



ZÁRÓDOLGOZAT

Készítették:

Bodolai Dániel

Galvács Máté

Miskolc

2022.

Miskolci SZC Kandó Kálmán Informatikai Technikum

Az 5-0613-12-03 számú Szoftverfejlesztő és -tesztelő technikus szakképesítés
záródolgozat

G&B Ruházati webáruház

Készítették:

Bodolai Dániel

Galvács Máté

Miskolc

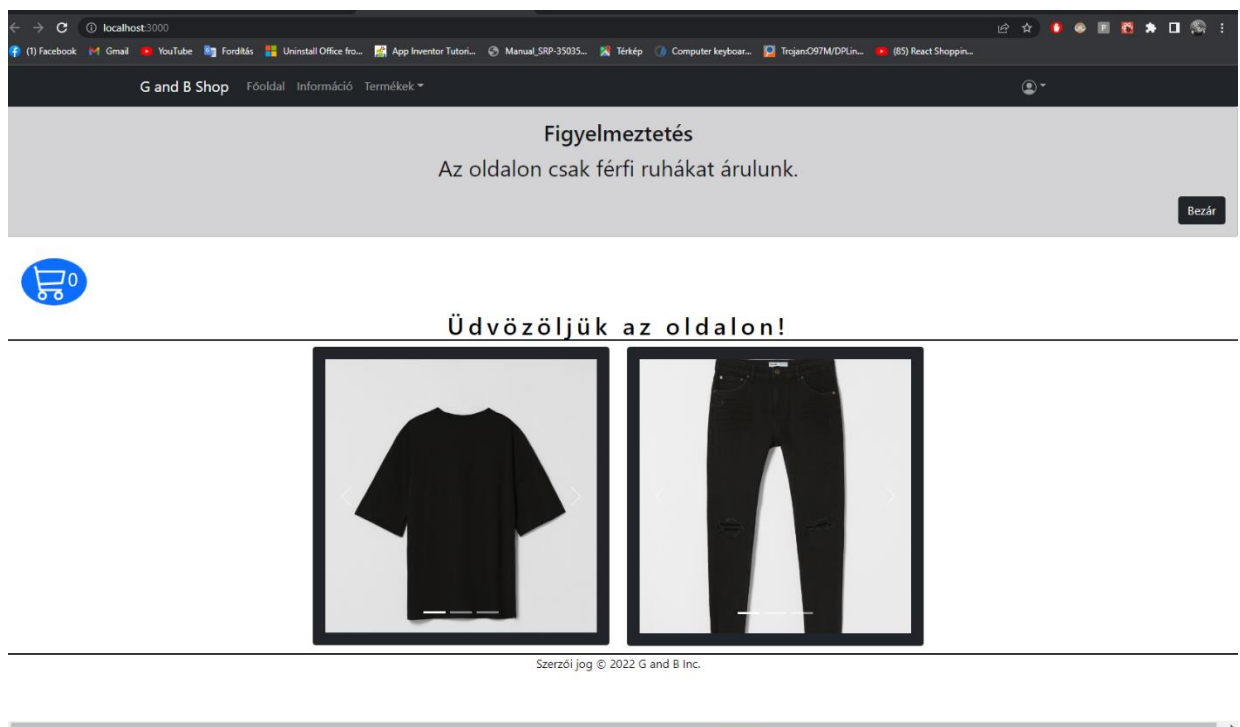
2022.

Tartalom

- 1.Bevezetés..... 4
- 2.Fejlesztői eszközök..... 5
 - 2.1.Visual Studio Code 5
 - 2.2. Node.js 5
 - 2.3. MongoDB 5
 - 2.4. Postman 6
 - 2.5. Jest..... 6
 - 2.6.Express..... 6
- 3. Fejlesztői környezet 6
 - 3.1. JavaScript 6
 - 3.2. React 7
- 3.Frontend 8
 - 3.1.Navigációs bár kivitelezése 8
- 4.Menüpontok 9
 - 4.1.Regisztráció 9
 - 4.2. Bejelentkezés 9
 - 4.3.Termék kiválasztás menüpont 10
 - 4.4.Frontend változásai adminisztrátori joggal 11
 - 4.5.Cart komponens- Kosár..... 12
- 5.Backend 12
 - 5.1.Tervezés..... 12
 - 5.2.Adatbázis 13
 - 5.3.Szerver..... 15
 - 5.4.Útvonalak..... 16
 - 5.4.1.Termékek 16
 - 5.4.2.Felhasználók..... 16
 - 5.4.3.Megrendelesek..... 17
- 6.Tesztelés 18
 - 6.1.Postman tesztek 18
 - 6.2.Jest tesztek..... 19
 - 6.2.1.Jest teszt- Backend 19
 - 6.2.2.Jest teszt- Frontend 20
- 7.Fejlesztési tervek..... 20
- 8.Források..... 20

1.Bevezetés

Szakdolgozat témájának egy online ruha rendelős web program tervezését és kivitelezését választottuk. Napjainkban sokan választják már inkább az online vásárlást és ezt a dolgot szerettük volna minél felhasználóbarátabbá, könnyen kezelhetővé és átláthatóbbá tenni, hogy azok számára is egyszerű legyen a kezelése, aki még nem csinált ilyet és nem jártas benne. A programunk tartalmaz egy regisztrációt, ami után az adatokat eltároljuk, és későbbiekben segít a felhasználónak, hogy rendelés leadásánál már ne keljen még egyszer megadni a személyes adatait. Az ötlet megvolt ezt követte a kivitelezés, amire rengeteg megvalósítási lehetőség volt, bár mi design és funkcionalitás tekintetében úgy gondoltuk, hogy a kevesebb néha több. Szokványosan elosztottuk a munkákat front-end és back-end részekre, illetve a többi részt pedig elosztottuk képességek és idő szerint, eleinte az volt számunkra tiszta, hogy egy React webalkalmazást szeretnénk csinálni, így neki is láttunk, megbeszéltük milyen alapvető funkcióink legyenek, majd egymás segítségével nekiültünk. A projekt során a megbeszélésekre, Discord-ot, tárolásra és verziókezelésre GitHub-ot, a feladatok számotartására pedig Trello-t használtunk (lásd [trello.png](#)). A projekt kapcsán megismerkedtünk új technológiákkal, amelyekre a későbbiek során részletesen kitérünk, valamint még a csapatban dolgozás előnyeivel. Véleményünk szerint, sokat fejlődünk a közös munka alatt, ami jó hangulatban telt és terveink között szerepel, ennek a feladatnak a továbbfejlesztése a közeljövőben.



1. ábra Az projektünk főoldala

2. Fejlesztői eszközök

2.1. Visual Studio Code

A program kódot és a tesztelést főként a Visual Studio Code-ban készítettük, ami számunka a különböző operációsrendszerek miatt volt számunka jó elsősorban, valamint a beépített Git támogatás miatt. A Visual Studio Code (vagy röviden VSCode) egy ingyenes nyílt forráskódú kód szerkesztő, melyet a Microsoft fejleszt Windows, Linux és OS X operációs rendszerekhez. Támogatja a hibakeresőket, továbbá képes az intelligens kódkezelésre (intelligent code completion) az IntelliSense segítségével. Ezen felül testre szabható, így a felhasználók megváltoztathatják a kinézetet (témát), megváltoztathatják a szerkesztő gyorsbillentyű-kiosztását, az alapértelmezett beállításokat és még sok egyebet. A Visual Studio Code az Electron nevű keretrendszeren alapszik, amellyel asztali környezetben futtatható Node.js alkalmazások fejleszthetők.

2.2. Node.js

A Node.js-re a front- és backend kapcsán is szükségünk volt és mind szerver, mind kliens oldalról használtunk harmadik féltől származó könyvtárat, a React-nél és a szervernél is kulcsfontosságú volt, a szerveret ezért is szokás NodeJS szerverként emlegetni. A Node.js egy szoftverrendszer, melyet skálázható internetes alkalmazások, vagy pedig webszerverek készítésére hoztak létre. A programok JavaScriptben írhatók, eseményalapú, aszinkron I/O-val a túlterhelés minimalizálására és a skálázhatóság maximalizálására. A Node.js a Google-féle V8 JavaScript-motorból, a libUV-ből és számos beépített könyvtárból áll. A könyvtárak telepítése front- és backend téren is szükséges a projektnél letöltés után, ezt az „npm i” paranccsal lehet megtenni konzol segítségével.

2.3. MongoDB

Az adatbázishoz MongoDB-t használtunk mivel ezt találtuk a legmegfelelőbbnek a projektünkhöz. A MongoDB nyílt forráskódú dokumentumorientált adatbázis szoftver, amelyet a MongoDB inc. (korábbi nevén *10gen*) fejleszt. A NoSQL adatbázisszerverek közé tartozik. A dokumentumokat JSON-szerű formátumban tárolja (BSON). A legnépszerűbb NoSQL adatbázis szoftver. Ez volt számunkra a legjobb megoldás, kötetlenebb és könnyebben irányíthatóbb, mellesleg a Node, illetve a React is sokkal jobban optimalizált erre. Az adatbázis létrehozásának három lehetséges útja volt, CLI, Compress, Atlas. Ezekben a különbség a vezérelhetőség, illetve a hozzáférés. A CLI az lokális, tehát semmiben nem különbözött volna a XAMPP-tól, egy szerveret futtat ezen CLI-n keresztül módosításokat végezhetünk és összeköthetjük ezt a szerveret az mi szerverünkkel. A Compress és az Atlas,

pedig alig eltérő és mi a legtöbb esetben ezeket használtuk. Ezeknél kaptunk egy grafikus felületet, ami megkönnyebbítette az adatbázis előzetes menedzselését. Ezek annyiban különböznek, hogy az adatbázis szervert maga a Mongo szervere futtatja, regisztráltunk és kiválasztottunk egy olyan szervert, ami a régiókban ingyenesen elérhető (Frankfurt). Az Atlas webes, a Compress pedig asztali alkalmazás, illetve az utóbbi lehetővé teszi, a tábláink, vagyis itt kollekcióink exportját, importját, illetve triggerok beállítását, amit itt szintén javascript nyelven tehetünk meg.

2.4. Postman

A backend-ünk végpontjainak manuális tesztelésére Postman-t használtunk, amivel előzetesen, sok hibát ki tudtunk szűrni és javítani. A Postman egy API platform a fejlesztők számára API-jaik tervezésére, elkészítésére, tesztelésére és iterálására és ez a világ legnagyobb API- központ a világon több mint 75 000 nyitott API-val.

2.5. Jest

A Jest frontend és backend tesztelés során is fontos volt számunkra, ebben kiviteleztuk az automatizált tesztelést, mind API mind renderek esetében. A Jest egy JavaScript tesztelésére szolgáló keretrendszer, amely elsősorban nagyszabású webalkalmazásokra lett optimalizálva. Rengeteg projektben használható és támogatja a JavaScript összes keretrendszerét.

2.6. Express

Az Express.js vagy egyszerűen az Express a Node.js háttér-webalkalmazás-keretrendszere, amely ingyenes és nyílt forráskódú szoftverként jelent meg az MIT licenc alatt. Webes alkalmazások és API-k kiépítésére tervezték. Erre a backend kapcsán volt szükségünk, hogy egyáltalán tudjunk szervert létrehozni, mivel ez a Node szerver egyik alapja.

3. Fejlesztői környezet

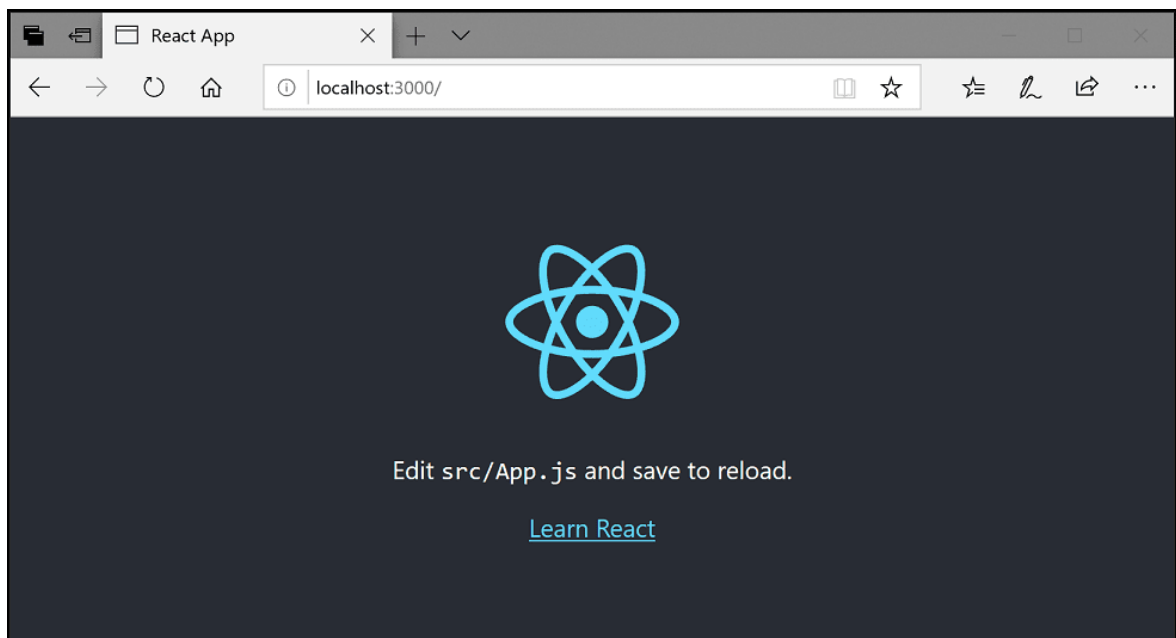
3.1. JavaScript

A JavaScript programozási nyelv egy objektumorientált, prototípus-alapú szkriptnyelv, amelyet weboldalakon elterjedten használnak. Eredetileg *Brendan Eich*, a Netscape Communications mérnöke fejlesztette ki; neve először *Mocha*, majd *LiveScript* volt, később „JavaScript” nevet kapott, és szintaxisa közelebb került a Sun Microsystems Java programozási nyelvéhez. A JavaScriptet először 1997–99 között szabványosította az ECMA „ECMAScript” néven. A jelenleg is érvényes szabvány az *ECMA-262 Edition 3*

(1999. december), ami a *JavaScript 1.5*-nek felel meg. Ez a szabvány egyben ISO szabvány is. A Microsoft 1995-ben kifejlesztette az Internet Explorert, ami a Netscape-el való böngészőháborúhoz vezetett. A Microsoft a Netscape Navigator JavaScript-feldolgozójának forráskódja segítségével létrehozta a sajátját, a JScriptet. A JScript először 1996-ban jelent meg a CSS kezdeti támogatása és a HTML néhány kiegészítése mellett. E megvalósítások merőben különböztek a Navigatorban alkalmazottaktól, ami megnehezítette a fejlesztőknek, hogy a weblapjaik mindkét webböngészőben jól működjenek, ami a „Netscape-en működik legjobban” és „Internet Exploreren működik legjobban” széles körű használatához vezettek sok éven át.

3.2. React

A React kód komponenseknek nevezett entitásokból áll. Ezek az összetevők újra felhasználhatók, és az SRC mappában kell őket létrehozni a Pascal Case névváltoztatást követően. Az összetevők a DOM egy adott eleméhez renderelhetők a React DOM könyvtár használatával. Egy komponens renderelésekor az értékeket a komponensek között átadhatjuk a "props"-on keresztül. React alkalmazás generálható az NPM (Node Package Manager) segítségével, de akár magunknak is felépíthető ez a struktúra Node inicializálás után, illetve készíthetünk olyan weboldalt, amiben csak néhány „div”-ben fut csak React alkalmazás, ezt konyhanyelven hibrid alkalmazásnak nevezik.



2. ábra NPM-ből generált React alkalmazás első indítása

3.Frontend

A frontend része Reactban készült el és próbáltuk minél egyszerűbbé tenni mivel szerintünk az egyszerű néha jobb alapon és úgy gondoljuk ez sikerült is.

A legtöbb dolgot kártyákkal valósítottuk meg. Azért ezt választottuk mivel ennek az elkészítése tűnt a legjobbnak és egyszerűnek a termékek és az ötletünk megvalósításának módjára. Az oldal legjobb megjelenítésének érdekében bootstrapet használtunk ez által telefonon és egyéb eszközökön is megfelelő a látvány, kivitelezés. A következő oldalakon látni lehet majd az oldalunk felépítését, kódolását. A frontend indítása konzolablakból az „npm start” paranccsal történik.

Amit még tudni kell mivel is készült és miket használtunk.

- Visual Studio Code
- React
- NPM (Node Package Manager) csomagok
- Bootstrap

3.1.Navigációs bár kivitelezése

A navigációs báron találhatóak meg a fő menü pontok, mint például egy bejelentkezés vagy regisztráció.

A regisztrációt vagy bejelentkezést követően a felhasználót a főoldalra fogja a program átirányítani. Itt a felhasználó már a megváltozott navigációs bárt fogja látni, ahol, láthatja, hogy természetesen eltűnt a bejelentkezés és a regisztráció gomb és helyette kapott egy profil fület, ahol a saját adatait tudja megtekinteni. A főoldalon található kettő darab slide (csúszka) ahol egy termék bemutató látható. Ez csúszka magától változik, de a felhasználó is tudja tovább léptetni.

G and B Shop Főoldal Információ Termékek ▾



3. ábra NavBar komponens, amikor a felhasználó be van jelentkezve

G and B Shop Főoldal Információ Termékek ▾

Bejelentkezés Regisztráció

4. ábra NavBar komponens, amikor senki sincs bejelentkezve

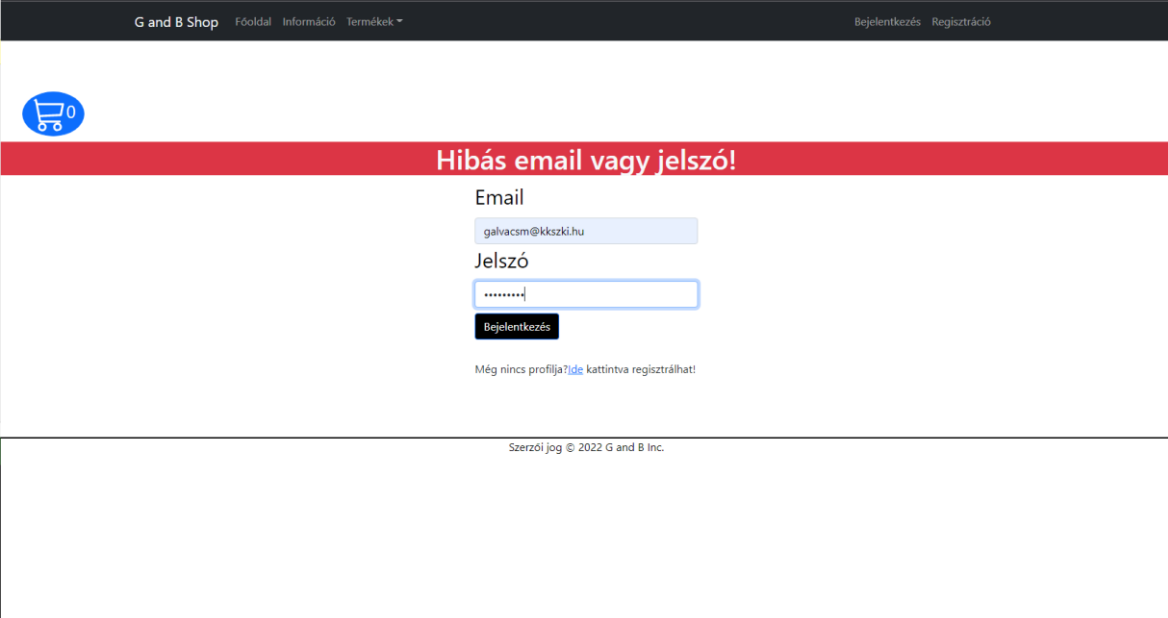
4. Menüpontok

4.1. Regisztráció

A regisztrációhoz formokat használtunk, ahol a felhasználónak meg kell adni a személyes adatait, ami később a vásárlás végrehajtásához kell, mivel nélküle nem lehet végrehajtani a folyamatot. A regisztráció végrehajtása közben, ha valamit nem tölt ki a felhasználó vagy esetleg rosszul adja meg a jelszót esetleg az emailt, akkor a program erre figyelmeztetni fogja róla. A regisztráció után már el is készült a saját profilja és kezdheti is a vásárlást. Sikeres regisztrációt követően a frontend a backendtől generált tokenet tárolja a böngésző localStorage tárhelyén, amit az első bejelentkezés után fog lecserélni tényleges access-tokenre, amennyiben az ideiglenes token egyezik a letárolttal.

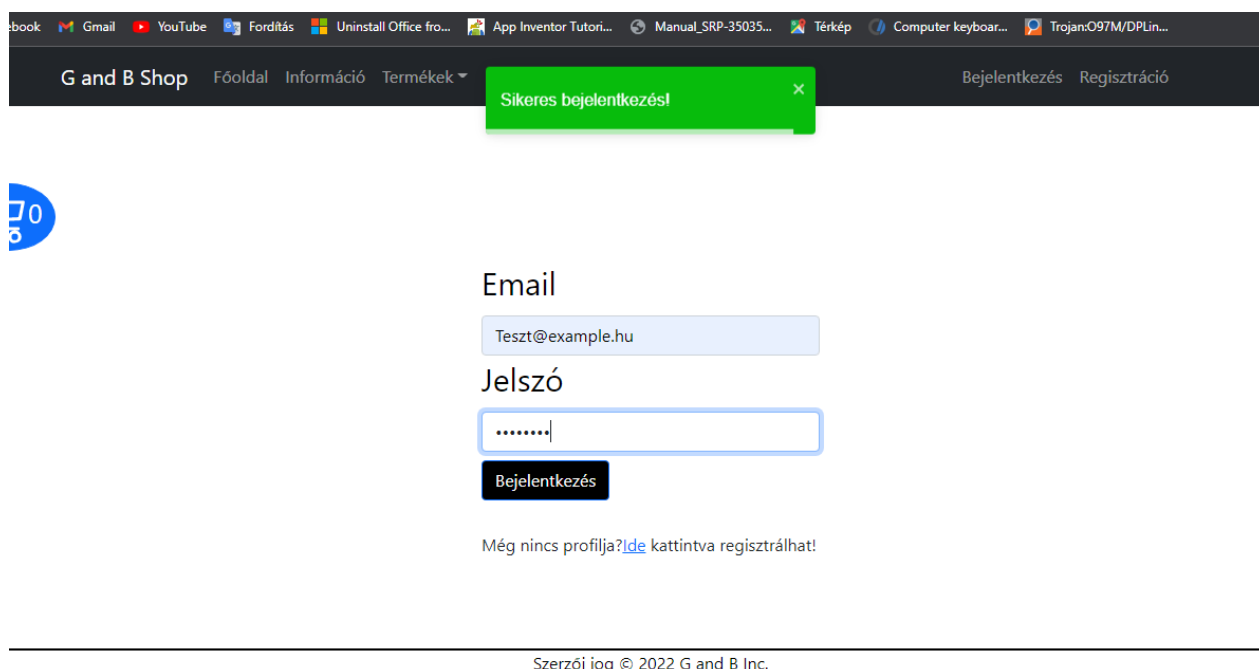
4.2. Bejelentkezés

A bejelentkezéshez a felhasználónak meg kell adnia az email címét és a jelszavát. Sikeres bejelentkezés esetén a főoldalra fogja a program dobni. Ha a bejelentkezés nem sikerült akkor fogja látni az adott személy, hogy ez mi miatt hiúsult meg. Ilyen hiba lehet például a jelszó elrontása vagy az email hiányossága, de bármi is a dolog hibája a program tájékoztatni fogja erről, illetve szerverhiba esetén is reagál, ezt a legtöbb esetben frontend részen validáltuk, viszont a hibaüzeneteket a backend küldi ki a felhasználó számára. Sikeres bejelentkezés esetén a böngésző localStorage-ában „userinfo” néven tároljuk a felhasználó adatait. A következő képeken látni lehet, azt mikor hibát talál az oldal és mikor sikeres a bejelentkezés.



The screenshot shows the login interface of the 'G and B Shop'. At the top, there is a dark navigation bar with the shop name and links to 'Főoldal', 'Információ', and 'Termékek'. On the right of this bar are links for 'Bejelentkezés' and 'Regisztráció'. Below the navigation bar, on the left, is a shopping cart icon with a '0' next to it. The main content area has a red banner at the top with the text 'Hibás email vagy jelszó!'. Below this banner, there are input fields for 'Email' (containing 'galvacsm@kkszk.hu') and 'Jelszó' (containing masked characters). A 'Bejelentkezés' button is positioned below the password field. At the bottom of the form area, a message states: 'Még nincs profilja? [Ide](#) kattintva regisztrálhat!'. The footer of the page contains the text 'Szerzői jog © 2022 G and B Inc.'

4. ábra Login üzenet sikertelen bejelentkezés esetén



5. ábra Login üzenet sikeres bejelentkezés esetén

4.3. Termék kiválasztás menüpont

A termékeket egy legördülő menüben találhatja meg a vásárló és nézheti meg mire kíváncsi.

6 termék fajtából tud választani, ezek a termékek a következők:

- Pulóverek
- Nadrágok
- Pólók
- Kabátok
- Cipők
- Kiegészítők

A termékfajta kiválasztását követően tudja kiválasztani a termék méretét. Emellett láthatja a termék árát és fajtáját is.

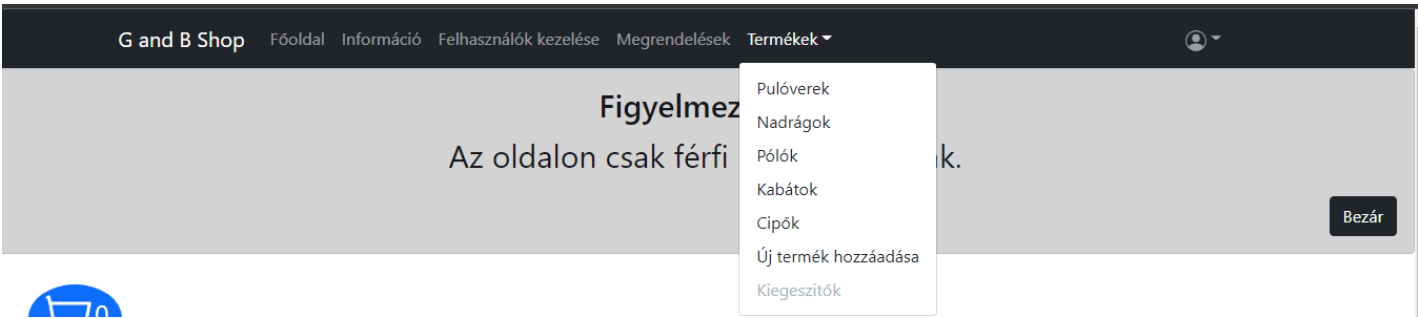
A ruházattokat kártyákba helyeztük. A telefonos használatkor ezek a kártyák egymás alá ugranak a bootstrapnek köszönhetően, így nem probléma más eszközök használata sem az oldalon. Ha valaki sikeresen megtalálta amire szüksége van méret kiválasztás után a rendelésre kattintva kosárba teheti, amennyiben egy adott méretnél többször kattintunk a rendelés gombra a kosárban halmozza az adott termék mennyiségét, ezt Toastify üzenetként látni is lehet.

4.4. Frontend változásai adminisztrátori joggal

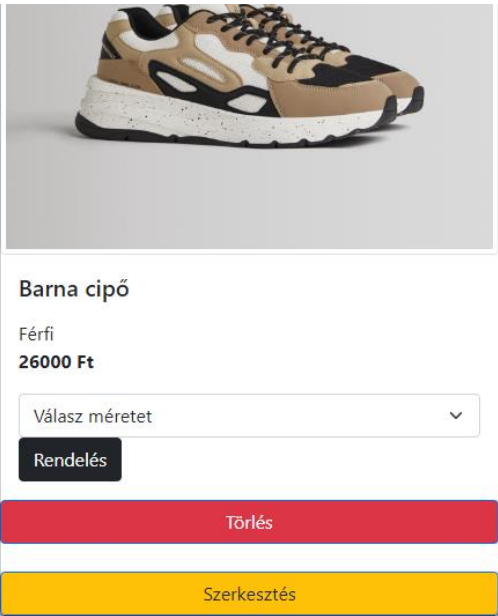
Az oldalunkhoz nem készült admin felület, ezért a frontendbe építettük ezt fel. Amennyiben a felhasználó egy adminisztrátor, amit egy adatbázisban tárolt logikai érték határoz meg, akkor számára sok új menüpont elérhető.

- Új termék hozzáadása
- Felhasználók kezelése
- Megrendelések kezelése
- A termékeknél minden kártyán megjelenik ez esetben 2 gomb, amivel elérhető egy adott termék szerkesztése, illetve törlése.

Ezen funkciók useEffecttel vizsgálva, le lettek tiltva egy átlagos felhasználó részére.



6. ábra Navbar változásai admin felhasználó esetén

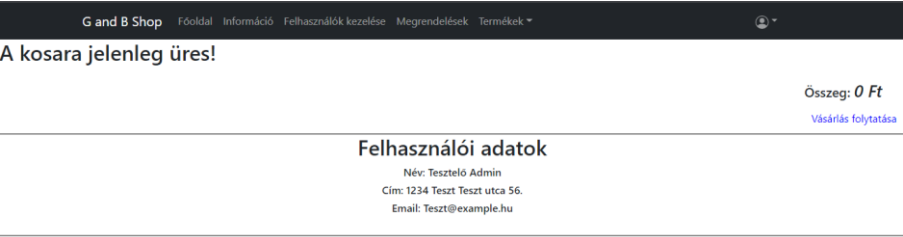


7. ábra Termék kártyák változása admin felhasználó esetén

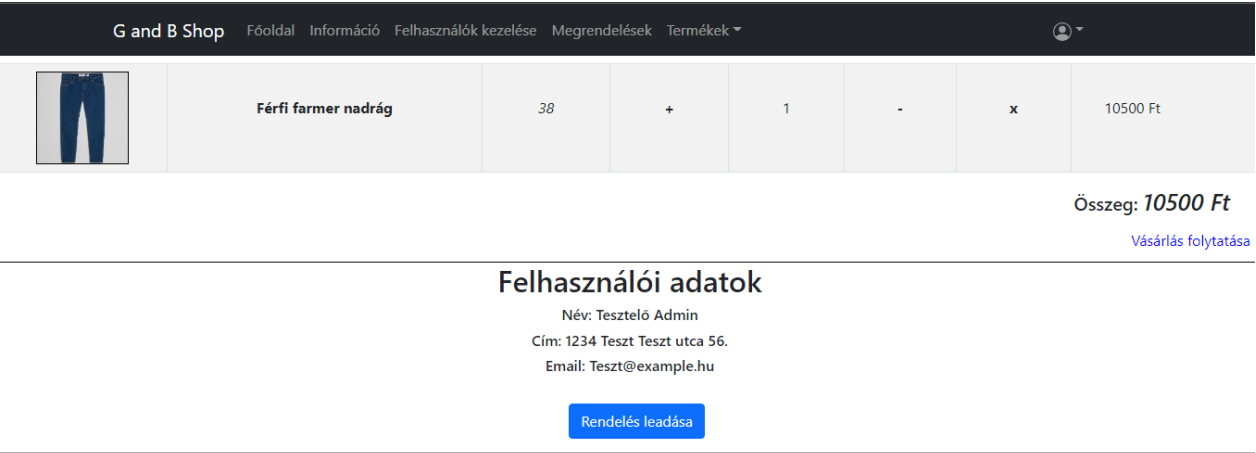
Minden fontosabb változtatásnál megkérdezi az admin felhasználót előtte, hogy biztos véghez szeretné-e vinni az adott módosítást. Későbbiek során ehhez még szeretnénk tenni egy validációt, ami az admintól jelszót vár a módosítások előtt. A módosítások állapotát a backend adja vissza a frontend számára. Az admin képes a felhasználók törlésére és más adminok jogának elvételére is, ezeket a „Felhasználók kezelése” menüpontban képes megtenni, ezen felül pedig a megrendelések menedzselésére is van lehetősége a „Megrendelések” menüpontban. A megrendeléseknél is kártyák jelennek meg, aminek színe mutatja állapotát, emellett a felhasználóknak és a termékeknek csak az azonosítója szerepel a kártyán, de ezekre kattintva modal ablakban megjelennek a bővebb adatok, amelyek szükségesek a rendeléshez.

4.5.Cart komponens- Kosár

A rendelések lebonyolításához, kosarat alkalmaztunk, localStorage segítségével, ott „cart” néven megszólítható. A kosár logója az admin oldalakat kivéve minden oldalon megjelenik és ez visz minket oda, illetve minden kosárba történő termék feltöltése után 3 másodperc múlva a kosárba navigál minket a rendszer. A rendelésnek előfeltétele a regisztráció, illetve, hogy a felhasználó be legyen jelentkezve, ezt a kosár jelzi is felénk, amennyiben ez nem történt meg, ellenkező esetben a felhasználó részleges adatai kerülnek kiírásra. A komponensen belül látjuk a megrendelt termékeinket, árát és mennyiségét, valamint, ha nincs termék, akkor a „A kosara jelenleg üres!” üzenetet látjuk. A termékek mennyisége növelhető és csökkenthető, az összeget a rendszer automatikusan számolja és kapott egy gombot, ami vissza navigál bennünket az előző oldalra.



8. ábra Üres kosár



9. ábra Kosár termékkel

5.Backend

5.1.Tervezés

Az elején nem gondoltuk volna, hogy a backend tervezése is legalább annyi időt vehet igénybe, mint a frontend kivitelezése. Kezdetben megbeszéltük a funkciókat és az útvonalak terveit és utána dolgoztunk megosztva. Így a backend Postman tesztek segítségével készült és későbbiek során lett összekötve a kész frontenddel. A backendünket először MySQL és ASP.net-ben képzeltük, mivel a tanulmányaink ezen szakaszán, ezzel

találkoztunk legtöbbször, el is kezdtük felépíteni ezt a rendszert, de sajnos a közepén kellett rájönnünk, hogy ez nem tudja feltétlenül, amit mi szeretnénk, habár az ASP-vel játható lett volna a munka az SQL nagyon sok helyen érezte velünk a kötöttséget és emiatt nem tudtunk teljesen szabadon gondolkodni. Az egyik iskolai projekt kapcsán találkoztunk a Javascript alapú szerverekkel és úgy döntöttünk ebben készítjük a projektünket.

5.2. Adatbázis

Mikor találkoztunk a MongoDB-vel, szinte megvilágosodtunk, illetve meg is könnyebbültünk, hiába dobhattunk több adatbázistervet a kukába. Tiszta lap, új ötletek, egyszerű megvalósítás ez volt a terv. A kezdeti öt, hat táblát sikerült leredukálnunk három a Mongóban használt collection vagyis magyarul kollekciónak. Nagy előny volt számunkra, hogy egy ilyen kollekciónak sokkal többféleképpen lehet adatot eltárolni, akár egy rekordban, amit itt inkább line-nak említenek, eltárolhatunk akár objektumot, vagy a javascripthez hasonló tömb struktúrát, array-t. A terveinken még látszódik azért, hogy SQL szerű volt a gondolkodásunk, mivel sokkal egyszerűbben, akár 1 vagy 2 kollekciónal megoldhattuk volna, de ezeket az új ismereteinkkel biztosan majd tovább fejlesztjük. Úgy kezdtük, hogy felmentünk a MongoDB weboldalára, beeregistráltunk, majd az Atlas felületen létrehoztunk egy adatbázist „szakdolgozat” néven. Ezután megtettük az alapvető beállításokat, amit a rendszer kért tőlünk, ezekhez tartozik, hogy be kellett állítanunk, hogy milyen IP-ről legyen megszólítható az adatbázis, illetve készítenünk kellett felhasználókat a Connection String-hez. Ezeknek a felhasználóknak, mi határozhattuk meg, hogy milyen feladatköre, jogrendszere lehet, ezzel a későbbiekben említve szerettünk volna játszani és esetlegesen a továbbfejlesztésben belevinni a rendszerbe. A projekt jelenleg a „teszt” felhasználónevű és „asdasd123” jelszóval ellátott, írás és olvasás joggal rendelkező profilt használja, valamint minden IP címről megszólítható, bár ez csak a fejlesztési környezet miatt, ezt a MongoDB a beállításnál hangsúlyozza, hogy nem ajánlott és nem megbízható, a szerver gép IP címét kell megadni.

A „szakdolgozat” nevű adatbázisunkban három kollekciónak található, felhasználó, termékek, megrendelések. Ezeket a projekt leadásakor json kiterjesztésű fájlként a Mongo Compass-ból exportálva a „szakdolgozatbackend” „DataBase” könyvtárban megosztottuk és helyesen a következőket kellene tartalmaznia: felhasználó (2 line) => 1 admin és 1 átlagos felhasználó, megrendelések (üres), termékek (21 line) => termékek (pulóver, nadrág, póló, cipő és kabát) tagolva.

Az ASP.net-hez hasonlóan a NodeJs-ben is képesek vagyunk modell létrehozására. Mivel SQL-ben meg tudjuk mondani, milyen adatokat tartalmazhat egy rekord, ott nincsen nagyobb jelentősége, de a Mongonál eléggé nagy szerepe van, mivel a line-ok különbözhetnek egymástól, ami nem minden esetben volt számunka jó. Ezért készült 2 modellünk, a felhasználók és a products (termékek). Ezeket a „szakdolgozatbackend/models” mappában lehet megtalálni és ezek tartalmazzák azt, hogy a felhasználó vagy termék milyen típusú adatokat tartalmazzon és az ehhez tartozó kulcsok neve. Szemléltetéskeppen:

felhasznalok.js

nev	type: String	required:true	-----
cim	type: String	required:true	-----
felhasznalonev	type: String	required:true	unique: true
jelszo	type: String	required:true	-----
email	type: String	required:true	unique: true
isAdmin	type: Boolean	required:true	-----

products.js

termekNev	type: String	required:true
meret	type: Array (JSON String)	required:true
Ar	type: Number	required:true
link	type: String	required:true
Tipus	type: String	required:true

A felhasznalok.js el lett látva egy function-el, mivel az SQL hiánya nélkül nem volt alapértelmezett md5 kódolás számunka, így ehhez a „bcrypt” harmadik féltől származó könyvtárat használtuk, ami tipikusan magába rejt egy hash-elésre alkalmas függvényt, amit kicsit átalakítva elértük, hogy a jelszó SALTLY hashelve kerüljön már letárolásra. Illetve a link kulcs alatt tárolt érték egy tényleges link, amelyet a Cloudinary webAPI segítségével hoztunk létre, majd mentettük el az adatbázisban, a projektben használt képeket a Cloudinary

szerverén tároljuk, amely ingyenes adatbázis képek számára és erősen támogatja mind a NodeJS szerver, mind a React-et.

Az utolsó kollekcióban, ami a megrendelesek, nem éreztük szükségét annak, hogy meglegyen szabva, hogy milyen legyen az alap, mivel egy ember akár több dolgot is rendelhet, illetve a frontendben elég sok helyen megtudtuk fogni azt, hogy hibás adatot ne lehessen felvinni, főleg mivel a megrendelés az SQL- hez hasonló módon, egy gyűjtő, a meglévő kollekciókból kaphat csak adatot és ezeknek az adatoknak is legalább a felét automatikusan adja át a böngésző a szervernek. Illetve, ami szembetűnő lehet, hogy sehol sem írtam ki az egyedi azonosítót, „id”-t. Ez azért van, mivel a mongo rendszere arra sajnos nem enged túl sok szabadkezet, minden line-hoz rendel egyedi azonosítót, amely kicsit eltérő a megszokott formától, ugyanis ez nem szám, pontosabban nem is számérték, ez egy megadott hosszúságú úgynevezett, ObjectId, ami csak a MongoDB rendszerében található típus. A Mongo dokumentációja szerint, ez egy számból és letárolási időből hashelt string típusú sor, ezt a line-ban mindig „_id” kulccsal jelenik meg. Megfelelő függvénnyel, akár vissza is fejthető letárolás idejére (dátum) formátumba és int típusú számra, amennyiben ez szükséges, illetve validálható, így nem kell felesleges kérést kiküldeni, ha invalid az azonosító egyszerűen megszakítjuk a folyamatot.

A projektben szükségünk volt 2 könyvtárra a mongot illetően, amelyek a „mongodb” és „mongoose” névre hallgatnak.


5.3.Szerver


A szerver Node.js inicializálásával hoztuk létre az „index.js” fájlban, elindításához a „npm run dev” parancsot kell használni a konzolban, ezek után a szerver a „localhost:5501”- es porton lesz megszólítható. Ehhez szükségünk volt az Express.js keretrendszer könyvtárára. A ConnectionString és egyéb fontos érték elrejtését a kódból „dotenv” harmadik féltől származó könyvtár segítségével oldottuk meg és a „env” fájlban megtalálhatóak, amelyet a projekt leadásakor a „szakdolgozatbackend” mappában nyilvánossá tesz. A szerver képes végrehajtani a CRUD műveleteket valahol teljes palettában, néhol pedig részletesen, ahol erre nincs szükség az üzleti logika terén, valamint képes az autentikációra és az adatbázis kapcsolatra. Véleményünk szerint ez a fajta JavaScript kód merőben eltér a hagyományostól, de könnyen megérthető és ennek elsajátítása nagyban segített, más technológiákkal készült backend megértésében is.


5.4.Útvonalak


5.4.1.Termékek


A termékekre úgy gondoltuk, hogy az általános CRUD műveleteket szeretnénk végrehajtani, mivel ezt a lehetőséget, úgyis csak adminisztrátori joggal rendelkező felhasználó használhatja, így feltételezzük, hogy tudja mit szeretne csinálni. Útvonalak röviden:

„/termekek”  „GET” metódussal, megszólítva egy általános lekérdezés, ami lekérdezi az összes terméket.


„/termekek/:id”  „GET” metódussal megszólítható útvonal, az URL-ben megadott ID-val rendelkező terméket, adja vissza, invalid azonosító esetén „invalid id” valid, de nem létező azonosító esetén „not found” üzenettel tér vissza.


„/termekek”  „POST” metódussal megszólítható útvonal, amely paraméterként vár, terméknevet, árat, típust, méretet és linket. Méretet, front-endben meghatározottan tömbben összegyűjtve, a linket pedig kigenerálva kapja meg, a cloudinary webAPI-ból, ennek működése olvasható a „cloudinary” résznél. Sikeres kérés esetén „sikeres feltöltés”, sikertelen feltöltés esetén „bad request” státuszt és „hiba!” üzenetet kapunk vissza.


„/termekek/:id”  „PUT” metódussal megszólítható útvonal, amely paraméterként várja egyazonokat a bemeneteket, mint a „POST”-nál, sikeres kérés esetén a termékkel tér vissza, invalid ID esetén „invalid id”, hibás kérés esetén „hiba!”, valamint, ha nem található az adott ID „not found!” üzenettel tér vissza.

„/termekek/:id”  „DELETE” metódussal megszólítható útvonal, sikeres törlés esetén az ID-val tér vissza, ellenkező esetben a már megszokott üzenettel.

5.4.2.Felhasználók

„/felhasznalok”  „GET” metódussal, megszólítva egy általános lekérdezés, ami lekérdezi az összes felhasználót.

„/felhasznalok:id”  „GET” metódussal megszólítható útvonal, az URL-ben megadott ID-val rendelkező felhasználót, adja vissza, invalid azonosító esetén „invalid id” valid, de nem létező azonosító esetén „not found” üzenettel tér vissza.

„/felhasznalo”  „POST” metódussal megszólítható útvonal, amely paraméterként vár, nevet, felhasználónevet, emailt, jelszót, címet, illetve egy

automatikusan false értékkel isAdmin paraméteret. Ezt használjuk a regisztrációhoz, sikeres regisztráció esetén, az adatokkal tér vissza, illetve egy JWT által generált access-tokennel. Hibás kérésnél „Hibás adatbevitel!” üzenet jelenik meg.

„/felhasznalo/login” —————> „POST” metódussal megszólítható útvonal, amely paraméterként vár felhasználónevet és jelszót, amennyiben az adatbázisban szerepel az email-jelszó páros, akkor sikeresnek könyveli el a kérést és 200-as státuszkód mellett, visszaadja a felhasználó adatait és access-tokenjét, hibás adatok megadásakor, 400-as (bad request) státuszkóddal és „hibás email vagy jelszó” üzenettel tér vissza.

„/felhasznalok/:id” —————> „DELETE” metódussal megszólítható útvonal, sikeres törlés esetén az ID-val tér vissza, ellenkező esetben „hiba a törléssel” üzenettel.

„/felhasznalok/:id” —————> „PUT” metódussal megszólítható útvonal, amely bemenetnek várja a regisztrációnál is használt paramétereket és változás esetén módosítja az adatbázisban a felhasználó adatait. Sikeres kérés esetén a felhasználó előző adataival tér vissza, hiba esetén 400-as státuszkóddal és „hibás bevitel” üzenettel.

„/allapotfel/:id” —————> „PUT” metódussal megszólítható útvonal, ez nem vár paramétert, az adott ID-val rendelkező felhasználó „isAdmin” státuszát állítja igaz értékre.

„/allapotLe/:id” —————> „PUT” metódussal megszólítható útvonal, ez nem vár paramétert, az adott ID-val rendelkező felhasználó „isAdmin” státuszát állítja hamis értékre.

Az adatbázisban 2 felhasználót hoztunk létre tesztelés céljából:

Email: TesztUser@example.hu

Email: Teszt@example.hu

Jelszó:user1234 =>hashelve

Jelszó:admin123 =>hashelve

5.4.3.Megrendelesek

„/megrendelesek” —————> „GET” metódussal megszólítható útvonal mely, visszaadja az összes terméket az adatbázisból.

„/megrendelesek/:id” —————> „DELETE” metódussal megszólítható útvonal, sikeres törlés esetén az ID-val tér vissza, ellenkező esetben „hiba a törléssel” üzenettel.

„/megrendelesek/:id” —————→ „GET” metódussal megszólítható útvonal, amely az URL-ből kiolvasott ID-val rendelkező megrendeléssel tér vissza.

„/megrendelesek/:fId” —————→ „POST” metódussal megszólítható útvonal, amely a felhasználótól tényleges értéket nem vár, a frontendben feldolgozott JSON kérést tárolja el (felhasznalo (id), megrendelt_termek (array), aktiv (boolean: false), lezart (boolean: false), datum (Date.now() generálja)).


„/megrendelesek/:id” —————→ „PUT” metódussal megszólítható útvonal, ez nem vár paramétert, az adott ID-val rendelkező megrendelés „aktiv” státuszát állítja igaz értékre.

„/megrendelesekLez/:id” —————→ „PUT” metódussal megszólítható útvonal, ez nem vár paramétert, az adott ID-val rendelkező megrendelés „lezart” státuszát állítja igaz értékre.

6. Tesztelés

6.1. Postman tesztek

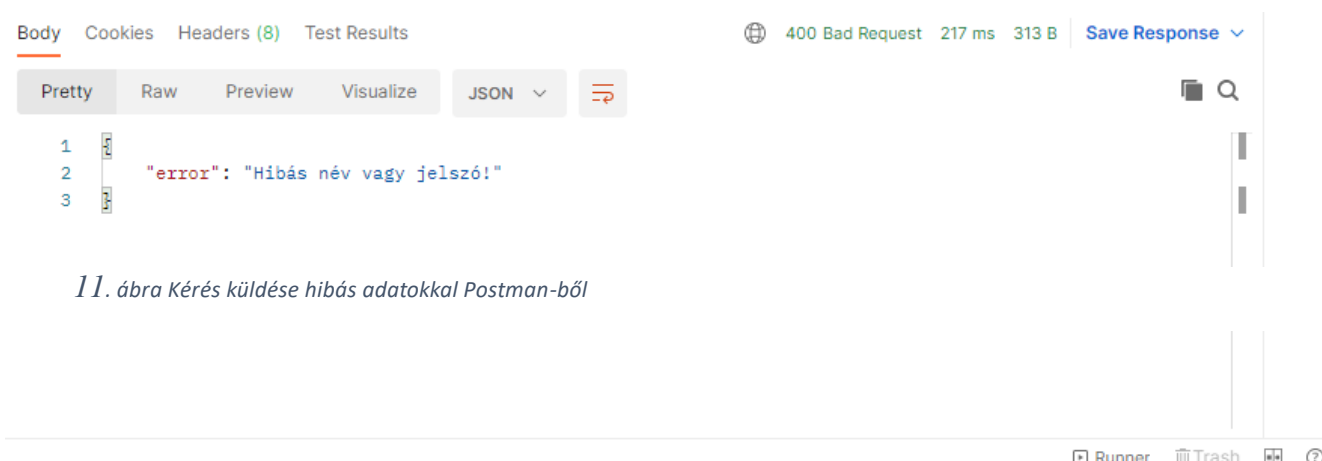
A munkafolyamatok felosztása végett szükségünk volt a backend készítésénél egy külső programra, ugyanis a backend kevésbé időigényes, illetve szükségünk volt arra, hogy lássuk azt, hogy egy API, hogyan is működik. A manuális és fejlesztés közbeni API tesztelésre Postman-t használtunk, mivel az iskolai tanulmányaink alatt ezzel ismerkedtünk meg a legjobban, illetve széleskörben használják. Több információ a 2.4. Postman címnél található a programról. A fejlesztés folyamán rengeteg hibát tudtunk kiszűrni ennek a segítségével backend szinten és sikerült minden esetben azt a választ kapnunk a szervertől, amit szeretnénk volna. Egy példa a „/felhasznalo/login” megszólításakor, visszatért a felhasználó adataival és a tokennel.



The screenshot shows the Postman interface with the 'Raw' tab selected. The response is a JSON object containing user details and a token. The JSON is as follows:

```
1 {
2   "_id": "625b57b07b4ec3045ced89fe",
3   "nev": "Teszt Felhasználó",
4   "felhasznalonev": "TesztUser",
5   "email": "TesztUser@example.hu",
6   "cim": "4321 Teszt Teszt utca 12.",
7   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYyNWY1N2IwN2I0ZWZMDQ1Y2Vkd0lmZSIsIm1hdCI6MTY1MDgyMzQ1NiwiZXhwIjojUxNjg3NDU2fQ.i_EkYQWRfIdgfyEJiHKT-3PADb7X8N6JPKgKIR1Gdo",
8   "isAdmin": false
9 }
```

10. ábra Kérés küldése megfelelő adatokkal Postman-ből



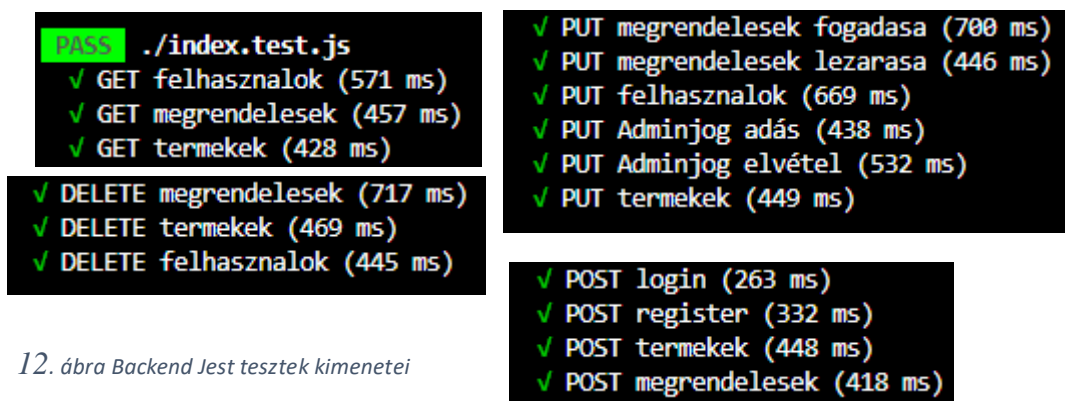
11. ábra Kérés küldése hibás adatokkal Postman-ből

6.2.Jest tesztek

A Jest volt számunkra a legelőnyösebb az automatizált tesztek készítésében frontend és kiegészítve backend téren is, mivel mindkettő JavaScript alapú. Először GitLab automatizált tesztel próbálkoztunk, de sajnos azzal nem működött úgy, ahogy szeretnénk volna. Kezdetben azt hittük a weboldal megfelelő működése és a Postman tesztek miatt, hogy a GitLab tesztek ok nélkül futnak hibára, viszont miután elkészítettük a Jest teszteket rá kellett jönnünk arra, hogy itt-ott maradt a rendszerben hiba, amire sajnos a GitLab pontos okot nem tudott mondani, de a Jest igen, ezek kijavítását addig csináltuk, amíg backend és frontend téren a tesztek szerint hibátlan, bár egy biztos, hogy hiba mindig lesz, de ezzel ki tudtuk szűrni a nagyobb hibákat, még az utolsó percekben is.

6.2.1.Jest teszt- Backend

A backend -ben szükségünk volt a „supertest” és a „jest” könyvtárakra, majd egy személyes parancs létrehozására, amely az „npm run test” -el hívható meg és elindítja a tesztünket. A teszt megtalálható az „index.test.js” fájlban, ami elsősorban API-kat tesztel és ezért volt szükségünk a supertest-re mivel azt teszi lehetővé, viszont néhol a hibátlan működéshez, valós létező adatokat kell megadnunk.



12. ábra Backend Jest tesztek kimenetei

6.2.2.Jest teszt- Frontend

Frontend téren egyszerűbb volt a tesztelés, mivel, ha az npm által generált React alkalmazást használjuk, abban automatikusan szerepel teszt fájl „App.test.js” néven. A jest könyvtár telepítése után, ezt a fájlt szerkesztettük, ahol a rendereket szerkesztettük és néztük meg.

```
✓ render Navbar (185 ms)
✓ render Polo (95 ms)
✓ render Profil (333 ms)
✓ render pulover (91 ms)
✓ render regisztracio (162 ms)
✓ render szerkesztes (118 ms)
✓ render új termék hozzáadása (234 ms)
✓ render login (173 ms)
```

13. ábra Jest tesztek kimenetei Frontend esetén

```
PASS src/App.test.js (10.06 s)
✓ render App (224 ms)
✓ render Home (124 ms)
✓ render cart (133 ms)
✓ render cipő (91 ms)
✓ render Elozmeny (182 ms)
✓ render ErrorPage (50 ms)
✓ render Felhasznalok (212 ms)
✓ render informacio (61 ms)
✓ render Kabat (153 ms)
✓ render Kiegészitok (31 ms)
✓ render loading (11 ms)
✓ render megrendeles (39 ms)
✓ render Nadrag (212 ms)
```

7.Fejlesztési tervek

A projekt folyamán nagyon sok mindenben kellett, egyszerűsíteniünk, de ha valóban szeretnénk buildelni az oldalunkat, a legfontosabb a fizetés funkció beépítése, ezt megtehettük volna a „SimplePay” webAPI használatával, de ennek beépítése tényleges fizetést várt volna el. Valamint szeretnénk a reszponzivitáson javítani és később akár mobilalkalmazást készíteni react native segítségével és profilkép feltöltést biztosítani.

8.Források

MongoDB Documentation, <https://www.mongodb.com/docs/> , letöltés dátuma: 2022.04.24.

MongoDB, <https://hu.wikipedia.org/wiki/MongoDB> , letöltés dátuma: 2022.04.24.

JavaScript, <https://hu.wikipedia.org/wiki/JavaScript> , letöltés dátuma: 2022.04.14.

What is Express.js?, <https://www.besanttechnologies.com/what-is-expressjs> , letöltés dátuma: 2022.04.24.

Express.js, <https://en.wikipedia.org/wiki/Express.js> , letöltés dátuma: 2022.04.24.

Node.js, <https://hu.wikipedia.org/wiki/Node.js> , letöltés dátuma: 2022.04.24.

About Node.js, <https://nodejs.org/en/about/> , letöltés dátuma: 2022.04.24.

React, <https://hu.reactjs.org/> , letöltés dátuma: 2022.04.24.

Postman (software), [https://en.wikipedia.org/wiki/Postman_\(software\)](https://en.wikipedia.org/wiki/Postman_(software)) , letöltés dátuma: 2022.04.24.