



PREMSELECT

Konderla Sámuel, Nagy Marcell István, Verebes Bálint Ignác

Nógrád Vármegyei Szakképzési Centrum Szent-Györgyi Albert
Technikum
Szoftverfejlesztő és -tesztelő technikus (szakma azonosító: 5-0613-12-03)

2025. április 28.

Tartalom

Bevezetés	4
1. Felhasználói dokumentáció.....	5
1.1. Főoldal bemutatása	5
1.2.1. Beágyazott Google Térkép.....	7
1.3. Posztok oldal bemutatása	8
1.3.1. Posztok feltöltése.....	9
1.4. Profil oldal bemutatása	10
1.4.1 Funkciók és használat	11
1.5. Bejelentkezés és regisztrációs oldal bemutatása	12
1.6 Licit oldal bemutatása	13
1.7. Kosár oldal bemutatása	15
1.8. Fizetés oldal bemutatása	16
1.8.1 Sikeres megrendelés	17
1.9. Menü	17
1.10 Lapozó	18
2. Fejlesztői dokumentáció.....	19
2.1. Felhasznált technológiák.....	19
2.1.1. Visual Studio Code	19
2.1.2. HTML.....	19
2.1.3. CSS.....	20
2.1.4. Php	20
2.1.5. JavaScript.....	20
2.1.6. Composer	20
2.1.7. MySql.....	21
2.1.8. Xampp	21
2.1.9. GitHub.....	21
2.1.9. DbDiagram.....	22

2.1.10. Microsoft Word	22
2.1.11. Discord.....	22
2.1.12 Trello.....	23
2.1.13 EchoApi	23
2.2. Technológiák áttekintése	23
2.2.1 PHP főbb jellemzői	23
2.2.2. HTML, CSS, JavaScript rövid bemutatása	24
2.3 Weboldal elérése	25
2.4. Projekt telepítése, és a webshop elindítása.....	25
2.5. Környezeti változók.....	26
2.6. Útvonalak	27
2.7. Backend felépítése	28
2.7.1. Felhasználói profil kezelése	29
2.7.2 Termékek kezelése.....	32
2.7.3 Licitek kezelése	36
2.7.4 Fizetés és kosár kezelése.....	39
2.8. Frontend felépítése (HTML, CSS, JavaScript).....	43
2.8.1. HTML oldalak szerkezete.....	43
2.8.2. Stíluslapok (CSS).....	43
2.8.3. Dinamikus műveletek (JavaScript, Fetch API)	44
2.9. API kialakítása	45
2.9.1. API végpontok rövid bemutatása.....	45
2.9.2. JSON formátumú kommunikáció.....	47
2.10. Biztonsági megfontolások	48
2.10.1. SQL Injection elleni védelem.....	48
2.10.2. Email validáció	49
2.10.3. Jelszó erősség ellenőrzése.....	49
2.10.4. Jelszó hash-elése.....	49
2.10.5. Üres mezők ellenőrzése	49

2.10.6. Email duplikáció ellenőrzése	50
2.10.7. JSON bemenet ellenőrzése	50
2.10.8. Tartalom típusa	50
2.11. Tesztelés.....	51
2.11.1 Get teszt	51
2.11.2 Delete teszt.....	52
2.11.3 Post teszt	53
2.11.4 Update.....	54
2.11.5. Frontend funkciók tesztelése	55
3. Az adatbázis célja	56
3.1. Tervezés megkezdése.....	56
3.2. Tervezési lépések	56
3.3. Egyedtipusok/egyedek meghatározása	56
3.4. Kapcsolatok meghatározása	57
3.5. N:M kapcsolatok felbontása	58
3.6. Táblák.....	58
3.6.1 user.....	58
3.6.2. settlement	59
3.6.3. counties	60
3.6.4. sales	60
3.6.5. products.....	60
3.6.6. image.....	61
3.6.7. brand.....	61
3.6.8. category	62
3.6.9. auction	62
3.7. Adatbázis diagram dbdiagram felületen	63
3.8. Adatbázis szerkezete phpMyAdmin felületen.....	64
3.9. Adatbázis továbbfejlesztési lehetőségek	65
4. Felmerült akadályok	65

5. Összefoglalás	65
6. Források	66

Bevezetés

A PremSelect egy olyan weboldal/webshop, ahol a ruházat kap főszerepet. Itt bármilyen márkás vagy hétköznapi ruházati termékek árusítása és vásárlása lehetséges. Weboldalunk eltérő módon kínál megoldást egy hétköznapi problémára, hogy elérhetőbbé tegye mindenki számára kedvenc termékeik megvásárlását akár töredék áron magánszemélyektől, de biztonságosan. A fő probléma jelenleg az ilyen jellegű termékek vásárlásával, hogy a legtöbb ember vagy nagyon sok pénzt költ el a ruhákra, vagy a közösségi médián esik csalás áldozatául, mivel ott nem tudják garantálni a vásárlás menetének biztonságos lebonyolítását.

A csapatunk három főből áll, rengeteg tapasztalatot szereztünk mind szakmai, mind a csapatmunka területén a munka készítése közben. Mélyebben megismerhettük a használt technológiákat, valamint a fejlettebb weboldalak összetettségével is találkozhattunk kutatómunkánk során. A munkamegosztás a következőképp zajlott: Nagy Marcell a háttérrendszerrel foglalkozott (backend), Konderla Sámuel feladata volt az adatbázis, Verebes Bálint pedig a weboldal megjelenéséért volt felelős (frontend). A munkamegosztástól függetlenül volt lehetőségünk egymás problémáival is foglalkozni/segítséget nyújtani, így mondhatjuk, hogy mindannyian dolgoztunk a webshop minden terén. A munka első ötlete még szeptemberben született meg, annyiban tért el az ötlet a jelenlegi projekttől, hogy kifejezetten csak cipők eladására és vételére kínált volna lehetőséget, ez az ötlet végül a magas konkurencia és a kevés egyedi bővíthetőségi lehetőség miatt került elvetésre, és ezt követően született meg a jelenlegi ötlet. Megfelelő tudással és a követelményekre felkészülten indultunk és végeztük a munkánkat köszönhetően a felkészítő tanárainknak.

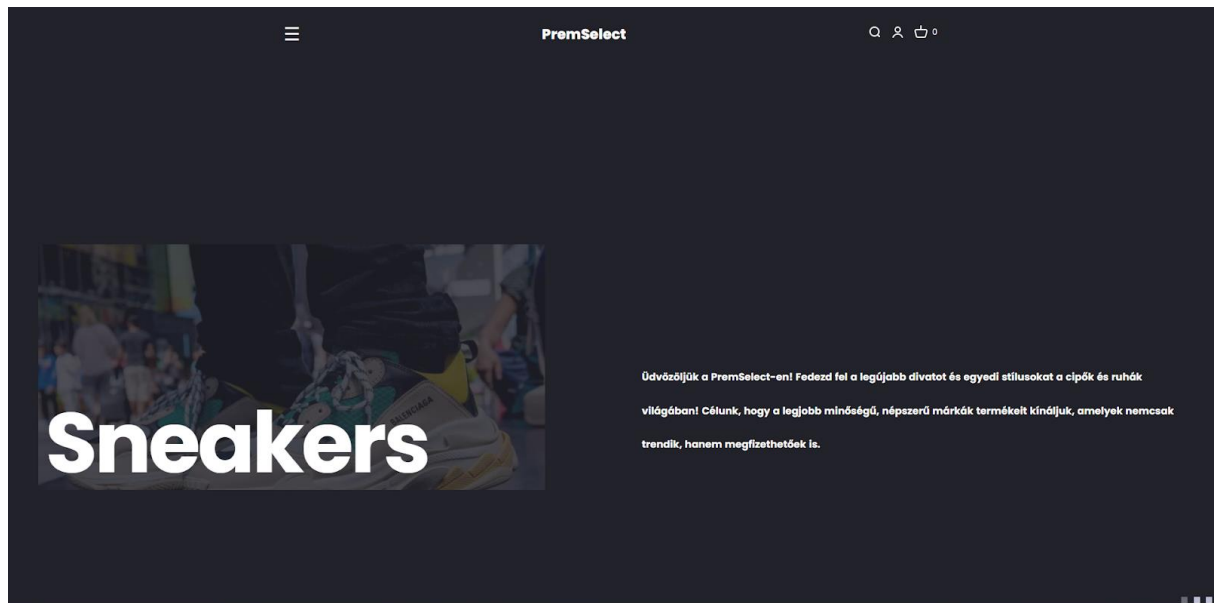
1. Felhasználói dokumentáció

A rendszer fejlesztésében Verebes Bálint, Konderla Sámuel és Nagy Marcell vettek részt. Mindhárman dolgoztak a backend és a frontend kialakításán, biztosítva a rendszer megfelelő működését és felhasználóbarát felületét. A backend fejlesztése során az adatok kezelésére és a szerveroldali logika megvalósítására helyezték a hangsúlyt, míg a frontend fejlesztése során a felhasználói élmény és az átlátható megjelenítés volt a fő szempont.

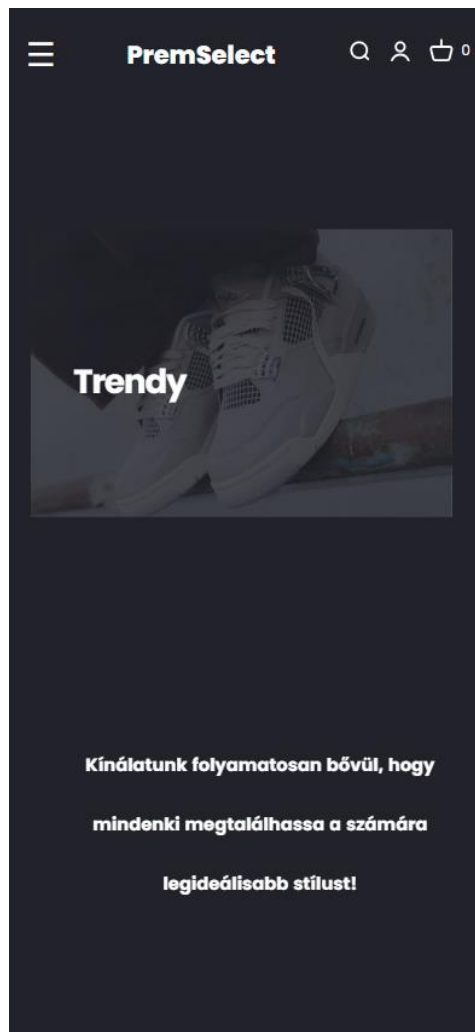
1.1. Főoldal bemutatása

A weboldal landing page-ének navigációs sávja a képernyő tetején helyezkedik el, és több fontos elemet tartalmaz. A bal felső sarokban található egy menü gomb, amely egy három vonalból álló ikon. Erre kattintva egy oldalsó menü nyílik meg, amely a weboldal főbb kategóriáit tartalmazza. A navigációs sáv jobb oldalán három ikon található: a kereső gomb, a profil gomb és a kosár gomb. A kereső ikon segítségével a felhasználók gyorsan kereshetnek a weboldalon elérhető termékek között. A profil ikon lehetőséget biztosít a regisztrációra, valamint a bejelentkezésre azok számára, akik már rendelkeznek fiókkal. A kosár ikonra kattintva a felhasználók megtekinthetik a kosárba helyezett termékeiket, és véglegesíthetik a vásárlást. A landing page fő tartalmában három nagy méretű banner kép található, amelyek a prémium minőségű cipők és ruházati termékek világát hirdetik, mellette pedig egy bemutatkozó szöveg helyezkedik el, amely ismerteti a weboldal célját és kínálatát.

A weboldal design szempontjából törekedtünk a szolid, de továbbra is látványos és modern kinézetre.



1. ábra Főoldal asztali nézetben



2. ábra Főoldal mobil nézetben

1.2. Kapcsolat oldal bemutatása

A főoldal menüjében a “kapcsolat” gombra kattintva a kapcsolat oldalunkon találja magát a felhasználó. A PremSelect kapcsolat oldala lehetőséget biztosít a felhasználóknak arra, hogy üzenetet küldjenek az ügyfélszolgálat számára, valamint információt kapjanak a fejlesztői csapat székhelyéről és elérhetőségeiről. A láblécben elérhetők az ügyfélszolgálat hivatalos elérhetőségei, beleértve az e-mail címet és a telefonszámot.

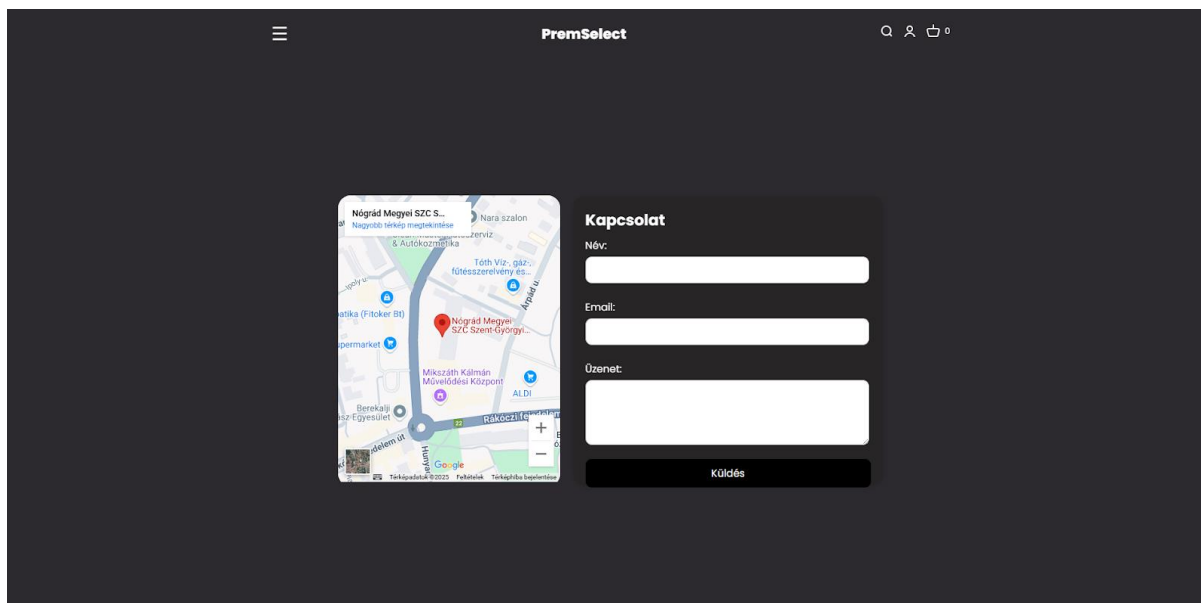
Az üzenetküldéshez a felhasználónak ki kell töltenie egy űrlapot, amely tartalmazza a nevét, e-mail címét és az üzenetét, majd a "Küldés" gombra kattintva továbbíthatja azt a fejlesztői csapat felé. Fontos megjegyezni, hogy az üzenetküldéshez bejelentkezés szükséges, amelyet a profil ikonra kattintva lehet elvégezni.

A navigáció is egyszerűen kezelhető, hiszen a felhasználók a "PremSelect" logóra kattintva visszatérhetnek a főoldalra, a kosár ikonra kattintva beléphetnek a fizetési oldalra, a kereső ikonra kattintva böngészhetnek a ruhák és cipők között, míg a bal felső sarokban található menü gomb segítségével megnyithatják a főmenüt és eljuthatnak ismét az oldal különböző részeire.

Ezen az oldalon lehetőség van problémák vagy javaslatok bejelentésére is, amelyeket a felhasználók az üzenetküldő űrlapon keresztül juttathatnak el az fejlesztői csapatnak. Összességében a PremSelect kapcsolat oldala egy felhasználóbarát és könnyen kezelhető felületet biztosít, amely lehetővé teszi a fejlesztői csapattal való hatékony kommunikációt, miközben átlátható módon biztosítja az elérhetőségeket és a navigációs lehetőségeket az oldalon.

1.2.1. Beágyazott Google Térkép

A weboldalunkon található egy beágyazott Google Térkép, amely a fejlesztői csapat hivatalos székhelyének pontos címét mutatja. A beágyazott térkép lehetővé teszi, hogy a látogatók könnyedén megtalálják a cég címét és elérhetőségét, valamint segít a navigációban.

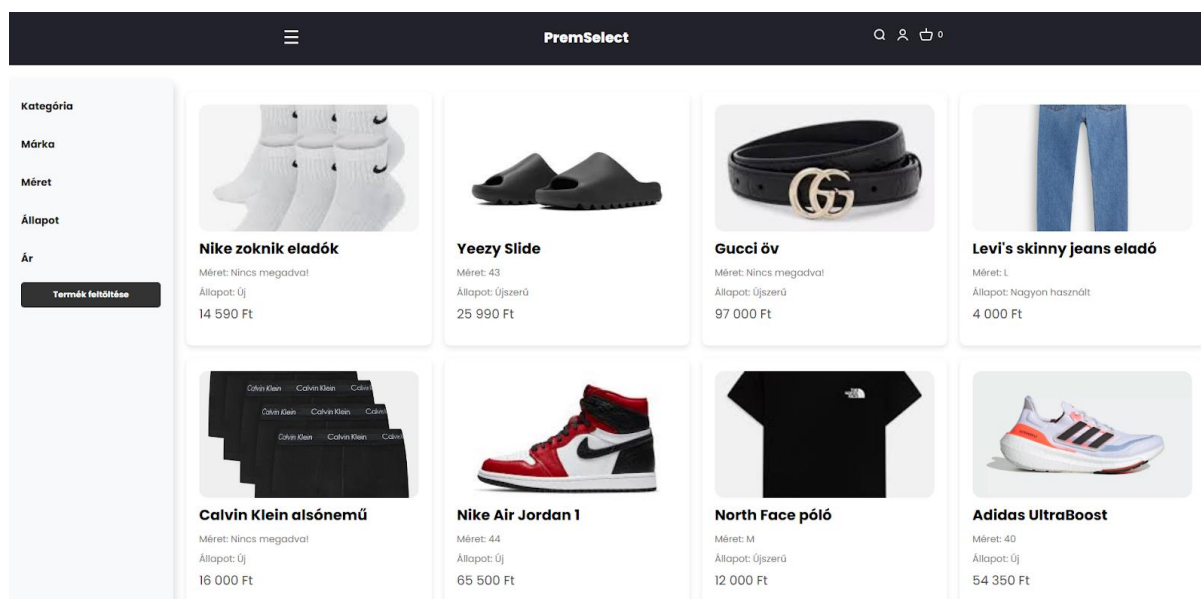


3. ábra Kapcsolat oldal asztali nézetben

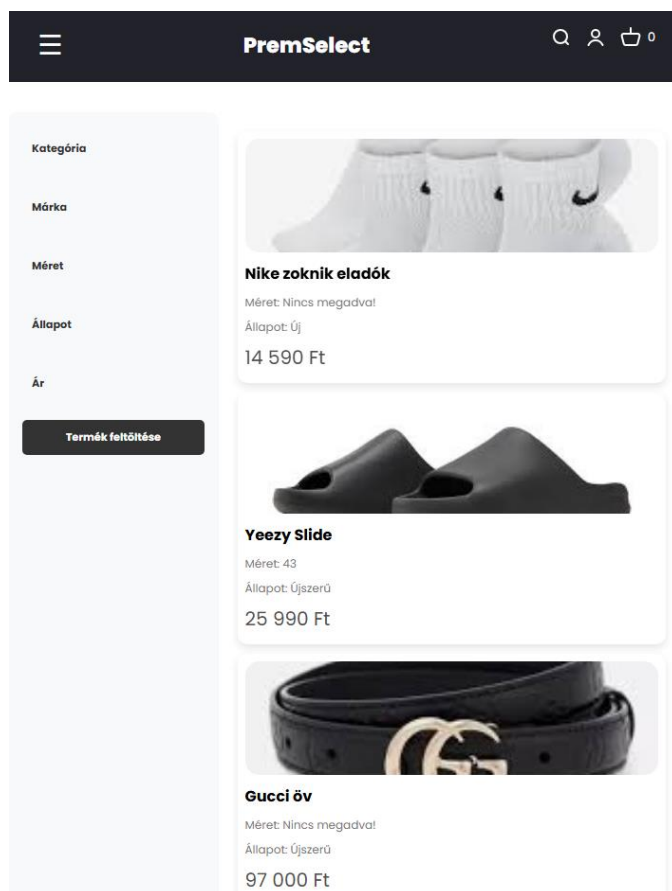
1.3. Posztok oldal bemutatása

Az "Összes termék" oldal a weboldal központi eleme, ahol a felhasználók megtekinthetik az összes feltöltött terméket, valamint lehetőséget találnak saját termékeik feltöltésére. Az oldalon az összes feltöltött termék listája látható. A termékekről készült képek, rövid leírások és árak alapján a felhasználók könnyen tájékozódhatnak a kínálatról. A termékekre kattintva további részletek és információk érhetők el. Saját termékek feltöltéséhez az oldalon található "Termék feltöltése" gombra kell kattintani. Ezután egy űrlap jelenik meg, ahol a termékre vonatkozó adatokat (név, leírás, ár, képek stb.) lehet megadni. A feltöltött termékek automatikusan megjelennek az "Összes termék" oldalon. Az oldal bal oldalán található menüben a termékek kategóriák szerint vannak csoportosítva. A kategóriákra kattintva a felhasználók szűkíthetik a keresést, és csak az adott kategóriába tartozó termékek jelennek meg. Az oldal tetején található navigációs sáv segítségével a felhasználók könnyen elérhetik a

weboldal más részeit, például a főoldalt, a felhasználói profilt vagy a kosarat.



4. ábra Összes termék asztali nézetben



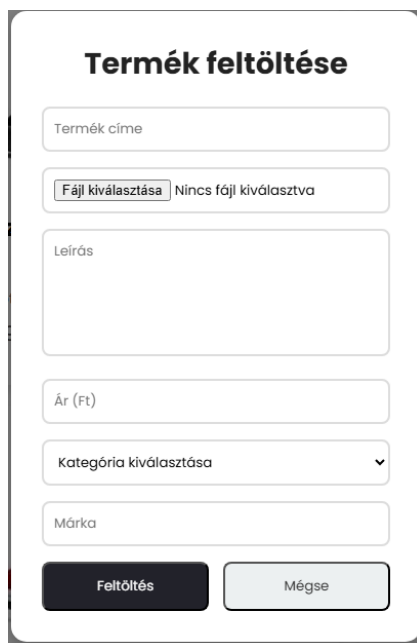
5. ábra Összes termék asztali nézetben

1.3.1. Posztok feltöltése

A Termék feltöltése űrlap segítségével új termékeket lehet hozzáadni a rendszerhez. Az első mezőbe a termék címét kell beírni, amely röviden és egyértelműen megnevezi a terméket. Ezt követően egy képfájl kiválasztására van lehetőség a Fájl kiválasztása gomb megnyomásával. A kiválasztott fájl neve megjelenik a gomb mellett. A kép feltöltése után a termék részletes leírását kell megadni, amely tartalmazhatja például a főbb jellemzőket, funkciókat vagy egyéb hasznos információkat.

A leírás kitöltését követően az ár megadása szükséges forintban, kizárólag szám formátumban. Ezután egy kategóriát kell kiválasztani a legördülő listából, amely meghatározza a termék típusát. A kategória kiválasztása után két új mező jelenik meg: a Méret és az Állapot kiválasztása. A méret megadására például ruházati termékeknél van szükség, míg az állapot mezőben azt lehet jelezni, hogy az adott termék új vagy használt állapotban van-e.

Végül a márka mező kitöltése következik, amelybe a termék gyártójának vagy forgalmazójának nevét kell beírni. Miután minden kötelező adatot megadtunk, a Feltöltés gombra kattintva lehet véglegesíteni és elmenteni a terméket a rendszerben. Amennyiben a feltöltést meg szeretnénk szakítani, a Mégse gomb segítségével elvethetjük a módosításokat, és az űrlap bezárul.



Termék feltöltése

Termék címe

Fájl kiválasztása Nincs fájl kiválasztva

Leírás

Ár (Ft)

Kategória kiválasztása ▼

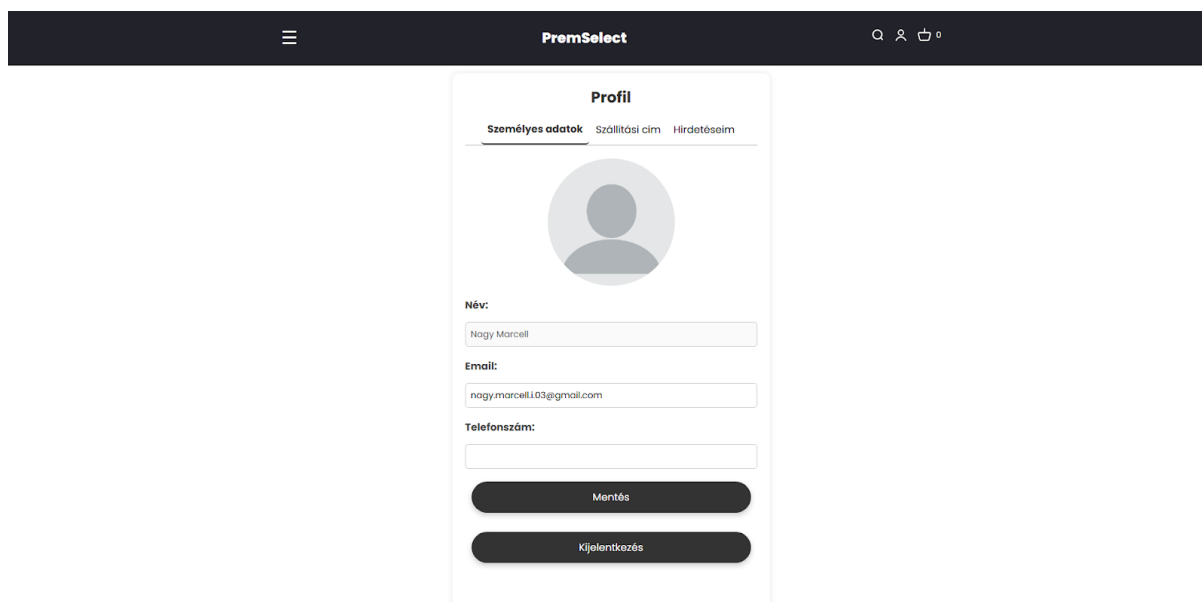
Márka

Feltöltés Mégse

6. ábra Posztok feltöltése

1.4. Profil oldal bemutatása

A profiloldal lehetőséget biztosít a felhasználók számára, hogy megtekintsék és módosítsák személyes adataikat, szállítási címüket és az általuk feltöltött termékeket. Az oldal három fő szekcióra van osztva: Személyes adatok, Szállítási cím és Hirdetéseim, amelyek között fülön keresztül lehet váltani. Ezen a felületen a felhasználók könnyedén szerkeszthetik adataikat, és a változtatásokat el is menthetik.



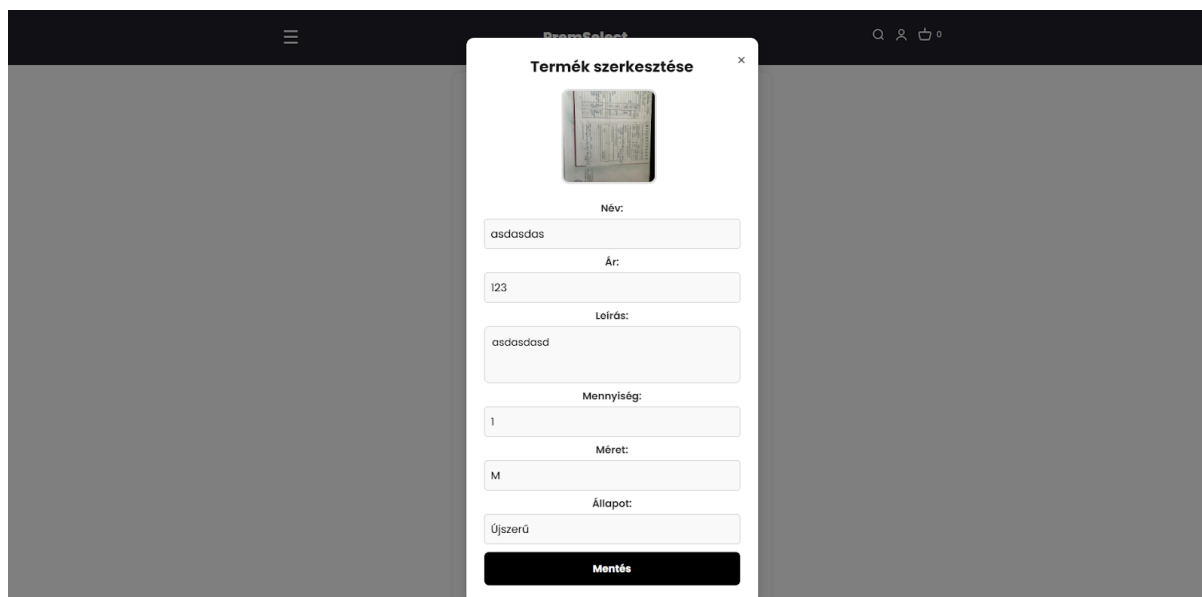
7. ábra Profil oldal asztali nézetben

1.4.1 Funkciók és használat

A Személyes adatok fül alatt a felhasználó megadhatja vagy módosíthatja az e-mail címét, telefonszámát és profilképét. Ehhez egyszerűen be kell írnia az új adatokat a megfelelő mezőkbe, majd a Mentés gombra kattintva elmentheti a módosításokat. Az e-mail cím és a telefonszám megadása elengedhetetlen lehet például a rendelési folyamat során, ezért fontos, hogy ezek az adatok pontosak legyenek.

A Szállítási cím fül alatt a felhasználó a rendeléseihez szükséges címadatokat kezelheti. Itt megadhatja az irányítószámot, a várost, a megyét, az utcanévét és a házszámot. Ha bármelyik adat megváltozik, a módosításokat a Mentés gomb megnyomásával rögzítheti a rendszerben, így biztosítva, hogy a csomagjai mindig a megfelelő címre érkezzenek.

A Hirdetéseim fül alatt a felhasználó láthatja az általa feltöltött termékeket. Két gomb jelenik meg, egy Szerkesztés és egy Törlés gomb. A törlés gombra kattintva a felhasználó azt a terméket kitörölheti. A Szerkesztés gombra kattintva a felhasználónak lehetősége van arra, hogy szerkesszen a termék adatait. Ha elégedett a felhasználó a változtatásokkal akkor a Mentés gombbal tudja menteni ezeket.



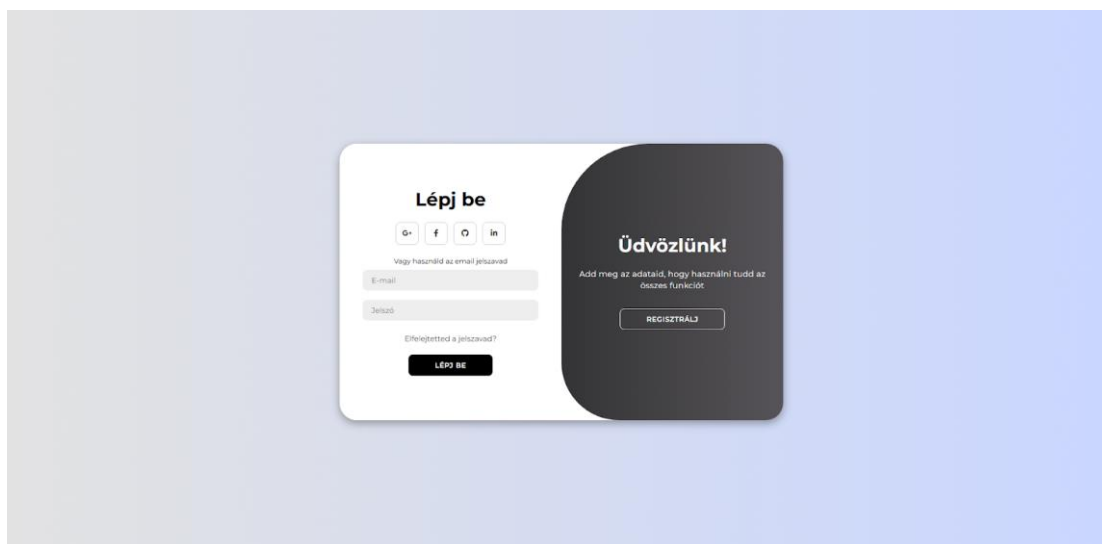
8. ábra Saját termék szerkesztése

A profiloldalon található egy Kijelentkezés gomb is, amely lehetőséget biztosít a felhasználóknak arra, hogy kilépjenek a fiókjukból. Ezzel a funkcióval biztonságosan befejezhetik a munkamenetüket, és elkerülhetik, hogy mások hozzáférjenek a személyes adataikhoz.

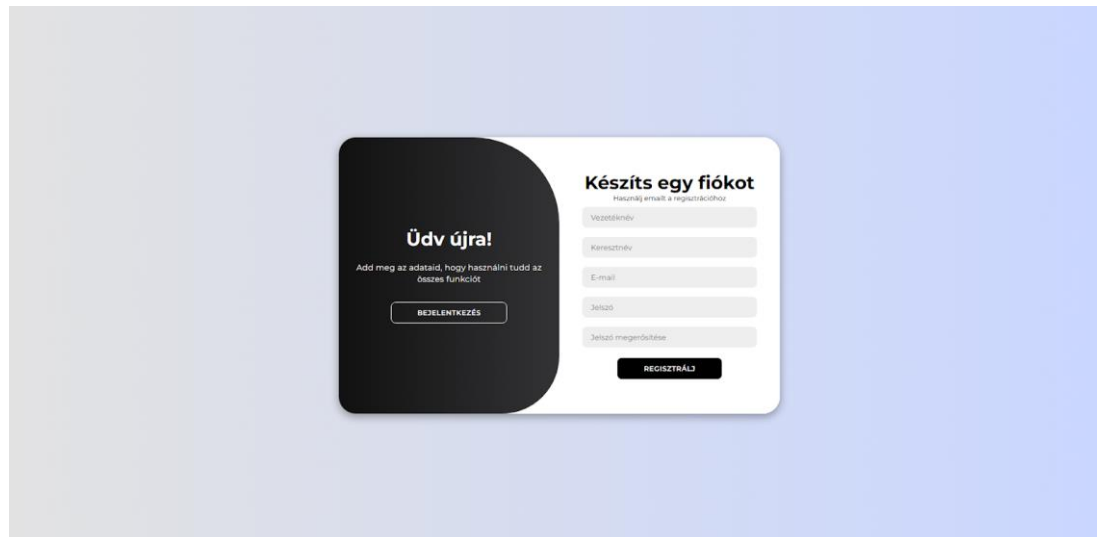
1.5. Bejelentkezés és regisztrációs oldal bemutatása

A PremSelect weboldal bejelentkezési és regisztrációs felülete a kezdőlapen található, és lehetőséget nyújt a felhasználóknak a weboldal funkcióinak elérésére. Amennyiben a felhasználó már rendelkezik fiókkal, a bejelentkezési űrlapon megadhatja e-mail címét és jelszavát, majd a "Lépj be" gombra kattintva bejelentkezhet. Ha a felhasználó még nem regisztrált, a "Regisztrálj" gombra kattintva új fiókot hozhat létre. A regisztrációs űrlapon a következő adatokat kell megadni: vezetéknév, keresztnév, e-mail cím, jelszó és jelszó megerősítése. A regisztrációt követően a felhasználó bejelentkezhet az újonnan létrehozott

fiókjával. Fontos, hogy a felhasználók érvényes e-mail címet adjanak meg a regisztráció során, és biztonságos jelszót válasszanak. A PremSelect weboldal bejelentkezési és regisztrációs felülete egyszerű és felhasználóbarát, így mindenki könnyen hozzáférhet a weboldal szolgáltatásaihoz.



9. ábra Bejelentkezés asztali nézetben

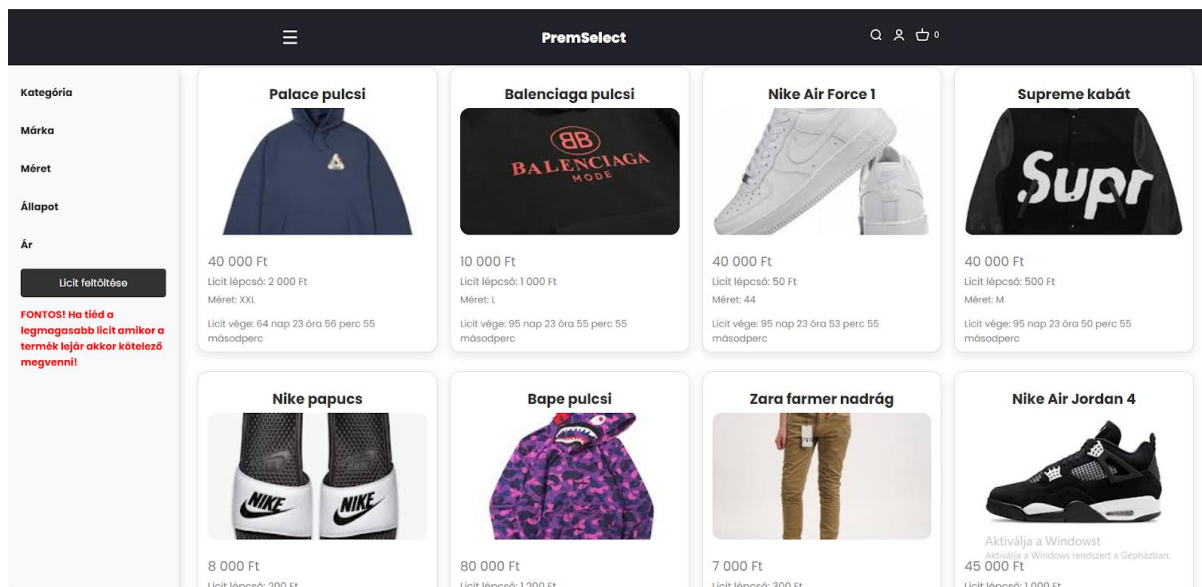


10. ábra Regisztráció asztali nézetben

1.6 Licit oldal bemutatása

A licit oldal lehetőséget biztosít a felhasználók számára, hogy könnyedén licitálhassanak különböző termékekre, valamint saját termékeiket is elhelyezhessék a piacon. Az oldal

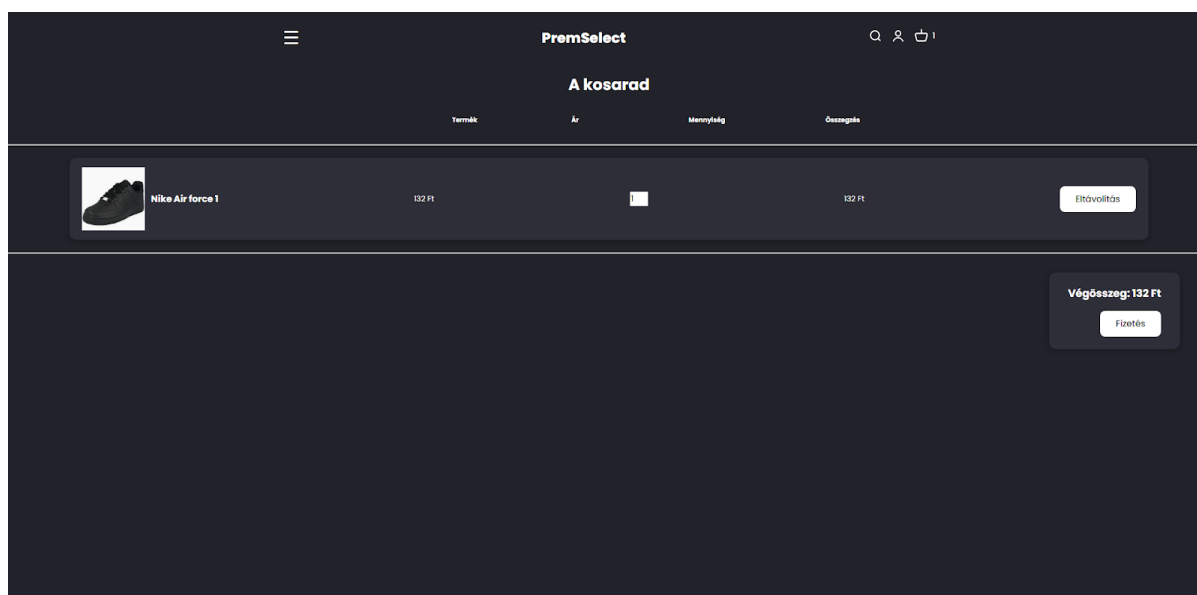
funkcionalitásai rugalmasan segítik mind a vásárlást, mind az értékesítést, ezáltal egy felhasználóbarát környezetet teremtve. A felhasználó a főmenüben található Licit gombra kattintva léphet be az oldalra, ahol azonnal áttekintheti az aktuálisan meghirdetett liciteket. Az oldal fő tartalmát az eddig feltöltött licitek listája alkotja, ahol minden egyes licitnél megjelenik a termék neve, a kezdő ár, a licitlépcső, a licit végének időpontja, a kategória, a márka, valamint a termék képe. Amennyiben egy adott termék felkelti a felhasználó figyelmét, rákattintva egy részletes nézet ugrik fel, amelyben a termékről minden fontos adat megjelenik, továbbá lehetőség van a termékkép kinagyítására is. Ha a felhasználó szeretne licitálni egy adott termékre, akkor a részletes nézetben található "Licit Emelése" gombra kattintva automatikusan megnöveli a licitet a megadott minimális lépcső értékével, az emelés sikerességéről pedig visszajelzést kap. A felhasználók nemcsak licitálhatnak, hanem saját termékeiket is licitre bocsáthatják. A feltöltési folyamat során meg kell adni a termék nevét, kezdőárát, licitlépcsőjét, kategóriáját, márkáját, a licit lejárat időpontját, valamint fel kell tölteni a termék képét. A feltöltés után a termék azonnal megjelenik a listában, ahol más felhasználók licitálhatnak rá. A felhasználók bármikor megtekinthetik az általuk feltöltött liciteket a "Saját liciteim" menüpontban, ahol listázva láthatják saját termékeiket és figyelemmel kísérhetik, hogyan alakul az árverés. A licit lejáratával a legmagasabb licitáló megnyeri a terméket, és az eladó kapcsolatba léphet vele az átadás megszervezése érdekében. A felhasználók kényelmét egy fejlett szűrőrendszer is szolgálja, amely lehetővé teszi a termékek közötti gyors keresést kategória, márka, ártartomány és licit lejárat idő szerint. Ez a funkció jelentősen megkönnyíti a keresést, és gyorsabbá teszi a megfelelő termék megtalálását. A licit oldal egy modern, felhasználóbarát piac, ahol bárki részt vehet az online aukciók világában. Legyen szó vásárlásról vagy eladásról, az oldal intuitív kezelése és fejlett funkcionalitása biztosítja a gyors és egyszerű licitálást. A termékfeltöltés, a licit figyelése, valamint a kényelmes keresés mind hozzájárulnak a sikeres felhasználói élményhez.



11. ábra Licit oldal, asztali nézetben

1.7. Kosár oldal bemutatása

A megvásárolni kívánt termékek a felhasználó személyes kosarába kerülnek, itt a kosár gombra kattintva a véglegesítés előtt lehetőség nyílik a termék/termékek megtekintésére és összegzésére (ez magában foglalja, hogy hány darab termék szerepel a kosárban valamint, hogy mennyi az áruk összesen). A fizetés gombra kattintva a felhasználó tovább léphet a fizetés oldalra.



12. ábra Kosár asztali nézetben

1.8. Fizetés oldal bemutatása

A fizetés oldalon a felhasználó megtekintheti a megvásárolni kívánt termékeket, de itt már nem módosíthatja ezeket. A szállítási adatokat megtudja nézni a felhasználó és módosítani tudja az általa kívánt adatokra. A szállítási adatok közé tartozik a név, e-mail cím, telefonszám, vármegye, irányítószám, város és az utca, házszám. Ezen adatok megadása kötelező és enélkül a felhasználó nem tudja véglegesíteni a megrendelését. Ha minden mező megvan adva a “Megrendelés leadás” gombra kattintva leadhatja rendelését.

The desktop view of the PremSelect checkout page features a dark header with a menu icon, the brand name 'PremSelect', and search, user, and cart icons. A progress bar shows '1. Kosár → 2. Fizetés → 3. Kész!'. The main content is split into two columns. The left column, titled 'Szállítási adatok', contains form fields for 'Teljes név:' (Konderla Sámuel), 'E-mail:' (samuelkonderla@gmail.com), 'Telefonszám:' (06203522651), 'Vármegye:' (Nógrád Vármegye), 'Irányítószám:' (2660), 'Város:' (Balassagyarmat), 'Cím:' (Kossúth utca 42.), 'Szállítási mód:' (Normál szállítás), and a checkbox for 'Számításhoz adatok megegyeznek a szállítással'. The right column, titled 'Rendelés összegzése', displays a table with 'Nike Air force 1' at 132 Ft and 'Szállítás' at 1 000 Ft, totaling 'Végösszeg 1 132 Ft'. A 'Megrendelés leadása' button and a '100% biztonságos fizetés' badge are at the bottom.

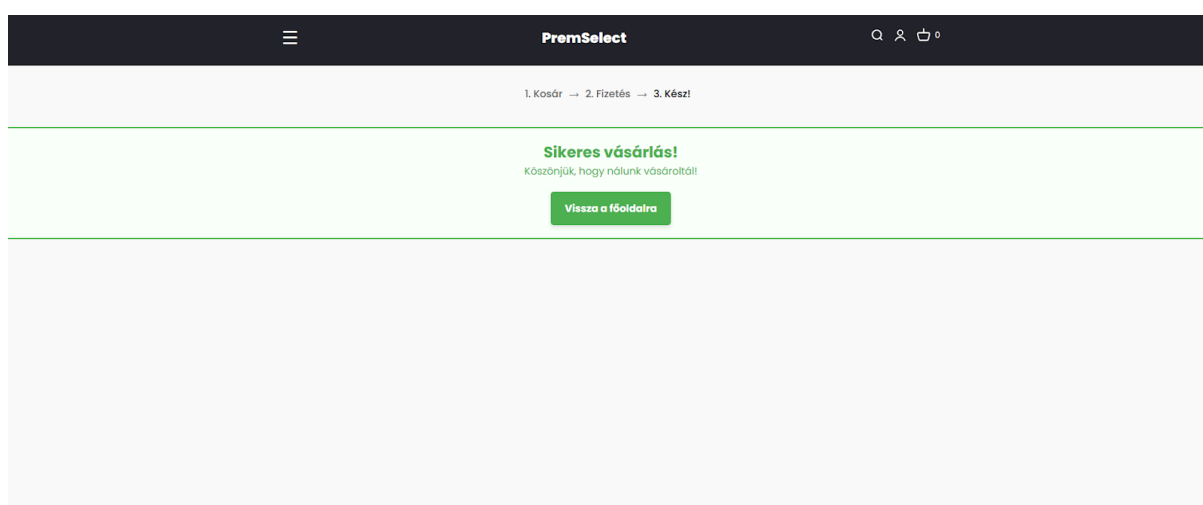
13. ábra Fizetés asztali nézetben

The mobile view of the PremSelect checkout page has a similar dark header. The progress bar shows '1. Kosár → 2. Fizetés → 3. Kész!'. The 'Szállítási adatok' form fields are stacked vertically: 'Teljes név:' (Konderla Sámuel), 'E-mail:' (samuelkonderla@gmail.com), 'Telefonszám:' (06203522651), 'Vármegye:' (Nógrád Vármegye), 'Irányítószám:' (2660), 'Város:' (Balassagyarmat), 'Cím:' (Kossúth utca 42.), 'Szállítási mód:' (Normál szállítás), and the checkbox for 'Számításhoz adatok megegyeznek a szállítással'.

14. ábra Fizetés telefonos nézetben

1.8.1 Sikeres megrendelés

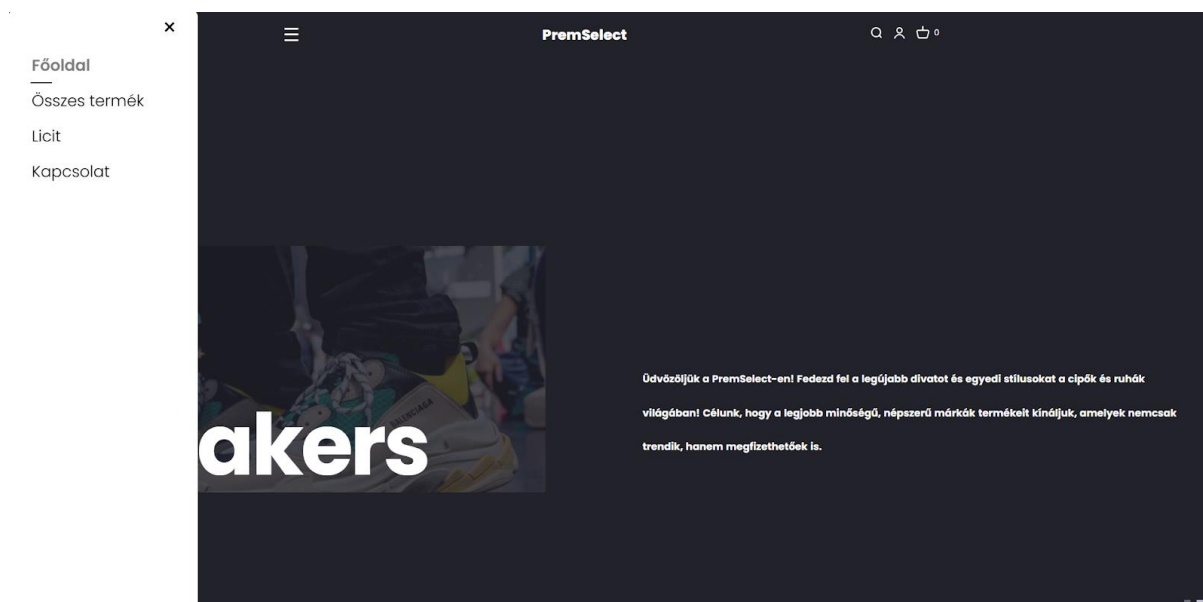
A sikeres megrendelés esetén a “Megrendelés gombra” kattintva megjelenik egy “Sikeres vásárlás” ablak, ami felkínálja a felhasználónak, hogy a visszatérjen a főoldalra, amit a “Vissza a főoldalra” gombra kattintva teheti meg.



15. ábra sikeres fizetés, asztali nézetben

1.9. Menü

A Menü gombra kattintva megjelenik a navigációs menü, amely lehetőséget biztosít a felhasználóknak, hogy könnyedén navigáljanak az oldal különböző szekciói között. A menü négy fő pontot tartalmaz: az első a Főoldal, amely az oldal alapvető tartalmát és információit mutatja be. A második a Össztermék oldal, ahol az összes elérhető termék listáját találhatják a felhasználók. A harmadik pont a Licit oldal, ahol a felhasználók nyomon követhetik a futó liciteket és részt vehetnek az aukciókon. Végül a Kapcsolat oldal található, ahol az érdeklődők elérhetik a weboldal üzemeltetőit és kapcsolatba léphetnek velük. A menü segít abban, hogy a felhasználók könnyen és gyorsan elérjék a kívánt oldalakat.



16. ábra A menü, asztali nézetben

1.10 Lapozó

Az oldalon két fő rész áll a felhasználók rendelkezésére: a Licit oldal, ahol különböző termékekre licitálhatnak, valamint az Összes termék oldal, ahol a teljes kínálatot böngészhetik. Ahhoz, hogy a sok termék között könnyedén eligazodjanak, az oldal alján, középen egy praktikus lapozó (paginátor) segíti őket. A lapozó szerepe, hogy átláthatóbbá tegye a böngészést: nem kell egyszerre minden terméket egy oldalon betölteni, hanem oldalakra bontva jelennek meg az ajánlatok. A felhasználó egyszerűen tud lépkedni az oldalak között, így gyorsabban találhatja meg a számára érdekes termékeket vagy liciteket. A lapozóval könnyen navigálhat előre és hátra, illetve közvetlenül is kiválaszthat egy adott oldalszámot. Ennek köszönhetően nincs szükség arra, hogy hosszasan görgessen egy végtelen listában, hiszen a kínálat kisebb, kezelhetőbb egységekben jelenik meg. Mind a Licit oldalon, mind az Összes termék oldalon világosan látszik, hányadik oldalon jár, és hogy mennyi oldal közül választhat. Az oldal kialakítása biztosítja, hogy a lapozás gördülékeny legyen, így a böngészés során nem kell újratöltésre várnia, minden gyorsan és folyamatosan működik. Ha a felhasználó például a licitálható termékek között keresgél, de az első oldalon nem talál számára megfelelő ajánlatot, néhány kattintással könnyedén továbbléphet a következő oldalra. Ugyanez igaz az összes termék böngészésekor is: a lapozó segítségével gyorsan és kényelmesen haladhat tovább, így mindig kézben tarthatja a keresést. A lapozó tehát egyszerűbbé, gyorsabbá és élvezetesebbé teszi az oldalon való eligazodást és vásárlást.

2. Fejlesztői dokumentáció

A vizsgaremek mappaszerkezete viszonylag egyszerű, de hasznos és átlátható ami kulcsfontosságú a fájlok közötti eligazodáshoz. A stíluslapokat egy “css” nevű mappában tároljuk, a JavaScript fájlokat pedig egy “js” nevű mappában helyeztük el.

2.1. Felhasznált technológiák

2.1.1. Visual Studio Code

A Visual Studio Code (VSCode) egy ingyenes, nyílt forráskódú kódszerkesztő, amelyet a Microsoft fejlesztett. A program számos programozási nyelvet támogat, például JavaScript, Python, C++, PHP, Java és sok más nyelvet, és széleskörű bővítményekkel bővíthető. A VSCode kiemelkedik gyorsaságával, testreszabhatóságával és könnyű használhatóságával. Tartalmaz intelligens kódkiegészítést, hibakeresést, beépített terminált, verziókezelést (Git integrációval), valamint a kódformázást és refaktorálást is. A platform széleskörű fejlesztői közösséggel rendelkezik, és folyamatosan bővül új funkciókkal és bővítményekkel. A VSCode tökéletes választás mind kezdő, mind tapasztalt fejlesztők számára a különböző programozási projektekhez.

2.1.2. HTML

A HTML (Hypertext Markup Language) a weboldalak alapvető építőköve, egy jelölőnyelv, amely a webes tartalom struktúráját határozza meg. A HTML dokumentumok alapja a címkék, amelyek meghatározzák, hogyan jelenjenek meg az egyes elemek a böngészőben. A HTML segítségével hozhatók létre különböző tartalmi elemek, mint például szövegek, képek, hivatkozások, táblázatok, űrlapok és médiafájlok.

A HTML alapvetően statikus tartalmat biztosít, de más technológiák, mint például a CSS (Cascading Style Sheets) és a JavaScript, segítenek a megjelenés és az interaktivitás fokozásában. A HTML5, a HTML legújabb verziója, számos új elemet és API-t kínál, mint például a video és audio lejátszókat, valamint javítja az alkalmazásfejlesztési lehetőségeket, például a helymeghatározást és a webes tárolást.

2.1.3. CSS

A CSS (Cascading Style Sheets) egy stíluslapnyelv, amely a weboldalak megjelenését és formázását határozza meg. A HTML-ben meghatározott tartalom megjelenését szabályozza, például a színek, betűtípusok, margók, elrendezések és animációk formázását. A CSS segítségével elválasztható a tartalom és a stílus, így tisztább, átláthatóbb és könnyebben karbantartható kódot hozhatunk létre.

A CSS három fő módon alkalmazható: inline (HTML elemen belül), internal (a HTML dokumentumban lévő <style> szakaszban) és external (különálló .css fájlként, amelyet a HTML dokumentum linkel). A CSS3, a CSS legújabb verziója, számos új lehetőséget kínál, mint például a rugalmas elrendezések (Flexbox, Grid), animációk, átmenetek és egyéb modern vizuális effektusok. A CSS segítségével reszponzív, felhasználóbarát weboldalak készíthetők, amelyek különböző eszközökön is megfelelően jelennek meg.

2.1.4. Php

A PHP (Hypertext Preprocessor) egy széles körben használt, nyílt forráskódú szerveroldali szkriptnyelv, amelyet elsősorban dinamikus weboldalak és webalkalmazások fejlesztésére alkalmaznak. Beágyazható HTML-kódba, és számos adatbáziskezelő rendszerrel, például MySQL-lel vagy PostgreSQL-lel is könnyedén integrálható. A PHP erőforráshatékony, platformfüggetlen és kiterjedt könyvtárkészlettel rendelkezik, amely megkönnyíti a webfejlesztést. Modern verziói támogatják az objektumorientált programozást és a különböző biztonsági mechanizmusokat, így alkalmas komplex rendszerek létrehozására is.

2.1.5. JavaScript

A JavaScript egy magas szintű, dinamikus programozási nyelv, amelyet elsősorban interaktív weboldalak és webalkalmazások fejlesztésére használnak. A böngészőoldali futtatás mellett a Node.js segítségével szerveroldalon is alkalmazható. Támogatja az eseményvezérelt és aszinkron programozást, valamint könnyen integrálható más technológiákkal, például HTML és CSS elemekkel. Széleskörű könyvtár- és keretrendszer-támogatása (például React, Vue, Angular) lehetővé teszi modern, reszponzív és skálázható webes megoldások fejlesztését.

2.1.6. Composer

A Composer egy széles körben használt függőségkezelő eszköz a PHP-hoz, amely megkönnyíti a külső csomagok és könyvtárak kezelését. Lehetővé teszi a fejlesztők számára, hogy egyszerűen telepítsenek és frissítsenek különböző PHP-s modulokat, miközben automatikusan kezeli azok függőségeit. A **composer.json** fájl segítségével deklarálhatók a projektben használt csomagok, a **Packagist** pedig a legnagyobb nyilvános csomagtárként szolgál. A Composer hozzájárul a hatékonyabb és modulárisabb PHP-fejlesztéshez, különösen a modern keretrendszerekkel, például a Laravel vagy Symfony esetén.

2.1.7. MySQL

A MySQL egy nyílt forráskódú, relációs adatbázis-kezelő rendszer (RDBMS), amelyet széles körben használnak webalkalmazások és üzleti rendszerek adatkezelésére. A Structured Query Language (SQL) segítségével lehetővé teszi az adatok hatékony tárolását, lekérdezését és kezelését. Gyorsasága, megbízhatósága és skálázhatósága miatt népszerű választás különböző platformokon, legyen szó kis weboldalakról vagy nagyvállalati rendszerekről. Gyakran használják PHP-val együtt a LAMP (Linux, Apache, MySQL, PHP) környezetben, és támogatja az olyan fejlett funkciókat, mint a tranzakciókezelés, replikáció és indexelés.

2.1.8. Xampp

A XAMPP egy ingyenes, nyílt forráskódú szoftvercsomag, amely egy teljes környezetet biztosít a webfejlesztéshez. Tartalmazza az Apache webszervert, a MySQL adatbázis-kezelőt (újabb verziókban MariaDB-t), valamint a PHP és Perl programozási nyelveket. A XAMPP segítségével a fejlesztők egyszerűen futtathatnak és tesztelhetnek webalkalmazásokat saját számítógépükön, anélkül hogy külön konfigurálnák az egyes komponenseket. Több operációs rendszeren (Windows, Linux, macOS) is elérhető, és egy beépített kezelőfelületet biztosít a szerverelemek könnyű irányításához.

2.1.9. GitHub

A GitHub egy webalapú platform, amely a Git verziókezelő rendszerre épít, és lehetővé teszi a fejlesztők számára, hogy közösen dolgozzanak projekteken, megoszthassák és kezelhessék kódjukat. A GitHub biztosítja a távoli tárolók (repositoryk) létrehozását, ahol a kód verziói tárolhatók, és a csapatok könnyen nyomon követhetik a változtatásokat, végezhetnek kódellenőrzéseket és integrálhatják a módosításokat. Emellett támogatja a nyílt forráskódú projekteket, és a közösségi együttműködés elősegítése érdekében lehetőséget ad a hibajegyek (issues), pull requestek és a kód

megvitatására. A GitHub emellett lehetővé teszi a folyamatos integrációt (CI/CD), dokumentációk kezelését és különféle fejlesztői eszközökkel való integrációt is.

2.1.9. DbDiagram

A dbdiagram.io egy online eszköz, amely lehetővé teszi adatbázis-diagramok gyors és egyszerű létrehozását. Az alkalmazás támogatja az ER-diagramok (Entity-Relationship) generálását, amelyeket adatbázisok tervezésére és dokumentálására használnak. A dbdiagram.io különböző adatbázis-kezelő rendszerekkel (pl. MySQL, PostgreSQL, SQLite) kompatibilis, és lehetőséget biztosít a diagramok exportálására SQL kód formájában, amely közvetlenül használható az adatbázisok létrehozásához. A felhasználók vizuálisan is ábrázolhatják az adatbázisok tábláit, kapcsolataikat, valamint a mezők típusait és kulcsait, így a fejlesztők könnyebben megérthetik és kommunikálhatják az adatbázis-struktúrákat a csapatukon belül.

2.1.10. Microsoft Word

A Microsoft Word egy széles körben használt szövegszerkesztő alkalmazás, amelyet a Microsoft fejlesztett ki. A program lehetővé teszi a felhasználók számára, hogy különféle dokumentumokat hozzanak létre, formázzanak, szerkesszenek és nyomtassanak, például leveleket, jelentéseket, önéletrajzokat és egyéb írásos anyagokat. A Word számos eszközt és funkciót kínál a szövegformázás, a képek és grafikonok beillesztése, táblázatok létrehozása, valamint az oldalak elrendezése terén. Emellett támogatja az együttműködést és a valós idejű szerkesztést, ha a dokumentumokat a felhőben, például OneDrive-on tárolják. A Word formátumai, mint a .doc és .docx, az iparági szabványok közé tartoznak, és széles körben támogatottak más szövegszerkesztők által is.

2.1.11. Discord

A Discord egy ingyenes, multimédiás kommunikációs platform, amelyet elsősorban játékosok és közösségek számára fejlesztettek, de ma már széleskörűen használják különböző érdeklődési körű csoportok, tanulók, munkacsoportok és vállalkozások is. A Discord lehetővé teszi a felhasználók számára, hogy szöveges, hang- és videóchatet folytassanak, valamint könnyedén megoszthassák a képeket, fájlokat és egyéb médiatartalmakat. A platform szerverekre épül, ahol a felhasználók különböző csatornákon kommunikálhatnak, amelyek lehetnek szövegesek vagy hangalapúak. A Discord emellett botokat és egyéb integrációkat is támogat, amelyek automatizálhatják a folyamatokat, például moderálást vagy értesítéseket. A platform elérhető asztali és mobil alkalmazásként, valamint böngészőn keresztül is.

2.1.12 Trello

A Trello egy könnyen használható online eszköz, amely segít a feladatok és projektek átlátható kezelésében. Táblák, listák és kártyák segítségével rendszerezhetjük a teendőinket, így mindig pontosan látható, milyen feladatok vannak hátra, min dolgoznak a csapattagok, és mi készült már el. A Trello különösen hasznos csapatmunkához, mivel lehetőséget ad a feladatok megosztására, felelősök kijelölésére, határidők beállítására és értesítések küldésére. Az egyszerű kezelőfelületnek köszönhetően gyorsan megtanulható, és rugalmasan használható bármilyen munkafolyamathoz, legyen szó egyéni teendőkről, csoportos projektekről vagy nagyobb céges folyamatokról. Emellett a Trello integrálható más népszerű alkalmazásokkal, például a Google Drive-val, a Slackkel vagy a GitHubbal, így még hatékonyabbá tehető a munka.

2.1.13 EchoApi

Az EchoAPI Visual Studio Code bővítmény lehetővé teszi API-k gyors tesztelését közvetlenül a szerkesztőben. HTTP kéréseket (GET, POST, PUT, DELETE) küldhetünk és a válaszokat, például JSON adatokat, könnyen megtekinthetjük. A bővítmény támogatja a fejlécek és az autentikáció kezelését is, így egyszerűsíti az API fejlesztését és tesztelését anélkül, hogy külön eszközt kellene használni.

2.2. Technológiák áttekintése

2.2.1 PHP főbb jellemzői

A PHP egy szerveroldali programozási nyelv, amit leginkább weboldalak fejlesztésére használnak. A lényege, hogy a kód a szerveren fut le, és a felhasználók a böngészőjükben már csak az elkészült HTML-t látják. Ez azért előnyös, mert így könnyen tudunk dinamikus oldalakat készíteni, ahol az adatok az adatbázisból töltődnek be, vagy a felhasználók által megadott információk alapján változnak.

A PHP nagyon sok adatbázist képes kezelni, például a MySQL-t, amit mi is használtunk a projektünk során, a MySQLi bővítménnyel együtt. Előnye, hogy Windows, Linux és macOS rendszereken is működik, így bármilyen fejlesztőkörnyezetet be tudunk állítani magunknak.

A PHP nyílt forráskódú, vagyis ingyenesen elérhető, és a közösség folyamatosan fejleszti új funkciókkal és hibajavításokkal. A nyelvben rengeteg beépített függvény található, amik megkönnyítették a munkánkat, és ha kellett, külső könyvtárakat is könnyen be tudtunk vonni. A PHP biztonsági megoldásai, például az adatellenőrzés vagy az SQL injekciók elleni védelem, szintén segítettek abban, hogy megbízhatóbbá tegyük a weboldalunkat.

A vizsgaremek során a PHP-t a háttérrendszer megvalósítására, adatbázis-műveletek kezelésére, API-k létrehozására és az adatok dinamikus feldolgozására használtuk.

2.2.2. HTML, CSS, JavaScript rövid bemutatása

A projektünk frontend részét HTML, CSS és JavaScript technológiák segítségével készítettük el. Ezek az alapok elengedhetetlenek ahhoz, hogy modern, jól működő weboldalt hozzunk létre, amely egyszerre esztétikus és könnyen kezelhető.

A HTML (HyperText Markup Language) a weboldal szerkezetét adja meg. Ezzel a nyelvvel határoztuk meg az oldalunk elemeit, például a címsorokat, bekezdéseket, űrlapokat és gombokat. A HTML-t arra használtuk, hogy az oldal tartalmát logikusan és átláthatóan felépítsük, így a felhasználók könnyen eligazodhattak a weboldalon.

A CSS (Cascading Style Sheets) segítségével formáztuk meg a HTML-elemeket. A CSS felelős az oldal kinézetéért: a színeket, betűtípusokat, méreteket, elrendezéseket és animációkat ezzel állítottuk be. Célunk az volt, hogy a weboldal ne csak funkcionálisan legyen jó, hanem modern megjelenésű és felhasználóbarát is legyen. Külön figyeltünk arra, hogy az oldal reszponzív legyen, tehát mobilon és nagyobb képernyőkön is jól nézzen ki.

A JavaScript-et arra használtuk, hogy a weboldalunk interaktív legyen. Ezzel a nyelvvel kezeltük például a gombok eseményeit, az adatok betöltését az API-ból, és a felhasználói műveletekre adott reakciókat. A JavaScript-nek köszönhetően tudtunk dinamikus tartalmat frissíteni anélkül, hogy az egész oldalt újra kellett volna tölteni. Ezen kívül a fetch API segítségével küldtünk kéréseket a szerver felé, amivel egyszerűen tudtunk adatokat lekérni vagy feltölteni.

Összességében a HTML, a CSS és a JavaScript együttműködésével sikerült egy olyan webalkalmazást készítenünk, amely jól néz ki, könnyen használható, és dinamikusan képes kommunikálni a szerverrel.

2.3 Weboldal elérése

A weboldalunk eléréséhez egy helyi fejlesztői környezetet használtunk, ahol a szerver futtatása után a böngészőben lehetett megnyitni a projektet. A projekt gyökérkönyvtárát egy webszerver `htdocs` mappájába helyeztük el, majd a `localhost` címen keresztül tudtuk elérni az oldalt.

A webalkalmazás így például a `http://localhost/Szakdoga` címen volt megnyitható. Ha valaki éles környezetben szeretné használni a weboldalt, akkor szükség van egy publikus webszerverre, például egy tárhelyszolgáltatóra, ahova a fájlokat fel kell tölteni, és megfelelően be kell állítani az adatbázis kapcsolatot is.

Fejlesztés közben mindig figyeltünk arra, hogy az oldalt többféle böngészőben (például Chrome-ban és Firefoxban) is teszteljük, hogy minden funkció megfelelően működjön.

2.4. Projekt telepítése, és a webshop elindítása

A weboldal telepítéséhez és rendeltetésszerű használatához szükség volt néhány programra. Ezek közé tartozott a PHP, a Git, egy MySQL szerver (mi a XAMPP-ot használtuk), valamint a Composer csomagkezelő.

Miután minden szükséges programot telepítettünk, a következő lépés az volt, hogy magát a weboldalt is letöltsük. Ezt egy egyszerű parancs segítségével oldottuk meg, amit bármilyen parancssor-kezelő terminálban be lehet írni:

```
git clone https://github.com/Marci02/Szakdoga.git
```

A parancs lefuttatása után a weboldal összes szükséges fájlja letöltődött az aktuális mappába. Ezután beléptünk a letöltött projekt mappájába, majd telepítettük a Composer-rel az összes szükséges modult is, a következő paranccsal: `composer install`

Miután a Composer telepítette az összes szükséges PHP csomagot, következett az adatbázis beállítása. A projekt működéséhez szükség van egy MySQL adatbázisra, amely tartalmazza a

szükséges táblákat és adatokat. Az adatbázis importálásához először elindítottuk a XAMPP-ot, és biztosítottuk, hogy a MySQL szerver fut. Ezután beléptünk a phpMyAdmin felületre, amely általában a `http://localhost/phpmyadmin` címen érhető el. Az adatbázis létrehozásához új adatbázist hoztunk létre, például `tesztAdat` néven, majd az adatbázis importálásához a phpMyAdmin-ban az „Importálás” fülre kattintottunk, és kiválasztottuk a projekthez tartozó SQL fájlt, amely tartalmazza az összes szükséges adatbázis struktúrát és táblát. Az SQL fájl importálása után az adatbázis a megfelelő táblákkal és adatokkal felállt, így a projekt működéséhez szükséges adatokat is tartalmazza.

Ha ez is elkészült, akkor a projekt készen állt a használatra. A XAMPP segítségével indítottuk a szerveret, és a böngészőben a `http://localhost/Szakdog` címen tudtuk megnyitni a webalkalmazást.

2.5. Környezeti változók

A projektünkben a környezeti változókat a `.env` fájlban tároltuk, hogy a fontosabb konfigurációs adatokat (például adatbázis kapcsolódási információkat) könnyen kezelhessük, és biztonságosan elrejtessük a forráskódtól.

A `.env` fájlunk tartalmazza az adatbázis kapcsolat adatait, mint például a következőket:

```
DBHOST=localhost
```

```
DBUSER=root
```

```
DBPASS=
```

```
DBNAME=tesztAdat
```

A környezeti változók betöltésére a `vlucas/phpdotenv` csomagot használtuk, amit a Composer segítségével telepítettünk. A projekt gyökérkönyvtárában létrehoztunk egy `.env` fájlt és abban tároltuk az adatbázishoz szükséges adatokat. A `.env` fájl használatával elértük, hogy az érzékeny adatokat (pl. adatbázis jelszó) ne tároljuk közvetlenül a forráskódban.

A `connect.php` fájlban a következő kóddal olvastuk be ezeket a változókat és hoztuk létre az adatbázis kapcsolatot:

```
require_once __DIR__ . '/vendor/autoload.php';

$dotenv = Dotenv\Dotenv::createImmutable(__DIR__);

$dotenv->load();

define("DBHOST", $_ENV['DBHOST']);

define("DBUSER", $_ENV['DBUSER']);

define("DBPASS", $_ENV['DBPASS']);

define("DBNAME", $_ENV['DBNAME']);

$dbconn = @mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME) or die("Hiba az
adatbázis csatlakozásakor!");

mysqli_query($dbconn, "SET NAMES utf8");
```

Ez a módszer biztosította, hogy a projekt különböző környezetekben (fejlesztés, tesztelés, éles környezet) is egyszerűen beállítható legyen, anélkül hogy a kódban változtatásokat kellett volna végezni.

2.6. Útvonalak

A projektben a routing megoldása natív HTML és PHP fájlokkal történt, keretrendszer használata nélkül. Minden oldal alapvetően egy statikus .html fájlként érhető el a böngészőből, amelyhez szorosan kapcsolódik legalább egy PHP fájl is, amely a háttérben végzi az adatok feldolgozását, az adatbázis műveleteket vagy más szerveroldali feladatokat. A kezdőlapot az index.html fájl tölti be, ahonnan a felhasználók továbbléphetnek a weboldal különböző funkcióihoz. A bejelentkezési folyamat a login1.html oldalon zajlik, ahol a beküldött adatok feldolgozását a login1.php fájl kezeli. A kapcsolatfelvételi űrlapot a kapcsolat.html oldal jeleníti meg, az üzenetküldést pedig a kapcsolatsend.php fájl valósítja meg a PHPMailer könyvtár segítségével. Az aukciókat a licit.html oldal mutatja be, a licitálással kapcsolatos szerveroldali feldolgozást pedig a licit.php végzi. A vásárlási folyamat a cart.html (kosár) és a checkout.html

(pénztár) oldalakon keresztül történik, ahol szintén külön PHP fájlok felelősek a beküldött adatok mentéséért és feldolgozásáért.

A projekt mappastruktúrája logikusan lett kialakítva: a frontendhez tartozó fájlok, például a CSS, JavaScript és képek a `css`, `js` és `img` mappákban találhatók, míg a szerveroldali működéshez szükséges PHP állományok a backend könyvtárban kaptak helyet. A szerveroldali adatbázis kapcsolatot minden PHP fájl egységesen a `connect.php` betöltésével valósítja meg, amely a projekt gyökérkönyvtárában található, és a környezeti változókat a `.env` fájlból olvassa ki a `vlucas/phpdotenv` csomag segítségével. A PHPMailer könyvtár a `phpmailer` mappában helyezkedik el, amely az üzenetküldéssel kapcsolatos funkciókat szolgálja ki. A felhasználó minden funkciót közvetlenül a megfelelő `.html` fájl megnyitásával érhet el a böngészőből, például a <http://localhost/Szakdoga/kapcsolat.html>, <http://localhost/Szakdoga/login1.html> vagy a <http://localhost/Szakdoga/osszetermek.html> címen keresztül.

2.7. Backend felépítése

A projektünk backend részét teljes egészében natív PHP nyelven készítettük el, külső keretrendszerek használata nélkül. A backend elsődleges feladata az volt, hogy kapcsolatot tartson fent az adatbázissal, kezelje a felhasználók által beküldött adatokat, feldolgozza az űrlapokat, valamint megfelelő válaszokat adjon a frontend felől érkező kérésekre. Az adatbázis-kezelést minden fájlban központilag a `connect.php` állomány segítségével oldottuk meg, amely egy előre elkészített `.env` fájlból olvassa ki az adatbázis-csatlakozási adatokat a `vlucas/phpdotenv` csomag használatával.

A különböző funkciókhoz külön PHP fájlokat hoztunk létre, hogy a kód átlátható és könnyen karbantartható legyen. Például a `login1.php` fájl a bejelentkezési űrlap adatait dolgozza fel, ellenőrzi az adatbázisban lévő felhasználókat, és kezeli a sikeres vagy sikertelen belépéseket. A `kapcsolatsend.php` fájl a kapcsolatfelvételi űrlapról érkező üzeneteket dolgozza fel és PHPMailer segítségével továbbítja azokat emailben. Az egyes aukciós licitek feldolgozásához a `licit.php` fájlt készítettük, amely biztonságosan menti a felhasználók ajánlatait az adatbázisba. A kosár és pénztár funkciókhoz tartozó adatok kezelését pedig szintén külön PHP fájlok valósítják meg.

Az API működéséhez is írtunk backend oldali kódokat, amelyek JSON formátumban adnak vissza adatokat. Ezeket a PHP fájlokat úgy készítettük el, hogy a válaszok megfelelő fejléc beállítással, `application/json` tartalomtípussal érkezzenek, biztosítva az adatok helyes feldolgozását a JavaScript oldalon. Törekedtünk a biztonságos adatkezelésre is, ezért minden bemeneti adatot megfelelően szűrünk és validálunk, valamint minden adatbázis művelet során karakterkódolást állítunk be az ékezetes karakterek helyes kezelése érdekében.

A backend fejlesztése során figyeltünk arra is, hogy a kódjaink modulárisak legyenek, vagyis minden funkció elkülönítve működjön, így a későbbi bővítések vagy módosítások könnyebben elvégezhetők.

2.7.1. Felhasználói profil kezelése

A felhasználói profil kezelésének célja, hogy lehetővé tegye a felhasználók számára a saját adataik elérését, módosítását és törlését, miközben biztosítjuk azok védelmét és integritását. Első lépésként a rendszer ellenőrzi, hogy a felhasználó be van-e jelentkezve. Ehhez az `$_SESSION['user_id']` ellenőrzése történik. Ha a session-ban szerepel a felhasználó azonosítója, akkor a rendszer visszaadja azt JSON formátumban, jelezve, hogy a felhasználó be van jelentkezve. Ha nincs aktív session, akkor egy hibaüzenetet küld vissza a rendszer:

```
<?php
if (isset($_SESSION['user_id'])) {
    echo json_encode(["success" => true, "user_id" => $_SESSION['user_id']]);
} else {
    echo json_encode(["success" => false, "message" => "Nincs bejelentkezve."]);
}
```

A felhasználói profil törlésére akkor van szükség, ha a felhasználó véglegesen szeretné eltávolítani a saját adatait a rendszerből. A profil törlésének folyamata magában foglalja a felhasználóhoz kapcsolódó termékek eltávolítását, majd a felhasználói fiók adatainak törlését. Ennek érdekében először a rendszer törli a termékeket, majd eltávolítja a felhasználót az adatbázisból. Ezt követően a rendszer biztosítja, hogy a felhasználó adatai teljesen eltávolításra

kerüljenek. A következő kódrészlet törli a felhasználóhoz kapcsolódó termékeket és a felhasználói adatokat az adatbázisból:

```
<?php
$deleteProductsStmt = $dbconn->prepare("DELETE FROM products WHERE user_id = ?");
$deleteProductsStmt->bind_param("i", $userId);
$deleteProductsStmt->execute();
$deleteUserStmt = $dbconn->prepare("DELETE FROM user WHERE id = ?");
$deleteUserStmt->bind_param("i", $userId);
$deleteUserStmt->execute();
```

Amikor a felhasználó szeretné megtekinteni a profilját, a rendszer lekérdezi a felhasználó adatokat az adatbázisból, és visszaadja őket JSON formátumban. A lekérdezés során az `$_SESSION['user_id']` segítségével azonosítjuk a bejelentkezett felhasználót, és lekérjük annak nevét, e-mail címét és egyéb adatokat. A következő kódrészlet végzi el a felhasználó adatainak lekérdezését:

```
<?php
$query = "SELECT id, name, email FROM user WHERE id = ?";
$stmt = $dbconn->prepare($query);
$stmt->bind_param("i", $_SESSION['user_id']);
$stmt->execute();
$result = $stmt->get_result();
$user = $result->fetch_assoc();
echo json_encode(["success" => true, "data" => $user]);
```

A felhasználói bejelentkezés a rendszerben a megadott e-mail és jelszó alapján történik. Ha a felhasználó adatai helyesek, akkor a rendszer beállítja a session-t a felhasználó azonosítójával, és sikeres válasz érkezik a frontend számára. Ha a megadott adatok nem egyeznek az adatbázisban tároltakkal, akkor egy hibajelzést küld vissza a rendszer:

```
<?php
$query = "SELECT id FROM user WHERE email = ? AND password = ?";
$stmt = $dbconn->prepare($query);
```

```

$stmt->bind_param("ss", $email, $hashedPassword);
$stmt->execute();
$result = $stmt->get_result();
if ($result->num_rows > 0) {
    $_SESSION['user_id'] = $result->fetch_assoc()['id'];
    echo json_encode(["success" => true]);
} else {
    echo json_encode(["success" => false, "message" => "Hibás e-mail vagy jelszó."]);
}

```

A kijelentkezés folyamata egyszerűen a session törlésével történik. Ezzel biztosítjuk, hogy a felhasználó ne férhessen hozzá többé a saját adataihoz anélkül, hogy újra bejelentkezne:

```

<?php
session_destroy();
echo json_encode(["success" => true]);

```

A felhasználók regisztrációja során a rendszer rögzíti a felhasználó nevét, e-mail címét és jelszavát. A jelszót bcrypt titkosítással tárolja az adatbázisban, biztosítva annak védelmét. Miután a felhasználói adatokat elmentette az adatbázisba, a rendszer új fiókot hoz létre, amelyet a felhasználó használhat a továbbiakban. A következő kódrészlet végzi el a felhasználó regisztrációját:

```

<?php
$query = "INSERT INTO user (name, email, password) VALUES (?, ?, ?)";
$stmt = $dbconn->prepare($query);
$stmt->bind_param("sss", $name, $email, $hashedPassword);
$stmt->execute();

```

A felhasználó címadatainak frissítése akkor szükséges, ha a felhasználó módosítani szeretné az otthoni címét. A cím módosítása a felhasználó session-ból származó azonosító alapján történik, és az új adatokat elmentjük az adatbázisba. Az alábbi kódrészlet végzi el a címadatok frissítését:


```
<?php
```

```
$query = "UPDATE user SET address = ? WHERE id = ?";
```

```
$stmt = $dbconn->prepare($query);
```

```
$stmt->bind_param("si", $address, $_SESSION['user_id']);
```

```
$stmt->execute();
```

Végül, ha a felhasználó személyes adatainak módosítására van szükség, például a neve vagy e-mail címe frissítésére, akkor a rendszer az új adatokat elmenti az adatbázisba a felhasználó session-jából származó azonosító alapján. Ezzel biztosítjuk, hogy a felhasználó naprakész adatokat tartson fenn a rendszerben, és könnyedén módosíthassa személyes adatait. Az alábbi kódrészlet frissíti a felhasználó nevét és e-mail címét:

```
<?php
```

```
$query = "UPDATE user SET name = ?, email = ? WHERE id = ?";
```

```
$stmt = $dbconn->prepare($query);
```

```
$stmt->bind_param("ssi", $name, $email, $_SESSION['user_id']);
```

```
$stmt->execute();
```

2.7.2 Termékek kezelése

A termékek kezeléséhez többféle funkció is tartozik a rendszerben. Ezek mind azt a célt szolgálják, hogy a felhasználók saját maguk tudják kezelni a feltöltött termékeiket, például új terméket hozzáadni, meglévőt módosítani vagy törölni.

Amikor egy felhasználó törölni akar egy terméket, először azt nézzük meg, hogy be van-e jelentkezve. Ha nincs, akkor nem engedjük meg neki a törlést. Ha igen, akkor a kérésből kiolvassuk, hogy melyik terméket szeretné törölni. A törlés úgy történik, hogy az adatbázisból kitöröljük azt a rekordot, amihez az adott felhasználó azonosítója is hozzá van rendelve. Így nem lehet más termékeit véletlenül vagy szándékosan törölni.

```
<?php
```

```
require_once __DIR__ . '/../connect.php';
```

```
session_start();
```

```
if (!isset($_SESSION['user_id'])) {
```

```

    echo json_encode(["success" => false, "message" => "Nincs bejelentkezve."]);
    exit;
}
$data = json_decode(file_get_contents("php://input"), true);
$product_id = $data['id'] ?? 0;
$query = "DELETE FROM products WHERE id = ? AND user_id = ?";
$stmt = $dbconn->prepare($query);
$stmt->bind_param("ii", $product_id, $_SESSION['user_id']);
$success = $stmt->execute();
echo json_encode(["success" => $success]);
?>

```

Ha a felhasználó meg akarja nézni, milyen termékeket töltött fel korábban, erre is van lehetőség. Ehhez szintén ellenőrizzük, hogy be van-e jelentkezve, majd az adatbázisból csak azokat a termékeket kérjük le, amelyeket ő töltött fel. Az adatokat egy tömbbe gyűjtjük össze, majd JSON formátumban küldjük vissza, hogy a frontend szépen ki tudja írni.

```

<?php
require_once __DIR__ . '/../connect.php';
session_start();
if (!isset($_SESSION['user_id'])) {
    echo json_encode(["success" => false, "message" => "Nincs bejelentkezve."]);
    exit;
}
$query = "SELECT id, name, price, description, quantity, size, `condition` FROM products
WHERE user_id = ?";
$stmt = $dbconn->prepare($query);
$stmt->bind_param("i", $_SESSION['user_id']);
$stmt->execute();
$result = $stmt->get_result();
$products = [];
while ($row = $result->fetch_assoc()) {
    $products[] = $row;
}

```

```
echo json_encode(["success" => true, "products" => $products]);  
?>
```

A webshop főoldalára azok a termékek kerülnek ki, amik még nem keltek el. Ezeket az adatokat is egy lekérdezéssel szedjük össze az adatbázisból. Közben hozzátesszük a termékekhez a hozzájuk tartozó képet, a márkát és a kategóriát is, hogy minden fontos adat megjelenjen. A találatokat úgy rendezzük, hogy a legfrissebbek legyenek elől.

```
<?php  
require_once __DIR__ . '/../connect.php';  
$query = "  
    SELECT  
        p.id, p.name, p.description, p.price, p.quantity, p.size, p.condition,  
        COALESCE(i.img_url, 'no-image.jpg') AS img_url,  
        b.brand_name, c.category_name  
    FROM products p  
    LEFT JOIN image i ON p.image_id = i.id  
    LEFT JOIN brand b ON p.brand_id = b.id  
    LEFT JOIN category c ON p.category_id = c.id  
    WHERE p.isSold = 0  
    ORDER BY p.uploaded_at DESC  
";  
$result = $dbconn->query($query);  
$products = $result->fetch_all(MYSQLI_ASSOC);  
echo json_encode(["success" => true, "products" => $products]);  
?>
```

Új termék feltöltésekor a felhasználó kitölti az adatokat a frontend oldalon, amit aztán a rendszer **POST** metódussal elküld a backendnek. A szerver először megint megnézi, hogy a felhasználó be van-e jelentkezve. Ha igen, akkor a kapott adatokat elmentjük az adatbázisba. Itt több mezőt is kezelünk, például a nevet, az árat, a leírást, a mennyiséget, a méretet, az állapotot, valamint a termék kategóriáját és márkáját is.

```

<?php
require_once __DIR__ . '/../connect.php';
session_start();
if (!isset($_SESSION['user_id'])) {
    echo json_encode(["success" => false, "message" => "Nincs bejelentkezve."]);
    exit;
}

$name = $_POST['name'];
$description = $_POST['description'];
$price = floatval($_POST['price']);
$quantity = intval($_POST['quantity']);
$size = $_POST['size'];
$condition = $_POST['condition'];
$category_id = intval($_POST['category_id']);
$brand_id = intval($_POST['brand_id']);

$query = "INSERT INTO products (user_id, name, description, price, quantity, size,
`condition`, category_id, brand_id) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";
$stmt = $dbconn->prepare($query);
$stmt->bind_param("issdissii", $_SESSION['user_id'], $name, $description, $price, $quantity,
$size, $condition, $category_id, $brand_id);
$success = $stmt->execute();
echo json_encode(["success" => $success]);
?>

```

Arra is van lehetőség, hogy egy már feltöltött termék adatait később módosítsuk. A változtatásokat JSON-ban küldi el a frontend, és a szerver az alapján frissíti az adatbázisban a megfelelő sort. Itt is meg van kötve, hogy csak az a felhasználó módosíthat egy terméket, aki eredetileg feltöltötte azt.

```

<?php
require_once __DIR__ . '/../connect.php';
session_start();
if (!isset($_SESSION['user_id'])) {
    echo json_encode(["success" => false, "message" => "Nincs bejelentkezve."]);

```

```

    exit;
}
$data = json_decode(file_get_contents("php://input"), true);
$product_id = $data['id'] ?? 0;
$name = $data['name'] ?? "";
$price = $data['price'] ?? 0;
$description = $data['description'] ?? "";
$quantity = $data['quantity'] ?? 0;
$size = $data['size'] ?? "";
$condition = $data['condition'] ?? "";
$query = "UPDATE products SET name = ?, price = ?, description = ?, quantity = ?, size = ?,
`condition` = ? WHERE id = ? AND user_id = ?";
$stmt = $dbconn->prepare($query);
$stmt->bind_param("sisissii", $name, $price, $description, $quantity, $size, $condition,
$product_id, $_SESSION['user_id']);
$success = $stmt->execute();
echo json_encode(["success" => $success]);
?>

```

Összességében tehát minden műveletnél figyelünk arra, hogy csak a bejelentkezett felhasználók tudjanak saját termékeket kezelni. Így biztonságos és egyszerű a rendszer használata, és mindenki csak a saját adataival dolgozhat.

2.7.3 Licitek kezelése

Az aukciók kezeléséhez több funkciót is kialakítottunk a rendszerben. Ezek mind azt a célt szolgálják, hogy az aukciók folyamata automatikusan, biztonságosan és gördülékenyen történjen, a felhasználók pedig könnyen tudjanak licitálni, aukciót létrehozni vagy az árakat frissíteni.

Amikor egy aukció lejár, a rendszer automatikusan ellenőrzi, hogy volt-e érvényes nyertes licitáló. Ha igen, az aukció adatai bekerülnek a `sales` táblába, rögzítve az eladó és a vevő

azonosítóját, valamint az aukció azonosítóját és a mennyiséget. A műveletet az alábbi kódrészlet végzi:

```
if ($buyer_id > 0) {  
    $stmtInsert = $dbconn->prepare("INSERT INTO sales (saler_id, buyer_id, product_id,  
    auction_id, quantity) VALUES (?, ?, NULL, ?, ?)");  
    $stmtInsert->bind_param("iiii", $saler_id, $buyer_id, $auction_id, $quantity);  
    if ($stmtInsert->execute()) {  
        echo json_encode(["success" => true, "message" => "A termék sikeresen hozzáadva a  
sales táblához."]);  
    } else {  
        echo json_encode(["success" => false, "error" => "Adatbázis hiba a sales táblába helyezés  
során."]);  
    }  
    $stmtInsert->close();  
}
```

Ezzel biztosítjuk, hogy minden sikeres aukció végén az eladás adatai megfelelően rögzítésre kerüljenek az adatbázisban.

Az aktuálisan futó aukciókat egy külön lekérdezéssel gyűjtjük össze. Csak azokat az aukciókat kérjük le, amelyeknek az aukció vége dátuma még nem járt le, így a felhasználók kizárólag érvényes, aktív liciteket látnak. A lekérdezés során nemcsak az aukció alapadatait töltjük be, hanem a hozzájuk tartozó képet, márkanévet és kategóriát is, hogy a megjelenítés teljes körű legyen:

```
$sql = "  
SELECT  
    a.id AS auction_id,  
    a.user_id AS owner_id,  
    a.name,  
    a.price AS original_price,  
    a.ho AS price,  
    a.ho_id,  
    a.stair,
```

```

a.auction_end,
a.size,
a.condition,
a.description,
i.img_url,
b.brand_name,
c.category_name
FROM auction a
LEFT JOIN image i ON a.image_id = i.id
LEFT JOIN brand b ON a.brand_id = b.id
LEFT JOIN category c ON a.category_id = c.id
WHERE a.auction_end > NOW()
ORDER BY a.uploaded_at DESC
";

```

Így az aukciók listázásakor minden fontos információ egy lekérdezésben elérhető, optimalizálva az adatforgalmat és a teljesítményt.

A rendszer arra is lehetőséget biztosít, hogy a licitálás során az aukció aktuális árát frissítsük. Amikor egy felhasználó magasabb árat ajánl, a backend módosítja az aukció rekordját, frissítve az aktuális licitösszeget (**ho**) és a legmagasabb licitáló azonosítóját (**ho_id**). Ez a frissítés az alábbi módon történik:

```

$sql = "UPDATE auction SET ho = ?, ho_id = ? WHERE id = ?";
$stmt = $dbconn->prepare($sql);
$stmt->bind_param("dii", $new_price, $user_id, $auction_id);
if ($stmt->execute()) {
    echo json_encode(["success" => true, "message" => "Az ár sikeresen frissítve."]);
} else {
    echo json_encode(["success" => false, "error" => "Nem sikerült frissíteni az árat."]);
}

```

Ezzel a megoldással mindig naprakészen követhető, hogy ki áll éppen az aukció élén, és mekkora összeggel.

Új aukció létrehozására is biztosítottunk lehetőséget a felhasználók számára. A termék részleteinek megadása után a frontend elküldi az adatokat a szervernek, amely ezeket egy új rekordként rögzíti az `auction` táblában. A feltöltéskor az induló ár, a lépcsőköz, az aukció vége dátuma, a méret, az állapot, a leírás, valamint az opcionális kép, márka és kategória adatok is bekerülnek az adatbázisba:

```
$sql = "INSERT INTO auction (user_id, name, price, ho, stair, auction_end, size, condition,
description, image_id, brand_id, category_id)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
$stmt = $dbconn->prepare($sql);
$stmt->bind_param("isddssssiiii", $user_id, $name, $price, $price, $stair, $auction_end, $size,
$condition, $description, $image_id, $brand_id, $category_id);
if ($stmt->execute()) {
    echo json_encode(["success" => true, "message" => "Az aukció sikeresen feltöltve."]);
} else {
    echo json_encode(["success" => false, "error" => "Nem sikerült feltölteni az aukciót."]);
}
```

Így minden új aukció megfelelő adatokkal indul, biztosítva a részletes termékbemutatót és a korrekt licitálási környezetet.

Összességében a rendszer minden pontján odafigyeltünk arra, hogy az aukciók kezelése átlátható, gyors és biztonságos legyen, és a felhasználók mindig naprakész adatok alapján tudjanak licitálni vagy saját aukciókat indítani.

2.7.4 Fizetés és kosár kezelése

A kosár és a fizetési rendszer működését csapatként több PHP-alapú háttérfolyamattal alakítottuk ki, amelyek biztosítják, hogy a felhasználók gördülékenyen kezelhessék vásárlásaikat. A kosár tartalmának megjelenítéséhez a vásárló azonosítóját használva kérdezzük le az adatbázist, a `sales` és `products` táblák összekapcsolásával. Így minden kosárban lévő termék nevét, árát, mennyiségét és képét is ki tudjuk olvasni:

```
$sql = "SELECT s.id AS sale_id, p.name AS product_name, s.quantity, p.price, p.image_url
```



```

FROM sales s
JOIN products p ON s.product_id = p.id
WHERE s.buyer_id = ?";
$stmt = $dbconn->prepare($sql);
$stmt->bind_param("i", $buyer_id);
$stmt->execute();
$result = $stmt->get_result();
$cartItems = [];
while ($row = $result->fetch_assoc()) {
    $cartItems[] = $row;
}
echo json_encode(["success" => true, "cart" => $cartItems]);

```

Ezzel a megoldással mindig naprakész adatokat tudunk biztosítani a vásárlóknak a kosaruk állapotáról.

Amikor valaki terméket próbál a kosárhoz adni, első lépésként ellenőrizzük, hogy nem a saját árúját szeretné megvásárolni. Ha az eladó és a vásárló nem azonos, a tételt egy új bejegyzéssel rögzítjük a `sales` táblában:

```

if ($saler_id === $buyer_id) {
    echo json_encode(["success" => false, "error" => "Nem adhatod hozzá a saját termékedet a kosárhoz."]);
    exit;
}
$stmt = $dbconn->prepare("INSERT INTO sales (saler_id, buyer_id, product_id, quantity)
VALUES (?, ?, ?, ?)");
$stmt->bind_param("iiii", $saler_id, $buyer_id, $product_id, $quantity);
if ($stmt->execute()) {
    echo json_encode(["success" => true]);
} else {
    echo json_encode(["success" => false, "error" => "Adatbázis hiba"]);
}

```

Ezzel biztosítottuk, hogy a rendszer csak valódi vásárlásokat engedélyezzen.

A kosárból történő eltávolítást is lehetővé tettük. Ha a felhasználó törölni szeretne egy elemet, a megadott terméket a `sales` táblából az adott vásárlóhoz kötve távolítjuk el:

```
$sql = "DELETE FROM sales WHERE id = ? AND buyer_id = ?";  
$stmt = $dbconn->prepare($sql);  
$stmt->bind_param("ii", $sale_id, $buyer_id);  
if ($stmt->execute()) {  
    echo json_encode(["success" => true, "message" => "A termék sikeresen törölve a  
kosárból."]);  
} else {  
    echo json_encode(["success" => false, "error" => "Nem sikerült törölni a terméket."]);  
}
```

Ezzel a lépéssel rugalmasan kezelhetővé tettük a kosár tartalmát, a felhasználók igényeihez igazodva.

Annak érdekében, hogy új termék hozzáadásakor ellenőrizni tudjuk az eladó személyét, létrehoztunk egy lekérdezést, amely visszaadja az adott termékhez tartozó eladó azonosítóját:

```
$sql = "SELECT user_id AS saler_id FROM products WHERE id = ?";  
$stmt = $dbconn->prepare($sql);  
$stmt->bind_param("i", $product_id);  
$stmt->execute();  
$result = $stmt->get_result();  
if ($row = $result->fetch_assoc()) {  
    echo json_encode(["success" => true, "saler_id" => $row['saler_id']]);  
} else {  
    echo json_encode(["success" => false, "error" => "Nem található eladó ehhez a  
termékhez."]);  
}
```

Ezzel a plusz ellenőrzéssel tovább növeltük a rendszer biztonságát.

A fizetési folyamat során a kosárban lévő tételeket áthelyezzük az `orders` táblába, véglegesítve ezzel a vásárlást. Minden vásárolt termék esetében rögzítjük a mennyiséget, az árakat, illetve a rendelés időpontját:

```
$sql = "INSERT INTO orders (buyer_id, product_id, quantity, total_price, order_date)
      SELECT buyer_id, product_id, quantity, (quantity * price), NOW()
      FROM sales
      WHERE buyer_id = ?";
$stmt = $dbconn->prepare($sql);
$stmt->bind_param("i", $buyer_id);
if ($stmt->execute()) {
    echo json_encode(["success" => true, "message" => "A rendelés sikeresen véglegesítve."]);
} else {
    echo json_encode(["success" => false, "error" => "Nem sikerült véglegesíteni a rendelést."]);
}
```

Így a kosár tartalma átlátható módon, egy lépésben kerül rögzítésre a rendelések közé.

A checkout előtt lehetőséget adtunk a felhasználói adatok frissítésére is. Ha valaki módosította a címét vagy az elérhetőségeit, ezeket az adatokat még a vásárlás véglegesítése előtt elmentjük:

```
$updateUser = "
      UPDATE user
      SET
        city_id = ?,
        street = ?,
        address = ?,
        firstname = ?,
        lastname = ?,
        email = ?,
        phone_number = ?
      WHERE id = ?
";
$stmt = mysqli_prepare($dbconn, $updateUser);
```

```
mysqli_stmt_bind_param($stmt, "isssssi", $cityId, $street, $houseNumber, $firstname,
$lastname, $email, $phone, $userId);
$success = mysqli_stmt_execute($stmt);
```

Ezzel a megközelítéssel biztosítottuk, hogy a vásárlás minden szempontból pontos és naprakész adatokkal történjen.

Összességében egy könnyen használható, megbízható kosárkezelő és fizetési rendszert hoztunk létre, amely folyamatosan figyeli az adatok helyességét, támogatja a vásárlói műveleteket, és biztosítja a tranzakciók biztonságát.

2.8. Frontend felépítése (HTML, CSS, JavaScript)

2.8.1. HTML oldalak szerkezete

A felhasználói felület HTML oldalai egyszerű, jól strukturált fájlokból állnak, melyek könnyen karbantarthatók és áttekinthetők. Minden oldal közös szerkezetre épül: a fejléc (header), a navigációs rész (nav), a fő tartalomterület (main) és a lábléc (footer) mindegyik oldalon megtalálható.

Az oldalak között megkülönböztetjük a publikus és a védett oldalakat. A publikus oldalak közé tartozik például a kezdőlap, a regisztráció és a bejelentkezés. A védett oldalak, mint a profil szerkesztése, a kosár és a fizetés, csak bejelentkezett felhasználók számára elérhetők.

A HTML oldalak felépítését minden fájlban egységesítettük, alapvázra építettük. A fejlécben megadjuk a karakterkódolást, a reszponzív megjelenítéshez szükséges meta beállításokat, és csatoljuk az oldalhoz tartozó specifikus CSS fájlt. A JavaScript fájlokat mindig az oldal alján csatoljuk, hogy a dokumentum teljes betöltődése után futtathassák a szkripteket. A fő tartalom a main elemben található, amely dinamikusan is módosítható.

2.8.2. Stíluslapok (CSS)

A weboldal megjelenését a minden oldalhoz tartozó egyedi stíluslapok szabályozzák. Minden HTML oldalhoz egy különálló CSS fájl tartozik, így minden oldal saját megjelenése külön-külön testreszabható. A stíluslapok biztosítják az oldal egységes vizuális megjelenését, alkalmazva az egységes színpalettát

és betűtípusokat. A reszponzív dizájn érdekében CSS media query-ket használunk, hogy az oldal különböző eszközökön is megfelelően jelenjen meg.

```
@media (max-width: 1200px) {
```

```
/*Monitorokhoz tartozó stílusok */
```

```
}
```

```
@media (max-width: 1024px) {
```

```
/*Tabletekhez tartozó stílusok */
```

```
}
```

```
@media (max-width: 768px) {
```

```
/*Mobiltelefonokhoz tartozó stílusok */
```

```
}
```

```
@media (max-width: 480px) {
```

```
/*Kis mobiltelefonokhoz tartozó stílusok */
```

```
}
```

A különböző komponensek, mint a gombok, űrlapok és kártyák külön osztályokkal vannak ellátva, így azok könnyen újra felhasználhatóak és módosíthatóak.

2.8.3. Dinamikus műveletek (JavaScript, Fetch API)

A weboldal dinamikus működését JavaScript biztosítja. Minden oldal saját, specifikus JavaScript fájlt tartalmaz, amely az adott oldal műveleteit kezeli. A JavaScript kódok feladata az oldalon történő események kezelése, mint az űrlapok beküldése, felhasználói műveletekre adott válaszok megjelenítése, valamint adatok betöltése és küldése a szerverről.

A fetch() metódus segítségével történik az adatküldés és -fogadás a háttérben a backend API-val, ami lehetővé teszi, hogy az oldal frissítés nélkül kommunikáljon a szerverrel.

Az adatokat JSON formátumban küldjük és fogadjuk a szerverről GET és POST metódusokkal. A válaszként kapott JSON adatokat feldolgozzuk és dinamikusan töltjük be a DOM-ba.

A hibakezelés minden fetch hívásnál fontos szerepet kap: minden esetleges hibát lekezelünk, és értesítjük a felhasználót, ha szükséges.

2.9. API kialakítása

A rendszerben egy RESTful API biztosítja a felhasználói adatok kezelését. Az API HTTP GET, POST, PUT és DELETE metódusokkal kommunikál, és JSON formátumban küldi és fogadja az adatokat. Az API végpontok különböző műveletek végrehajtására szolgálnak, mint például a felhasználók adatainak lekérdezése, hozzáadása, frissítése és törlése. A kommunikáció a megfelelő HTTP metódusokkal történik, biztosítva az adatok hatékony kezelését.

2.9.1. API végpontok rövid bemutatása

Az API különböző metódusokkal támogatja az adatkezelést. A GET kérés segítségével lekérhetjük a felhasználói adatokat. Ha megadunk egy **id** paramétert, akkor csak a konkrét felhasználó adatai érkeznek vissza, különben az összes felhasználó információi elérhetők. Az alábbi kódrészlet biztosítja a felhasználók adatainak lekérését:

```
if ($_SERVER["REQUEST_METHOD"] === "GET") {  
    if (isset($_GET['id'])) {  
        getUser($_GET['id']);  
    } else {  
        getUsers();  
    }  
}
```

A POST kérés új felhasználó hozzáadására szolgál. Ilyenkor a felhasználó adatokat JSON formátumban kell elküldeni, hogy a rendszer azokat feldolgozza és hozzáadja az adatbázishoz. Az alábbi kódrészlet mutatja a felhasználó hozzáadását:

```

if ($_SERVER["REQUEST_METHOD"] === "POST") {
    addUser();
}

function addUser() {
    global $dbconn;
    $data = json_decode(file_get_contents("php://input"), true);
    $sql = "INSERT INTO user (firstname, lastname, email, password) VALUES (?, ?, ?, ?)";
    $stmt = $dbconn->prepare($sql);
    $stmt->bind_param("ssss", $data['firstname'], $data['lastname'], $data['email'],
    $data['password']);
    $stmt->execute();
    echo json_encode(["message" => "User added successfully", "id" => $dbconn->insert_id]);
}

```

A PUT metódus a meglévő felhasználói adatok frissítésére szolgál. A kérést követően a felhasználó adatainak módosítása történik, ha az id és az új adatok meg vannak adva. Az alábbi kódrészlet bemutatja a frissítés folyamatát:

```

if ($_SERVER["REQUEST_METHOD"] === "PUT") {
    $data = json_decode(file_get_contents("php://input"), true);
    if (isset($data['id'])) {
        updateUser($data['id'], $data);
    }
}

function updateUser($id, $data) {
    global $dbconn;
    $sql = "UPDATE user SET firstname = ?, lastname = ?, email = ?, password = ? WHERE id
    = ?";
    $stmt = $dbconn->prepare($sql);
    $stmt->bind_param("ssssi", $data['firstname'], $data['lastname'], $data['email'],
    $data['password'], $id);
    $stmt->execute();
}

```

```
echo json_encode(["message" => "User updated successfully"]);  
}
```

A DELETE kérés lehetővé teszi a felhasználó törlését az adatbázisból. A kérés tartalmazza a törlendő felhasználó `id`-jét, és ha az helyesen van megadva, akkor a rendszer eltávolítja a felhasználót. A következő kódrészlet biztosítja a felhasználó törlését:

```
if ($_SERVER["REQUEST_METHOD"] === "DELETE") {  
    $data = json_decode(file_get_contents("php://input"), true);  
    if (isset($data['id'])) {  
        deleteUser($data['id']);  
    }  
}
```

```
function deleteUser($id) {  
    global $dbconn;  
    $sql = "DELETE FROM user WHERE id = ?";  
    $stmt = $dbconn->prepare($sql);  
    $stmt->bind_param("i", $id);  
    $stmt->execute();  
    echo json_encode(["message" => "User deleted successfully"]);  
}
```

2.9.2. JSON formátumú kommunikáció

Minden API válasz JSON formátumban érkezik, így biztosítva a megfelelő adatcserét a kliens és a szerver között. A válaszok mindig tartalmazzák a művelet eredményét, és a hibás kérések esetén hibaüzenetet adnak vissza. Az API a válaszokat HTTP státuszkódokkal kíséri, így az alkalmazás könnyen felismerheti a sikeres vagy hibás műveleteket.

A válaszok tartalmazzák a művelet eredményét, például sikeres adatbevitel esetén a következő JSON válasz érkezik:

```
{
```



```
"message": "User added successfully",  
"id": 1  
}
```

Hibás kérés esetén a válasz egy hibaüzenetet tartalmaz:

```
{  
  "error": "Failed to add user"  
}
```

A JSON válaszok egyszerűen feldolgozhatók különböző frontend alkalmazásokban. Az API válaszai mellett a fejléc beállítások is biztosítják, hogy a kommunikáció megfelelően történjen:

```
header("Content-Type: application/json");
```

Ez a beállítás garantálja, hogy minden válasz helyes formátumban érkezzen, és a válaszok könnyen értelmezhetők legyenek. Az API által biztosított kommunikáció biztosítja az egyszerű adatkezelést és integrációt különböző alkalmazásokkal.

2.10. Biztonsági megfontolások

A projekt során több fontos biztonsági intézkedést alkalmaztunk, hogy megvédjük az adatokat és a felhasználói információkat a különböző támadásokkal szemben.

2.10.1. SQL Injection elleni védelem

A kódunkban fontos szerepe van az SQL injection elleni védelemnek. Ehhez a `mysqli_prepare` és `mysqli_stmt_bind_param` függvényeket használjuk, amelyek segítségével biztonságosan dolgozunk az adatbázissal. A bemeneti adatokat paraméterként kezeljük, így elkerüljük, hogy a felhasználó rosszindulatú SQL kódot illeszthessen be a lekérdezésekbe. Az alábbi példában látható, hogyan biztosítjuk, hogy a felhasználói email cím ne okozhasson problémát az adatbázisban végrehajtott keresések során:

```
$query = "SELECT COUNT(*) FROM user WHERE email = ?";  
$stmt = mysqli_prepare($dbconn, $query);  
mysqli_stmt_bind_param($stmt, 's', $email);
```

2.10.2. Email validáció

A felhasználói adatokat mindig érvényesítenünk kell, különösen az email címeket. Az `filter_var` függvény használatával biztosítjuk, hogy csak helyes formátumú email címek kerülhessenek az adatbázisba. Ez megakadályozza, hogy a rendszer hibás adatokat fogadjon el, és ezzel csökkenti a potenciális problémák kockázatát.

```
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {  
    echo json_encode(['message' => 'Érvénytelen e-mail cím']);  
    exit;  
}
```

2.10.3. Jelszó erősség ellenőrzése

A gyenge jelszavak biztonsági kockázatot jelentenek, ezért a felhasználók által választott jelszavakat erősségük alapján ellenőrizzük. A jelszó minimum hosszát, a kisbetűk, nagybetűk és számok meglétét is megvizsgáljuk, hogy biztosítsuk, hogy minden jelszó megfelelő biztonsággal rendelkezzen.

```
if (strlen($password) < 8 || !preg_match('/[A-Z]/', $password) || !preg_match('/[a-z]/',  
$password) || !preg_match('/[0-9]/', $password)) {  
    echo json_encode(['message' => 'A jelszónak legalább 8 karakter hosszúnak kell lennie, és  
tartalmaznia kell kisbetűket, nagybetűket és számokat']);  
    exit;  
}
```

2.10.4. Jelszó hash-elése

A jelszavak biztonságos tárolása érdekében minden felhasználói jelszót egy erős hash algoritmus segítségével titkosítunk, így még ha valaki hozzáfér is az adatbázishoz, nem férhet hozzá a felhasználói jelszavakhoz. Ehhez a `password_hash` függvényt alkalmazzuk, amely a Bcrypt algoritmus segítségével titkosítja a jelszavakat.

```
$hashedPassword = password_hash($password, PASSWORD_BCRYPT);
```

2.10.5. Üres mezők ellenőrzése

A regisztrációs űrlapon minden kötelező mezőt érvényesítenünk kell, hogy a felhasználó ne hagyhassa üresen a fontos adatokat. Ez segít abban, hogy a rendszer minden szükséges információval rendelkezzen, mielőtt a felhasználó adatainak tárolására kerülne sor.

```
if (empty($firstname) || empty($lastname) || empty($email) || empty($password)) {  
    echo json_encode(['message' => 'Kérlek töltsd ki az összes mezőt!']);  
    exit;  
}
```

2.10.6. Email duplikáció ellenőrzése

Az adatbázisban tárolt email címeknek egyedinek kell lenniük. Ezért minden regisztráció előtt ellenőrizzük, hogy az adott email cím már szerepel-e a rendszerben, hogy elkerüljük a duplikált regisztrációkat. Ezzel megelőzhetjük, hogy a felhasználók többször regisztráljanak ugyanazzal az email címmel.

```
$query = "SELECT COUNT(*) FROM user WHERE email = ?";  
$stmt = mysqli_prepare($dbconn, $query);  
mysqli_stmt_bind_param($stmt, 's', $email);  
mysqli_stmt_execute($stmt);  
mysqli_stmt_bind_result($stmt, $emailExists);  
mysqli_stmt_fetch($stmt);
```

2.10.7. JSON bemenet ellenőrzése

A rendszerünk a felhasználói adatokat JSON formátumban várja, ezért minden bejövő adatot ellenőrzünk a `json_decode` segítségével. Ha a bemenet nem érvényes JSON formátumú, akkor hibaüzenetet küldünk vissza, és leállítjuk a folyamatot.

```
$data = json_decode(file_get_contents('php://input'), true);  
if (!$data) {  
    echo json_encode(['message' => 'Invalid JSON data']);  
    exit;  
}
```

2.10.8. Tartalom típusa

A válaszok formátumát a `header("Content-Type: application/json");` fejléc segítségével állítjuk be, hogy minden válasz JSON formátumban érkezzon a felhasználóhoz. Ez biztosítja, hogy az alkalmazás mindig egységes formátumban kommunikáljon, és megkönnyíti az adatfeldolgozást a kliensoldalon.

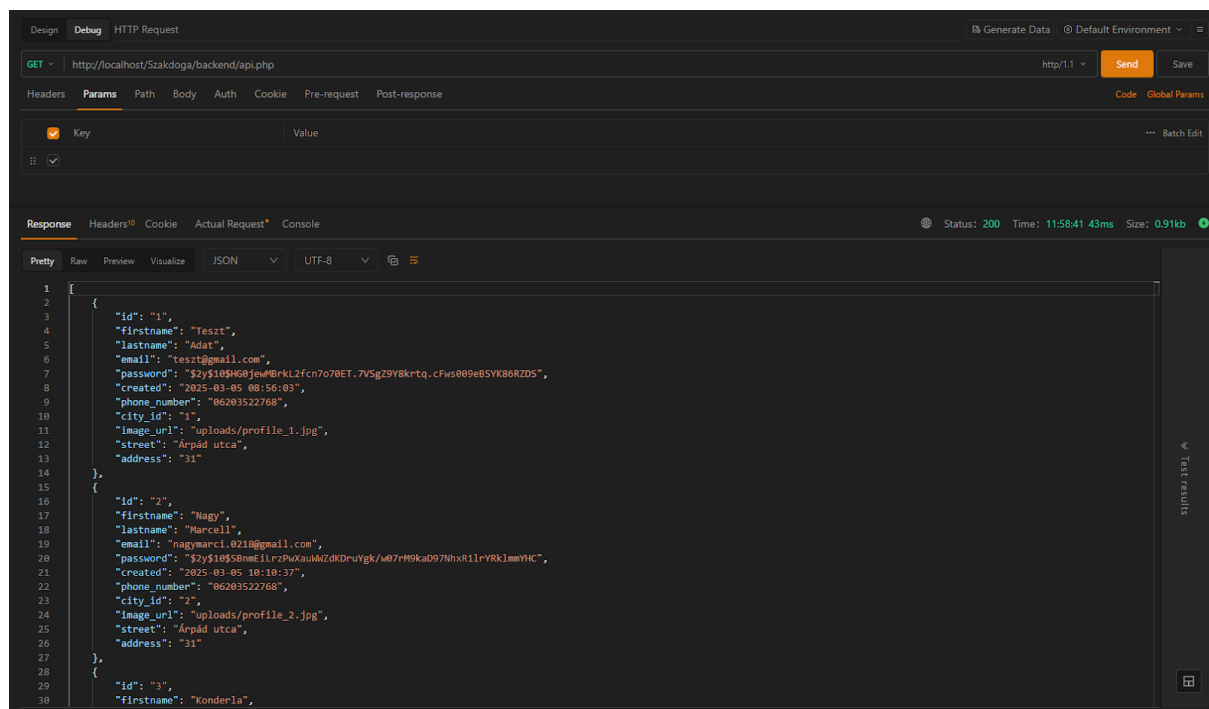
```
header("Content-Type: application/json");
```

2.11. Tesztelés

2.11.1 Get teszt

Ez a dokumentáció bemutatja a felhasználói adatok lekérésére szolgáló API végpontot, amelyet az echoApi segítségével tesztelünk. A kérés HTTP GET módszerrel történik a következő URL-re: `http://localhost/Szakdoga/backend/api.php`. Az API válasza JSON formátumban tartalmazza az adatbázisban tárolt felhasználói rekordokat.

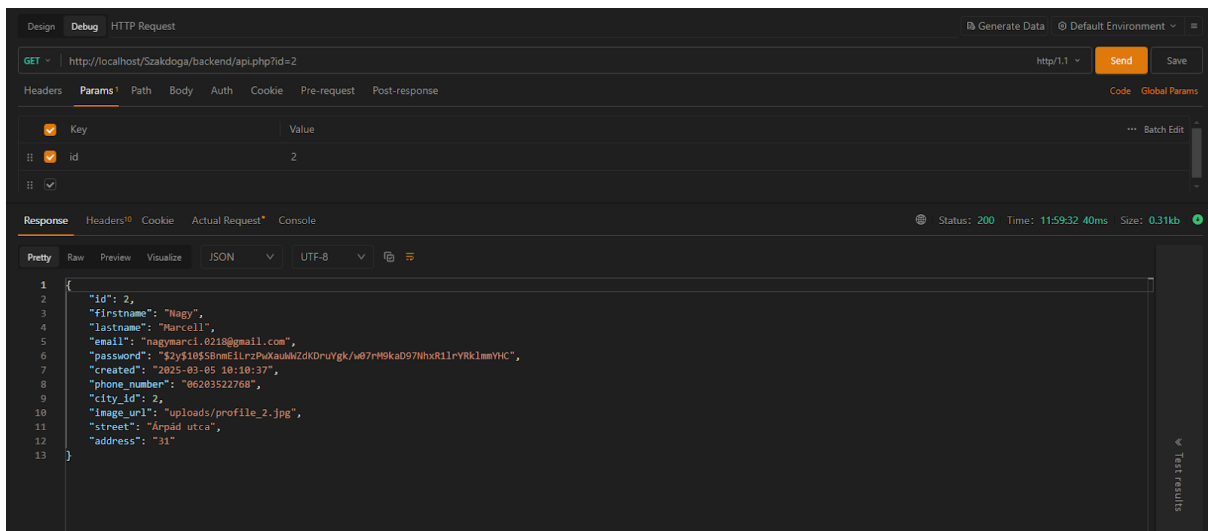
A kérés nem igényel URL paramétereket vagy törzstartalmat, és minden elérhető felhasználói adatot visszaad. A válasz egy JSON tömb, amely az alábbi mezőket tartalmazza: id, firstname, lastname, email, password, created, phone_number, city_id, image_url, street, address.



17. ábra Get teszt

2.11.1.1 Get teszt ID-re

Ebben a dokumentációban bemutatjuk a felhasználói adatok lekérésére kialakított API végpontot, amelyet EchoApi segítségével teszteltünk. A kérést HTTP GET módszerrel végeztük a következő URL-címre: <http://localhost/Szakdoga/backend/api.php>. A lekéréshez szükséges egyetlen URL-paraméter, az **id**, amely a kívánt felhasználó azonosítóját adja meg. A rendszer válaszként egy JSON formátumú objektumot küld vissza, amely tartalmazza a lekért felhasználó adatait. A kapott adatszerkezet a következő mezőkből áll: id, firstname, lastname, email, password, created, phone_number, city_id, image_url, street és address. A sikeres válasz 200-as HTTP státuszkóddal érkezik. A fejlesztés során közösen alakítottuk ki az adatstruktúrát és a válaszformátumot, hogy az könnyen feldolgozható és jól bővíthető legyen.



18. ábra Get teszt ID-re

2.11.2 Delete teszt

A felhasználói adat törlése HTTP DELETE módszerrel történik a következő URL-re: http://localhost/Szakdoga/backend/api.php?id={felhasználó_id}.

A szerver a törlendő felhasználó egyedi azonosítóját (id) várja URL paraméterként.

A sikeres törlés esetén a szerver az alábbi JSON választ küldi vissza:

```
{
  "message": "User deleted successfully"
}
```

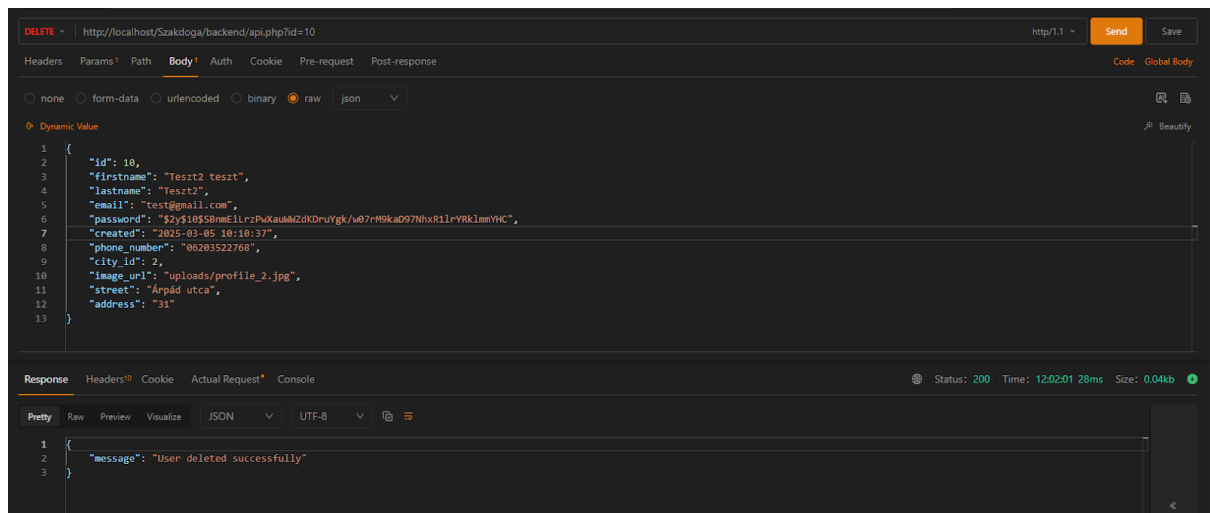
A szerver megfelelő válaszkódokat ad vissza:

200 OK – Sikeres törlés esetén.

400 Bad Request – Hiányzó vagy érvénytelen id esetén.

404 Not Found – Ha a megadott id nem található.

500 Internal Server Error – Szerverhiba esetén.



19. ábra Delete teszt

2.11.3 Post teszt

A felhasználói adatok hozzáadására szolgáló API-végpont a `http://localhost/Szakdoga/backend/api.php?id={felhasználó_id}` címen érhető el, ahol a HTTP metódus POST. Az új felhasználó adatait a kliens JSON formátumban küldi el a kérés törzsében. A beküldött adatok tartalmazzák a felhasználó egyedi azonosítóját, nevét, e-mail címét, jelszavát (amely bcrypt-tel van titkosítva), a fiók létrehozásának időpontját, telefonszámát, a város azonosítóját, profilképének URL-jét, az utcanevet és a házszámot.

Sikeres kérésre a szerver válaszként visszaad egy megerősítő üzenetet, valamint a létrehozott felhasználó adatbázisban rögzített azonosítóját:

```
{
```

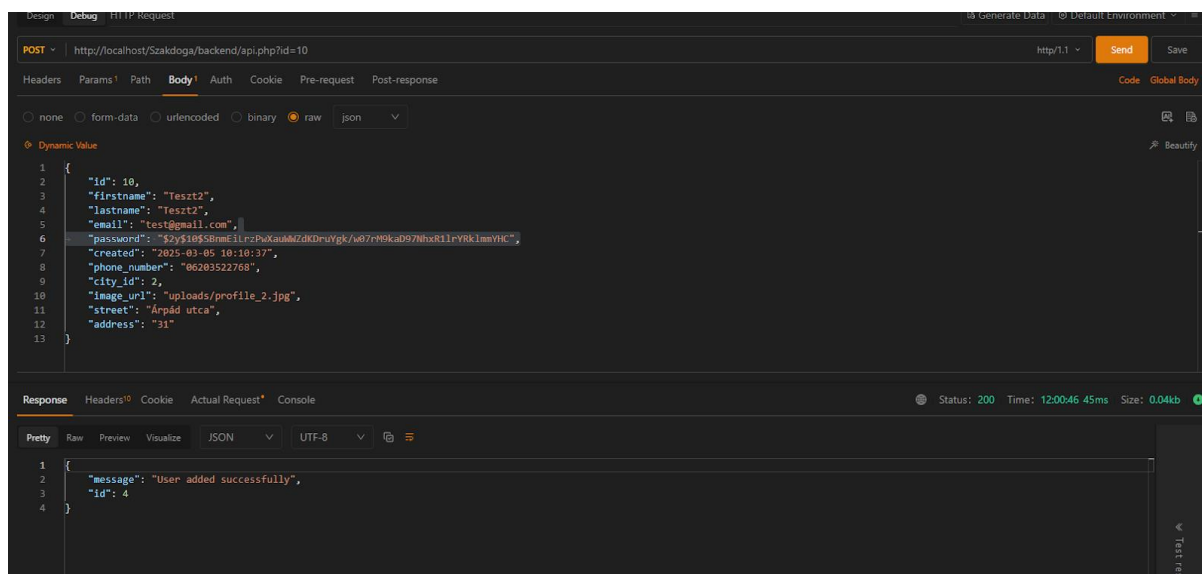
```
"message": "User added successfully",
```

```
"id": 4
```

}

HTTP válaszkódok:

- 200 OK – Sikeres hozzáadás.
- 400 Bad Request – Hibás vagy hiányos adat esetén.
- 500 Internal Server Error – Szerverhiba esetén.

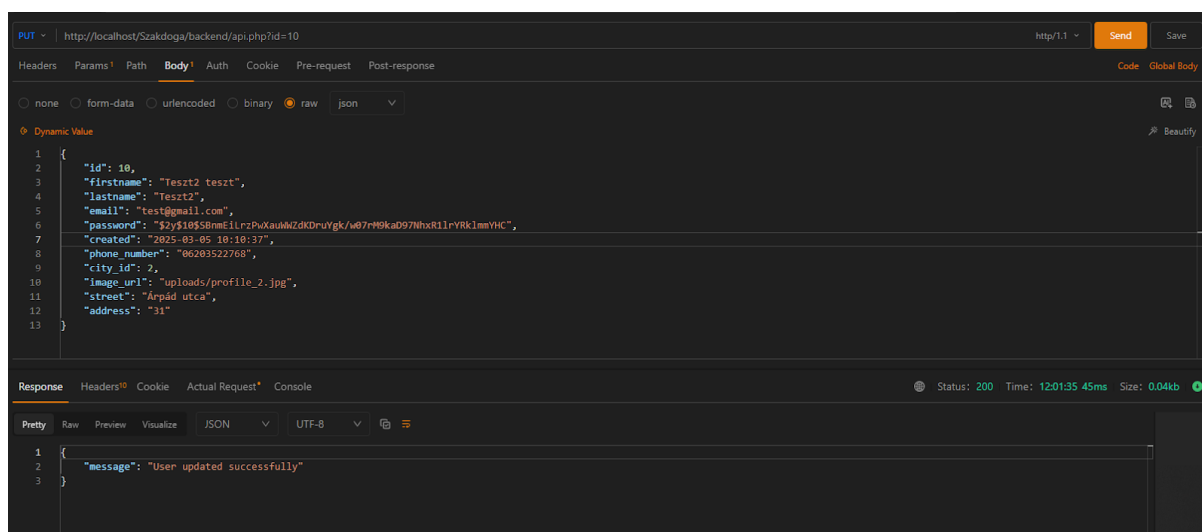


20. ábra Post teszt

2.11.4 Update

A meglévő felhasználói adatok frissítése HTTP PUT kéréssel történik. A kérést ugyanarra az URL-re kell küldeni, mint a hozzáadás vagy törlés esetén, az URL-ben az érintett felhasználó azonosítóját megadva. A kérés törzse tartalmazza a módosított adatokat. A szerver válasza megerősíti a sikeres frissítést.

Mindhárom művelet esetén a szerver visszajelzése tartalmaz egy rövid üzenetet, amely alapján eldönthető, hogy a művelet sikeresen végrehajtásra került-e. Az esetleges hibák esetén hibaüzenet és megfelelő HTTP státusz kód kerül visszaküldésre, például ha hiányos az adat vagy a megadott azonosító nem található.



21. ábra Update teszt

2.11.5. Frontend funkciók tesztelése

Az oldal frontend teszteléséhez a Google által fejlesztett Lighthouse böngészőbővítményt használtuk. Ez az eszköz lehetővé teszi az oldal teljesítményének, sebességének, helyességének és keresőoptimalizáltságának mérését. A bővítmény telepítése után megnyitottuk a tesztelni kívánt oldalt, amely esetünkben egy publikus poszt volt, hiszen itt várható a legtöbb felhasználói aktivitás. A Chrome böngésző beépített fejlesztői eszköztárában a Lighthouse fülre navigáltunk, majd az ott található „Analyze page load” gombra kattintva elindítottuk az oldal elemzését.

Az elemzés során az eszköz öt kategóriában értékelte az oldalt, mindegyik területen legfeljebb 100 pont szerezhető. A teljesítményre 97 pontot kaptunk, ami azt jelzi, hogy az oldal gyorsan tölt be. Ezen a téren korábban a Google API-ról töltöttük be a betűkészleteket, azonban a teszteredmények javítása érdekében áttértünk arra, hogy saját szerverről szolgáljuk ki őket. Ezáltal a betöltési idő jelentősen csökkent. A Lighthouse még javasolta a felhasználatlan JavaScript kódok csökkentését is, azonban erre jelenleg nincs lehetőségünk, mivel minden oldal ugyanazt a JavaScript fájlt használja.

Az akadálymentesség szempontjából 90 pontot értünk el. A jelentés szerint néhány gomb nem rendelkezik hozzáférhető névvel, illetve bizonyos linkek nem tartalmazznak megkülönböztethető szöveget. Ezek a problémák főként a képernyőolvasót használó

látogatókat érinthetik. Bár technikailag javítható lenne, design szempontból nem szeretnénk kiemelni ezeket az elemeket, ezért ebben a formában hagytuk őket.

A legjobb gyakorlatok kategóriában 100 pontot szereztünk, ami azt mutatja, hogy az oldal minden modern fejlesztési szabványnak megfelel. Ez többek között azt jelenti, hogy az oldal HTTPS-t használ, elkerüli az elavult API-kat, és a konzolban sem találhatóak hibák vagy figyelmeztetések.

A SEO, vagyis a keresőoptimalizálás területén 91 pontot kaptunk. A Lighthouse jelentése alapján az egyetlen hiányosság az volt, hogy a dokumentum nem tartalmaz meta leírást. Ez szándékos döntés volt a részünkről, mivel egyelőre nem célunk, hogy az oldal megjelenjen a keresőmotorok találati listáiban.

3. Az adatbázis célja

Az adatbázisunk a PremSelect nevű webshopunkhoz tartozó adatokat tárolja, mely webshop magánszemélyek közötti vásárlást és eladást teszi lehetővé, ebbe beleértve a felhasználók (vásárlók illetve eladók) adatait, a szállítási és felhasználói elérhetőséget, valamint az adott termékeket és azok tulajdonságait, amiket a felhasználók töltöttek fel a weboldalra.

3.1. Tervezés megkezdése

Az adatbázis tervezésénél figyelembe kellett vennünk a különböző kapcsolati struktúrákat, a normál formázást, valamint a hatékony lekérdezhetőséget. A kiindulási pontunk a webshopban megjelenő termékek és felhasználók voltak.

3.2. Tervezési lépések

Az adatbázis tervezésével sok nehézséggel találkoztunk, mivel ez volt az első komplikált adatbázisunk. Az órán használt <https://dbdiagram.io> felületén terveztük meg az adatbázist, mivel egy könnyen használható felülettel rendelkezik. A nehézségekkel a kapcsolatok pontos meghatározásakor találkoztunk.

3.3. Egyedtypusok/egyedek meghatározása

user: A felhasználók adatait tárolja.

settlement: A település adatait tárolja.

counties: A településhez (**settlement** tábla) tartozó megye adatait tárolja.

sales: Az eladások adatait tárolja.

products: Az adatbázisban szereplő (feltöltött) termékek adatait tárolja.

brand: A termékhez tartozó márkákat tárolja.

image: A termékhez és a felhasználóhoz tartozó képeket tárolja (az adott termékről)

category: A termékhez tartozó kategóriát tárolja

auction: Az adatbázisban jelenleg szereplő (feltöltött) aukciókat, liciteket tárolja.

3.4. Kapcsolatok meghatározása

user → **settlement:** A user tábla city_id mezője hivatkozik a settlement tábla id mezőjére. (1:N kapcsolat)

settlement → **counties:** A settlement tábla county_id mezője hivatkozik a counties tábla id mezőjére. (1:N kapcsolat)

sales → **user (buyer, seller):** A sales tábla saler_id és buyer_id mezői a user tábla id mezőjére hivatkoznak. (N:1 kapcsolat)

sales → **products:** A sales tábla product_id mezője a products tábla id mezőjére hivatkozik. (N:1 kapcsolat)

sales → **auction:** A sales tábla auction_id mezője az acution tábla id mezőjére hivatkozik. (N:1 kapcsolat)

products → **user:** A products tábla user_id mezője a user tábla id mezőjére hivatkozik. (1:N kapcsolat)

products → **category:** A products tábla category_id mezője a category tábla id mezőjére hivatkozik. (1:N kapcsolat)

products → **brand**: A products tábla brand_id mezője a brand tábla id mezőjére hivatkozik. (1:N kapcsolat)

products → **image**: A products tábla image_id mezője a image tábla id mezőjére hivatkozik. (1:N kapcsolat)

auction → **user**: Az auction tábla user_id mezője a user tábla id mezőjére hivatkozik. (1:N kapcsolat)

auction → **image**: Az auction tábla image_id mezője a image tábla id mezőjére hivatkozik. (1:N kapcsolat)

3.5. N:M kapcsolatok felbontása

- Egy termékhez több kép is tartozhat, és egy kép több termékhez is, ezért egy külön termék_image kapcsoló tábla készült.
- Az aukciókat több user is követheti, és egy user több aukciót is követhet, ezt egy auction_watchlist tábla kezeli.
- Egy felhasználó több aukción is licitálhat, és egy aukción több felhasználó is licitálhat, ezért létrehoztunk egy bids táblát.
- Egy felhasználó több terméket is elmenthet kedvencnek, és egy termék több felhasználónak is lehet a kedvence, ezért egy favorites tábla készült.

3.6. Táblák

3.6.1 user

mező neve	mező típusa	mező leírása	megjegyzés
id	bigint	user azonosítója (pk)	not null auto increment

firstname	varchar(100)	vezetéknév	not null
lastname	varchar(100)	keresztnev	not null
email	varchar(255)	a user email címe	not null
password	varchar(255)	a user jelszava	not null
created	timestamp	a fiók létrehozásának ideje	
phone_number	varchar(255)	a user telefonszáma	not null
city_id	int	a user irányítószáma	not null
image	varchar(255)	a user profilképe	
street	varchar	a user utca neve	
adress	varchar	a user házszáma	

3.6.2. settlement

mező neve	mező típusa	mező leírása	megjegyzés
id	int	a település azonosítója	not null
postcode	int	az irányítószám azonosítója(PK)	not null
name	varchar(255)	a user városának neve	not null
county_id	int	a megye azonosítója	not null

3.6.3. counties

mező neve	mező típusa	mező leírása	megjegyzés
id	int	a megye azonosítója(PK)	not null
name	varchar(255)	a megye neve	not null

3.6.4. sales

mező neve	mező típusa	mező leírása	megjegyzés
saler_id	bigint	az eladó azonosítója	not null
buyer_id	bigint	a vevő azonosítója	not null
product_id	bigint	a termék azonosítója	null
quantity	int	a vásárlásnak a db száma	not null
auction_id	bigint	a licit termék azonosítója	null
sold_at	datetime	a vásárlásnak a dátuma	not null

3.6.5. products

mező neve	mező típusa	mező leírása	megjegyzés
id	bigint	a feltöltött termék azonosítója	auto increment, not null
user_id	bigint	a terméket feltöltött felhasználó azonosítója	not null
name	varchar(255)	a feltöltött termék neve	not null

category_id	int	a termék kategóriájának az azonosítója	not null
brand_id	int	a termék gyártójának az azonosítója	not null
price	int	a termék ára	not null
description	varchar(255)	a termék leírása	not null
image_id	int	a termék képének az azonosítója	not null
uploaded_at	datetime	a termék feltöltésének ideje	not null
quantity	int	a terméknek a feltöltött darabszáma	not null
size	varchar(255)	a termék mérete	not null
condition	varchar(255)	a termék állapota	not null
isSold	tinyint(1)	logikai, 1 ha eladva, 0 ha nem	not null

3.6.6. image

mező neve	mező típusa	mező leírása	megjegyzés
id	int	a képnek az azonosítója	not null, auto increment
img_url	varchar(255)	a képnek az elérési útvonalát tárolja	not null

3.6.7. brand

mező neve	mező típusa	mező leírása	megjegyzés
id	int	a gyártónak az azonosítója	not null, auto increment
brand_name	varchar(255)	a gyártónak a neve	not null

3.6.8. category

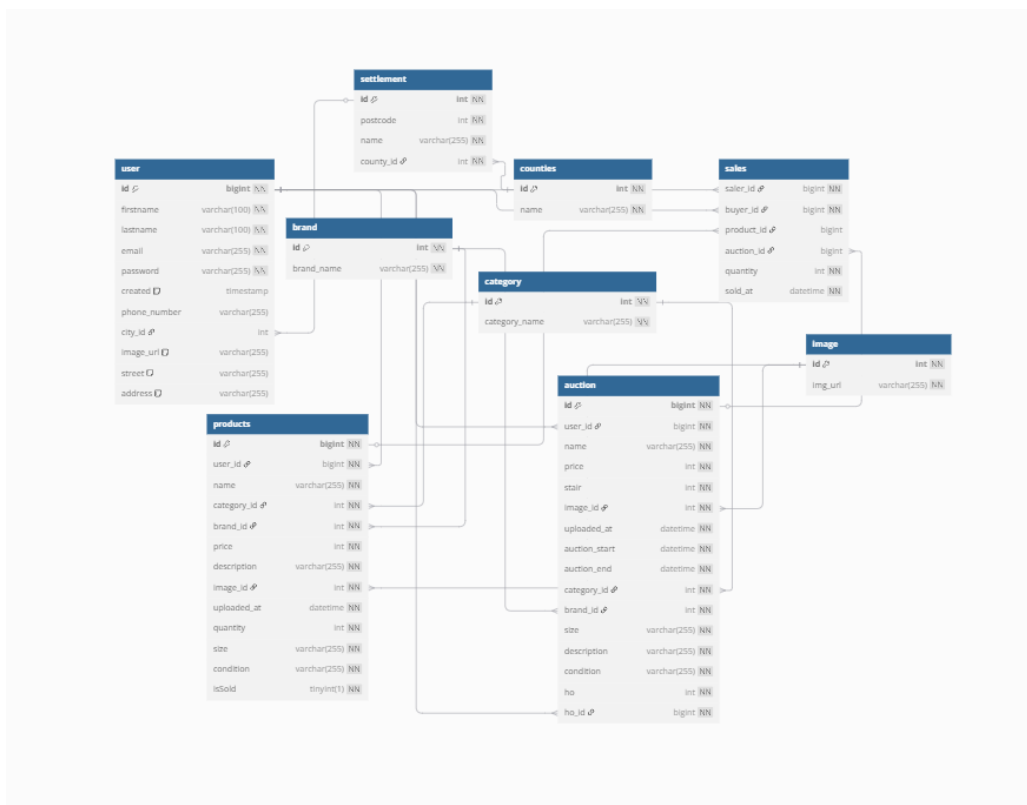
mező neve	mező típusa	mező leírása	megjegyzés
id	int	a kategóriának az azonosítója	not null, auto increment
category_name	varchar(255)	a kategória neve	not null

3.6.9. auction

mező neve	mező típusa	mező leírása	megjegyzés
id	int	a licit azonosítója(PK)	not null auto increment
user_id	bigint	a licitet feltöltött felhasználó azonosítója	not null
name	varchar(255)	a feltöltött licit neve	not null
price	int	a feltöltött licit ára	not null
stair	int	a licitlépcső ára	not null
image_id	int	a feltöltött kép azonosítója	not null
uploaded_at	datetime	a licit feltöltésének ideje	not null
auction_start	datetime	a licit kezdetének ideje	not null
auction_end	datetime	a licit végének ideje	not null

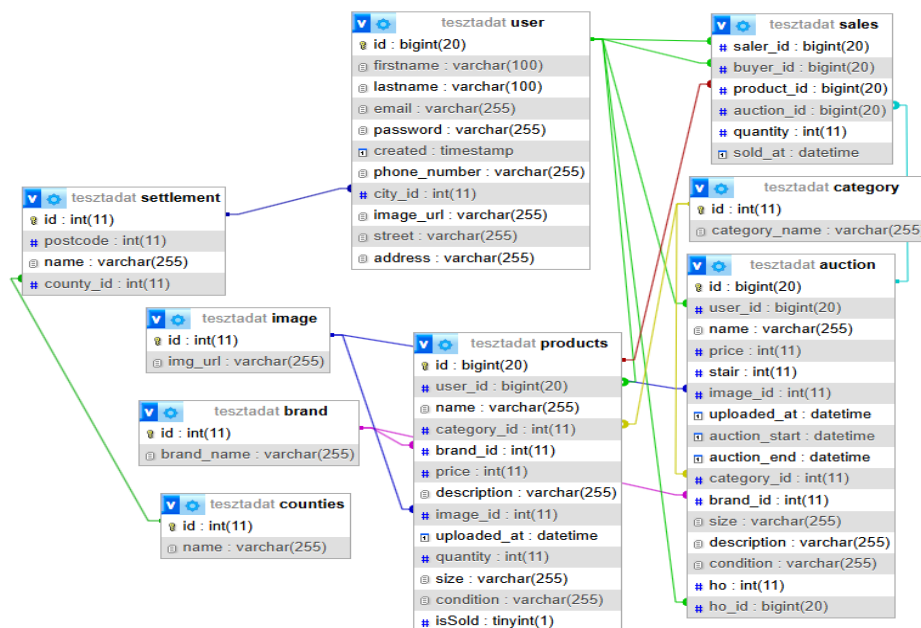
category_id	int	a termék kategóriájának az azonosítója	not null
brand_id	int	a termék gyártójának az azonosítója	not null
size	varchar(255)	a termék mérete	not null
description	varchar(255)	a termék leírása	not null
condition	varchar(255)	a termék állapota	not null
ho	int	a legmagasabb licit ára	not null
ho_id	bigint	a legmagasabb licitet adó felhasználó id-ja	not null

3.7. Adatbázis diagram dbdiagram felületen



22. ábra Az adatbázis DBDiagram felületen

3.8. Adatbázis szerkezete phpMyAdmin felületen



3.9. Adatbázis továbbfejlesztési lehetőségek

- Az adatbázis kiterjesztése nemzetközi szintre, mert a magyarországi településeket tárolja.
- Egy “kedvelés” gomb hozzáadása, ami eltárolná a felhasználó által kedvelt termékeket amik még nem lettek megvásárolva.
- Felhasználókhoz hozzáadni a tulajdonságot, hogy kitörölhetőek legyenek.
- “user” táblába “jogkör” elhelyezése (logikai mező).

4. Felmerült akadályok

- Egy új termék felvételénél elmaradt a termékek méretének és állapotának felvétele az adatbázisba, ez a hiba később az adatbázis és a feltöltés teljes átdolgozását igényelte, hogy orvosolni tudjuk a problémát.
- A kártyák helyes megjelenítésén is sokat kellett dolgoznunk, a feltöltési hibákból adódóan elsőre helytelen adatokat kaptak meg a kártyák.

5. Összefoglalás

A közös munkát nagyban segítette, hogy az iskolában személyesen is együtt tudtunk dolgozni. A live coding, vagyis élő kódolás módszerével gyorsan haladtunk, ötleteinket azonnal egyeztettük és valós időben valósítottuk meg. Emellett otthon, saját gépeinken is folytattuk a fejlesztést, ahol a megszokott környezetünkben még hatékonyabban tudtunk dolgozni.

Jelenleg az alkalmazás helyi környezetben működik, interneten még nem érhető el. A jövőben tervezzük, hogy az oldalt egy publikus felületen is elérhetővé tesszük, amihez domain vásárlására, szerveres háttér kiépítésére és az infrastruktúra átalakítására lesz szükség.

A fejlesztési tervek között szerepel egy beépített fizetési rendszer kialakítása, amely lehetővé teszi, hogy a vásárlók közvetlenül az oldalon belül bonyolíthassák le a tranzakciókat. Emellett szeretnénk egy adatbázis-alapú keresőmotort is létrehozni, amely gyorsabbá és pontosabbá teszi a termékek közötti keresést.

A vásárlás és eladás támogatására egy chat funkció bevezetését is tervezzük, amely lehetőséget ad a felhasználók közötti közvetlen kommunikációra. Kiemelt figyelmet fordítunk a felhasználói biztonságra is: egy figyelmeztetési rendszer kiépítését tervezzük, amely kiszűri a nem megfelelő magatartást, és szükség esetén lehetőséget ad a szabálysértők kizárására.

Hosszú távú céljaink között szerepel a rendszer nemzetközi piacra való bevezetése is, ami az adatbázis átalakítását és a háttérrendszer nemzetközi szabványokhoz való igazítását igényli.

A felhasználói bázis növekedésével az adatbázist is szeretnénk tovább bővíteni, hogy minél többféle terméket és nagyobb mennyiségű adatot tudjunk kezelni, ezáltal is növelve az oldal funkcionalitását és felhasználóbarát jellegét.

6. Források

- balazskicks.com
- <https://www.w3schools.com/>
- <https://getbootstrap.com/>
- <https://stackoverflow.com/questions>
- <https://codepen.io/>