

# **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

Tartalomjegyzék.....	2
Project leírás.....	5
Téma- és technológia választás .....	5
Téma részletesebben kifejtve .....	5
Csapat összetétele, munkamegosztás .....	6
Jegyzőkönyv (1. mérföldkő) .....	6
Korábbi mérföldkő ábrák, tartalmak .....	7
Fejlesztői információk.....	9
Felhasznált nyelvek, technológiák, szoftverek.....	9
Java – Spring keretrendszer.....	9
MySQL 8.0.....	9
React JS .....	10
GitHub.....	10
Használt szoftverek, fejlesztői környezetek .....	10
SpringToolSuite4 .....	10
Navicat .....	11
GitHub Desktop.....	11
Frontend .....	12
Bootstrap .....	12
jQuery.....	12
Font Awesome.....	12
Adatbázis, tárolt adatok, adatvédelem.....	12
Bcrypt.....	13
Alkalmazás üzemeltetése .....	13
Szükséges szoftverek.....	13
Hardware követelmények.....	14
Beüzemelési javaslat .....	14
Üzemeltetési javaslatok.....	16
Alkalmazás felépítése, működése .....	17
Felépítés .....	17
Adatbázis réteg (layer) .....	17
Backend réteg (layer) .....	17
Frontend réteg (layer).....	17
Adatbázis .....	18

Frontend (React JS) .....	19
Összesítő.....	19
Backend (Java Spring app).....	20
Összesítő.....	20
Autentikáció .....	21
Adatbázis – Alkalmazás kapcsolat .....	23
Felhasznált kevésbé ismert csomagok.....	23
spring-boot-starter-mail.....	23
itextpdf, pdfbox .....	24
jackson-datatype-jsr310 .....	24
Controllerek.....	25
App controller .....	25
Auth Controller .....	26
User controller.....	28
Admin controller .....	30
Service, és Config osztályok .....	33
EmailService .....	33
pdfService.....	33
tokenService .....	33
ServiceConfiguration .....	33
WebSecurityConfig.....	33
Egyéb (nem adatbázis táblát leíró) modell osztályok, enumok.....	34
Tesztelés .....	34
Továbbfejlesztési lehetőségek.....	36
Felhasználói kézikönyv .....	37
Oldal felkeresése .....	37
Menüsor.....	37
Főoldal.....	37
Kereső.....	38
Szűrés kategóriákra .....	38
Autentikáció, profil beállítások .....	39
Regisztráció .....	39
E-Mail cím megerősítése.....	39
Bejelentkezés.....	40
Szállítási adatok megadása.....	40

Jelszó megváltoztatása .....	40
Elfelejtett jelszó.....	41
Rendelési folyamat.....	41
Termék kosárba rakása.....	41
Kosár tartalmának megtekintése, kosárban lévő termék törlése, mennyiség módosítása	42
Rendelés leadása .....	43
Adminisztrátori teendők.....	44
Új termék rögzítése .....	44
Rendelések megtekintése, státuszmódosítás .....	44
Felhasználók (és rendelésiek) megtekintése, jogosultságok módosítása .....	45
Új kategória létrehozása .....	46
Új variáció létrehozása .....	46
Termékek lekérése, módosítása.....	47
Kategóriák lekérése / módosítása .....	47
Variációk lekérése / módosítása .....	48
Termék felvétele / törlése kategóriából .....	48
Termék felvétele / törlése variációból .....	49

## Project leírás

### Téma- és technológia választás

A beadandó feladat témájának választásakor mindenképpen olyan projectet szerettünk volna választani, melyből nem csak tanulhatunk, hanem valósághű feladatban piacképes tapasztalatokat is szerezhethetünk. Így esett a választásunk egy **webshop** megvalósítására, amik napról-napra egyre népszerűbbek. A témához kapcsolódóan a megvalósítás módjában kötöttségekről beszélhetünk, melyek bár nem „írott szabályok”, napjainkban a korszerű megvalósítás a modularitás és a praktikum több dolgot is meghatározott. Ezek például a kliens-szerver-adatbázis közötti kapcsolat, melynek megvalósítására a legcélszerűbb választás napjainkban egy API szerver, melyre tetszőlegesen tudunk klienseket fejleszteni. Az API szerveren belül REST API-t (reprezentációs állapotátvitel) választottuk, melynek legfőbb tulajdonsága, hogy a kliens-szerver kommunikáció során az adatok JSON formátumban kerülnek megosztásra a másik féllel. A felhasználható technológiák kapcsán a java nyelv használata kötött volt, ezáltal a backend megvalósítására Spring keretrendszer alatt, a java nyelvre esett a döntés. A frontendre szintén napjaink egyik legmodernebb megoldását választottuk, ezért egy REACT JS app felel. Az API szerverek előnye, hogy a szerverrel http kéréseken keresztül kommunikálhatunk, ezáltal szükség esetén bármilyen platformra egyszerűen fejleszthetünk hozzá klienst.

### Téma részletesebben kifejtve

Egy webshop fejlesztése során nehéz konkrét célokat kitűzni, és a fejlesztés elején meghatározni, hogy pontosan milyen funkciókkal fog bírni, melynek oka, hogy a fejlesztés során folyamatosan előjöhetnek újabbnál újabb ötletek. Nem volt ez nálunk se, ugyanis amikor a témát kiválasztottuk még csak gondolatként se merült fel, hogy például a rendelés leadásakor egy pdf dokumentumot fog előállítani a program, melyet emailben kiküld a felhasználó részére. Ez nem meglepő ugyanis amennyiben megrendelésre készítünk alkalmazást, (jelen esetben gondoljunk valamilyen webshopra) a megrendelővel történő egyeztetés során felsorolja, hogy milyen funkciókat szeretne, viszont itt is felmerülhet, hogy valamit elfelejt, vagy esetleg idő közben (akár rajta kívül álló okokból, pl.: jogszabályváltozás) még valamivel ki kell egészítenünk az alkalmazást.

Amikor az első mérföldkő keretein belül kiválasztottuk az alkalmazás funkcióit, kevesebb, mint fele annyi dolog került rögzítésre, mint amit végül az alkalmazásba integráltunk (beleértve a tervezett funkciók kibővítéseit is).

Kezdeti megbeszélés során tervezett funkciók: bejelentkezés / regisztráció, termékek böngészése, kereső, termékek kosárba rakása, majd rendelés leadása. Adminisztrátorok számára termékek rögzítése, módosítása, rendelések kezelése.

Ezekkel ellentétben még az alkalmazást kiegészítettük például email cím megerősítéssel, vagy épp digitális „számla” kiállítással, és kiküldéssel, valamint rengeteg tervezett funkciót egészítettünk ki, fejlesztettünk tovább (pl.: 3 féle kereső).

### Csapat összetétele, munkamegosztás

Csapatnévnek az „n+1.csapat” nevet választottuk, és jelenleg három tagunk van, a csapatösszetétel szerencsésnek mondható.:

1. **Harkai Martin:** Frontend felelős; React js tapasztalatok, kliens oldali problémák megoldása
2. **Rába Krisztián:** Design, és műszaki megoldások; Alkalmazás megjelenésének tervezése, megvalósítása, hibakeresés, tesztelés
3. **Vass-Horváth Balázs:** Backend, és adatbázis felelős; Szerveroldali problémák kivitelezése, adatbázis tervezés, és megvalósítása

Ahogy a fenti leírásban is jól látszik, az alkalmazás fejlesztésének minden részét sikerült lefedni, ugyanis a csapat összetételének köszönhetően minden témakör lefedésére van tapasztalt, a témában jártasabb csapattagunk, mindazonáltal az elsődleges célunk most a tanulás, és a tapasztalat szerzés volt, ezért igyekeztünk mindig közösen dolgozni mindenben, valamint a főbb modulok tervezése kivétel nélkül a teljes csapat részvételével történt. Köszönhetően a 6 éves ismeretségnek, és a korábbi közös tanulásoknak, projecteknek, valamint a folyamatos kommunikációnak kiváló volt az összhang, és problémamentesen tudtunk együtt dolgozni.

A csapatunk weboldala: <https://kgaming.ddns.net/suli/a/>

### Jegyzőkönyv (1. mérföldkő)

A megbeszélésünk első 10 percében a projecthez nem tartozó témákról beszéltünk. Ezt követően megbeszéltük a heti feladatok beosztását. Rába Krisztián vállalta a project weboldalának elkészítését, míg Harkai Martin, és Vass-Horváth Balázs közösen a prezentáció, és a jegyzőkönyv elkészítését vállalták. Vass-Horváth Balázs létrehozott Githubon egy projectet, amely végig fogja kísérni az egész féléves munkánkat. Harkai Martin, és Rába Krisztián is csatlakoztak a repohoz, a meghívásuk e-mailes formában történt.

Megbeszélésünk második pontjának fő témája a weboldal pontos megbeszélése volt. Mivel Rába Krisztián vállalta ennek elkészítését, ezért ő mondta el elsőként az ötletét.:

-Mi lenne ha kék témával csinálnánk egy egyszerű, átlátható weblapot, ami elegendő lenne az órai feladatra? Valami könnyű kellene, hogy egy raspberry ki tudjon szolgálni egyszerre több klienst. – Kérdezte Krisztián elmélyülve a feladaton.

-Biztos, hogy elég lesz az? - kérdezte Martin.

-Biztos, elvégre wordpress weboldal is felmerült órán, mint elfogadott opció. - mondta álmosan Balázs.

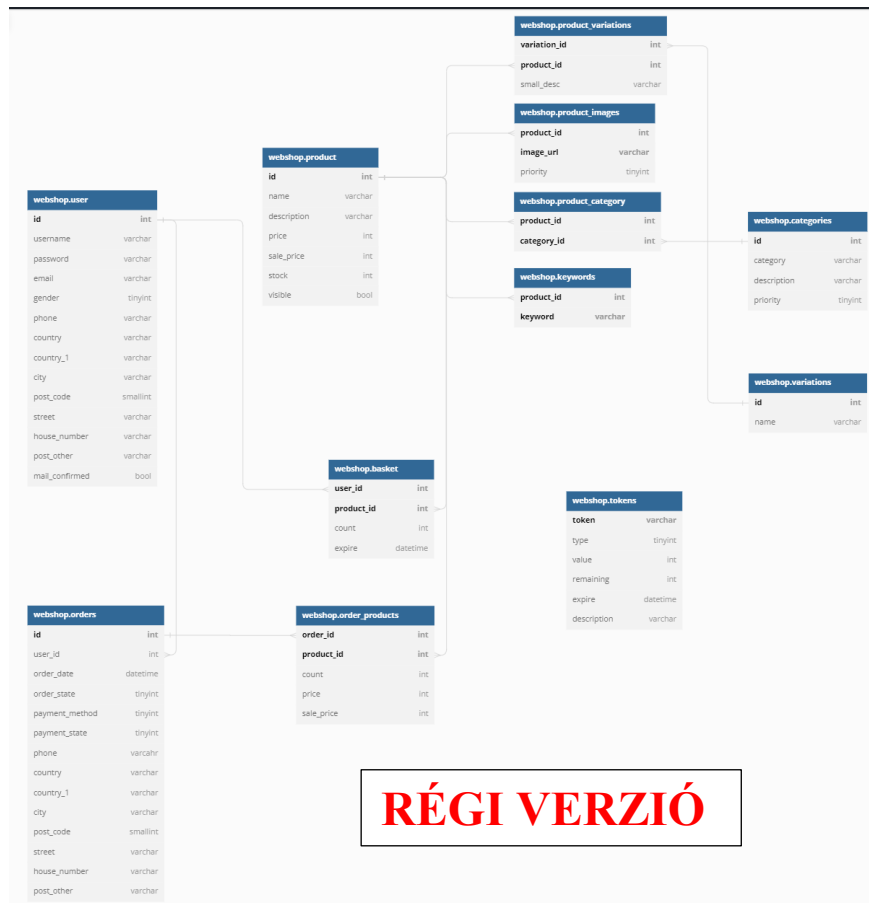
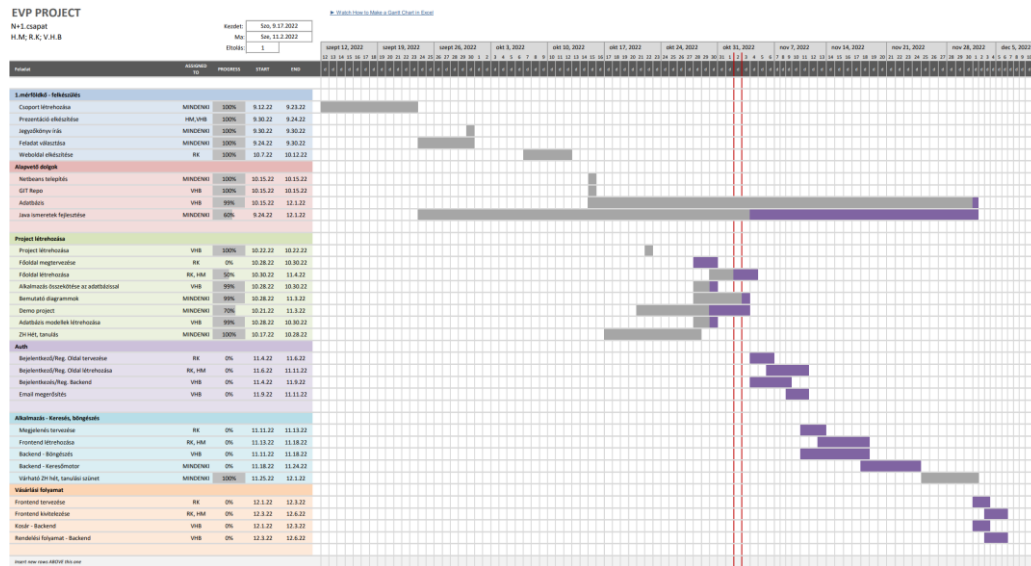
Mivel Harkai Martin, és Vass-Horváth Balázs kevésbé szeretnek weboldal tervezéssel, és designolással foglalkozni, ezért elfogadták Rába Krisztián felvetését.

A megbeszélésünk harmadik témájaként a beadandó szoftver került szóba. Elsőként a feladat leírásait, és követelményeit néztük át. Szomorúan fogadtuk, hogy Java nyelven kell megvalósítani, ugyanis szívesebben dolgoztunk volna C# nyelven, ASP.NET környezetben. Röviden összefoglaltuk, hogy a végső megvalósításhoz milyen technológiákat fogunk használni. Adatbázisként MYSQL relációs adatbázis mellett döntöttünk egyhangú közös megegyezés alapján. Szintén közös megegyezés alapján a projectet a legmodernebb



# Eseményvezérelt programozás: Budget Webshop

## n+1.csapat



**RÉGI VERZIÓ**



## 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, amikor is még a Sun Microsystems fejlesztette. 2009-ben az Oracle cég tulajdonába került egy felvásárlás eredményeképpen, és napjainkban is ők felelnek a fejlesztéséért. Érdekes, 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\\_\(programoz%C3%A1si\\_nyelv\)](https://hu.wikipedia.org/wiki/Java_(programoz%C3%A1si_nyelv))

Spring - [https://hu.wikipedia.org/wiki/Spring\\_keretrendszer](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. Alternatívaként 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 úgy mond 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 beírni az npm install-t, hogy lehetőleg elkerüljük az incidenseket.

## GitHub

A GitHub a Microsoft egy leányvállalataként a verziókezelés, verziókövetés lehetőségét biztosítja. A projekt munka 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 utolsósorban 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 virtuális adat tárolására, nyomon követésére. A GitHub továbbá lehetőséget nyújt számunkra kisebb weboldalak tárolására, valamint kódjaink publikálására, és még megannyi hasznos funkcióra, melyeket szinte a végtelenségig lehetne sorolni. Alternatívaként rengeteg lehetőséget felsorolhatnánk, 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

## Eseményvezérelt programozás: Budget Webshop n+1.csapat

modulokat építhetünk be a projektünkbe automatikusan, nem kell ezt manuálisan elvégeznünk. Teljes értékű java fejlesztői környezet is, mivel Eclipse 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.

## 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 létrehozásához, 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 „szöveggént” 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 tárolt 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 csökkenthetjük 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 bcrypt titkosítással (saltot is magában foglalja) kerül eltárolásra, mely napjaink egyik legbiztonságosabb megoldása.

### Bcrypt

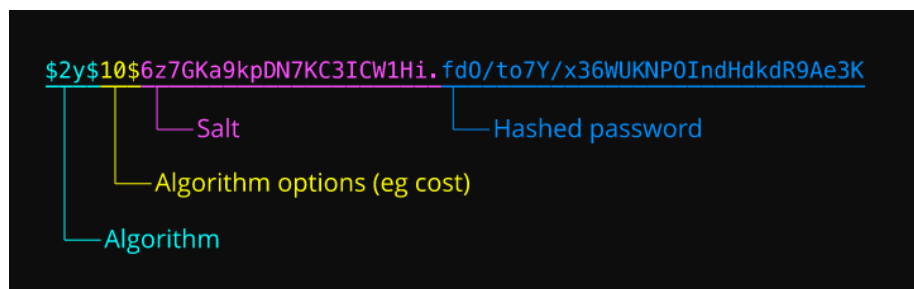
A Bcrypt napjaink egyik legelterjedtebb, legbiztonságosabb megoldása (ezt alátámasztja, hogy Open/FreeBSD, valamint bizonyos linux disztribúciókon, mint például SUSE is ezt használják alapértelmezett jelszókódolásként). A titkosítás kiválasztásakor nem volt könnyű dolgunk, ugyanis rengeteg hasonlóan biztonságos vetélytársa van (Pl.: SHA – 1, 256, 512 stb.), valamint vannak napjainkban már kevésbé biztonságos titkosítások is (Pl.: md5). A Bcrypt mellett szólt, hogy könnyen integrálható Spring környezetbe, mindezek mellett pedig minden esetben magában foglalja a jelszó „sózását” (salt-ozás), mely annyit jelent, hogy a jelszóhoz minden mentéskor hozzárendel egy véletlenszerűen generált karaktersorozatot, ezáltal ha a felhasználó ugyan azt a jelszót adja meg, vagy több felhasználónak egyazon jelszava van, a titkosított formátumuk akkor is eltér.

Az eltárolt adat formátuma \$hash algoritmus azonosító (bcrypt esetén 2a)\$algoritmus erőssége\$salt 22karakter` `hash-elt jelszó`

Konkrét példa:

\$2a\$10\$uAVO0AnPU8bYK9ryWXML5.mho2OUNBRUnLnPmmKSS8f9YOd1DgagO

- \$2a\$10\$ - bcrypt algoritmus azonosítója, algoritmus erőssége ( $2^{10}$ )
- uAVO0AnPU8bYK9ryWXML5 – salt
- mho2OUNBRUnLnPmmKSS8f9YOd1DgagO hash-elt jelszó



### 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

Eseményvezérelt programozás: Budget Webshop  
n+1.csapat

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 szerezhethetünk be. A fejlesztés során ebből a 18.2.1-es verziókéddal 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

- 32GB RAM
- 10GB/s stabil internetkapcsolat
- 10TB háttértár, dinamikusan bővítve
- 4.0GHZ 8mag/16szál

#### 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.

```
#JPA db conn
spring.datasource.url=jdbc:mysql://kgaming.ddns.net:3306/webshop
spring.datasource.username=myadmin
spring.datasource.password=mintaPwd
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.format_sql=true
```

Az alkalmazás E-mail-t küld a felhasználónak, SMTP-n keresztül. Tesztelés esetén ezt a google (gmail) szerverén bonyolítottuk le. Annak érdekében, hogy leveleket küldhessünk, be



# Eseményvezérelt programozás: Budget Webshop

## n+1.csapat

kell állítanunk a levelező rendszerünk elérhetőségét, melyet szintén az application.properties fájlban tehetünk meg. Amennyiben gmailt szeretnénk használni (javasolt a kétlépcsős azonosítás használata, majd alkalmazás jelszó generálása), abban az esetben csak a felhasználónevünket, illetve a jelszavunkat (alkalmazásjelszóra) kell módosítanunk.

Alkalmazás jelszó beállításáról részletesebben:  
<https://support.google.com/accounts/answer/185833?hl=hu>

```
#SMTP mail sender
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=evpwebshop
spring.mail.password=mintaPwd
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

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.

[illegible]

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 portra, 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 szolgáló **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

## Eseményvezérelt programozás: Budget Webshop n+1.csapat

fog indulni az alkalmazásunk, mely automatikusan meg fog nyílni az alapértelmezett böngészőnkben is (Első indításkor előfordulhat, hogy ez egy kicsit időigényesebb művelet).

```
D:\programok\2021\Git\EVP_Project\Website>npm start

> incidenshop@0.1.0 start
> react-scripts --openssl-legacy-provider start

(node:17108) [DEP0111] DeprecationWarning: Access to process.binding('http_parser') is deprecated.
(Use `node --trace-deprecation ...` to show where the warning was created)
[HPM] Proxy created: / -> http://localhost:8080
Starting the development server...
Compiled with warnings.
```

### 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).

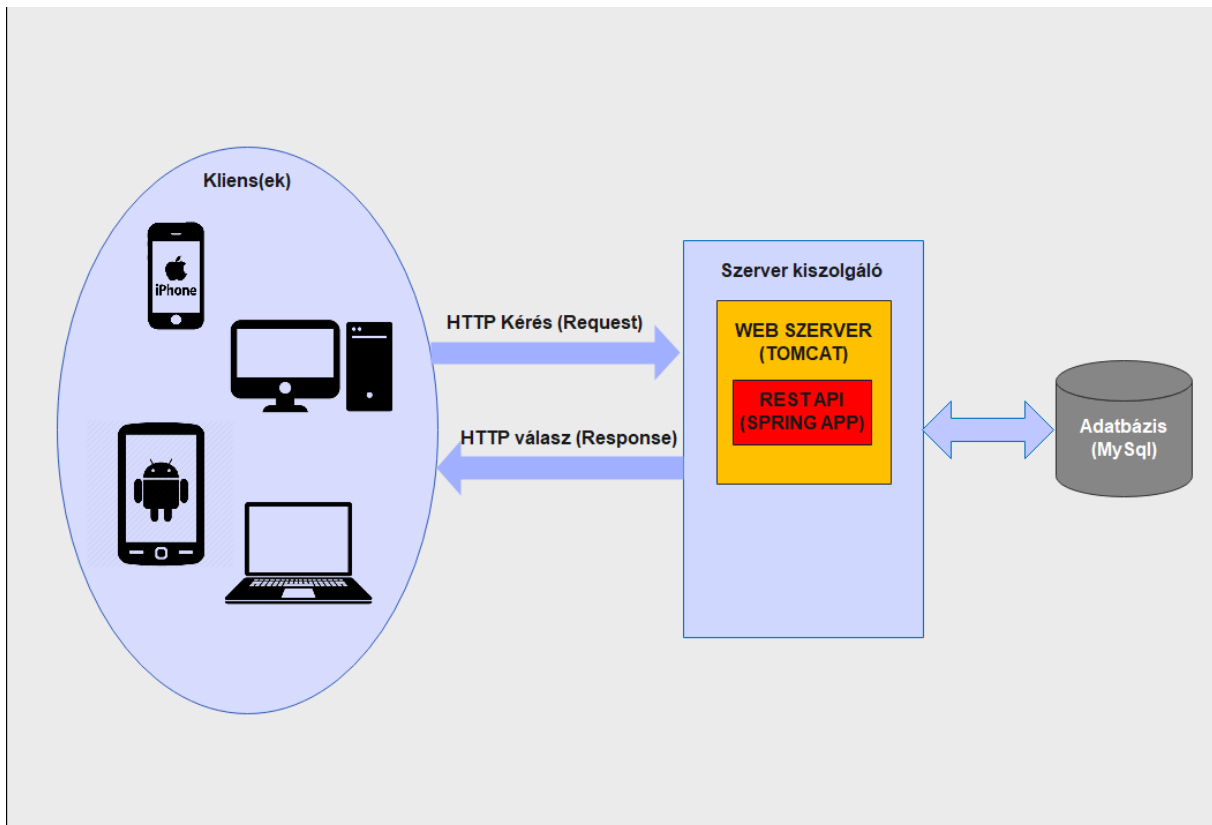
Az adatbázisunk védelmének érdekében javasolt leszűkíteni az adatbázisfelhasználók hatáskörét, jogosultságait (root felhasználót érdemes letiltani a felhasználók beállítása után). Az adatbázis kapcsolat biztonságát növelhetjük például IP-cím szűréssel, vagy SSH tunnel segítségével, valamint tűzfal használatával.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



## Alkalmazás felépítése, működése

### Felépítés

Az alkalmazásnak három fő rétege van. Az adatbázis, a Backend (szerver), és a Frontend (kliens). A felhasználók ezek közül a rétegek közül csak a frontendet látják. Esetünkben a frontend szerepét egy React JS alkalmazás, a backendet egy spring rest API, az adatbázist pedig egy MySql server látja el. A rétegek közötti kommunikációt az alábbi ábra szemlélteti:



A rétegek feladata funkciójukat tekintve:

### Adatbázis réteg (layer)

Az adatbázis feladata a felhasználók, és a webshop termékeinek biztonságos, rendezett tárolása. Az adatbiztonság érdekében ezt a felhasználók közvetlenül semmilyen módon nem érhetik el. Természetesen az adatbázis a szerveroldali rétegek közé tartozik.

### Backend réteg (layer)

A backend (szerver) legfőbb célja, hogy biztosítsa a kliens, és az adatbázis közötti biztonságos kapcsolat kialakítását. Segítségével garantálható, hogy az adatbázisba csak ellenőrzött adatok kerüljenek be, valamint a tárolt adatokhoz csak az arra jogosult kliensek (felhasználók) férjenek hozzá. A réteg természetesen szerveroldali réteg.

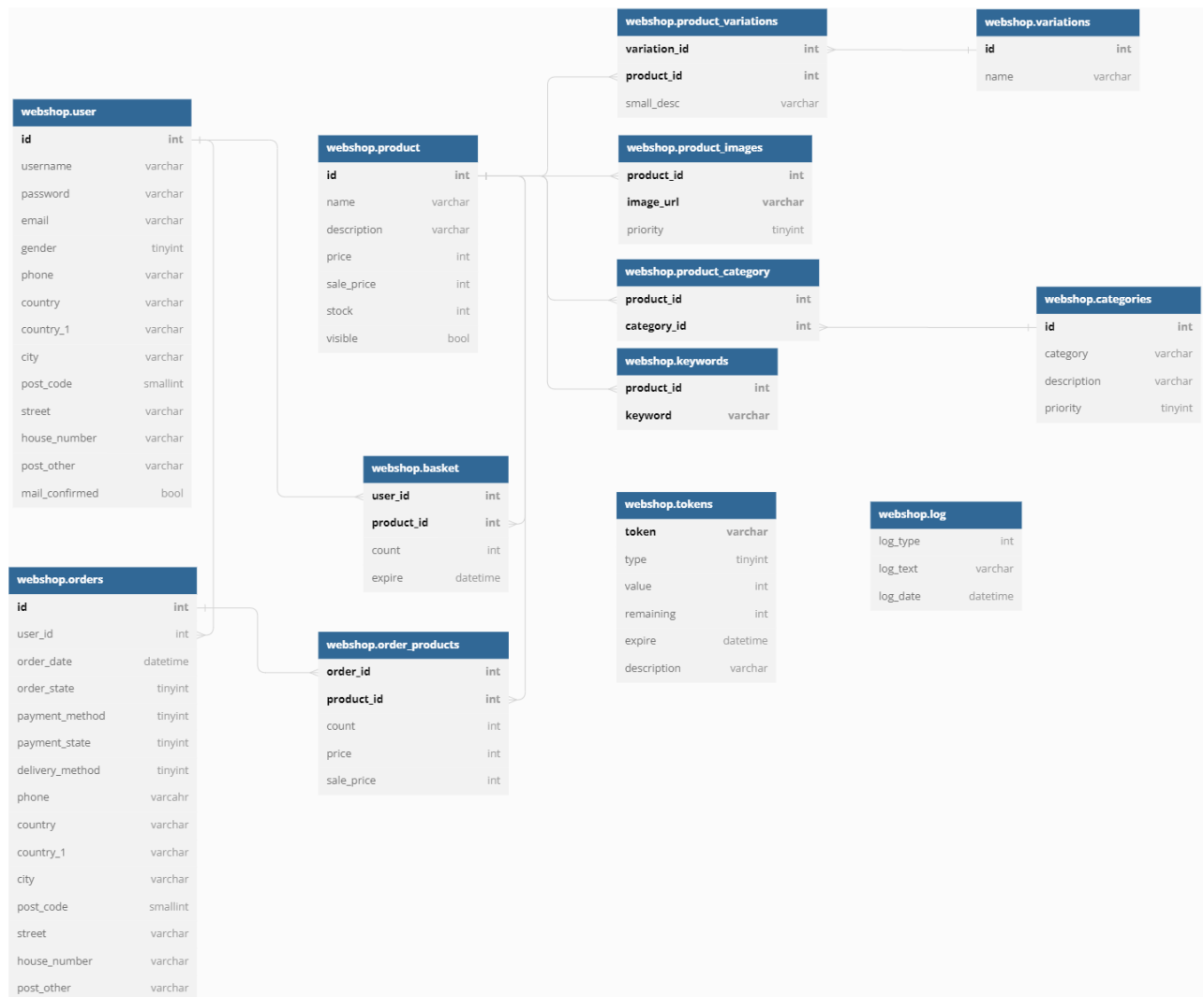
### Frontend réteg (layer)

A frontend (kliens) feladata, hogy a felhasználó számára értelmezhető, és használható módon megjelenítse a szükséges adatokat, valamint hogy a felhasználó a szerver felé kéréseket küldjön az adatok kezelését illetően, majd a kérésre kapott választ a felhasználó számára megjeleníti.

## Eseményvezérelt programozás: Budget Webshop n+1.csapat

### Adatbázis

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



Bizonyos adatok beszurása 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 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 szerinti pillanatnyi dátum-idő)

## Frontend (React JS)

### Összesítő

Az alkalmazásnak a frontend részét a React-ra bíztuk, hiszen számos könyvtárat támogat, amelyek egyszerűbbé teszik a kliens műveleteket, de még a back-endel való kommunikációt is. Az App.js file nekünk a „gyökér” file-unk, amiben el vannak helyezve komponensek. (Igazából az index.js a főgyökér, mert ebben van az App komponens) Itt lett megépítve a routolás, ami tulajdonképpen azt mondja meg, hogy bizonyos elérési útvonalon, milyen tartalmak jelenjenek meg a felhasználónak, de még azt is szabályozza, hogy az adott jogosultsággal rendelkező fél(pl: Felhasználó) elérheti-e az adott útvonalat az oldalon. Egyébként a routing-ot a react-router-dom külsős javascript csomaggal lett elintézve, amit a node csomagkezelőjével installálhatunk. (Megtalálhatjuk a további csomagokat a node\_modules mappában). Megtalálhatunk imporált stíluslapokat is az App.js-ben, ami az egyes komponenseket teszi szebbé. Az App.css és az index.css-ben leírt stíluslap a globális stílusért felel. A MobileMenu.js osztály a mobilos nézetért felel. A setupProxy.js-ben pedig a middleware van beállítva. A main\_page mappába belekukkantva megtalálhatjuk a főoldalért felelős js fileokat. Eleinte a fejléc és a főoldal teste 2 külön rész volt, de technikai okok miatt, egybe kellett paszírozni őket, ezért a ProductComponent.js-ben találhatjuk meg, amit igazából először betöltünk az eszközünkre. Ebben a függvényben jelennek meg igazából a Reactnak a funkciói(usestate, useeffect), számos állapotokat(pl termékek) tudunk beállítani, aminek az egyik előnye, hogy, ha megváltozik az állapota, akkor az oldalunkat nem kell újra betölteni, ez pl a keresésnél jöhet elő. A useeffect függvénybe vannak bepakolva a lekérések(termékek, kategória), amelyek aszinkron futnak le(a legtöbb lekérés ilyen módszerrel fut), amíg a lekérések feldolgozás alatt vannak, addig jelezzük a kliensnek, hogy pl a termékek betöltés alatt vannak. Több helyen is igénybe vettük a JSX egyik feature-t, ami tulajdonképpen a feltételes megjelenítést jelentené, ha magyarítanánk(conditional rendering), ez azért is lehet jó, mert ugye nem minden terméknek lehet képe és, ha azt megakarnánk jeleníteni, akkor hiba lépne fel. A CRUD műveleteket az npm által támogatott axios-al lett megoldva, hiszen támogatja az összes műveletet(létrehozás, olvasás, frissítés, törlés), amikre nekünk szükségünk lehet egy webshopos alkalmazásban, ráadásul le is egyszerűsíti azokat. Az axios logikáját a services mappában lehet megtalálni, több fájlra szétbontva (ProductService.js, UserService.js, PayService.js, BasketService.js, AuthService.js, AuthHeader.js). Tovább haladva a többi komponensre, elmondhatjuk, hogy általában minden felületnek (főoldal, bejelentkezés, kapcsolat) különböző Header fájlja van. A header(fejléc) segítségével tud könnyen navigálni a felhasználó az oldal különböző pontjaira. Ellentétben a lábléc-el, ugyanis az oldal legalján szinte az összes elem statikus(Footer.js), ami azt jelenti nekünk, hogy csak egy ilyen komponens létezik. A footerben megtalálhatjuk a szokásos copyrights szöveget és nyilván a csapatunknak a nevét. A főoldal struktúráját ezek a fileok építik fel nekünk összességében. A bejelentkezés és regisztráció logikáját a login\_page mappában találhatjuk

meg, amit a login.js és a register.js reprezentál. Egy fiatalos, baráti megjelenést alkalmaztunk, a stílust a css mappában tároltuk. A betű stílus kódját, pedig a fonts mappában tároljuk, de még hozzájárulnak a megjelenéshez az images és a vendor mappák. A forgetPassword.js fájl rendezi el nekünk az új jelszó beállítását a felhasználóhoz, ha esetleg elfelejtene, de párba állhat vele a newPassword.js is, hiszen ezzel is új jelszót tud beállítani az ember. Az email cím megerősítésének a visszajelzését az email\_page mappában találhatjuk, ami igazából 2 fájlt tartalmaz a mailConfirmFailed.js-t és a mailConfirmSuccess.js-t, feltétel szerint kapja meg a személy az adott oldalt, ha sikertelen a megerősítés, akkor a mailConfirmFailed.js-t jelenítjük meg(megerősítés sikertelen), ha pedig sikertelen, akkor a mailConfirmSuccess.js-t(megerősítés sikeres) varázsoljuk elő a megfigyelőnek. Az error\_page-ben figyelhetjük meg a 404-es hiba oldalért felelős 404error.js-t, ebben a mappában is szerepel css, 404.css néven, de még akár mozgóképeket is találhatunk(gif), egyébként a 404error.js-t, akkor jelenítjük meg, amikor a felhasználó rossz útra tér az alkalmazáson belül(olyan útvonalat akar elérni, amit nem talál az alkalmazás) pl: /incidenskar útvonal elérése esetén. Előre cammogva, a contact\_page mappában vehetjük szemügyre a Kapcsolatért felelős Contact.js fájlt, ami tulajdonképpen egy szimuláció csak, de jövőben elképzelhető, hogy a felhasználó ténylegesen fel tudja venni velünk a kapcsolatot, szimuláció alatt azt értjük, hogy adatai megadása után, csak egy alert függvény fut le a weboldalon. A user mappában vannak a felhasználói adatok, pl profile.js és Orders.js és a Board fileok(ezek döntenek el, hogy milyen tartalom jelenjen meg jogosultság alapján). A profile.js az adott felhasználónak a profilját jeleníti meg, ahol a megrendelő megfigyelheti pl a szállítási adatait, de még az email címét is. Ha kívánja a jelszavát is megtudja változtatni. Az Orders.js fájlban tekintheti meg az eddigi rendeléseit, egy adott rendeléshez megnézheti a rendelt termékeket és a további adatait. Egyéb fájlok közé sorolható ebben a mappában az EventBus.js, ami az események kezeléséért szolgál. A cart fájl hordozóba belenézve megfigyelhető a ShoppingCart.js, ami magától értődik, hogy a kosár oldalért felel. Ez a komponens felel a felhasználónak a kosaráért, tudja módosítani tartalmát, pl, hogy mennyi terméket rendeljen, de akár törölni is tud belőle. A kosár oldal alján jelenik meg a kliensnek a végösszeg, amit minden frissítésnél újraszámolunk! A kosár oldalról lett linkelve a fizetés oldal(a felhasználó csak innen éri el ezt az oldalt, máskülönben vissza irányítjuk a kosár oldalra), ami a payment mappában található és a Payment.js felel érte. Itt a kliens a személyes adatai kitöltése után tudja rögzíteni a rendelését, egyébként az adatok helyességét jelezzük a felhasználónak egy alert szövegdozban. A components mappában egyéb fájlok találhatóak meg, amik az oldal megjelenéséért és az adatok helyes struktúrájáért felel. Összességében ennyi lenne a front-end rész

## 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 kontrollereihez 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 controller tartozik az alkalmazáshoz, melyek mindegyike tartalmazza a szükséges adatbázis repokat:

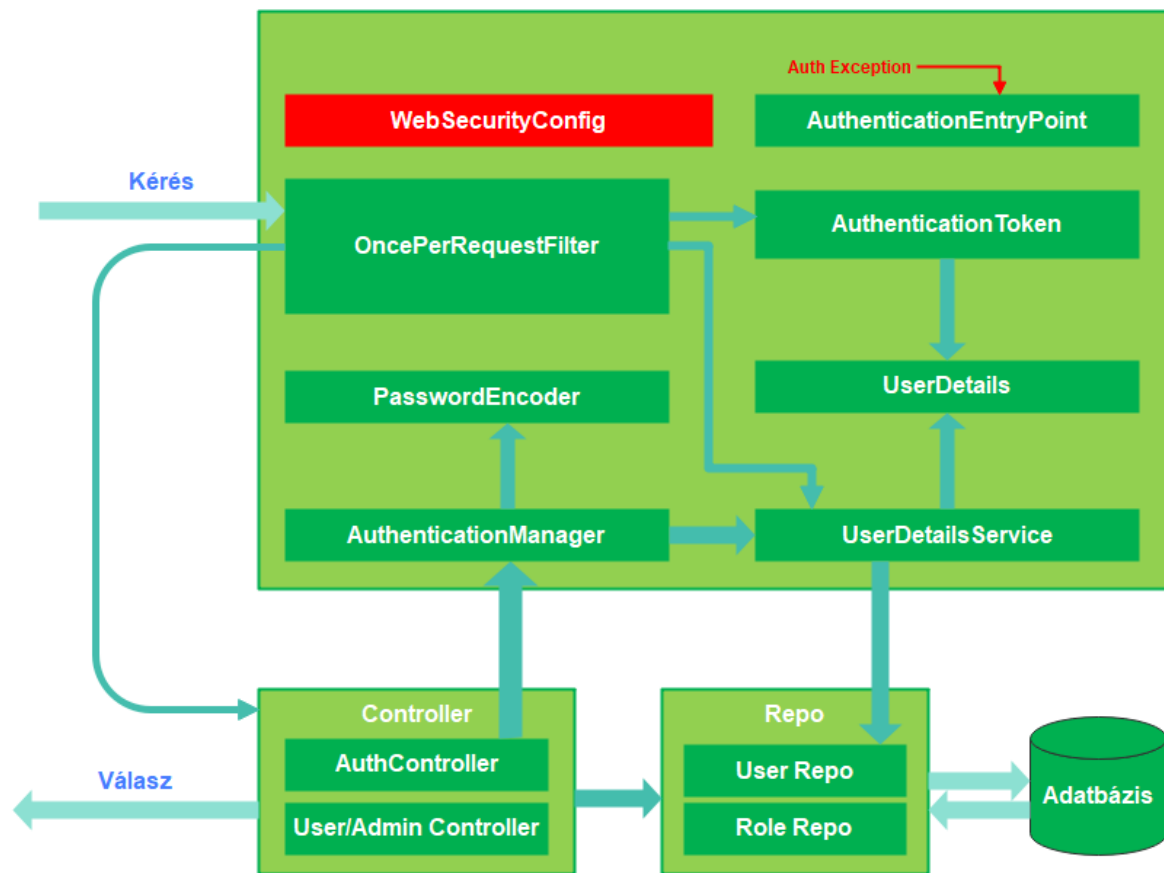
- **AppController:** ipcím:port/api/app/metódus címen várja a hozzá érkező kéréseket. A controller eléréséhez nem szükséges semmilyen jogosultság, és bejelentkezés se. Az összes olyan funkciót tartalmazza, melyhez nincs szükség bejelentkezésre vagy jogosultságra (Pl.: termékek, kategóriák visszaadása; 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, melyhez spring csomagokat használtam fel (spring-boot-starter-security, spring-boot-starter-validation, jjwt). Az autentikációs folyamatot az alábbi ábra szemlélteti:



A bejelentkezési folyamat feldolgozásának folyamata:



### Adatbázis – Alkalmazás kapcsolat

Az alkalmazás adatbázist használ. A rétegek közötti kapcsolat JPA segítségével valósul meg, mely végtelenül leegyszerűsíti az adatbázis kezelést modell alapú működésével. Ez gyakorlatban annyit jelent, hogy modell osztályokat kell létrehozunk, amely leírja az adatbázis táblánkat (egyedek), majd egy repository segítségével már el is érjük a kódunkat. A repository lényege, hogy csak egy interface-t hozunk létre, majd ez alapján automatikusan „létrejön” az implementáció. Szükség esetén definiálhatunk saját query-ket is, nem feltétlenül szükséges az így létrejött mudoulokat használnunk, melynek főleg komplex lekérdezések esetén van létjogosultsága. Az adatbázis kapcsolatot a beüzemelési javaslatok menüpont alatt olvasható módon kell beállítani. Amennyiben egy táblában az elsődleges kulcsunk több mezőből tevődik össze, abban az esetben sajnos egy kulcs osztályt kell létrehozunk, majd a modellünknek ezt jeleznünk kell, ugyanis alaphoz csak egy tagból álló kulcsot tud kezelni.

Az alkalmazásban az adatbázist leíró modellek, repository-k, kulcs osztályok listája:

Adatbázis tábla	Táblát leíró „modell” osztály	Repository (interface)	Kulcs osztály
<b>user</b>	User	UserRepository	-
<b>orders</b>	Orders	OrdersRepository	-
<b>product</b>	Product	ProductRepository	-
<b>basket</b>	Basket	BasketRepository	BasketId
<b>order_products</b>	OrderProducts	OrderProductsRepository	OrderProductsId
<b>product_variations</b>	ProductVariations	ProductVariationsRepository	ProductVariationsId
<b>product_images</b>	ProductImages	ProductImagesRepository	ProductImagesId
<b>product_category</b>	product_category	ProductCategoryRepository	ProductCategoryId
<b>keywords</b>	Keywords	KeywordsRepository	KeywordsId
<b>tokens</b>	Tokens	TokensRepository	-
<b>variations</b>	Variations	VariationsRepository	-
<b>categories</b>	ProductCategories	ProductCategoriesRepository	-
<b>log</b>	Log	LogRepository	-

### Felhasznált kevésbé ismert csomagok

#### spring-boot-starter-mail

Az alkalmazásból lehetőségünk van E-mailt küldeni a felhasználóink (vagy bárki) számára. Az alkalmazásban ezt a funkciót jelenleg az E-mail megerősítésére, elfelejtett jelszó esetén új jelszó beállítására, valamint leadott rendelések esetén a számla kiküldésére használjuk. A csomag 2.5.6-os verziója került „beimportálásra”, mely 2021 októberében jelent meg. Jelenleg elérhető már a 3.0.1-es verzió is, de ezt nem egész 1 hete tették elérhetővé, valamint

Eseményvezérelt programozás: Budget Webshop  
n+1.csapat

a 3-as főverzió sem egész 1 hónapja jelent meg, ezért még nem tartom kizártnak, hogy hibákat tartalmaz, ezért nem használtam. A csomag apache license 2.0 alatt érhető el, részletesebb információkat a <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-mail> linken találhatunk róla. A tesztelés során a gmail szerverein keresztül használtuk az alkalmazást, a beállításáról részletesebben a beüzemelés menüpont alatt olvashatunk.

[itextpdf, pdfbox](#)

Egy webshop esetén (is) a vásárlást követően a felhasználó számára számlát kell kiállítani. A digitálisan kiállított számla esetén fontos, hogy a felhasználó ne tudja azt módosítani, ezért PDF formátumban érdemes azt eljuttatni számára. A PDF dokumentum létrehozásához itextpdf csomagot használtam fel, erről részleteket a <https://search.maven.org/artifact/com.itextpdf/itextpdf> linken találunk. A csomag GNU AFFERO GENERAL PUBLIC LICENSE alatt érhető el. Segítségével nagyon egyszerűen állíthatunk elő pdf dokumentumot, valamint menthetjük el azt az alkalmazást futtató hardware merevlemezére.

[jackson-datatype-jsr310](#)

Javában alapvetően a JSON formátumra alakítást (JSON parse), illetve a JSON formátumból visszaalakítást az ObjectMapper osztály segítségével végezhetjük, azonban sajnos ez az osztály alpból nem képest a LocalDateTime (összetett) adattípust át, valamint visszaalakítani, ezért szükségünk van erre a kiegészítő csomagra. Működés részletes leírása a hozzá tartozó config osztály leírásánál.



### Controllerek

#### App controller

A controllert az api/app/metódus címen érhetjük el. Tartalmazza azokat a funkciókat, melyek meghívásához jogosultság, és bejelentkezés 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
<b>GET</b> api/app/testDebug	public ResponseEntity<String> getDebug()	Teszt metódus, Pl.: új funkciók ideiglenes tesztelése
<b>GET</b> api/app/productsListing	public ResponseEntity<String> productsListing()	Összes tárgy kilistázása a hozzá tartozó minden adattal (képek, kategóriák, variációk stb.)
<b>GET</b> api/app/categoriesListing	public ResponseEntity<String> categoriesListing()	Összes termékkategória listázása
<b>GET</b> api/app/findProductsByCategory	public ResponseEntity<String> findProductsByCategory(int categoryId)	Összes termék listázása a megadott kategóriában
<b>GET</b> api/app/findProductsByFilterText	public ResponseEntity<String> findProductsByFilterText(String filterText)	Hagyományos keresés, összes termék visszaadása, melynek leírása/neve tartalmazza a megadott karaktersorozatot
<b>GET</b> 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.
<b>GET</b> 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.

<b>GET</b> api/app/getProductsFromSameVariation	public ResponseEntity<String> getProductsFromSameVariation(int variationId)	Az összes termék visszaadása, mely a megadott azonosítójú termékvariációkhoz tartozik
<b>GET</b> api/app/ getProductById	public ResponseEntity<String> getProductById(long prodId)	Adott azonosítóval rendelkező tárgy visszaadása

### 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
<b>POST</b> api/auth/signin	public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest)	Felhasználó beléptetése, JWT token visszaküldése sütikbe, felhasználó adatai (jogosultságok, felhasználónév, email) leküldése.
<b>POST</b> 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.
<b>POST</b> api/auth/signout	public ResponseEntity<?> logoutUser()	Kijelentkezés. Tárolt JWT token törlése sütikből, token „inaktiválása”.
<b>GET</b> 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).

Eseményvezérelt programozás: Budget Webshop  
n+1.csapat

<b>GET</b> api/auth/confirmMail	public ResponseEntity<?> confirmMail(long userId, String tokenKey)	E-mail megerősítése. Visszairányít a kliensoldalra. (elérése közvetlen szerveroldalra hivatkozás alapján, kliens oldal kihagyásával)
<b>POST</b> 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 felhasználóhoz kötődő, jogosultságot nem igénylő műveleteket tartalmazza, azonban bejelentkezés előfeltétele az elérésének. Bejelentkezés hiányában (token) 401-es HTTP error (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
<b>GET</b> api/user/testDebug	public ResponseEntity<String> getDebug()	Teszt metódus, Pl.: új funkciók ideiglenes tesztelése
<b>POST</b> api/user/addItemToBasket	public ResponseEntity<MessageResponse> addItemToBasket(long productId, int count)	Termék hozzáadása kosárba
<b>DELETE</b> api/user/removeItemFromBasket	public ResponseEntity<MessageResponse> removeItemFromBasket(long productId)	Termék kivétele kosárból
<b>POST</b> api/user/updateProductCountInBasket	public ResponseEntity<MessageResponse> removeItemFromBasket(long productId, int newCount)	Kosárban lévő termék mennyiségének módosítása
<b>POST</b> api/user/updateDeliveryAddress	public ResponseEntity<MessageResponse> updateDeliveryAddress(String phone, String country, String country_1, 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.

<b>GET</b> api/user/getDeliveryAddress	public ResponseEntity<String> getDeliveryAddress()	Felhasználó szállítási adatai
<b>POST</b> api/user/completeOrder	public ResponseEntity<?> completeOrder(String phone, String country, String country_1, 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.
<b>GET</b> api/user/getOrders	public ResponseEntity<?> getOrders()	A felhasználó korábbi rendeléseinek lekérése
<b>GET</b> api/user/getOrderProducts	public ResponseEntity<?> getOrderProducts(long orderId)	A felhasználó megadott rendeléséhez tartozó információk (termékek, azok mennyisége, ára stb.) lekérése
<b>POST</b> api/user/changePassword	public ResponseEntity<?> getOrderProducts(String oldPwd, String newPwd)	Felhasználó jelszavának megváltoztatása. Feltételként a regisztrációnál is érvényes min. 6 karakter él. Ha sikeres kijelentkezteti a felhasználót.
<b>GET</b> 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 errorrt 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
<b>POST</b> api/admin/insertNewProduct	public ResponseEntity<MessageResponse> insertNewProduct(@RequestBody ProductReqRep newProduct)	Új termék rögzítése
<b>POST</b> api/admin/updateProduct	public ResponseEntity<?> updateProduct(long productId, @RequestBody ProductReqRep updateProduct)	Termék módosítása
<b>POST</b> api/admin/createCategory	public ResponseEntity<MessageResponse> createCategory(String category, String smallDesc, byte priority)	Kategória létrehozása
<b>POST</b> api/admin/createVariation	public ResponseEntity<MessageResponse> createVariation(String variation)	Termékvariáció létrehozása
<b>GET</b> api/admin/getCategories	public ResponseEntity<String> getCategories()	Kategóriák lekérése
<b>GET</b> api/admin/getVariations	public ResponseEntity<String> getVariations()	Termékvariációk lekérése

<b>POST</b> api/admin/updateCategory	public ResponseEntity<MessageResponse> updateCategory(long id, String category, String smallDesc, byte priority)	Kategória módosítása
<b>POST</b> api/admin/updateVariation	public ResponseEntity<MessageResponse> updateVariation(long id, String variation)	Termékvariáció módosítása
<b>POST</b> api/admin/addProductToCategory	public ResponseEntity<MessageResponse> addProductToCategory(long prodId, long catId)	Termék felvétele kategóriába
<b>DELETE</b> api/admin/removeProductFromCategory	public ResponseEntity<MessageResponse> removeProductFromCategory(long prodId, long catId)	Termék kivétele kategóriából
<b>POST</b> api/admin/addProductToVariation	ResponseEntity<MessageResponse> addProductToVariation(long prodId, long varId, String desc)	Termék felvétele variációkba
<b>DELETE</b> api/admin/removeProductFromVariation	public ResponseEntity<MessageResponse> removeProductFromVariation(long prodId, long varId)	Termék törlése variációkból
<b>GET</b> api/admin/getAllOrder	public ResponseEntity<?> getOrders()	Összes rendelés lekérése

<b>GET</b> api/admin/findOrderById	public ResponseEntity<?> findOrderById(long ordeId)	Rendelés lekérése ID alapján
<b>GET</b> api/admin/getOrderProducts	public ResponseEntity<?> getOrderProducts(long orderId)	Rendeléshez tartozó adatok (termékek, azok mennyisége, ára stb.) lekérése
<b>POST</b> api/admin/updateOrderState	public ResponseEntity<?> updateOrderState(long orderId, byte newState)	Rendelés státuszának módosítása
<b>GET</b> api/admin/getUsers	public ResponseEntity<?> getUsers()	Az összes felhasználó visszaadása, a „rejtett” adatok nélkül (amit visszaad: ID, username, email, E-mail státusz, rendelések)
<b>POST</b> api/admin/updateAdminRights	public ResponseEntity<?> updateAdminRights(long userId, @RequestBody String rolesJson)	A megadott azonosítójú felhasználó jogosultságainak beállítása. A jogosultságokat JSON formátumú string gyűjteményben várja.



### *Service, és Config osztályok*

Az alkalmazás több „extra” modult tartalmaz, melyek rend szerint valamilyen Service osztályban helyezkednek el.

#### EmailService

Az alkalmazás fel van készítve, hogy E-mailt küldjön a felhasználóknak a Google (Gmail) SMTP szerverén keresztül. Az elektronikus levelezés implementálása az EmailServiceImpl Service osztályban történt, az EmailService (interface) alapján. A levelek tartalmának tárolásáért az EmailDetails modell (osztály) felel, mely tartalmazza a levél címzettjét, tárgyát, törzsét, valamint opcionálisan csatolmányának hivatkozását.

Az osztály, és metódusainak részletes leírása a javadocban olvasható.

#### pdfService

Az alkalmazás PDF dokumentumot készít minden rendeléshez tartozó „számláról” azok leadásakor, mely elkészítése a pdfService osztályban történik. A létrehozott számla tartalmazza a rendelés azonosítóját, a megrendelőt, az üzlet elérhetőségeit, valamint a rendelés adatait (szállítási, fizetési mód, végösszeg, leadás dátuma), illetve a megrendelt termékek listáját táblázatosan felsorolva (tárgy neve, db ár, rendelt mennyiség, összesített ár). A létrehozott dokumentum az alkalmazás főkönyvtára/receipts mappába kerül mentésre order\_”rendelés azonosító”\_szamla.pdf néven. A modul leírásáról részletesebben a javadocban olvashatunk.

#### tokenService

Az alkalmazás többféle tokennel is dolgozik, melyek száma tovább bővíthető a tokenTypes (enum) osztályban. A tokenek kezelésére hoztam létre a tokenServices osztályt, mely jelenleg a jelszó változtatáshoz, valamint az email cím megerősítéshez szükséges tokent állítja elő. Az előállítás módja: Random karaktersorozat 6 karakter hosszúságban + felhasználó azonosító(id) + random UUID a „-” karakterek nélkül.

A tokeneket előállítás után az adatbázisban eltárolja, melynek típusa, valamint az annak felhasználására jogosult felhasználó azonosítója is meghatározásra kerül.

#### ServiceConfiguration

A ServiceConfiguration konfigurációs osztály elsődleges célja, hogy azon osztályok módosított (konfigurált) példányát adja vissza, melyből az alapbeállítás nem megfelelő számunkra. Jelenleg ObjectMapper osztályt állítunk elő benne, melyre azért volt szükség, mivel alapból ez nem tudja a LocalDateTime összetett adattípust JSON formátumra alakítani, valamint visszaolvasni. Ennek orvoslása céljából itt példányosítunk belőle egy olyan típust, melyhez regisztráljuk a JavaTimeModule-t (a csomagoknál már említett jackson-datatype-jsr310 csomag tartalmazza), valamint ezt úgy konfiguráljuk, hogy letiltjuk a dátumtípusok „időbélyeg”-ként (timestamps) történő kezelését, majd az így konfigurált példányt adjuk vissza.

#### WebSecurityConfig

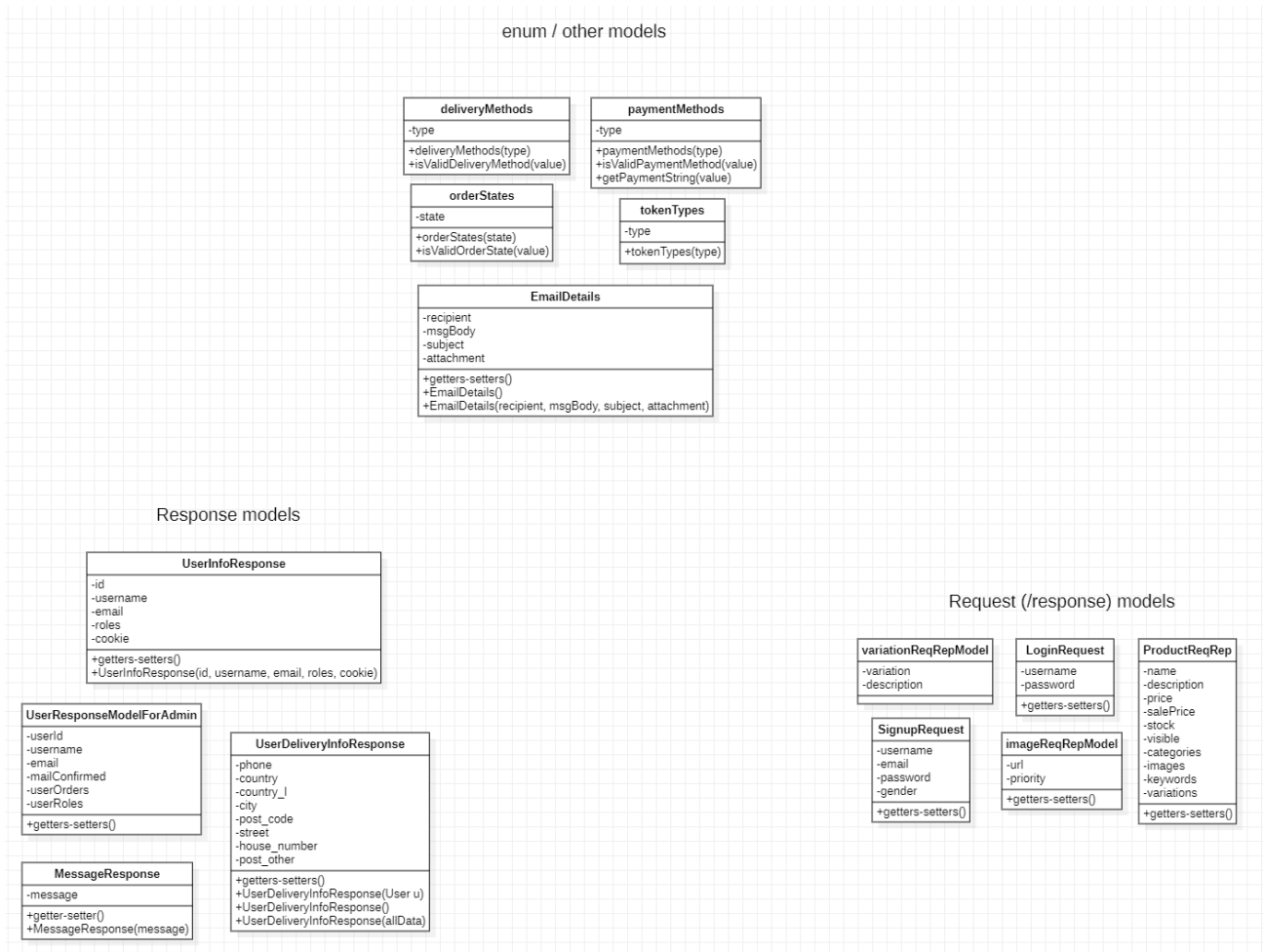
A WebSecurityConfig osztály a WebSecurityConfigurerAdapter leszármazottja. Célja, hogy az API hozzáférést korlátozzuk (jogosultság, bejelentkezés controllerekre bontva), valamint

## Eseményvezérelt programozás: Budget Webshop n+1.csapat

az autentikációhoz szükséges funkciókat elérjük (jelszó hashelő, auth token szűrő stb.).  
Részletes leírás a javadocban.

### Egyéb (nem adatbázis táblát leíró) modell osztályok, enumok

Az alkalmazás tartalmaz néhány olyan modell osztályt is, mely nem adatbázis táblát ír le. Ezek jellemzően adatellenőrzési célt szolgálnak (enumok, melyek tartalmazzák az adott típus lehetséges értékeit), vagy a kliens-szerver kommunikáció során nagyobb mennyiségű adatot tartalmazó kérések átláthatóbb, könnyebben feldolgozhatóságát segítik elő (jellemzően RequestBody-ból „kinyert” osztályleíró modell). Ezek osztálydiagrammja (részletes leírás javadocban):



### Tesztelés

Az alkalmazás fejlesztése során törekedtünk a modularitásra, melynek köszönhetően az alkalmazás külön feladatot ellátó funkcióit külön-külön (gyakran azokon belül is kisebb részekre bontva) tudtuk tesztelni. Minden egyes kérés létrehozása után annak tesztelését Postman alkalmazás segítségével végeztem, mellyel kéréseket tudtam küldeni a szervernek, és annak eredményét is láthattam. Ez alatt a tesztelés alatt igyekeztem minden funkció minden lehetséges hibáját tesztelni, ugyanis egy web api esetén a kérések eredményét JUnit4 tesztek

# Eseményvezérelt programozás: Budget Webshop

## n+1.csapat

segítségével nem célszerű kivitelezni. A háttérben futó modulok tesztelésére (adatbázis kapcsolat, adatbázissal kommunikáció, adatbázis műveletek) JUnit4 teszt készült.

Mivel az API-hoz frontendet is fejlesztettünk, ezáltal tágabb értelemben felfoghatjuk úgynevezett tesztvezérelt fejlesztési módszernek is (A meglévő szerverhez fejlesztettünk klienst), ezáltal a tesztelés következő fázisaként szolgált, hogy a szerver megfelelően viselkedik a kliens adott kérése esetén. Végül a tesztelés utolsó fázisaként végfelhasználói tesztet is végeztünk, mely során az alkalmazás minden funkcióját bejárva győződhettünk meg annak megfelelő működéséről. Egy kérés (jelen esetben a login) tesztelésének mintája:

http://localhost:8080/api/auth/signin

POST

http://localhost:8080/api/auth/signin

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

```

1  {
2    "username": "superadmin",
3    "password": "0123456789"
4  }

```

Body

Cookies (1)

Headers (15)

Test Results

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

```

1  {
2    "id": 16,
3    "username": "superadmin",
4    "email": "superadmin@ikwebshop.hu",
5    "roles": [
6      "ROLE_ADMIN2",
7      "ROLE_CUSTOMER"
8    ]
9  }

```

http://localhost:8080/api/auth/signin

POST

http://localhost:8080/api/auth/signin

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

```

1  {
2    "username": "superadmin",
3    "password": "123456789"
4  }

```

Body

Cookies

Headers (14)

Test Results

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

```

1  {
2    "path": "/api/auth/signin",
3    "error": "Unauthorized",
4    "message": "Bad credentials",
5    "status": 401
6  }

```

http://localhost:8080/api/auth/signin

POST

http://localhost:8080/api/auth/signin

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

```

1  {
2    "username": "superadmin",
3    "password": "0123456789"
4  }

```

Body

Cookies (1)

Headers (15)

Test Results

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

```

1  {
2    "path": "/api/auth/signin",
3    "error": "Unauthorized",
4    "message": "Bad credentials",
5    "status": 401
6  }

```

### Továbbfejlesztési lehetőségek

Napjainkban egyre jobban terjed el az online bevásárlás, melynek rengeteg előnye van, jellemzően nem csak hogy mindent meg tudunk venni a neten is, mint amit boltokban, hanem már az online boltokban sokkal nagyobb a kínálat. Az üzletek számára a költséghatékonyabb működés mondható a legfontosabb szempontnak (nem kell pénztárosokat, árufeltöltőket fizetni). Vannak általános célú funkciók, azonban nem meglepő, hogy a legtöbb megrendelő más-más egyedi igényeket diktál, ezáltal egy webshopot minden esetben modulárisra, könnyen tovább fejleszthetővé kell tervezni, megvalósítani.

A szerver oldal esetében talán a leghasznosabb hátralévő funkció a további jogosultságokhoz kötődő funkciók megvalósítása (Pl.: akciók esetén a VIP felhasználók előbb látják, mint a hagyományos felhasználók, beszállítók számára a raktárkészlet megtekintése, módosítása, online rendszeren keresztül történő árubeszerezés, futárok rendszerének összekapcsolása az oldallal, ezáltal tudnák módosítani a rendelés státuszát miután kiszállították).

Mivel egy API-ról beszélünk, ezáltal semmiképpen nem szabad megfélekedni a platformfüggetlenségről, jelenleg „csak” webes alkalmazás áll rendelkezésre az alkalmazáshoz, mindazonáltal ez mellé érdemes lehet platformspecifikus alkalmazást fejleszteni (Android APP, IOS App, asztali alkalmazás), melyek közül a mobilos alkalmazások megvalósítása szinte pillanatok alatt, minimális módosítással elvégezhető a react JS-nek köszönhetően (React Native segítségével a meglévő kódunk minimális módosításával exportálhatjuk ki a mobil alkalmazásunkat android, és ios eszközre is).

A szerveroldal ezen túl több funkcióval is rendelkezik, melyet az idő szűkében sajnos nem jutott időnk a kliensbe integrálni. Ilyen például az adott termékcsaládba tartozó termékek felajánlása termék részletezésekor, mint „hasonló termék”.

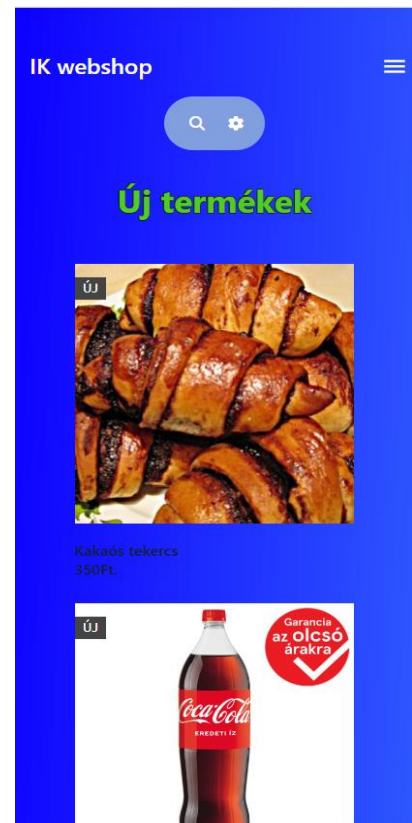
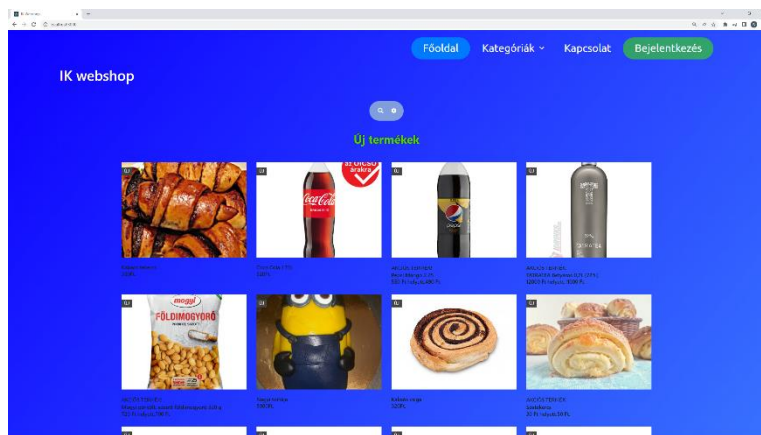
Az alkalmazásban a fejlesztés során főleg az elsődleges kimenetet használtuk naplózás, hibakeresés, és debugolás céljából, azonban ez egy élesen futó alkalmazás esetén egy esetleges szerverleállás után kinyerhetetlenné válik. Létrehoztam egy log táblát, mely működőképes az alkalmazásban, viszont a fejlesztés során egyszerűbb volt a konzolt nézni hibakereséskor, mint adatbázist böngészni, ezáltal adatbázisba történő logolások nincsenek.

Eseményvezérelt programozás: Budget Webshop  
n+1.csapat

## Felhasználói kézikönyv

### Oldal felkeresése

A webshopot bármilyen eszközről felkereshetjük (internetkapcsolat függvényében), mely rendelkezik internet böngészővel. Az alkalmazást a <http://localhost:3000> (szerver domain) címen érhetjük el. Az alkalmazás fejlesztése során elsődleges szempont volt a platformfüggetlenség, ezáltal az alkalmazás minden eszközön használhatóan jelenik meg böngészőtől, és eszköztől (képernyő, operációs rendszer) függetlenül.

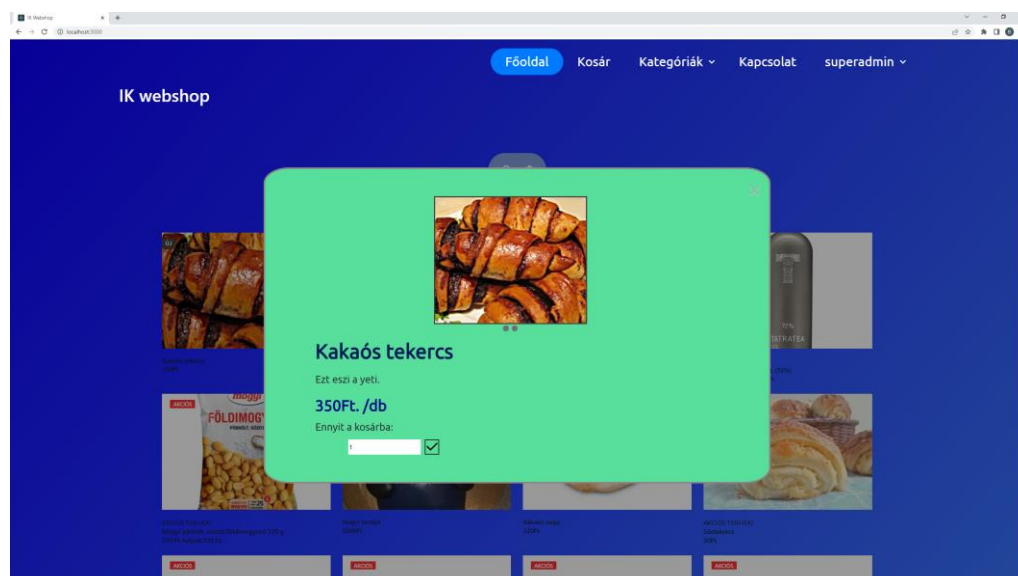


### Menüsor

A menüsor tartalma bejelentkezési státusztól, jogosultságtól, és az aktuális oldaltól függően változik.

### Főoldal

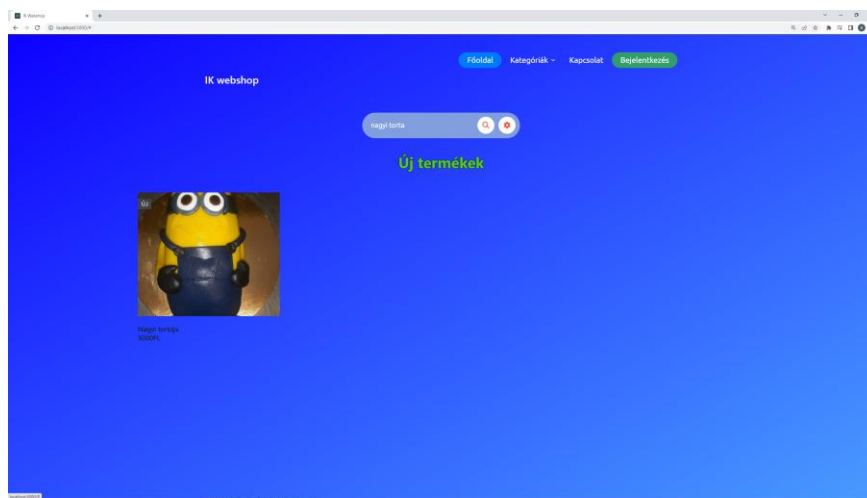
A főoldalon találjuk a webshop termékeit. A termék részleteinek megtekintéséhez kattintsunk az adott termék képére. Amennyiben kosárba szeretnénk rakni terméket, ahhoz bejelentkezésre van szükség.



## Kereső

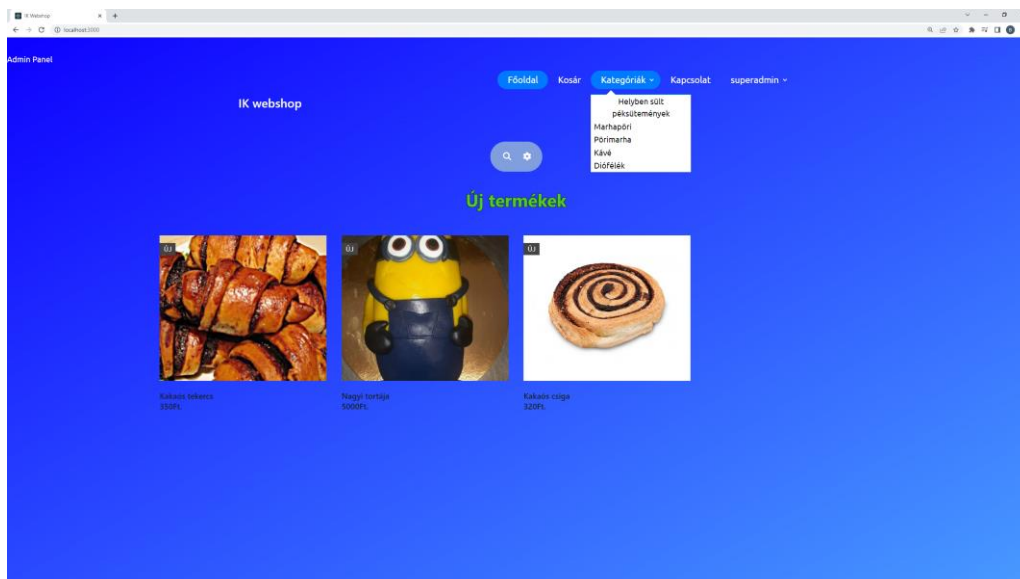
Az alkalmazás egy speciális, háromfunkciós keresővel van ellátva.

1. Hagyományos keresés (szöveges): Ebben az esetben azokat a termékeket láthatjuk, melyek neve, vagy leírása tartalmazza a keresőmezőben megadott karaktersorozatot.
2. Pontatlan kulcsszavas keresés: Ebben az esetben azokat a termékeket látjuk viszont, melyek a megadott kulcsszavak bármelyikét tartalmazza. A kulcsszavakat szóközzel elválasztva kell megadni.
3. Pontos kulcsszavas keresés: Ebben az esetben azokat a termékeket látjuk viszont, melyek a megadott kulcsszavak mindegyikét tartalmazza. A kulcsszavak megadása szóközzel elválasztva lehetséges.



## Szűrés kategóriákra

Lehetőségünk van arra is, hogy a termékek kategória alapján jelenjenek meg. Ehhez a felső menüsorban a kategóriák menüpont alól válasszuk ki, hogy mely kategóriába tartozó termékeket szeretnénk látni.

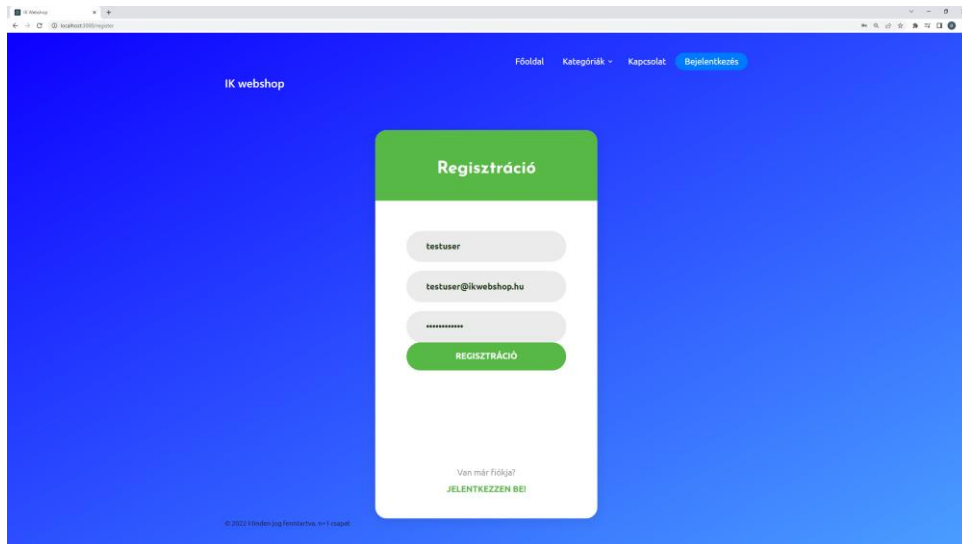


## Eseményvezérelt programozás: Budget Webshop n+1.csapat

### Autentikáció, profil beállítások

#### Regisztráció

Regisztrációra a regisztrációs form kitöltésével van lehetőségünk. A regisztráció során több kitétel is van.: Felhasználónév minimum 3, maximum 40 karakter, és még nem létezik a rendszerben; Email cím: Megfelelő (valós) E-mail, és még nem létezik a rendszerben; Jelszó: Minimum 6, maximum 40 karakter. Sikeres regisztráció után a felhasználó a megadott E-mail címre kapni fog egy levelet, mely tartalmazza az E-mail cím megerősítéséhez tartozó teendőket.



#### E-Mail cím megerősítése

Annak érdekében, hogy a felhasználó rendelést tudjon leadni, meg kell erősítenie a profilhoz tartozó E-mail címet. Ezt a regisztrációkor kapott mailben lévő hivatkozásra kattintva teheti meg, miután a rendszer értesíti a felhasználót a megerősítés eredményéről.

### IK Webshop - Sikeres regisztráció

Beérkező levelek x



evpwebshop@gmail.com

címzett: én ▼

Kedves vhalazs2!

Köszönjük, hogy regisztrált az IK Webshopba. Az E-mail címe megerősítéséhez kérjük kattintson [ide](#)

Üdvözlettel: IK Webshop

← Válasz

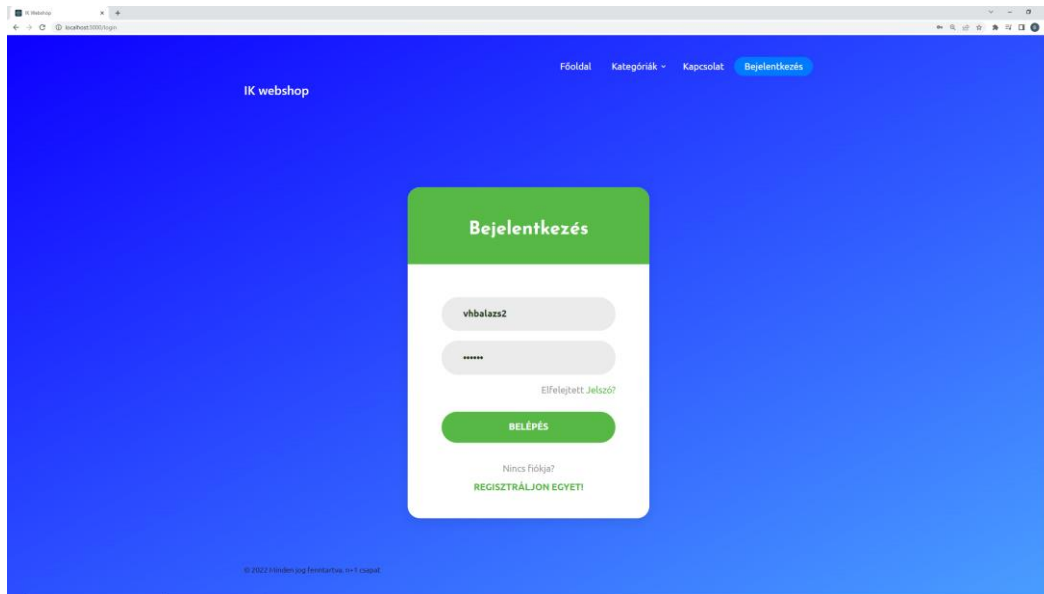
→ Továbbítás



## Eseményvezérelt programozás: Budget Webshop n+1.csapat

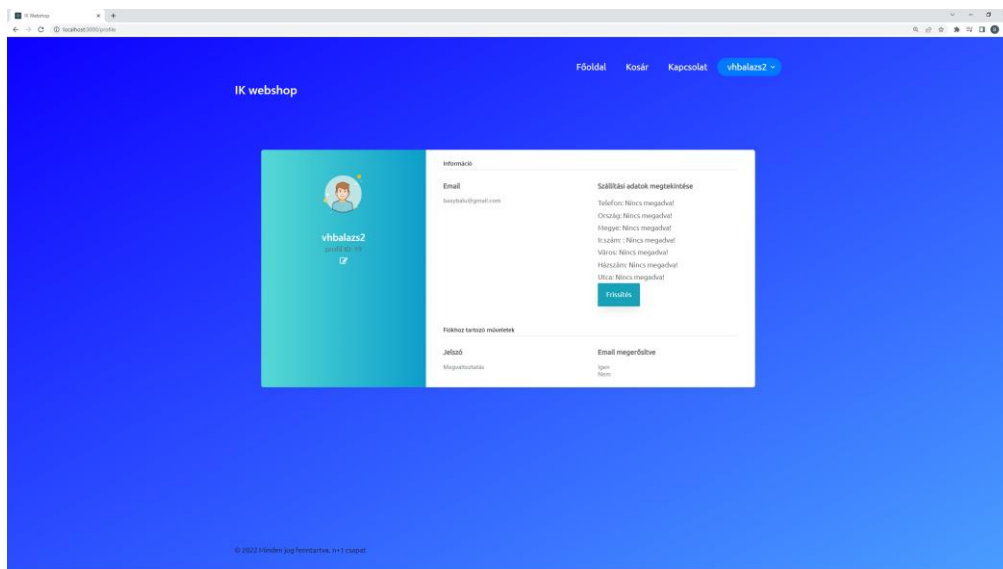
### Bejelentkezés

Az alkalmazásba történő bejelentkezéshez regisztrált fiókra van szükségünk. Annak érdekében, hogy termékeket rakhassunk a kosarunkba be kell jelentkezünk, de nem szükséges, hogy az E-mail címünket megerősítsük (erre a rendelés leadásához van szükség).



### Szállítási adatok megadása

Amennyiben szeretnénk gyorsítani a rendelési folyamatot, a profilunkban beállíthatjuk a szállítási címünket, és ez rendeléskor automatikusan ki fog tölteni már. Erre a profil menüpont alatt van lehetőségünk a szállítási adatok részénél.

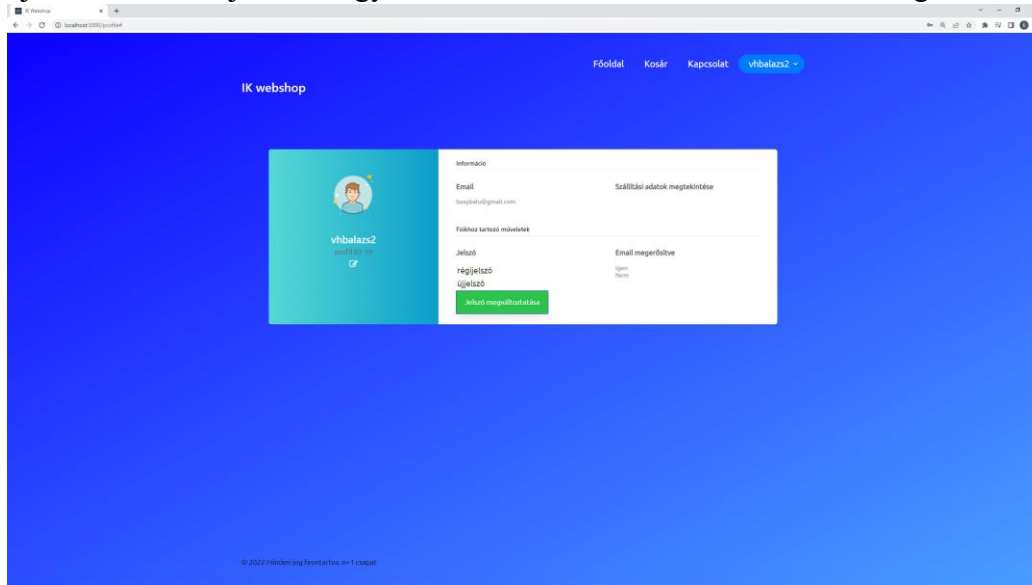


### Jelszó megváltoztatása

Amennyiben új jelszót szeretnénk beállítani a fiókunkhoz, abban az esetben a profil menüpontra kell mennünk, majd a jelszó rész alatt a megváltoztatás felíratra kattintva megjelenik két szövegdozso. A felsőbe kell beírni a régi jelszavunkat, majd alá az új jelszót. Miután kitöltöttük a mezőket kattintsunk a jelszó megváltoztatása gombra. Az



újonnan beállítót jelszóra ugyan azok a feltételek vonatkoznak, mint regisztrációkor.



### Elfelejtett jelszó

Amennyiben a felhasználó elfelejtette a jelszavát lehetősége van újat megadni. Ehhez navigáljunk a bejelentkezés / elfelejtett jelszó pontra. Adjuk meg az Email címünket amiről regisztráltunk, majd kattintsunk a gombra. Ekkor a rendszer küldeni fog nekünk egy levelet a megadott címre, ebben kattintsunk rá a hivatkozásra, mely elnavigál az oldalra, ahol megadhatjuk az új jelszavunkat. Itt írjuk be a kívánt jelszót, majd kattintsunk a gombra.



**Kedves vhbalazs!**

Az új jelszavának megadásához kérjük kattintson [ide](#)

Üdvözléssel: **IK Webshop**

### Rendelési folyamat

#### Termék kosárba rakása

Amennyiben rendelni szeretnénk, előbb a megrendelni kívánt termékeket a kosarunkba kell raknunk. Egy rendelés alkalmával bármennyi termékből rendelhetünk, azonban csak azokat a termékeket rakhatjuk a kosarunkba, melyből a megadott mennyiség készleten van.

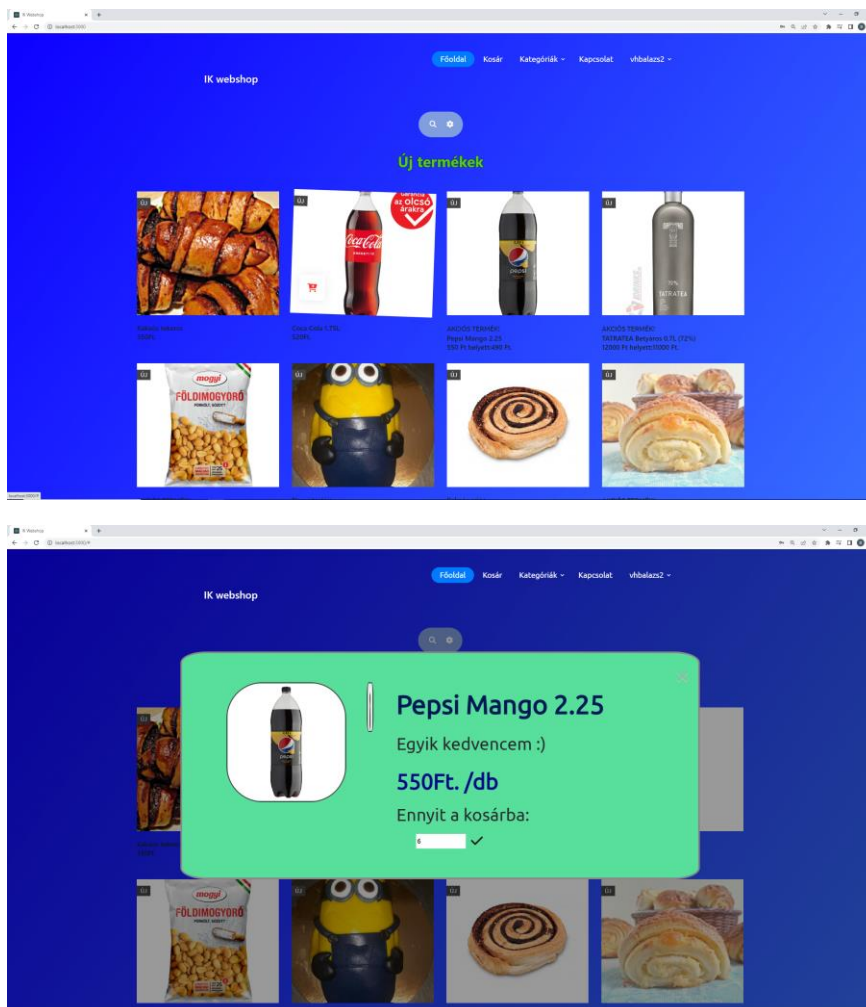
Kosárba rakni terméket kétféleképpen tudunk. Amennyiben csak 1 darabot szeretnénk, abban az esetben a termék képe fölé fókuszálva megjelenik egy kosár ikon, erre kell kattintanunk. Ha többet szeretnénk rendelni a termékből, abban az esetben a termék részleteit meg kell

## Eseményvezérelt programozás: Budget Webshop n+1.csapat

nyitnunk a képre kattintva, és itt kell megadnunk a kívánt mennyiséget, majd a doboz mellett található pipára kell kattintani.

### FIGYELEM

- Egy termék csak egyszer szerepelhet a kosarunkban, azonban a mennyiségét módosíthatjuk (1X2 rendben, 2X1 hiba).
- A termék kosárba rakása nem jelenti azt, hogy az adott terméket lefoglalta a felhasználó, tehát előfordulhat, hogy a kosárba rakás, és a rendelés leadása közben az adott termékből kifogy a készlet.
- A termék végső ára a rendelés leadásának pillanatában kerül meghatározásra, nem pedig a kosárba rakásakor.

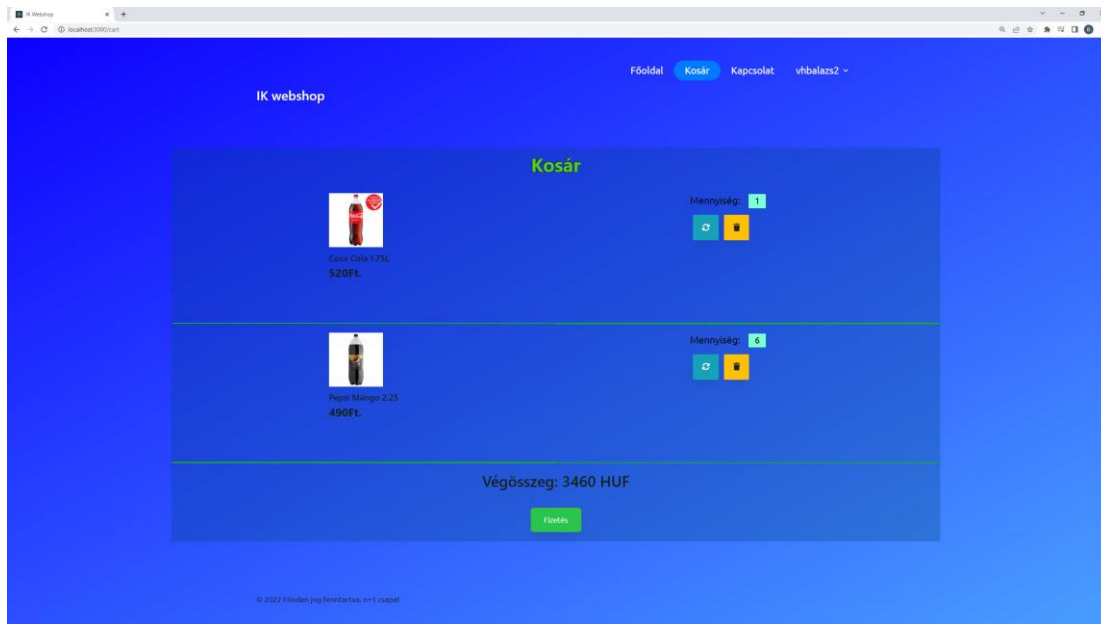


Kosár tartalmának megtekintése, kosárban lévő termék törlése, mennyiség módosítása  
A kosarunkban lévő termékeket a kosár menüpont alatt tekinthetjük meg. Itt láthatjuk a rendelésünk végösszegét aktuális állapota szerint.

Szintén itt törölhetünk a termékeket a kosarunkból, amennyiben mégsem kívánjuk megrendelni. Ehhez a termék mellett található sárga háttérű szemetes gombra kell kattintanunk.

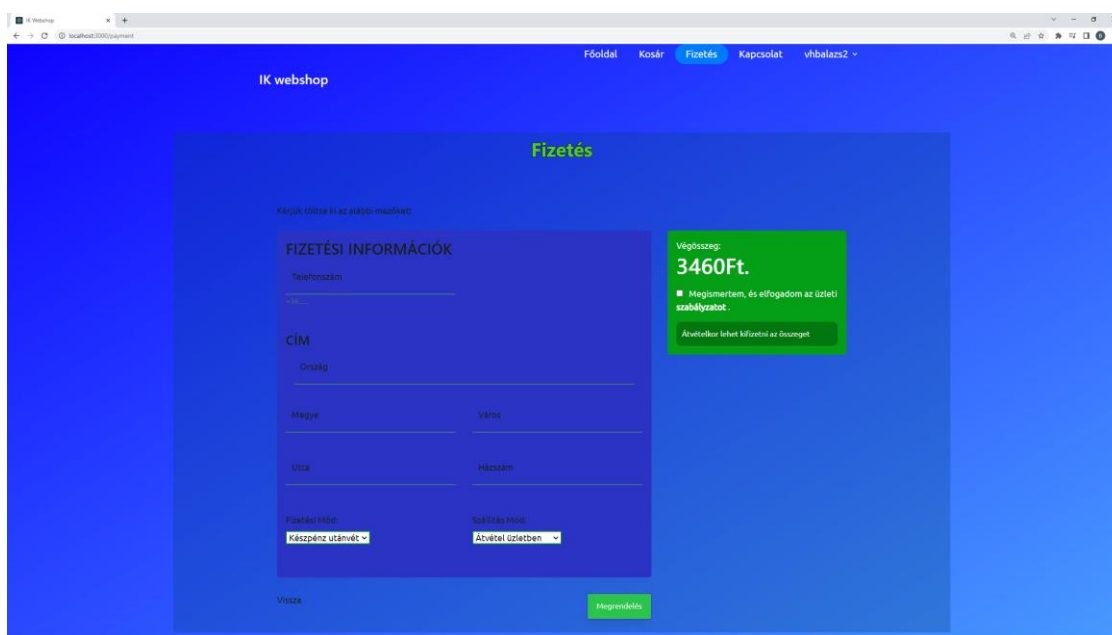
## Eseményvezérelt programozás: Budget Webshop n+1.csapat

Amennyiben módosítani szeretnénk a kosárban lévő termékek számát valamely termék esetén, abban az esetben a mellette található dobozba írjuk be a kívánt mennyiséget, majd kattintsunk a kék hátterű gömbölyű nyilakra.



### Rendelés leadása

Rendelésünk leadásához a kosarunkba kell navigálni. A kosár alján találunk egy Fizetés gombot, melyre kattintva elnavigál minket az oldal a rendelésösszesítő oldalra. Itt ki kell töltenünk minden mezőt a szállítással kapcsolatban. A fizetési módok közül válasszuk ki a kívánt fizetési módot (Átutalás, KP utánvét, Bankkártyás utánvét), majd válasszuk ki az átvételi módot (átvétel üzletben, átvétel postán, GLS futár, DPD futár, MPL futár, Foxpost, pick-pack pont). A végső rendeléshez minden esetben el kell fogadnunk az üzleti szabályzatot.



## Adminisztrátori teendők

Az adminisztrátori funkciókat az adminpanel menüpont alatt érhetjük el.

### Új termék rögzítése

Amennyiben új terméket szeretnénk felvenni, navigáljunk az adminpanel új termék létrehozása pontra. Töltsük ki a mezőket az igényeink szerint, majd kattintsunk az új termék felvétele gombra. Termékekhez képeket úgy rendelhetünk, hogy a szövegmezőbe „Kép URL `szóköz` prioritás” formában adjuk meg a képeket soronként.

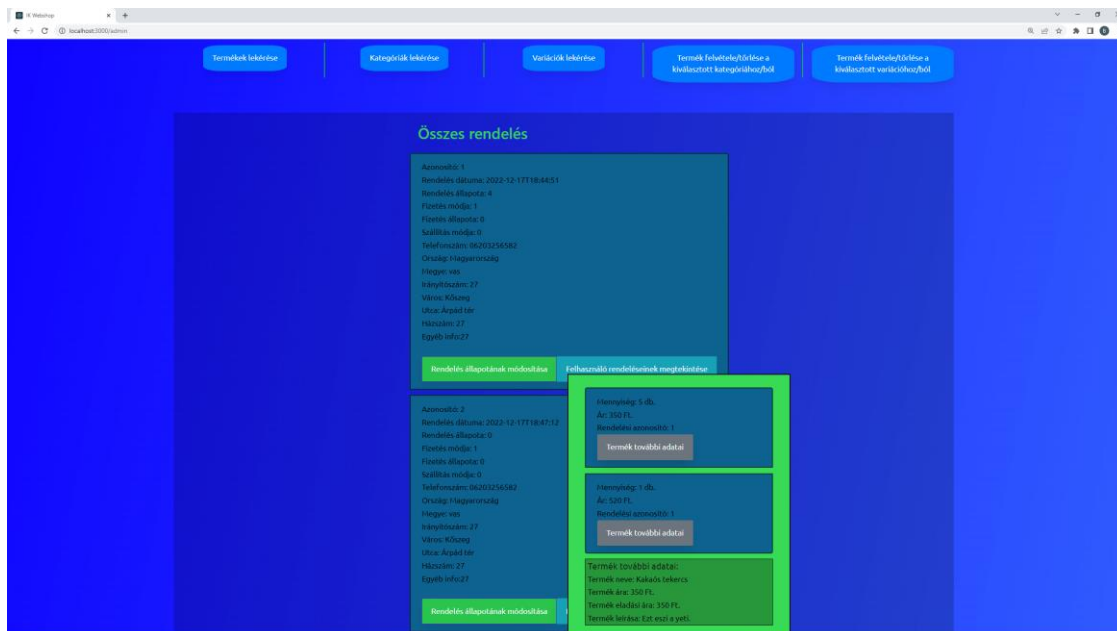
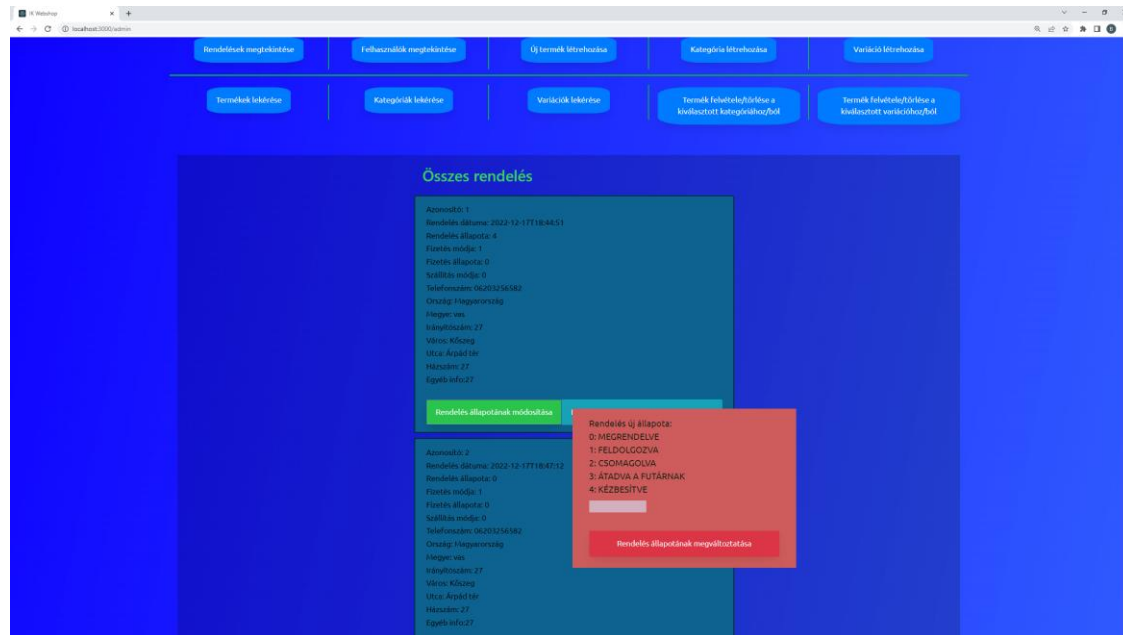
The screenshot shows the 'Admin panel' interface with a grid of buttons at the top: 'Rendelések megtekintése', 'Felhasználók megtekintése', 'Új termék létrehozása', 'Kategória létrehozása', 'Variáció létrehozása', 'Termékek lekérzése', 'Kategóriák lekérzése', 'Variációk lekérzése', 'Termék felvétele/törzése a kiválasztott kategóriához/fől', and 'Termék felvétele/törzése a kiválasztott variációhoz/fől'. The 'Új termék adatai' form is displayed, containing the following fields:

- Termék neve:
- Termék leírása:
- Termék kategória (bármely több is): ☐ Élelmiszer ☐ Készletanyag ☐ Kiegészítő ☐ Alkatrész ☐ Egyéb
- Termék variációja (bármely több is): ☐ Nincs ☐ Kis tálca ☐ Kiseb. porcelán kupa
- Termék ára:
- Termék eladási ára:
- Termék Készlet:
- Képek formátum: 'f'enter" sortolás URL, "szóköz" prioritás  
példa: <https://api.hug.hu/img/77cef0b8-ac48-4ee6-aef5-32203afa118c/5925235d-ada9-4424-82fd-1a95a688e6c0.jpg> 1  
<https://media.makememe.org/created/s-egy-csipetnyi.jpg> 2
- Termék láthatósága:
- Új termék felvétele

### Rendelések megtekintése, státuszmódosítás

Az alkalmazás egyik legalapvetőbb funkciója, hogy adminisztrátorként nyomon követhessük, hogy ki mit rendelt az áruházunkból, és ezt eljuttassuk a vásárlónak. Adminisztrátorok számára ezért elérhető egy rendelések megtekintése funkció az adminpanel alatt, ahol megnézhetjük a leadott rendelések listáját, a rendelésekhez tartozó termékeket, valamint beállíthatjuk, hogy a rendelés éppen milyen fázisban tart. 5 rendelési státusz különböztetünk meg: Megrendelve, Feldolgozva, Csomagolva, Átadva a futárnak, Kézbessítve. Ezeket a rendelés állapotának módosítása gombra kattintva adhatjuk meg, a megfelelő számot beírva. A rendeléshez tartozó termékeket (részletek) a felhasználó rendelések gombra kattintva tudjuk listázni.

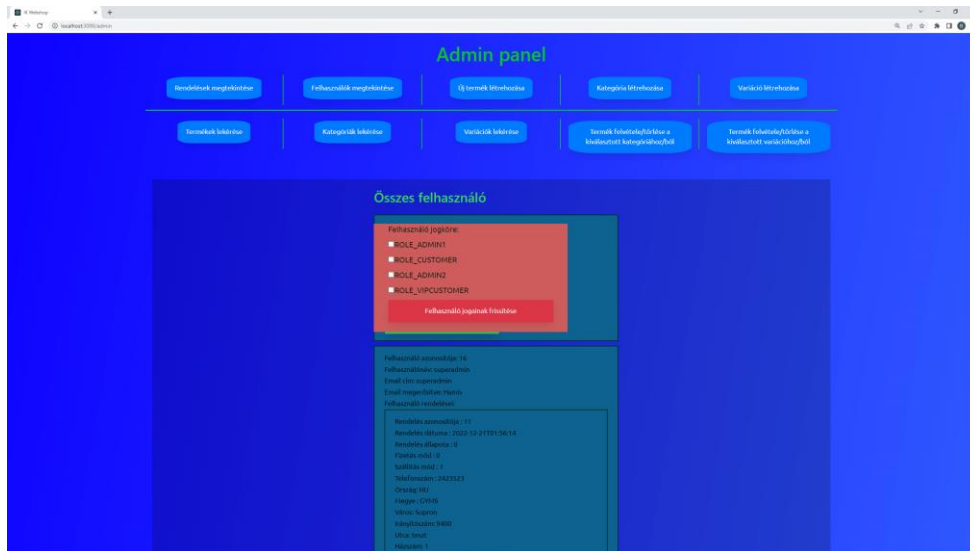
## Eseményvezérelt programozás: Budget Webshop n+1.csapat



Felhasználók (és rendelések) megtekintése, jogosultságok módosítása

Az alkalmazásban több jogosultság is elérhető, de jelenleg ezek közül létjogosultsága csak a „user”, az „admin1”, és „admin2” privilégiumoknak van. A jogosultságok felhasználói fiókhoz tartoznak, azokat módosítani „admin2” jogosultsággal lehet, azonban a felhasználókat listázni, valamint a rendeléseiket megtekinteni admin1 jogosultsággal is lehet.

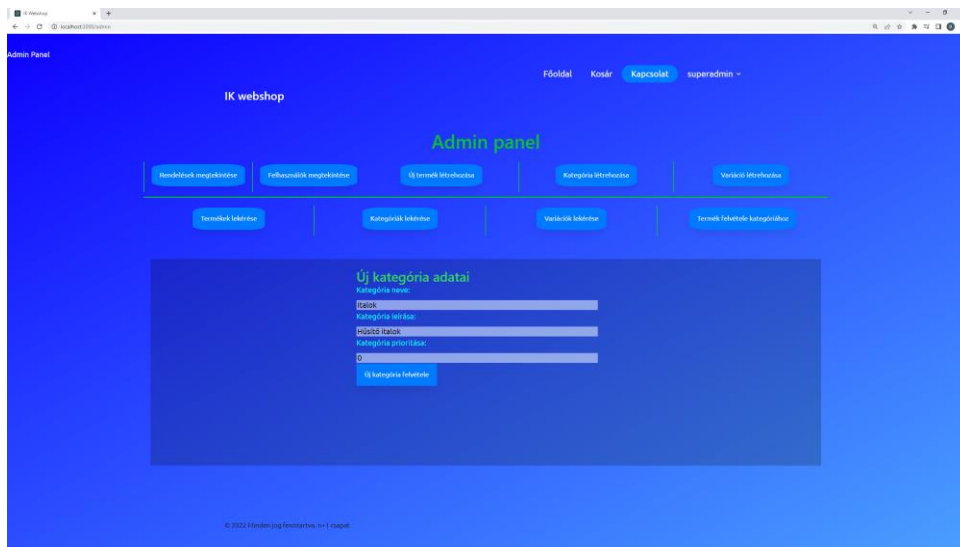
## Eseményvezérelt programozás: Budget Webshop n+1.csapat



### Új kategória létrehozása

A termékeket lehetőségünk van kategóriákba sorolni, mellyel megkönnyítjük, hogy a vásárlóink megtalálják a keresett terméket, ugyanis termékek megjelenítésekor meg lehet határozni, hogy csak adott kategóriába tartozó termékek jelenjenek meg (pl.: italok, péksütek).

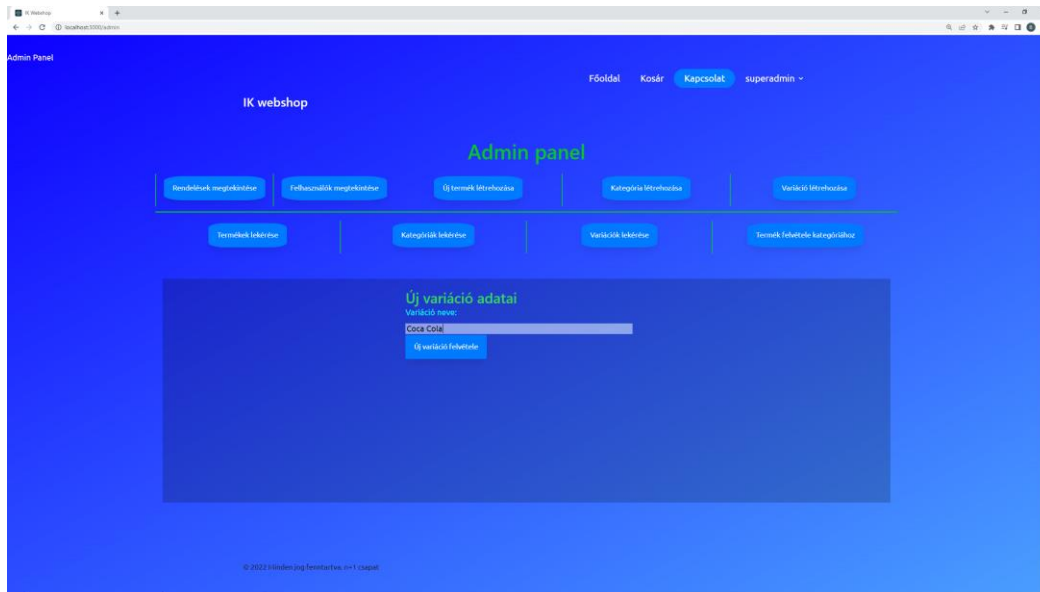
Új kategória létrehozásához navigáljunk az adminpanel kategória létrehozása pontjába, majd töltsük ki az adatokat, és kattintsunk a kategória létrehozására. A prioritás funkciója, hogy ez alapján állítja sorrendbe a szűrő a kategóriákat.



### Új variáció létrehozása

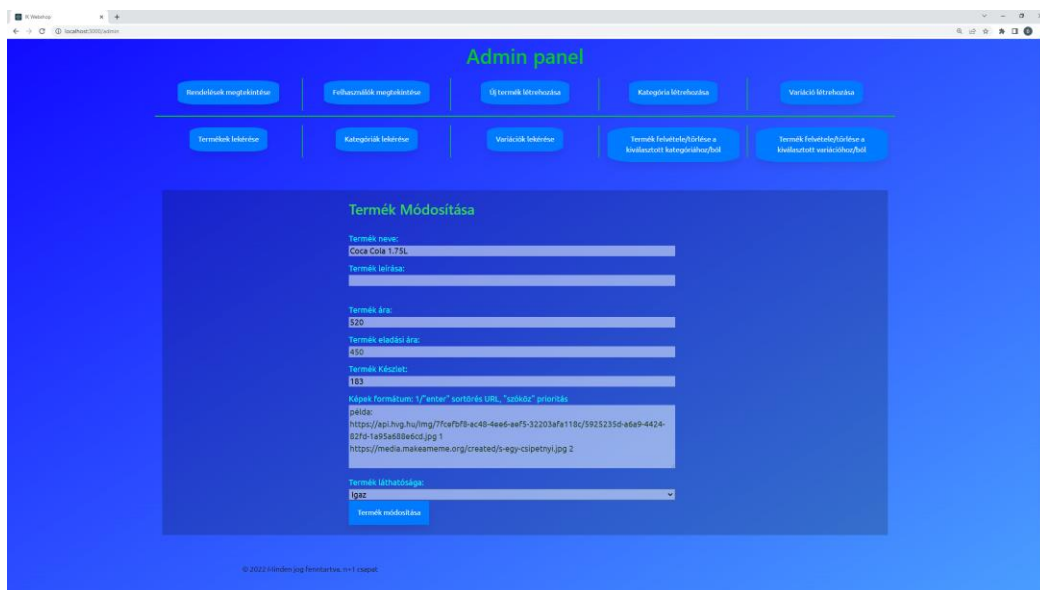
Az alkalmazásban lehetőségünk van termékvariáció létrehozására is. Ennek célja, hogy a azonos „termékcsaládba” tartozó termékeket csoportba rendezzük (Pl.: Coca Cola 0.5L, Coca Cola 1.75L). Termékcsalád létrehozásához az adminpanel variáció létrehozása alatt van lehetőség. Töltsük ki a szövegdobozt, majd kattintsunk a létrehozás gombra.

## Eseményvezérelt programozás: Budget Webshop n+1.csapat



### Termékek lekérése, módosítása

Amennyiben meg szeretnénk nézni, hogy milyen termékek szerepelnek a webshopunkban, vagy módosítani szeretnénk azokat, navigáljunk az adminpanel termékek lekérése pontjába. Módosításhoz a kívánt terméknel kattintsunk a módosítás gombra. A megjelenő ablakon módosítsuk a terméket a kívánt módon, majd kattintsunk a termék módosítása gombra.

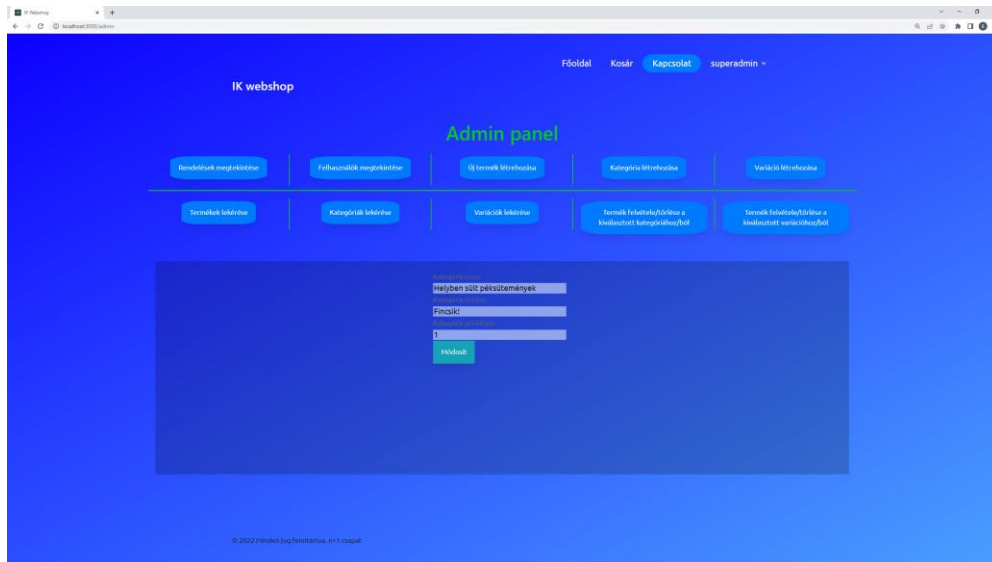


### Kategóriák lekérése / módosítása

A termékkategóriák megtekintéséhez navigáljunk az adminpanel / kategóriák lekérése menüpontra. Amennyiben valamelyik kategóriát módosítani szeretnénk kattintsunk rá a hozzá tartozó módosítás gombra, módosítsuk kívánság szerint az adatokat, majd kattintsunk rá a mentés gombra.

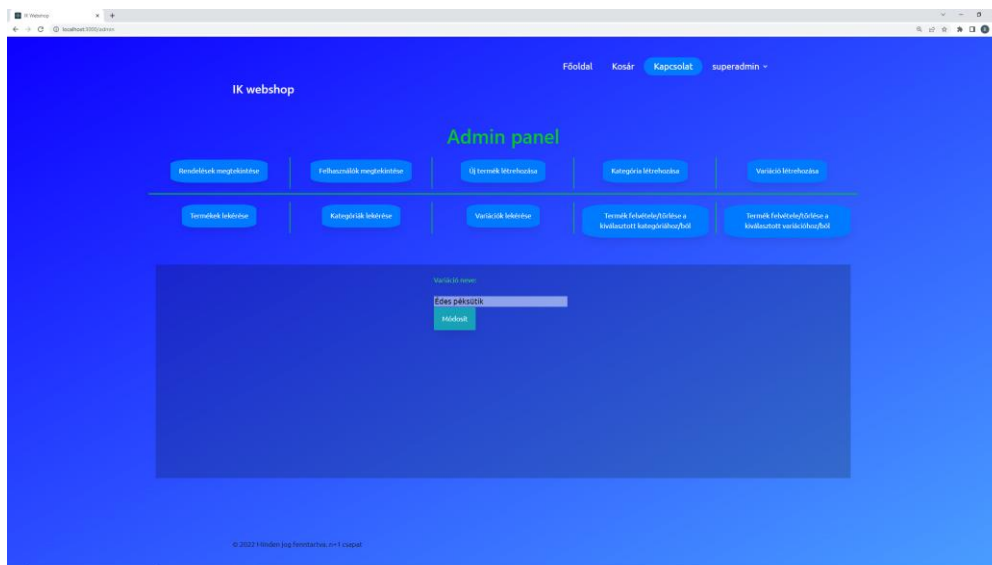


## Eseményvezérelt programozás: Budget Webshop n+1.csapat



### Variációk lekérése / módosítása

A termékcsaládok módosításához navigáljunk az adminpanel variációk lekérése pontjába, majd kattintsunk a módosítani kívánt variációhoz tartozó módosítás gombra. Módosítsuk igény szerint a variációt, majd kattintsunk a módosít gombra.

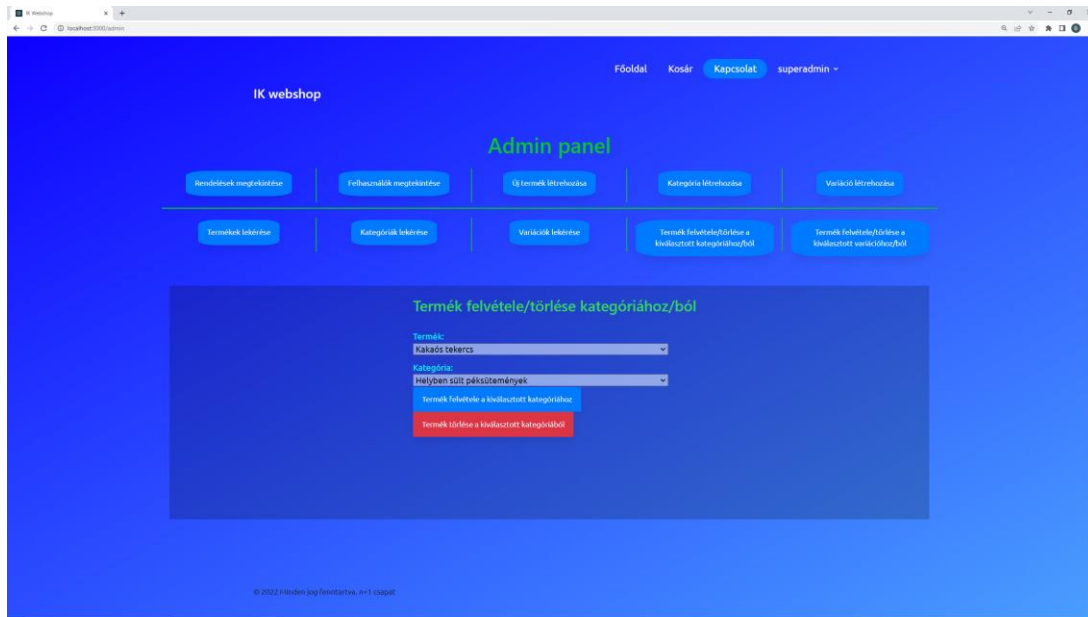


### Termék felvétele / törlése kategóriából

Amennyiben terméket szeretnénk kategóriához adni menjünk az adminpanel termék felvétele kategóriához pontjába. Válasszuk ki, hogy melyik terméket mely kategóriába szeretnénk berakni, majd kattintsunk a termék felvétele a kiválasztott kategóriához gombra. Amennyiben törölni szeretnénk a kategóriából, természetesen arra a gombra kattintsunk.



## Eseményvezérelt programozás: Budget Webshop n+1.csapat



### Termék felvétele / törlése variációból

Amennyiben termékcsaládba kívánunk felvenni terméket, navigáljunk az adminpanel / termék felvétele, vagy törlése variációból pontra. Válasszuk ki a kívánt terméket, és variációt (hozzáadásnál adjuk meg a rövid leírást igény szerint), majd kattintsunk a megfelelő gombra.

