, és egy `beszerzes.txt` nevű fájlba menti az adatokat.
2. **Spring Boot Alkalmazás**: Ez a rész a termékek adatbázisba történő mentését, valamint az értékesítési adatok kezelését végzi.
    1. A rendszer egy REST API-n keresztül biztosítja az értékesítési műveletek végrehajtását és adminisztrálását.
    2. A REST API -hoz kliensként a POSTMAN -t használd.
3. **SQL-Script:** készítsd el a Workbench-ben a projektet az "adattárolás sql"-ben szintet megvalósító SQL scriptet a Specifikáció utolsó bekezdése alapján!

A projekt Jackson digram (-szerűen) ábrázolva:





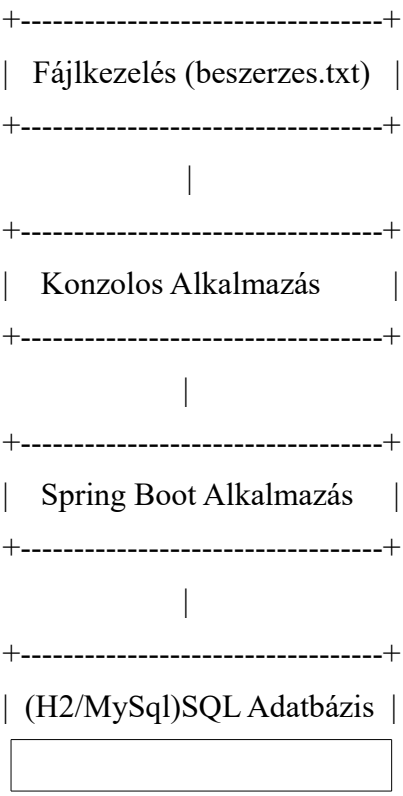
2. **Konzolos Alkalmazás**



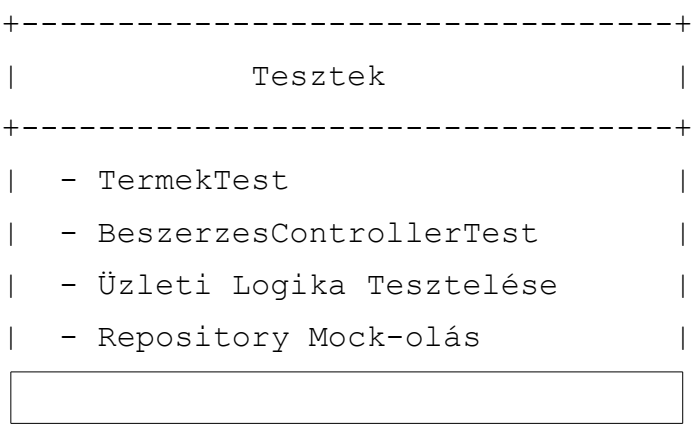
3. **Spring Boot Alkalmazás:**

```
+----------------------------------+
|   Spring Boot Alkalmazás         |
+----------------------------------+
|  - GyartasApplication            |
|  - REST API                      |
|  - Üzleti Logika (Service)       |
|  - Repository                    |
|  - Adatbázis (H2/MySQL)          |
+----------------------------------+
```

4. **Adatáramlás:**



5. **Tesztek**



Magyarázat:

- **Konzolos Alkalmazás**: Főként a `TermekBeszerzes` osztályt és a fájlkezelést tartalmazza, ami a `beszerzes.txt` fájllal dolgozik.
- **Spring Boot Alkalmazás**: A `GyartasApplication` osztályból indul, amely REST API-t biztosít. Az üzleti logikát a `Service` réteg, az adatkezelést pedig a `Repository` és a MySQL adatbázis végzi.
- **Adatáramlás**: A folyamat a fájlkezeléstől indul, amelyet a konzolos alkalmazás dolgoz fel. Ezután a Spring Boot alkalmazás dolgozza fel tovább az adatokat, és végül az eredményeket az adatbázisba menti.
- **Tesztek**: A projekt különböző részeihez tartozó tesztosztályok biztosítják, hogy az üzleti logika, a REST API és az adatbázis műveletek megfelelően működjenek.

Ez a diagram segít abban, hogy jobban átlásd a projekt szerkezetét, és könnyebben megértsd a komponensek közötti kapcsolatokat.

1. Konzolos alkalmazás:

A konzolos alkalmazás a beszerzési adatok adatbevitelének a lekezelésére szolgál, és a következő funkciókat valósítja meg:

- **1/a, Termékek beszerzése**: A felhasználó megadhatja a beszerzendő termékek nevét, mennyiségét és egységárát.
- **Fájlkezelés**: A program az adatokat a `beszerzes.txt` fájlban tárolja (pontos-vesszővel elválasztott, ASCII formátumú UTF-8 kódolású, egyszerű szövegfájlként, még az 1/a pont részeként).
    - Az output valahogy így néz ki:
    - +------------------------------+
    - | 1. feladat adatok felvitele |
    - +------------------------------+
    - Beszerzesi lista elmentve a 'beszerzes.txt' nevu fajlba (a projekt-gyoker mappaban kell lennie).

- **2/a feladatként** a program listát készít a bevitt adatokról.
    - Az output valahogy így néz ki:
    - +------------------------------+
    - | 2/a. feladat a listakeszites |
    - +------------------------------+
    - Beszerzes(1. tetel)
    - {
    - beszerzett cikkelem : "elso termek",
    - beszerzett mennyiseg : 500 db,
    - beszerzesi ar : 1250.25 HUF
    - }
    - Beszerzes(2. tetel)
    - {
    - ...
- **Üzleti logikák:**

    - A program képes a beszerzési lista alapján, a beszerzés értékének kiszámítására:
        - +---+
        - | 2/b/i. feladat a heti beszerzés erteke |

- +----------+
- Az egy heti beszerzes erteke: 11656588,29 HUF

- a beszerzési lista alapján, az egy éves tervezett beszerzési állomány kiszámítására:
  - +----------+
  - | 2/b/i. feladat a beszerzés erteke |
  - +----------+
  - A kovetkezo ev tervezett beszerzesi allomanyanak erteke (inflacioval szamitva): 630388294,72 HUF

Fájl Struktúra (belső szerkezet UTF-8, pontos-vesszővel elválasztott (split";") szövegfájl:

- **beszerzes.txt**: A termékek beszerzési adatait tartalmazza a következő formátumban:

  `elso termek;500;1250.25`
- `masodik termek;2500;125.92`
- `harmadik termek;2489;4305.61`

Példa az adatbeviteli folyamatra: A bevitelkori ellenőrzéshez használd fel a modelhez tartozó .toString() , illetve a szintén itt található .toConsole(sztring) metódusokat (a konzolra való megjelenítést segítendően).
Például ha a megrendelés kiszolgálható-e kérdést vizsgálnád egy lehetséges kiíratás, hogy mit csinál pontosan:

```
termek.toConsole("teszthez Eladni=" + eladasra.getRendeltMennyiseg()  );
```

Akkor ezt írja ki egy példa esetén:

```
Termek(teszthez Eladni=20)
{

       termeknev : "elso termek",

       eredeti_mennyiseg : 100 db,

       ebbol_eladva : 70 db,

       egysegar : 1250.25 HUF


}
```

Figyelj arra, hogy amikor a SCANNER osztály egy numerikus adatból INT-típust vesz el, akkor mindent ami a tizedespont után áll, "otthagy" – így az ENTER karaktert is (amit a bevitel végény nyomtál)... ez ugye a következő adat elvételekor tud fura dolgokat produkálni!

```
Megnevezes: elso termek
Mennyiseg: 500
Egysegar: 1250.25
```

A `beszerzes.txt` fájl tartalmát a Spring Boot alkalmazás fogja beolvasni és tovább feldolgozni.

---

2. Spring Boot Alkalmazás

A Spring Boot alapú alkalmazás a beszerzett termékeket adatbázisba menti, és lehetővé teszi az értékesítési folyamatok kezelését.

2.1 Termékkezelés

A beszerzési adatok beolvasása után a rendszer a következő funkciókat biztosítja:

- **Beszerzések SQL adatbázisba mentése**: A `beszerzes.txt` fájl tartalmát beolvassa, és minden egyes terméket, amely még nem szerepel az adatbázisban, eltárol a `Termek` entitásként.
- Az entitás létezésére irányuló ellenőrzést a "megnevezes" atributum alapján kell megvalósítani!
- Fontos: Tehát csak azt írd az adatbázisodba, amely még nem létezik.

2.2 Értékesítési Rendszer

Az értékesítési rendszer a következő funkciókat valósítja meg:

- **Értékesítési adat rögzítése**: Egy adott termék értékesítési adatai kerülnek mentésre előzőleg a `Termek` entitás `eladva` mezőjének frissítésével majd pedig ennek sikeres függvényében az értékesítési adat tárolásával az `ErtekesitesAdat` entitásban.
- **Értékesítési adatok lekérdezése**: Az eddig rögzített összes értékesítési adat lekérdezése és listázása.
- **Egy termék eladható mennyiségének a lekérdezése:** Rákeresve a `Termek` entitásra, annak a `mennyiseg - eladva` mező egyenlege mutatja meg az eladható mennyiséget.
- **Itt is, a Minden `Termek entitásra` irányuló keresést:** kizárólag a megnevezésre kell magvalósítani!

2.3 REST API

Az API a következő végpontokat biztosítja (ezeket POSTMAN -ből tudod használni):

- **GET /beszerzesek/adatatvetel**: A `beszerzes.txt` fájl tartalmából az SQL-be való adatok átvitelét.
- **GET /beszerzesek/lista**: Az összes beszerzési adat listázása.
- **GET /beszerzesek/beszerzes/{termekNev}**: egy konkrét beszerzési adat listázása
- **GET /termek/termeklista**: Az összes beszerzési adat listázása.

- **POST /eladasok/megrendeles**: Egy új értékesítési adat rögzítése. Először ellenőrzi, hogy van-e elegendő készlet a kért termékből, majd elvégzi a termék eladási adatainak frissítését és az értékesítési adat mentését.
- **GET /eladasok/lista**: Az összes eddigi értékesítési adat listázása.
- **GET /beszerzesek/beszerzes/{termekNev}**: egy konkrét termék eladható mennyiségi adata listázása

3, A JUnit Tesztelés.

Összefoglaló a "`TermekService`" és az "`ErtekesitesService`" osztályokhoz:

a, A tesztkörnyezet

A tesztelési környezetet Spring Boot alapú JUnit tesztekkel és egyszerű JUnit tesztekkel valósítjuk meg.

A Spring Boot tesztelési keretrendszerének használata lehetővé teszi a teljes Spring alkalmazás kontextusának betöltését, így a tesztek során az alkalmazás valós működéséhez hasonló környezetben tudjuk vizsgálni az egyes szolgáltatásokat.

b, Fontos részletek – csak emlékeztetőül**:**

Az egyszerű **`JUnit`** teszteknél a `@Mock`, a **`SpringBoot`** teszteknél viszont `@MockBean` annotációval kell ellátnunk a tesztelni kívánt osztályunknak a mockolni szükséges külső függőségeit! Továbbá az egyszerű JUnit tesztnél az injektort a `@InjeckMock` annotáció-, a `@SpringBootTest` esetében pedig az `@Autowired` annotációval hívjuk!

És ugye injektálni mindig a tesztelni kívánt osztályt kell... illetve fontos még: hogy a springboot JUnit teszt esetében a teljes springes környezet betöltődik, míg az egyszerű JUnit esetében csak a mockolt függőségek és az injektált, vagy példányosított osztályok és metódusok dolgoznak.

A JUnit tesztek a **`Beszerzes`, `Termek`** és **`BeszerzesController`** osztályokhoz:

3/1. TermekTest() osztály:

A `TermekTest` osztály célja a `Beszerzes` modell osztály egyes funkcióinak tesztelése. A tesztelés során a következő feladatokat kell elvégezni:

- A **konstruktor()** tesztelése **(`testConstructor`)**:
    - Készíts egy új `Beszerzes` objektumot a konstruktor használatával, és ellenőrizd, hogy az objektum megfelelően inicializálódott-e a megadott adatokkal.
    - A teszt során ellenőrizni kell, hogy a termék neve, mennyisége és ára helyesen került-e eltárolásra.
- A **ToString()** metódus tesztelése **(`testToString`)**:
    - Ellenőrizd, hogy a `Beszerzes` osztály `toString` metódusa a várt formátumban adja-e vissza a termék adatait.
    - A tesztnek biztosítania kell, hogy a `toString` kimenet pontosan tükrözi az objektum állapotát, beleértve a formázást és a mezők értékeit.

3/2. BeszerzesControllerTest() osztály:

A `BeszerzesControllerTest` osztály célja a `BeszerzesController` REST API végpontjainak tesztelése. A Web MVC tesztelés segítségével a következő feladatokat kell elvégezni:

- **GET Kérés Tesztelése a Beszerzési Adatok Lekérdezéséhez**

(`testGetBeszerzesesek`):

- Web MVC tesztet kell végrehajtani, amely szimulálja a `GET /beszerzesek/lista` HTTP kérést.
- A teszt során a következő lépéseket kell végrehajtani:
  - Állítsd be a mock-olt `BeszerzesController`-t, hogy a `getBeszerzesek()` metódus egy előre meghatározott beszerzési listát adjon vissza.
  - Készíts egy HTTP `GET` kérést a `/beszerzesek/lista` végpontra, és ellenőrizd, hogy a válasz státusza `200 OK`.
  - A válaszban lévő JSON objektumokat ellenőrizni kell, hogy a megfelelő mezőket tartalmazzák-e, és azok értékei helyesen egyeznek-e meg a várt adatokkal.
    - Például: az első elemnek az "Alma" nevű terméket kell tartalmaznia, 10-es mennyiséggel és 2.5 árral, míg a második elemnek a "Banán" nevű terméket kell tartalmaznia, 20-as mennyiséggel és 3.0 árral.

Függőségek Mock-olása és konfigurálása:

- A `TermekTest` osztályban a `ProjectAdmin` szolgáltatás injektálása történik, amely biztosítja a szükséges adminisztratív funkciók elérhetőségét a tesztelés során. Mivel a `ProjectAdmin` nem közvetlenül szerepel a tesztekben, nincs szükség külön mock-olására ebben az osztályban.
- A `BeszerzesControllerTest` osztályban a `BeszerzesController` mock-olása történik a `@MockBean` annotációval. Ez biztosítja, hogy a Web MVC tesztek során a kontrollert a teszt környezet izoláltan kezeli, és a metódushívások előre beállított válaszokat adnak vissza.

Célok

A tesztelés célja, hogy biztosítsa a `Beszerzes` modell, valamint a `BeszerzesController` megfelelő működését. A modellek és a REST API végpontok tesztelése garantálja, hogy az alkalmazás megfelelően kezeli a termékadatokat, és a kliens oldalon a megfelelő válaszokat kapja.

3/3. A `TermekService` osztály `@SpringBootTest` -el való tesztelése során a következő fontos üzleti logikai elemeket vizsgáljuk a `TermekServiceSpringBootTest` osztállyal:

- **Beszerzési adatok fájlból történő beolvasása (`beszerzestTxtbolKiolvas`)**:
  - A teszt során a fájlolvasás helyettesítésére egy mockolt `BeszerzesFileReader` szolgál. Ellenőrizzük, hogy a beolvasott adatok helyesen kerülnek visszaadásra, ha a megadott termék megtalálható a beszerzési adatok között!
- **Beszerzési adatok SQL adatbázisba mentése (`beszerzestSqlbe`)**:

- Itt azt vizsgáljuk, hogy a fájlból beolvasott termékek helyesen kerülnek-e át az SQL adatbázisba, ha még nem szerepelnek benne. A teszt során ellenőrizzük a termékek helyes hozzáadását és a listák kezelését!

3/4, Egy `JUnit` tesztosztály a `ErtekesitesService` osztály üzleti logikájának tesztelésére. A tesztjeink elsősorban a következő metódusokra összpontosítanak:

Az `ErtekesitesServiceJUnitTest` osztály az alábbi üzleti logikai funkciókat vizsgálja:

- **Értékesítési adat rögzítése (`ertekesitesRogzitese`)**:
  - A teszt során ellenőrizzük, hogy az értékesítési adat helyesen kerül-e eltárolásra, és hogy a termék eladott mennyisége megfelelően frissül-e. A mockolt `TermekRepository` és `ErtekesitesRepository` biztosítja a függőségek izolálását.
- **Megrendelés kiszolgálhatóságának ellenőrzése (`kiszolgalhatoE`)**:
  - A teszt ezen a ponton azt vizsgálja, hogy egy adott termékből a rendelni kívánt mennyiség rendelkezésre áll-e. A termék raktárkészlete és eladott mennyisége alapján történjen az ellenőrzés (egyébként is ez egy külön service-metódusban kell legyen).
- **Eladható mennyiség lekérdezése (`mennyiEladhatoVanBelole`)**:
  - Ezen tesztben azt ellenőrizzük, hogy egy adott termékből mennyi eladható mennyiség áll rendelkezésre. Irjunk külön tesztet **arra az esetre, ha a termék nem létezik**, biztosítva, hogy a rendszer ilyenkor nulla értéket adjon vissza (Ez a való gyakorlatban rengeteg probléma forrása tud lenni!).

Mockolt Függőségek és Konfiguráció

Mindkét tesztosztályban a függőségeket mockoltuk a `@Mock,@MockBean` annotáció valamelyikével, hogy a tesztjeink izoláltak maradjanak, és ne függjenek a valós adatbázis apcsolattól vagy fájlrendszertől. Az `@InjectMocks` és `@Autowired` annotációk pedig biztosították, hogy a megfelelő osztályok (service-ek) a megfelelő mock-olt függőségekkel működjenek.

4. Adatbázis Szerkezet

A projekt alapvetően a beépülő memória adatbáziskezelőre épít (H2) de a megfelelő konfigurációval (az application.properties-ben) más, pl MySQL adatbáziskezelő használata is lehetséges a beszerzési és értékesítési adatok tárolására.

Megjegyzés: alapértelmezésben a H2 minden buildelést követően "elfelejti" az adatokat, tehát újra be kell tölteni azt a beszrzes.txt -ből – és így az eladáésok is elvesznek -, de megfelelően paraméterezve ez átállítható!

4.1 `Termek` Tábla

Ez a tábla tárolja a termékek adatait, beleértve a nevüket, a mennyiséget, az eladott mennyiséget, és az egységárat.

```
Termek (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    nev VARCHAR(255) NOT NULL,
    mennyiseg INT NOT NULL,
    eladva INT DEFAULT 0,
    ar DOUBLE NOT NULL
);
```

4.2 `ErtekesitesAdat` Tábla

Ez a tábla tárolja az egyes értékesítési adatokat.

```
ErtekesitesAdat (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    termekNev VARCHAR(255) NOT NULL,
    eladottMennyiseg INT NOT NULL,
    eladasiAr DOUBLE NOT NULL
);
```

5. Az SQL-hez Tesztadatok:

Az alábbiakban található SQL parancsok segítségével előre definiált tesztadatokat tölthetsz fel a MySQL adatbázisodba – melyet azonban előzőleg létre kell hoznod.

Ez egy külön feladatod. Az eredményt pedig egy sql-sript-page -en megoldva kimented egy script.sql elnevezésű szkript fájlba. (Lásd még később is).

```
-- Tesztadatok feltöltése a Termek táblába
INSERT ...

-- Tesztadatok feltöltése az ErtekesitesAdat táblába
INSERT ...
```

5/1. Adatbázis struktúra létrehozása séma elnevezése legyen : "ertekesitesrendszer"

```
CREATE ...
USE ertekesitesrendszer;

CREATE TABLE Termek (...);

CREATE TABLE ErtekesitesAdat (...
    FOREIGN KEY (termek_id) REFERENCES Termek(id) ON DELETE CASCADE
);
```

5/2. Adatok felvitele

```
-- Termékek felvitele
INSERT INTO Termek (...)
            VALUES (...);
INSERT ...
INSERT ...

-- Értékesítési adatok felvitele
INSERT INTO ErtekesitesAdat (...)
            VALUES (...);
INSERT ...
INSERT ...
```

5/3. Alapvető lekérdezések

```
-- Lekérdezés az összes termék adatainak megjelenítésére
SELECT ...

-- Lekérdezés:
--  - az összes értékesítési adat megjelenítésére,
--  - termék szerint csoportosítva
SELECT ... FROM ErtekesitesAdat JOIN Termek ON ... ORDER BY termekNev;

-- Egy adott termék (az első termék) értékesítési adatainak lekérdezése
```

```
SELECT ... FROM ErtekesitesAdat JOIN ... ON ... WHERE nev = ...;
```

5/4. Szelektív frissítés (UPDATE)

```
-- A 2 sorszámú (itt ne a nevet add meg!) termék eladott mennyiségének
-- frissítése (adj hozzá 100 eladott terméket a 2-es sorszámúnál)
-- (a Workbench alapbeállítása nem engedélyezi a Primary kulcs nélküli
UPDATE/DELETE parancsok végrehajtását):
UPDATE ...
```

5/5. Szelektív törlés (DELETE)

```
-- A 3--as termék összes értékesítési adatának törlése (a Workbench
-- alapbeállítása nem engedélyezi a Primary kulcs nélküli UPDATE/DELETE
-- parancsok végrehajtását)
DELETE ...

-- Az 1-es tétel egyik eladott mennyiséghez tartozó értékesítési adat törlése
DELETE ...
```

6. SQL- ismereteid tesztelési folyamata, a tesztfeladatok:

A projekt tesztelése során a hallgatóknak a következőket kell elvégezniük:

1. **Beszerzések feltöltése**: Használjátokk a konzolos alkalmazást a `beszerzes.txt` fájl feltöltéséhez.
2. **Adatok átmentése (átemelése) az SQL-be**: Indítsátok el a Spring Boot alkalmazást, hogy a `beszerzes.txt` fájl tartalmát átvigyétek az adatbázisba.
3. **Értékesítések rögzítése és lekérdezése**: Használják a REST API-t az értékesítési adatok rögzítésére és lekérdezésére.

Végül az SQL feladathoz:

Mentsétek el az SQL parancsokat egy `skript.sql` fájlba (ezért célszerű egy SQL-script lapon dolgozni és az adatmentés a <CTRL-S> billentyűkombinációval indíthatod el a MySQL Workbench használatával. A szkriptfájlt a projekted gyökérmappájába mentsd el, hogy minden együtt legyen!

Vizsgafeladat Pontozási Szabályzat:

1/a) OOP (Objektum Orientált Programfejlesztés) – a konzolalkalmazás:

- **Feladat**: A "TermekBeszerzes" a gyártásból vagy egyéb módon beszerzett termékeket tartja nyilván eladás céljából.
- Ez a konzolalkalmazás indítja el a vizsgaprojektet.
- A feladtok – melyet az alkalmazásnak tudnia kell - a Specifikáció első részében pontosan meg lettek határozva.

Az ide vonatkozó vizsgafeladatok és értékelésük maghatározása:

1. **Hibakeresés és javítás:**
    - A kódban **három komolyabb** hiba-csoport (**nem egy hiba, lehet 5-6 hiba, hiányos előkészíté, stb, de három csoportban, metódusban**) van elrejtve, amelyek miatt a kód nem fut végig. Ezeknek a hibáknak a kijavítása egyenként 10-10 pontot ér.
    - **Adható pontszám**: 30 pont
2. **Hiányzó kódrészek megírása:**
    - **2.a)** Írj egy metódust, amely konzolra listázza a `beszerzes.txt` tartalmát.
        - **Adható pontszám**: 30 pont
    - **2.b)** Írj egy metódust, amely két dolgot ír ki a konzolra
        - **i)** Számolja ki az összes megrendelés együttes értékét és írja ki a konzolra.
        - **ii)** Az előző pontban kiszámolt értéket tekintse 1 hét bázisadatának, és +4%-kal indexálva számolja ki, mennyi lesz az 1 év várható rendelésállomány értéke.
    - **Adható pontszám**: 40 pont

1/b) Backend-fejlesztés – a Spring alkalmazás:

- **Feladat**: A "`gyartasiRendelesKezeles`" egy Spring alapú projekt, amely kiterjeszti a projektet. A feladatok – melyeket az alkalmazásnak tudnia kell - a Specifikáció előző fejezeteiben ki vannak dolgozva.
- **REST API használata**: A feladat REST API-alapú, ezért szükség lehet (pl) a POSTMAN eszköz használatára.

Az ide vonatkozó vizsgafeladatok és értékelésük maghatározása:

Hibakeresés és javítás:

- A kód számos hibát rejt. Ebbe több dolog is benne foglaltatik, akár reláció jel hibás/fordított/stb, akár téves referencia hivatkozás, inicializáció, fölösleges metódus hivatkozás, szükséges metódus hivatkozás hiánya - de bármi más is lehet.

Hiányzó kódrészek megírása:

- a megadott osztályokban, a megadott metódusnevek alá, a funkció megfelelő

ellátásához szükséges kódrészletek elkészítése.

- **Adható pontszám**: 200 pont

1/c) Teszteseteket kell kidolgozni: minden funkcionalitásra és üzleti logikára. A feladatok részletesen kidolgozottak a specifikáció előző fejezeteiben.

- **Egyes hibás tesztek javítása:**
  - bármilyen hiba lehet, annotáció hiánya, mockolás hiánya, hibás mockolás, inicializáció hiánya, stb.
- **Tesztesetek hiányzó kódrészeinek megírása:**
  - Figyeljetek oda, hogy milyen típusú a teszt! (pl, hogy egyszerű JUnit, vagy SpringBoot)
    - **Adható pontszám**: 200 pont

2) SQL - MySQL Script

- **Feladat**: A második feladat egy MySQL script elkészítése, amelyet a "VizsgaSpecifikáció" részletez. A scriptet egy munkafelületen kell elvégezni, majd be kell másolni a projekt gyökerében található "sql.script" fájlba.
  - **Adható pontszám**: 100 pont

Összefoglalva:

Az adható pontszámok együttes értéke 600 pont.

A vizsgát (elsősorban) akkor teintjük sikeresnek, ha az alkalmazás fut.

A vizsgát akkor is sikeresnek tekintjük, ha elérted az 51 %-ot!

Köszönjük a tanfolyamon való részvételt!

Jó egészséget, sok-, sok jövőbeli munkát, ezzel együtt sikerélményt és sok üzleti sikert, az itt tanultak alkalmazásához!

Szobonya László Zoltán
Anzek Informatika Kft

+3670/634 8 777