Váci SZC Petőfi Sándor Műszaki Technikum,

Gimnázium és Kollégium

SZOFTVERFEJLESZTŐ ÉS TESZTELŐ

Záródolgozat

Cím: Gamer’s Market

Készítette: Házi Gréti

Csapattárs: Kerti Benjámin

2025

Nyilatkozat

Alulírott Házi Gréti tanuló kijelentem, hogy a záródolgozat saját és csapattársam tudásának eredménye, a felhasznált eszközöket pontosan megjelöltem. A leadott példány a dokumentációnak, valamint digitális formátuma mind formátumban és mind tartalomban megegyezőek, eltérést nem tartalmaznak.

Kelt.: .....

…….……..…..

tanuló

KONZULTÁCIÓS LAP

Szoftverfejlesztő és tesztelő

A tanuló neve: …… Osztálya: ……

A záródolgozat témája: ...…….

……..

……..

Sorszám:	A konzultáció időpontja:	A konzulens aláírása:
1.		
2.		
3.		
4.		
5.		

A záródolgozat beadható.

Aszód, ……

……

konzulens

Tartalomjegyzék

1. Bevezető

1.1. A probléma rövid ismertetése

A Gamer's Market egy olyan webshop, amelyet kifejezetten azok számára hoztunk létre, akik szívesen töltik szabadidejüket videojátékokkal, de nem tudják, hol találhatják meg azokat olcsóbban, kényelmesebben és megbízható környezetben. Célunk, hogy egy olyan webshopot biztosítsunk, ahol a felhasználók könnyedén beszerezhetik a legújabb vagy éppen a régebbi játékokat, mindezt egyszerűen és gyorsan.

A videojátékok világában való elmélyült tapasztalatunknak köszönhetően tisztában vagyunk a vásárlók igényeivel. Webshopunk kialakításakor a megbízhatóságra, a felhasználóbarát élményre és a versenyképes árakra összpontosítottunk. A Gamer's Market tehát nem csupán egy online bolt, hanem egy olyan közösségi tér, ahol a játékosok könnyedén rátalálhatnak kedvenc játékaikra és gyorsan hozzájuthatnak azokhoz. A weboldalunk dizájnja modern, letisztult és a hideg színek dominálnak, amelyek segítik a kényelmes böngészést, miközben a vásárlási élmény gyors és zökkenőmentes. Az oldal egyszerű navigálhatóságával és könnyen áttekinthetőségével igyekszünk a lehető legjobb felhasználói élményt biztosítani. Személyesen is átéltem, hogy milyen fontos a vásárlói élmény: amikor játékokat rendeltem, gyakran találtam magam olyan helyzetben, hogy a legjobb ajánlatot ugyan nem találtam meg a nagyobb webáruházakban, de sokszor nem voltam biztos abban sem, hogy az oldal valóban megbízható. Éppen ezért szerettem volna létrehozni egy olyan helyet, ahol a vásárlók nem kell, hogy ilyen problémákkal szembesüljenek.

A Gamer's Market célja, hogy biztos alapokon álljon, és olyan környezetet biztosítson, ahol a felhasználók minden információt könnyedén elérhetnek, miközben a vásárlás egyszerű és gyors folyamat marad. Mivel én is az ilyen típusú megoldásokat kerestem, így próbáltam olyan webshopot létrehozni, amely számomra is kényelmes lenne, és remélem, hogy mások számára is azzá válik.

2. Felhasználói dokumentáció

2.1. A program általános specifikációja

A Gamer's Market egy olyan webshop, amely a videojátékok iránt érdeklődők számára kínál könnyű és megbízható vásárlási lehetőséget. Az oldal célja, hogy felhasználóbarát, gyors és kényelmes platformot biztosítson a legújabb és régebbi videojátékok beszerzéséhez. Az alábbiakban ismertetem a rendszerkövetelményeket, a program telepítését valamint használatát.

2.2. Rendszerkövetelmények

2.2.1. Hardver követelmények

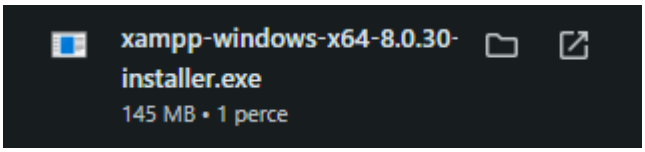
- Memória → 2 GB (Ez a minimális szint, amit biztosítanunk kell.)
- Processzor → 2 mag, 4 GHz (Ez a minimális szint, amit biztosítanunk kell.)
- Operációs rendszer → Windows 10, 11 (Ez a minimális szint, amit biztosítanunk kell.)
- Kijelző →360px X 640px (Ez a minimális szint, amit biztosítanunk kell.)

2.2.2. Szoftver követelmények

- Chrome → 37-es verzió
- Firefox → 34-es verzió
- Opera → 24-es verzió
- Safari → 7-es verzió

2.3. A program telepítése, felkeresése

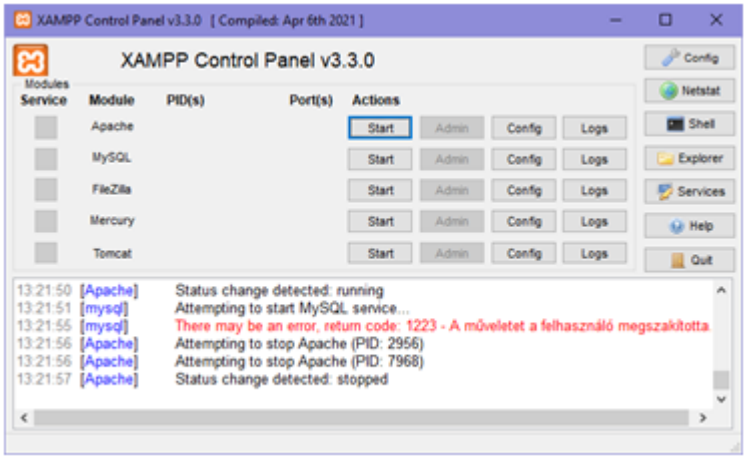
XAMPP (7.2.4 verzió)



1. ábra: Xampp telepítése

2. ábra: Xampp ikonja

phpMyAdmin (8.2.12)



3. ábra: Xampp control panel

4. ábra: phpMyAdmin ikonja

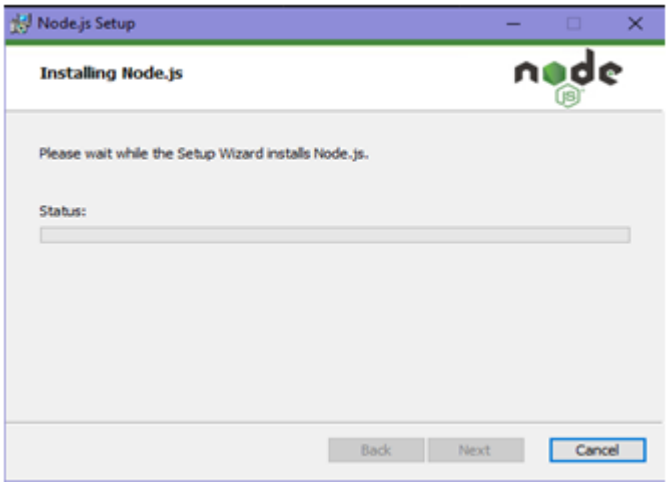
A XAMPP megnyitásakor az alábbi elemeket kell elindítani, az adatbázis elérése érdekében:

→ Apache Start

→ MySQL Start

Ez a felület automatikusan átirányítja felhasználót a phpMyAdmin böngésző alkalmazásába. Ezek után az “importálás” fülön ki kell választani a forrásfájlok között, a “database” mappából a “gamersmarket.sql” fájlt, majd az “indítás” gombbal, lefutattjuk az adatbázisunkat.

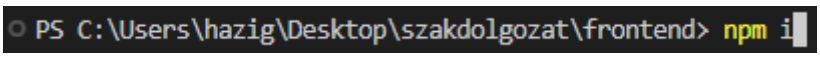
Node.js (23.3.0 verzió)



5. ábra: Node.js telepítése

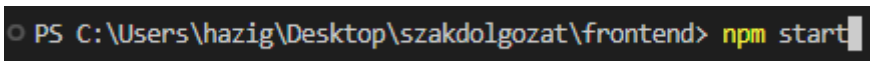
6. ábra: Node.js ikonja

A React alkalmazás fejlesztéséhez az npm (Node Package Manager) kódjait használtam, amelyek lehetővé teszik a csomagok egyszerű telepítését és kezelését. A Visual Studio-ban a "Terminal" menüponton belül, az alábbi kódsort kell begépelni, hogy letöltsük a szükséges fájlokat: "npm i".

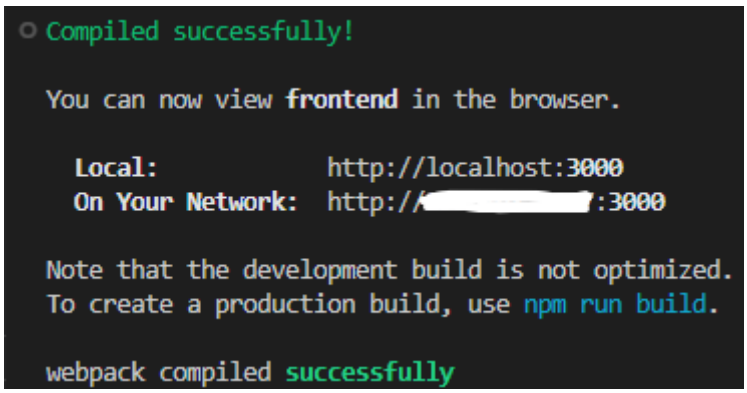


7. ábra: node_modules telepítése

Miután ez letelepült, az indításhoz a következő kód kell: "nmp start".

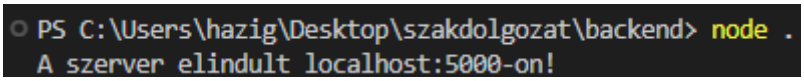


8. ábra: npm start kódja



9. ábra: Sikeres indítás

A szerver indításához, a felhasználó nyissa meg a "backend" mappát, majd a "Terminal" menüpontba, a következő kódsorra lesz szüksége: "node ."



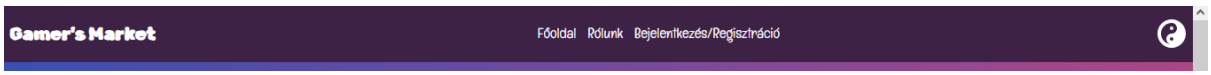
10. ábra: Szerver indítása

Ezzel a szerver el is indult. A weboldalhoz először mindig a backendet kell indítani utána pedig a frontendet.

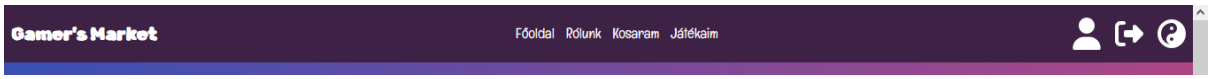
2.4. A program használatának részletes leírása

2.4.1. Menüsor

A menüsornak két verziója van: alap és bejelentkezett. Az alap menüsoron megjelenik az oldalunk neve, a Főoldal, a Rólunk, a Bejelentkezés/Regisztráció és a Tematika választó. A bejelentkezett menüsoron megtalálható ugyanúgy a Főoldal és Rólunk menüpont, de a bejelentkezést követően a menüsáv annyiban változik, hogy megjelent 2 új pont; a Kosaram és a Játékaim. Ikonok terén a tematika választó maradt az alap helyén, ehhez hozzárendeltünk még 2 újat, melyek nem mások, mint a Profil módosítása és a Kijelentkezés ikon.



11. ábra: Menüsor bejelentkezés előtt



12. ábra: Menüsor bejelentkezés után

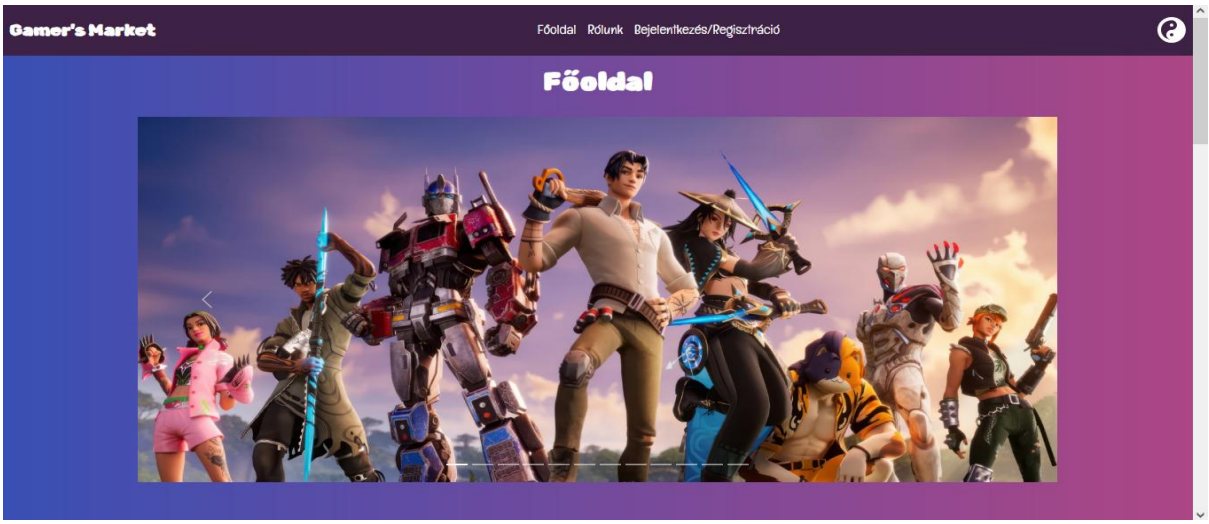
2.4.2. Lábléc

A Láblécbe az Elérhetőségek fülhöz 4 linket csatoltunk, melyek a következők; Discord, Facebook, X és Instagram. Ezekhez, a hivatalos oldalukat csatoltuk, ugyanis webshopunknak még nincs hivatalos közösségi oldala egyik platformon sem. Ezt a későbbiekben továbbfejlesztjük és orvosoljuk.

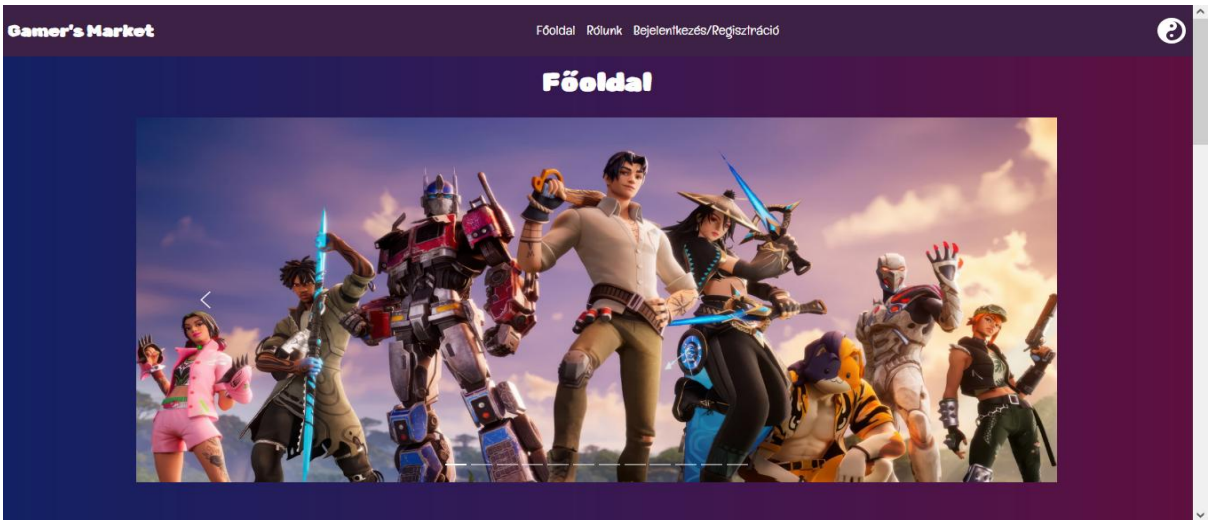
13. ábra: Lábléc

2.4.3. Főoldal

A főoldalon két körhinta és 12 kártya helyezkedik el. Az oldal tetején képek vannak, míg az alján videók a körhintában. A kártyák bejelentkezés után megváltoznak, megjelenik rajtuk a kosár gomb is. A körhinta elemek és a kártyák maga a következő játékokat tartalmazzák; Fortnite, Minecraft, PUBG, Forza Horizon 5, The Witcher 3, GTA 5, Resident Evil 7, Counter-Strike 2, Sims 4, EA FC 25, CoD:Warzone, Red Dead Redemption 2. A képeket a világhálóról töltöttem le és helyeztem el a webshopon. A főoldalon megtalálható még egy tematika választó ikon is, melyet a “yin-yang” szimbolizál. Az ikonra kattintva felhasználó tudja változtatni a webshop háttérszínét “világos” és “sötét” mód között. Az alábbi képeken demonstrálom ezeket:



14. ábra: Világos főoldal

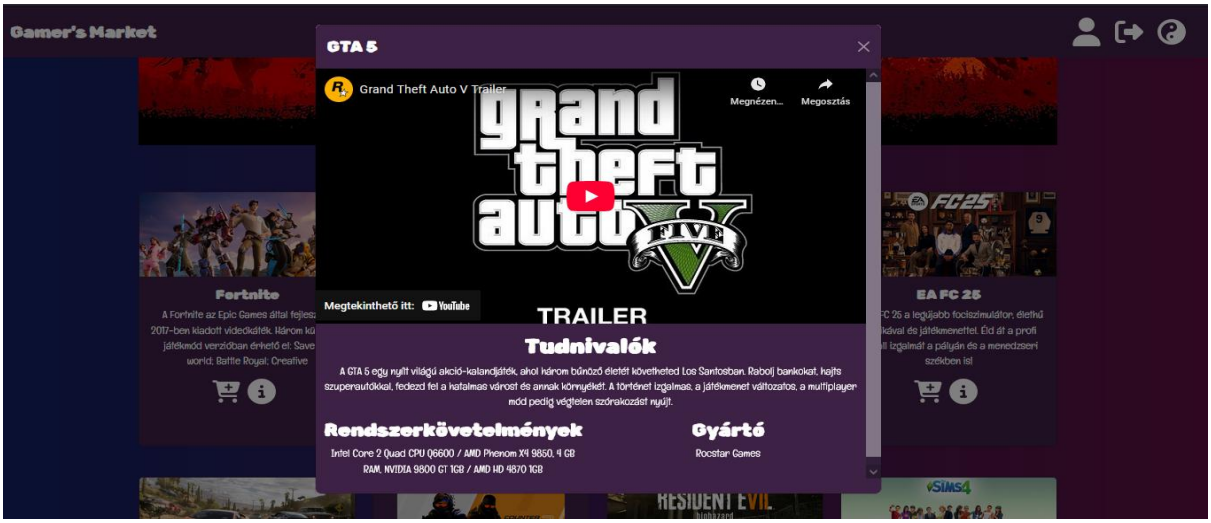


15. ábra: Sötét főoldal

2.4.4. Információs gomb / Modal

Az információs gomb, egy “i” betű ikonként jelenik meg a webshopon. Ha a felhasználó rákattint a weboldalon, az egy “Modal”-ként nyílik meg, ebben megjelenítve az adott kártya tartalma. Mielőtt a felhasználó rákattint az információs gombra a kártyán, először a játék nevét, képét és leírását tekintheti meg. Miután rákattintott a vásárló, elé tárul az adott kártya összes adata, beleértve a játék címe, egy előzetes videó róla, egy bővebb leírás, gépigény és a játékot forgalmazó cég neve is.

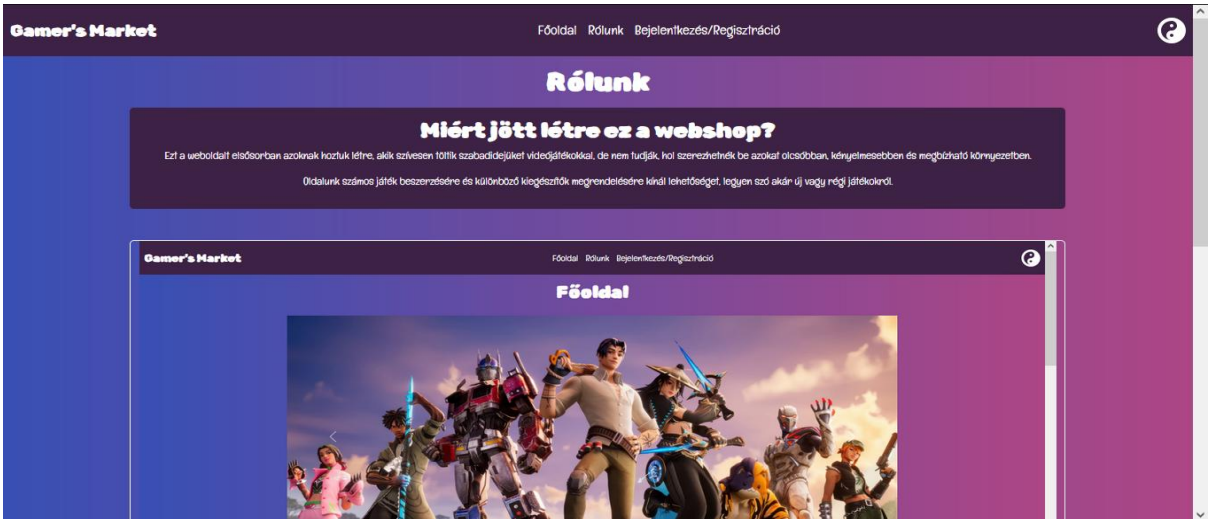
Az információs gomb a bejelentkezés után is ugyanígy működik, annyi különbséggel, hogy az oldal látogató kosárba is tud helyezni egyes terméket.



16. ábra: Információs gomb/Modal

2.4.5. Rólunk

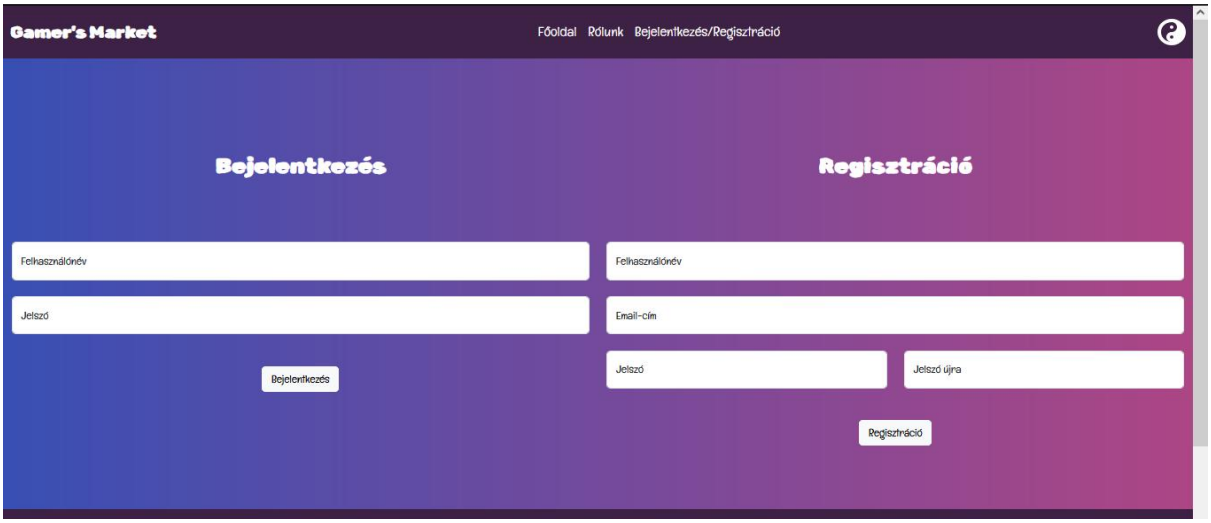
A rólunk oldalon két információk, egy kép és 3 kártya helyezkedik el. A kártyában a készítők, azaz mi vagyunk és a konzulensünk, aki felügyelte a projekt menetét. Az oldalon van egy fotó a Főoldalról a design kedvéért és egy kisebb leírás az oldal tetején, az oldal létrejöttéről.



17. ábra: Rólunk

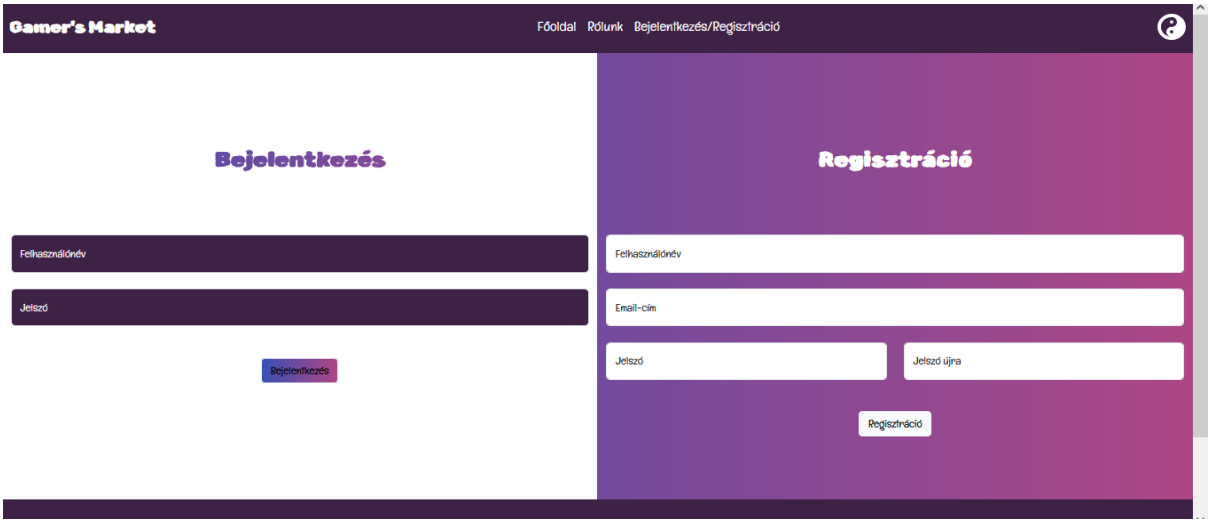
2.4.6. Bejelentkezés/Regisztráció

A bejelentkezés/regisztráció oldalon két oszlopban helyezkedik el a bejelentkezés és a regisztráció. A vásárláshoz regisztrálni kell, így előbb a megadott mezőkkel regisztrálni, majd be is kell lépni.

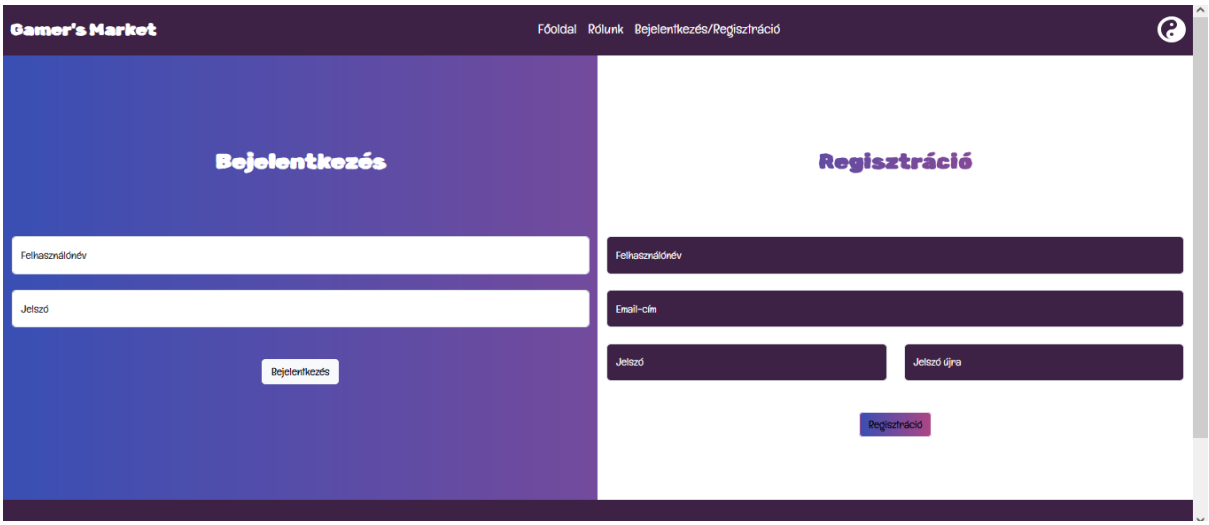


18. ábra: Bejelentkezés/Regisztráció

A bejelentkezés/regisztráció oldal szépsége, hogy ha rávisszük az adott oszlopra az egeret, akkor fehér színű lesz a háttér színe. Azért működik ez így, hogy kihangsúlyozza éppen melyik résznél vagyunk, ezáltal egyértelművé válik a felhasználónak hol van jelenleg a hangsúly a weboldalon.



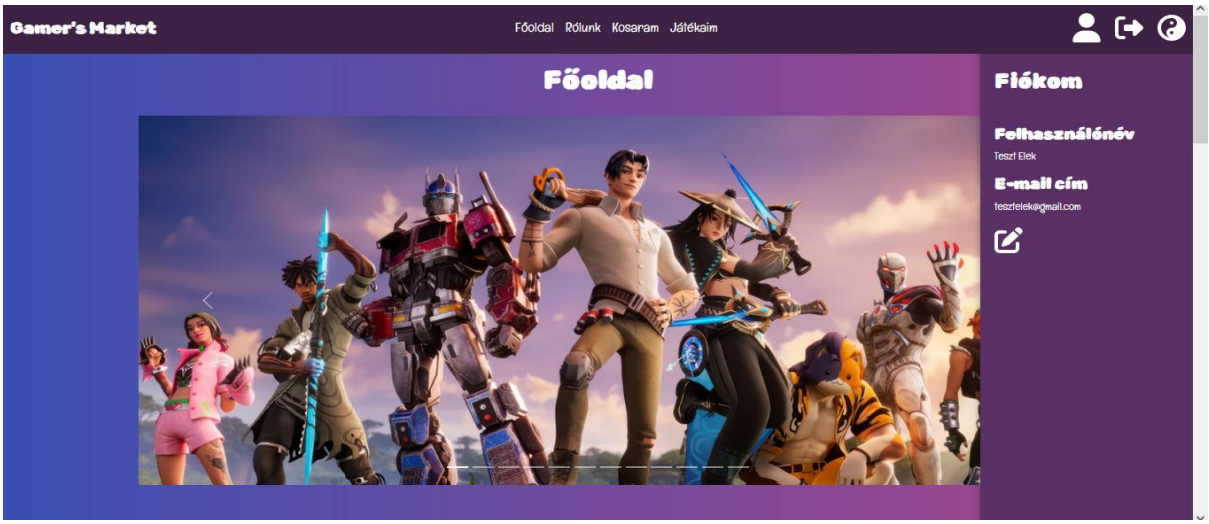
19. ábra:Bejelentkezés/Regisztráció (egér a bal oldalon)



20. ábra: Bejelentkezés/Regisztráció (egér a jobb oldalon)

2.4.7. Fiókom

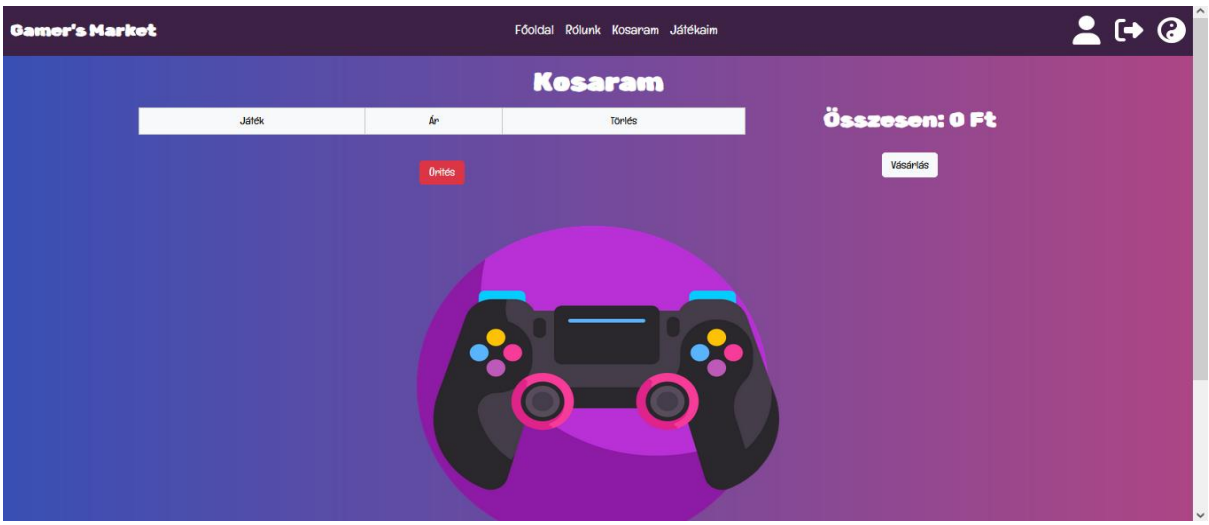
A fiókom részleg csak bejelentkezés után elérhető és bármely oldalon. Itt módosíthatjuk felhasználónevünket és email címünket.



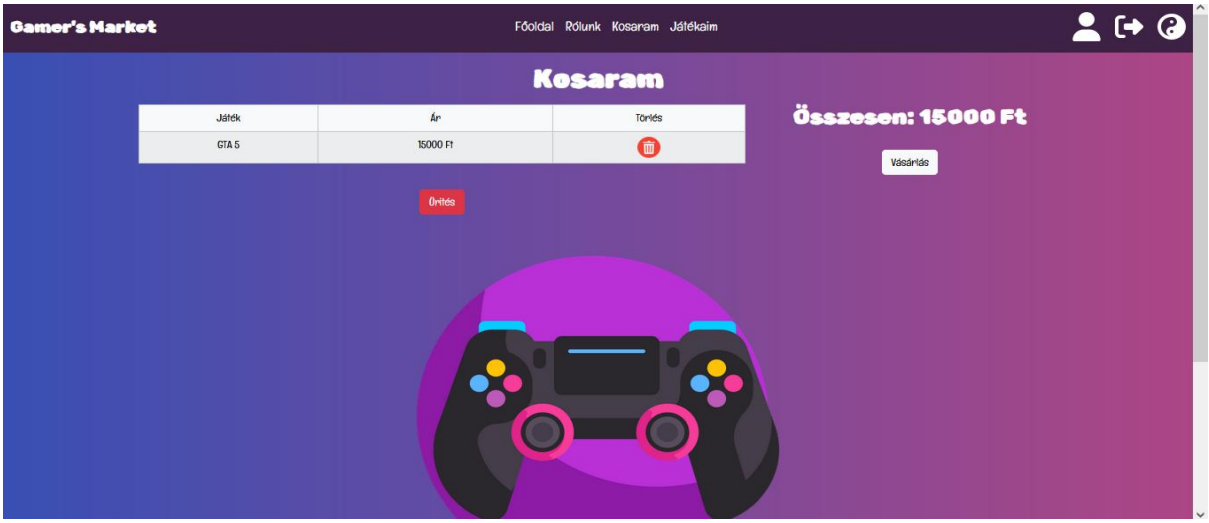
21. ábra: Fiókom

2.4.8. Kosaram

A kosaram oldalon táblázatban vannak a kosár tartalmai bal oldalt, míg jobb oldalt az összeg és a vásárlás gomb. Itt láthatja a felhasználó a megvásárolt termékeinek összesített árát.



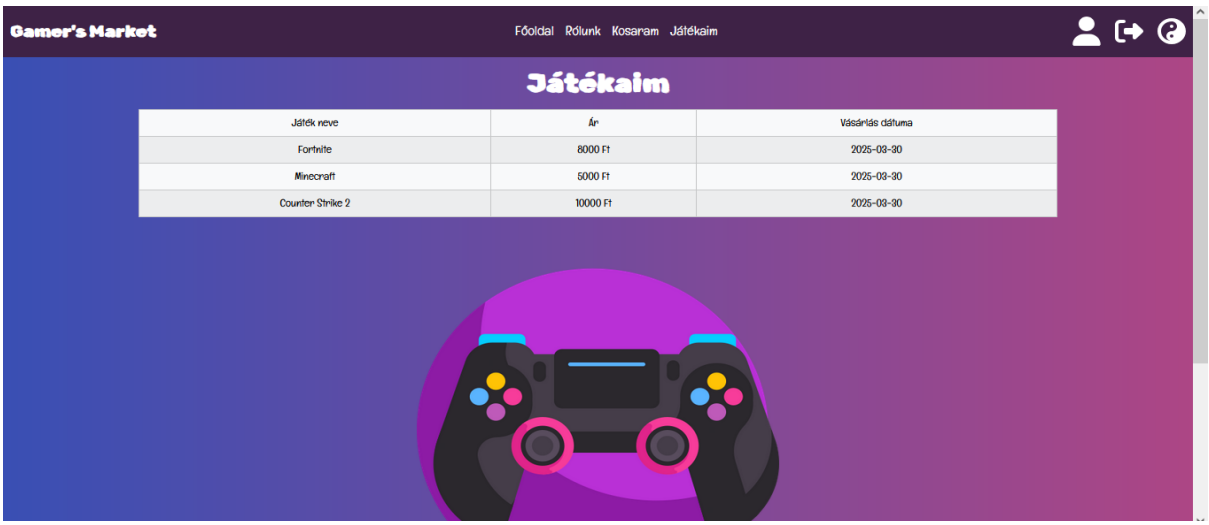
22. ábra: Üres kosár



23. ábra: Teli kosár

2.4.9. Játékaim

A játékaim oldalon a megvásárolt játékaink találhatóak meg táblázatban foglalva. Az oldalon 3 kategória szerint vannak elmentve a vásárlás adatai; az adott játék neve, a játék ára és a játék megvásárlásának az ideje, hogy a vásárló észben tudja tartani tranzakcióit.



24. ábra: Játékaim

3. Fejlesztési dokumentáció

3.1. Témaválasztás indoklása

A szakdolgozat témájának kiválasztásakor fontos szempontnak tartottuk, hogy a projektünk egyszerre feleljen meg a személyes érdeklődésüknek és az eddig tanultak elsajátításának.

A vizsgaremek elkészítéséhez React-ot használtam, mellyel egy dinamikus és felhasználóbarát felületet alakítottam ki, a React-Bootstrap pedig az oldal reszponzívitásáért volt felelős. A fejlesztés során a Visual Studio Code-ot használtam, itt történt meg a kódolás lényeges létrejötte. A munka haladását és gördülékenységét a GitHub jelentette, ugyanis ezt a felületet használtuk a kód tárolására és megosztására egymás között. A Github repository az alábbi linken található meg: https://github.com/Gretika7/szakdolgozat

A webshop felépítése a szerver-kliens architektúrára épül, amely lehetővé teszi a frontend és a backend közötti hatékony kommunikációt. A backend fejlesztéséhez Node.js-t használt csapattársam, amely egy aszinkron, eseményalapú JavaScript futtatókörnyezet. Az adatokat adatbázisban tároljuk, amelyet importálni kell a XAMPP-ban.

A frontend és a backend közötti kommunikáció HTTP protokollon keresztül valósul meg, amely biztosítja az adatok biztonságos és gyors átvitelét. A frontend React komponensei a felhasználói interakciókra reagálnak és a backend API-ján keresztül kérik le az adatokat, amelyeket az adatbázis tárol.

3.2. Az alkalmazott fejlesztői eszközök

A projekt során számos fejlesztői eszközt és technológiát alkalmaztam melyek a következők voltak;

3.2.1. Kimondott fejlesztői felület

- Visual Studio Code (VS Code)
- XAMPP
- Node.js
- Google Docs
- GitHub
- Discord

3.2.2. A programozási nyelv maga

(JavaScript XML) a React által használt szintaxis, amely lehetővé teszi, hogy HTML-szerű elemeket írjunk JavaScript kódban. Ez nagyban megkönnyíti a felhasználói felületek fejlesztését, mivel a komponensek felépítése és a logika összekapcsolása intuitívabbá válik. A JSX használata egyszerűsíti a kód olvashatóságát és karbantarthatóságát, valamint hatékonyabbá teszi a weboldalak dinamikus tartalmának kezelését.

3.2.3. Használt csomagok alkalmazása

- npm start: Ez a parancs indítja el a React alkalmazást a fejlesztői szerveren, lehetővé téve a változtatások azonnali megtekintését.
- node . : Ezzel a paranccsal futtattam a Node.js szkripteket a háttérben, például a szerver oldali logikát vagy az adatbázis kapcsolatot.
- npm install: Ezzel a paranccsal telepítettem a szükséges Node.js csomagokat és függőségeket, amelyek elengedhetetlenek voltak a webshop működéséhez. A telepített csomagok a package.json-ben található “dependencies” szekcióban találhatók és tartalmazzák a projekt működéséhez szükséges könyvtárakat és eszközöket.

3.3. Adatmodell leírása

Az adatbázis létrehozása csapattársam feladat volt. Az adatbázis neve a gamersmarket névre hallgat, amelyben 3 tábla lett elhelyezve. A táblák nevei: games, shopping és users. A táblák között van kapcsolat, mégpedig a games.id = shopping.gameid, users.id = userid.

3.3.1. games tábla mezőnevei

- id → INT, AUTO_INCREMENT, PRIMARY KEY
- title → VARCHAR (255)
- short_description → VARCHAR (255)
- long_description → TEXT
- system_requirements → TEXT
- company → VARCHAR (255)
- picture → VARCHAR (255)
- link → VARCHAR (255)
- price → INT

3.3.2. shopping tábla mezőnevei

- userid → INT, FOREIGN KEY
- gameid → INT, FOREIGN KEY
- purchase_date → DATE

3.3.3. users tábla mezőnevei

- id → INT, AUTO_INCREMENT, PRIMARY KEY
- username → VARCHAR (255)
- email → VARCHAR (255)
- password → VARCHAR (255)

A games táblába 12 játék adata van beszúrva a mező neveknek megfelelően, míg a többiben nincs semmi adat. A másik két táblában csak a felhasználó által jelennek meg adatok.

25. ábra: Táblák kapcsolása

3.4. Részletes feladatspecifikáció,algoritmusok

3.4.1. Játékok kosárba fetchelése

```
export default () => {
    const [data, setData] = useState([]);
    useEffect(()=>{
        const fetchCard = async () =>{
            try {
                const response =  await fetch("http://localhost:5000/games",{
                    headers:{
                        "Content-Type":"application/json"
                    }
                });
                if(!response.ok){
                    throw new Error("Nem sikerült a játékokat leszedni!");
                }
                const jsonData = await response.json();
                setData(jsonData);
            }catch(err){
                console.error(err);
                alert(err.message);
            }
        };
        fetchCard();
    }, []);
```

26. ábra: Játékok fetch-je

```
<Row className="mt-4 mb-4">
    {data && data.map((item, index) => (
        <Col key={index} xs={12} md={3}>
            <Card data={item} addToCart={() => addToCart(item)}/>
        </Col>
    ))}
</Row>
```

27. ábra: Játékok kiíratása

A webshopoknál gyakran szükséges külső API-ból származó adatokat megjeleníteni. A React fetch API-ja hatékony eszközt biztosít az aszinkron adatlekéréshez. A következő kódrészlettel részletesen bemutatom ennek gyakorlati kivitelezését.

A komponens elsősorban a "useState" eseménykezelővel kezeli a lekérdezett játék adatokat (data). A kezdeti érték egy üres tömb. A komponens betöltésekor a "useEffect" parancs meghívja a "fetchCard" aszinkron függvényt. A "fetchCard" függvény a fetch API-t használja lekéréshez, ami a "http://localhost:5000/games" végponton keresztül fut. A fetch függvény egy “GET” kérést küld a megadott URL-re és a "headers" opcióval beállítja annak visszaadási értékét, melynek válaszát a "Promise" objektum tárolja. Beállítjuk a szerver válaszokat is az "ok" és az "error"

parancsokkal. A "setData" segítségével a lekérdezett adatok (jsonData) komponens állapotba kerülnek. A "try/catch" blokk segít a komponensnek kezelni a fetch kérés során fellépő komplikációkat. Az adatok megjelenítéséhez szükségünk lesz pár "Row" és "Col" komponensre. A "Row" komponens egy sort hoz létre, míg a "Col" egy oszlopot deklarál. Az ezeknél megadott "xs", "md" parancsok méretezési beállítások, melyek ahhoz kellenek, hogy az oldal tökéletesen illeszkedjen bármely képernyőmérethez. A "data.map()" függvény segítségével a lekérdezett adatok tömbjének minden elemére létrehozunk egy "Card" komponenst.

A "Card" komponens egy rugalmas és bővíthető tartalom tároló.

3.4.2. Regisztrációs felület fetchelése

```
const [regUsername, setRegUsername]=useState("");
const [regEmail, setRegEmail]=useState("");
const [regPassword, setRegPassword]=useState("");
const [regPasswordAgain, setRegPasswordAgain]=useState("");
const [error, setError]=useState("");

const [logUsername, setLogUsername]=useState("");
const [logPassword, setLogPassword]=useState("");

const navigate = useNavigate();

const handleRegister = async (e)=>{
    e.preventDefault();
    try {
        if (regPassword === regPasswordAgain) {
            const response = await fetch("http://localhost:5000/register",{
                method:"POST",
                headers: {
                    "Content-Type":"application/json",
                },
                body: JSON.stringify({
                    "username": regUsername,
                    "email": regEmail,
                    "password": regPassword
                })
            });
            const data = await response.json();
            if(response.ok){
                alert("Sikeres regisztráció!");
                setRegUsername("");
                setRegEmail("");
                setRegPassword("");
                setRegPasswordAgain("");
                navigate("/login-or-registration");
            }else{
                setError(data.message);
            }
        }
        else {
            alert("A két jelszó nem egyezik!");
        }
    } catch (err) {
        setError("Valami hiba történt!");
    }
}
```

28. ábra: Regisztráció fetch-je

Ez a kódrészlet egy React alapú regisztrációs felületet valósít meg, mely lehetővé teszi az új felhasználó részére, hogy saját fiókot hozhasson létre.

A "regUsername", "regEmail", "regPassword" állapotkezelők a bejelentkezéshez szükséges fő adatokat tárolják el. "regPasswordAgain"-el egyfajta megerősítő eljárást hajtottunk végre. A "setError" állapotkezelőben a regisztráció során fellépő hibák üzeneteit tároljuk el. A "useNavigate" állapotkezelő a React Router segítségével jön létre, mely egyfajta navigációs függvényt hoz létre, amellyel a webshop különböző oldalaira lehet átirányítani a felhasználót. A "const handleRegister = async (e) => {...}" egy aszinkron függvény, amely a regisztrációs űrlap elküldésekor hívódik meg, míg az "e.preventDefault()" parancs megakadályozza az űrlap újratöltését. A "try/catch" hibakezelő blokk, amely a regisztráció során fellépő hibákat kezeli. Az "if" elágazás "===" eleme, ellenőrzi, hogy a megadott jelszó és megerősítő jelszó egyezik-e. Ha egyezik, akkor az oldal egy "POST" kérést küld a szervernek a felhasználó adataival, amely a "http://localhost:5000/register"-en található meg, majd a szerver válaszát JSON formátumban tárolja el. Ha a szerver sikeresen megkapta a kéréseket akkor egy "alert"-ben jelzi azt a felhasználónak, majd a "navigate" paranccsal, átirányítja azt a bejelentkezési felületre.

Sikertelen regisztráció során a felhasználónak különböző "error" üzeneteket küld a szerver, melyekből rá tud következtetni a hibájára.

3.4.3. Bejelentkezési felület fetchelése

```
const handleLogin = async (e)=>{
    e.preventDefault();
    setError(null);
    try{
        const response = await fetch("http://localhost:5000/login",{
            method: "POST",
            headers:{
                "Content-Type":"application/json",
            },
            body:JSON.stringify({
                "username": logUsername,
                "password": logPassword
            })
        });
        const data = await response.json();
        if(!response.ok){
            setError(data.message);
            return;
        }
        alert("Sikeres bejelentkezés!");
        const token = data.token;
        if (!token) {
            alert("Hiba!");
        }
        else {
            localStorage.setItem("token", token);
            navigate("/");
            window.location.reload();
        }
    }catch(err){
        setError("Valami hiba történt a bejelentkezés során!");
    }
}
```

29. ábra: Bejelentkezés fetch-je

A kódrészlet egy React alapú bejelentkezési felületet mutat be, amely lehetővé teszi a felhasználónak, hogy előzetes regisztráció után, be tudjanak lépni a weboldalba.

A kódrészlet elején a “useState” állapotkezelő segítségével egy változót definiálunk, a bejelentkezés során fellépő hibák tárolására. A “handleLogin” függvény a bejelentkezési űrlap elküldésekor hívódik meg és ez a függvény aszinkron módon kezeli a bejelentkezési műveleteket. Az “e.preventDefault()” parancs, megakadályozza az alapértelmezett űrlap küldési viselkedését. A fetch API-val “POST” kérést küld a /login végpontra a felhasználó adataival, majd a szerver válaszát JSON formátumban dolgozza fel. Az “await” részben ellenőrizzük, hogy a szerver válasza sikeres-e volt. A "(!response.ok)" ellenőrzi, hogy a szerver válasza sikeres volt-e, aztán ha az sikertelen volt, beállít egy hibaüzenetet, majd visszatér a függvényből. Ha sikeresnek bizonyul az ellenőrzés, megjelenít egy sikeres bejelentkezés megerősítő üzenetet, aztán pedig kinyeri a tokent a szerver válaszából.

A következőkben, annak létezését ellenőrzi, majd elmenti az adott böngésző helyi tárolójába, aztán átnavigál a főoldalunkra és újratölti az oldalt.

Token generáláskor ha hiba merül fel, a szerver hibaüzenetet küld. Ez a kódrészlet kimondottan a biztonságos bejelentkezésért felel, a “token” alapú hitelesítéssel.

3.5. Forráskód

3.5.1. A React-Bootstrap szerkezet gyakorlati felépítése

Ez a kódrészlet tökéletesen bemutatja az egész vizsgaremekre kiható React-Bootstrap alkalmazását. Ez a Card.jsx komponens dinamikusan kezeli a felhasználó bejelentkezési állapotát és lehetővé teszi a kiválasztott termékek kosárba helyezését. Ez a kódrészlet megfelelő példaként szolgál, hogy bemutassa oldalam dinamikáját és interaktivitását.

A Card.jsx komponens a következő főbb elemekből épül fel:

- Importok
- Állapotkezelés
- Mellékhatás
- Kártya megjelenítés
- Feltétek
- Modál

A kód legeslegelején a szükséges modulok beimportálása történik. A react-bootstrap könyvtárából a “Card” és “Image” komponenseket telepítjük be, hisz ezek adják a kód alapját. Következő lépésben a ”Modal” komponens jön sorra, mely az adott termék részletes adattárolásáért felel, majd a 2 állapotkezelő parancs, a “useState” és “useEffect”. Az oldal keretét megadják a “Col” és “Row” parancsok. A “data” és az “addToCart” függvény segít a kosárba helyezés funkcionalitásának létrejöttében. A “useState” állapotkezelő segítségével, a komponens kezeli a felhasználó bejelentkezési adatait (isLogged). A “useEffect” a komponens betöltésekor fut le, és ellenőrzi, hogy a felhasználó be van-e jelentkezve a “localStorage”-"token" értékének lekérdezésével. A “Card” komponens segítségével egy termék kártya kerül megjelenítésre, amely tartalmazza a következőket; a termék képe (Card.Img), címe (Card.Title), rövid leírását (Card.Text). Az “isLogged” állapot

alapján a komponens feltételesen jelenít meg egy “Kosár” ikont (Image). Amennyiben a felhasználó be van jelentkezve, az ikon kattintható és meghívja az “addToCart” függvényt. A React-Bootstrap keretrendszere a Bootstrap alapjaira épül, ezáltal lehetővé teszi, hogy az oldalamon létre tudjak hozni reszponzív elemeket. A “Row” komponens egy sor létrehozására szolgál, míg a “Col” komponensek oszlopokat definiálnak a soron belül. A “Col” komponensek xs, sm, md, lg, xl és xxl méretezésekkel rendelkeznek, amelyek lehetővé teszik a különböző képernyőméretekhez az alkalmazkodást.

```
import {Card, Image} from 'react-bootstrap';
import Modal from "./Modal.jsx";

import { useState, useEffect } from "react";

export default ({ data, addToCart }) => {
    const [isLogged, setIsLogged] = useState(false);

    useEffect(() => {
        const token = localStorage.getItem("token");
        setIsLogged(!!token);
    }, []);

    return (
        <Card className='mt-4 mb-4' style={{backgroundColor:"#3e2246" , color:"white", textAlign:"center", height:
        "400px"}}>
            <Card.Img variant="top" src={`../images/games_images/${data.picture}`}  />
            <Card.Body>
                <Card.Title className='title'>{data.title}</Card.Title>
                <Card.Text>{data.short_description}</Card.Text>
                {isLogged ? <Image style={{cursor:"pointer", width:"40px", marginRight:"10px"}} src ="../images/cart.
                svg" title="Kosár" onClick={addToCart}></Image>: ""}
                <Modal data={data}></Modal>
            </Card.Body>
        </Card>
    );
}
```

30. ábra: Card.jsx

3.5.2. Tematika választás funkciójának részletes leírása

```
const storedTheme = localStorage.getItem("theme") || "light";
const [theme, setTheme] = useState(storedTheme);
const [showSidePanel, setShowSidePanel] = useState(false);
const [isYingYangClicked, setIsYingYangClicked] = useState(false);

const toggleTheme = () => {
    const newTheme = theme === "light" ? "dark" : "light";
    setTheme(newTheme);
    localStorage.setItem("theme", newTheme);
    document.body.className = newTheme;
};

const handleUserClick = () => {
    setShowSidePanel(!showSidePanel);
};

const handleYingYangClick = () => {
    setIsYingYangClicked(!isYingYangClicked);
    toggleTheme();
};

document.body.className = theme;
```

31. ábra: Tematika választó

```
<Image
    className={`yingyang-icon ${isYingYangClicked ? 'active' : ''}`}
    style={{ cursor: "pointer", width: "40px", marginRight: "10px", marginLeft: "10px" }}
    src="../images/yingyang.svg"
    title="Téma váltás"
    onClick={handleYingYangClick}
/>
```

32. ábra: Tematika választó ikonja

```
body.light {
    background: linear-gradient(to right, #394fb4, #ad4685);
}

body.dark {
    background: linear-gradient(to right, #122164, #5e1040);
}
```

33. ábra: Tematika css-e

Ez a kódrészlet egy React komponens részlete, mellyel kedvünkre tudjuk állítani az oldal tematikáját "világos" és "sötét" mód között egy oldalsó panel megnyitása közben. A "storedTheme" rész lekéri a korábban mentett témát a böngésző helyi tárolójából. Ha nincs mentett, akkor alapértelmezetten világosra rakja.

A "[theme,setTheme] = useState(storedTheme)" létrehoz egy állapotváltozót a téma tárolására és frissítésére, melynek kezdeti értéke a "storedTheme". A "showSidePanel" elemmel, létrehozunk egy állapotváltozót az oldalsó panel megjelenítéséhez/elrejtésére, mely "false" értékkel kezdetlegesen rejtett. A "setIsYingYangClicked" változóval, kattintásra tárolja a hozzá rendelt utasítást és ennek is kezdeti értéke ugyanúgy "false" lesz. A téma váltást maga, a "toggleTheme" függvény kezeli. Ehhez tartozik a "newTheme" elem is, ahol "===" parancsok között tudjuk eldönteni, hogy "light" (világos) vagy "dark" (sötét) mód lépjen érvénybe a weboldalon. Ezeket a "setTheme(newTheme)" parancsal tudjuk firssíteni. A "localStorage.setItem"-el elmentjük az újonnan létrehozott témát az aktuális böngésző helyi tárolójába. "className"-ek között határoztuk meg az ikon CSS dizájnolását. Az oldalsó panel kezeléséhez, annak megjelenítéséhez és elrejtéséhez kattintással, a "handleUserClick" függvényt használjuk. Ezen felül, a "setShowSidePanel"-t is alkalmazzuk, mely állapotváltozó tulajdonságából kiindulva, megjeleníti vagy akár el is tudja rejteni a panelt. A Yin-Yang ikon kezeléséhez, itt is hasonlóan járunk el, mint a panel megnyitásánál, ugyanis egy "handleYingYangClick" függvényre lesz szükségünk a kattintások kezeléséhez. Hasonló esethez hűen, itt is a "setIsYingYangClicked" állapotváltozóval fordítjuk meg az alapértékeket. A "toggleTheme()" meghívja a témaáltáshoz nélkülözhetetlen függvényt. A háttér

megváltoztatásához a következő kóddal tudunk változást elérni; "document.body.className = theme", ugyanis a "body" elemet így aktiváljuk, ami a weboldalunk alaptörzse. Az ikon megjelenítéséhez és annak alkalmazásához az "<Image>" kóddal kell kezdenünk, majd ezután a "className"-el dinamikusan beállítjuk az ikon osztálynevét, az "isYingYangClicked" állapotváltozó alapján. Ezt követően beállítjuk az ikon stílusát a "style" elemmel. Az ikon elérési forrását és címét az "src" és "title" parancsokkal érthetjük el. Legvégül, de nem utolsó sorban kötelező beállítanunk az "onClick" parancsot, ugyanis ez beállítja az ikon kattintáskezelőjét a "handleYingYangClick" függvényre, ami által lép működésbe az ikon a weblapon.

A "body.light" és a "body.dark" beállítja a "<body>" elem stílusát, ha annak tematikája világos vagy pedig sötét.

3.6. Tesztelési dokumentáció

A webshop elkészítése közben számos olyan hibákat véltem felfedezni, mely olykor kihatással volt az oldalra, olykor pedig inkább funkcionálisan befolyásolta azt. Programozás közben folyamatosan ellenőriztem magam és a munkám, így a végtermék összességében elfogadható lett.

Íme pár probléma, melyek meglettek oldva valamilyen formában vagy pedig továbbfejlesztési célként abszolút kitűzhetőnek gondoltam:

- Amikor a felhasználó leteker a weboldal legaljára, majd oldal váltáskor, az nem dobja Őt vissza a tetejére automatikusan. Továbbfejlesztési lehetőségek között ez is szerepel.
- A backend nem egyidőben haladt a frontenddel, így a fetcheknél saját magam kellett ideiglenes szervert írjak, hogy tudjam tesztelni a frontenden lévő funkciókat.
- Olykor újra kell tölteni manuálisan a weboldalt, ugyanis nem töltődnek be a megadott adatok. Ez is a továbbfejlesztési terveket erősíti.
- Bejelentkezés után a kosárba helyeztem egy játékot. Kosár oldalra kattintva, a kosárban találtam elmentve a kiválasztott játékot, tehát sikeres volt a művelet.
- A képek igényes, oldalhoz igazított méretezése és formázása is a jövőbeli munkálatokat erősítik.

- Kezdetben nem tudtam, hogy hogyan valósíthatnám meg azt, hogy Bejelentkezés után az oldal navigációs sávja váltson át/frissüljön. Ezt a későbbiekben orvosoltam egy "if" elágazással.
- A képek letöltése után, nem állítottam át a formátumot, ezért rosszul hívtam meg a kódban, mely a webshop kisebb összeomlását eredményezte. Kis gondolkodás elteltével rájöttem, hogy mi a hiba és kijavítottam azokat.

3.7. Továbbfejlesztési lehetőségek

Mivel ez egy fiktív webshop, kiadásra nem szeretnénk bocsátani, így ha a közeljövőben ki szeretnénk adatni, a következő dolgokat kellene továbbfejleszteni a webshopon:

→Webhely bérlése vagy megvásárlása, licensek megszerzése, fizetési módok leprogramozása.

→ Komment szekció, ahol a felhasználó tud véleményt formálni, vagy maga az adott játékról vagy építő jellegű kritikát fogalmazhat meg a webshop fejlesztéséhez.

→ Keresőmotor létrehozása, hogy a vásárló még inkább céltudatosan tudjon válogatni kedve szerint.

→ Játékok adatbázisának bővítése, szinte korlátlan játék mennyiség.

4. Összegzés

A szakdolgozat írása során számos kezdeti nehézséggel kellett szembenéznem. Eleinte nem volt könnyű eldönteni, hogy honnan is kezdjem, hiszen a téma kiválasztása és a megfelelő források összegyűjtése komoly kihívást jelentett.

A konzultációk során a konzulens segített tisztázni a kérdéseket és irányt mutatott a megvalósításban. Ahogy haladtam a kiszabott feladat részekkel, egyre inkább fejlődtem maga a kódolásban, a kutatásban és a problémák megoldása is könnyebbé váltak. A konzultációk alatt a konzulens és a csapattársam folyamatosan támogatott, tanácsokat adott és segített a felmerülő akadályok leküzdésében.

A csapatban való együttműködés is kulcsszerepet játszott; a csoporttársammal való kommunikáció és eszmecserék rengeteget tettek a projektünk elkészítéséhez. Legelső konzultációnk alkalmával kiosztottuk egymás közt a feladatokat, majd folyamatos egyeztetések során elkészült a projektünk.

A szakdolgozat elkészítése során nemcsak szakmai ismereteket sajátítottam el, hanem rengeteget fejlődött a problémamegoldó képességem és az időm produktívabb beosztása is. Ezek az élmények és kihívások hozzájárultak a szakmai és személyes fejlődésemhez és bízom benne, hogy a jövőben is hasznosítani fogom ezeket.

5. Forrásjegyzék

React:

https://react.dev/learn/installation

React-Bootstrap:

https://react-bootstrap.netlify.app/

XAMPP:

https://www.apachefriends.org/download.html

Visual Studio Code:

https://code.visualstudio.com/download

Npm:

https://www.npmjs.com/

Github:

https://desktop.github.com/download/

JSX:

https://legacy.reactjs.org/docs/introducing-jsx.html

Chrome:

https://en.wikipedia.org/wiki/Google_Chrome

Firefox:

https://en.wikipedia.org/wiki/Firefox

Opera:

https://en.wikipedia.org/wiki/Opera_(web_browser)

Safari:

https://en.wikipedia.org/wiki/Safari_(web_browser)

Font Awesome:

https://fontawesome.com/

Flaticon:

https://www.flaticon.com/

Molnár Máté Norbert órai anyagok:

https://drive.google.com/drive/folders/1yEeW86-o30DlX70XIV2obDheUKhsdRL6

6. Ábrajegyzék