**Eseményvezérelt programozás**

**Beadandó projectmunka**

**Csapatunk tagjai:**

Harkai Martin - H1RWPP – h1rwpp@inf.elte.hu

Rába Krisztián - YP5Y25 - yp5y25@inf.elte.hu

Vass-Horváth Balázs – B0XUPT - b0xupt@inf.elte.hu

**Választott téma:**

**IK WEBSHOP**

Tárgykód: IKSEK-17EVPROGEG

Gyakorlatvezető: Soós Sándor

# Tartalomjegyzék

# Fejlesztői információk

## Felhasznált nyelvek, technológiák, szoftverek

### Java – Spring keretrendszer

A java napjaink egyik legnépszerűbb objektumorientált programozási nyelve. Fejlesztése egészen az 1990-es évek elejére nyúlik vissza, amikoris még a Sun Microsystems fejlesztette. 2009-ben az Oracle cég tulajdonába került egy felvásárlás eredményeképpen, és napjainkban is ők felelnek a fejlesztéséért. Érdekesség, hogy egy kávéscsésze lett az ikonja a nyelvnek, melyet kávézás közben találtak ki. Népszerűségét valószínűleg annak köszönheti, hogy platformtól függetlenül fejleszthetünk alkalmazásokat a segítségével. A platformfüggetlenséget egyedi módon valósította meg a cég. Az általunk megírt forráskódból a fordító valójában nem egy futtatható (exe) állományt hoz létre, hanem úgynevezett java bájtkódot, mely a Java Virtual Machine (JVM) virtuális gépen fog futni. A megvalósítás előnye a hardware, és platform függetlenség, ugyanakkor hátránya, hogy ez lassítja a programunk futását, valamint növeli az erőforrásigényét. Annak érdekében, hogy java alkalmazást fejleszthessünk szükségünk van az úgynevezett Java Development Kit-re (JDK), mely a fejlesztői eszközök mellett tartalmazza a Java Runtime Environment-et (JRE) is.

A Spring egy nyílt forráskódú, java keretrendszer, melynek első korai verziója 2003-ban jelent meg, míg az első stabil kiadást 2004-ben adták ki. A java nyelvvel együtt a keretrendszer is folyamatos fejlődésnek örvend. Kezdetben csak néhány modul volt elérhető, és asztali alkalmazások fejlesztését gyorsította. Napjainkban már több-száz kiegészítő modult importálhatunk be, leggyakrabban webes alkalmazások fejlesztésére használjuk. Néhány hasznos modul például az adatbázis-kezelést gyorsító JDBC, vagy az autentikációhoz használgató oAuth2 modul, melyek használatával rengeteg időt nyerhetünk egy alkalmazás fejlesztésekor. Fontos jellemzője továbbá, hogy jól használható dokumentációja van, ezért a java nyelv ismeretében könnyen nem fog gondot okozni a használata. Spring alkalmazást ma már néhány kattintással is létrehozhatunk, csak a project típusát (maven / gradle), nyelvét (java / kotlin), a spring verzióját, és a project adatait (név, leírás, package név, java verzió) kell megadnunk, valamint kiválaszthatjuk a használni kívánt kiegészítő modulokat.

Java - <https://hu.wikipedia.org/wiki/Java_(programoz%C3%A1si_nyelv)>

Spring - <https://hu.wikipedia.org/wiki/Spring_keretrendszer>

### MySql 8.0

A MySql az Oracle Corporation által fejlesztett SQL alapú relációs adatbázis szerver. A MySql mára az egyik legelterjedtebb adatbázis-kezelő olyan okok miatt, mint például a nyílt forráskód, a megbízhatóság, a stabilitás, valamint az, hogy platform független, tehát elérhető többek között Windows, Linux, MacOS X, FreeBSD, és szinte minden további operációs rendszeren. Az adatbázis kezeléséhez szükségünk van továbbá egy adatbázis kezelő szoftverre, mint például a Navicat, vagy a MySql Workbench. MySql adatbázis kezelő szoftverek szinte minden operációs rendszerre léteznek, még Androidra, vagy IOS-re is. Alternatívájaként érdemes megemlíteni az MSSQL-t, amely hasonló hatékonysággal képes ellátni az adatbázis feladatokat, azonban mivel a MySql-t az Oracle cég fejleszti, ezért (is) érdemesebb java alkalmazásokhoz

MySql - https://hu.wikipedia.org/wiki/MySQL

### React JS

A React vagy más néven ReactJS egy ingyenes, open-source front-end Javascript könyvtár, aminek a segítségével könnyedén hozhatunk létre felhasználói felületeket(User Interface) komponensekkel. Előfeltétele, hogy rajta legyen a Node a helyi rendszeren, ha ez teljesül, akkor akár egy parancs kiadásával a cmd-ben vagy bash-ben létre is hozhatjuk a React projektünket. Nagy szeretettel használják az SPA(Single-Page Application) miatt is, hiszen a kliensnek nem kell újra töltenie az oldalt. A React state-eket vagyis „állapotokat” kezel és jeleníti meg a DOM-on(Document Object Model). A state-k automatikusan frissülnek, ha változás történik rajtuk, anélkül, hogy újra betöltenénk az oldalt. A komponensek újra felhasználhatóak(Pl: Ha több oldalon akarjuk ugyanazt a tartalmat megjeleníteni), de még úgymond props-ot is kaphat, ami azért jó, mert a komponensek között értékeket tudunk átadni. A komponenseket exportálni kell, hogy azt másik komponensben be tudjuk importálni. Léteznek függvény és osztály-alapú komponensek, amik abban különböznek, hogy a függvényes komponensekkel hook-okat, magyarosítva „horgonyokat” használhatunk, viszont az osztály alapúaknál már nem megengedett. Beépített Hook-ok: useState, useEffect, useContext, amiket legtöbbször a statek és a mellékhatások kezelésére használják a napokban. Talán ezért is kedveltebb a függvényes megoldás. Vannak Lifecycle metódusok is, amikkel meghatározhatjuk, hogy a komponensek frissüljenek-e. A react egyébként a JSX szintakszist használja, ami nagyban hasonlít a JavaScript-hez, egyik előnye, hogy elfogadja a HTML tageket, de még a CSS-t is. Az npm-el telepíthetünk további JS modulokat (a mi projektünk pl a react-router-dom-ot használja a route-olás megoldásához), amik a node\_modules mappában vannak tárolva. Fontos: A node\_modules mappát mindig ignoráljuk, amikor commit-olunk! Ha többen dolgozunk egy react projekten, akkor indítás előtt(npm start) érdemes beiírni az npm install-t, hogy lehetőleg elkerüljük az incidenseket.

### GitHub

A GitHub a Microsoft egy leányvállalataként a verziókezelés, verziókövetés lehetőségét biztosítja. A projectmunka elkészítése közben fontos volt, hogy párhuzamosan fejleszthessük az alkalmazásunkat. Ezen felül lehetőséget biztosít számunkra, hogy egy esetleges hibás fejlesztést követően visszaállítsuk a kódot egy régebbi verziójára, és nem utolsó sorban egy esetleges adatvesztés következtében is vissza tudom nyerni a korábbi munkánkat. Segítségével nyomon lehet követni, hogy miként fejlődik a program, mikor milyen fejlesztéseket, újítások kerülnek bele. Bár legfőképpen forráskódjainkat tároljuk itt, de lehetőség van például dokumentáció, Wikipédia, integrációs könyvtárak (LIB-ek), és még rengeteg virtuálisadat tárolására, nyomon követésére. A GitHub továbbá lehetőséget nyújt számunkra kisebb weboldalak tárolására, valamint kódjaink publikálására, és még megannyi hasznos funkcióra, melyeket szinte a végtelenségekig lehetne sorolni. Alternatívaként rengeteg lehetőséget felsorolhatnék, de talán a legismertebb az azonos alapokra épülő GitLab.

GitHub - <https://en.wikipedia.org/wiki/GitHub>

## Használt szoftverek, fejlesztői környezetek

### SpringToolSuite4

A SpringToolSuite4 egy Eclipse-re épülő spring (java) fejlesztői környezet, a VMware cégtől. A többi fejlesztői környezettel szemben hatalmas előnye, hogy segítségével letölthető modulokat építhetünk be a projectünkbe automatikusan, nem kell ezt manuálisan elvégeznünk. Teljes értékű java fejlesztői környezet is, mivel Eclips alapra épül, ezáltal tökéletesen használhatjuk az alkalmazásunk fejlesztése során.

### Navicat

A Navicat egy 2002-ben, a PremiumSoft CyberTech Ltd. által bemutatott grafikus adatbáziskezelő szoftver, mely támogatja többek között a MySql-t, MariaDB-t, MongoDB-t, Oracle, SQLite-ot, PostgreSQL-t és az MSSQL-t is. A Navicat elérhető Linux, Windows, és MacOS X rendszerekre is. Felhasználó barát felületének köszönhetően átlátható, egyszerűen használható. A hagyományos lekérdezéseken túl adatbázis modellezésre, tervezésre, és adatbázis kapcsolatok kialakítására is egyaránt alkalmas. Az adatbáziskezeléshez alternatíva lehet az ingyenesen elérhető MySql Workbench.

Navicat - <https://en.wikipedia.org/wiki/Navicat>

### GitHub Desktop

A GitHub Desktop a GitHub saját fejlesztéstű szoftvere. Célja, hogy a GitHub-on kezelt munkáinkat könnyebben, átláthatóbban, a szóban forgó asztali alkalmazással elérhessük. Elérhető Windows, valamint MacOS, rendszerekre, viszont amennyiben Linux alapú rendszerünk van, sajnos más alternatívát kell keresnünk. Fontos megemlíteni, hogy a GitHub mobilos alkalmazást is fejlesztett azonos a célra, amely elérhető androidos, és IOS-es eszközökre is. A program lehetőséget biztosít a GitHub verziókövető rendszer legtöbb funkciójának elérésére, miután bejelentkeztünk a GitHub fiókunkkal. Ezen funkciók közül a leglényegesebbek például a projektünk klónozása az adott számítógépre, a módosítások feltöltése, valamint a korábbi verziók visszaállítása, de összehasonlíthatjuk vele a programunk két verzióját, valamint megnézhetjük, mikor mit módosítottunk rajta. Bár a Visual Studio rendelkezik beépített funkcióval, mely segítségével letölthetjük a projektünk friss verzióját, valamint feltölthetjük a módosításainkat, ez messze elmarad funkcionalitás, felhasználói felület, és kezelés terén is a felhasználóbarát GitHub Desktop-tól. Hasonló alkalmazásként megemlíthető például a Fork.

### Visual Studio Code

### Frontend

#### Bootstrap

A bootstrap egy nyílt forráskódú CSS keretrendszer, ami nagyban elősegíti a nyers html oldal formázását. A dokumentációja nagyon részletes, és könnyen el lehet sajátítani a hatalmát. Bár van olyan dolog, amit nem képes megoldani, de alternatívákat bőven lehet találni az interneten.

#### jQuery

A jQuery egy gyors, kisméretű, de nagyon hasznos és sok oldalú JavaScript könyvtár.  
Bármilyen körülmények közt megállja a helyét, kezdők és tapasztaltak is szeretik használni, mert egyszerű és hatékony.  
 Animációk létrehozásátó, az események vezérlésén át, szerver-kliens kommunikációt is beleértve, mindennel megbirkózik. Szinte az összes böngésző támogatja.

#### Font Awesome

A Font Awesome (röviden FA icons) egy olyan css könyvtár, ami rengeteg ikont tartalmaz. Mivel „szövegként” jelenik meg, ezért nagyon testre szabható, és olyan éles lesz, amennyire csak a kijelző engedi. Ennek a hátránya az, hogy csak 1 színű lehet, amilyen a beállított szöveg színe.  
Van belőle internetről lehúzott verzió (mindig a legfrissebb) és a helyileg tártolt verzió (elég nagy a letöltendő fájl méret).

## Adatbázis, tárolt adatok, adatvédelem

A program működéséhez szükséges adatok tárolásáért egy MySql adatbázis felel. A tárolt adatok közötti kapcsolat az adatbázis ábrán látható. Adatintegritási problémák elkerülése végett 3 szinten (Kliens – Szerver - Adatbázis) is szigorú ellenőrzésen mennek keresztül az adatok az eltárolás előtt. A kliens oldali ellenőrzés célja főleg a felhasználó informálása, és a felesleges adatforgalom generálásának elkerülése a célja. A legfőbb ellenőrzés szerveroldalon, valamint az adatbázisban zajlik, ahol az egyes táblák közötti kapcsolatok kialakításán túl az adatbázisnak küldött adatok ellenőrzése (Pl.: karakterlánc hossza, numerikus típusok min/max értéke stb.) is megtörténik (CONSTRAINT).

Egy olyan alkalmazásnál, ahol a felhasználó személyesebb adatait is eltároljuk (Pl.: név, szállításhoz lakcím) kiemelkedően fontos az adatvédelem, hogy jogosulatlan személyek ne férhessenek hozzá az ügyfelek adataihoz. Ennek érdekében szükséges, hogy az alkalmazáson belül is helyén kezeljük a jogosultságokat (Például egy átlag felhasználó ne férhessen hozzá a többi felhasználó adataihoz), valamint megfelelően védjük az adatbázisunkat a külső támadásoktól. A külső behatolások megakadályozása érdekében első, és legfontosabb, hogy az adatbázisunk felhasználói megfelelően legyen beállítva (minden felhasználó csak a szükséges adatbázisokhoz/táblákhoz férjen hozzá), valamint erős jelszóval védjük a felhasználót. Tovább szűkíthetjük a kockázatot SSH tunnel használatával, valamint IP cím szűréssel. Webes alkalmazás révén, a kliens-szerver közötti kommunikáció biztonsága érdekében HTTPS protokoll használatát kell előnyben részesíteni a HTTP protokollal szemben.

A felhasználók jelszava minden esetben SHA256-os titkosítással kerül eltárolásra, mely napjaink egyik legbiztonságosabb megoldása.

## Alkalmazás üzemeltetése

### Szükséges szoftverek

Első lépésként meg kell határoznunk, hogy milyen **operációs rendszer** alatt fogjuk üzemeltetni az alkalmazásunkat. A felhasznált technológiák kiválasztása során nagy hangsúlyt fektettünk a platformfüggetlen megvalósítás lehetőségére, ezáltal mind az adatbázis (mysql), mind a backend (java), valamint a frontend (react js) is elérhető az ismert operációs rendszerekre (Pl.: Windows, Linux disztribúciók, Free BSD). Az operációs rendszer meghatározásakor érdemes valamilyen szerverfuttatásra specializált rendszert választani (Pl.: Windows server, Ubuntu stb.).

Az adatainak tárolására egy **MySql** szerver gondoskodik. Első lépésként a MYSQL szerver 8.0 telepítése javasolt. Ez a fejlesztő hivatalos oldaláról tölthető le az alábbi linkről: <https://dev.mysql.com/downloads/mysql/> illetve a csomagkezelő szoftverekből is le lehet húzni. Az adatbázisunk kezeléséhez használhatunk parancssoros megoldást, ebben az esetben az alábbi módon kapcsolódhatunk: (“C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql.exe” -uroot -ppassword), azonban érdemes valamilyen adatbázis kezelő szoftvert használnunk, mint például az ingyenes mysql workbench, vagy a (nem ingyenes) navicat.

A backendet egy (java) **Spring alkalmazás** szolgálja ki. A fejlesztés során 16-os java verziót használtunk, a garantált működéshez szükséges az említett verziójú JDK letöltése, és installálása, melyet az oracle oldaláról tölthetünk le: <https://www.oracle.com/java/technologies/javase/jdk16-archive-downloads.html>. Más JDK verziók használata esetén az alkalmazást nem teszteltük, ezért nem garantált a megfelelő működés.

A megjelenésért (frontend) **React JS** app felel. Ennek működéséhez Node JS-re van szükségünk, amit <https://nodejs.org/en/> linken szerezhetünk be. A fejlesztés során ebből a 18.2.1-es verziókóddal rendelkező változatot használtuk, működése ez alatt garantált. A Node JS telepítéséhez szükséges további kiegészítők (Pl.: Python) a Node JS telepítésével együtt települ, ha elfogadjuk.

### Hardware követelmények

.

.

.

.

.

.

.

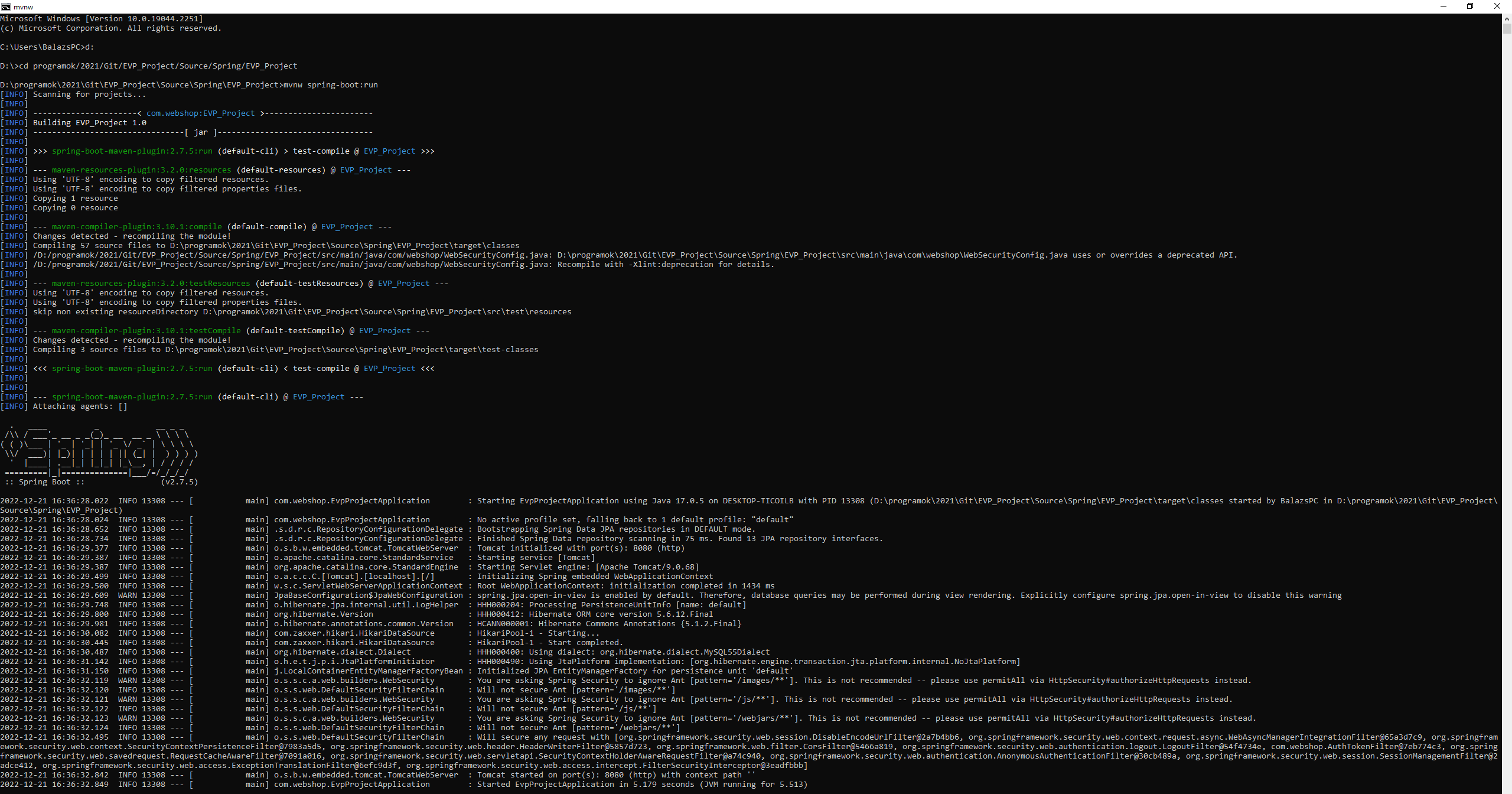
### Beüzemelési javaslat

Beüzemelés első lépéseként javasolt az **adatbázis struktúránk felállítása**. Ehhez az adatbáziskezelő szoftverünkben importáljuk be a webshop.sql állományt, mely tartalmazza az adatbázis létrehozását, a táblastruktúra kialakítását beleértve a táblák közötti kapcsolatokat, valamint az adatbázis által elvégzett adatellenőrzéseket. Második lépésként importáljuk be az insertData.sql állományt is, mely az alkalmazás-jogosultságokat adja hozzá a megfelelő táblához. Ellenőrizzük, hogy minden adatbázis utasítás megfelelően lefutott, és a tábláink megfelelően létrejöttek.

Miután az adatbázisunk létrejött, állítsuk be a Spring alkalmazásunkban annak **elérését**. Ezt az src/main/resources/application.properties fájlban tehetjük meg. Az adatbázisunk elérhetőségét a spring.datasource.url=jdbc:mysql:// után adhatjuk meg „ipcím:port/adatbázisnév” formátumban. Az adatbázishoz tartozó felhasználónkat spring.datasource.username= után adjuk meg, a hozzá tartozó jelszót pedig a spring.datasource.password= rész után. Beüzemelési céllal más itt található konfigurációt ne módosítsunk.

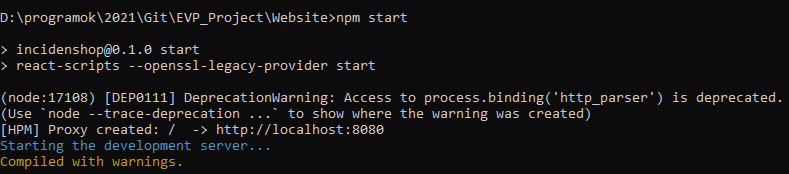


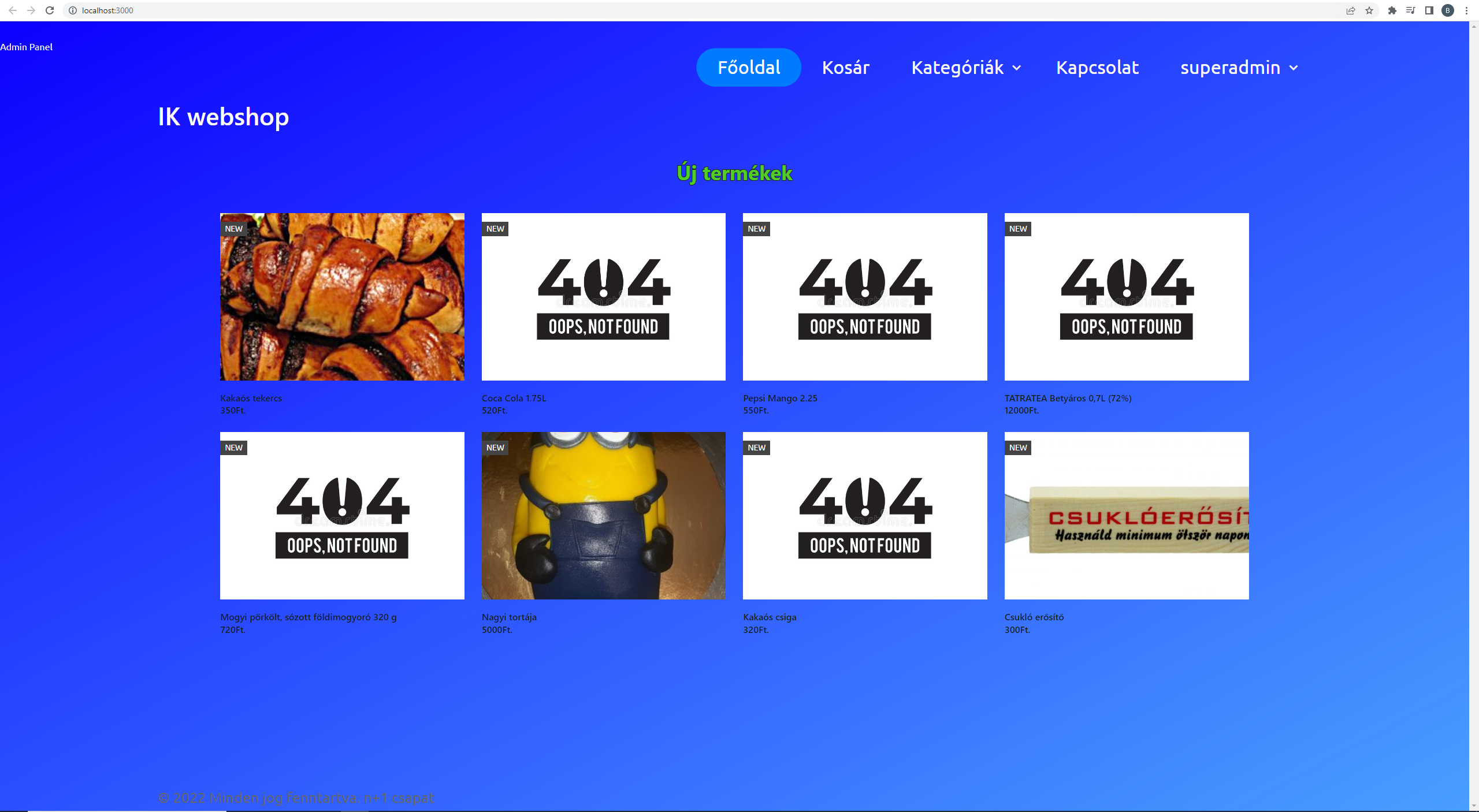
Ha mindent jól csinálhatunk el is indíthatjuk a backend szerverünket, ehhez windows alatt parancssorban, linux alatt pedig terminálban navigáljunk el az alkalmazás főkönyvtárába. majd adjuk ki a „mvnw spring-boot:run” parancsot. Amennyiben mindent jól csináltunk el is indult az alkalmazásunk.



Lehetséges hibák: Az alkalmazás által használt 8080-as port már használatban van: ebben az esetben állítsuk le az azt használó alkalmazást, vagy állítsuk át az alkalmazás portját egy szabad protra, majd a frontendben is adjuk meg az új portot (az utóbbi megoldás nem javasolt azért sem, mert más porton nem teszteltük az alkalmazást, valamint az alkalmazás által használat 8080-as TCP/UDP port egy „nevezetes port”, mely kimondottan controllerek számára fenntartott.).

Az alkalmazás megjelenését szólgáló **React JS app** (**frontend**) elindításához szintén parancssorban/terminálban navigáljunk el a react appunk főkönyvtárába. Első indítás előtt adjak ki itt az „npm i” utasítást, mely letölti a szükséges komponensek legfrissebb verzióját. Miután ezzel végeztünk adjuk ki az „npm start” parancsot, melynek hatására rövidesen el is fog indulni az alkalmazásunk, mely automatikusan meg fog nyílni az alapértelmezett böngészőnkben is (Első indításkor előfordulhat, hogy ez egy kicsit időigényesebb művelet).





### Üzemeltetési javaslatok

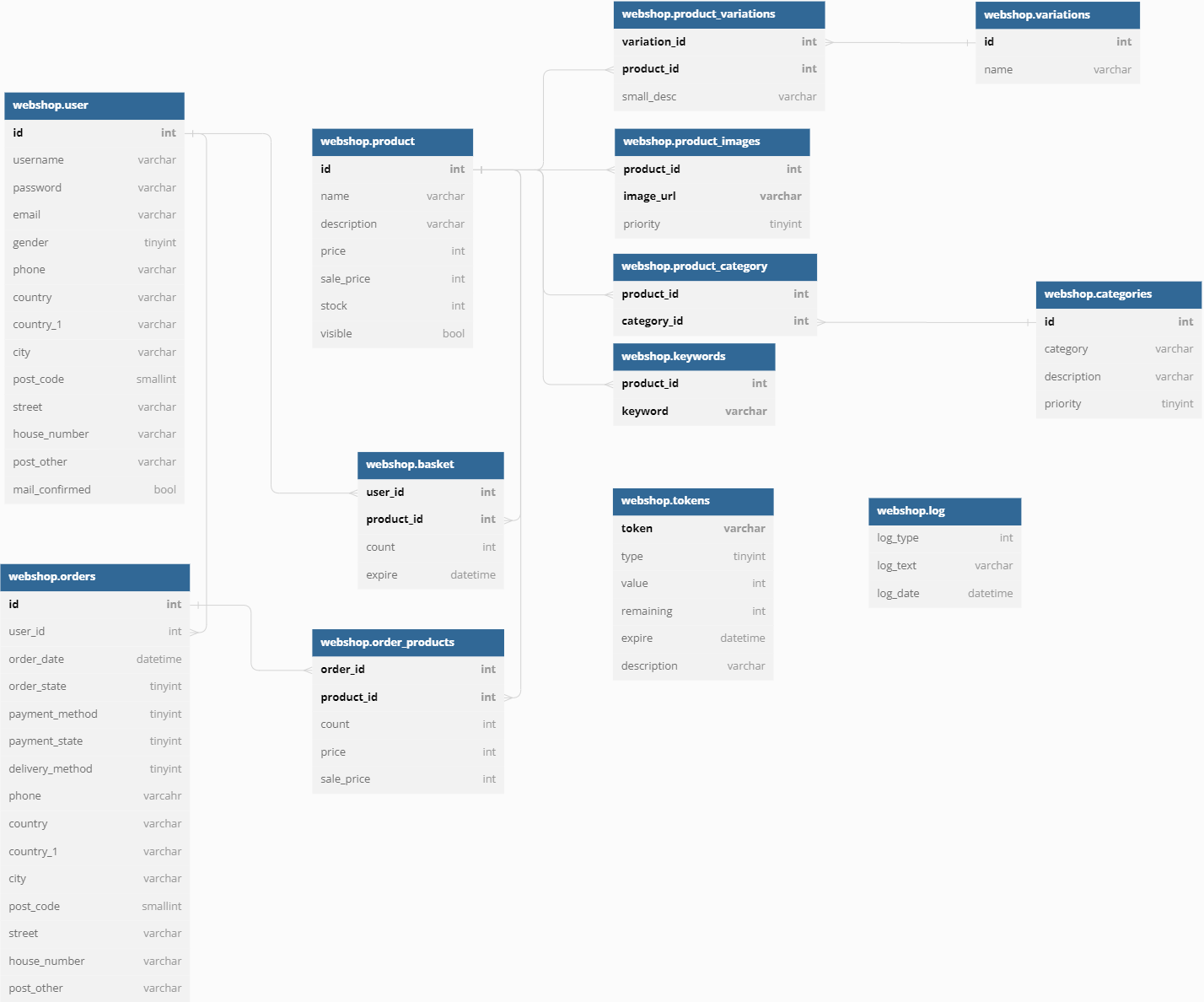
Az alkalmazást javasolt HTTPS protokollon keresztül használni, ugyanis ebben az esetben a kliens-szerver közötti kommunikáció egy úgynevezett kétkulcsos titkosítással zajlik, melyen egy SSL/TLS fölötti HTTP protokollal valósul meg. Ezzel elérhetjük, hogy a kliens-szerver közötti kommunikációt csak a két fél tudja „értelmezni”, így ha esetleg valaki lehallgatná a kommunikációt, akkor se tudja értelmezni annak adatait (különösen fontos a bejelentkezés/regisztráció, személyes adatok, valamint online fizetés esetén).

Az adatbázisunk védelmének érdekében javasolt leszűkíteni az adatbázisfelhasználók hatáskörét, jogosultságait (root felhasználót érdemes letiltani a felhasználók beállítása után). Az adatbázis kapcsolat biztonságát növelhetjük például IP-cím szűréssel, vagy SSH tunnel segítségével, valamint tűzfal használatával. Csak biztonságos gépen, biztonságos internetkapcsolat mellett jelentkezzünk be, mindig jelentkezzünk ki használat után, jelszavunkat ne mentsük el.

## Alkalmazás felépítése, működése

### Adatbázis

Az alkalmazás adatainak tárolásáért felelős adatbázis felépítése az alábbi képen látható:



Bizonyos adatok beszúrása előtt az adatbázis (is) ellenőrzést végez azok helyességéről, melynek célja az adatintegritási problémák elkerülése. Karakterlánc (VARCHAR) típusú adatok esetén minden esetben meghatározott annak maximum hossza, valamint rengeteg mezőre alkalmazva van a NOT NULL módosító, így ezeknek kötelező értéket adni. Ezeken túl ellenőrzött adatok:

* webshop.user. gender: Értéke 1, vagy 2 lehet
* webshop.product.price, webshop.product. sale\_price: Értékük 0-nál nagyobb
* webshop. order\_products.count: Értéke nagyobb, mint 0
* webshop.basket. count: Értéke nagyobb, mint 0

Bizonyos mezők alapértelmezett értéket kapnak, amennyiben nem határozzuk meg azokat külön:

* webshop.product.stock: 0
* webshop.product.visible: true
* webshop.product\_images.priority: 0
* webshop.categories.priority: 0
* webshop.orders.order\_date: NOW() (adatbázis szerver szerinti pillanatnyi dátum-idő)
* webshop.order\_products.count: 1
* webshop.basket.count: 1
* webshop.tokens.value: 1
* webshop.tokens.remaining: -1
* webshop.log.log\_date: NOW() (adatbázis szerver szerinti pillanatnyi dátum-idő)

### Backend (Java Spring app)

#### Összesítő

Az alkalmazás alapvetően egy REST API szerverként funkcionál. Célja, hogy a kliens, és az adatbázis között biztosítsa a biztonságos átjárást. A kliens a szerver controllereihez küldi a kéréseit, melyre választ kap. Az adatbázishoz történő kapcsolódás Spring Data JPA formában történik, mely lényegesen leegyszerűsíti az adatkezelést ORM (Object Relation Mapping) formában. Működését tekintve a repository interfaceket kell létrehoznunk csak az azt leíró modellel, viszont az implementációt a Spring már elvégzi helyettünk. Szükség esetén természetesen hozhatunk létre natív queryket, valamint felülírhatjuk a Spring által definiált kéréseket is.

Jelenleg 4 contoller tartozik az alkalmazáshoz, melyek mindegyike tartalmazza a szükséges adatbázis repokat:

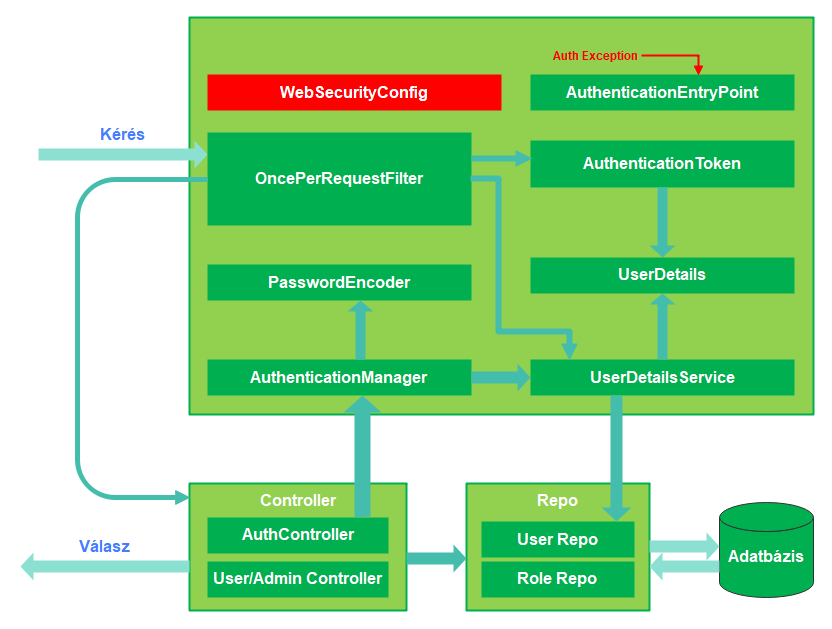
* **AppController**: ipcím:port/api/app/metódus címen várja a hozzá érkező kéréseket. A controller eléréséhez nem szükséges semmilyen jogosultság, és bejelentkezés se. Az összes olyan funkciót tartalmazza, melyhez nincs szükség bejelentkezésre vagy jogosultságra (Pl.: termékek, kategóriák visszaadása; részletek: javadocban).
* **AuthController**: Az autentikációhoz kapcsolódó funkciókat tartalmazza (Ki/Bejelentkezés, Regisztráció). Eléréséhez természetesen nincs szükség jogosultságra.
* **UserController**: Az alkalmazás azon controllere, melyet kizárólag bejelentkezés után lehet elérni. Tartalmazza az összes olyan funkciót, melyre a felhasználónak szüksége van a vásárlási folyamat lebonyolításához. A metódusok pontos részletei megtalálhatóak a kapcsolódó javadoc dokumentációban.
* **AdminController**: Csupán bejelentkezett, ADMIN1, vagy ADMIN2 jogosultsággal rendelkező felhasználó éri el a controller funkcióit. Tartalmazza az adminisztrátorok számára elérhető webshop menedzsment funkciókat. A funkciók részletes leírása, listája a javadoc kapcsolódó részén érhető el.

#### Autentikáció

Az alkalmazás legtöbb funkciójának eléréséhez bejelentkezés szükséges. Az autentikáció JWT tokenes megoldással történik. Az autentikációs folyamatot az alábbi ábra szemlélteti:



A bejelentkezési folyamat feldolgozásának folyamata:



### Adatbázis kezelése

### Controllerek

#### App controller

A controllert az api/app/metódus címen érhetjük el. Tartalmazza azokat a funkciókat, melyek meghívásához jogosultság, és bejelentkezés se szükséges. Mint az összes controller, tartalmazza az adatbázis repokat.

A controller az alábbi kéréseket szolgálja ki (részletes leírást a **javadoc** tartalmaz):

|  |  |  |
| --- | --- | --- |
| **Elérési út** | **Függvény** | **Rövid leírás** |
| **GET**  api/app/testDebug | public ResponseEntity<String> getDebug() | Teszt metódus, Pl.: új funkciók ideiglenes tesztelése |
| **GET**  api/app/productsListing | public ResponseEntity<String> productsListing() | Összes tárgy kilistázása a hozzá tartozó minden adattal (képek, kategóriák, variációk stb.) |
| **GET**  api/app/categoriesListing | public ResponseEntity<String> categoriesListing() | Összes termékkategória listázása |
| **GET**  api/app/findProductsByCategory | public ResponseEntity<String> findProductsByCategory(int categoryId) | Összes termék listázása a megadott kategóriában |
| **GET**  api/app/findProductsByFilterText | public ResponseEntity<String> findProductsByFilterText(String filterText) | Hagyományos keresés, összes termék visszaadása, melynek leírása/neve tartalmazza a megadott karaktersorozatot |
| **GET**  api/app/findProductsByKeywordAny | public ResponseEntity<String> findProductsByKeywordAny(String keywordText) | Speciális keresés kulcsszavak alapján, azokat a termékeket adja vissza, melyhez a megadott kulcsszavak valamelyike (szóköz az elválasztó) tartozik. |
| **GET**  api/app/findProductsByKeywordAll | public ResponseEntity<String> findProductsByKeywordAll(String keywordText) | Speciális keresés kulcsszavak alapján, azokat a termékeket adja vissza, melyhez a megadott kulcsszavak mindegyike (szóköz az elválasztó) tartozik. |

#### Auth Controller

A controllert az api/auth/metódus címen érhetjük el. Az autentikációhoz kapcsolódó műveleteket tartalmazza. Jogosultság nem szükséges a műveletek eléréséhez, valamint a kijelentkezés kivételével a bejelentkezés se. Tartalmazza az összes szükséges repot, valamint a szükséges szolgáltatásokat (service), melyek a PasswordEncoder, a jwtUtils, az AuthenticationManager, az EmailService, valamint a tokenService.

A controller az alábbi kéréseket szolgálja ki (részletes leírást a **javadoc** tartalmaz):

|  |  |  |
| --- | --- | --- |
| **Elérési út** | **Függvény** | **Rövid leírás** |
| **POST**  api/auth/signin | public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest) | Felhasználó beléptetése, JWT token visszaküldése sütikbe, felhasználó adatai (jogosultságok, felhasználónév, email) leküldése. |
| **POST**  api/auth/signup | public ResponseEntity<?> registerUser(@Valid @RequestBody SignupRequest signUpRequest) | Felhasználó regisztrálása. Adatok ellenőrzése több lépcsőfokon. Email, Felhasználónév „unique” adattag. Jelszó hossza min. 6 karakter. E-mail megerősítő mail kiküldése. |
| **POST**  api/auth/signout | public ResponseEntity<?> logoutUser() | Kijelentkezés. Tárolt JWT token törlése sütikből, token „inaktiválása”. |
| **GET**  api/auth/askNewPasswordMail | public ResponseEntity<?> askNewPasswordMail(String email) | Token generálása új jelszó beállításához. Elfelejtett jelszavas E-mail kiküldése a regisztrált E-mail címre, benne a hivatkozással (userId+tokenKey). |
| **GET**  api/auth/confirmMail | public ResponseEntity<?> confirmMail(long userId, String tokenKey) | E-mail megerősítése. Visszairányít a kliensoldalra. (elérése közvetlen szerveroldalra hivatkozás alapján, kliens oldal kihagyásával) |
| **POST**  api/auth/newPassword | public ResponseEntity<?> newPassword(long userId, String tokenKey, String newPassword) | Új jelszó beállítása. Az emailben kapott linkből szükséges hozzá a token, és az id. |

#### User controller

A controllert az api/user/metódus címen érhetjük el. A felhasználókhoz kötődő, jogosultságot nem igénylő műveleteket tartalmazza, azonban bejelentkezés előfeltétele az elérésének. Bejelentkezés hiányában (token) 401-es HTTP errort (unauthorized) fogunk kapni. Tartalmazza az összes szükséges repot, valamint a szükséges szolgáltatásokat (service), melyek az EmailService, a PdfService, valamint a PasswordEncoder.

A controller az alábbi kéréseket szolgálja ki (részletes leírást a **javadoc** tartalmaz):

|  |  |  |
| --- | --- | --- |
| **Elérési út** | **Függvény** | **Rövid leírás** |
| **GET**  api/user/testDebug | public ResponseEntity<String> getDebug() | Teszt metódus, Pl.: új funkciók ideiglenes tesztelése |
| **POST**  api/user/addItemToBasket | public ResponseEntity<MessageResponse> addItemToBasket(long productId, int count) | Termék hozzáadása kosárba |
| **DELETE**  api/user/removeItemFromBasket | public ResponseEntity<MessageResponse> removeItemFromBasket(long productId) | Termék kivétele kosárból |
| **POST**  api/user/updateProductCountInBasket | public ResponseEntity<MessageResponse> removeItemFromBasket(long productId, int newCount) | Kosárban lévő termék mennyiségének módosítása |
| **POST**  api/user/updateDeliveryAddress | public ResponseEntity<MessageResponse> updateDeliveryAddress(String phone, String country, String country\_l, String city, short post\_code, String street, String house\_number, String post\_other) | A felhasználóhoz mentett szállítási adatok módosítása. Amennyiben létezik, automatikusan kitöltődik rendeléskor, de azt módosíthatjuk is. |
| **GET**  api/user/getDeliveryAddress | public ResponseEntity<String> getDeliveryAddress() | Felhasználó szállítási adatai |
| **POST**  api/user/completeOrder | public ResponseEntity<?> completeOrder(String phone, String country, String country\_l, String city, short post\_code, String street, String house\_number, String post\_other, short paymentMethod, short deliveryMethod) | Rendelés leadása. A rendelés leadása feltételekhez kötött: A felhasználó E-mail címe meg lett erősítve, a szállítási adatok megfelelően ki lettek töltve. Minden termékből raktáron van a megfelelő mennyiség, amennyiben valamiből hiányzik, a rendelés nem lesz leadva (tranzakció, vagy minden lefut, vagy semmi). Kosár ürítése, termékek hozzáadása a rendeléshez. Rendeléshez tartozó „számla” generálása, eltárolása, majd kiküldése E-mailben a felhasználónak a megadott E-mail címre. |
| **GET**  api/user/getOrders | public ResponseEntity<?> getOrders() | A felhasználó korábbi rendeléseinek lekérése |
| **GET**  api/user/getOrderProducts | public ResponseEntity<?> getOrderProducts(long orderId) | A felhasználó megadott rendeléséhez tartozó információk (termékek, azok mennyisége, ára stb.) lekérése |
| **POST**  api/user/changePassword | public ResponseEntity<?> getOrderProducts(String oldPwd, String newPwd) | Felhasználó jelszavának megváltoztatása. Feltételként a regisztrációnál is érvényes min. 6 karakter él. |
| **GET**  api/user/getBasketProducts | public ResponseEntity<String> getBasketProducts() | A felhasználó kosárban lévő termékeit adja vissza |

#### Admin controller

A controllert az api/admin/metódus címen érhetjük el. Az adminisztrátori feladatokhoz kötődő, admin1, admin2 jogosultságot igénylő műveleteket tartalmazza. Bejelentkezés hiányában (token) 401-es (unauthorized), jogosultság hiányában pedig 403 (Forbidden) HTTP errort fogunk kapni. Tartalmazza az összes szükséges adatbázis repot.

A controller az alábbi kéréseket szolgálja ki (részletes leírást a **javadoc** tartalmaz):

|  |  |  |
| --- | --- | --- |
| **Elérési út** | **Függvény** | **Rövid leírás** |
| **POST**  api/admin/insertNewProduct | public ResponseEntity<MessageResponse> insertNewProduct(@RequestBody ProductReqRep newProduct) | Új termék rögzítése |
| **POST**  api/admin/updateProduct | public ResponseEntity<?> updateProduct(long productId, @RequestBody ProductReqRep updateProduct) | Termék módosítása |
| **POST**  api/admin/createCategory | public ResponseEntity<MessageResponse> createCategory(String category, String smallDesc, byte piority) | Kategória létrehozása |
| **POST**  api/admin/createVariation | public ResponseEntity<MessageResponse> createVariation(String variation) | Termékvariáció létrehozása |
| **GET**  api/admin/getCategories | public ResponseEntity<String> getCategories() | Kategóriák lekérése |
| **GET**  api/admin/getVariations | public ResponseEntity<String> getVariations() | Termékvariációk lekérése |
| **POST**  api/admin/updateCategory | public ResponseEntity<MessageResponse> updateCategory(long id, String category, String smallDesc, byte priority) | Kategória módosítása |
| **POST**  api/admin/updateVariation | public ResponseEntity<MessageResponse> updateVariation(long id, String variation) | Termékvariáció módosítása |
| **POST**  api/admin/addProductToCategory | public ResponseEntity<MessageResponse> addProductToCategory(long prodId, long catId) | Termék felvétele kategóriába |
| **DELETE**  api/admin/removeProductFromCategory | public ResponseEntity<MessageResponse> removeProductFromCategory(long prodId, long catId) | Termék kivétele kategóriából |
| **POST**  api/admin/addProductToVariation | ResponseEntity<MessageResponse> addProductToVariation(long prodId, long varId, String desc) | Termék felvétele variációkba |
| **DELETE**  api/admin/removeProductFromVariation | public ResponseEntity<MessageResponse> removeProductFromVariation(long prodId, long varId) | Termék törlése variációkból |
| **GET**  api/admin/getAllOrder | public ResponseEntity<?> getOrders() | Összes rendelés lekérése |
| **GET**  api/admin/findOrderById | public ResponseEntity<?> findOrderById(long ordeId) | Rendelés lekérése ID alapján |
| **GET**  api/admin/getOrderProducts | public ResponseEntity<?> getOrderProducts(long orderId) | Rendeléshez tartozó adatok (termékek, azok mennyisége, ára stb.) lekérése |
| **POST**  api/admin/updateOrderState | public ResponseEntity<?> updateOrderState(long orderId, byte newState) | Rendelés státuszának módosítása |