



győri szakképzési centrum

Jedlik Ányos  
Gépipari és Informatikai  
Technikum és Kollégium



9021 Győr, Szent István út 7.

+36 (96) 529-480

+36 (96) 529-448

OM: 203037/003

jedlik@jedlik.eu

www.jedlik.eu

## Záródolgozat feladatkiírás

Tanuló(k) neve<sup>1</sup>: Tamás Róbert, Rabóczki Erik, Gonda Szabolcs Krisztián  
Képzés: nappali  
Szak: 5 0613 12 03 Szoftverfejlesztő és tesztelő technikus

### A záródolgozat címe: „NASAPC” PC webáruház

Konzulens: Horváth Norbert  
Beadási határidő: 2022. 04. 29.

Győr, 2022. 04. 29

---

**Módos Gábor**  
igazgató

---

<sup>1</sup> Szakmajegyzékes záródolgozat esetében több szerzője is lehet a dokumentumnak, OKJ-s záródolgozatnál egyetlen személy ad le záródolgozatot.



győri szakképzési centrum

Jedlik Ányos  
Gépipari és Informatikai  
Technikum és Kollégium



9021 Győr, Szent István út 7.

+36 (96) 529-480

+36 (96) 529-448

OM: 203037/003

jedlik@jedlik.eu

www.jedlik.eu

## Konzultációs lap<sup>2</sup>

	A konzultáció		Konzulens aláírása
	ideje	témája	
1.	2022.02.15.	Témaválasztás és specifikáció	
2.	2022.03.14.	Záródolgozat készültségi fokának értékelése	
3.	2022.04.17.	Dokumentáció véglegesítése	

## Tulajdonosi nyilatkozat

Ez a dolgozat a saját munkánk eredménye. Dolgozatunk azon részeit, melyeket más szerzők munkájából vettünk át, egyértelműen megjelöltük.

Ha kiderülne, hogy ez a nyilatkozat valótlan, tudomásul vesszük, hogy a szakmai vizsgabizottság a szakmai vizsgáról kizár minket és szakmai vizsgát csak új záródolgozat készítése után tehetünk.

Győr, 2022. április 15.

Tanulók aláírásai:

Rabóczki Erik

Tamás Róbert

Gonda Szabolcs Krisztián

<sup>2</sup> Szakmajegyzékes, csoportos konzultációs lap

## Tartalomjegyzék

<b>1. Bevezetés:</b>	<b>5</b>
<b>2. Specifikáció:</b>	<b>6</b>
2.1 Projekt fejlesztői és a munkamegosztás:	7
<b>3. Adatbázis</b>	<b>8</b>
<b>4. JavaScript</b>	<b>10</b>
4.1 A JavaScript története	11
4.2 Működése	11
4.3 Mire képes a JavaScript?	12
<b>5. Node.js</b>	<b>13</b>
5.1 Célja	13
5.2 Modulok	14
5.3 npm	14
5.4 Express	15
5.4.1 Indoklás	15
5.4.2 Előnyök	16
5.5 Express funkciók	17
5.5.1 Gyorsabb szerveroldali fejlesztés	17
5.5.2 Middleware	17
5.5.3 útválasztás	17
5.5.4 Sablonmotorok	17
5.5.5 Hibakeresés	18
5.5.6 Összegzés	18
<b>6. Frontend</b>	<b>20</b>
6.1 Navigation komponens	20
6.2 Footer komponens	21
6.3 Home page	21
6.4 About us	21
6.5 Product (user)	22
6.6 Signup és Login	23
6.7 Wishlist	26
6.8 Orders és a Cart	27
6.9 Admin products	28
6.10 Password change	30
6.11 Selenium teszt	31
6.12 Függőségek	33



<b>7. Backend.....</b>	<b>35</b>
7.1 Bevezetés .....	35
7.2 Belépő fájl .....	35
7.3 Termékek lekérdezése .....	35
7.4 Regisztráció (Register & Verify Account) .....	36
7.5 Bejelentkezés.....	38
7.6 Új jelszó igénylése.....	39
7.7 Kívánság lista (WishList) .....	40
7.8 Megrendelések (Orders).....	41
7.9 Értékelések (Reviews).....	42
7.10 Admin.....	43
7.10.1 Termék létrehozása .....	44
7.10.2 Termék módosítása .....	45
7.10.3 Termék törlése.....	46
7.11 Backend teszt.....	47
7.11.1 Bevezetés.....	48
7.11.2 Bejelentkezés .....	49
7.11.3 Munkamenet (Session) .....	49
7.11.4 Termékek / Termék ID alapján.....	50
7.11.5 Kívánság lista (WishList).....	51
7.11.6 Termék létrehozása .....	52
7.11.7 Termék módosítása .....	52
7.11.8 Termék törlése.....	53
7.11.9 Új jelszó igénylése .....	53
7.11.10 Termék módosítása (Modify product).....	54
<b>8. Android app .....</b>	<b>54</b>
8.1 Android Alkalmazás.....	54
8.2 Az alkalmazás felépítése:.....	55
<b>9. Kommunikáció a csapattagok között: .....</b>	<b>59</b>
<b>10. Felhasznált források: .....</b>	<b>59</b>

# 1. Bevezetés:

Manapság 2022-ben és az elmúlt években egyre nagyobb teret hódítottak az online vásárlási felületek, és az IT szféra fejlődésének köszönhetően egyre jobban arra törekszünk, hogy amit csak lehet, azt valamilyen online felületen végezzük; ez az ügyintézésről fogva a megbeszélések (meeting) és találkozók megtartásán keresztül egészen a vásárlásig terjedt ki. Ezért döntöttünk úgy, hogy mi sem állunk a változás útjába és a növekvő igénynek megfelelően, mi is ezen a területen szeretnénk létrehozni egy projektet, amely teljesen életképes a ma, azaz 2022 igényeinek, teljes mértékben megfelelő és kielégítő felhasználói élményt nyújt.

A weboldal tárgyát/témáját illetően több lehetőség is megfogalmazódott;

Például:

- Bútorbolt
  - Indoklás: A környezetünkben több ilyen irányultságú cég található, a projektet megpróbáltuk volna náluk értékesíteni.
  - Elhatározás: Végül is úgy döntöttünk, hogy más ötleteket is megvizsgálunk.
- „OnlySelfies” online tartalommegosztó oldal
  - Indoklás: Magas kereslet a hasonló tartalommegosztó oldalak iránt.
  - Elhatározás: Ezt a praktikát etikátlannak találtuk és így az ötletet elvetettük.
- Online kaszinó
  - Indoklás: Az online tartalomgyártók promócióinak köszönhetően, ez az iparág világszerte nagy teret hódított az elmúlt pár év alatt.
  - Elhatározás: A piac telítettsége miatt más ötlet mellett döntöttünk.



## 2. Specifikáció:

Végsősoron egy webáruház megvalósítása mellett döntöttünk. Mivel napjainkban az online vásárlás nagy teret hódított, ezért kézenfekvőnek tűnt, hogy mi is egy webshopot készítsünk. Véleményünk szerint ez a projekt kellően összetett és a modern igényekhez mérten aktuális ahhoz, hogy az elvárásoknak megfeleljen.

A projektünk egy teljes értékű, adatbázissal támogatott webáruház, különválasztott backend és frontend használatával.

A webshop teljesértékű felhasználói élményt nyújt: a vásárló tud böngészni a feltüntetett termékek között, rá tud keresni konkrét termékekre, vagy kategóriák szerint szűrni. A kiválasztott termékről tud bővebb információhoz jutni és akár regisztráció nélkül is a kiválasztott terméket kosárban tudja tárolni, továbbá a felhasználó regisztráció nélkül is képes rendelést leadni.

Szükségünk van keretrendszerre, illetve azok alkalmazására, hiszen ezeknek a segítségével, jóval gyorsabban dolgozhatunk, gyorsabban valósíthatjuk meg az adott funkcionalitásokat. A keretrendszer lényege az, hogy a különböző alkalmazásokban legnagyobb gyakorisággal használt elemeket egy helyre csoportosítja, gyakorlatilag készen kínálva ezeket a fejlesztők számára. A programozók így nagyon sok elvégzendő munkától szabadulnak meg. A keretrendszerek nagymértékben könnyítik az egyes tipikus feladatok elvégzését. A komplexebb folyamatokat leegyszerűsíti, tiszta és újra felhasználható kódot írhatunk velük. Megkönnyíti a hibakeresést és a tesztelést, és így rugalmasabb alkalmazásokat eredményez.

Mindezek következtében úgy döntöttünk, hogy a backend-et Express.js, a frontend-et Vue.js segítségével írjuk meg. Adatbázishoz NoSQL-t használunk, amit MongoDB-vel fogunk kezelni.



## 2.1 Projekt fejlesztői és a munkamegosztás:

Ezt a projektet egy három fős fejlesztői csapat alkotta melynek tagjai:

- Rabóczki Erik
- Gonda Szabolcs Krisztián
- Tamás Róbert

A tagok közötti munkamegosztás a következő módon történt:

### **Tamás Róbert:**

- Backend fejlesztéséhez szükséges technológiák kiválasztása
- Backend létrehozása és fejlesztése
- Backend teszt létrehozása
- Backend és az ehhez tartozó műveletek, munkamenet és technológiák dokumentálása
- Launcherek létrehozása
- NoSQL adatbázis struktúra megvalósítása

### **Gonda Szabolcs Krisztián:**

- Android applikáció fejlesztéséhez szükséges technológiák kiválasztása
- Android applikáció létrehozása és fejlesztése
- Android applikáció tesztelése
- Android applikáció és az ehhez tartozó műveletek, munkamenet és technológiák dokumentálása
- MySQL adatbázis létrehozása
- Életszerű, futárok számára tervezett applikáció megalapozása

### **Rabóczki Erik:**

- Frontend fejlesztéséhez szükséges technológiák kiválasztása
- Frontend létrehozása és fejlesztése
- Frontend teszt létrehozása
- Frontend és az ehhez tartozó műveletek, munkamenet és technológiák dokumentálása
- NoSQL adatbázis struktúra finomítása
- Egyedi stílus elemek és komponensek létrehozása



### 3. Adatbázis

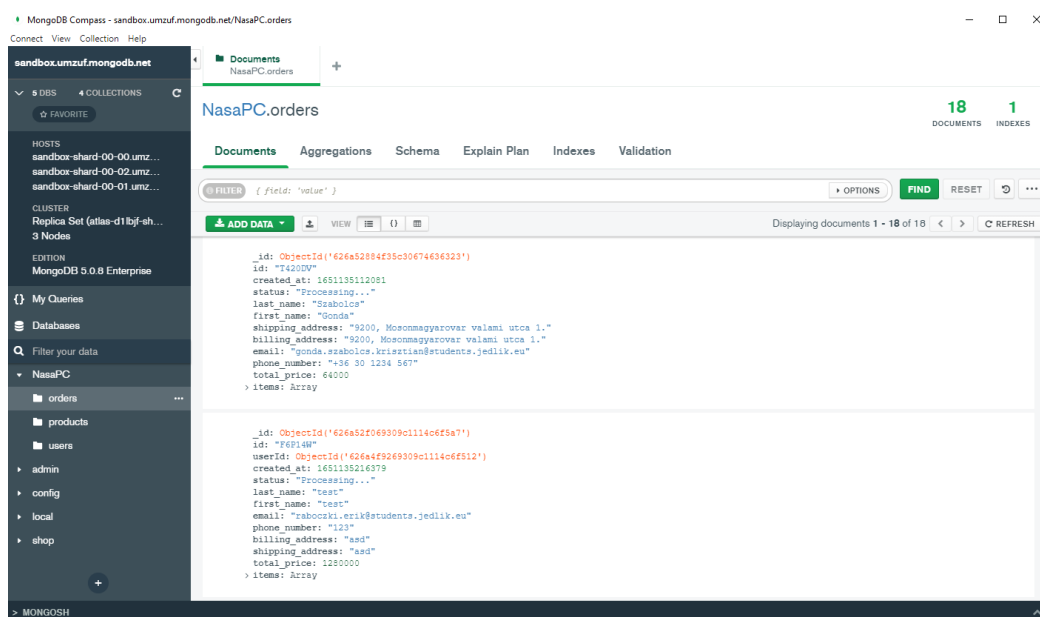
Ami az adatbázis típusát illeti, szintén két lehetőség közül választottunk: a MySQL és a NoSQL között.

Az adatstruktúrák jelentősen különböznek, de mind a két megoldás teljesen átlátható, tehát ez nem volt szempont. A NoSQL az adatokat BSON formátumban tárolja majd küldés előtt alakítja át JSON-be, ez miatt az adatforgalom gyorsabb, bár ez az előny leginkább nagyobb adathalmazoknál mutatkozik meg, a mi projektünknel ez az előny elhanyagolható.

Végössoron a könnyebb kezelhetőség és átláthatóság és az miatt, hogy talán a jövőben a NoSQL-ben szerzett rutinnak majd nagy hasznát vehetjük, eléggnek bizonyult ahhoz, hogy ezt a megvalósítási módot válasszuk.

Annak érdekében, hogy a fejlesztés során az adatbázisunkkal a lehető legátláthatóbban tudjunk dolgozni egy mongoose nevű npm csomagot telepítettünk a backenden, és a mongoose metódusaival dolgoztunk.

Szintén hasznos eszköz a mongodb compass is, ez szoftver nagyon egyszerűen egyszerű GUI felületet biztosít az adatbázisunkkal kapcsolatos egyszerűbb műveletek végrehajtásához.







Ahogy az a fenti mongoddb compass GUI felületéről készült képen is látszik, mindössze három kollekcióra volt szükség.

Külön kollekcióban tároljuk a felhasználókat és azok adatait, külön a termékeket és azok adatait és végül a rendeléseket szintén a hozzájuk tartozó adatokkal.

Alapvetően elmondhatjuk, hogy a NoSQL-ben az egyes rekordok milyensége, ez alatt értem a rekordok tulajdonságait többek között, sokkal kötetlenebb, mint a MySQL-ben. De annak érdekében az rekordoknak (például egy bizonyos termék az egy rekord) tudjunk egy kötött adatstruktúrát adni úgynevezett schemákat alkalmaztunk.

A schéma gyakorlatilag egy séma, megadhatjuk benne, hogy az egyes kollekcióban lévő adatok milyen tulajdonságokkal rendelkezzenek, továbbá az egyes tulajdonságokkal szemben is több lehetőséget biztosít. Ilyen lehetőség lehet például az, hogy a schemán belül tudjunk rögtön validálni az adatot.

```
},
discount: {
  type: Number,
  default: 0,
  required: true,
  validate: {
    validator: function (d) {
      return d >= 0 && d <= 100;
    },
    message: "Discount must be between 0 and 100 (including 0 and 100)",
  }
},
reviews: [
```

Ebben az esetben például az egyes termékek discount tulajdonságát tudjuk befolyásolni a következő módokon:

Csak számot fogad, 0 alapértelmezett értékkel, a megadása kötelező és csakis 0-100 -ig terjedhet az értéke.

De nem is feltétlenül csak integer, boolean, string stb típusú értékeket adhatunk meg, akár egy egész tömb is szolgálhat tulajdonságként.

Ilyen megoldást alkalmaztunk a rendelések(orders) és a kívánságlista(wishlist) rögzítése érdekében is.

```
total_price: Number,  
payment_method: String,      tamasrobert, last month • Models  
items: [  
  {  
    product: { type: Object, required: true },  
  }  
]  
{ versionKey: false });
```

Tehát a NoSQL használatával nincs szükség összekötő táblák alkalmazására, a kód és a táblák mennyisége is csökken a könnyebb átláthatóság, ugyanazon adatok könnyebb tárolása és a lekérdezések sebességének növelése érdekében.

Az egyes kollekciókhoz írt scemákat úgynevezett modellekben tároljuk, értelemszerűen a backend is három modellt tartalmaz hiszen három schemánk van, és az ezekkel a modellekkel való további interakciót, a modell beimportálásával rögtön lehetővé tudjuk tenni hisz a mongoose által nyújtott beépített metódusok azonnal használhatóak.

## 4. JavaScript

A JavaScript (röviden JS) egy programozási nyelv, amelyet kifejezetten az internetre fejlesztettek ki. A legtöbb webböngésző szoftver, és a modern okostelefonok is mind támogatják a JavaScriptet.

A JavaScriptet első sorban arra használják, hogy gazdagabb, felhasználóbarát élményeket teremtsenek vele az internetet böngészők számára, például dinamikusan frissülő weboldalakot, intuitív felhasználói felületeket, menüket, párbeszédpaneleket, 2D-s és 3D-s grafikákat, interaktív térképeket, videólejátszókat, és számos egyéb elemet, illetve funkciót. A JavaScript ilyesfajta, webböngészőkben történő alkalmazását kliensoldalú JavaScriptnek szokás nevezni.

A JavaScript az internetes háromszög egyik eleme, a másik kettő pedig a CSS, illetve a HTML. A HTML feladata a weboldal leírása (szövegek, grafikák stb.), a CSS pedig a weboldal megjelenéséért felel. A JavaScript nem esszenciális, de a fent említett hasznos funkciók miatt nagyon fontos eleme a webdizájnak, hiszen dinamikus weboldalakot lehet létrehozni vele.

## 4.1 A JavaScript története

A JavaScript fejlesztését 1995-ben kezdte el a Netscape Communications, az akkoriban népszerű Netscape böngésző készítője. A céljuk egy jobb felhasználói élmény megalkotása volt egy „ragasztó nyelv” segítségével. Brendan Eich-et kérték fel arra, hogy beágyazza a Scheme programnyelvet. Azonban mivel akkoriban a Java volt az új, felkapott programnyelv, úgy döntöttek, hogy az új nyelv szintaxisát inkább ahhoz hasonlóra készítik majd el – így született meg a JavaScript, amely egy nyelvben ötvözte a Scheme funkcióit, a SmallTalk objektumorientáltságát, és a Java szintaktikáját.

## 4.2 Működése

Miután egy webböngésző betölt egy weboldalt, és átvizsgálja a HTML kódját, elkészít egy úgynevezett dokumentum objektum modellt (DOM) a weboldal tartalmairól. A DOM egy élő képet mutat a JavaScript kódnak a weboldallról, és a kód módosításokat végezhet a DOM-on, mielőtt az megjelenne a felhasználónak. Mindez maximum néhány másodperc alatt történik.

A böngésző továbbá lehetővé teszi, hogy a kód felhasználói műveletekre aktiválódjon, például egérmozgásokra, gombkattintásokra stb. Mindez rengeteg lehetőséget ad egy JavaScript programozó kezébe, aki bármilyen kisebb vagy nagyobb alkalmazást készíthet különböző feladatok elvégzésére.

Mint említettük, amikor egy böngésző szoftver betölt egy webhelyet, a HTML vizsgáló kielemez a HTML kódot, és elkezdi felépíteni a dokumentum objektum modellt. Amikor a vizsgáló CSS vagy JavaScript kódot talál (legyen az dokumentumon belüli, vagy kívülről betöltött), átnyújtja azt a CSS vagy JavaScript elemző motornak. A JavaScript motor betölti a külső JavaScript fájlokat vagy az inline JS kódot, de még nem futtatja le őket azonnal – előbb megvárja, hogy a HTML és a CSS elemzés befejeződjön.



Miután az elemzés kész, a JavaScript abban a sorrendben fut le, amilyen sorrendben az elemző találkozott vele az oldalon – definiálásra kerülnek a változók és a függvények, a függvények meghívódnak, aktiválódnak az eseménykezelők, és így tovább. Ezek mind frissítik a HTML elemző által felállított DOM-ot, és végül a weboldal megjelenik a böngészőben.

### **4.3 Mire képes a JavaScript?**

A JavaScript egy komplett programnyelv-fordító, amely közvetlenül a webböngésző szoftverekben működik. A JavaScripttel mindent meg lehet oldani, amit a Java-val, vagy bármilyen hasonló, általános programozási nyelvvel, például:

- Deklarálhatunk vele változókat
- Értékeket tárolhatunk és hívhatunk meg
- Függvényeket definiálhatunk és használhatunk
- Saját osztályokat hozhatunk létre
- Külső modulokat használhatunk

A JavaScript egy olyan sokoldalú programozási nyelv, hogy még vírusokat és rosszindulatú alkalmazásokat is lehet írni benne – sajnos rengeteg hacker ki is használja ezt az óvatlan felhasználók ellen. Ez lehetővé teszi a támadóknak, hogy böngésző sütiket, jelszavakat, bankkártya-adatokat és egyéb személyes adatokat lopjanak a felhasználóktól, vagy akár konkrét vírusokat telepítsenek a számítógépekre. Éppen ezért fontos volt a weboldalunk számára, hogy egy funkcióban gazdag, de biztonságos felületet biztosítsunk a felhasználó(k)-nak.



## 5. Node.js

A Node.js (vagy másnéven Node) egy rendszer, melyben JavaScript-ben írhatunk szerver oldali alkalmazásokat, tehát a böngészőn kívül. Maga a rendszer C/C++-ban íródott, és egy esemény alapú I/O rendszert takar a Google V8 JavaScript motorja felett. Az egyik lényeges különbség például a PHP-hoz képest azon kívül, hogy JavaScript-ben írjuk az alkalmazásunkat, hogy nincsen szükségünk HTTP szerverre (mint amilyen az Apache httpd), mivel mi magunk írjuk a HTTP szerverünket.

### 5.1 Célja

A Node célja, hogy egy egyszerű felületet adjon arra, hogy skálázható hálózati alkalmazásokat írjunk. Azt tudjuk, hogy a JavaScript a világon a jelenleg egyik legnépszerűbb programozási nyelv, többek között ezért esett erre a nyelvre a választás. A JavaScript nyelvnek vannak más sajátosságai, amelyek szintén előtérbe kerülnek a rendszer használata során. Ahogy a böngészőben, úgy a Node alatt is egyetlen esemény hurokban (event loop) fut a program, és ez adja az egyik legfontosabb előnyt.

A Node.js-t úgy írták meg, hogy (szinte) minden esemény aszinkron legyen, ezért a program sosem blokkolódik, azaz nem kell várni, hogy egy művelet befejeződjön, vele párhuzamosan futtathatunk további műveleteket. Ez pontosan ugyan úgy működik, mint a böngészőben levő XHR kérések vagy más események, mint például a click, mouseover. Ez az alkalmazás folyamatosabb futását, több alkalmazás párhuzamosítását, valamint az egész rendszer optimálisabb működését teszi lehetővé.



## 5.2 Modulok

A Node.js alapértelmezetten is rengeteg modullal érkezik. A modulok a CommonJS specifikáció szerint íródtak, és ennek megfelelően a `require('modulneve');` paranccsal tudjuk behúzni őket az alkalmazásunkba. Ezek egy objektumot tesznek elérhetővé, és egy változóhoz rendelve később kényelmesen tudjuk használni is őket. Az alap telepítésben elérhető modulokról a Node hivatalos API dokumentációjában olvashatunk.

## 5.3 npm

A Node hivatalos csomagkezelője az npm, amellyel kényelmesen listázhatjuk, telepíthetjük, frissíthetjük és eltávolíthatjuk az elérhető modulokat, valamint mi magunk is publikálhatjuk az általunk fejlesztett modulokat az npm rendszerébe. Az npm azaz Node Package Manager - vagy Node Csomag Kezelő - node programok telepítésére és karbantartására használható. A rendszerben jelenleg több százezer csomag van és naponta több milliót installálnak programozók. Ma már az npm a node szerves része, úgyhogy nem kell külön installálni.

### Csomagok helyi (local) telepítése:

Egy csomagot a helyi mappába a nagyon egyszerű `npm install <csomag_neve>` paranccsal lehet installálni.

Például: `npm install lodash`.

A parancs lefuttatása után a node létrehoz (ha még nincs) egy `node_modules` mappát és ha kilistázzuk a tartalmát (`ls node_modules`) akkor láthatjuk, hogy megjelent a `lodash` könyvtár.



## Csomagok globális telepítése:

Egy csomagot kétféle képpen lehet telepíteni, az egyik a már fentebb említett helyi telepítés, a másik pedig a globális. Globálisan olyan csomagokat szokás telepíteni, amiket többnyire terminálból akarunk futtatni (pl: grunt, jshint). Maga a parancs nagyon hasonló az előzőhöz: `npm install -g jshint`.

## 5.4 Express

Az Express egy ingyenes és nyílt forráskódú webalkalmazás keretrendszer a Node.js számára. A webes alkalmazások gyors és egyszerű tervezésére és építésére használják. Az Express a Node keretrendszere, ami azt jelenti, hogy a kód nagy része már előre meg van írva a programozók számára. Az Express segítségével egyetlen oldal-, többoldalas vagy hibrid webalkalmazást is létrehozhatunk.

Az Express JavaScript könyvtára segít a programozóknak hatékony és gyors webalkalmazások létrehozásában. Az Express meglehetősen leegyszerűsíti a Node használatát, megkímélve a programozókat, azáltal, hogy magába foglal olyan komponenseket, melyek a fejlesztés során gyakran használtak.

### 5.4.1 Indoklás

A legértékesebb eszköz minden vállalkozásban az idő. A döntő többségének a programozónak meg kell küzdeniük azért, hogy rövid időn belül hatékony webalkalmazásokat hozzanak létre. De a webalkalmazások kódolása és tesztelése rendkívül időigényes. Ez az, ahol az Express.js életmentővé válik a programozók számára.

Az Express felére csökkentheti a kódolási időt, de ettől függetlenül gyors, tiszta és jó alkalmazást fejleszthetünk. Nemcsak csökkenti az időt, hanem



csökkenti a webalkalmazások létrehozásához szükséges erőfeszítéseket a különböző funkciók segítségével, amely segíthet a mentális egészség megőrzésében.

Az Express használatának másik oka a JavaScript. Az Express.js lehetővé teszi még a kezdők számára is, hogy belépjenek a webes alkalmazásfejlesztés világába, mert támogatja a JavaScriptet. A JavaScript nagyon könnyen megtanulható bárki számára, még akkor is, ha nem rendelkezik előzetes ismeretekkel más nyelvekről.

Minden vállalkozás másik fontos eszköze a pénz. Fontos, hogy a pénzt hatékonyan használjuk a nyereség maximalizálása érdekében. Mivel az Express.js egy nyílt forráskódú és ingyenes rendszer, amely számos nagyszerű funkciót kínál, nincs különösebb ok arra, hogy ne használjuk.

#### **5.4.2 Előnyök**

- Gyorsrá és egyszerűvé teszi a webalkalmazások fejlesztését.
- Könnyen konfigurálható és testre szabható.
- Különböző köztes szoftvermodulokat tartalmaz, amelyekkel kérés és válasz metódusok segítségével további feladatok hajthatók végre.
- Könnyen integrálhatóak a különböző sablonmotorok, mint például Jade, Vash, EJS, Pug.
- Lehetővé teszi a Middleware (köztes szoftver) használatát.
- Lehetővé teszi REST API-kiszolgáló létrehozását.
- Könnyen csatlakoztatható adatbázisokhoz, mint például a MongoDB, Redis, MySQL





## **5.5 Express funkciók**

### **5.5.1 Gyorsabb szerveroldali fejlesztés**

Az Express a Node számos általánosan használt funkcióját biztosítja olyan funkciók formájában, amelyek könnyen használhatók bárhol a programban. Ez megszünteti a több órás kódolás szükségességét, és így időt takaríthatunk meg. A gyorsabb fejlesztés következtében, egyszerűbb nehezebb feladatokat ellátni rövid idő alatt, ami természetesen azt is jelenti, hogy magasabb bevétel várható. Le egyszerűsítve; pénzt és időt spórolhatunk meg.

### **5.5.2 Middleware**

A Middleware (köztes szoftver) a program olyan része, amely hozzáfér az adatbázishoz, a klienskérésekhez és a többi köztes szoftverhez. Elsősorban az Express különböző funkcióinak szisztematikus szervezéséért felelős.

### **5.5.3 útválasztás**

Az Express.js egy rendkívül fejlett útválasztási mechanizmust biztosít, hasonlóképpen a Laravelhez.

### **5.5.4 Sablonmotorok**

Az Express.js olyan sablonmotorokat biztosít, amelyek lehetővé teszik a fejlesztők számára, hogy dinamikus tartalmat építsenek a weboldalakon HTML-sablonok létrehozásával a szerveroldalon. Ilyen sablon lehet például az EJS vagy a Pug.



### 5.5.5 Hibakeresés

A hibakeresés elengedhetetlen a webes alkalmazások sikeres fejlesztéséhez. Az Express.js megkönnyíti a hibakeresést azáltal, hogy hibakeresési mechanizmust biztosít, amely képes pontosan meghatározni az alkalmazásban a pontos részt, amely hibákat tartalmaz.

### 5.5.6 Összegzés

Az Express egy ingyenes és nyílt forráskódú webalkalmazás keretrendszer a Node.js számára. A webes alkalmazások gyors és egyszerű tervezésére és építésére használják. Az Express a Node keretrendszere, ami azt jelenti, hogy a kód nagy része már előre meg van írva a programozók számára. Az Express segítségével egyetlen oldal-, többoldalas vagy hibrid webalkalmazást is létrehozhatunk.

Az Express egy elterjedt, és sikeres rendszert mert:

- Könnyű megtanulni az alapokat. Sok frontend fejlesztő már ismeri a JavaScriptet, tehát nem kell új nyelvet tanulniuk ahhoz, hogy megtanulják az Express-t.
- Sokkal könnyebbé teszi a backend fejlesztést, sok időt és pénzt spórolhatunk meg.
- Sablonmotorok segítségével frontendet is könnyen létrehozhatunk.
- Az Express egyszerűen testre szabható, hogy megfeleljen a fejlesztési igényeknek.
- Az Express biztosít köztes szoftverrendszert, így könnyebben szétválasztható a kód, könnyebben olvasható, jobban érhető lesz. Így a fejlesztés is gyorsabban, hatékonyabban haladhat.



A frontend terén is több lehetőséget vettünk fontolóra:

A Typescript alapú Angular és a Javascript alapú Vue.js lehetőségét mérlegeltük. Egyedül natív kódolással hasonló okok miatt most sem szerettünk volna dolgozni, mindenképpen most is az imént említett lehetőségekből szerettünk volna választani és natív Javascript kódot csak akkor szerettünk volna alkalmazni, amikor olyan dolgot szerettünk volna megvalósítani, amelyet a framework nem tartalmaz.

Az Angular bár napjainkban egy elég elterjedt megoldás, a környezetünkben is elismert cégek programoznak benne, mi személyes preferencia alapján a Vue.js mellett döntöttünk.

Szeretnénk megjegyezni, hogy bár most nem az Angulárra esett a választásunk, de mindenképpen megéri most a közös projekt mellett és a későbbiekben is, nagy hangsúlyt fektetni rá, hiszen a jelenlegi ismereteink alapján elmondhatjuk, hogy sok lehetőség van benne. A TypeScript bár egyesek mondhatják, hogy nehezebb nyelv, mint a JavaScript, mindenképpen megéri a bele fektetett időt, melyet később majd a személyes gyakorló projektek által igenis bele fogunk fektetni.

A frontedet Vue.js-ben írjuk!

A Vue.js egy nagyon könnyen megtanulható, és használható JavaScript könyvtár. Alacsony Learning Curve-vel rendelkezik.

A Vue.js-t annak ellenére, hogy könnyebben elsajátítható, rengeteg feladatra könnyedén fel lehet használni. Kifejezetten szeretik startupok is használni, valamint kiváló párosítást alkot Laravel keretrendszerrel.

Igaz most ezt az előnyt nem fogjuk kihasználni.

Szintén pozitívumnak könyveltük el, hogy a Vue.js-ben minden eddiginél könnyebben tudtunk teszt adatokat felvenni olyan struktúrában, amilyenben majd már élesben fogja kapni az adatokat és így a kezdetektől fogva koncentrálni tudtunk egy dinamikus frontend létrehozására.

## 6. Frontend

Mielőtt a frontend fejlesztésének neki láttunk volna, megkellett határoznunk, hogy milyen legyen a dizájn, és hogy milyen komponenseket szeretnénk belerakni a projektbe.

A dizájn és a komponensek meghatározásának az érdekében, egy meeting keretein belül a döntéseinket egy dizájn demo elkészítésével örökítettük meg.

Ez nem a végleges verzió, sokkal inkább egy iránymutató, arról hogy a dizájn terén milyen irányba menjen el a frontend. Ami a dizájn milyenségét illeti több oldalról is inspirációt merítettünk: többek között a tchibo oldaláról.

Már a projekt létrehozásakor tudtuk, hogy a bootstrap külső komponenseket, stílus elemeket és a grid rendszert szeretnénk majd használni ezért ez már a projekt elején telepítésre került. További a Primevue által felkínált listákat és formokat is fontolóra vettük.

### 6.1 Navigation komponens

Bár az imént említett források tartalmaznak előre elkészített navigation panelt, úgy döntöttünk, hogy nem vagyunk teljes mértékben megelégedve a felkínált lehetőségekkel ezért itt, egy külső komponens használata helyett, saját komponens megírása mellett döntöttünk.

A komponenssel kapcsolatban több alap kikötésünk is volt:

Az első kikötés a reszponzivitás volt, jól kell mutatnia mind mobilon mind nagyobb felbontású képernyőkön. Ezt a kritériát és a tényt, hogy mobilon nézetben nehezebb több információt megjeleníteni anélkül, hogy a felhasználót elárasztanánk üzenetekkel, a weboldal átláthatóságát csökkentenénk és a felhasználói élményt rontanánk, ez a feladat olyan megoldást igényelt, amely lehetővé teszi, hogy a felhasználó kontrollálni tudja a megjelenített információ mennyiségét.

Az imént említett gondolatokat figyelembe véve, felismertük, hogy a felhasználónak nemszükséges minden pillanatban látnia a menüt mobil nézetben, ezért annak és a navigációs menükhöz kapcsolódó paneleknek a megjelenítését dinamikussá, láthatóságát egy fix pozíciójú gomb által testre szabhatóvá tettük.

Továbbá megfigyelve több modernebb kialakítású weboldalt, azt tapasztaltuk, hogy az a dizájn melyet a mobileszközökön való adatmegjelenítésnél használnak teljes képernyős nézetben is egyre nagyobb teret hódít, ez a megközelítés tetszett nekünk így hasonló dizájn fejlesztésébe kezdtünk.



Ami a menüpontokat illeti az eredeti elképzelés szerint a weboldal dizájn teljes mértékben dinamikus lett volna és minden adatot adatbázisból kértünk volna le. De végül ebből az elképzelésből engedtünk az adatforgalom csökkentése érdekében.

## 6.2 Footer komponens

A footer manapság egy standard komponensnek számít minden weboldalon, a megléte gyakorlatilag már-már elvárt. Ami a funkcióját illeti elsősorban két célunk volt a létrehozásával. Először is bizonyos információk megjelenítése, többek között a felhasznált technológiák logóinak feltüntetése, copyright jogok feltüntetése és a weboldal logója.

Másodszor pedig némely navigációs link ismételt megjelenítése. Erre azért van szükség, mert a látogatónak a figyelmét nagyobb valószínűséggel tudjuk megragadni, hogyha egyes menüpontok a weboldal tetején és a footer segítségével a weboldal alján is megjelennek, hiszen így a további szolgáltatásainkra például a fő oldal áttekintése után ismét felhívjuk a figyelmet.

## 6.3 Home page

A fejlesztés során több elképzelés is született a fő oldal megvalósításával kapcsolatban. Az alapelv az volt, hogy a fő oldalnak figyelemfelkeltőnek kell lennie, de nem is szabad túl sok információval elárasztania a felhasználót. Az imént említett navigációs panel kialakításának módját figyelembe véve és azt, hogy alapértelmezetten a navigation komponens nem jelenik meg, így, több adatot tudunk megjeleníteni úgy, hogy az még ne legyen zavaró.

A fő oldal elsődleges célja a látogató figyelmének a felkeltése és a további böngészésre való ösztönzés, ezért a fontosabb szolgáltatásaink linkjei a főoldalon akár háromszor is megjelenhetnek különböző formában.

További szempont volt a megfelelő vizuális megjelenítés is. Tehát az információ ízléses tálalása. Az egyik ilyen módszer volt a carousel komponens felhasználása, mely periodikusan egy képlistán megy végig magyarázó szöveg kíséretében. Ez a fajta megjelenítés napjainkban nagyon elterjedt és jól mutat mind típusú kijelzőn, hiszen a komponens reszponzivitás könnyen megoldható. A weboldalunkon a bootstrap carousel külső komponenst használtuk fel.

A forráskód átírásával a komponens dinamikusan a megadott adatok alapján generálódik le.

## 6.4 About us

Az about us oldal egy egyszerű oldal ahol magunkról, a projekt céljáról és olyan dolgokról melyek, a látogatónak szükségesek lehetnek, osztunk meg alap információkat. Talán ez a legegyszerűbb oldal a projektben ugyanis,

egyszerű kiíratásról van itt szó listákba a pragrafusokba rendezve.

## 6.5 Product (user)

Az oldal elsődleges célja hogy az elérhető termékek közül a látogató tudjon böngészni és egyéb műveleteket végre hajtani. A termékek úgynevezett cardok segítségével vannak feltüntetve. Mind a bootstrapnek és a primevue-nak vannak card moduljai, de ezek nemteljesen feleltek meg az elvárásainknak.

A cardokkal kapcsolatban a következő elvárásaink voltak:

1. A cardnak tartalmazni kell egy lekicsinyített képet a termékről.
2. A termék nevét és árát felkell tudnunk rajta tüntetni.
3. Ha a felhasználó vagy egy gombra kattintva vagy mouseover esemény következtében a cardnak a méretét meg kell növelnünk annyira, hogy az utóbbi információk mellett még egy részletesebb leírást is megtudjunk jeleníteni.

```
</div>
<div class="col-6" style="text-align: right">
  <DataViewLayoutOptions v-model="layout" />
</div>
</div>
</div>
```

A termékek megjelenítésére több megjelenítési mód is rendelkezésre áll.

A products oldalon a felhasználó több műveletet és képes végrehajtani, névlegesen a terméket kosárba tudja helyezni, a választott termékről bővebb információhoz juthat, ehez a művelethez a product-details oldalra való átlépés szükséges, és a lehetőség nyílik a termék kívánságlistára, való felvételére is.

Ezekhez a műveletekhez backend api-hívásokra lesz szükség.

A termék kosárba való felvételéhez, a felhasználónak nemszükséges regisztrálnia és bejelentkeznie, hiszen a kosár tartalmát nem adatbázisban tároljuk, azonban a kívánságlistára való felvételhez alap követelmény hogy



a felhasználó be legyen jelentkezve, az api hívás csak így lehetséges, a hívás előtt egy session api hívásával, illetve a visszakapott boolean változó értékével validáció történik. Igaz, már az oldal renderelése előtt lefut egy hasonló validáció, hogy olyan gombokat, menüket és információkat ne osszunk meg melyekre a felhasználónak nincs jogosultsága. Ez azért fontos, mert így el tudjuk kerülni, azt hogy összezavarjuk a felhasználót.

```
addToCart(_id) {
  var cartItem = {_id, quantity: 1};
  var match = false;
  if(!JSON.parse(localStorage.getItem('cart'))) {
    localStorage.setItem('cart', JSON.stringify([cartItem]));
  } else {
    var locArr = JSON.parse(localStorage.getItem('cart'));
    for (let i = 0; i < locArr.length; i++) {
      if(locArr[i]._id == cartItem._id) {
        match = true;
        locArr[i] = ({_id: cartItem._id, quantity: (locArr[i].quantity+1)});
      }
    }
    if(!match) locArr.push({_id: cartItem._id, quantity: 1});
    localStorage.setItem('cart', JSON.stringify(locArr));
  }
},
addToWishList(_id) {
  AccountDataService.addToWishList(_id)
    .then(()=>{})
    .catch(err => {console.log(err.response.data.error)});
},
```

Hogy nem regisztrált felhasználók is használni tudják a weboldalt, a kosár lokálisan kerül eltárolásra.

## 6.6 Signup és Login

A signup oldal teszi lehetővé, hogy a felhasználó regisztrálni tudja magát az oldalra, és hogy aztán élvezhesse annak előnyeit.



A regisztrációs folyamathoz egy form kitöltése szükséges, melyen több hiba lehetőséget is figyelembe kellett vennünk.

1. A megadott adatokat validálni kell, hogy véletlenül se lehessen hibás adatokat megadni.
2. A felhasználónak további segítséget kell nyújtani az által hogy:
  - a. Segítünk megfelelő erősségű jelszót választani a Primevue: password komponens segítségével.
  - b. Redukáljuk az elírás lehetőségét azáltal, hogy validációt készítünk a jelszó kétszeres lekérése után.
  - c. A form kitöltése után validáljuk a megadott adatokat.

Továbbá az kitöltött mezőket adatkötésekkel tároljuk.

```
    },  
    Validation(){  
        if (this.name != null) {  
            this.vname = true  
        }  
        if (this.email != null) {  
            this.vemail = true  
        }  
        if (this.password != null) {  
            this.vpassword = true  
        }  
        if (this.data != null) {  
            this.vdate = true  
        }  
        if (this.accept != null) {  
            this.vaccept = true  
        }  
    },  
    closeDialog() {  
        this.showMessage = false  
    }  
}
```

A felhasználó nem hagyhat egy mezőt sem üresen



```
Signup() {  
    this.Validation()  
    if (this.vname == true &&  
        this.vemail == true &&  
        this.vpassword == true &&  
        this.vdate == true &&  
        this.vaccept == true ) {  
        AccountDataService.SignUp({"email": this.email,
```

A signup api csakis sikeres validáció után kerül meghívásra.

A validációk sikeressége után, egy api hívás segítségével folytatódik a regisztrációs folyamat. Egy felugró ablak jelzi, hogy a link mellyel az Account aktiválható a megadott emailre lesz elküldve.

Az emailben elküldött link által való fiók aktiválását követően, a fiók adatbázisban való tárolása megtörténik. Ez úgy valósul meg, hogy az emailben elküldött link, betöltésekor a frontend az URL-ből kinyeri az aktiváláshoz szükséges token, melyet felhasználva egy API hívást követően az adatbázisban az account aktiválásra kerül.

Ha ez sikerült egy új felugró ablak tájékoztatja a felhasználót a művelet sikerességéről vagy az esetleges hibákról, majd átirányítja a főoldalra.

A login oldalon hasonló képpen, a validációs műveletek után egy login api hívást követően a frontend átnavigálja a felhasználót a főoldalra, ami egy ismételt session api hívással megállapítja, hogy a kliensoldalon való session valid, így a bejelentkezett userek számára elérhető tartalmak renderelésre kerülnek.

Ilyen tartalmak többek között, a navbar tartalma, és hozzáférés egyes pagek-hez.

```
<Dialog v-model:visible="this.showMessage" :breakpoints="{ '960px': '80vw' }" :style="{ width: '30vw' }" position="top">
  <div class="flex align-items-center flex-column pt-6 px-3">
    <i class="pi pi-sign-in" :style="{fontSize: '5rem', color: messageColor }"></i>
    <h5 id="DialogHeader">{{this.messageHeader}}</h5>
    <p :style="{lineHeight: 1.5, textIndent: '1rem'}">
      {{this.messageText}}
    </p>
  </div>
  <template #footer>
    <div class="flex justify-content-center">
      <Button label="Cancel" @click="closeDialog()" class="p-button-text" />
    </div>
  </template>
</Dialog>
```

```
errorDialog(message) {
  this.showMessage = true
  this.messageHeader = "ERROR"
  this.messageText = message
  this.messageColor = "red"
},
successDialog() {
  this.showMessage = true
  this.messageHeader = "Success"
  this.messageText = "Success, only one final step is required for your registration. Please check your email for activation instructions."
  this.messageColor = "green"
}
```

Mind a signup mind a login és egyéb oldalakon is úgynevezett Dialog felugró ablakkal oldottuk meg a felhasználó tájékoztatását a műveletek állásáról.

## 6.7 Wishlist

A wishlist a weboldal azon eszközei közé tartozik, melyeknek előnyeit csak regisztrált felhasználók élvezhetik. Ugyanis ez egy adatbázisban tárolt lista készítésének a lehetőségével ruházza fel a felhasználót. Így az ezen a listán feltüntetett termékek árában történő változásokat, potenciális leértékeléseket sokkal könnyebben nyomon lehet követni.

A lista szabadon szerkeszthető, a tulajdonosa szabad hozzáférést kap minden ezzel kapcsolatos tevékenységre.

## 6.8 Orders és a Cart

A felhasználó a vásárlás során az „add to cart” gombra kattintva tudja a kiválasztott termékeket a kosárba helyezni. Az hogy egy termék a kosárban van, önmagában nemjelent semmit, a rendelés csak akkor lesz véglegesítve, ha a felhasználó tovább folytatja a vásárlást a megfelelő gombra kattintva. Ha ezt megteszi, akkor a megfelelő adatok rögzítése után (regisztrált felhasználóknak ez a lépés kihagyható, elmentett fizetési adatok esetén) a cart azaz kosár tartalma átkerül az orders azaz rendelések listába. A megrendelések listában lévő termékek már elvannak különítve a vásárlónak és a termékek már nincsenek számon tartva, mint elérhető termékek, úgyis mondhatjuk, hogy ami a többi látogatót illeti, számukra a rendelések listában lévő termék már eladott így raktáron nem elérhető termékként van kezelve.

Az orders listából a megrendelt terméket el lehet távolítani, így azok ismét nem eladott, eladásra szánt terméknek tekinthetők. A sikeres kiszállítás és kifizetés után, az orders lista tartalma törlődik.

```
validationFunction()  
{  
  if (this.validation.vlast_name != null) {  
    this.validation.vlast_name = true  
  }  
  if (this.validation.vfirst_name != null) {  
    this.validation.vfirst_name = true  
  }  
  if (this.validation.vemail != null) {  
    this.validation.vemail = true  
  }  
  if (this.validation.vphone_number != null) {  
    this.validation.vphone_number = true  
  }  
  if (this.validation.vbilling_address != null) {  
    this.validation.vbilling_address = true  
  }  
  if (this.validation.vshipping_address != null) {  
    this.validation.vshipping_address = true  
  }  
  
  this.data.items = this.cartItems  
},
```

Az adatok a rendelés leadása előtt egy validációs folyamaton mennek keresztül.

## 6.9 Admin products

Az admin products célja különbözik a mindenki számára publikus products oldaltól. Ehhez az oldalhoz a hozzáférés csakis validált adminok számára lehetséges.

Az ezen az oldalon használt szerkesztett és teljes mértékben testre szabott primevue modul segítségével az admin hatékonyon tud API hívásokkal hozzáadni, szerkeszteni és törölni termékeket, tehát úgy is mondhatjuk,



hogy egy kész CRUD product interface áll a rendelkezésükre a könnyű és hatékony adatbázis kezelés érdekében, mindezt egy oldalon.

A CRUD műveletekkel való munkának minden változatát további felugró panelek könnyítik meg, lehetővé téve programozási előismerettel nem rendelkező kollégák számára is az admin munkakör gyors, kielégítő szintű ellátását.

A felugró ablakokkal, üzenetekkel és interfacekkel az admin folyamatos visszajelzést kap a munkája állapotáról illetve annak sikerességéről. Illetve a kritikus műveletek, mint például a törlés előtt felugró megerősítő ablakok minimalizálják a véletlenül elkövetett törlések és változtatások előfordulását.

Fontos megemlíteni, hogy az admin products interface használata elsősorban számítógépről való munkavégzéshez lett tervezve. Természetesen ez nem azt jelenti, hogy itt a reszponzivitást mint szempontot mellőztük, de egy ekkora táblázatot ésszerűtlen lenne, mobilnézetben megjeleníteni úgy, hogy mellőznénk az X irányú csúszka használatát. Természetesen a csúszkától eltekintve, mint minden más az oldalon ez az oldal is a reszponzivitás alapelvei szerint lett tervezve.

Ez az oldal, mint minden más admin oldal nem elérhető csak adminok számára. Annak érdekében, hogy ez a kritérium megvalósulhasson. Az oldal betöltésekor minden alkalommal a háttérben lefut egy autentikációs algoritmus, ami két dolgot ellenőriz: elsősorban hogy a felhasználó be van e jelentkezve, másodszor pedig hogy a felhasználó rendelkezik e admin



jogosultsággal. Ha ezek közül a kritériumok közül bármelyik nem teljesül, akkor az oldal egy kényszerített átirányítással visszalép a főoldalra.

## **6.10 Password change**

A jelszó változtatás egy két lépcsős művelet, elsőnek például a login oldalon a jelszó változtatást kérvényezni kell. Ez a megfelelő gombra való kattintással lehetséges, ez után egy felugró ablakban a felhasználónak meg kell adni az emailcímét. Fontos hogy azt az email címet adja, meg amivel már előzőleg regisztrált az oldalra.

Hibás adat esetén vagy bármely egyéb probléma esetén az oldal ennek megfelelően informálja a felhasználót. Ha egy érvényes email cím megadása megtörtént, akkor a backend, a felhasználót emailben fogja értesíteni a művelet sikerességéről és egy token és link emailben történő elküldésével a jelszó változtatás következő lépése kezdetét veheti.

A linkre kattintva, a backend ellenőrzi a linkben lévő tokent és a token hitelességének megfelelően, lehetőség nyílik egy új jelszó megadásához. A megfelelő gomb lenyomásával az API amely a jelszóváltoztatást elvégzi meghívásra kerül, és miután a változtatás megtörtént, az oldal átnavigálja a user-t a login oldalra.

## 6.11 Selenium teszt

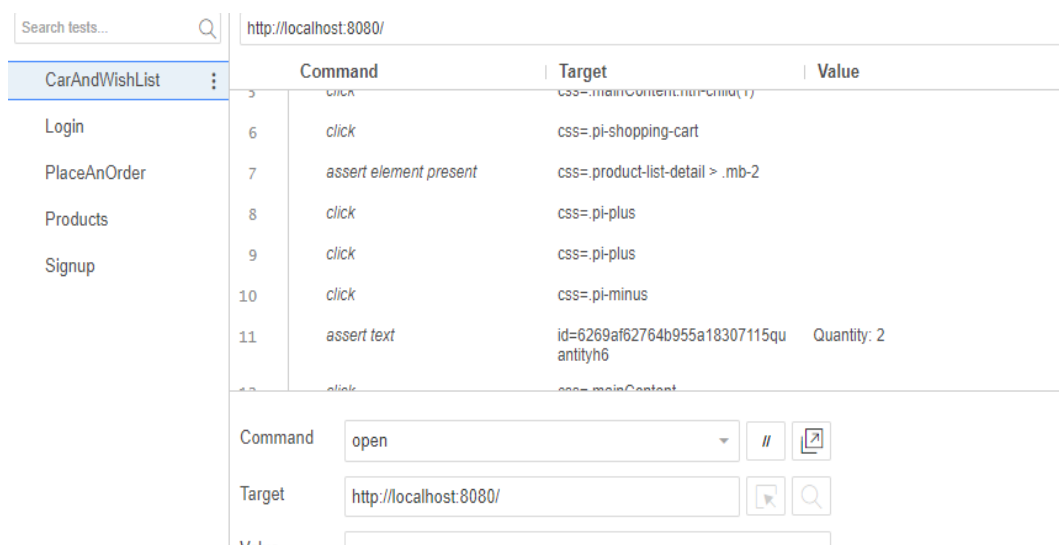
A frontend teszteléséhez egy Selenium IDE nevű megoldást használtunk.

A Selenium IDE egy rendkívül praktikus, elsősorban weboldalakon automatizált tesztek készítésére használt böngésző kiegészítő.

Természetesen a Selenium IDE természetéből adódóan, a frontend teszteléséhez csakis ennek az extension-nek a telepítésére, majd a teszt, az extension segítségével való beimportálására van szükség.

Amint az imént említett lépések megvalósításra kerültek, csakis egy gombnyomás választja el a tesztelőt a tesztek azonnali lefuttatásához.

Természetesen, ha a weboldalt a localhost:8080 -no futtatjuk lokálisan akkor szükség lesz a frontend függőségeinek telepítésére, illetve a frontend elindítására. Ezekről a műveletekről részletesebb információt talál a projekt README elnevezésű fájljaiban és egy kattintással megvalósítható a launcher mappában levő batch fájl futtatásával.



Search tests...	http://localhost:8080/
CarAndWishList	
Login	
PlaceAnOrder	
Products	
Signup	

Command	Target	Value
click	css=mainContent:mainContent	
click	css=pi-shopping-cart	
assert element present	css=product-list-detail > .mb-2	
click	css=pi-plus	
click	css=pi-plus	
click	css=pi-minus	
assert text	id=6269af62764b955a18307115quantity6	Quantity: 2

Command: open

Target: http://localhost:8080/

Value:



De hogyan is működik a Selenium IDE. A fent látható képen láthatjuk az extension GUI felületét. Az elkészített tesztek egy sor utasítást fognak végrehajtani, egy meghatározott sorrendben. A folyamatot figyelemmel kísérhetjük a teszt futtatásakor meghívásra kerülő böngészőben, továbbá a Selenium GUI felületén is láthatjuk mely utasítások elvégzése zajlott le sikeresen. Természetesen több, vagy akár az összes teszt egymás után való lefuttatására is van lehetőség.

Ami a projekt frontend tesztelését illeti, néhány dolgot fontos figyelembe venni, amikor a teszteket lefuttatjuk.

Egyes tesztek, mint például a password reset teszt és a signup test lefuttatása külön kell, hogy megtörténjen hiszen emailben történő hitelesítés is szükséges a műveletek befejezéséhez, amit nem lehetséges ilyen eszközökkel tesztelni. Itt a tesztelő aktív beavatkozására van szükség az emailben elküldött link által történő validáció elvégzéséhez. Továbbá a teszteket, a fentről lefelé történő sorrendben kell lefuttatni, a hiteles eredmények elérésének érdekében.

A Selenium IDE több böngészőre is telepíthető, mi a frontend tesztelését Google Chrome segítségével végeztük el.



## 6.12 Függőségek

Fontos tudni, hogy a projekt több külső komponenst is használ annak érdekében, hogy optimálisan működni tudjon. Minden ilyen komponens tartozzon az a backendhez vagy a frontendhez megtalálható a package.json nevű fájlban.

```
},  
"dependencies": {  
  "@popperjs/core": "^2.11.2",  
  "@vuelidate/core": "^2.0.0-alpha.39",  
  "@vuelidate/validators": "^2.0.0-alpha.28",  
  "axios": "^0.26.1",  
  "bootstrap": "^5.1.3",  
  "core-js": "^3.6.5",  
  "primeflex": "^3.1.3",  
  "primeicons": "^5.0.0",  
  "primevue": "^3.12.2",  
  "quill": "^1.3.7",  
  "vue": "^3.0.0",  
  "vue-advanced-cropper": "^2.7.0",  
  "vue-router": "^4.0.12"  
},  
"devDependencies": {  
  "@vue/cli-plugin-babel": "~4.5.0",  
  "@vue/cli-plugin-eslint": "~4.5.0",  
  "@vue/cli-service": "~4.5.0",  
  "@vue/compiler-sfc": "^3.0.0",  
  "babel-eslint": "^10.1.0",  
  "eslint": "^6.7.2",  
  "eslint-plugin-vue": "^7.0.0",  
  "sass": "^1.49.8",  
  "sass-loader": "^10.1.1"  
}
```

Itt kétfajta függőségekről találhatunk listát. Az láthatjuk a dependencies listát, amiben azok a modulok vannak melyek szükségesek a végleges weboldal működéséhez, és láthatjuk a devDependencies listát is melyek csak a fejlesztés során szükségesek.



Ezen függőségek a megfelelő mappába navigálva egy commandline interface-en belül történő „npm i” parancs kiadásával egyszerre telepíthetők egy node\_modules nevű mappába. A parancs kiadása és a sikeres telepítés után már nincs teendőnk.

A parancs csakis a node.js npm csomagkezelőjével adható ki, ezért a Node.js megfelelő telepítése szükséges. A Node.js meglété feltételezve a függőségek telepítése és a projekt elindítása a launchers mappában levő batch fájl futtatásával is egy kattintással elvégezhető.

A legfontosabb külső komponensek közé tartozik a vue illetve az ahhoz tartozó komponensek (vue-router, vue-advanced-cropper) melyek a vue-advanced-cropper kivételével a frontend keretrendszerét képezik. Továbbá a sass és sass-loader az scss kiterjesztésű fájlok felhasználásához, a bootstrap és primevue, továbbá az azokhoz tartozó egyéb csomagok melyekből több stíluselemet, a rácsrendszert (grid rendszert) és modult is felhasználunk és végül, amelyet még külön megszeretnék említeni az az Axios, amely segítségével a Restful API hívások intézése válik lehetségessé.



## 7. Backend

### 7.1 Bevezetés

A backend a webshophoz Express.js keretrendszer segítségével került megvalósításra; elsősorban a már meglévő tudás miatt került választásra.

A backend egy teljes értékű REST API-t valósít meg, amit a frontend kényelmesen tud hívni. Fontos cél volt, hogy a kód rendezett, átlátható legyen, hiszen, hogyha bármilyen fejlesztésre van szükség, egyszerűen lehessen bővíteni vagy átírni a kódot, még akkor is, hogyha a fejlesztő még nem látta előtte a projektet. Adatbázishoz NoSQL-t használtunk, amit MongoDB-vel kezeltünk. A NoSQL nem sémaérzékeny, mint az SQL, ettől függetlenül célszerű sémákat készíteni; célszerű felkészíteni a programot megfelelő adatok fogadására, hiszen nem lenne ésszerű, hogyha értelmetlen adatok kerülnének felvitelre.

### 7.2 Belépő fájl

A program indításakor az `app.js` nevű, a backend főkönyvtárában megtalálható fájl kerül meghívásra. Ez a fájl felelős a program előkészítésért, itt valósulnak meg a modellek, controllerek és modulok importálása is.

### 7.3 Termékek lekérdezése

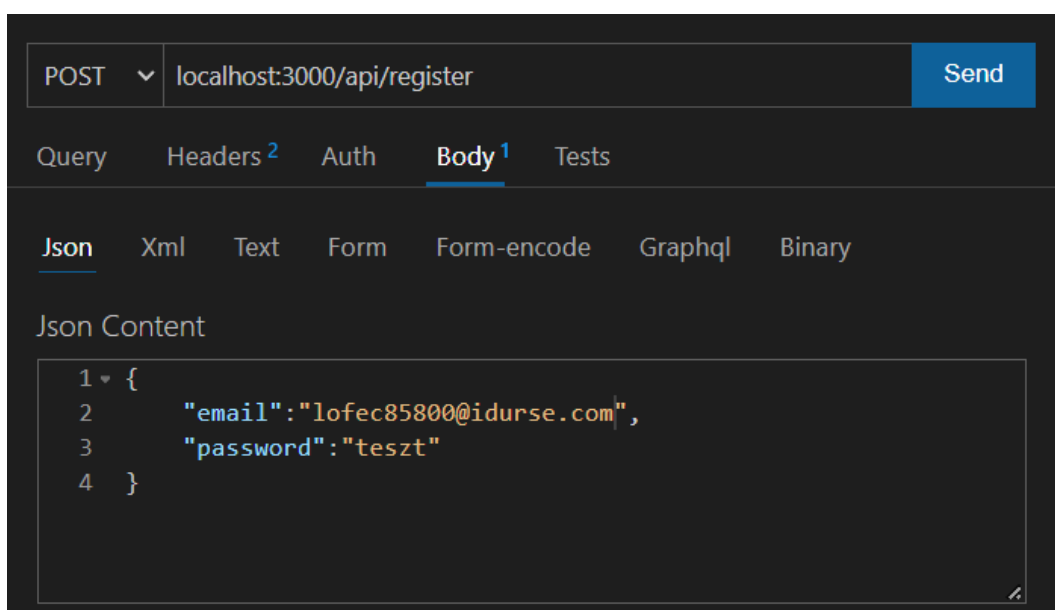
Termékek megtekintéséhez nincs szükség hitelesítésre, hiszen vendégként is lehet vásárolni. Egy specifikus termék adatai is lekérhetők, amelyhez szintén nem szükséges, hogy a felhasználó be legyen lépve.

## 7.4 Regisztráció (Register & Verify Account)

Habár a felhasználók bejelentkezés nélkül is tudnak vásárolni, kényelmi szempontból érdemes regisztrálni. A felhasználó eltárolhatja adatait, így azokat később nem kell megadnia újra és újra. Regisztráláskor kötelezően csak egy valódi email cím és egy jelszó az elvárás, így azok nem megléte hibaüzenetet von maga után.

Regisztráció igényt követően a várható felhasználónak az adatbázisban létrejön a profilja, és egy activatorToken mező is. Ennek feladata, hogy amíg ez létezik, addig a felhasználó nincsen aktiválva.

A tokent a regisztráló email-ben kapja, amely átirányítja egy útvonalra, aminek a feladata, hogy végrehajtsa az aktiválást. Az adatbázisból kikerül az activatorToken, a felhasználó mostmár aktív. Természetesen hibák esetén a megfelelő üzenet kerül visszaküldésre. Amennyiben egy felhasználó regisztrált, de nem aktiválta azt, és megpróbál újra regisztrálni, akkor is a megfelelő hibaüzenet jelenik meg.





FELADÓ	TÁRGY	MEGTEKINTÉS
tamas.robert1@students.jedlik.eu	NasaPC - Account activation	>

## Account activation

Click on this link to activate your account: <http://localhost:8080/activate/KKX4erlejVKAPymFyWMo5WSpDwwu28P7>

GET  Send

Status: 200 OK   Size: 32 Bytes   Time: 113 ms

Response   Headers <sup>9</sup>   Cookies   Results   Docs <sup>New</sup>

```
1 {
2   "message": "Account activated!"
3 }
```

Status: 400 Bad Request   Size: 36 Bytes   Time: 52 ms

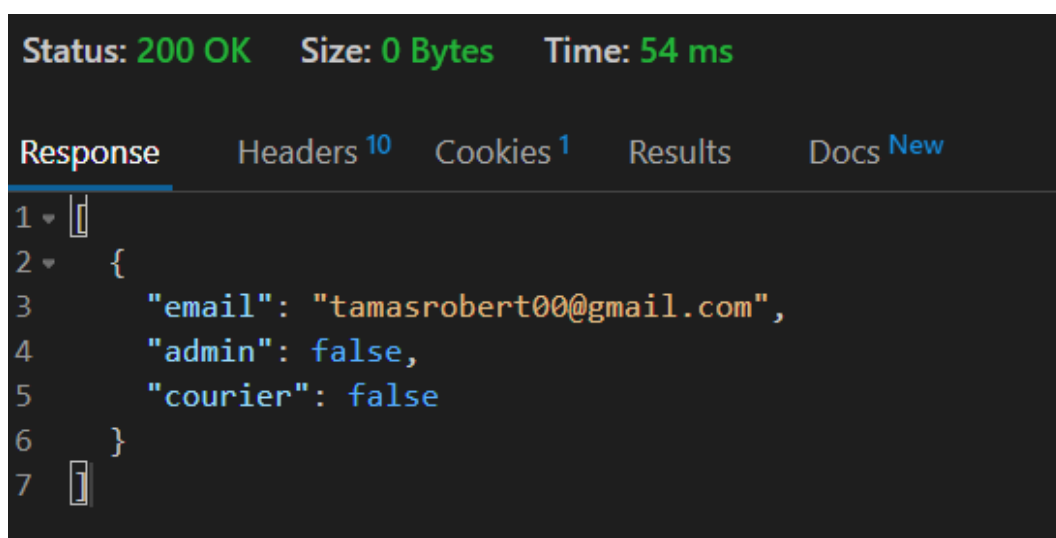
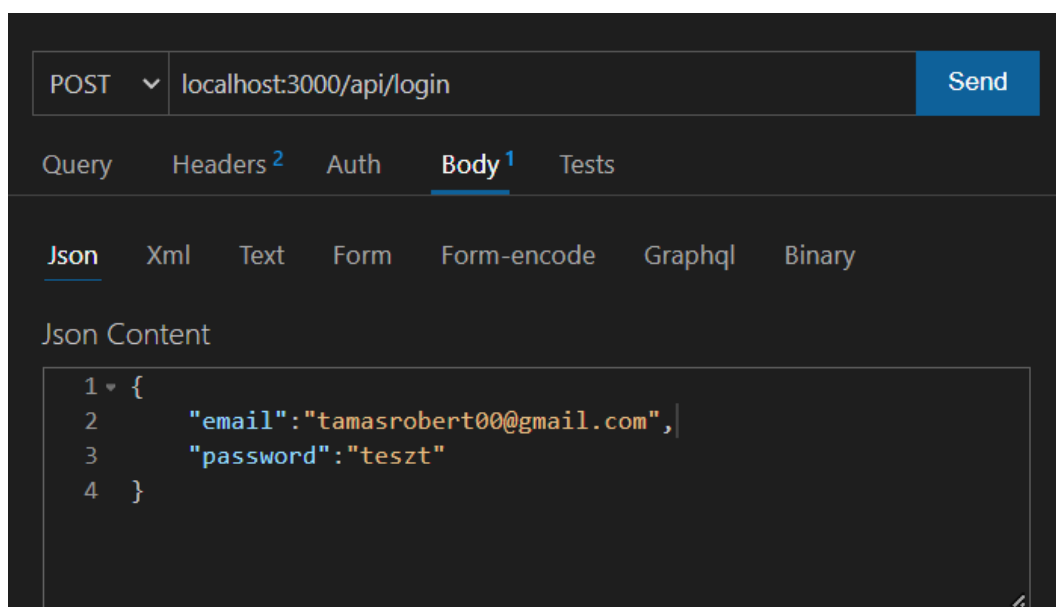
Response   Headers <sup>9</sup>   Cookies   Results   Docs <sup>New</sup>

```
1 {
2   "error": "Email is already in use!"
3 }
```

## 7.5 Bejelentkezés

Belépésnél megfelelő email és jelszó bevitele után a felhasználó kap egy LOCAL\_KEY nevű cookie-t, amelynek a hitelesítésben van szerepe. Ennek hiányában nem képes a weboldal funkcióit használni.

```
const session = makeid(32);  
res.cookie('LOCAL_KEY', session)  
User.updateOne({ email }, { $set: { session } })
```



## 7.6 Új jelszó igénylése

Lehetőség van új jelszó kérésére is a felhasználónak, hiszen nem biztos, hogy emlékszik a régre, vagy csak újat szeretne létrehozni fiókjának biztonsága érdekében. Elvárt mező csakis az email, amelyben kap egy linket, amely segítségével tudja frissíteni a jelszavát.

POST localhost:3000/api/request-password-change Send

Json Content

```
1 {
2   "email": "tamasrobert00@gmail.com"
3 }
```

Status: 200 OK Size: 43 Bytes Time: 55 ms

Response Headers 9 Cookies Results Docs New

```
1 {
2   "message": "Password change request sent!"
3 }
```

NasaPC - Requested password change Beérkező levelek x

tamas.robert1@students.jedlik.eu a(z) sendgrid.net domainen keresztül  
címzett: én

New password

Click this link to change your password: <http://localhost:8080/change-password/RjkgownEwkrDW3Z3moIzILTJnDQIxvmz>

POST localhost:3000/api/change-password Send

```
1 {
2   "token": "RjkgownEwkrDW3Z3moIzILTJnDQIxvmz",
3   "newPassword": "newpass"
4 }
```

Status: 200 OK Size: 44 Bytes Time: 103 ms

Response Headers 9 Cookies Results Docs New

```
1 {
2   "message": "Password changed successfully!"
3 }
```

## 7.7 Kívánság lista (WishList)

A felhasználóknak lehetőségük van saját kívánságlista létrehozására. Ebben a listában termékek helyezhetők, de kikötés az, hogy egy termék többször nem szerepelhet. Természetesen ez a probléma a program által kezelve van. A kívánság lista le is kérdezhető, hasonlóan a termékekhez, de itt már elvárt a hitelesítés.

POST

localhost:3000/api/add-to-wishlist/62615b4807b9781aa86fb5ee

Send

Status: 200 OK   Size: 54 Bytes   Time: 748 ms

Response

Headers<sup>9</sup>

Cookies

Results

Docs <sup>New</sup>

1

{

2

"message": "Product has been added to your wishlist."

3

}

Status: 400 Bad Request   Size: 82 Bytes   Time: 100 ms

Response

Headers<sup>9</sup>

Cookies

Results

Docs <sup>New</sup>

1

{

2

"error": "Failed adding product to your wishlist! Product is already wishlisted!"

3

}

GET

localhost:3000/api/wishlist

Send

Status: 200 OK   Size: 374 Bytes   Time: 51 ms

Response

Headers<sup>9</sup>

Cookies

Results

Docs <sup>New</sup>

{}

≡

1

{

2

{

3

"quantity": 1,

4

"discount": 0,

5

"\_id": "62615a5407b9781aa86fb5eb",

6

"name": "ASUS ROG STRIX B450-F GAMING II",

7

"description": "AMD AM4 B450 Gaming ATX motherboard with DDR4 4400 MHz support, AI Noise-Canceling Microphone, M.2 with heatsink, USB 3.2 Gen 2, SATA 6 Gbps and Aura Sync RGB lighting.",

8

"price": "36990",

9

"category": "motherboard",

10

"path": "asus-rog-strix-b450-f-gaming-ii.jpg"

11

}

12

}



## 7.8 Megrendelések (Orders)

Felhasználóknak természetesen lehetőségük van rendelések leadására. A frontenden tárolt kosár tartalmát alakítja át rendeléssé. Rendelés leadását követően a vásárló e-mailes tájékoztatót kap arról, hogy miket vásárolt.

POST
localhost:3000/api/place-order
Send

Status: 200 OK
Size: 27 Bytes
Time: 151 ms

Response
Headers<sup>9</sup>
Cookies
Results
Docs<sup>New</sup>

```

1 {
2   "message": "Order placed!"
3 }

```

### Your order has been recieved by us!

Dear Robert Tamas!

Thank you for using our webshop!


If you chosed pre-payment. You shall transfer the purchase price on the basis of the fee request sent by the following e-mail, writing the exact order ID in a notice: JGPDM5 Our bank account number: 12345678-12345678-12345678 Owner: NasaPC Important! If you have not received a fee request, please contact our customer service team before making a transfer.

**Your order ID: JGPDM5**

**Personal information:**

Customer:	Robert Tamas
Email:	<a href="mailto:tamasrobert00@gmail.com">tamasrobert00@gmail.com</a>
Contact phone number:	+36301234567
Billing address:	Wonderland
Ship-to Address:	Wonderland

**Ordered products:**

Image	Product	Total
	1x AMD Ryzen 5 5600X 6-core, 12-Thread Unlocked Desktop Processor with Wraith Stealth Cooler	74.990 Ft
	<b>Total</b>	<b>74.990 Ft</b>

Payment method: cash on delivery

Order date: Tue Apr 26 2022 17:22:26 GMT+0200 (Central European Summer Time)

Payable on receipt: 74.990 Ft

## 7.9 Értékelések (Reviews)

A felhasználóknak van lehetőségük megjegyzés/értékelés hozzáfűzéséhez.

Egy felhasználó egy termékhez csak egyszer tud hozzászólni.

Természetesen van lehetősége a megjegyzését törölni. Kötelezően elvárt mező a rating és a comment.

POST ▼ localhost:3000/api/post-review/626853cefcffd060e07c6791 Send

1 ▼ {

2     "rating": "5",

3     "comment": "Love this product, works like a charm."

4     }

Status: 200 OK   Size: 28 Bytes   Time: 154 ms

Response

Headers <sup>9</sup>

Cookies

Results

Docs <sup>New</sup>

1 ▼ {

2     "message": "Review posted!"

3     }

```
_id: ObjectId("626853cefcffd060e07c6791")
quantity: 1
discount: 0
reviews: Array
  0: Object
    userId: ObjectId("62605f2b6942781664790dda")
    email: "tamasrobert00@gmail.com"
    rating: "5"
    comment: "Love this product, works like a charm."
name: "Kingston-FURY-16GB-KIT-DDR4-3200MHz-CL16-Beast-Black"
description: "RAM memória - 2x8GB, PC4-25600, 16-18-18 CL, feszültség: 1,35 V, passz..."
price: "30000"
category: "RAM"
path: "Kingston-FURY-16GB-KIT-DDR4-3200MHz-CL16-Beast-Black.jpg"
```

Status: 400 Bad Request   Size: 72 Bytes   Time: 100 ms

Response

Headers <sup>9</sup>

Cookies

Results

Docs <sup>New</sup>

1 ▼ {

2     "error": "Failed posting review! You have already review this product!"

3     }

## 7.10 Admin

Adminisztrátorok csakis manuálisan állíthatók, és csakis ők tudnak a termékek létrehozásával, módosításával foglalkozni. Az adatbázisban kapnak egy „admin” nevű tulajdonságot, amely ellenőrzésre kerül, ha a felhasználó létre kíván hozni egy terméket. Természetesen, ez igaz a termékek módosítására és törlésére is.

```
const session = req.cookies['LOCAL_KEY'];  
  
if (!session) return res.sendStatus(401);  
  
User.findOne({ session, 'admin': true })
```

```
_id: ObjectId("626060166942781664790ddf")  
> wishList: Array  
email: "admin@nasapc.com"  
password: "$2b$05$2FzyvC3MIX0V7fNzhCJkX.Raxeru6w5t28P8YVaalqy1KkM6F6.16"  
admin: true
```

Sikertelen azonosítás esetén a funkció nem elérhetőek, a megfelelő hibakód visszaküldésre kerül.

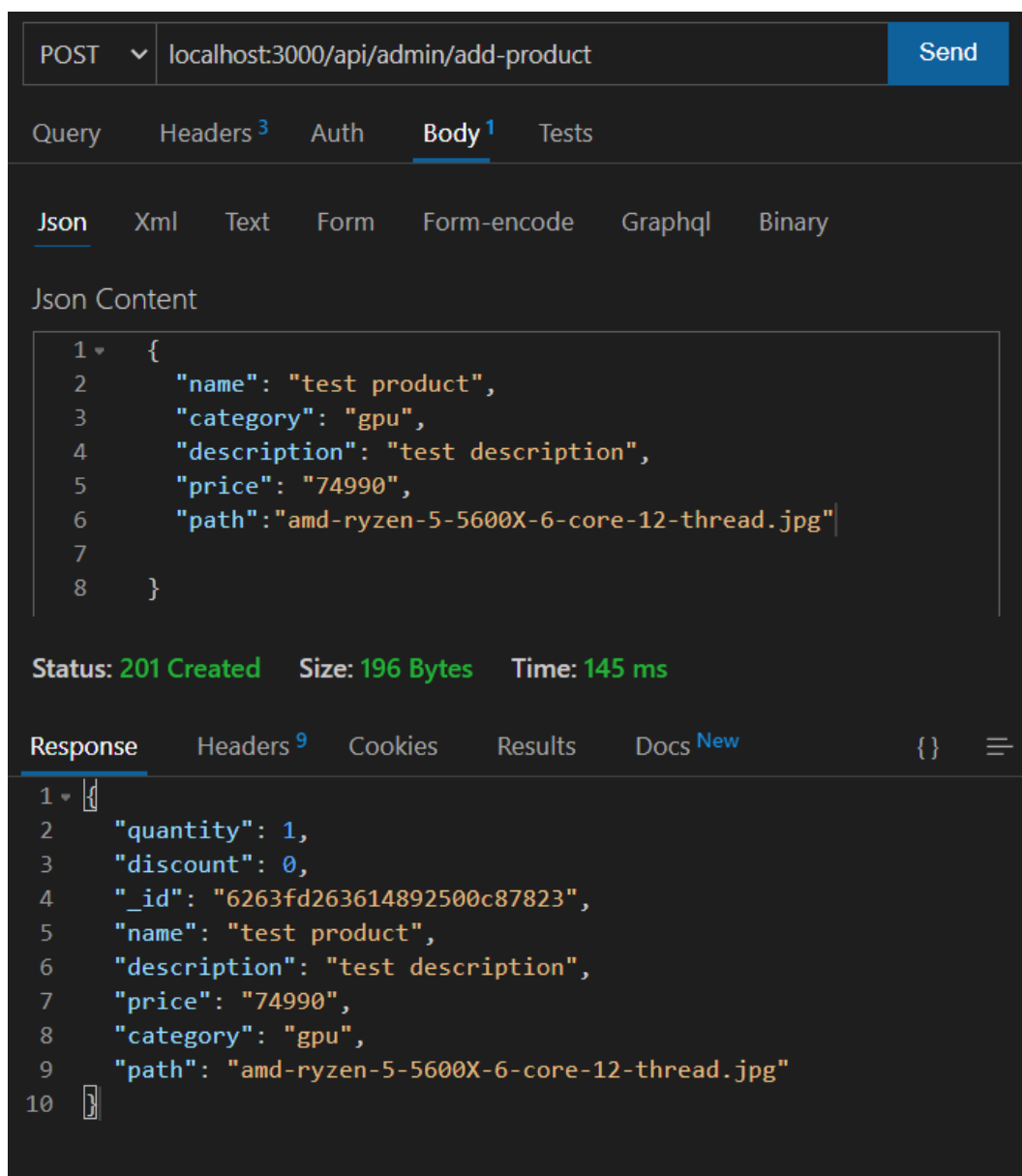
Status: 401 Unauthorized Size: 12 Bytes Time: 2 ms

Response Headers<sup>9</sup> Cookies Results Docs [New](#)

1 Unauthorized

### 7.10.1 Termék létrehozása

Új termék létrehozásához a már említett admin jog szükséges. Új termék felvitelkor kötelezően elvárt mezők közé tartozik a termék neve, leírása, ára, kategóriája. Az akció mértéke nem elvárt, alapból 0, azaz 0% akció lesz a terméken. Létrehozás után a termék darabszáma 1-re lesz beállítva, ami módosítható.



POST localhost:3000/api/admin/add-product Send

Query Headers<sup>3</sup> Auth **Body<sup>1</sup>** Tests

Json Xml Text Form Form-encode Graphql Binary

Json Content

```
1 {
2   "name": "test product",
3   "category": "gpu",
4   "description": "test description",
5   "price": "74990",
6   "path": "amd-ryzen-5-5600X-6-core-12-thread.jpg"
7 }
8
```

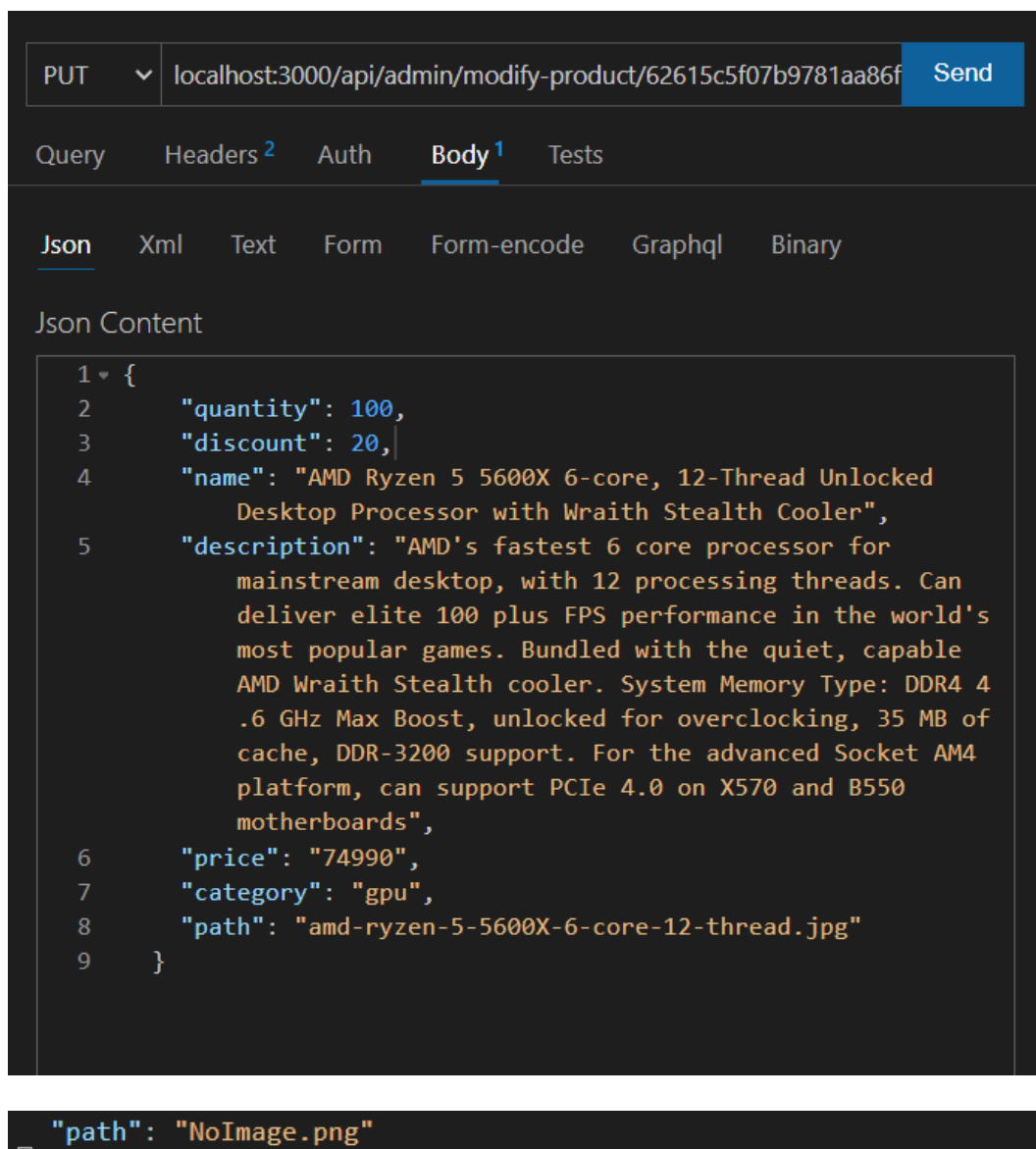
Status: 201 Created Size: 196 Bytes Time: 145 ms

Response Headers<sup>9</sup> Cookies Results Docs [New](#) {} ≡

```
1 {
2   "quantity": 1,
3   "discount": 0,
4   "_id": "6263fd263614892500c87823",
5   "name": "test product",
6   "description": "test description",
7   "price": "74990",
8   "category": "gpu",
9   "path": "amd-ryzen-5-5600X-6-core-12-thread.jpg"
10 }
```

### 7.10.2 Termék módosítása

Hasonlóképpen, mint a termékek létrehozásához, a termékek módosítását is csak adminisztrátor végezheti. Kötelezően elvárt mező nincsen, amely mezőket nem adjuk meg, azok nem kerülnek módosításra, a path kivételével. Amennyiben a path-ot nem adjuk meg, az automatikusan a NoImage.png path-ot választja magának.



```
PUT localhost:3000/api/admin/modify-product/62615c5f07b9781aa86f Send

Query Headers2 Auth Body1 Tests

Json Xml Text Form Form-encode Graphql Binary

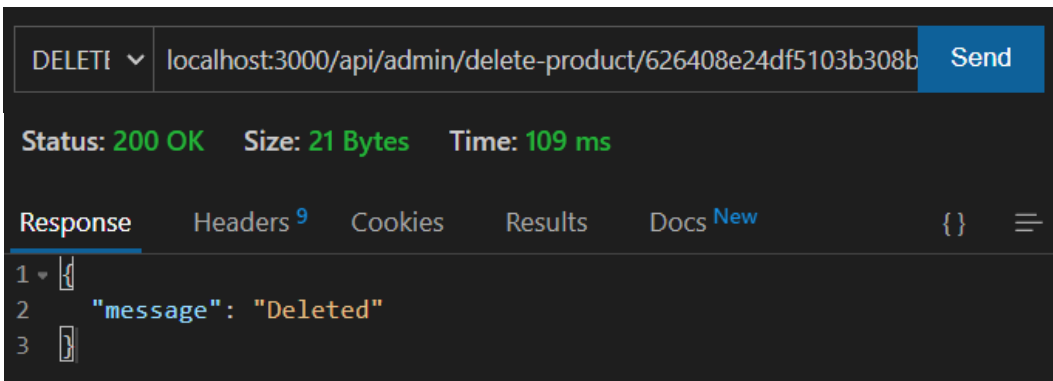
Json Content

1 {
2   "quantity": 100,
3   "discount": 20,
4   "name": "AMD Ryzen 5 5600X 6-core, 12-Thread Unlocked
      Desktop Processor with Wraith Stealth Cooler",
5   "description": "AMD's fastest 6 core processor for
      mainstream desktop, with 12 processing threads. Can
      deliver elite 100 plus FPS performance in the world's
      most popular games. Bundled with the quiet, capable
      AMD Wraith Stealth cooler. System Memory Type: DDR4 4
      .6 GHz Max Boost, unlocked for overclocking, 35 MB of
      cache, DDR-3200 support. For the advanced Socket AM4
      platform, can support PCIe 4.0 on X570 and B550
      motherboards",
6   "price": "74990",
7   "category": "gpu",
8   "path": "amd-ryzen-5-5600X-6-core-12-thread.jpg"
9 }
```

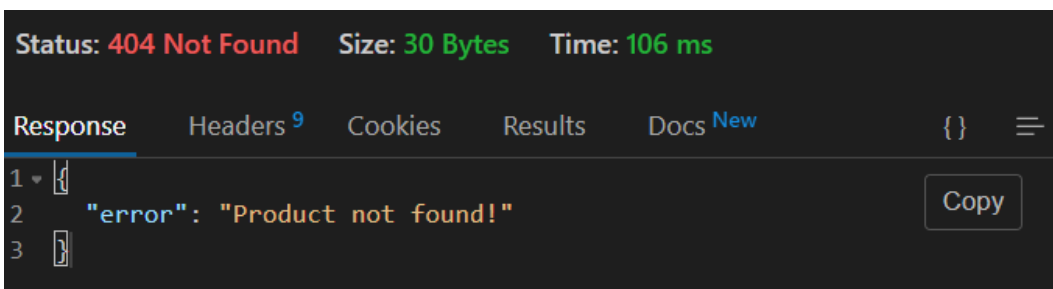
```
"path": "NoImage.png"
```

### 7.10.3 Termék törlése

Termék törlését is csakis adminisztrátor végezheti. Az útvonalban átadott termék ID-ja alapján a program megkeresi majd kitörli a terméket.



Nem létező termék esetén hibaüzenet visszaadása megtörténik.



Fontos észben tartani a kívánságlistát. Nem jó megoldás az, hogyha egy kitöröl termék még mindig ott lenne törlés után valamelyik, esetleg több felhasználó listájában. Éppen ezért, a függvény végigmegy a létező felhasználók WishList-jén és frissíti azokat.



Termék törlése után:



## 7.11 Backend teszt

Egy jó programhoz elvárás, hogy legyenek tesztek. A tesztek segíthetnek a program ellenőrzésében, megkönnyítve a munkánkat, hiszen, ha egy új funkciót beépítünk a programba, elég lefuttatni a már létező teszteket az ellenőrzésre. A Thunder Client „tesztelés” nem megfelelő, hiszen idő szempontjából nagyon lassú minden funkciót letesztelni. Ezért a cél az volt, hogy a futtatható teszt fájl kerüljön megvalósításra. Egy Windows batch fájlal is indítható teszt lett a végeredmény, amely a backend funkcióit teszteli. Bármiféle hiba esetén, az adott teszt vagy tesztek nem futnak le.

```
-----
WebShopBackend API Tests:
-----
Back-end server started on port: 3000
✓ Should be able to login (848ms)
✓ Should be able to get all products (101ms)
✓ Should be able to get a product by id (55ms)
✓ Should be able to get wishlist (51ms)
✓ Should be able to add a product to wishlist (164ms)
✓ Should be able to remove a product from wishlist (160ms)
✓ Should be able to get a session (52ms)
✓ Should be able to post a new product (119ms)
✓ Should be able to modify the new product (316ms)
  Wishlist update for all users is done.
  If the product existed in any wishlist, it is now deleted.
✓ Should be able to delete the new product (157ms)
✓ Should be able to request password change (63ms)
✓ Should be able to logout (167ms)

-----
WebShopBackend API Tests:
-----
Back-end server started on port: 3000
✓ Should be able to login (816ms)
✓ Should be able to get all products (103ms)
1) Should be able to get a product by id
✓ Should be able to get wishlist (54ms)
2) Should be able to add a product to wishlist
3) Should be able to remove a product from wishlist
4) Should be able to get a session
✓ Should be able to post a new product (120ms)
✓ Should be able to modify the new product (255ms)
  Wishlist update for all users is done.
  If the product existed in any wishlist, it is now deleted.
✓ Should be able to delete the new product (158ms)
✓ Should be able to request password change (62ms)
✓ Should be able to logout (171ms)

8 passing (2s)
4 failing
```

### 7.11.1 Bevezetés

A teszteléshez a chai-http nevű modult használtam. Először is különálló teszt mappát hoztam létre, amiben található egy JavaScript fájl. A backend főkönyvtárából kiadva az npm test parancs ezt a test JavaScript fájl fogja elindítani.

Először is importáltam a modulokat, amelyekre szükségem lesz a tesztek megírásához. A sütik kezelésére csakis úgynevezett „agent” típusú változó volt alkalmas. Ez a legfontosabb eleme a kódnak, agent nélkül nem fogok tudni sütiket alkalmazni, ami problémás lenne, hiszen maga a backend használ sütit.

```
// required modules
let app = require('../app');
let chai = require('chai');
let chaiHttp = require('chai-http');
var _ = require('lodash');
const Product = require('../models/product');
const { expect } = require('chai');
```

Ezen kívül létrehoztam báb adatokat, amelyek automatikusan felhasználásra kerülnek a program futása közben, nincs szükség, hogy kézzel állítsuk be.

```
// dummy-data for add-product
const product = new Product({
  "name": "test name",
  "category": "phone",
  "description": "test description",
  "price": "69",
  "path": "test path",
  "discount": 100
});

// dummy-data for modify-product
const updatedProduct = new Product({
  "name": "test name updated",
  "category": "phone",
  "description": "test description updated",
  "price": "360",
  "path": "test path updated",
  "discount": 80
});
```

Egy hibaellenőrző logikát is tartalmaz a kód, ami eltárolja a hibákat, hiszen több is előfordulhat; ezeket mind érdemes ki is íratni.

```
// error storage
let error = product.validateSync();
let errorCount = 0;
if (error) { errorCount = _.size(error.errors) }
```



### 7.11.2 Bejelentkezés

A teszt a backend legtöbb funkcióját leteszteli. Admin belépéssel kezd, hiszen előnyös lenne, hogyha az admin funkciókat is le lehetne ellenőrizni. Az admin is tulajdonképpen egy felhasználó, így a felhasználót érintő funkciók is tesztelhetők.

```
it('Should be able to login', function (done) {
  agent
    .post('/api/login')
    .send({ email: 'admin@nasapc.com', password: 'admin' })
    .end(function (err, res) {
      expect(res).to.have.status(200);
      expect(agent).to.have.cookie('LOCAL_KEY');
      done();
    });
});
```

A teszt elküldi a megfelelő útvonalra az adatokat, és azt követően vizsgálja, hogy milyen válasz érkezett (end szekció). Jelen esetben az az elvárandó, hogy a kód 200-as http státusz kódot küld vissza. Ezen felül elvárt, hogy maga a meghívó agent tartalmazza a bejelentkezéskor kapott sütit. Ez utóbbi elengedhetetlen a folytatáshoz, enélkül nem lehetséges az autentikációt igénylő funkciók letesztelése.

### 7.11.3 Munkamenet (Session)

```
it('Should be able to get a session', function (done) {
  agent
    .get('/api/session')
    .end(function (err, res) {
      expect(res).to.have.status(200);
      expect(res.body[0]).to.have.property('email');
      expect(res.body[0].email).to.equal('admin@nasapc.com');
      expect(res.body[0].admin).to.equal(true);
      expect(agent).to.have.cookie('LOCAL_KEY');
      done();
    });
});
```

A bejelentkezést követően az aktuális munkamenet lekérdezése a cél. Elvárás, hogy belépés után 200-as státusz kód kerül visszaküldésre. Ezen kívül a válaszban elvárt, hogy legyen email mező, amelynek értéke megegyezik az admin e-mailjével. Természetesen nem felejtendő a süti sem. További elvárás, hogy az admin mező értéke true legyen, hiszen szeretném az admin funkciókat is ellenőrizni.

### 7.11.4 Termékek / Termék ID alapján

Mindezek előtt leteszteltem a nem autentikációt igénylő végpontok között: Az összes termék lekérdezését, és egy specifikus termék lekérdezését. Utóbbit a termék ID-je alapján. Ezek a funkciók nem igényelnek ellenőrzést, hiszen vendégek is járhatnak az oldalon, és nekik is meg kell adni a lehetőséget, hogy böngésszék a termékeket, vagy egyetlen termék adatait is megtekintsék.

```
it('Should be able to get all products', function (done) {
  agent
    .get('/api/products')
    .end(function (err, res) {
      expect(res).to.have.status(200);
      expect(res.body[0]).to.have.property('_id');
      expect(res.body[0]).to.have.property('name');
      expect(res.body[0]).to.have.property('description');
      expect(res.body[0]).to.have.property('category');
      expect(res.body[0]).to.have.property('price');
      expect(res.body[0]).to.have.property('reviews');
      done();
    });
});

it('Should be able to get a product by id', function (done) {
  agent
    .get('/api/product/6269af62764b955a18307115')
    .end(function (err, res) {
      expect(res).to.have.status(200);
      expect(res.body).to.have.property('_id');
      expect(res.body).to.have.property('name');
      expect(res.body).to.have.property('description');
      expect(res.body).to.have.property('category');
      expect(res.body).to.have.property('price');
      expect(res.body).to.have.property('reviews');
      done();
    });
});
```

A teszt a válaszban elvárja, hogy létezzenek a fontosabb mezői a terméknek; amennyiben ez nem valósul meg, akkor a teszt ezen része hibával tér vissza.

### 7.11.5 Kívánság lista (WishList)

Visszatérve a bejelentkezést igénylő funkciókhoz, előnyös lenne a kívánság lista működését is letesztelni. Ez a teszt csak akkor fog lefutni, hogyha az agent-nek létezik sütije. Amennyiben nem létezik, ez a teszt hibával tér vissza, pontosabban 401-es hibakóddal.

```
it('Should be able to add a product to wishlist', function (done) {  
  agent  
    .post('/api/add-to-wishlist/6269af62764b955a18307115')  
    .end(function (err, res) {  
      expect(res).to.have.status(200);  
      expect(res.body).to.have.property('message');  
      expect(res.body.message).to.equal('Product has been added to your wishlist.')  
      done();  
    });  
});
```

Elvárás, hogy a 200-as státusz kód érkezzon a válaszban. Szintén elvárás, hogy a válasz törzsében létezzen egy message nevű mező, aminek meg kell egyeznie a backend-ről visszaküldött helyes válasszal. Amennyiben ez megérkezett, a funkció helyesen működik.

Azt is érdemes lenne ellenőrizni, hogy lekérhető a kívánságlista tartalma.

```
it('Should be able to get wishlist', function (done) {  
  agent  
    .get('/api/wishlist')  
    .end(function (err, res) {  
      expect(res).to.have.status(200);  
      expect(res.body).to.be.an.instanceOf(Object);  
      done();  
    });  
});
```

Jelent esetben a válaszban elvárás a 200-as státusz kód, és az, hogy a válasz törzse egy objektumnak felel meg. Ezen kívül szeretném ellenőrizni, hogy adott termék törölhető a kívánságlistából. Amennyiben nem találja a kívánságlistában a terméket, vagy egyéb hiba van a tesztben, akkor hibával kell visszatérnie. Elvárás, hogy sikerüljön kitörölni a nemrég kívánság listához adott terméket.

```
it('Should be able to remove a product from wishlist', function (done) {  
  agent  
    .post('/api/remove-from-wishlist/6269af62764b955a18307115')  
    .end(function (err, res) {  
      expect(res).to.have.status(200);  
      expect(res.body).to.have.property('message');  
      expect(res.body.message).to.equal('Your Wishlist has been updated.')  
      done();  
    });  
});
```

### 7.11.6 Termék létrehozása

Az admin műveleteit illetően, új termék felvitele egy elvárt funkció, ezért kihagyhatatlan az ellenőrzése.

```
it('Should be able to post a new product', function (done) {  
  
  if (errorCount !== 0) {  
    throw error.errors;  
  }  
  
  agent  
    .post('/api/admin/add-product')  
    .send(product)  
    .end(function (err, res) {  
      expect(Object.keys(res.body).length).toEqual(10)  
      expect(res).to.have.status(201);  
      expect(res.body)  
        .to.be.an.instanceof(Object)  
        .that.includes.all.keys(['_id', 'name', 'price', 'description', 'category', 'path', 'discount',  
      expect(agent).to.have.cookie('LOCAL_KEY');  
      done();  
    });  
});
```

Ebben a tesztben több mező is ellenőrzésre kerül, és maga a funkció rendkívül fontos, ezért érdemes nem csak egy általános hibával visszatérni hiba esetén, hanem az összes hibát kidobni a képernyőre.

Először elküldésre kerül a báb adat, majd a válaszban elvárás, hogy 201-es státusz kód kerüljön visszaadásra. Ezen kívül a válasz törzsének 10 kulcsot kell tartalmaznia, amelyek név szerint elvártak (például \_id, name, price).

### 7.11.7 Termék módosítása

A termék módosításának lehetősége is egy olyan funkció, amelynek hibás működése nem tolerálható. Itt is hiba, vagy hibák esetén az összes hiba visszaküldésre kerül. Azonban jó lenne az utolsó elemet, vagyis amit a tesztben felvittünk, megtalálni. Ezért még a végpont hívása előtt az összes termék lekérdezésre kerül, melyek közül egy, a legfrissebb kerül átadásra egy prod nevű változóba. Mindezek után elvárás, hogy a válasz 200-as státusz kódot adjon vissza, a válasz törzse objektum legyen, majd hasonlítsa össze a visszaadott értékeket a kód elején megadottakkal.

```
it('Should be able to modify the new product', function (done) {  
  
  if (errorCount !== 0) {  
    throw error.errors;  
  }  
  
  Product.findOne({}, {}, { sort: { '_id': -1 } }, function (err, prod) {  
    agent  
      .put('/api/admin/modify-product/' + prod._id)  
      .send(updatedProduct)  
      .end(function (err, res) {  
        expect(res).to.have.status(200);  
        expect(res.body)  
          .to.be.an.instanceof(Object)  
          .that.includes.all.keys(['_id', 'name', 'price', 'description', 'category', 'path', 'discount']);  
        expect(res.body.name).toEqual(product.name + ' updated')  
        expect(res.body.category).toEqual(updatedProduct.category)  
        expect(res.body.description).toEqual(product.description + ' updated')  
        expect(res.body.price).toEqual(updatedProduct.price)  
        expect(res.body.path).toEqual(product.path + ' updated')  
        expect(res.body.discount).toEqual(updatedProduct.discount)  
        expect(agent).to.have.cookie('LOCAL_KEY');  
        done();  
      });  
  });  
});
```

### 7.11.8 Termék törlése

Termékek törlése is olyan funkció, melynek megléte elengedhetetlen, éppen ezért tesztet igényel.

Hasonlóképpen a termék módosításához, ez a teszt is megkeresi a legutolsó / legfrissebb terméket, majd törli azt. Elvárt a 200-as státusz kód, és a megfelelő megerősítő üzenet megléte.

```
it('Should be able to delete the new product', function (done) {  
  
  if (errorCount !== 0) {  
    throw error.errors;  
  }  
  
  Product.findOne({}, {}, { sort: { '_id': -1 } }, function (err, prod) {  
    agent  
      .delete('/api/admin/delete-product/' + prod._id)  
      .end(function (err, res) {  
        expect(res).to.have.status(200);  
        expect(res.body).to.be.an.instanceOf(Object)  
          .that.has.property('message');  
        expect(res.body.message).to.equal('Deleted')  
        expect(agent).to.have.cookie('LOCAL_KEY');  
        done();  
      });  
  });  
});
```

### 7.11.9 Új jelszó igénylése

Általános felhasználói funkció a jelszóváltoztatás. Ebben a tesztben csak maga a jelszó változtatási kérelem kerül ellenőrzésre.

```
it('Should be able to request password change', function (done) {  
  agent  
    .post('/api/request-password-change')  
    .send({ email: 'admin@nasapc.com' })  
    .end(function (err, res) {  
      expect(res).to.have.status(200);  
      expect(res.body).to.have.property('message');  
      expect(res.body.message).to.equal('Password change request sent!')  
      expect(agent).to.have.cookie('LOCAL_KEY');  
      done();  
    });  
});
```

A végpont csak egy email mezőt vár a kérés törzsében. Elküldés után a válaszban 200-as státusz kód az elvárt. Ezen kívül a message mező megléte és a megfelelő megerősítő üzenet is.

#### 7.11.10 Termék módosítása (Modify product)

A legutolsó teszt csak annyit ellenőriz, hogy képes-e kijelentkezni a felhasználó.

```
it('Should be able to logout', function (done) {  
  agent  
    .get('/api/logout')  
    .end(function (err, res) {  
      expect(res).to.have.status(200);  
      expect(agent).to.not.have.cookie('LOCAL_KEY');  
      done();  
    });  
});
```

Az elvárás ebben az esetben az, hogy a válasz 200-as státusz kódot tartalmazzon, és hogy az agent-nek ne legyen már LOCAL\_KEY nevű sütije.

## 8. Android app

### 8.1 Android Alkalmazás

A webshop egyszerűbben használható a mobil alkalmazás segítségével, hiszen így a felhasználó könnyebben tud navigálni a termékek és menük között, mint egy weboldalon. Mivel az alkalmazás telepítésre kerül az eszközön ezzel a felhasználó adatforgalma is csökkenthető, hiszen nem szükséges az összes fájlt letölteni, csak, ami az adatbázisból kell.

A telefonos alkalmazásnál több szempontot is figyelembe vettünk, hogy melyik eszközöket érné meg legjobban célozni. Mivel országunk túlnyomó többsége Android operációs rendszerrel ellátott eszközöket használnak, így Java programozási nyelven írtuk meg az alkalmazást az Android Studio program segítségével. Ezzel, hogy natív kódot tudunk írni az eszközre, sokkal gyorsabban futtatható, optimalizált és kevesebb tárhelyet foglaló



alkalmazást érhetünk el. Hátrányban sajnos elmondható, hogy így iOS-en nem futtatható az alkalmazás, de más szempontból se lett volna előnyös, hiszen iOS-re az alkalmazást csak megbízható fejlesztőtől lehetne telepíteni, míg Android-ra ez nem szükséges.

Az alkalmazásban a backend-et használva tudunk adatokat kinyerni az API segítségével, hiszen az adatok az adatbázisban vannak eltárolva. A frontend szerverre is szükség van, hiszen a képek csak ott vannak eltárolva.

Mivel hogy a projekthez nem béreltünk VPS-t, így nincs állandó IP cím, ahonnan az adatokat le lehessen kérdezni, ezért az Android alkalmazás program kódjában folyamatosan meg kell változtatni a backend és a frontend URL-nek a címét, hogy a telefonos alkalmazás a megfelelő helyről kérje le a szükséges adatokat. Ezeket a szerver címeket a Variables osztályban érhetjük el, ahol az adatok nagy része van tárolva.

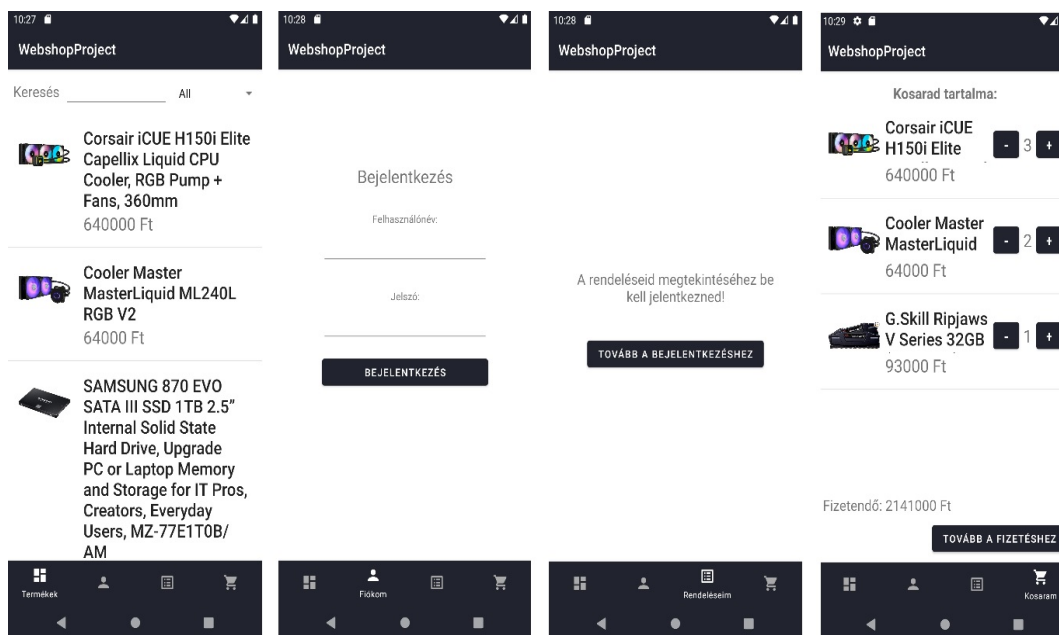
```
public class Variables {  
    //DON'T USE LOCALHOST OR 127.0.0.1, USE PRIVATE IPV4 ADDRESS INSTEAD  
    private static final String backendUrl = "http://10.0.41.16:3000"; //backend is needed for requesting data from the database  
    private static final String frontendUrl = "http://10.0.41.16:8080"; //frontend is needed for requesting images, because it's not stored in the backend
```

## 8.2 Az alkalmazás felépítése:

Az Android alkalmazás tartalmaz egy alsó navigációs menüt 4 elemmel:

- Termékek listája (Főoldal)
- Profil (Hitelesítés)
- Rendelések (Csak bejelentkezett felhasználóknak)
- Kosár

## Illusztráció a menükről:



A termékek listája a [GET] [http://<backend\\_szerver>:<port>/api/products](http://<backend_szerver>:<port>/api/products) – ról töltődnek be. Ilyenkor megjelenítésre kerül a termék neve, ára és képe. Hogyha a felhasználó részletesebb leírást szeretne kapni, akkor rákattinthat a listában lévő termékekre, így elolvashatja a termék leírását és a képet is nagyobb felbontásban láthatja, de ezen túl itt adhatja hozzá a terméket is a kosárhoz.

A termék sikeres hozzáadását egy felugró kis ablak (Toast) jelzi.

A profil menüpontban a felhasználó betud jelentkezni az alkalmazásban. Ez az oldal önmagában nem szolgáltat túl sok információt, de a későbbiekben még bővíthető lehetne egyéb funkciókkal, pl. jelszó változtatás, vagy egyéb





felhasználónak szánt beállítás megváltoztatásához. Jelenleg csak a bejelentkezett felhasználó e-mail címét jeleníti meg ez az oldal.

A megrendelések kilistázásához külön route-ok lettek létrehozva, hiszen az Android-os alkalmazással nem lehetett megoldani a Cookie-k kezelését, vagy csak nagyon nehezen lehetett volna, ezért az adatokat a [GET] `http://<backend_szerver>:<port>/api/mobile/getUserOrders/<userId>/<sessionKey>` címről tudjuk lekérdezni. A lekérdezett adatokból egy listát generál az alkalmazás a felhasználónak, itt megtekintheti a meglévő megrendelésének adatait, mint a: rendelés azonosító (6 jegyű kód), megrendelt termékek száma, a megrendelt termék neve és a fizetett ár.

Végül az utolsó menüpontban a kosarunk tartalmát láthatjuk. Ha nincs semmi a kosárban, akkor az „Üres a kosarad” felirat jelenik meg, ilyenkor elvan rejtve a megrendelés gombja, és a összesen fizetendő felirat. De hogyha valami hozzáadásra kerül, akkor ezek megjelennek és automatikusan frissülnek a kosár állapotának változásáról. A kosárban a terméknel a mínusz (-) és plusz (+) jelekkel tudjuk a megrendelni kívánt termékből a darabszámot módosítani, hogyha ez az érték 1 alá csökkenne, akkor automatikusan eltávolítódik a termék a kosárból.

Hogyha a felhasználó kiválasztotta a számára megfelelő termékeket, akkor kosárban a „tovább a megrendeléshez” gomb segítségével egy új oldalra navigáljuk, majd itt egy űrlapot kell kitölteni-e, ahol meg kell adnia a nevét, telefonszámát, email címét és a szállítási címet. Külön el kell fogadni a vásárlással kapcsolatos szabályzatot, ahol a checkbox-ot be kell pipálni, különben a rendelés nem kerül feladásra, és egy hiba üzenetet kapunk. De

ugyan így a többi mező is validálásra kerül, így üres mezőkkel nem lehet rendelést feladni. A rendelés feladásakor a [POST]

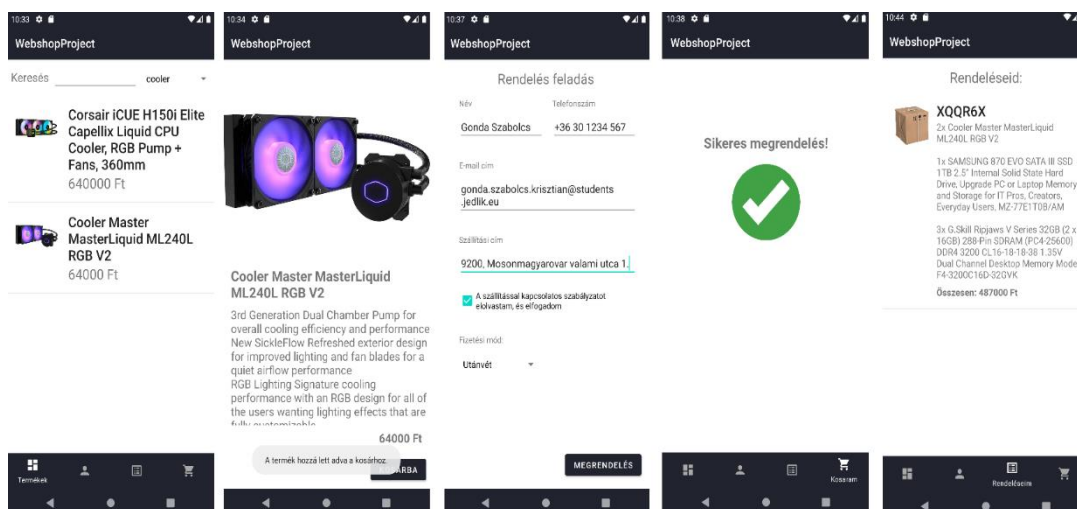
`http://<backend_szerver>:<port>/api/mobile/place-order/<sessionKey>`

route hívódik meg, ahol a body-ban átadásra kerülnek a következő értékek:

- first\_name
- last\_name
- email
- phone\_number
- shipping\_address
- billing\_address
- items

A rendelés feladása után a felhasználót átirányítjuk a következő oldalra, ahol a „Sikeres rendelés feladás” fogadja. Ezután a bejelentkezett felhasználó megtudja nézni a feladott rendelését.

Illusztráció a megrendelés folyamatáról:





## 9. Kommunikáció a csapattagok között:

A csapat tagjainak a fejlesztés során volt lehetősége személyes kapcsolattartásra így a közös munka zavartalanul tudott zajlani a fejlesztés túlnyomó részében, ám amikor ez az állapot nem állt fenn, több különböző módot is felhasználtunk a kapcsolattartásra.

Az elsődleges módja a kommunikációnak a Facebook Messenger használata volt, illetve az azon való chatelés, de a fejlesztés első fázisaiban más módszereket is fontolóra vettünk.

A Discord is nyújtott volna egy stabil platformot a zavartalan kommunikáció fenntartására. A Discord egy kommunikációs platform, amely lehetőséget nyújt több fél közti chatelés, streamelés és meetek indítására, továbbá képek fájlok és egyéb kisebb tartalmak közzétételére.

A projektet a GitHubon keresztül fejlesztettük. A GitHub segítségével nyomon tudtuk követni ki, mit tett közé, és esetleges ütközések esetén könnyen tudtuk a hibákat kezelni.

## 10. Felhasznált források:

[Mit is jelent pontosan a JavaScript kifejezés és hogyan működik, mire jó? \(matebalazs.hu\)](https://matebalazs.hu)

[Node.js alapok · Weblabor](#)

[What is Express.js? | Why should use Express.js? | Features of Express.js \(besanttechnologies.com\)](https://besanttechnologies.com)

[Index | Node.js v17.8.0 Documentation \(Node.js.org\)](https://nodejs.org)

[Introduction | Vue.js \(vuejs.org\)](https://vuejs.org)