**Debreceni SZC Mechwart András Gépipari és Informatikai Technikum**



Pcbazár

Számítógépes hardver adás-vételi platform

Készítette:

**Hajdú Attila Patrik**

**Kanál Sándor**

Debrecen

2023

[Bevezetés 2](#_Toc1143285095)

[1.1 Téma: PcBazár 2](#_Toc149369486)

[1.2 Témaválasztás indoklása 2](#_Toc625357681)

[1.3 Funkcionalitás 2](#_Toc253151392)

[1.4 Célcsoport 3](#_Toc401562419)

[Fejlesztői dokumentáció 4](#_Toc757699944)

[2.1 Fejlesztői környezet hardverének ismertetése 4](#_Toc455850222)

[2.2 Fejlesztés során használt szoftverek 4](#_Toc34227614)

[2.3 Programnyelv választás indoklása 5](#_Toc656839816)

[2.4 Adatszerkezetek 6](#_Toc460058678)

[2.5 Backend és frontend algoritmusok 13](#_Toc1999215284)

[2.6 Tesztdokumentáció: 19](#_Toc1956179141)

[2.7 Bejelentkezéshez szükséges adatok 21](#_Toc2103313181)

[2.8 Fejlesztési lehetőségek 21](#_Toc1936038617)

[Felhasználói dokumentáció 22](#_Toc916839665)

[3.1 Szükséges eszközök a weboldal használatához 22](#_Toc1311219652)

[3.2 Webalkalmazáshoz szükséges szoftverek 22](#_Toc1114118978)

[3.3 Webalkalmazás elindítása 23](#_Toc1566292100)

[3.4 Bejelentkezéshez szükséges adatok 24](#_Toc946031213)

[3.5 Webalkalmazás használatának az ismertetése 24](#_Toc1209708950)

[3.5.1 Főoldal 24](#_Toc481451577)

[3.5.2 Regisztráció 25](#_Toc877667059)

[3.5.3 Bejelentkezés 27](#_Toc229782668)

[3.5.4 Navigációs sáv 28](#_Toc540848056)

[3.5.5 Hirdetésfeladás 29](#_Toc620301171)

[3.5.5 Kosár funkció 30](#_Toc1082358664)

[3.5.6 Hirdetéseim funkció 31](#_Toc1212233102)

[3.5.7 Módosítás funkció 32](#_Toc2101360670)

[3.5.8 Rendeléseim funkció 33](#_Toc1217719050)

[3.5.9 Részletek funkció 34](#_Toc926234812)

[3.5.10 Keresés funkció 35](#_Toc1220519595)

[3.5.11 Product funkció 36](#_Toc470984351)

[3.5.12 Sidebar (Oldalsáv) 37](#_Toc40104381)

[Összefoglaló 38](#_Toc630092969)

# Bevezetés

## 1.1 Téma: PcBazár

A következő dokumentációban a PcBazár munkanévre hallgató projektünket szeretném bemutatni. A Pcbazár egy tematikus piactér webalkalmazás, amely kizárólag számítógépes alkatrészekre összpontosít, lehetőséget biztosítva a felhasználók számára, hogy könnyedén értékesítsenek vagy vásároljanak hardverkomponenseket.

## 1.2 Témaválasztás indoklása

A Pcbazár webalkalmazás témaválasztása egy közös döntés eredménye. A választás személyes érdeklődésünk, a piaci igények felismerése és a szakmai fejlődés iránti elkötelezettségünk indokolta. A meglévő piacterek nem kínálnak megfelelő lehetőségeket azok számára, akik számítógépes alkatrészekkel szeretnének kereskedni. Ezért úgy döntöttünk, hogy egy tematikus platformot hozunk létre, amely kifejezetten erre a célra szolgál. Az informatika gyors fejlődése miatt folyamatosan növekszik az igény az új és a használt számitógépes alkatrészek iránt. A technológia elterjedésével egyre több ember keres megoldásokat a költséghatékony hardverbeszerzésre, például használt termékek vásárlása. Ezzel szemben az eladok számára is egyre fontosabbá válik, hogy olyan platformot találjanak, ahol célzottan és hatékonyan érhetik el a potenciális vásárlókat.

## 1.3 Funkcionalitás

A Pcbazár webalkalmazás készítése izgalmas kihívást jelent számunkra, mivel számos funkcióval kell rendelkeznie, mint például a kosárba funkció, rendelés funkció, hirdetésfeladás funkció a regisztráció és a bejelentkezés. A Pcbazár egy olyan webalkalmazás, amely a számítógépes alkatrészek adás vételére szolgál. Az alkalmazás célja, hogy tematikus piacteret biztosítson mind az eladók, és a vásárlók számára, egyszerű gyors és hatékony használatóságot kínálva. Azonban a Pcbazár webalkalmazás készítése lehetőséget add arra, hogy jobban megismerjük a webfejlesztési folyamatokat, és bővítsük tudásunkat a front-end és back-end fejlesztés területén.

## 1.4 Célcsoport

A Pcbazár webalkalmazás célja, hogy egy tematikus piacteret biztosítson azoknak a felhasználóknak, akik számítástechnikai alkatrészeket szeretnének vásárolni vagy eladni. Az alkalmazás egyszerű gyors használatot kínál mindazok számára, akik költséghatékonyan szeretnének hozzájutni alkatrészekhez vagy eladni felesleges eszközeiket.

**Fejlesztői dokumentáció**

## 2.1 Fejlesztői környezet hardverének ismertetése

Processzor: Intel Core i5 6500 3.60GHz

Memória (RAM): 8 GB DDR4

Tárhely: 512 GB SSD

Videokártya: NVIDIA GT 1030, 2GB GDDR5

Operációs rendszer: Windows 10 64-bit

Monitor: 24” Full HD (1920x1080) kijelző

## 2.2 Fejlesztés során használt szoftverek

1.Visual Studio Code (VS code)

A Visual Studio Code egy könnyű, mégis rendkívül sokoldalú forráskód szerkesztő, amely széles körben támogatja a különböző programozási nyelveket és keretrendszereket.

Bővítmények:

Django Extension: A Django keretrendszer támogatása érdekében.

Prettier: Automatikus formázás

Integrációk:

Git verziókövetéssel és a terminál funkcióval közvetlenül a szerkesztőből dolgozhattunk.

Debugging eszközök:

Hatékony hibakeresési funkciók álltak a rendelkezésre a fejlesztés gyorsítása érdekében mint például: python debugger

2. Google Chrome

Az webalkalmazás fejlesztése során a Google Chrome fejlesztői eszközeit használtuk, amelyek számos hasznos funkcióval segítették a fejlesztési folyamatot.Ezen eszközök segítségével tudtuk a HTML, CSS és JavaScript kódokat elemezni valamint hibakeresést a böngészőn belül. A Google Chrome alapeszközei mellett bővítményeket is használtunk például a React Developer Tools, CORS Unblock .

3. Node.js

A Node.js szolgált alapként a frontend fejlesztési környezethez.

npm (Node Package Manager):

npm-t használtuk a szükséges React csomagok és bővítmények telepítéséhez, pédául React Router, axios

Az npm segítségével készítettük el a React Build verziót a “production build”-t.

4. Django

A backend fejlesztéshez a Python alapú Django keretrendszert választottuk. Ez biztosította a gyors fejlesztést, és a RESTful API-k készítéséhez szükséges eszközöket. Az alkalmazás fejlesztése során az SQLite adatbázist használtuk, amely beépített a django-ba, könnyen kezelhető adatbázis megoldást kínált.

5. Git

A verziókezeléshez a Git-et használtuk. Ez lehetővé tette a csapatmunkát és a projekt különböző verzióinak kezelését. Távoli tárolóként a GitHub szolgált, ahol a projektkódot tároltuk és megosztottuk egymással

## 2.3 Programnyelv választás indoklása

JavaScript (React)

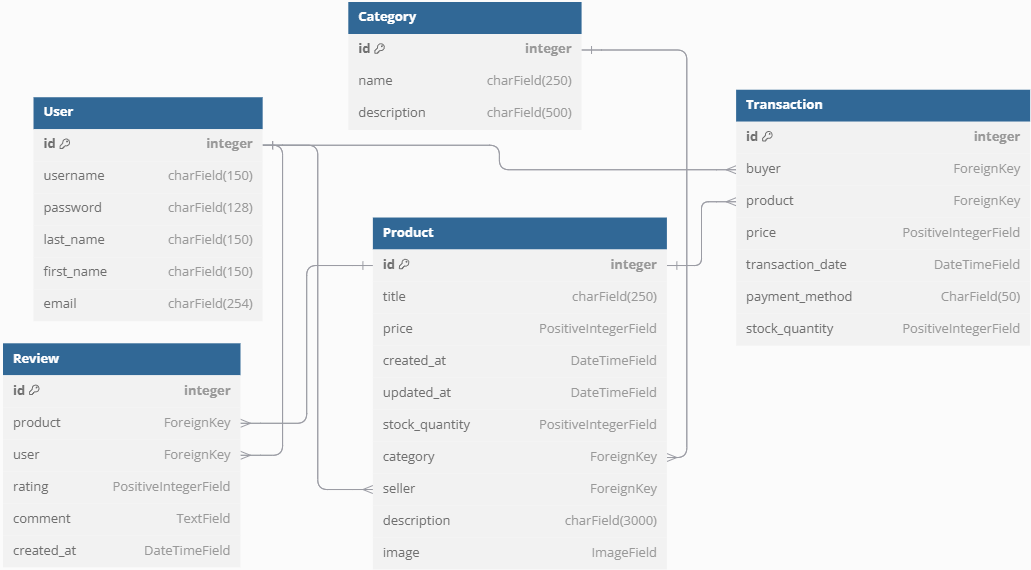
A React egy népszerű JavaScript könyvtár, amely kifejezetten a felhasználói felületek (UI) fejlesztésére készült. Lehetővé téve a komponens alapú fejlesztést, ami azt jelenti, hogy a felhasználói interfészt kisebb, újra használható komponensekből építjük fel. A virtual DOM technológia optimalizálja a teljesítményt. A React platformfüggetlen, és számos könyvtárral bővíthető így lehetőséget nyújt interaktív, korszerű webalkalmazások létrehozására

Python (Django)

A Django egy magas szintű Python webkeretrendszer, amely azzal a céllal, hogy gyors fejlesztést és tiszta, pragmatikus tervezést tegyen lehetővé. A django nagy hangsúlyt fektet a biztonságra, és számos beépített védelemmel rendelkezik a közismert webes támadások ellen, mint például az SQL injection, CSRF támadások. A Django erős adatbázis kezelési képességekkel rendelkezik és ORM-el (Object-Relational Mapping) rendelkezik. A Django ORM segítségével az adatbázis táblák könnyen leképezhetők Python osztályokra. Az ORM segítségével megkönnyíthetjük a nagy mennyiségű adat kezelését és a skálázhatóságot.

## 2.4 Adatszerkezetek

A webalkalmazásban használt adatmodellek (táblák) az alábbiakban kerülnek bemutatásra. Ezek a modellek az SQLite adatbázisban tárolják az alkalmazás számára fontos információkat, mint például felhasználói adatok, termékek, hirdetések, tranzakciók, hozzászólásokat, értékeléseket

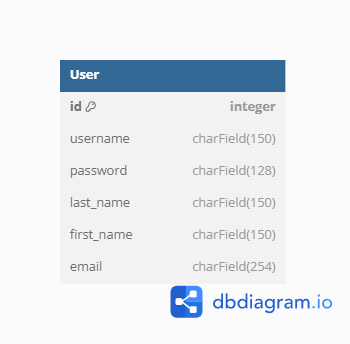


1. User Modell

A webalkalmazásban használt User modell valójában a Django beépített User modellje, amely egy jó megoldást a felhasználói hitelesítéshez és adatkezeléshez. Ezáltal nem volt szükség arra, hogy felhasználói adatokat tároló modellt hozzunk létre manuálisan mivel a Django már biztosít egy előre definiált User modellt. A Django beépített User modelljét csak be kell importálni a models.py-be:

from django.contrib.auth.models import User

A webalkalmazásban azonban csak a User modell néhány mezőjét használom a username, password, llast\_name, first\_name és email attribútumokat. Ezek a mezők elegendőek ahhoz, hogy a felhasználók alapvető azonosítását és a regisztrációs műveletek végrehajtását biztosítsák. A beépített User modell használatának az előnye, hogy időt takarít meg mivel nem kell saját hitelesítési modellt létrehozni.



1. Category modell

A Category modell a webalkalmazás kategóriáinak kezelésére szolgál. Ez a modell lehetővé teszi a kategóriák definiálását, amelyekbe különböző termékek sorolhatóak

class Category (models.Model):

name = models.CharField(max\_length=250)

description = models.CharField(max\_length=500,null=True,blank=True)

def \_\_str\_\_(self) -> str:

return self.name

class Meta():

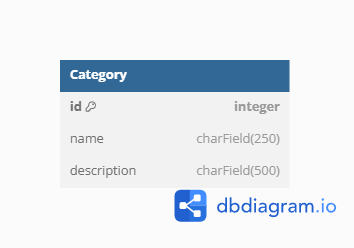
verbose\_name\_plural = "Categories"

A name mező a kategória nevét tárolja. Ez a mező kötelező, és legfeljebb 250 karakter hosszú szöveget tartalmazhat típusa CharField.

A description opcionális leírás a kategóriához. Segít részletezni mit takar az adott kategória. Legfeljebb 500 karakter hosszú szöveget tartalmazhat. Az adatbázisban található érték lehet null és az admin felületen is kitöltetlenül hagyható.

Az \_\_str\_\_ függvény meghatározza, hogyan jelenik meg a modell objektuma szöveges formában. Visszatérési értéke a name mező értéke így az adminisztrációs felületen a kategória neve jelenik meg.

A Meta osztály verbose\_name\_plural attribútuma meghatározza, hogy a modell neve többesszámban, hogy jelenjen meg.



1. Product modell

A Product modell a webalkalmazás alapvető eleme, amely a termékek adatait kezeli. Ez a modell tartalmazza azokat az információkat, amelyek a hirdetések vagy a termékleírások megjelenítéséhez szükségesek.

class Product(models.Model):

title = models.CharField(max\_length=250)

price = models.PositiveIntegerField(default=0)

created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True)

stock\_quantity = models.PositiveIntegerField(default=0)

category = models.ForeignKey(Category,on\_delete=models.CASCADE)

seller = models.ForeignKey(User, on\_delete=models.CASCADE)

description = models.CharField(max\_length=3000, null=False, blank=False)

image = models.ImageField(upload\_to="static/images", default='static/images/default.png')

def \_\_str\_\_(self) -> str:

return self.title

A title mező a termék neve. Ez a mező kötelező és legfeljebb 250 karakter hosszú lehet típusa CharField.

A price mező a termék ára, csak pozitív egész számokat fogad el. Alapértelmezett értéke 0 és típusa PositiveIntegerField.

A created\_at mező a termék létrehozásának dátuma ideje. Automatikusan beállításra kerül a termék mentésekor típusa DateTimeField. Paraméternek a auto\_now\_add van megadva, ami automatikusan beállítódik létrehozáskor. Az értékét soha nem frissíti újra, így tükrözi a létrehozás időpontját.

Az updated\_at mező a termék utolsó módosításának dátuma és ideje. Automatikusan frissül minden változtatáskor típusa DateTimeField. Paraméternek az auto\_now van megadva, így automatikusan frissül minden egyes változtatáskor az aktuális időpontra.

A stock\_quantity mező a termék raktárkészlete, amely csak pozitív egész számokat fogad el.Alapértelmezett értéke 0, típusa PositiveIntegerField.

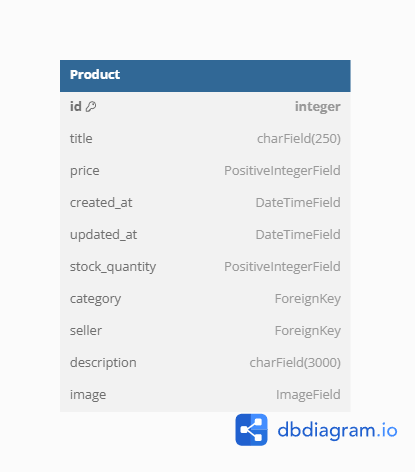
A category mező idegen kulcs a termékhez kapcsolódó kategória. Kapcsolatot hoz létre a Category modellel. Típusa ForeignKey. Paraméternek az on\_delete=models.CASCADE van megadva, ami azt jelenti, ha a kategóriát törlik a hozzá tartozó termékek is törlésre kerülnek.

A seller mező idegen kulcs a termék eladója. Kapcsolatot hoz létre a Django beépített User modellel. Típusa ForeignKey. Paraméternek az on\_delete=models.CASCADE van megadva, ami azt jelenti, ha a felhasználót törlik a hozzá tartozó termékek is törlésre kerülnek.

A description mező a termék részletes leírása. Ez a mező kötelező, és legfeljebb 3000 karaktert tartalmazhat típusa CharField.

Az image mező a termékhez tartotó kép. Az alapértelmezett kép static/images/default.png van beállítva, ha a kép feltöltés nem történik meg. Ellenben, ha a kép feltöltésre kerül akkor a static/images könyvtárba kerül.

Az \_\_str\_\_ függvény meghatározza, hogyan jelenik meg a modell objektuma szöveges formában. Visszatérési értéke a name mező értéke így az adminisztrációs felületen a kategória neve jelenik meg.



1. Transaction modell

A Transaction modell az alkalmazás vásárlási tranzakcióit kezeli, rögzítve a vásárlót, a terméket és az egyéb tranzakciós adatokat. Ez a modell biztosítja, hogy minden vásárlási esemény nyomon követhető legyen.

class Transaction(models.Model):

buyer = models.ForeignKey(User, on\_delete=models.CASCADE)

product = models.ForeignKey(Product, on\_delete=models.CASCADE)

price = models.PositiveIntegerField(default=0)

transaction\_date = models.DateTimeField(auto\_now\_add=True)

payment\_method = models.CharField(max\_length=50)

stock\_quantity = models.PositiveIntegerField(default=0)

def \_\_str\_\_(self):

return str(self.id) + " - " + (self.buyer.username) + " - " + (self.product.title)

A buyer mező idegen kulcs a tranzakciót végrehajtó vásárló. Kapcsolatban áll a Django beépített User modellel. Típusa ForeignKey. Paraméternek az on\_delete=models.CASCADE van megadva, ami azt jelenti, ha a felhasználót törlik a hozzá tartozó tranzakciók is törlésre kerülnek.

A product mező idegen kulcs a megvásárolt termék, amely kapcsolatot hoz létre a Product modellel típusa ForeignKey. Paraméternek az on\_delete=models.CASCADE van megadva, ami azt jelenti, ha a terméket törlik a hozzá tartozó tranzakciók is törlésre kerülnek.

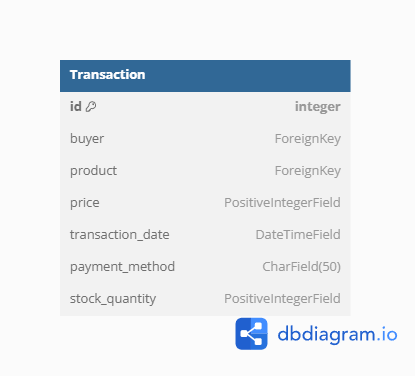
A price mező a vásárlás során kifizetett ár, csak pozitív egész számokat fogad el. Alapértelmezett értéke 0 és típusa PositiveIntegerField.

A transaction\_date mező a tranzakció létrehozásának dátuma ideje. Automatikusan beállításra kerül a tranzakció mentésekor típusa DateTimeField. Paraméternek a auto\_now\_add van megadva, ami automatikusan beállítódik létrehozáskor. Az értékét soha nem frissíti újra, így tükrözi a létrehozás időpontját.

A payment\_method mező a fizetés módot tárolja, például készpénz,bankkártya, vagy online fizetés és legfeljebb 50 karakter hosszúságú lehet típusa CharField.

A stock\_quantity mező tranzakció során értékesített termékmennyiség. A mező csak pozitív egész számokat fogad el.Alapértelmezett értéke 0, típusa PositiveIntegerField.

Az \_\_str\_\_ függvény meghatározza, hogyan jelenik meg a modell objektuma szöveges formában. Visszatérési értéke az id, buyer, product mező értéke így az adminisztrációs felületen a transzakció azonosítója, felhasználónév és a termék neve jelenik meg.



1. Review modell

A Review modell a termékhez kapcsolódó értékelések és véleményeket kezeli. Lehetővé teszi a felhasználók számára, hogy értékeljék a termékeket egy ötfokozatú skálán, és opcionálisan szöveges megjegyzést is hagyjanak.

class Review(models.Model):

product = models.ForeignKey(Product, on\_delete=models.CASCADE)

user = models.ForeignKey(User, on\_delete=models.CASCADE)

rating = models.PositiveIntegerField(validators=[MinValueValidator(1),MaxValueValidator(5)])

comment = models.TextField(null=True,blank=True)

created\_at = models.DateTimeField(auto\_now\_add=True)

def \_\_str\_\_(self) -> str:

return (self.user.username) + " - " + (self.product.title)

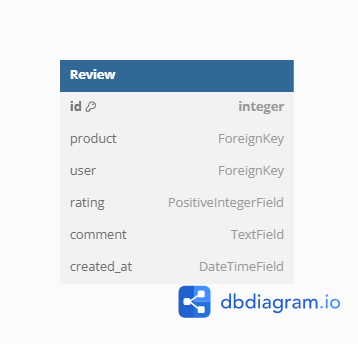
A product mező idegen kulcs a termék, amelyhez az értékelés tartozik, amely kapcsolatot hoz létre a Product modellel típusa ForeignKey. Paraméternek az on\_delete=models.CASCADE van megadva, ami azt jelenti, ha a terméket törlik a hozzá tartozó értékelések is törlésre kerülnek.

A user mező idegen kulcs a felhasználó, aki az értékelést írta. Kapcsolatban áll a Django beépített User modellel. Típusa ForeignKey. Paraméternek az on\_delete=models.CASCADE van megadva, ami azt jelenti, ha a felhasználót törlik a hozzá tartozó értékelések is törlésre kerülnek.

rating mező a termékek értékelése egy ötfokozatú skálán (1-5) típusa PositiveIntegerField. Az értékek korlátozását a validátorok biztosítják.

A comment mező opcionális szöveg megjegyzés, amelyben a felhasználó részletesen kifejtheti a véleményét a termékről típusa TextField. Az adatbázisban található érték lehet null és az admin felületen is kitöltetlenül hagyható.

Az \_\_str\_\_ függvény meghatározza, hogyan jelenik meg a modell objektuma szöveges formában. Visszatérési értéke az user, product mező értéke így az adminisztrációs felületen a felhasználónév és a termék neve jelenik meg.



## 2.5 Backend és frontend algoritmusok

Hirdetésfeladás backend oldal

@api\_view(['POST'])

@permission\_classes([IsAuthenticated])

def Add(request):

# Kérési adatok beolvasása

title = request.data.get('title')

description = request.data.get('description')

price = request.data.get('price')

stock\_quantity = request.data.get('stock\_quantity')

category\_id = request.data.get('category')

if not all([title, description, price, stock\_quantity, category\_id]):

return Response({'error': 'Hiányzó mezők! Kérjük, töltsön ki minden mezőt.'}, status=status.HTTP\_400\_BAD\_REQUEST)

try:

category = models.Category.objects.get(id=category\_id)

except models.Category.DoesNotExist:

return Response({'error': 'A megadott kategória nem létezik.'}, status=status.HTTP\_400\_BAD\_REQUEST)

product = models.Product(

title=title,

description=description,

price=price,

stock\_quantity=stock\_quantity,

category=category,

seller=request.user

)

if 'image' in request.FILES:

product.image = request.FILES['image']

product.save()

serializer = serializers.ProductSerializer(product)

return Response({'product': serializer.data}, status=status.HTTP\_201\_CREATED)

Az @api\_view(['POST']) dekorátor biztosítja, hogy csak POST típusú HTTP kérések fogadhatók el a nézetben.

Az @permission\_classes([IsAuthenticated]) dekorátor azt biztosítja, hogy csak bejelentkezett felhasználók (autentikáltak) küldhetnek kérést a végpontra. A kódban a request.data.get metódusokkal olvassuk be a kérésből érkező adatokat: title, description, price, stock\_quantity és category\_id. Ha bármelyik mező hiányzik, egy hibaüzenetet küldünk vissza a felhasználónak, és a státusz kódját 400 Bad Request-re állítjuk. A megadott category\_id alapján a try blokkban lekérdezzük a Category objektumot az adatbázisból. Ha nem található ilyen kategória, akkor az except blokkból egy hibaüzenetet küldünk vissza, és a státusz kódját szintén 400 Bad Request-re állítjuk. Ha a kategória létezik, létrehozunk egy új Product objektumot, amelyet a beérkezett adatokkal töltünk ki: title, description, price, stock\_quantity, category, és a bejelentkezett felhasználó (request.user) mint eladó. Az új terméket elmentjük az adatbázisba a product.save() metódussal. A sikeres mentést követően a terméket egy serializer segítségével JSON formátumban visszaküldjük a kliensnek a válaszban, a státusz kódot pedig 201 Created-ra állítjuk.

Hirdetésfeladás funkció frontend oldal

function Add() {

const [categories, setCategories] = useState([])

const [title, setTitle] = useState('')

const [description, setDescription] = useState('')

const [price, setPrice] = useState('')

const [stockQuantity, setStockQuantity] = useState('')

const [image, setImage] = useState(null)

const [category, setCategory] = useState('')

const [message, setMessage] = useState('')

useEffect(() => {

fetch("http://127.0.0.1:8000/api/GetAll/")

.then(response => response.json())

.then(data => {

setCategories(data.category);

})

}, []);

useEffect(() => {

if (categories.length > 0 && !category) {

setCategory(categories[0].id)

}

}, [categories]);

async function handleSubmit(e) {

e.preventDefault();

const formData = new FormData();

formData.append('title', title);

formData.append('description', description);

formData.append('price', price);

formData.append('stock\_quantity', stockQuantity)

if (image) {

formData.append('image', image);

}

formData.append('category', category);

const response = await fetch('http://127.0.0.1:8000/api/Add/', {

method: 'POST',

credentials: 'include',

headers: {

'X-CSRFToken': document.cookie.match(/csrftoken=([^;]+)/)[1],

},

body: formData,

});

const data = await response.json();

if (response.ok) {

setMessage(data.message)

window.location.href = "/"

}

else {

setMessage(data.message)

}

};

return (

<div className="add">

<div className="add-container">

<h2>Hirdetés feladása</h2>

<form onSubmit={handleSubmit}>

<div className="add-form">

<label htmlFor="title">Cím:</label>

<input

type="text"

name="title"

id="title"

placeholder='Cím'

value={title}

onChange={(e) => setTitle(e.target.value)}

autoComplete='off'

required

/>

</div>

A UseDtate hook használatával kezeljük az űrlap adatait, például a title, description, price, sticjQuantity, valamint az opcionális image és a kiválasztott category. A categories állapot a szerveren elérhető kategóriák listáját tartalmazza, míg a message az esetleges visszajelzéseket jeleníti meg. Az első useEffect a komponens betöltődésekor HTTP-kérést küld az összes kategória lekérdezésére. A kapott adatokat a categories állapotba menti. A második useEffect automatikusan beállítja az alapértelmezett kategóriát az első elemre, ha még nincs kiválasztva és egyben figyeli a categories állapotot, ha változik a categories akkor a második useEffect újra lefut és beállítja az első kategória id-át alapértelmezett értéknek. Az ürlap beküldésekor az onSubmit esemény során a handleSubmit függvény fut le. Az onChange eseménykezelők az űrlap input és select mezőiben figyelik a felhasználó által bevitt adatokat és minden változáskor frissítik a kapcsolódó állapotot. A formData objektumba kerülnek az ürlap mezői title, description, price, stoctQuantity beleértve a képfájlt is, ha meg lett adva. Az adatok a fetch segítségével kerülnek elküldésre a backend API-hoz, ahol egy POST kérés történik a megfelelő végponttal. A kérelem fejlécében beállításra kerül a CSRF token a biztonság érdekében. A credentials: 'include' opció biztosítja, hogy a süti alapú authentikáció működjön. A válasz JSON adatként kerül feldolgozásra. Ha a kérelem sikeres, egy üzenetet jelenít meg, és a felhasználót átirányítja az főoldalra, Hibás kérés esetén hiba üzenetet jelenít meg.

Kosár funkció backend oldal

@api\_view(['POST'])

@permission\_classes([IsAuthenticated])

def Cart(request):

try:

for item in request.data.get('items', []):

product\_id = item.get('product\_id')

quantity = item.get('quantity')

payment\_method = item.get('payment\_method')

if not all([product\_id, quantity, payment\_method]):

return Response({'error': 'Hiányzó mezők! Kérjük, töltsön ki minden mezőt.'}, status=status.HTTP\_400\_BAD\_REQUEST)

product = models.Product.objects.get(id=product\_id)

if quantity > product.stock\_quantity:

return Response({'error': f'Nincs elegendő készlet a(z) {product.title} termékhez.'}, status=status.HTTP\_400\_BAD\_REQUEST)

transaction = models.Transaction.objects.create(

buyer=request.user,

product=product,

price=product.price \* quantity,

payment\_method=payment\_method,

stock\_quantity=quantity

)

product.stock\_quantity -= quantity

product.save()

return Response({'message': 'Tranzakciók sikeresen létrehozva.', 'transactions': transactions}, status=status.HTTP\_201\_CREATED)

except models.Product.DoesNotExist:

return Response({'error': 'A megadott termék nem létezik.'}, status=status.HTTP\_404\_NOT\_FOUND)

Az @api\_view(['POST']) dekorátor biztosítja, hogy csak POST típusú HTTP kérések fogadhatók el a nézetben. Az @permission\_classes([IsAuthenticated]) dekorátor azt biztosítja, hogy csak bejelentkezett felhasználók (autentikáltak) küldhetnek kérést a végpontra. A kódban a for ciklus segítségével iterálunk a request.data.get('items', []) által visszaadott lista elemein (product\_id, quantity, payment\_method). Ez a request.data-ban található items kulcs értékét adja vissza. Ha az items nem létezik, egy üres listát ([]) használ alapértelmezett értékként. Ha bármelyik mező hiányzik, egy hibaüzenetet küldünk vissza a felhasználónak, és a státusz kódját 400 Bad Request-re állítjuk. A megadott product\_id alapján a try blokkban lekérdezzük a Category objektumot az adatbázisból ellenőrizzük a termék létezését. Ha a termék létezik, ellenőrizzük a készletet, amennyiben a kért mennyiség nagyobb, mint a készleten lévő mennyiség stock\_quantity, 400 Bad Request-re állítjuk és hibaüzenetet küldünk vissza. Amennyiben a kért mennyiség kisebb, mint a készleten lévő mennyiség létrehozunk egy új Transaction objektumot, amelyet a beérkezett adatokkal töltünk ki: product, product.price \* quantity, payment\_method, quantity és a bejelentkezett felhasználó (request.user) mint vásárló. A create metódust létrehozza és el is menti a transaction objektumot az adatbázisba. Ezért nincs szükség külön save() hívásra. Mivel a termék készlete csökken a tranzakcióval ezt le is kell vonni a Product modellben lévő készletből (product.stock\_quantity) majd menteni a módosított mennyiségre product.save(). Sikeres mentést követően 201 created választ küldünk, amely tartalmazza a tranzakció azonosítóját és egy üzenetet. Ha nem található ilyen kategória, akkor az except blokkból egy hibaüzenetet küldünk vissza, és a státusz kódját szintén 404 Not Found-ra állítjuk.

2.6 Tesztdokumentáció:

Űrlapkitöltési hibák

Az űrlapok például regisztráció, hirdetésfeladás, bejelentkezés, hirdetésmódosítás estén előfordult, hogy hiányosan töltötték ki a mezőket. Ennek következményeként az alkalmazás nem reagált megfelelően: sem mentés, sem hibaüzenet nem történt vagy hibaüzenetet dobott a backend. A frontend React részben az adott mezők required attribútumának az használatával biztosítottam, hogy az űrlapot csak akkor lehessen beküldeni, ha minden szükséges mező ki van töltve. Ez megelőzte a nem kívánt hibákat és javította a felhasználói élményt.

<input

type="text"

name="title"

id="title"

placeholder='Cím'

value={title}

onChange={(e) => setTitle(e.target.value)}

autoComplete='off'

required // kötelező kitölteni

/>

Privát oldalak elérése bejelentkezés nélkül

Bejelentkezés nélkül is hozzáférhettek olyan oldalakhoz, amelyeket csak bejelentkezett állapotban szabadott volna elérni, mint például hirdetésfeladás, kosár funkció, hirdetéseim, rendeléseim. Ezt az URL manuális kézi megadásával tudták megtenni. Létrehoztam egy PrivateRoute nevű komponenst, amely ellenőrizte a felhasználói authentikációt. Ez biztosította, hogy a nem bejelentkezett felhasználók automatikusan átirányításra kerüljenek a bejelentkezési oldalra, ha manuálisan akarnak olyan oldalt elérni, amit csak bejelentkezve lehetne megtenni.

const PrivateRoute = ({ isLoggedIn, children }) => {

return isLoggedIn ? children : <Navigate to="/login" replace />;

};

Publikus oldalak elérése bejelentkezett állapotban

A bejelentkezett felhasználók elérhették azokat az oldalakat is (pl. login vagy registration), amelyeket csak kijelentkezett állapotban kellett volna megjeleníteni. Hasonlóan a privát oldalakhoz, létrehoztam egy PublicRoute komponenst, amely ellenőrizte, hogy a felhasználó be van-e jelentkezve. Bejelentkezett felhasználók esetében automatikusan átirányította őket a főoldalra.

const PublicRoute = ({ isLoggedIn, children }) => {

return !isLoggedIn ? children : <Navigate to="/" replace />;

};

Közös kosár több felhasználó esetén

A kosár funkcióban jelentkezett egy súlyos probléma: ha egy felhasználó kosárba helyezett termékeket, majd kijelentkezett, és egy másik felhasználó jelentkezett be, az újonnan bejelentkezett felhasználó a korábbi felhasználó kosarát is láthatta. Ennek oka, hogy a kosár tartalma a böngésző localStorage-ében volt tárolva, amely nem különböztette meg a felhasználókat, így mindenki számára elérhetővé vált.

A probléma megoldására bevezettünk egy egyedi azonosítót, amely minden felhasználót elkülönít a kosár kezelésében. Bejelentkezéskor a felhasználónevet elmentettem a localStorage-be, és a kosáradatokat egy olyan kulccsal tároltam, amely tartalmazza az adott felhasználó nevét (pl. `cart\_${username}`). Ezáltal minden felhasználó kosara külön tárolódik, és csak az adott felhasználó férhet hozzá.

const username = localStorage.getItem('username') || ""; // Felhasználónév lekérdezése a localStorage-ből

localStorage.setItem(`cart\_${username}`, JSON.stringify(updatedCart)); // egyedi kulcsal tároltam el a kosarat a böngészőben

Kosár funkció készlet túllépési hiba

A hiba, hogy a frontend kosárkezelés során a kosárba több terméket lehetett hozzáadni, mint amennyi ténylegesen elérhető a raktárkészletben. A hibát úgy oldottuk meg, hogy létrehoztunk egy boolean típusú változót, amely ellenőrzi a kosárban lévő termékek darabszámát és a készlet mennyiségét. Ha a kosárban lévő mennyiség nagyobb vagy egyenlő a raktárkészletnél, akkor a változó igaz értéket kap, és ennek hatására letiltottuk a "Kosárba" gombot, valamint a gomb szövegét "Elfogyott"-ra módosítottuk. Ha a darabszám nem haladja meg a készletet, a gomb változatlan marad, és a felhasználó továbbra is hozzáadhatja a terméket a kosárhoz. Ez a megoldás biztosítja, hogy a felhasználó ne tudjon több terméket hozzáadni, mint amennyit a készlet lehetővé tesz.

Boolean típusú változó, amely ellenőrzi a darabszámot:

const isOutOfStock = cartItem && cartItem.quantity >= item.stock\_quantity;

<button

onClick={() => addToCart(item)}

className="add-to-cart-button"

disabled={isOutOfStock}

>

{isOutOfStock ? "Elfogyott" : "Kosárba"}

</button>

</div>

## 2.7 Bejelentkezéshez szükséges adatok

Felhasználónév: admin

Jelszó: admin

## 2.8 Fejlesztési lehetőségek

Értékelések és hozzászólások hozzáadása:

Fejleszthetjük a rendszert, hogy lehetőséget biztosítson a felhasználóknak termékek értékelésére és hozzászólások írására. Ez lehetővé tenné a vásárlók számára, hogy véleményt mondjanak a termékekről, segítve ezzel a többi vásárlót a döntéshozatalban. Az értékeléseket csillagokkal és szöveges véleményekkel is kiegészíthetnénk, hogy részletesebb visszajelzéseket kapjunk.

Több kép hozzáadása egy termékhez:

Jelenleg a rendszer csak egyetlen képet enged feltölteni egy termékhez, de egy fejlesztési lehetőség lenne, hogy lehetővé tegyük a felhasználók számára több kép hozzáadását is. Ezáltal a termékek bemutatása sokkal részletesebbé válna

Felhasználói dokumentáció

## 3.1 Szükséges eszközök a weboldal használatához

A weboldalhoz való hozzáféréshez és használatához csak egy internetkapcsolattal rendelkező eszközre van szükséged, például:

* Számítógép
* Okostelefon
* Tablet

## 3.2 Webalkalmazáshoz szükséges szoftverek

Python

Mivel a django python alapú webes keretrenszer így magát a python-t is fel kell telepítenünk a legfrisebb verziót.

Django

A backend fejlesztéséhez használt Python alapú webes keretrendszer. Django biztosítja az adatbázis-kezelést, az API-kat, az autentikációt és a többi szükséges backend logikát.

Node.js (Node Package Manager)

A React frontend fejlesztéséhez használt JavaScript könyvtár, amelynek a telepítéséhez szükséges a Node.js. A Node.js a fejlesztéshez szükséges környezetet biztosítja.

Böngésző

* Google Chrome
* Microsoft Edge

## 3.3 Webalkalmazás elindítása

A webalkalmazás megnyitásához nyisd meg a fájlkezelőt. Navigálj arra a mappára, ahol a webalkalmazás található. A címsorba, ahol a mappa elérési útja szerepel a helyére írd be, hogy “cmd” és nyomd meg az ENTER-t így megjelenik a parancssor. Futtasd az alábbi kilenc parancsot:

**1. py -m venv .**

**2. Scripts\activate**

**3. pip install -r requirements.txt**

**4. cd frontend**

**5. npm install**

**6. npm run build**

**7. cd..**

**8. cd backend**

**9. py manage.py runserver**

Amennyibben sikeresen futtattad a parancsokat a böngésződben már az alábbi URL-en eléred a webalkalmazást:  
  
 [**http://127.0.0.1:8000/**](http://127.0.0.1:8000/)

## 3.4 Bejelentkezéshez szükséges adatok

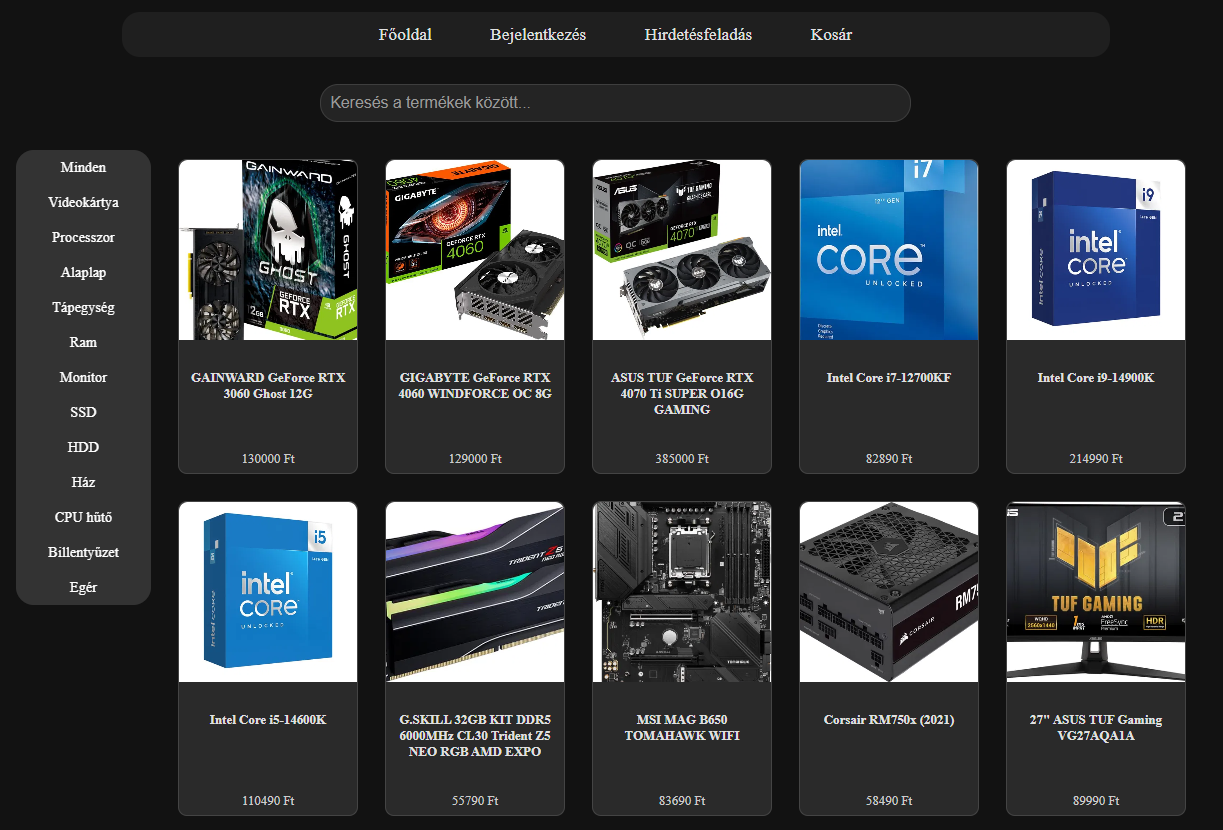
Felhasználónév: admin

Jelszó: admin

## 3.5 Webalkalmazás használatának az ismertetése

### 3.5.1 Főoldal

Amikor a felhasználó nincs bejelentkezve akkor ebben az állapotban bizonyos funkciók korlátozottak vagy rejtettek maradnak. Nem bejelentkezett felhasználóként megtekintheted a termékeket, használhatod a keresőt és a kategóriaszűrőt, valamint bejelentkezhetsz vagy regisztrálhatsz. A kosár és a hirdetésfeladás funkció látható, de csak bejelentkezett állapotban érhetők el.



### 3.5.2 Regisztráció

A felhasználók a webalkalmazásban regisztrálhatnak, amelyhez az alábbi öt adat megadása szükséges:

Felhasználónév:

* Legalább 6 karakter hosszúnak kell lennie.
* Bármilyen karaktert tartalmazhat.
* Nem maradhat üresen; a rendszer nem engedi elküldeni az űrlapot üres mező esetén.

Jelszó:

* Legalább 6 karakter hosszúnak kell lennie.
* Bármilyen karaktert tartalmazhat.
* Nem maradhat üresen; a rendszer nem engedi elküldeni az űrlapot üres mező esetén.

E-mail cím:

* Érvényes e-mail formátumban kell megadni (például: [felhasznalo@domain.hu](mailto:felhasznalo@domain.hu)).
* A "@" előtt bármilyen karaktert tartalmazhat.
* Nem maradhat üresen; a rendszer nem engedi elküldeni az űrlapot üres mező esetén.

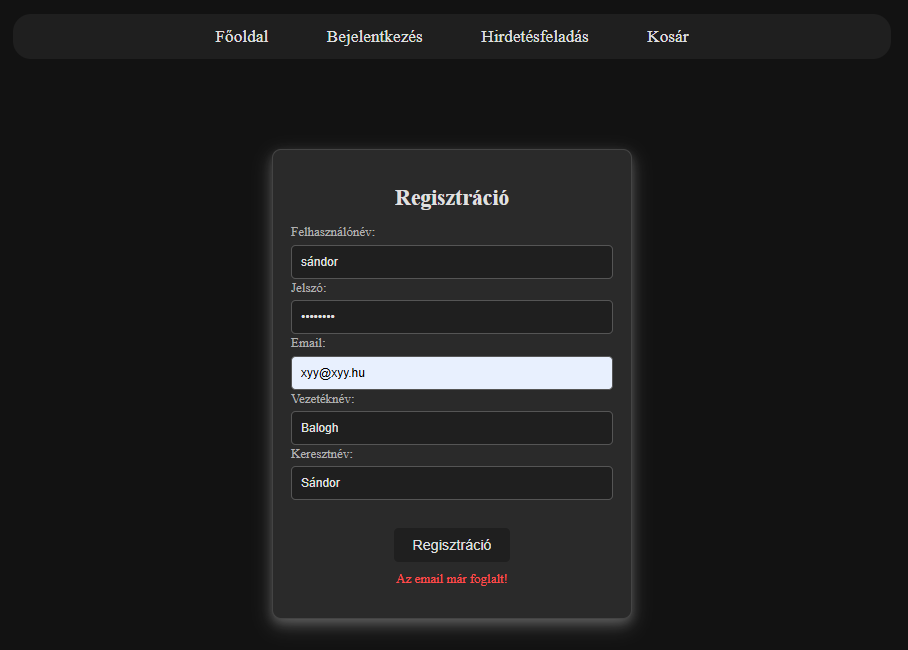
Vezetéknév:

* Nincs megadott minimális vagy maximális hosszúság.
* Bármilyen karaktert tartalmazhat.
* Nem maradhat üresen; a rendszer nem engedi elküldeni az űrlapot üres mező esetén.

Keresztnév:

* Nincs megadott minimális vagy maximális hosszúság.
* Bármilyen karaktert tartalmazhat.
* Nem maradhat üresen; a rendszer nem engedi elküldeni az űrlapot üres mező esetén.

A rendszer minden mező esetében ellenőrzi a megadott adatokat, és hibaüzenetet jelenít meg, ha nem felelnek meg a fenti kritériumoknak. Csak akkor küldhető el az űrlap, ha minden mező helyesen van kitöltve. Regisztráció után átirányít a bejelentkezéshez. Valamint, ha foglalt lenne az email vagy a felhasználónév azt jelzi egy backend-től kapott hibaüzenettel a Regisztráció gomb alatt.



### 3.5.3 Bejelentkezés

A webalkalmazás bejelentkezési funkciója egyszerű és biztonságos, a felhasználóknak a következő adatokat kell megadniuk:

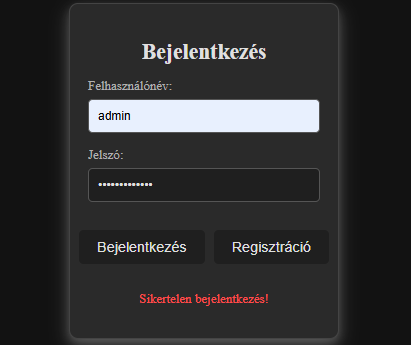
Felhasználónév:

* A beviteli mező nem maradhat üres.
* A rendszer ellenőrzi, hogy a megadott felhasználónév létezik-e az adatbázisban.
* Az automatikus kiegészítés funkció ("username") segíti az adatbevitel gyorsaságát.

Jelszó:

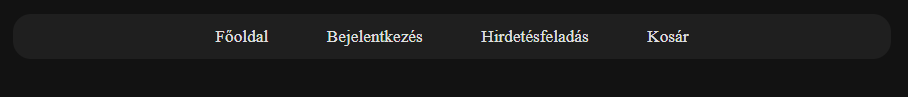
* A beviteli mező nem maradhat üres.
* A jelszó automatikusan rejtett módon kerül megadásra a "password" típusú inputmezőnek köszönhetően.
* Az automatikus kiegészítés funkció ("current-password") segíti a gyors bejelentkezést.

Amennyiben sikeres a bejelentkezés átirányít a főoldalra. Valamint hitelesítési hiba esetén hibaüzenetet küld a backend-től amit a gombok alatt jelenít meg.

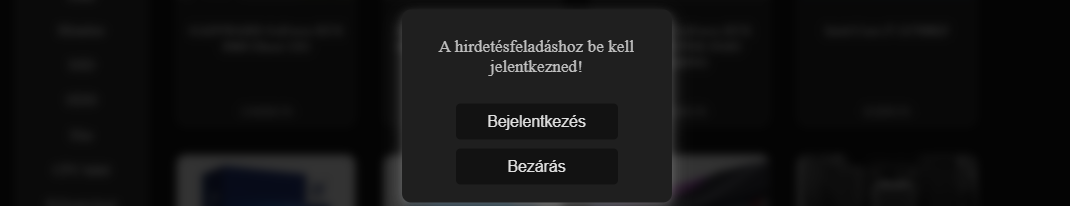


### 3.5.4 Navigációs sáv

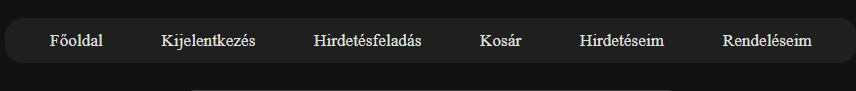
A navigációs sáv nem bejelentkezett állapotban 4 menüpontot kínál: Főoldal, Bejelentkezés, Kosár, és Hirdetésfeladás. Azonban a Kosár és a Hirdetésfeladás funkciók kizárólag bejelentkezett felhasználók számára érhetők el.

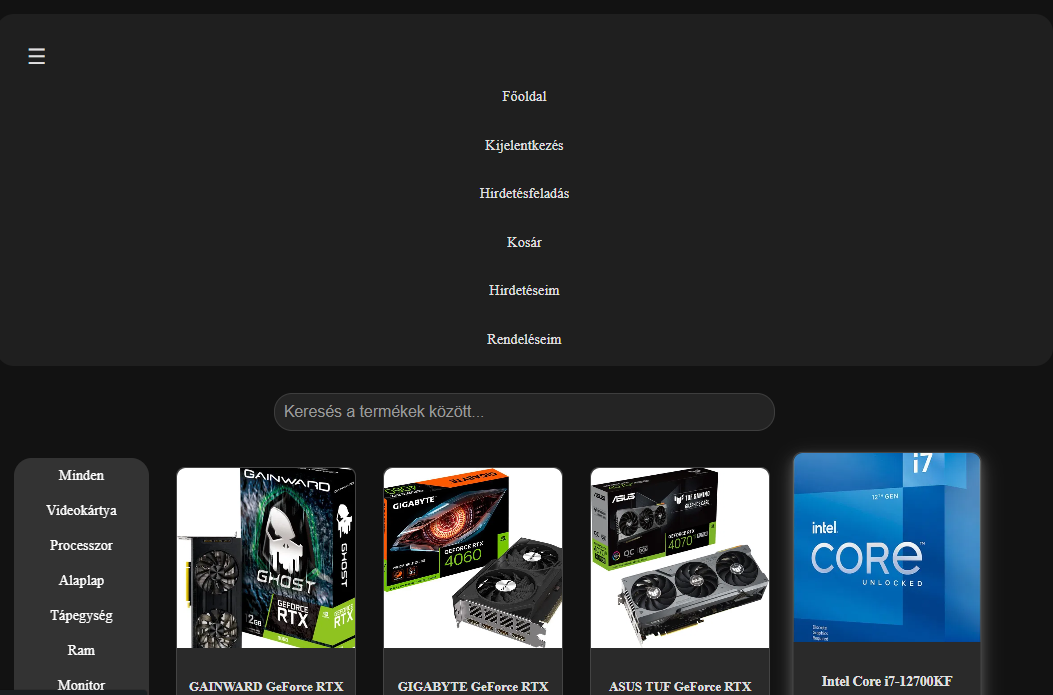


Amennyibben a hirdetésfeladásra vagy a Kosár funkcióra kattint a felhasználó abban az esetben kis ablak (Modal komponens), ami választást kínál vagy bejelentkezel vagy bezárod az ablakot.



A navigációs sáv bejelentkezett állapotban 6 menüpontot kínál: Főoldal, kijelentkezés, Hirdetésfeladás, Kosár, Hirdetéseim, Rendeléseim és mindegyik elérhető.

És ha a navigációs sáv szélessége lecsökken 1200px alá akkor átvált hamburgermenüs dizájnra:



### 3.5.5 Hirdetésfeladás

A hirdetés feladása funkció lehetővé teszi, hogy a felhasználók termékeiket vagy hirdessék a platformon csak bejelentkezve érhető el. A feladásához az alábbi adatokat kell megadni:

**Cím:** A termék neve, amely nem maradhat üresen, és minden típusú karaktert tartalmazhat.

**Leírás:** A termék részletes bemutatása, amely szöveges információkat tartalmaz a termékről nem maradhat üresen, és minden típusú karaktert tartalmazhat.

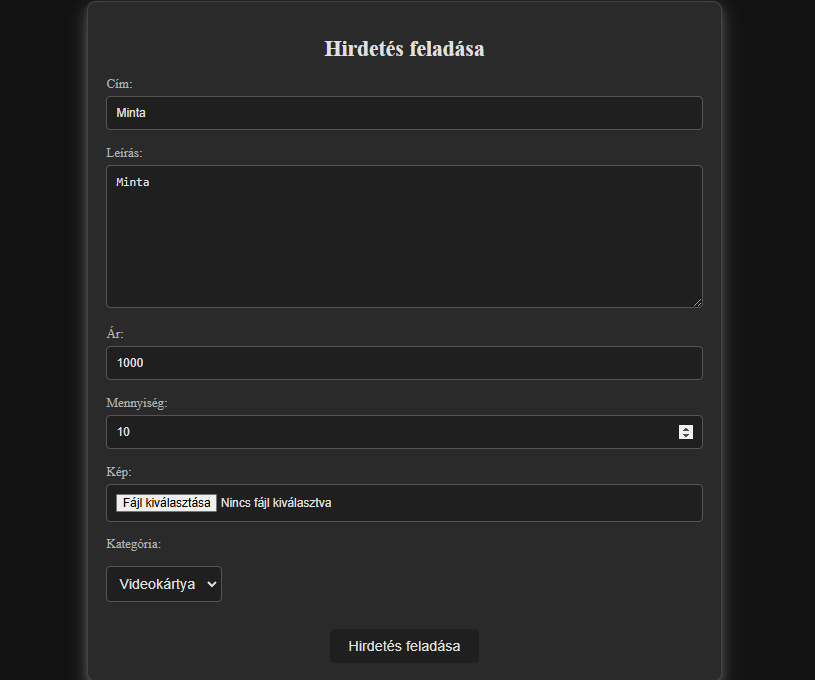
**Ár:** A termék ára, csak szám típust fogad el

**Mennyiség:** A raktáron lévő darabszám, csak szám típust fogad el

**Kép:** A termékhez tartozó kép feltöltése (a felhasználónak lehetősége van fájlt feltölteni). Amennyiben nem tölt fel képet alapértelmezett kép lesz beállítva. jpg és png formátummal tesztelt

**Kategória:** A termék kategóriájának kiválasztása az előre meghatározott kategóriák közül.

A hirdetés feladása előtt minden mezőt kötelezően ki kell tölteni. A rendszer biztosítja, hogy minden adat helyes formátumban kerüljön be, mielőtt a felhasználó elküldheti a hirdetést. Sikeres hirdetésfeladás esetén tovább irányít a főoldalra és jelzi is üzenettel.



### 3.5.5 Kosár funkció

**Kosár tartalmának megjelenítése:**

Ha a kosár üres, a "A kosaram üres." üzenet jelenik meg.

Ha vannak termékek a kosárban, akkor azok neve, ára, mennyisége jelenik meg és törlni is lehet az adott terméket.

**Mennyiség módosítása:**

A felhasználó módosíthatja a termékek mennyiségét egy numerikus input mezőn keresztül, ahol a minimum érték 1 és a maximum érték a készleten lévő darabszám.

**Összesített ár:**

A kosárban lévő összes termék ára automatikusan kiszámításra kerül.

**Fizetési mód kiválasztása:**

A felhasználó választhat a következő fizetési módok közül: bankkártya, készpénz, PayPal.

**Fizetés:**

A "Fizetés" gomb akkor válik aktívvá, ha a felhasználó kiválasztotta a fizetési módot, és van termék a kosárban.

Vásárlás során megjelen egy üzenet a sikeres vagy hibás tranzakcióról.



### 3.5.6 Hirdetéseim funkció

A Hirdetéseim oldal lehetővé teszi a felhasználók számára, hogy megtekintsék és kezeljék saját hirdetéseiket. Az alábbi funkciók érhetők el:

Hirdetések listája:

Ha a felhasználónak van hirdetése, akkor a hirdetések listázásra kerülnek. Minden hirdetés tartalmazza a következő információkat:

A hirdetés képe

A hirdetés címe

Az ár (forintban)

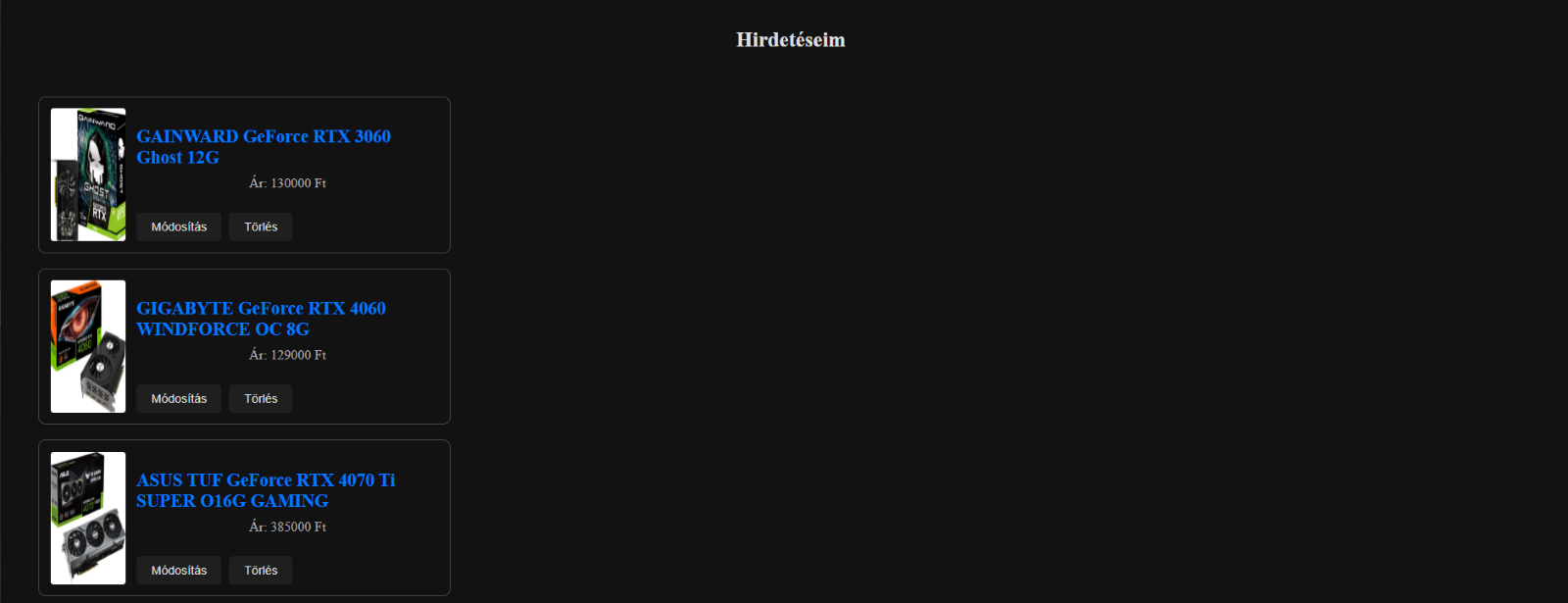
A hirdetés mellett két gomb található:

Módosítás: Lehetőséget biztosít a hirdetés módosítására. Erre kattintva a felhasználó átirányításra kerül a hirdetés módosítási oldalára.

Törlés: A hirdetés törlésére szolgál. A törlés esetén felugrik egy Modal ablak (Modal komponens) ahol megerősítheted a törlést vagy visszaléphetsz.

Nincs hirdetés:

Ha a felhasználónak nincsenek hirdetései, akkor egy üzenet jelenik meg: "Nincsenek hirdetéseid."



### 3.5.7 Módosítás funkció

A Hirdetés Módosítása funkció szinte megegyezik az új hirdetés hozzáadásával, azzal a különbséggel, hogy:

A kategória nem módosítható.

Az űrlap mezői előre kitöltődnek a már elmentett adatokkal, amelyek szerkeszthetők.



### 3.5.8 Rendeléseim funkció

A Rendeléseim oldal lehetővé teszi a felhasználók számára, hogy nyomon kövessék a korábbi rendeléseiket. A következő funkciók állnak rendelkezésre:

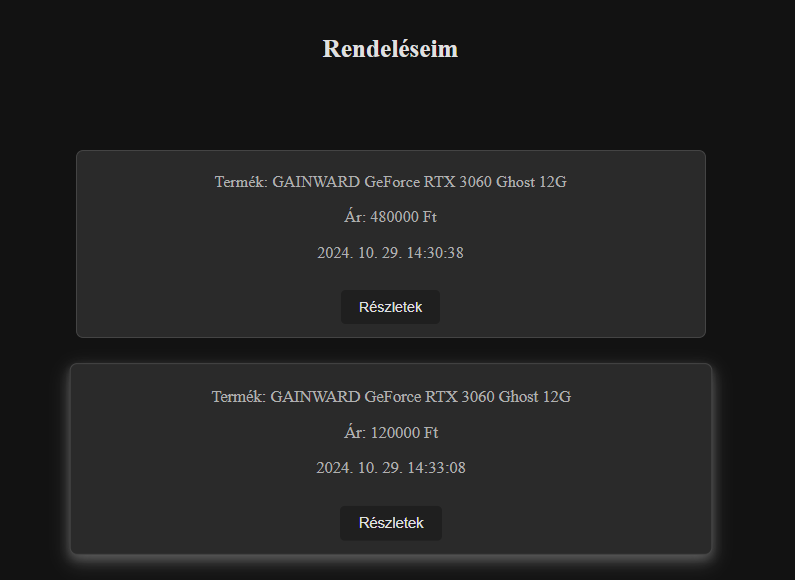
Rendelések listája:

Ha a felhasználónak nincs rendelése, akkor a "Nincsenek rendeléseid. Kérlek, nézz vissza később!" üzenet jelenik meg.

Ha a felhasználónak van rendelése, azok listázása történik, ahol minden rendelés tartalmazza a termék nevét, árát és a tranzakció dátumát.

Részletek megtekintése:

Minden rendeléshez tartozik egy "Részletek" gomb, amellyel a felhasználó további információkat kérhet a rendelésről.



Részletek megtekintése:

Minden rendeléshez tartozik egy "Részletek" gomb, amellyel a felhasználó további információkat kérhet a rendelésről.

### 3.5.9 Részletek funkció

A Részletek funkció lehetőséget nyújt egy adott rendelés részleteinek megtekintésére:

* Termék neve
* Vásárló felhasználóneve
* Rendelés ára
* Mennyiség
* Fizetési mód
* Tranzakció dátuma

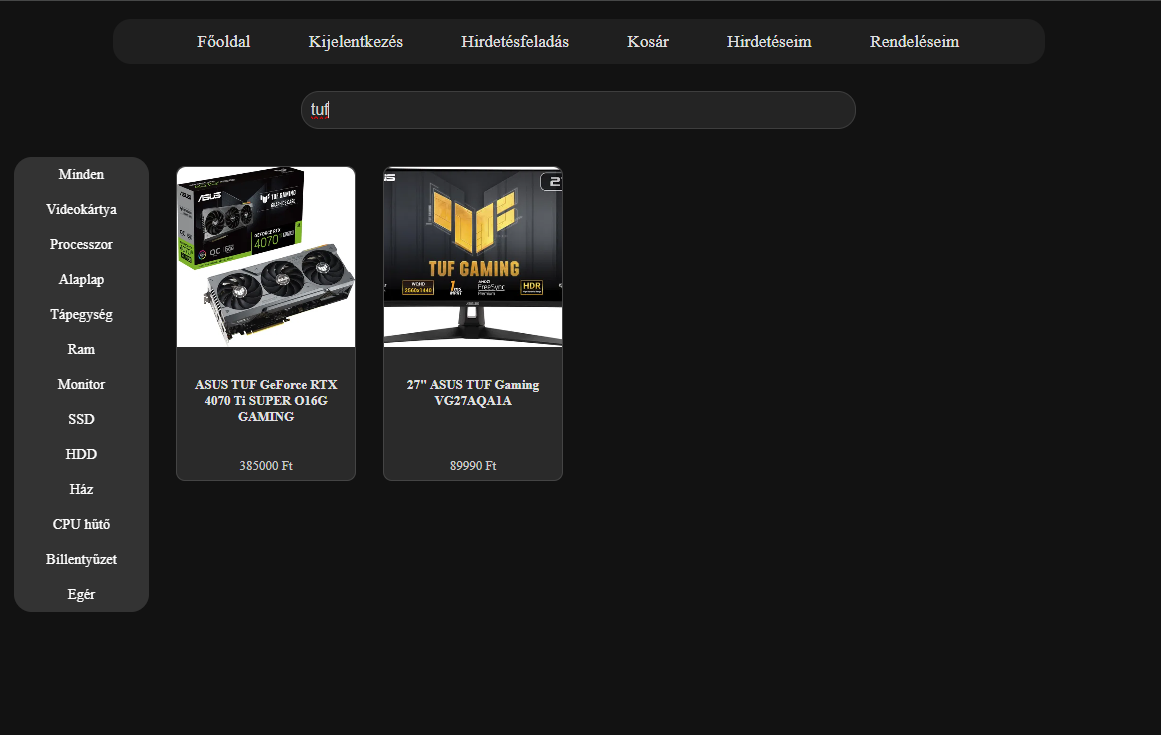


### 3.5.10 Keresés funkció

Keresés funkció lehetőséget biztosít a termékek között való gyors és egyszerű keresésre.

A keresőmezőbe beírt kifejezés valós időben szűri a termékeket a nevük alapján. Tehát, ha több termékben is megtalálható az a kifejezés akkor azokat is kilistázza

A keresés kis- és nagybetű-független, így a felhasználók kényelmesen kereshetnek bármilyen formátumban.



### 3.5.11 Product funkció

A Termék részletező funkció részletes információkat nyújt az adott termékről.

**Termékadatok megjelenítése:**

A termék címe, ára, leírása, készletinformációi, feltöltési dátuma és az eladó neve jelenik meg.

**Kép megjelenítése:**

A termékről feltöltött kép is bemutatja az árut.

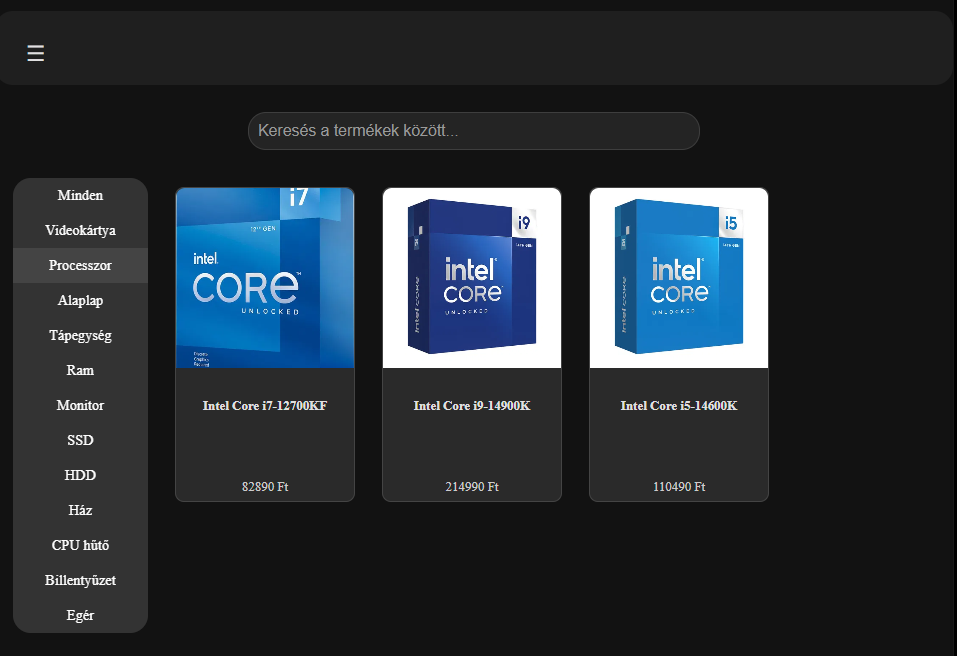
**Kosárba helyezési lehetőség:**

Bejelentkezett felhasználók számára aktív a Kosárba gomb.

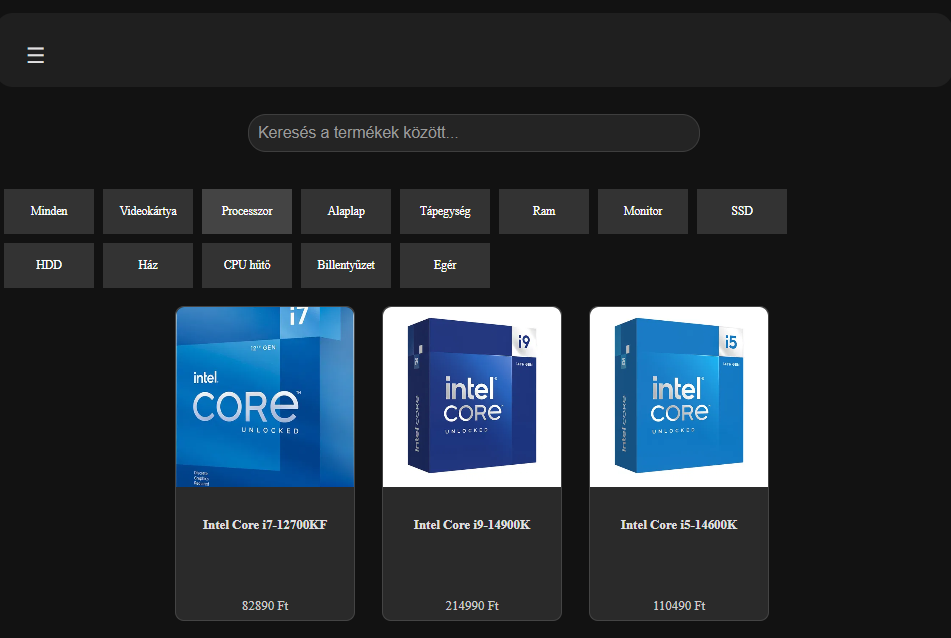
Ha a termék készlete elfogyott, a gomb automatikusan letiltásra kerül, és a "Elfogyott" szöveg jelenik meg.

### 3.5.12 Sidebar (Oldalsáv)

Az oldalsáv (Sidebar) a kategóriák alapján történő szűrést teszi lehetővé, megkönnyítve a felhasználók számára a termékek közötti navigációt. Az oldalsáv elemei között a "Minden" opcióval az összes termék megjeleníthető. Az egyes kategóriák kiválasztásával csak az adott kategóriába tartozó termékek jelennek meg.



**Reszponzív megjelenés:** 768 pixel alatti képernyőnél az oldalsáv automatikusan csempés kinézetre vált, javítva a mobilfelhasználói élményt.



# Összefoglaló

Rendkívül elégedettek vagyunk az elkészült webalkalmazásunkkal, hiszen sikerült megvalósítani azokat az alapvető funkciókat, amelyeket a projekt kezdetén kitűztünk magunk elé. Az oldal jól teljesíti az elvárásainkat, és úgy érezzük, hogy egy szuper alapot hoztunk létre, amire a jövőben tovább építkezhetünk. Büszkék vagyunk arra, hogy a projekt eredménye nemcsak technikai szempontból értékes, hanem a felhasználók számára is kényelmesen használható.

A fejlesztési folyamat során számos kihívással szembesültünk, de minden ilyen helyzet lehetőséget adott arra, hogy tanuljunk és fejlődjünk. Például nagy hangsúlyt fektettünk arra, hogy az alkalmazás reszponzív legyen, így különböző eszközökön is jól működjön. A keresés és a szűrés funkciók implementálása különösen izgalmas feladat volt, hiszen ezek jelentősen hozzájárulnak a felhasználói élményhez.

Természetesen az elkészült munka nem jelenti a projekt végét, hanem egy kiindulópontot. Már most vannak terveink arra, hogy további fejlesztésekkel bővítsük az oldalt. Ilyen például a termékek értékelésének teljes körű működése, ami lehetővé teszi a felhasználók számára, hogy visszajelzést adjanak a vásárlásaikról. Emellett a hozzászólások az adott termékhez és a felhasználói profilok testreszabhatóságának bevezetése is fontos lépés lesz, hiszen ezek az újítások tovább növelhetik az oldal közösségi jellegét és használhatóságát.

Összességében büszkék vagyunk az eddig elért eredményekre, és izgatottan várjuk, hogy a projekt hogyan fejlődhet tovább. A tapasztalatok és a tanulságok, amelyeket a fejlesztési folyamat során szereztünk, nemcsak ehhez a projekthez, hanem a jövőbeli munkáinkhoz is nagyban hozzájárulnak.