



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>: Szabó Tamás Martin, Kovács Bence Dominik  
Képzés: nappali munkarend  
Szak: 5 0613 12 03 Szoftverfejlesztő és tesztelő technikus

## A záródolgozat címe: „Electrical shop” webshop

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

Győr, 2021. 10. 01

---

Módos Gábor  
igazgató

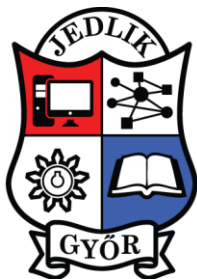
---

<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>4</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.

---

Szabó Tamás Martin

---

Kovács Bence Dominik

---

<sup>4</sup> Szakmajegyzékes, csoportos konzultációs lap

Szabó Tamás Martin és Kovács Bence Dominik

**2/14F**

**„Electrical shop”**

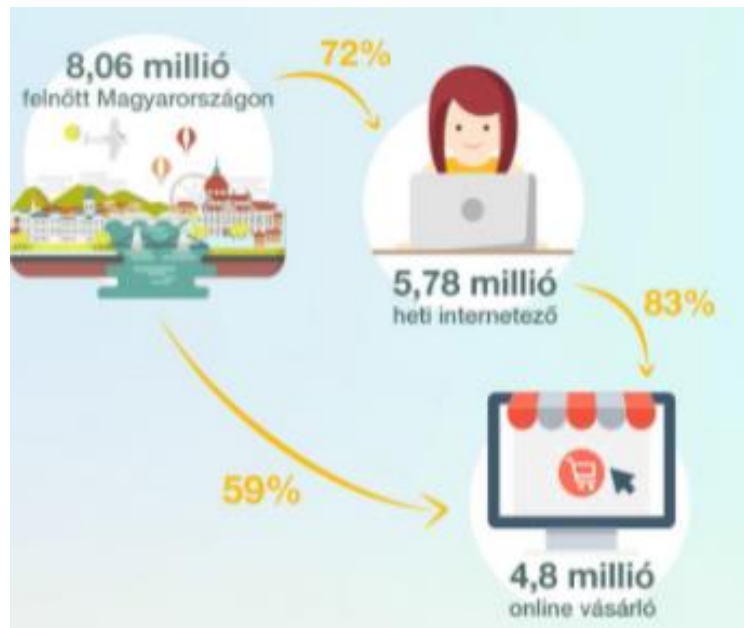
# Tartalomjegyzék

1. Bevezető .....	6
1.1. Project alapvető jellemzői .....	7
1.2. Felhasználói felület kinézete .....	7
2 Használt szoftverek, keretrendszerek .....	8
2.1. Vue.js .....	8
2.2. Node.js.....	9
2.3 MongoDB.....	10
2.4. Firebase .....	11
3 Frontend (Vue Js).....	12
3.1. Mappa struktúra.....	12
3.2. Használt pluginok .....	12
3.3. Navigációs bár .....	13
3.4. Bejelentkezés és Regisztráció .....	16
<b>3.4.1. Beépítés a programba .....</b>	<b>17</b>
3.5 Komponensek.....	19
<b>3.5.1. Főoldal.....</b>	<b>19</b>
<b>3.5.2. Admin felület .....</b>	<b>20</b>
<b>3.5.3. Termék hozzáadása .....</b>	<b>21</b>
<b>3.5.4. Termékeink .....</b>	<b>22</b>
<b>3.5.6. Footer.....</b>	<b>24</b>
3.6. Reszponzív dizájn.....	25
<b>3.6.1 Mobiltelefon használat.....</b>	<b>25</b>
<b>3.6.2 Oldalunk reszponzivitása.....</b>	<b>26</b>
<b>3.6.3 Képek az oldalról .....</b>	<b>27</b>
4. Backend (Node.js, Mongodb).....	27
4.1 Models/Schema.....	28
4.2 Controllers.....	28
4.3 Routes.....	29
4.4 Server.js, db.config.js és Adatbázis .....	30
4.5 Backend tesztek.....	30

4.5.1 Új termék felvétele jó, és rossz ID esetén.....	31
4.5.2 Egy termék lekérdezése jó és rossz ID alapján .....	32
4.5.3 Egy termék frissítése ID alapján .....	33
4.5.4 Termék lekérdezése név(title) alapján.....	34
6. Összegzés.....	36
6.2 Jövőbeli tervek.....	37
6.1 Felhasznált források .....	38

# 1. Bevezető

Napjainkban nagyon sok ember szeret vásárolni akár üzletekben, akár nagyobb üzletláncokban, kisebb kereskedésekben, de ez alapvetően mindenki számára egy fárasztó tevékenységnek minősül, mivel el kell menni otthonról megvenni, amit szeretnénk és ez sok időt fel tud emésztetni. Ezért napjaink szerves részévé váltak az internetes boltok és üzletek (webshopok) amik



megkönnyítik az emberek számára a vásárlást, és nem beszélve sok időt tudunk vele megspórolni. Az internetezők négyötöde online vásárló, 43%-uk legalább havonta vásárol, évente átlagosan 12 alkalommal.

Szakedolgozatunk tartalma egy letisztult, egyszerűen kivitelezett elektronikai üzletnek webshopja. Aminek a szerveroldali kiszolgálását (Backend) Node.js szoftverrendszer látja el. A felhasználóval közvetlen kapcsolatba levő (Frontend) megvalósításához a Vue.js keretrendszer segítette a munkánkat. Az adatbázis létrehozását, kialakítását és kivitelezését pedig a MongoDB dokumentumorientált adatbázis szoftver tette lehetővé.

Azért választottuk ezeket a szoftvereket és keretrendszereket, mivel tanulmányaink során nagyon szimpatikussá vált a használatuk és tanulási nehézségük, illetve könnyen találtunk hozzá segédanyagokat és oktató videókat az interneten. Alapvetően a munkamegosztás nem okozott nehézséget számunkra, mivel a tervezési fázisban tudtuk, hogy ki melyik részével szeretne foglalkozni a projektnek, viszont, ha elakadtunk akkor a másik segítségével megoldottuk az adott problémát. A munka gördülékenyen ment.

## 1.1. Project alapvető jellemzői

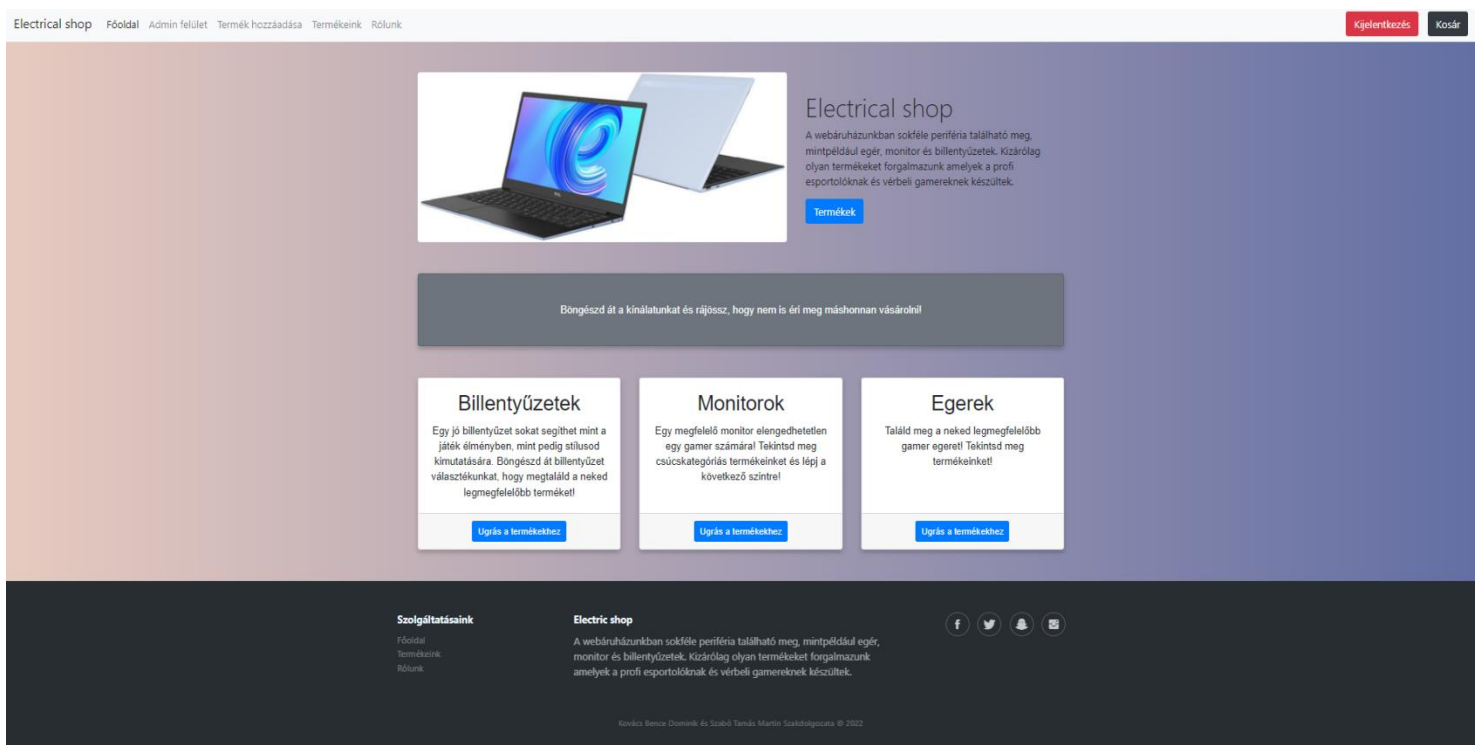
**Project neve:** Electrical shop

**Project kategóriája:** Webshop

**Project főbb funkciói**

- 1. Oldalhoz tartozó felhasználó létrehozása, tárolása, validálása
- 2. Termékek felvétele, tárolása, szerkeszthetősége, állapot frissítése.
- 3. Reszponzív stílus

## 1.2. Felhasználói felület kinézete

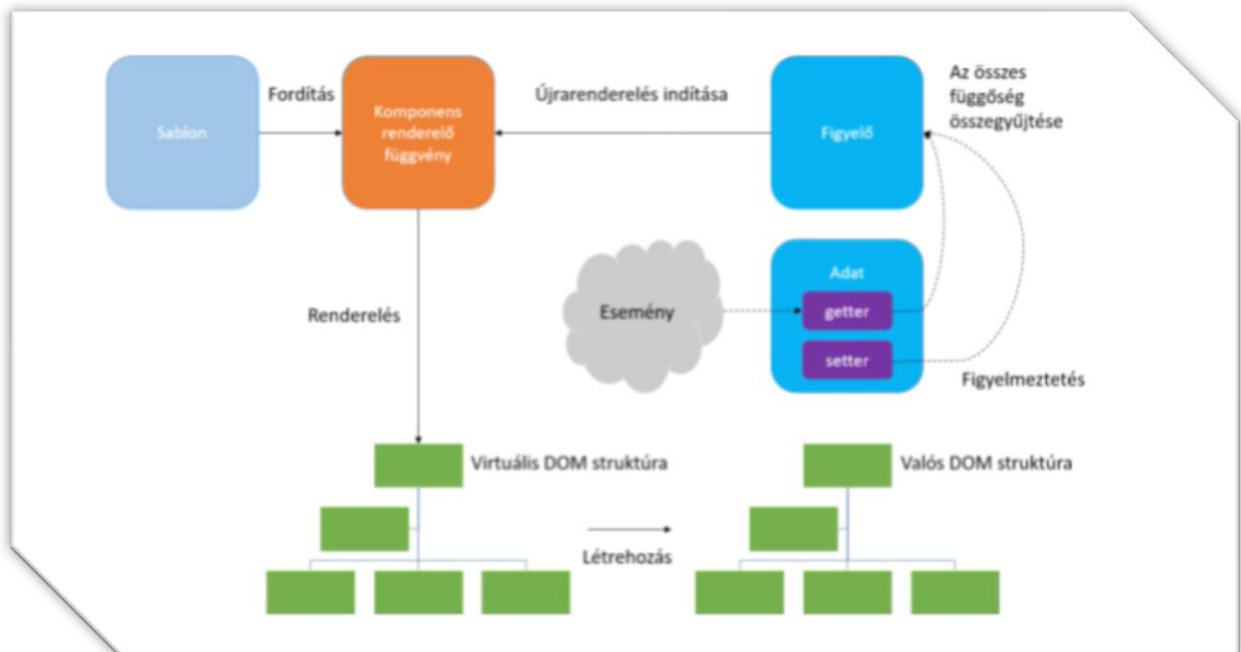


## 2 Használt szoftverek, keretrendszerek

### 2.1. Vue.js

A Vue.js egy reaktív JavaScript keretrendszer. A legfontosabb funkciója az adat összekötés (**data binding**). Segítségével létrehozhatunk egy oldalas weboldalakat, valamint komponenseket is hozhatunk létre. A Vue.js az fejlesztői szerint egy nélkülözhetetlen eszköz a modern webfelületek hatékony fejlesztésében.

A Vue.js egy HTML-alapú sablon szintaxist használ, amiben lehetőségünk adódik, hogy



összekössük a lerenderelt DOM-t a lepdányosított Vue-val kapcsolatos adatokat.

Számos JavaScript keretrendszer, beleértve a Vue.js-t is az úgynevezett Virtuális DOM-t használja. Elméletileg a DOM-ra gondolhatunk úgy, ami reprezentálja a HTML dokumentum felépítését. Gyakorlatilag a DOM egy olyan adatstruktúra, ami akkor jön létre, amint a HTML dokumentumot a böngésző lefordítja. virtuális DOM renderelő függvényekké.





## 2.2. Node.js

Node.js egy olyan futtató környezet, ami lehetőséget nyújt JavaScriptben írt programok futtatására szervereken. JavaScriptet leginkább a kliens oldalon szokták használni, a böngészőn keresztül. A böngésző applikáció fejlesztők számára olyan motorokat kell biztosítani, amik a JavaScript kódot gépi nyelvként interpretálják és le is futtatják.



Annyi könnyedséget nyújt, ha már dolgoztunk frontend oldalon, nem kell egy teljesen új nyelvet megtanulni ahhoz, hogy a backend-et is implementáljuk. A másik előnye az, hogy a non-blocking természetéből adódóan alkalmas egyszerre több ezer bejövő 'egyszerű' kérés kezelésére.



## 2.3 MongoDB

A MongoDB nyílt forráskódú dokumentumorientált adatbázis szoftver, amelyet a MongoDB inc. fejleszt. A NoSQL adatbázisszerverek közé tartozik. A dokumentumokat JSON-szerű formátumban tárolja (BSON). A MongoDB-t olyan nagyobb felhasználók is használják.

A MongoDB fejlesztését 2007-ben kezdték a 10gen-nél, amikor a cég egy platform szolgáltatás fejlesztésén dolgozott. 2009-ben a szoftvert nyílt forráskódúvá tették önálló termékként.

Az 1.4-es verzió 2010 márciusi kiadásával a fejlesztő csapat éles üzemre késznek tartja a terméket.

### Főbb tulajdonságai

- **Ad hoc lekérdezések**

A MongoDB támogatja a keresést mező alapján, érték-tartomány alapján vagy reguláris kifejezéssel. A lekérdezések visszaadhatják a dokumentum egy meghatározott részét és tartalmazhatnak **javascript** funkciókat is.

- **Replikáció**

A MongoDB támogatja a master-slave replikációt. Ebben az esetben a master hajthat végre írás műveleteket, a slave szerverek másolják az adatokat, olvasásra biztonsági mentésre használhatóak. A slave adatbázisok képesek új master adatbázist választani, ha a master meghibásodik.

- **Terheléselosztás**

A MongoDB horizontálisan skálázható sharding használatával. A fejlesztőnek kell shard kulcsot választania, amely meghatározza, hogyan lesz elosztva gyűjtemény adathalmaza.



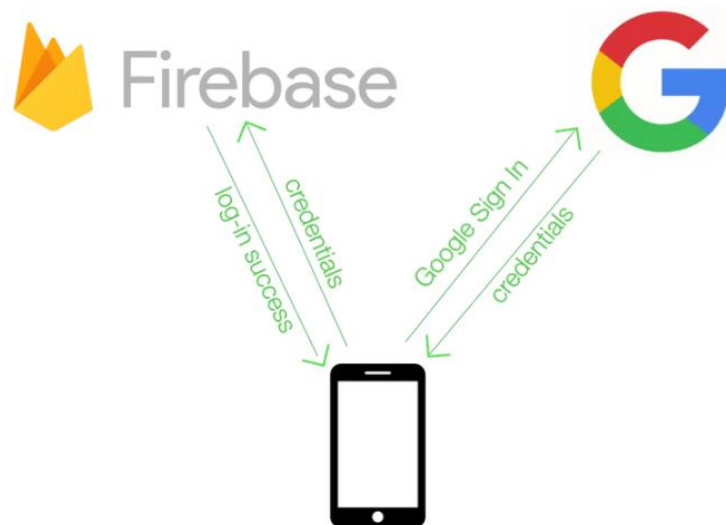
## 2.4. Firebase

### Mi is az a Firebase?

A Firebase egy olyan szolgáltatáscsomag a Google-től, amin keresztül a fenti funkciók mindegyike (és még sok minden más is) megvalósítható backend infrastruktúra és fejlesztő nélkül, egyszerűen, gyorsan akár teljesen ingyen. Firebase önmagában nem egy konkrét termék, hanem sok kisebb-nagyobb, rendkívül hasznos tool egy nagy integrált egésszé gyúrva.

### Szolgáltatásai

Az egyik ilyen szolgáltatás a valós idejű adatbázis. Ezzel a szolgáltatással elfelejthetjük a http requesteket a backend felé és helyette websocketeket használhatunk, aminek a gyorsasága konkrétan már csak az internet sebességünktől függ. Az adatbázisban történő változásokról is azonnal értesítést kaphatunk push notification segítségével. Egy hátulütője van csak ennek a szolgáltatásnak, hogy csak nosql adatbázist hozhatunk létre.



A Firebase-ben van egy különálló szolgáltatás csak az autentikációra. A Firebase Autentikáció támogatja a sima email + jelszó párossal a bejelentkezést (OAuth2 segítségével), de ezen felül még támogatja a Facebook, Google, Twitter és GitHub bejelentkezést. A különféle social media platform bejelentkezés támogatásával meg tudjuk növelni a felhasználók számát, mivel sokkal egyszerűbb az átlag felhasználóknak így bejelentkezni.

## 3 Frontend (Vue Js)

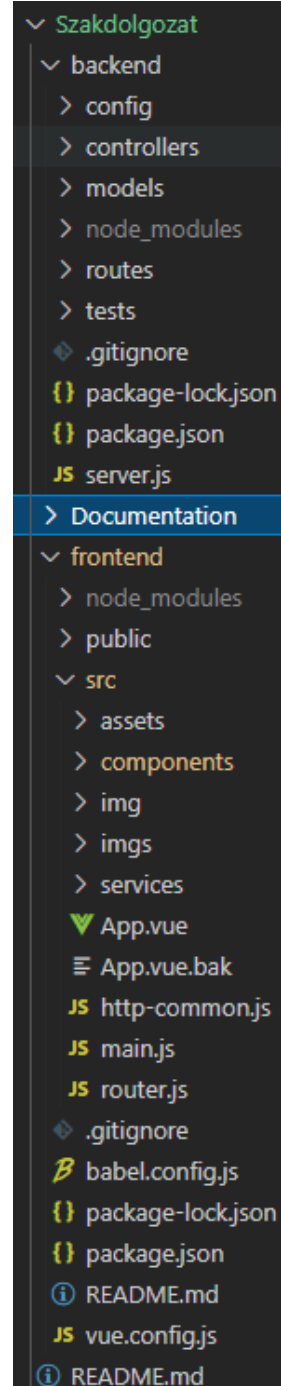
### 3.1. Mappa struktúra

A fő szempont, amit követtünk az, hogy külön komponensekre tudjuk bontani az oldalt és egy olyan mappa struktúrát tudjunk létrehozni, ami átlátható és rendszerezett és könnyen lehessen vele dolgozni.

A projektet először két fő szekcióra bontottuk, ami nem lett más, mint a backend és a frontend mappa. Ezáltal elértük, hogy a munkánk teljesen átlátható legyen. Ezeken belül megtalálhatóak a keretrendszerhez tartozó mappák.

### 3.2. Használt pluginok

A fő szempontunk az volt a project kezdetekor, hogy minimalizálni tudjuk a használt bővítmények, illetve programok használat, hogy ezzel maximalizálni tudjuk az oldal betöltésének gyorsaságát és teljesítményét. Elsősorban a végső kinézethez és stílus megvalósításához a Bootstrap CSS keretrendszert alkalmaztuk, ami kiválóan segített abban, hogy a weboldalon található komponenseken könnyedén meg tudjuk formálni az adott elemet, illetve nem utolsó sorban, a végleges mobilos kinézetben is sok szerepet kapott. Másodszor a felhasználók autentikációját, bejelentkeztetését, és regisztrációját a Firebase nevű szoftverfejlesztő készlet valósította meg. Ezáltal könnyen tudtuk kezelni vagy akár módosítani a felhasználókat.



```

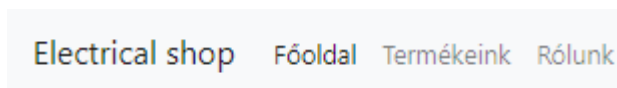
Szakdolgozat
├── backend
│   ├── config
│   ├── controllers
│   ├── models
│   ├── node_modules
│   ├── routes
│   ├── tests
│   ├── .gitignore
│   ├── package-lock.json
│   ├── package.json
│   └── server.js
├── Documentation
└── frontend
    ├── node_modules
    ├── public
    ├── src
    │   ├── assets
    │   ├── components
    │   ├── img
    │   ├── imgs
    │   ├── services
    │   ├── App.vue
    │   ├── App.vue.bak
    │   ├── http-common.js
    │   ├── main.js
    │   ├── router.js
    │   ├── .gitignore
    │   ├── babel.config.js
    │   ├── package-lock.json
    │   ├── package.json
    │   ├── README.md
    │   ├── vue.config.js
    │   └── README.md

```

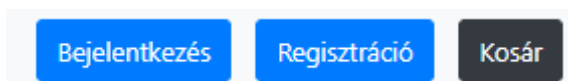
### 3.3. Navigációs bár

A navigációs bár kialakításánál szerettünk volna több lehetőséget beleépíteni a kinézetbe, viszont sajnos nem valósultak meg az ötleteink, ezért egy egyszerűbb, de funkcióját tökéletesen ellátó navigációs bár-t hoztunk létre.

A terület bal szélén található a menü ikonok, illetve az oldal neve, amik a Vue.js segítségével a komponsekkel van összeköttetésben.

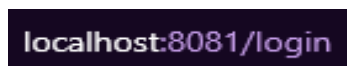


A bár jobb oldalán pedig megtalálható a bejelentkezéshez és regisztrációhoz tartozó gomb, illetve a kosár ikonja.



Az összeköttést a komponsekkel egy külön route script látja el, amiben megtalálható minden útvonal, ami segít a pontos elérésben. Minden útvonalhoz megtalálható az url sávban az egyedi elérés.

Ezzel könnyedén meg tudtuk oldani frontend a könnyű elérését oldalak között. Nem utolsó sorban könnyen lehet hozzá adni új útvonalat, illetve törölni is, ha szükség lenne rá.



```
{
  path: "/products",
  alias: "/products",
  name: "products",
  component: () => import("../components/ProductsList")
},
[
  {
    path: "/products/:id",
    name: "product-details",
    component: () => import("../components/Product")
  },
  {
    path: "/add",
    name: "add",
    component: () => import("../components/AddProduct")
  },
  {
    path: "/userproducts",
    name: "user-products",
    component: () => import("../components/UserProducts")
  },
  {
    path: "/login",
    name: "login",
    component: () => import("../components/Login")
  },
  {
    path: "/register",
    name: "register",
    component: () => import("../components/Register")
  },
  {
    path: "/",
    alias: "/home",
    name: "home",
    component: () => import("../components/Home")
  },
]
```

A panel el lett látva egy olyan biztonsági funkcióval, ami segít abban, hogy külön válassza az átlag user-t az admin felhasználótól. Első körben megírtunk egy függvényt, ami detektálja, hogy ki az aki be van lépve és ki az aki nincs.

```
firebase.auth().onAuthStateChanged(function (user) {
  if (user) {
    isLoggedIn.value = true; // Ha be van jelentkezve a user
  } else {
    isLoggedIn.value = false; // Ha nincs bejelentkezve a user
  }
});
onBeforeMount(() => {
  firebase.auth().onAuthStateChanged((user) => {
    if (!user) {
      // router.replace("/home");
    } else if (route.path == "/login" || route.path == "/register") {
      router.replace("/home");
    }
  });
});
```

A függvény kezdetekor segítségül hívtuk a firebase beépített függvényét, ami megvizsgálja a frontenden bekért és a felhasználó által beírt adatot. Ezután az if utasítás megvizsgálja, hogy a felhasználó sikeresen bejelentkezett vagy regisztrált és az isLoggedIn nevű változóban eltároljuk a megfelelő értéket.

Ezek után egyszerű dolgunk volt hiszen egy változóban tudtuk eltárolni a belépés sikerességét. Visszatérve az oldal elején említett biztonsági funkcióhoz a navigációs bár csakis akkor jeleníti meg módosításhoz vagy épp termék felvételhez szükséges gombokat, ha az isLoggedIn változó értéke igaz.

```
<li class="nav-item">
  <span v-if="isLoggedIn">
    <router-link to="/products" class="nav-link">
      Admin felület</router-link>
    </span>
  </li>
  <span v-if="isLoggedIn">
    <router-link to="/add" class="nav-link">Termék hozzáadása</router-link>
  </span>
  <router-link to="/userProducts" class="nav-link">Termékeink</router-link>
  <router-link to="/about" class="nav-link">Rólunk</router-link>
</ul>
  <span v-if="isLoggedIn">
    <a href="#" class="btn btn-danger" @click="Logout">Kijelentkezés</a>
  </span>
  <span v-else>
    <router-link to="/login" class="btn btn-primary" style="margin-right:15px">Bejelentkezés</router-link>
    <router-link to="/register" class="btn btn-primary">Regisztráció</router-link>
  </span>
```

A program tökéletes működéséhez először egy v-if direktívát kell alkalmazni, ami csak azt a menüpontot jeleníti meg, amelyik a span tag között található.

Electrical shop   Főoldal   Admin felület   Termék hozzáadása   Termékeink   Rólunk

Amint a felhasználó bejelentkezett a két gomb a bejelentkezés és a regisztráció eltűnik és helyette megjelenik a kijelentkezés lehetőség. Ez is pontosan megegyezik az előzőleg említett megoldással.

Kijelentkezés

Kosár

```
<span v-if="isLoggedIn">
  <a href="#" class="btn btn-danger" @click="Logout">Kijelentkezés</a>
</span>
<span v-else>
```

A felhasználó-ból való kiléptetéshez egy külön függvényt írtunk, aminek a lényege, hogy a gombra való rányomáskor történik egy click esemény, aminek a fő funkciója, hogy a user ki tudjon lépni biztonságosan a fiókjából.

```
const Logout = () => {
  firebase
    .auth()
    .signOut()
    .then(() => console.log("Signed out"))
    .catch((err) => alert(err.message));
};
```

### 3.4. Bejelentkezés és Regisztráció

Alapvetően sokat gondolkodtunk, hogy mivel is lehetne megvalósítani a felhasználók sikeres regisztrációját, illetve tárolását. Először az első fő ötletünk az adatbázisban való mentés volt, viszont a kivitelezését nem tudtuk megoldani ezért választottuk a Google Firebase-t, mivel sokkal egyszerűbben és átláthatóbban tudtuk beintegrálni a programunkba a felhasználók autentikációját.

Search by email address, phone number, or user UID					Add user	↺	⋮
Identifier	Providers	Created ↓	Signed In	User UID			
gamertomi1024@gmail.com	✉	Apr 27, 2022	Apr 27, 2022	wwA1WqZ8guQg1crw0otTwrMcfr...			
kovacs.bence.dominik@st...	✉	Apr 25, 2022	Apr 26, 2022	s6DBT9puZVWQoozcC1N0rx5REe...			
tomcsi009@gmail.com	✉	Apr 25, 2022	Apr 27, 2022	1uyjyqSH8mMeOlyGBrdMuEdDZej1			
Rows per page: 50					1 - 3 of 3	⏪	⏩

A fenti képen látható, hogy minden az oldalon regisztrált felhasználót el tudunk menteni, illetve tárolhatjuk a hozzá tartozó információkat.

Mostani állapotában a programnak, csak email és jelszóval való regisztráció működik. Későbbiekben szeretnénk ezen fejleszteni, mégpedig különböző szociális oldalak használatával, mint például (Facebook, Instagram). A Firebase erre is lehetőséget biztosít számunkra.

#### SDK setup and configuration

☐ npm  ☐ CDN  ☒ Config 

Firestore configuration object containing keys and identifiers for your app:

```
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "AIzaSyBic4YmGsK10bVKwvA9zfCDSeGw4KOM1I",
  authDomain: "frontend-75644.firebaseio.com",
  projectId: "frontend-75644",
  storageBucket: "frontend-75644.appspot.com",
  messagingSenderId: "197108772078",
  appId: "1:197108772078:web:d251fb3d22bd55402257ff",
  measurementId: "G-1BP29RSHR1"
};
```





### 3.4.1. Beépítés a programba

Ez a lépés után el kellett készíteni, azt a két függvényt, ami feldolgozza és elküldi a Firebase adatbázisunknak a kért bejelentkezést vagy regisztrációt. Megírása nem volt bonyolult mert a telepített plugin rendelkezésünkre bocsájt rengeteg beépített függvényt. Ami segítségével könnyedén tudjuk feldolgozni a frontenden található input mezőből található adatokat.

```
<script>
import { ref } from "vue";
import firebase from "firebase";
export default {
  setup() {
    const email = ref("");
    const password = ref("");

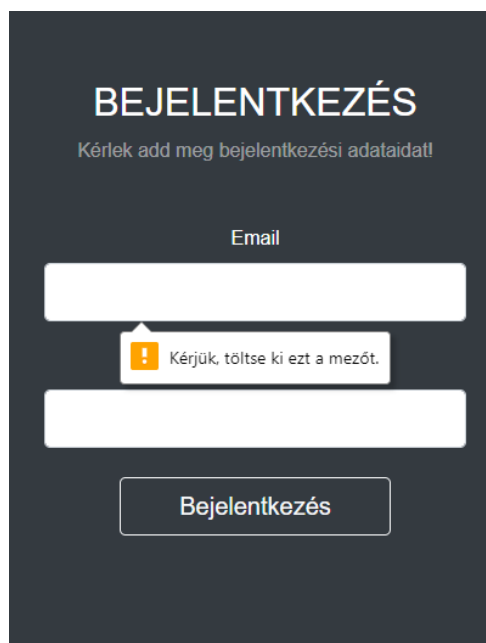
    const Login = () => {
      firebase
        .auth()
        .signInWithEmailAndPassword(email.value, password.value)
        .then((data) => console.log(data))
        .catch((err) => alert(err.message));
    };
    return {
      Login,
      email,
      password,
    };
  },
};
</script>
```

```
<script>
import firebase from 'firebase';
import { ref } from 'vue';

export default {
  setup() {
    const email = ref("");
    const password = ref("");

    const Register = () => {
      firebase
        .auth()
        .createUserWithEmailAndPassword(email.value, password.value)
        .then(user => {
          alert(user);
        })
        .catch(err => alert(err.message));
    };
    return {
      Register,
      email,
      password
    };
  }
};
</script>
```

A két képen látható kódrészletben található két olyan metódus, ami segít nekünk az input mezőből kiszedni az adatokat, illetve létrehozni. Először definiáltuk a kettő fő változónkat, amiben lett eltárolva az aktuális érték, ezek az email és a password változók. Ezt követően már csak használnunk kellett őket. A program dob egy hibaüzenetet, ha a felhasználó rossz adatot ad meg, mint például, ha nem tölti ki a mezőket, vagy helytelenül adja meg a jelszavát.



A frontenden létrehoztuk a megfelelő felületet ahhoz, hogy a user kényelmesen és gyorsan tudja elvégezni a kívánt kérést.



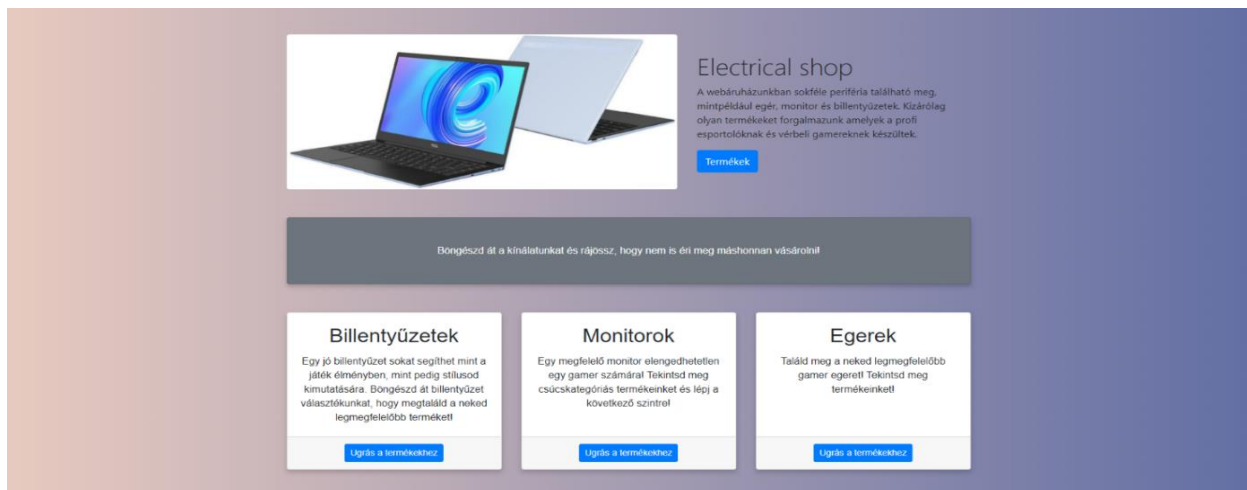
```
<div>
  <section class="vh-100 gradient-custom">
    <div class="container py-5 h-100">
      <div class="row d-flex justify-content-center align-items-center h-100">
        <div class="col-12 col-md-8 col-lg-6 col-xl-5">
          <div class="card bg-dark text-white" style="border-radius: 1rem;">
            <div class="card-body p-5 text-center">
              <div class="mb-md-5 mt-md-4 pb-5">
                <h2 class="fw-bold mb-2 text-uppercase">Bejelentkezés</h2>
                <p class="text-white-50 mb-5">Kérlek add meg bejelentkezési adataidat!</p>
                <form @submit.prevent="Login">
                  <div class="form-outline form-white mb-4">
                    <label class="form-label" for="typeEmailX">Email</label>
                    <input type="text" id="typeEmailX" class="form-control form-control-lg" v-model="email" required />
                  </div>
                  <div class="form-outline form-white mb-4">
                    <label class="form-label" for="typePasswordX">Jelszó</label>
                    <input type="password" id="typePasswordX" class="form-control form-control-lg" v-model="password" required />
                  </div>
                  <button class="btn btn-outline-light btn-lg px-5" type="submit">Bejelentkezés</button>
                </form>
                <div class="d-flex justify-content-center text-center mt-4 pt-1">
                  <p class="mb-0">Nincs még fiókod? <a class="text-white-50 fw-bold" <router-link to="/register">Hozz létre egy újat</router-link></a>
                </p>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
```

HTML kódrészlet a Login ablakról

### 3.5 Komponentensek

### 3.5.1. Főoldal

Azért választottuk a Vue Js, mert úgy gondoltuk, hogy ebben a keretrendszerben tudjuk átláthatóan és rendszerezve tartani a weboldalon található oldalainkat. Minden külön oldalt és menüpontot különválasztottunk, hogy egyszerűbb legyen a használata. Elsőként a Főoldalra térnék ki, ahol is, egy letisztult, információkkal ellátott dizájnt kapunk. Több szövegsor is utal a webshop témájára, hogy mit is értékesít a cég ezek mellett pár kép helyezkedik el illusztrációként.



Az oldal alján egy grid box található, amiben több kártya helyezkedik el, amin a weboldalon árusított termékek fajtáit ismerhetjük meg.

```
<div class="container px-4 px-lg-5">
<!-- Heading Row-->
<div class="row gx-4 gx-lg-5 align-items-center my-5">
    <div class="col-lg-7"></div>
    <div class="col-lg-5">
        <h1 class="font-weight-light">Electrical shop </h1>
        <p>A webáruházunkban sokféle periféria található meg, mint például egér, monitor és billentyűzetek. Kizárólag olyan termékeket forgalmazunk amelyek a profi esportolóknak és vérbél
    </div>
</div><!-- Call to Action-->
<div class="card text-white bg-secondary my-5 py-4 text-center">
    <div class="card-body">
        <p class="text-white m-0">Böngészd át a kínálatunkat és rájössz, hogy nem is éri meg máshonnan vásárolni!</p>
    </div>
</div><!-- Content Row-->
<div class="row gx-4 gx-lg-5">
    <div class="col-md-4 mb-5">
        <div class="card h-100">
            <div class="card-body">
                <h2 class="card-title">Billentyűzetek</h2>
                <p class="card-text">Egy jó billentyűzet sokat segíthet mint a játék élményben, mint pedig stílusod kimutatására. Böngészd át billentyűzet választékunkat, hogy megtaláld
                <div class="card-footer"><a class="btn btn-primary btn-sm" href="#!">Ugrás a termékekhez</a></div>
            </div>
        </div>
    </div>
    <div class="col-md-4 mb-5">
        <div class="card h-100">
            <div class="card-body">
                <h2 class="card-title">Monitorok</h2>
                <p class="card-text">Egy megfelelő monitor elengedhetetlen egy gamer számára Tekints meg csúcskategóriás termékeinket és lépj a következő szintre!</p>
                <div class="card-footer"><a class="btn btn-primary btn-sm" href="#!">Ugrás a termékekhez</a></div>
            </div>
        </div>
    </div>
    <div class="col-md-4 mb-5">
        <div class="card h-100">
            <div class="card-body">
```

### 3.5.2. Admin felület

Törekedtünk arra az admin felületnél, hogy minél egyértelműbb legyen a használata a felhasználónak. Megtalálható az oldalon egy lista, ami azért felel, hogy az adatbázisba felvett és eltárolt termékeket a nevük szerint megkeresse és a termék listában megjelenítse.

Keresés

A terméklista tartalmazza az adatbázisban tárolt termékeket és a hozzájuk tartozó tulajdonságokat. Mindemellett az egyik eszközre rákattintva megjelenik a szerkesztés gomb, ahol több adatot tudunk változtatni, mint például a képet, a termék árát és leírását.

Termék lista

Kérem válasszon ki egy terméket

SteelSeries Rival 650 Wireless Egér

Logitech G Pro Wireless Egér

SteelSeries Prime Mini Wireless Egér

SteelSeries Apex 7 TKL

HyperX Alloy FPS

SteelSeries Apex Pro

Zowie Benq XL2411

ASUS 24,5" FULLHD VG258Q

ASUS ROG SWIFT PG278QE 27"

Összes törlése

Termék lista

Termék

SteelSeries Rival 650 Wireless Egér

Logitech G Pro Wireless Egér

SteelSeries Prime Mini Wireless Egér

SteelSeries Apex 7 TKL

HyperX Alloy FPS

SteelSeries Apex Pro


Zowie Benq XL2411

ASUS 24,5" FULLHD VG258Q

ASUS ROG SWIFT PG278QE 27"

Összes törlése

Termék neve: ASUS 24,5" FULLHD VG258Q



Termék leírás: Az ASUS exkluzív GamePlus forróbillentyű egy játék közbeni bővítést kínál, miközben a GameVisual optimalizálja a látványt

Termék ár: 70.999

Termék státusza: A termék jelenleg nincs raktáron

Szerkesztés

A tulajdonságok mellett megjelenik egy státusz jelző sáv is, ami mutatja nekünk, hogy a termék éppen van-e készlete vagy nincs. Ez termékeink komponensben megjeleni a felhasználó számára. Másrészt lehetőségünk van az adott terméket törölni vagy éppen frissíteni az adatokat.

Termék

Termék megnevezése

ASUS 24,5" FULLHD VG258Q

Kép url

<https://dlcdnmings.asus.com/websites/glo1>

Termék leírása

Az ASUS exkluzív GamePlus forróbillentyű

Termék ára

70.999

Termék státusza: A termék jelenleg nincs készleten

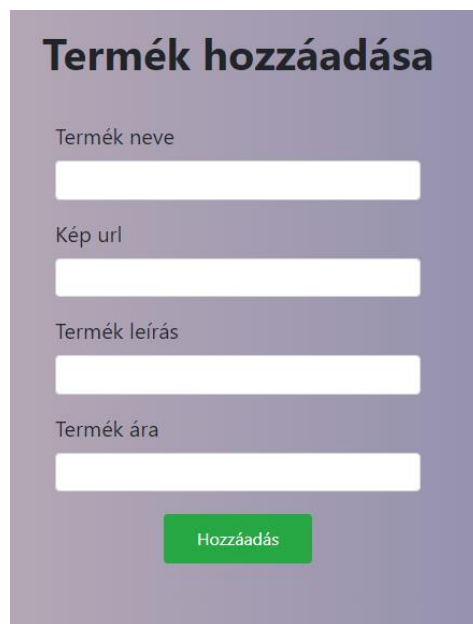
Raktáron

Törlés

Frissítés

### 3.5.3. Termék hozzáadása

A termékek hozzáadása minimális stílust kapott ez miatt tökéletesen értelmezhető, hogy milyen típusú eszközt szeretnénk feltölteni az oldalra. Négy fajta értéket különböztettünk meg. Termék nevét, termékről képet, ami lehetőség szerint PNG formátumú, hogy az oldal stílusához illeszkedjen. Ezt egyelőre kizárólag webes elérési úttal tudtuk megvalósítani, ehhez a keresett kép url címét kell bemásolnunk az input mezőbe. Termék leírása, amiben egy rövid ismertetőt tudunk adni a produktumról és utolsó sorban a termék árát tudjuk beállítani, ahol meg van oldva, hogy az admin csakis kizárólag számot adhat meg.



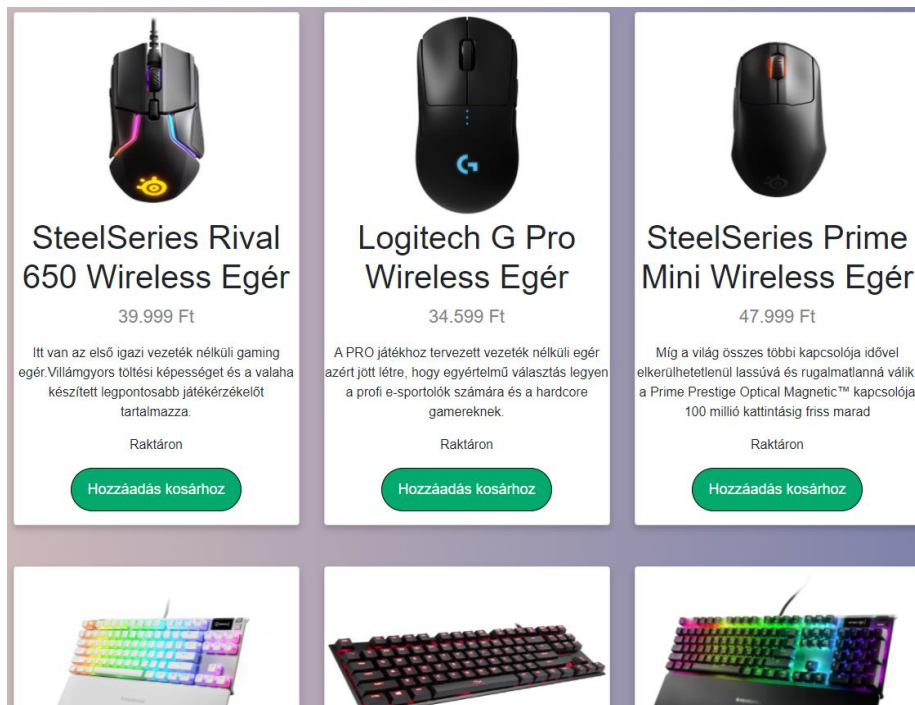
```
<div>
  <h1 class="addProdtitle">Termék hozzáadása</h1>
  <div class="submit-form">
    <div v-if="!submitted">
      <div class="form-group">
        <label for="title">Termék neve</label>
        <input type="text" class="form-control"
          id="title" required v-model="products.title" name="title"/>
      </div>
      <div class="form-group">
        <label for="title">Kép url</label>
        <input type="text" class="form-control"
          id="img" required v-model="products.img" name="img"/>
      </div>
      <div class="form-group">
        <label for="description">Termék leírás</label>
        <input type="text" class="form-control" id="description" required v-model="products.description" name="description"/>
      </div>
      <div class="form-group">
        <label for="price">Termék ára</label>
        <input type="number" name="price" id="price" class="form-control" required v-model="products.price">
      </div>
      <div class="d-flex justify-content-center">
        <button @click="saveProduct" class="btn btn-success" id="submitbtn">Hozzáadás</button>
      </div>
    </div>
    <div v-else>
      <h4>A termék sikeresen felkerült a weboldalra!</h4>
      <button class="btn btn-success" @click="newProduct">Tovább</button>
    </div>
  </div>
</div>
```

### 3.5.4. Termékeink

Termékek megjelenítésél egy Bootstrap segítségével készített grid box-ban találhatóak kártyák három oszlopba és sorba rendezve. A kártyák a hozzáadást követően alkalmazkodnak a hozzáadott képek méretéhez, amivel, megakadályoztuk, hogy a card-ok elveszítsék méretüket.

```
.card img{  
  max-height: 450px;  
  max-width: 350px;  
  height: 250px;  
}
```

Ezen felül a termékek automatikusan a felvételkor bekerülnek az oldalra.



A frontend az axios segítségével dolgozza fel a backend api hívásait, a ProductsDataService.js segítségével.

A v-for direktívát alkalmazza végig megyünk a frontenden feldolgozott tömbünkön amely a productokat tartalmazza és ezzel megjelenítjük az adatbázisban lévő termékeinket és azok tulajdonságait egy bootstrap card stílusban.

Ebben a kártyában még található egy "Hozzáadás kosárhoz" felíratú gomb, amelynek a @click esemény kezelője meghívja az AddToCart() függvényt a methods szekcióból amelynek segítségével a terméket a kosárba tudjuk helyezni.

```

<div>
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/ionicons/2.0.1/css/ionicons.min.css">
<div class="container">
  <div class="row row-cols-1 row-cols-md-3 g-4" id="carddesign">
    <div class="col" v-for="product in products" :key="product._id">
      <div class="card">
        
        <h1>{{ product.title }}</h1>
        <p class="price">{{ product.price }} Ft</p>
        <p>{{ product.description }}</p>
        <p>
          {{ product.published ? "Raktáron" : "A termék jelenleg nincs raktáron!" }}
        </p>
        <p><button @click="AddToCart(product)">Hozzáadás kosárhoz</button></p>
      </div>
    </div>
  </div>
</div>
</div>

```

A kosár fejlesztését nagyon a végére hagytuk, ezért sajnos csak minimális alap funkciókat sikerült beletennünk. A ClearCart() függvény teljes egészében törli a kosarunk tartalmát, valamint a RemoveFromCart() függvény eltávolít egy darab terméket.

A "Megrendelés" gomb pedig egy értesítést küld a felhasználónak, hogy megrendelése sikeresen sikerült, valamint teljes egészében kiűríti a kosár tartalmát.

Kosarad

Kosár törlése

Zowie Benq XL2411	72.999 Ft	Eltávolítás
ASUS 24.5" FULLHD VG258Q	70.999 Ft	Eltávolítás
ASUS ROG SWIFT PG278QE 27"	255.999 Ft	Eltávolítás
HyperX Alloy FPS	23.999 Ft	Eltávolítás

Megrendelés

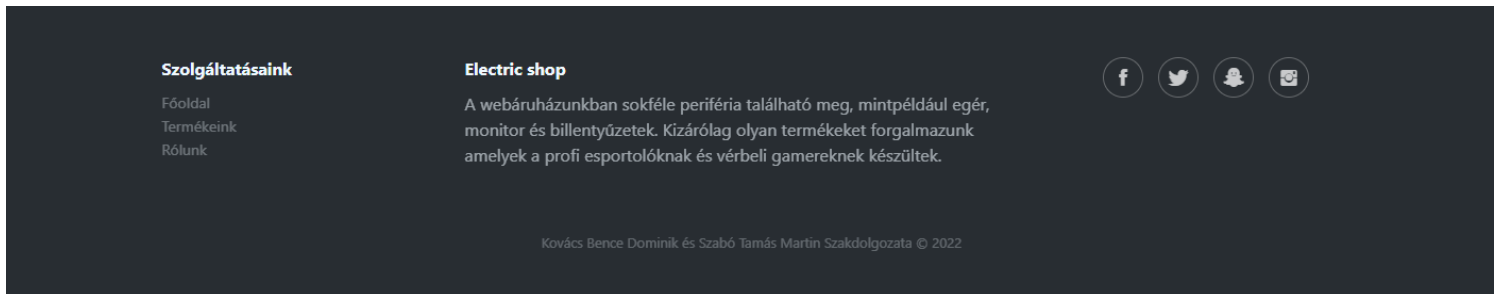
```

<div class="col">
  <h2>Kosarad</h2>
  <table class="table table-borderless">
    <thead>
      <tr>
        <th scope="col"><button class="btn btn-danger" @click="ClearCart(product)">Kosár törlése</button></th>
        <th scope="col"></th>
        <th scope="col"></th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="item in cart" :key="item._id">
        <td>{{item.title}}</td>
        <td>{{item.price}} Ft</td>
        <td><button class="btn btn-danger" @click="RemoveFromCart(index)">Eltávolítás</button></td>
      </tr>
    </tbody>
  </table>
  <button @click="Pay()" class="btn btn-success">Megrendelés</button>
</div>

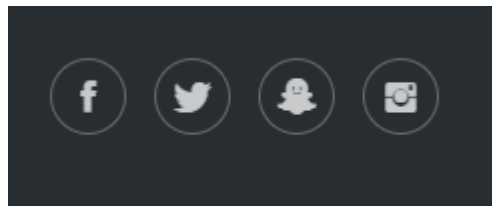
```

### 3.5.6. Footer

Az oldalak alján megtalálható egy kis információs panel, aminek jelentősége, hogy a page aljára görgetve is könnyedén tudjunk váltani a menüpontok között. Ezek a panelek kizárólag olyan oldalra kerültek, ahol egy bemutatkozás található.



Található a footer-ben ezen kívül egy social pagek-re hivatkozó gombsor, amihez hozzá tudjuk csatolni az oldalhoz vagy céghez tartozó közösségi oldalakat.



Ehhez tartozó iconokat, a bootstrap beimportált css csomagjából használtuk fel.

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/ionicons/2.0.1/css/ionicons.min.css">
```



## 3.6. Reszponzív dizájn

### 3.6.1 Mobiltelefon használat

Az emberek a mai időkben sokkal több időt töltenek a telefon használatával, mint a számítógép vagy laptop előtt ülve. Ez mind a weboldalkészítés, mint a webshopok üzemeltetőinek nagy előnyt tud jelenteni, hiszen bárhol és bármikor képesek vagyunk vele elérni az internetet és

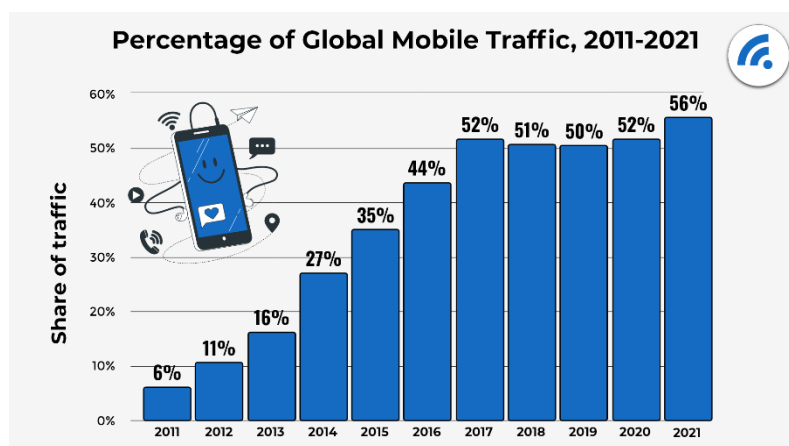
ezáltal rákeressünk valamire. Jobb oldalt látható egy ábra, ami nagyon szépen megmutatja nekünk azt, hogy bármilyen interneteléréssel bíró eszköz közül a mobiltelefont használják többen közel 4%-kal. Az oldal alján elhelyezett ábrán pedig láthatjuk, hogy mennyit nőtt százalékos arányban a telefon használata 2011 és 2021 között.

Számunkra ez azért fontos mivel, egy weboldalt már nem csak a PC-s nézetben kell elkészítenünk, hanem a mobilos vagy akár tabletes nézetben. Mobilos nézetre fókuszáltunk, hogy tökéletesen legyen és semmi ne csússzon szét mikor telefonos nézetben vagyunk. Ehhez a Bootstrap kiváló választás volt, hiszen rendelkezik egy előre megírt eszközkészlettel, amivel sokkal rugalmasabban és gördülékenyebben fejleszthetjük az oldalunkat. Reszponzív felület fejlesztésénél nagy előnyünkre vált, hogy a keretrendszer folyékony részponzív keretrendszert használ, aminek alapvetően három töréspontja van 768 px, 992 px és 1200 px. Weboldalunkon minden oldal megtartja a formáját és igazodik a felbontás eltéréseihez. Ezért mobil nézetben is kiválóan elvégezhetjük a teendőinket az oldalon.

Desktop vs Mobile Internet Usage Statistics in 2020 (Global)

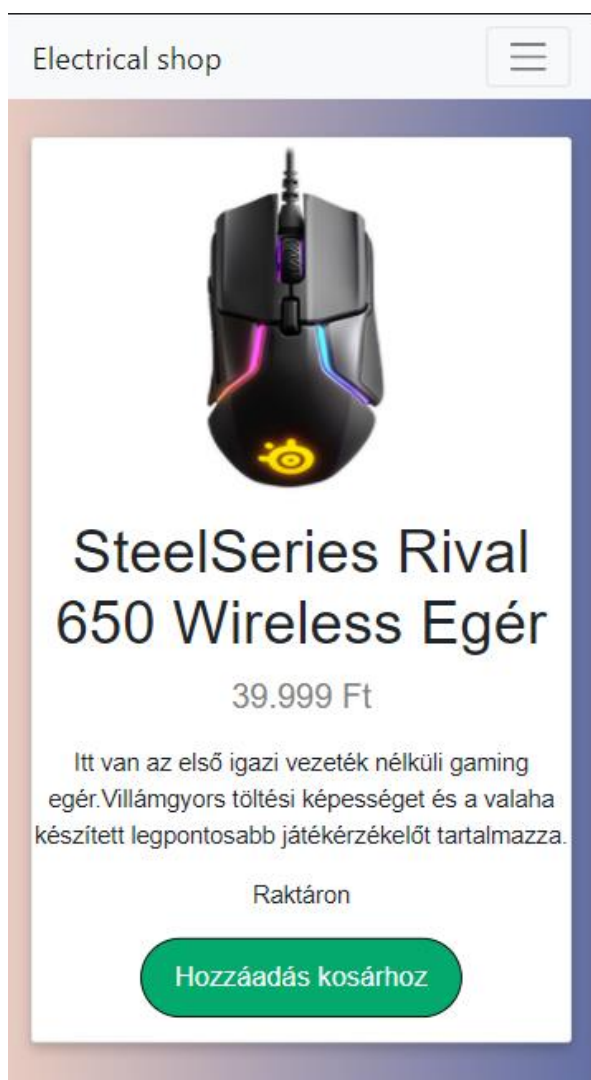


HighSpeedInternet.net



### 3.6.2 Oldalunk reszponzivitása

A navigációs bár telefonos nézetben egy lenyíló menüvé alakul, amiben megtalálhatóak a menüpontok egymás alatt. Elérése egyszerű mivel, ún. „burger icon” segítségével a rányomásakor lenyílik a menü és mikor csukott állapotban van akkor nem zavaró, mivel nem takar ki semmit az oldalunkból. A jobb oldali ábrán látható a kinyitott menü, a lenti ábrán pedig a csukott, illetve, a kártyák elrendezésének megjelenése, ahol lefele görgetve tudunk böngészni a termékek között.



### 3.6.3 Képek az oldalról

Electrical shop

## Termék

Termék megnevezése

SteelSeries Prime Mini Wireless Egér

Kép url

<https://media.steelseriescdn.com/thumbs>

Termék leírása

Míg a világ összes többi kapcsolója időve

Termék ára

47.999

**Termék státusza:** Raktáron

Nincs raktáron Törlés Frissítés


Electrical shop

## Termék lista

SteelSeries Rival 650 Wireless Egér
Logitech G Pro Wireless Egér
SteelSeries Prime Mini Wireless Egér
SteelSeries Apex 7 TKL
HyperX Alloy FPS
<b>SteelSeries Apex Pro</b>
Zowie Benq XL2411
ASUS 24,5" FULLHD VG258Q

## Termék

**Termék neve:** SteelSeries Apex Pro



**Termék leírás:** Első osztályú állítható mechanikus kapcsolók a testreszabható gombonkénti érzékenységhez

**Termék ár:** 81.999

**Termék státusza:** Raktáron

Szerkesztés

## 4. Backend (Node.js, Mongodb)

### 4.1 Models/Schema

A models mappában található a products Schema melyben az adatokat reprezentáljuk. Mongoose-ban minden a Schema-val kezdődik. Minden Schema egy MongoDB kollekcióra mutat, amelyben a dokumentumok találhatóak.

Egy Mongoose Schema-t a mongoose.Schema paranccsal tudunk létrehozni amelyben megadjuk a megadni kívánt propertyket és azok tulajdonságait például típusukat (Number, String, Date stb) vagy akár még specifikusabb tulajdonságokat is adhatunk nekik, például required, minLength, maxLength.

```
module.exports = mongoose => {  
  const Products = mongoose.model(  
    "products",  
    mongoose.Schema(  
      {  
        title: String,  
        description: String,  
        price: Number,  
        published: Boolean,  
        img: String  
      },  
      {timestamps: true}  
    )  
  );  
  return Products;  
}
```

### 4.2 Controllers

A controllers mappában tároljuk a controllereket. A controllerekben logika kódokat tárolunk. Elemezzük ki az exports.create kódrészletet. Az arrow függvény segítségével egy tisztább függvényt tudunk létrehozni, ez a függvény egy kérést(**req**) és egy választ(**res**) vár.

Egy If elágazással megvizsgáljuk, hogy a req.body.title-ben létezik-e adat. Ha nem, akkor egy üzenettel jelezzük, hogy a "Product nem lehet üres". Egy új terméket a const product segítségével tudunk létrehozni, ahol egy objektumban megadjuk a propertyket.

Ha ez sikerült, akkor a mongoose beépített .save() metódusát hívjuk segítségül amivel eltároljuk és

```
exports.create=(req,res)=>{  
  //Validate request  
  if (!req.body.title) {  
    res.status(400).send({message:"Product can not be empty!"});  
    return;  
  }  
  // Create a product  
  const product = new Products({  
    title: req.body.title,  
    description: req.body.description,  
    price: req.body.price,  
    img: req.body.img,  
    published: req.body.published ? req.body.published : false  
  });  
  // Save product in the database  
  product  
    .save(product)  
    .then(data=>{  
      res.send(data);  
    })  
    .catch(err =>{  
      res.status(500).send({  
        message:  
          err.message || "Some error occurred while creating new product."  
      });  
    });  
};
```

létre hozzuk productot, documentumot az adatbázisunkban létrehozott kollekcióba.

A `save()` egy aszinkron típusu metódus. Ha valami hiba lépne fel, akkor a `catch` ágban létrehozott hibaüzenetet küldjük vissza. Az `exports.findAll`-al az összes terméket keressük meg, itt a `mongoose` által beépített `.find()` metódust vettük segítségül. Ahol egyes termékek tulajdonságait akarjuk lekérni vagy megváltoztatni vagy törölni, ott a `.findOne()`, `.findByIdAndUpdate()` illetve a `.findByIdAndRemove()` metódusokat használtuk.

## 4.3 Routes

A routes mappában található `products.routes.js` fájlban adjuk meg a kéréseket és hogy ezen kérések melyik útvonalon történjenek valamint a hozzájuk tartozó controllert adjuk meg.

A kérések különbözőek lehetnek. a GET kérés egy dokumentum fogadására használatos. A HTML, a képek, a JavaScript, CSS stb dokumentumok fogadásának ez a fő módszere. A legtöbb adat, ami a böngészőbe betöltődik ennek a módszernek köszönhető.

A POST kérés adatot küld a szervernek, ez lehet például egy html űrlap. A POST kérés adatait az üzenettest tartalmazza.

A PUT feltölti a megadott adatokat, a DELETE pedig törli azokat.

```
const { products } = require("../models");

module.exports = app =>{
  const Products = require("../controllers/products.controller");
  var router = require("express").Router();
  //Create new product
  router.post("/", Products.create);
  // Retrieve all products
  router.get("/", Products.findAll);
  //Retrieve a single product with id
  router.get("/:id", Products.findOne);
  //update a product with id
  router.put("/:id", Products.update);
  //Delete a product with id
  router.delete("/:id", Products.delete);
  //Delete all products
  router.delete("/", Products.deleteAll);
  app.use('/api/products',router);
};
```

## 4.4 Server.js, db.config.js és Adatbázis

A db.config.js tartalmazza a clusterünk elérési url-jét, a server.js-ben betöltjük és az app.use() metódussal használatba vesszük a használatos csomagjainkat melyek a cors és az express. Ezenkívül itt csatlakozunk az adatbázisunkhoz.

A tests mappában található a Postman-ben végzett backend api json fájlja, a tesztekéről a következő pontban fogunk írni.

Az adatbázisban a products nevű kollekcióban tároltuk el a termékeknek a tulajdonságait, illetve a létrehozásának a pontos dátumát és a produktum frissítésének dátumát, ha ez sorra kerül. A termék státuszának értékét egy boolean változóban tároltuk el, ahol egy igaz vagy hamis logikai értéket kap, aminek a változása befolyásolja a termék raktáron levő értékét.

```
const express = require("express");
const cors = require("cors");
const app = express();

var corsOptions = {
  origin: "http://localhost:8081"
};

app.use(cors(corsOptions));
app.use(express.json());
app.use(express.urlencoded({extended: true}));

const db = require("../models");
db.mongoose
  .connect(db.url, {
    useNewUrlParser: true,
    useUnifiedTopology: true
  })
  .then(() => {
    console.log("Connected to the database!");
  })
  .catch(err => {
    console.log("Cannot connect to the database!", err);
    process.exit();
  });

app.get("/", (req, res) => {
  res.json({ message: "Welcome to the app" });
});

require("../routes/products.routes")(app);

const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on ${PORT}.`);
});
```

ADD DATA	VIEW			
<pre>{   "_id": "ObjectId('626ae0b51aa47894a450ed07')",   "title": "SteelSeries Rival 650 Wireless Egér",   "description": "Itt van az első igazi vezeték nélküli gaming egér.Villámgyors töltési ...",   "price": 39.999,   "published": true,   "img": "https://media.steelseriescdn.com/thumbs/catalogue/products/00994-rival...",   "createdAt": "2022-04-28T18:45:09.333+00:00",   "updatedAt": "2022-04-28T20:22:23.393+00:00",   "__v": 0 }</pre>				
<pre>{   "_id": "ObjectId('626ae14f1aa47894a450ed0a')",   "title": "Logitech G Pro Wireless Egér",   "description": "A PRO játékhöz tervezett vezeték nélküli egér azért jött létre, hogy e...",   "price": 34.599,   "published": true,   "img": "https://resource.logitechg.com/d_transparent.gif/content/dam/gaming/en...",   "createdAt": "2022-04-28T18:47:43.303+00:00",   "updatedAt": "2022-04-28T19:21:43.778+00:00",   "__v": 0 }</pre>				
<pre>{   "_id": "ObjectId('626ae3901aa47894a450ed0d')",   "title": "SteelSeries Prime Mini Wireless Egér",   "description": "Míg a világ összes többi kapcsolója idővel elkerülhetetlenül lassúvá é...",   "price": 47.999,   "published": true,   "img": "https://media.steelseriescdn.com/thumbs/catalog/items/62426/8b1374dcdf...",   "createdAt": "2022-04-28T18:57:20.003+00:00",   "updatedAt": "2022-04-28T19:25:54.994+00:00",   "__v": 0 }</pre>				



## 4.5 Backend tesztek

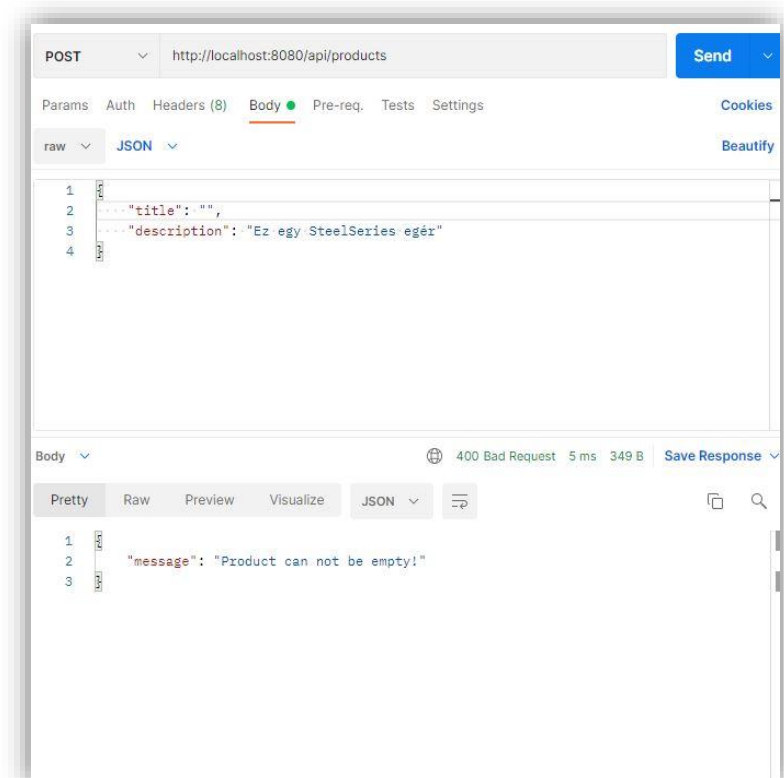
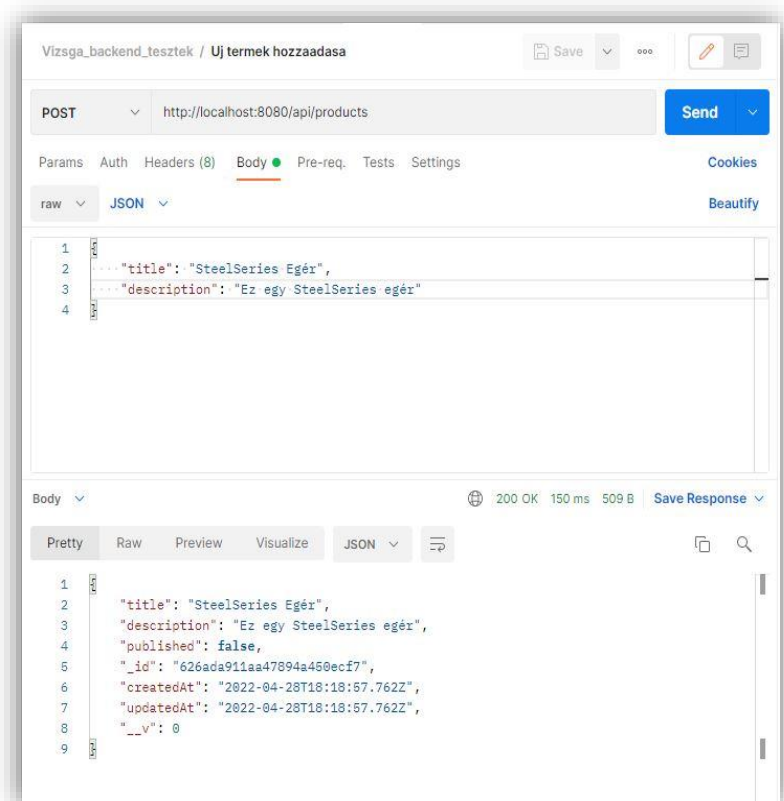
### 4.5.1 Új termék felvétele jó, és rossz ID esetén.

Első tesztünk egy új termék felvétele, hozzáadása volt. Egy POST kéréssel adatot küldünk json fájlban a `http://localhost:8080/api/products` elérési útra. A json fájlban megadtuk az új termék nevét (**title**) illetve leírását (**description**).

A válaszban a várt eredményt kaptuk, az új termék sikeresen felkerült az adatbázisba.

Következő tesztünkben továbbra is egy új termék felvételét teszteltük, azonban itt a termék nevét (**title**) üresen hagytuk.

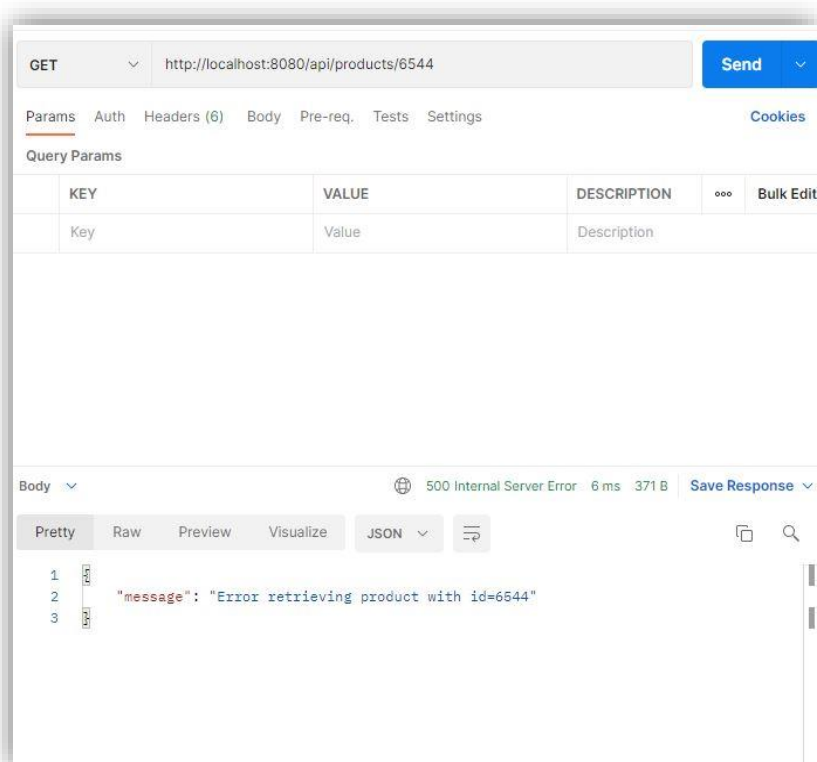
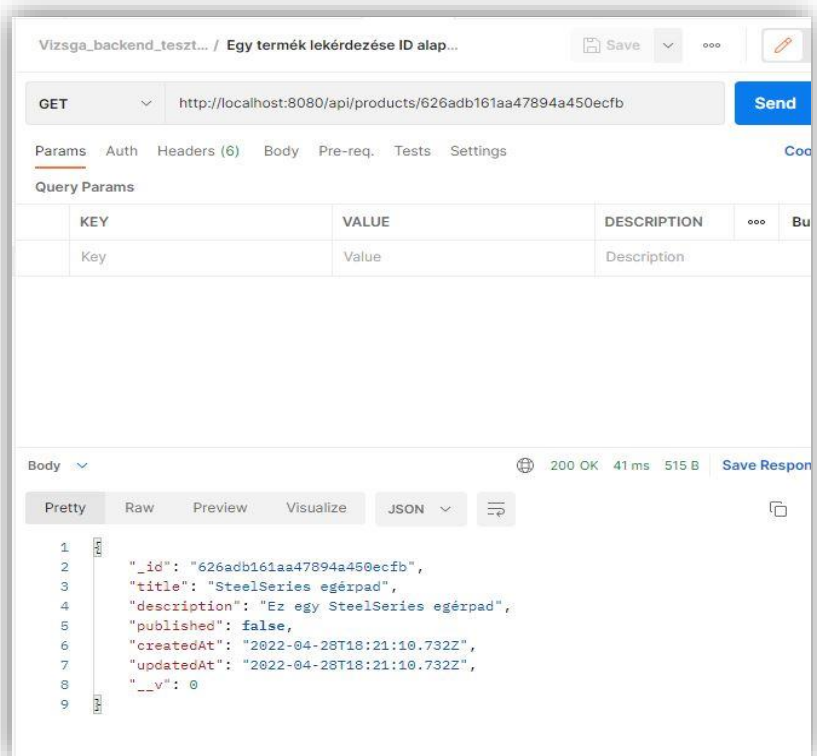
Mint látható a termék nem került fel az adatbázisba és a várt hibaüzenetet kaptuk.



## 4.5.2 Egy termék lekérdezése jó és rossz ID alapján

Ebben az esetben egy termék adatait akartuk lekérdezni. Mint látható egy GET-es kérést küldtünk a `http://localhost:8080/api/products/:id` címre. A válaszban visszakaptuk az adatbázisban létező ID-vel az ID-hez tartozó terméket és annak adatait.

Továbbra is egy termék lekérdezésénél tartunk, viszont itt az url-ben szereplő ID rossz, azaz nem létezik az adatbázisban. A válaszban látható a kapott és várt hibaüzenet.

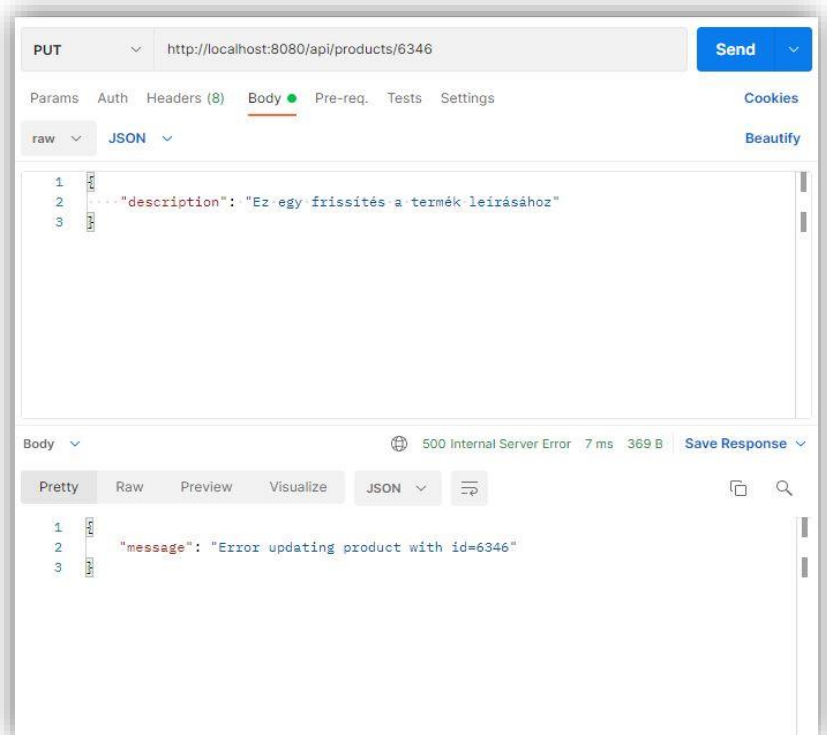
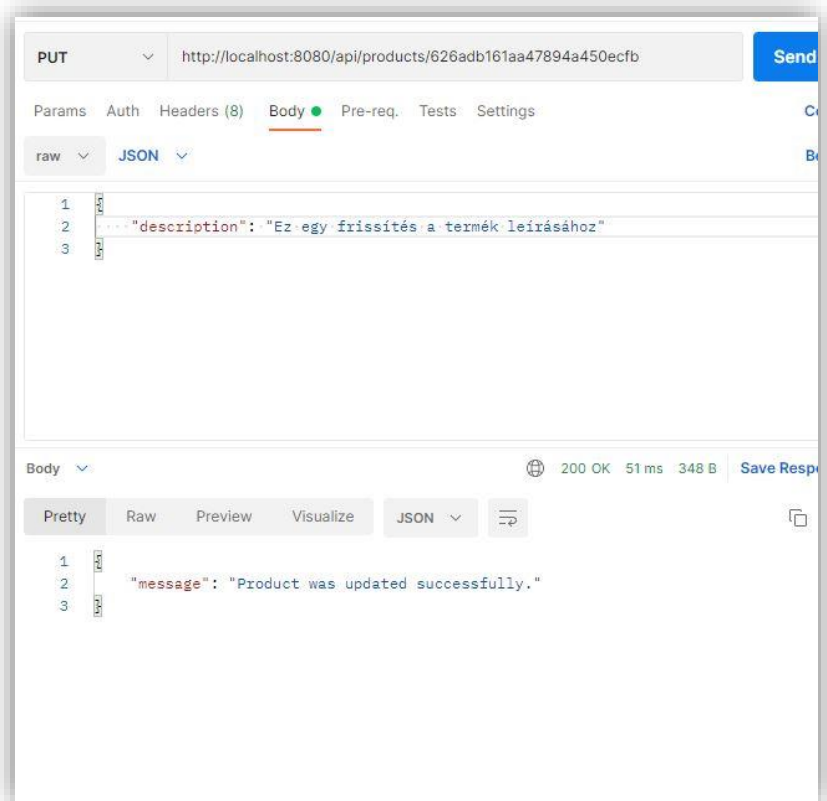




### 4.5.3 Egy termék frissítése ID alapján

Egy termék frissítése esetén a PUT metódust, használjuk, `http://localhost:8080/api/products/:id` url-en. Mint látható, a json-ben a "description" propertyt szerettük volna frissíteni és a válaszban láthatjuk, hogy a frissítés sikeres volt.

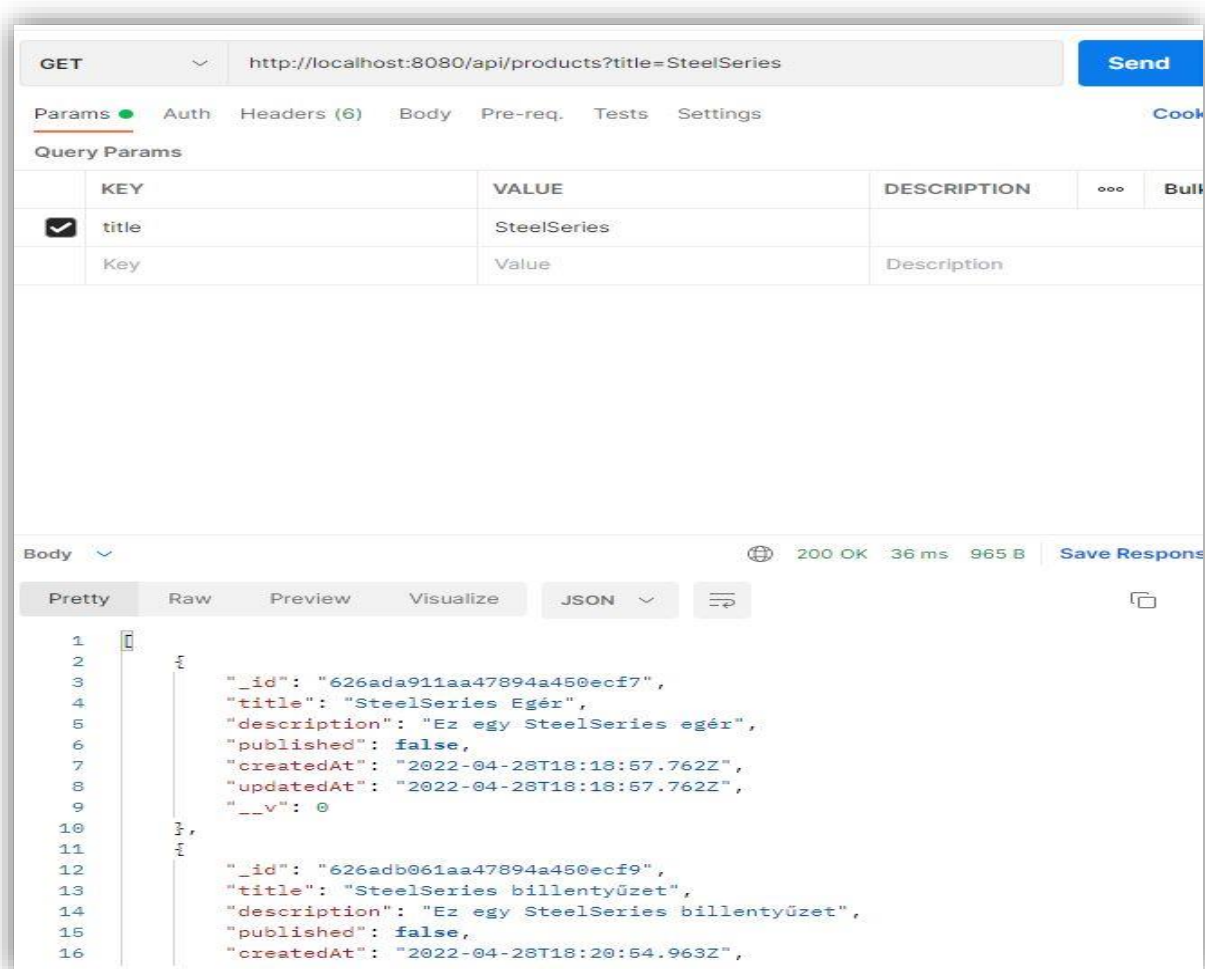
Nézzük meg mi, történik ha egy olyan terméket szeretnénk frissíteni, amely nem szerepel az adatbázisban. Továbbra is a PUT kérést használjuk, a url-ben azonban egy az adatbázisban nem létező ID-t adunk meg. A válaszban megkaptuk, hogy hibába ütköztünk, így nem történt változás az adatbázisban.



## 4.5.4 Termék lekérdezése név(title) alapján

Ismételten egy GET-es kérést küldünk a szervernek, ugyanis azt szeretnénk lekérdezni, hogy van-e egy adott termékük. Az url ebben az esetben a következőképp néz ki: `http://localhost:8080/api/products?title=SteelSeries`.

Mint látható ez a GET kérés is sikeres volt és megkaptuk az összes SteelSeries terméket.

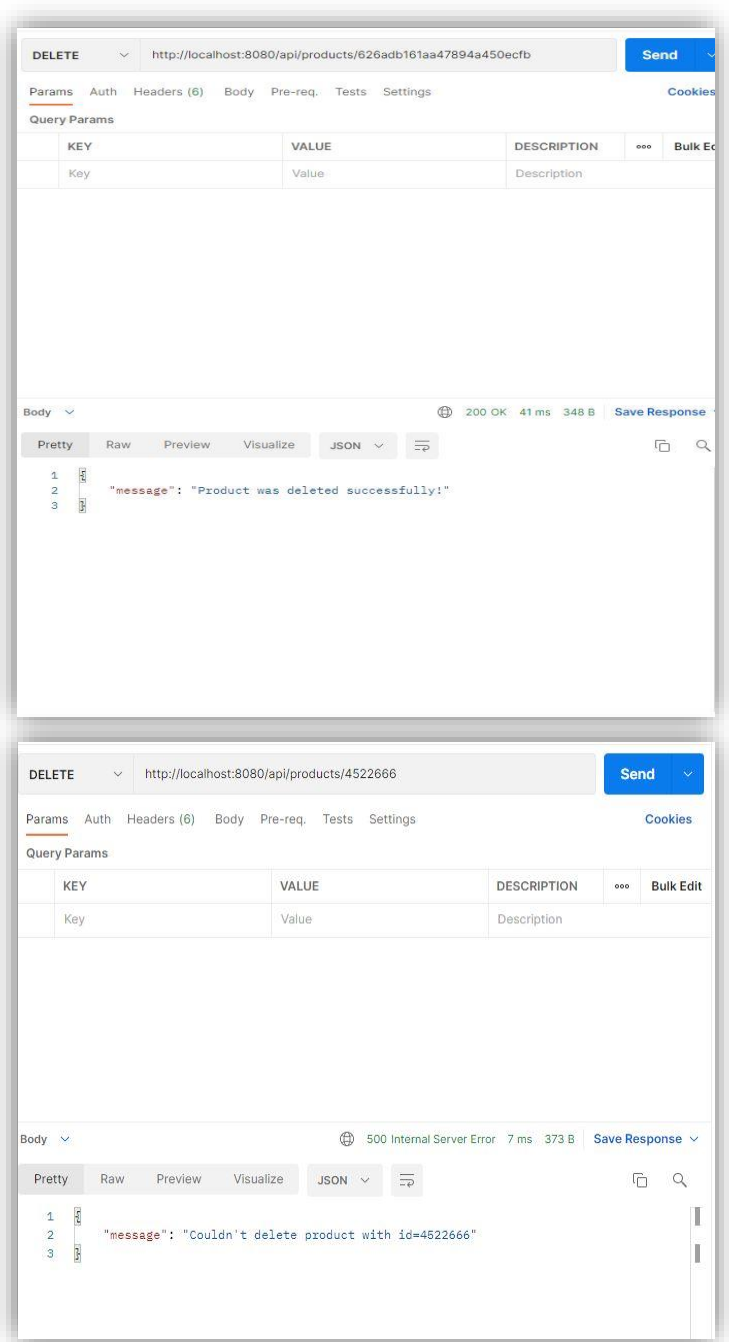


### 4.5.5 Termék törlése ID alapján

DELETE kérést küldtünk a szokásos url-re egy adatbázisban szereplő ID-vel. A válaszban látható, hogy a törlés sikeres volt. DELETE kérést küldtünk a szokásos url-re egy adatbázisban szereplő ID-vel. A válaszban látható, hogy a törlés sikeres volt. Ismét DELETE kérést küldtünk, azonban az url-ben szereplő ID egy nem létező ID az adatbázisban, így mint ahogy a válaszban is látni lehet, nem sikerült semmit sem eltávolítanunk az adatbázisból a megadott ID-vel.

Utolsó tesztünk alatt egy DELETE kérést küldtünk a `http://localhost:8080/api/products` url-re. A válaszban látható, hogy az összes termék, szám szerint 8 dokumentum törlésre sikerült amit a tesztek alatt felvittünk az adatbázisban.

Az összes backend api tesztről található egy json kiterjesztésű fájl, amelyben láthatóak a dokumentációban is elvégzett tesztek.



## 6. Összegzés

Összességében egy olyan webshopot szerettünk volna létrehozni, olyan termékekkel, amiket napi szinten használunk és közel állnak hozzánk. A kivitelezéstől a fejlesztési fázisig izgalmas, de ezzel szemben egy elég nehéz munkán vagyunk túl. Rengeteg kutató munka, illetve kód értelmezés és gondolkodás után elért egy bizonyos állapotot a szakdolgozatunk, amin tudjuk, hogy rengeteg fejleszteni való van, de mi büszkék vagyunk rá, mert nagyon sok olyan pillanat volt, amikor elakadtunk egy adott résznél és nem tudtunk tovább haladni ilyenkor a másik segítsége oldotta meg a problémát. A feladatok megosztása az elején picit zavart keltett bennünk, hiszen nem tudtuk, hogyan is kellene nekilátnunk a munkának és két olyan keretrendszert választottunk, ami mind kettőnknek ismeretlen és új volt az év kezdetén. Az tanév során megismerkedtünk új dolgokkal, mint két framework-kel és amit tanultunk, azokat elemeket beépíteni a projectbe. Úgy érezzük, hogy egy része nagyon jól sikerült másrészt pedig tartalmaz olyan elemeket, amik roppant hiányosak, de ezek a jövőbeli terveink közt vannak, hogy minél előbb kijavítsuk ezeket a hibákat és tanuljunk belőlük.

## 6.2 Jövőbeli tervek

Elsősorban több olyan dolog van, ami javításra szorul és azok az elemek javítva lesznek hamarosan. Nagyon sok fejlesztési ötletünk van mivel lehetne még jobbra tenni a webshopot első sorban egy fejlesztett navigációs bár ami egy szebb és igényesebb designt fog kapni, másrészt a bejelentkezéshez szükség oldalt át tesszük egy felugró ablakra, ami sokkal szebb megjelenést kölcsönöz. A főoldalt fejlettebbé tenni azáltal, hogy beleépítenénk egy bootstrap carousel-t a navigációs bár alá, ahol egy automatikusan megjelenő slideshow-t láthatunk képekkel. Sokkal több terméket szeretnék megjeleníteni és ezzel összekapcsolva a termékekre egy kategorizáló rendszert kiépíteni, hogy sokkal szélesebb körű választék lehessen az oldalon. Ezen felül pedig utolsó sorban a kosár oldalt létrehozni és a minimális működés helyett egy komplexebb termékrendező és összegző felületet megvalósítani.

## 6.1 Felhasznált források

- <https://getbootstrap.com>
- <https://vuejs.org>
- <https://nodejs.org/en/docs/>
- <https://stackoverflow.com>
- <https://codepen.io>
- <https://firebase.google.com>
- <https://github.com>
- <https://www.youtube.com/watch?v=FMPHvxqDrVk>
- <https://www.w3schools.com>