

Szakképesítés megnevezése: Szoftverfejlesztő és -tesztelő

Azonosító száma: 5 0613 12 03

Vizsgaremek



Készítették:

Szabó Zoltán

Osztály:13.C

Oktatási azonosító: 72595527403

Szabó Bence Dániel

Osztály 13.C

Oktatási azonosító: 72595526863

Békéscsaba, 2024/2025

Tartalomjegyzék

Bevezetés	2
Témaválasztás indoklása	2
Célkitűzés	3
Fejlesztői dokumentáció	3
Fejlesztői Környezet	3
Felhasznált programok/eszközök	3
Adatbázis felépítése, működés.....	5
Adatbázis ismertetése.....	5
Adatbázis felépítése.....	5
Books és kategóriák tábla	5
Users tábla	6
User_tokens tábla.....	7
Game tábla	7
Wishlist tábla.....	8
Orders tábla.....	8
Ordered_book tábla	9
Password reset tábla.....	9
Frontend	9
Navigáció.....	9
Kezdőoldal felépítése.....	10
További oldalak	10
Footer	11
Ajax	12
Backend.....	12
Backend felépítése.....	12
Regisztráció.....	13
Login.....	13
Logout	14
Check token	14
Order validation és process	15
Admin oldal	15
Kívánságlista	16
Cart	17
A játék.....	17
Grafikai rész	17

Funkcionális rész	17
Kartyak.cs	17
Database.cs	18
Reszponzivitás	18
User interface	18
User Experience	19
Restful és rest apis útvonalak	20
Restful API	20
Rest API	20
Tesztelés	21
Index teszt.....	21
Registration teszt.....	21
Login teszt	21
Játék teszt.....	22
Felhasználói dokumentáció.....	22
Webshopunk célja	22
Használati igények	22
A webshop környezetünk felállítása	22
Állapotok	25
Kezdő oldal	26
Regisztráció és bejelentkezés	27
Egy könyv saját kilistázása	27
Admin oldal	28
Összefoglalás.....	28
További célkitűzéseink.....	28
Irodalom jegyzék.....	29

BEVEZETÉS

TÉMAVÁLASZTÁS INDOKLÁSA

A témaválasztásunk célja egy **online könyvesbolt** megtervezése és megvalósítása volt. Jelenleg Magyarországon kevés korszerű könyves webshop működik, a meglévő áruházak többsége elavult megjelenésű, és funkcióik sem felelnek meg a mai vásárlói igényeknek. A mi BookShopunk könyveket kínál, amelyeket a felhasználók kényelmesen, online vásárolhatnak meg. A sablont a felhasználói élményre fókuszálva alakítottuk ki, modern, letisztult és könnyen

kezelhető designt alkalmazva. Így webshopunk mind funkcionalitásában, mind megjelenésében megfelel a mai elvárásoknak. A felhasználóbarát felület lehetővé teszi a gyors és egyszerű vásárlást. Emellett regisztrált vásárlóink számára egy bónusz játékot is kínálunk, amelyben naponta egyszer játszhatnak, és értékes kedvezményeket, illetve kuponokat nyerhetnek.

CÉLKITŰZÉS

Nagyon fontos alappillérként kezeltük a **SEO** (Search Engine Optimization) alapelveinek alkalmazását, hogy a webáruház éles indulásakor a Google megfelelő helyezést biztosítson, ezáltal növelve az organikus keresések számát és a webshop értékét. Mivel Localhost környezetben dolgoztunk, a SEO-t egyelőre csak alap szinten tudtuk megvalósítani, de az élesítéskor ez könnyedén beüzemelhető lesz. Célunk volt egy széles termékkínálat létrehozása, így több mint **300** könyvet regisztráltunk a webshopba, amely a jövőben egyszerűen bővíthető. Külön **admin** felületet alakítottunk ki, ahol könyveket és kategóriákat tudunk hozzáadni, módosítani, törölni. Fontos szempont volt, hogy a webshop bejelentkezés nélkül is használható legyen, bizonyos korlátozásokkal, hogy a regisztráció előnyei is érvényesüljenek. Oldalunk nem gyűjt privát adatokat (például IP-címeket), így biztonságos felhasználói élményt biztosít. Vásárlás regisztráció nélkül is lehetséges, azonban kuponok így nem szerezhetők vagy válthatók be.

FEJLESZTŐI DOKUMENTÁCIÓ

FEJLESZTŐI KÖRNYEZET

Webshopunk a projekthez méltó módon webes felületen érhető el. Teljes **Frontend- és Backend-rendszert** építettünk ki, hogy megkönnyítsük a fejlesztést, hiszen a különválasztott rétegek kezelése átláthatóbbá és gyorsabbá teszi a munkát. Emellett létrehoztunk egy asztali alkalmazást is, amely egy játék, és közvetlenül a weboldalról indítható. A fejlesztéshez a **Visual Studio Code** lehetőségeit, valamint különböző extension-eket használtunk, továbbá **XAMPP** környezetben is végeztünk módosításokat, hogy az útvonalkezelést egyszerűsítsük. Ezzel a fejlesztői munka hatékonyságát és pontosságát növeltük. Bár a közös munka miatt külön fejlesztői környezeteket is kialakíthattunk volna, mi végig **Git** verziókezelőt használtunk, így bármilyen gépen, bármikor tudtunk dolgozni a projekten.

FELHASZNÁLT PROGRAMOK/ESZKÖZÖK

A teljes Frontend- és Backend-fejlesztést a **Visual Studio Code**-ban végeztük. Fontos szempont volt, hogy az általunk használt összes programnyelvet kezelje, és olyan beépített funkciókat biztosítson, mint a kódkiegészítés, Live Server, driverek, beépített Git-támogatás és a strukturált fejlesztést segítő bővítmények. A projekt körülbelül 70%-a PHP-alapú, ami egy webshop esetében teljesen megszokott, hiszen folyamatosan adatbázis-műveleteket végzünk: adatokat kérünk le és mentünk. A két réteg közötti kommunikációt több segédréteg támogatja, például a regisztráció során a telefonszámot különböző formátumokból (pl. +3630..., 0630...) egységesen, a magyar szabvány szerint alakítjuk át a rendszerben.

Az asztali alkalmazásunk, amely egy játék a regisztrált felhasználóknak van kialakítva **Visual Studio 2022**-ben készült C# nyelven. Fontos kitétel volt, hogy indítható legyen a weboldalról, amelyet meg is oldottunk, illetve az adatbázissal való teljes kommunikációt is képítettük, tehát bele írni, és kiolvasni is tud adatokat, amely lehetővé teszi a dinamikus adatkezelést és nyeremények könyvelését, a felhasználó játék elindításának idejét is lementjük, amellyel kritérium vizsgálatokat végzünk a továbbiakban.

XAMPP biztosította nekünk az adatbázis elindítását, vezérlését, és Apache szerver indítását. Az általunk használt verziók 8.0.30-8.2.12 problémamentesen működött minden, azonban eltérő verziók, amelyekben elemi változások vannak bizonyos modulokban komplikációk léphetnek fel. Fontos, hogy a vizsga remekünkbe eltérő útvonalakat használtunk, illetve mappák neveit máshogy deklaráltuk, mint amelyen néven hivatkozunk rájuk, ez azért történt, hogy a strukturalitás megmaradjon, azonban a fejlesztés mégis egyszerűen, közérthetően menjen, egyszerűsített útvonalak pedig az ellenőrzést és a tesztelést is segítették.

A **GitHub**-ot használtuk, ugyanis globálisan szinte bárhol elérhető, illetve a repository-k klónozható bárki számára. Ahogy említettem korábban a Visual Studio Code és a Visual Studio 2022-es programokban is bele van építve, és egy egyszerű bejelentkezéssel hozzáférünk a fiókunkhoz. Ennél robosztusabb megoldást nem is találhattunk volna, teljesen kiszolgálta az igényeinket a fejlesztés során.

A weboldal logikája összefoglalva Front és Backend struktúra, amely dinamikusan képi le az adatokat az adatbázisból, a kommunikáció asszinkron módon történik, hogy a dinamikusság megmaradjon, ne kelljen az oldalt újratölteni. Ez a **AJAX** (*Asynchronous JavaScript and XML*) mechanikájával történik, amely zökkenőmentesen, gyorsan tölti be az adatokat, az oldal újragenerálása nélkül. A rugalmasságot és a teljes dinamikusságot Javascript-el érjük el, ugyanis így sok időt spórolhatunk meg, illetve a kód ismétlést is elkerüljük. A JS kód részekben „Crazy JS” kódnak felel meg, amelyre nincsen pontos szakkifejezés, de valami ilyesmi lenne:

„Advanced JavaScript Hacks vagy Creative Coding”. Ezt a decorate.js-be pontosan látni lehet, ahol egy változékony háttért generálunk le, amelynek alapja ebben az esetben matematika. Mindenek felett azonban az volt a lényeg, hogy érjük el a teljes kommunikációt, tehát minden-mindennel kommunikáljon, ez természetesen sikerült is.

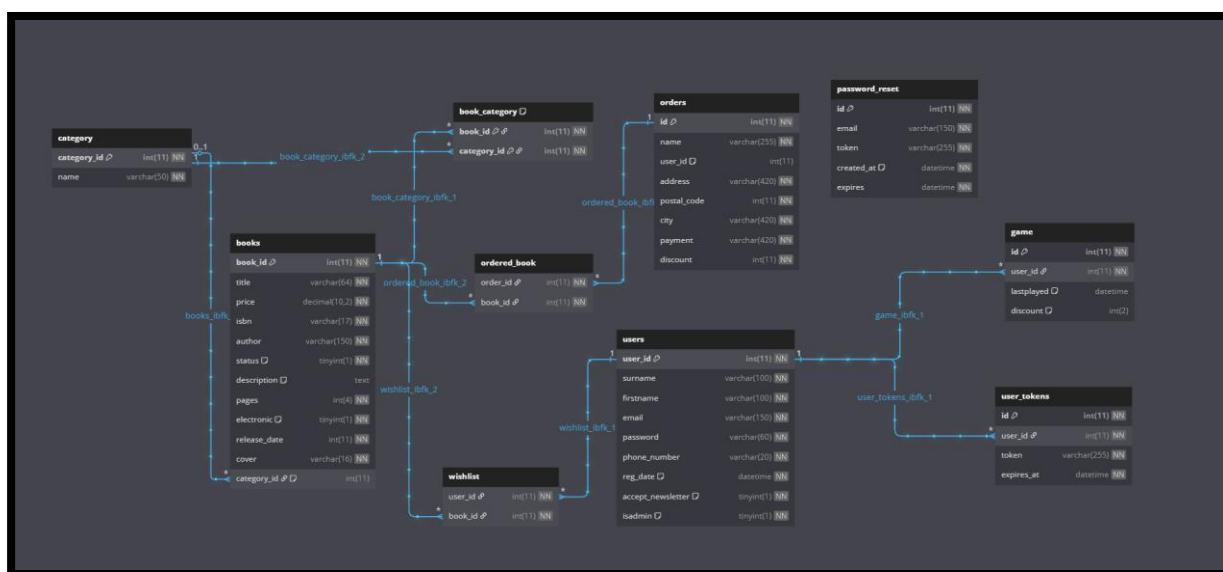
ADATBÁZIS FELÉPÍTÉSE, MŰKÖDÉS

ADATBÁZIS ISMERTETÉSE

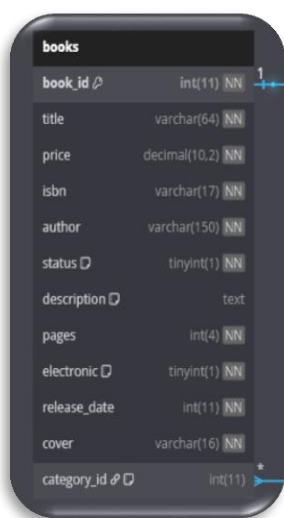
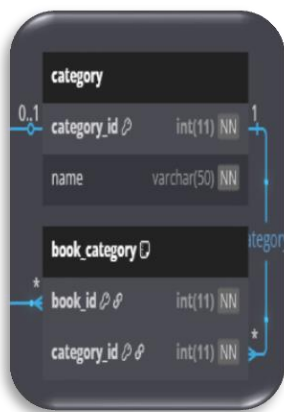
A projektünk megtervezésekor a modellalapú gondolkodás mellett döntöttünk, és a **Database First** fejlesztési modellt választottuk. Ennek a módszernek a lényege, hogy először az adatbázis szerkezetét alakítjuk ki: megtervezzük a táblákat, definiáljuk a mezőket, és meghatározzuk az egyes entitások közötti kapcsolatokat. A programkód fejlesztését ezután az elkészült adatbázis-struktúrához igazítottuk.

A fejlesztés során **MySQL**-t használtunk, amely kiválóan alkalmas gyors, megbízható és jól skálázható adatbázisok létrehozására. A Database First megközelítés nemcsak MySQL esetén, hanem más adatbázis-kezelő rendszerek (például PostgreSQL vagy Microsoft SQL Server) mellett is alkalmazható. Webáruház típusú rendszereknél ez a modell különösen hasznos, hiszen a jól strukturált adatbázis már a kezdetektől biztosítja a stabil működés alapjait. Így a fejlesztési idő jelentősen lerövidül, a hibalehetőségek csökkennek, és az esetleges befektetett költségek is minimalizálhatóak.

ADATBÁZIS FELÉPÍTÉSE



BOOKS ÉS KATEGÓRIÁK TÁBLA



A **books** táblában találhatóak a könyvek alapvető információi. A **book_id** egyedi kulcs, melyen keresztül csatlakozik és kapcsolatban áll a további táblákkal. A **Cover** tartalmazza a képet, amely át van konvertálva, egy 16 hosszú varcharra, így körülbelül 3000 könyvet is fel tudunk tölteni. A képek fellettek töltve file-ként és ezek nevei lettek eltárolva az adatbázisban. Próbálkoztunk blob-al, azonban azzal nem tudtuk kellően jól megjeleníteni a képeket, így az első verzióval dolgoztunk. Feltöltéskor az admin választhat, hogy a könyv elérhető-e vagy még nem (*olyan eset, hogy elfogyott, vagy nincs raktáron, előre rendelhető stb.*), illetve, hogy elektronikus-e vagy nem. Elektronikus könyveknek nincs isbn-je, azonban a mi weboldalunk esetében generálunk nekik, ugyanis minden könyv 'unique' azonosítóval rendelkezik. A **category_id**-n keresztül csatlakozik a kategóriához, amely a nagy kategória megnevezése, ilyen pl. Történelmi, Fantasy, Képregény stb. azonban ez még csatlakozik a **book_category**-hoz, amely egy párosító tábla. Itt történik a kategóriához való könyv hozzárendelés, egy könyv több kategóriához is kapcsolódhat, amely fontos, hiszen sok könyv sorolható más-más kategóriába. A **category** tábla a kategória neveket tárolja el, ezeket említettem feljebb. Fontos a szeparáltság és strukturáltság, ezért amit lehet külön-külön tárolunk el táblákban.

USERS TÁBLA



A **users** tábla tartalmazza a felhasználói adatokat, amelyeket regisztrációnál megad. A jelszó természetesen hash-elve van, binhex függvény segítségével, amely röviden annyit takar, hogy a jelszó 60 karaktert hosszú és bycrypted. A telefonszám megadásánál minden esetben átformázzuk, hogy csakis magyar +36 (30) vagy +36 (20) formában jelenjenek meg, ugyanis regisztrációnál megadható ugyan ilyen formában, vagy esetleg 3630 vagy 0630 stb. Lényegében egy volt a cél, az egységesítés, amelyet a fentebb említett konvertálással oldottunk meg. A

felhasználó bejelentkezése után, hogy pl. oldalváltáskor vagy tranzakciónál, oldalon való interaktálásnál tokennel azonosítsuk. Mi nem a **JWT** tokenet használtuk, hanem generálunk egy 16 számjegyből álló **hexademical secure token**, amely számokból (1-9) és hexaszámokból (10-A, 11-B, 11-C) stb. Fontos megemlíteni, hogy `user_id` egy auto incrementes mező, mely növeli az értéket minden egyes új regisztrált felhasználó után. Ez nagyon hasznos, ugyanis a felhasználókat, tranzakciónál, vagy beazonosító adatbázis műveleteknél könnyen, egyszerűen tudjuk lekérdezni, létrehozni. A regisztráció amikor megtörténik a pontos időt eltároljuk, így tudunk regisztrációs statisztikát is készíteni, amely **SEO** weboldalaknál nagyon jól jön, ugyanis látjuk melyek azok az időszakok amikor a webshopunk aktív. Ha a hírlevelet valaki elfogadja, ezzel jelezi számunkra, hogy nyitott az emailek fogadására. Ezt a funkciót nem építettük teljes mértékben ki, hiszen az „email magneting” egy külön szakterületként is felfogható, és nagyon időigényes lett volna, különböző, aktuális ajánlatokról üzeneteket küldeni, azonban a jövőre való tekintettel mégis elkészítettük ennek az alapját. Az utolsó mező az „isadmin” amely egy boolean típusú mező, és ennek az értéke mondja meg, hogy ki rendelkezik admin joggal. Ha az értéke 1 akkor admin, ha 0 akkor felhasználó. Ha admin jogot akarunk valakire ruházni, akkor az adatbázisban kell módosítanunk ennek a mező értékét. Nem akartuk, hogy ha valaki admin a weboldalon, akkor közvetlenül bárkire tudjon ruházni jogokat, ezért jogot csak az adatbázist vezérlő személy tudja. A webshopunknál szinte minden adatot dinamikusan az adatbázisból töltünk be, ezért a legnagyobb jogkör az adatbázis tulajdonosa, illetve kezelője.

USER_TOKENS TÁBLA



A **user_tokens** tábla a tokeneket kezeli, amikor valaki belép a weboldalra azonosítjuk. Nem névvel vagy emailel, hanem tokennel, ugyanis ez a legbiztonságosabb módszer, és valószínűleg a legpopulárisabb is. Bejelentkezésnél eltároljuk a token, amely egy bizonyos ideig érvényes, majd ha lejárt a token élettartama, akkor törlésre is kerül az adatbázisban. A token **bin2hex** függvénnyel generálódik le véletlenszerűen, és 64 karakter hosszú.

GAME TÁBLA

A **game** táblában a játék adatai, hogy a felhasználó pontosan mikor játszott, és nyert-e leárazást. Minden felhasználó napi egyszer játszhat, és minden játszása log-olva lesz az adatbázisban, hogy

game	
id	int(11) NN
user_id	int(11) NN
lastplayed	datetime
discount	int(2)

pontosán lássuk, mikor játszott és mennyit nyert. Ez azért is jó, mert ha valami bug adódik véletlen, vagy egy hibát akarnak kihasználni esetlegesen, akkor látni tudjuk, hogy mikor ki játszott, így mindenki számára tisztességes játékot tudunk biztosítani. Ha valaki soha nem játszott még, csak regisztrált akkor egy alapértelmezett dátum kerül megadásra amely a „2000-01-01” ez azért is fontos, mert biztosítja, hogy a felhasználó nem játszott még, tehát 24 órán biztosan kívül esik. A „discount”/kupon amelyet szerezhetsz, lehet 10% és 20%. Ezt pontosan a játékba írt algoritmus határozza meg.

WISHLIST TÁBLA

wishlist	
user_id	int(11) NN
book_id	int(11) NN

A **wishlist** táblába mentődik minden egyes kívánságlistához adásnál. Ez azért fontos, mert ha kijelentkezünk, és mondjuk másnap bejelentkezik a felhasználó, akár más eszközről, akkor megtalálja a kívánságlistájában az oda hozzáadott könyvet. Az adatbázisban csak a user_id és a book_id van eltárolva, azonban a kívánságlistában betölti a könyv teljes adatait pl. cím, kép, így egy kényelmes módszer minden felhasználó számára, aki csak nézelődik és ha megtetszik neki egy könyv hozzáadja, és le is tárolódik. Azonban ha nem vagyunk bejelentkezve, akkor számunkra a kosár mellett nem lesz elérhető a kívánságlista ikonja. Ez azért is fontos mert user_id-t mentünk és legyen előnye annak, ha valaki beregisztált és felhasználóvá vált.

ORDERS TÁBLA

orders		
1	id	int(11) NN
	name	varchar(255) NN
2	user_id	int(11)
	address	varchar(420) NN
	postal_code	int(11) NN
	city	varchar(420) NN
	payment	varchar(420) NN
	discount	int(11) NN

Az **orders** táblában megtaláljuk a rendeléseket. A rendelések lehetnek bejelentkezett felhasználóhoz kötve, az ilyen tagok tudnak kupont érvényesíteni, amelyet a rendelés leadása után már a pontos, kiszámolt árral kerül be az adatbázisba. Számlázási névhez mindig kell nevet adni, ugyanis vásárolhatunk és ha nem magunknak akarjuk, hanem egy családtagunknak, így tudjuk neki is számlázni, amely előny ilyen helyzetekben. Ha regisztráció nélkül vásárlunk, akkor egyértelműen fontos, hogy tudjuk kinek kell kiállítani a számlát, azonban fontos ilyen esetekben nem tudunk kupont érvényesíteni.

ORDERED_BOOK TÁBLA

ordered_book		
	order_id	int(11) NN *
*	book_id	int(11) NN

Az **ordered_book** táblában az egyes felhasználókhoz megrendelt könyvek vannak beírva, egy felhasználóhoz nyilván több rendelés, és több könyv is társulhat.

PASSWORD_RESET TÁBLA

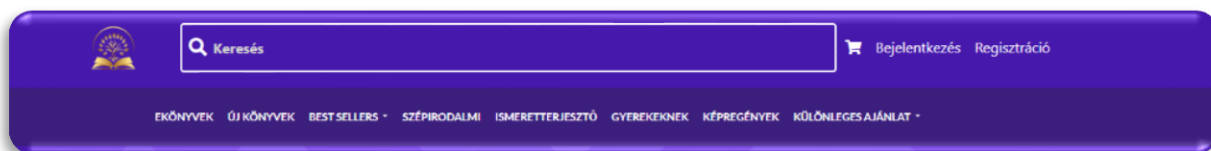
password_reset		
	id	int(11) NN
	email	varchar(150) NN
	token	varchar(255) NN
	created_at	datetime NN
	expires	datetime NN

A **password_reset** tábla eltárolásra kerül a felhasználó email címe, és generálva lesz egy token, amely 1 óráig fog élni. Ha a felhasználó új jelszót szeretne, mivel elfelejtette, vagy csak változtatni akar, akkor meg kell adja az email címet, majd erre az email címre szinte azonnal kap egy emailt, amely egy linket tartalmaz. Ha a linkre kattint, akkor az **URL**-ben látni is lehet a token. A token 1 óráig érvényes, és amint ezt a token-t legeneráltuk, el is tároljuk az adatbázisban, de azonban ha a token 1 óra után lejár, akkor törlésre kerül, és érvénytelen lesz, tehát többször nem lehet majd felhasználni. Ez fontos a védelem miatt is, és hogy az adatbázisban ne tároljunk el már felesleges adatokat. Amint a linken keresztül új jelszót állítbe, akkor a users táblában update-olásra kerül a korábbi jelszó, és az új jelszó lesz természetesen érvényes.

FRONTEND

NAVIGÁCIÓ

A navigációt két részre osztottuk, amely annyit takar, hogy van egy **felső (top)**, és van egy **main**. A felső navigációban szerepel egy keresősáv amely adott könyvre, szerzőre való keresést segíti elő. Megtalálhatjuk a keresőmezőtől balra a logo-t, amely általunk létrehozott BookShop logója. Találhatunk a jobb szélén egy bejelentkezés és egy regisztrációs gombot, illetve azok mellett balra egy kívánságlista és kosár ikont.



A main navigáció tartalmazza az egyes kategóriákat, amelyeket megjelöltünk, vagy a Bestseller könyveket illetve különleges ajánlatokat. Itt tudunk navigálni különböző oldalakra, illetve esetleg böngészni a dropdown menü egyes pontjai között.

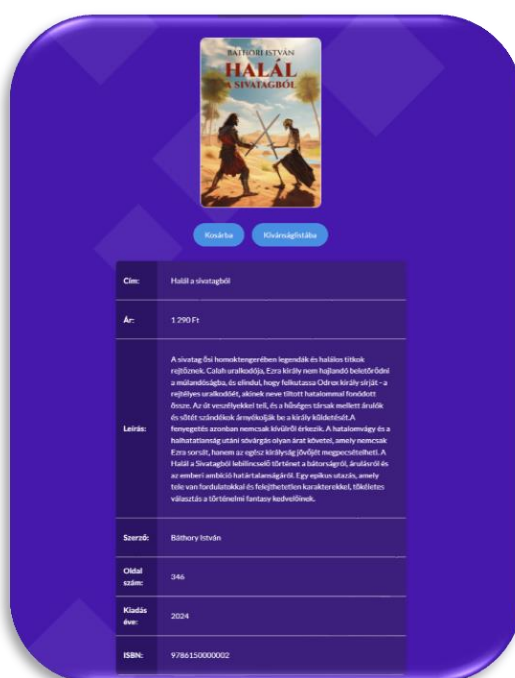
KEZDŐOLDAL FELÉPÍTÉSE

Minden oldal betöltéskor az **index.php** oldalra leszünk navigálva, amely a webshop kezdőfelülete, landing page-e. Itt láthatjuk a könyveket, amelyek kártyákban töltődnek be, betöltésre kerül a könyv képe, címe, illetve az ára. Egy sorba egy kategória van, amelyet léptetni tudunk nagyobb kijelzőkön, kisebb kijelzőkön csúszka szerepel. Ha egy könyvre visszük a kurzort, akkor a vásárlás gomb jelenik meg, alulról tolódik fel, mellyel egy adott könyvet a kosárhoz tudunk hozzáadni. Az oldal háttere nem egy kép, hanem egy **Advanced Javascript**-ben megírt kód, amelyben négyzeteket generálunk le, és ezeket forgatgatjuk különböző irányba, ezzel effektet generálunk, amely sokat hozzátesz a kinézethez.

TOVÁBBI OLDALAK

Ha tovább navigálunk, akár az alsó navigációban, vagy egyed adott könyvre keresünk, vagy a képére kattintunk, akkor további oldalakat érhetünk el. Ilyenek a kategóriák oldalai, amelyben a kategóriához tartozó összes könyvet megjelenítsük. Ugyan úgy kártyás megjelenítést használunk, és **grid** rendszert, amely lehetővé teszi, hogy 1 sorban pontosan 4 könyvet tudjunk megjeleníteni, és a sorok addig generálódnak, amég vannak további könyvek betöltésre. A könyvek felett megjelenik az adott kategória címe, és felette egy gomb, amely a főoldalra navigál vissza.

Ha egy könyv borító képére kattintunk rá, akkor a könyv betöltésre kerül egy következő oldalon, ahol láthatjuk, hogy milyen információk tartoznak hozzá.



A könyv képe alatt továbbá a kívánságlistához és akár a kosárba is hozzá tudjuk adni, amely a kényelmes vásárlást és böngészést eredményezi.

FOOTER

Az oldal footerjében található promóciós szöveg, illetve vásárláshoz tartozó információk. Jelenleg nem rendelkezünk még Facebook oldallal, azonban az oldal megnyitásakor egyszerűen csak a **linket** kell hozzáfűzni, és már működik is. Szinte minden webshopnak van Facebook oldala, ezért gondoltuk mi is, hogy esetlegesen ha megnyit a webshop akkor mi is csinálunk egy közösségi oldalt. A legeslegalsó szekcióban láthatjuk a **Cookie** tájékoztatót, amely



tájékoztatást ad, hogy az oldal milyen információkat tárol el cookie-ban, illetve az oldal böngészésekor ezeket alapvetően elfogadjuk. **Adatvédelem** alsó menüpont a felhasználó adatvédelmezési módszereiről beszél, illetve, hogy nem gyűjtünk „illetéktelen” adatokat. Az **Általános Szerződési Feltételek (ÁSZF)** amely tartalmazza pl. a vásárláskor a vásárló jogait, vagy a jóállást, illetve további információkat szolgál a vétel, eladás és termékekről.

AJAX

A termékek megjelenítése dinamikusan történik, amely azt jelenti, hogy a backend végzi a lekérzést, az útvonalak által pedig egyszerű kommunikációt biztosít a frontend és backend közvetítő rétegének, az ajaxnak. A **JSON** formában levő adatokat, az ajax eléri és egy fetch, már dolgozni tud is vele. `fetch('info_backend.php?bookId=${bookId}`)` egy általunk használt fetch, amely bookID-ját kapja meg, és ehhez csak a hozzá tartozó információkat csatolunk `const coverPath = '/bookshop/web/database/covers/' + v.cover + '.png';` mondjuk itt a képet például. Ajax azt segíti elő, hogy aszinkron módon küld és fogad adatot, miközben az oldal nem frissül újra, tehát a dinamikusságot megőrzi. Erre egy példa, ha pl. egy könyvet beleteszek a kosárba, akkor nem frissül újra az egész oldal, mégis bele kerül a könyv a kosárba, ezzel azt is elértük, hogy az oldal gyors, és folyamatos interakciókat segít elő. Továbbá a szerveret sem terheli, mivel nem kell mindig minden **HTML** elemet újragenerálni a backend által, illetve a mi webshopunkba csak nyers adatokat küld vissza pl. JSON-t. Jobb modularitást is elérhetővé teszi, amely azt eredményezi, hogy a backend és a frontend logikailag különválik, és a kommunikációs réteg az **AJAX** lesz. Pl. a frontend lekérdezi a könyveket, egy végpontról, és saját maga jeleníti meg. Fontos megemlíteni, hogy nem mindenhol használunk AJAX-ot, ugyanis valahol már elég egy **REST API** amelyből le tudunk képezni könyveket, azonban vannak bonyolultabb kérések, ahol érdemes volt használni, mint kommunikációs réteg.

BACKEND

BACKEND FELÉPÍTÉSE

Webshopunk nagyrészt backendre épült a dinamikus betöltések, illetve a termékek letárolása mind adatbázisból történik, emelett elenyésző a frontend kód mennyisége. Törekedtünk arra, hogy átláthatóan tudjunk kommunikálni backend és frontend között, amelyre több módszert is használtunk. **Három állapotot** állítottunk fel, hogy a webshopot tudjuk bejelentkezés nélkül használni, itt korlátozott funkciók vannak, de tud rendelést leadni, amely a backenden keresztül megy, van a bejelentkezett állapot, vagy jobban mondva tag, és utolsó sorban admin, aki rendelkezik saját admin felülettel. Oldalakat is a backendnek megfelelően építettük ki, van egy

main (**index.php**), amelyet nevezhetünk „*landing page*”-nek, hiszen az oldal megnyitásakor a felhasználók ide érkeznek. Különböző kategóriák vannak bővebben, amely lehetővé teszi a kategóriás böngészést, ezeket al-oldalaknak nevezzük. Vannak „*Best Seller*” könyveink, amelyek a legtöbb eladott könyvet, vagy a legfelkapottabbakat jelzik, ezekre kattintva teljes adatot kapunk meg. Ha más könyvről akarunk teljes adatot megkapni akkor a borítóképre kell kattintanunk, amely tovább vezényel minket a books.php-ra ahova leképezzük az adott könyvet **ID** alapján.

REGISZTRÁCIÓ

Regisztrálni a regisztrációs oldalon tudunk, ahol adatokat kell megadnunk, pl. email, telefonszám, jelszó stb. A regisztráció egy `session_start()`-al kezdődik, és kapcsolódunk az adatbázishoz. **UserRegistration** osztályon belül külön függvények segítségével validáljuk az adatokat, illetve a telefonszámot megformázzuk a könnyebb megértés és szinkronizálás végett. *(Jelenleg csak Magyar telefonszámok elfogadása engedélyezett)*. Megvizsgáljuk, hogy adott adatokat ne lehessen duplikálni, ilyen az email cím és telefonszám. Ha valamilyen hibát kapunk akkor kiírjuk a kijelzőre, így a felhasználó pontosan tudja milyen adat foglalt, vagy miben ejtett hibát. A jelszó szigorúan vett 8 karakterből kell álljon, tartalmaznia kell betűt, számot és speciális karaktert. A telefonszám és jelszó validálását a **preg_match** függvénnyel végezzük, ugyanis ebben pontosan meg lehet határozni milyen karaktereket szeretnénk szerepeltetni, vagy hogy miket nem. Ezután ezeket az adatokat beszúrjuk stmt (*statement*) segítségével, amelyet előbb előkészítünk a **prepare** függvénnyel, ennek az előnye az, hogy előre előkészítjük az adat beszúrását és biztonságosan futtatjuk le, ezáltal megelőzhetjük az SQL injection támadásokat. A **bind_param** adja hozzá a paramétereket biztonságosan a statement-hez, és `execute()` hajtja végre az utasítást. Ha a felhasználó olyan adatokat adott meg, amelyek megbuknak a validáció során, akkor a beszúrás nem lesz végrehajtva, rollbackel vissza lesz húzva minden általa elkezdett utasítás. A véglegesítés egy **commit()** utasítással ér véget, amely egyben a hitelesítés végét is jelenti, ilyenkor kapunk egy üzenetet, amelyet session-be tárolunk el, hiszen oldalt váltunk, és ez egy egyszerű módja. A sessionbe, egyszerű **success** üzenetet adunk át. A hibákat minden esetben throwoljuk **new exception**-nal, és megadjuk a hibát, vagy ha helytelen adatot ad meg, akkor azt jelezzük üzenettel. Amikor regisztrál valaki a validáció után az adatok listába kerülnek és ha hiba van akkor ebből a listából kerül ki a hiba, az error-kat is sessionből kiolvassuk az új location-ön, majd exittel bezárul, a regisztrációs folyamat.

LOGIN

A **login.php** felelős a bejelentkezés hitelesítéséről, amely egy **Auth** (*Authentication*) osztályba van definiálva. A login function megkeresi a felhasználót email alapján ezt a **getUserByEmail** függvény végzi, ha a felhasználó megvan, akkor a jelszavát ellenőrizzük, melyet a **password_verify** függvény teszi lehetővé. Itt megnézzük az általa megadott jelszót, természetesen bejelentkezésnél, majd azt amelyet alpból megadott amikor regisztrált, és ha egyezik akkor bejelentkeztetjük. Ekkor vissza navigáljuk az **index.php**-oldalra és a regisztráció/bejelentkezés helyett egy Üdv „név” lesz látható, ezzel is jelezve, hogy sikeresen bejelentkezett, és új státusza **isLoggedIn** true. Hiba esetén sessionból kiiratjuk a hibát, majd újra megpróbálhat bejelentkezni, vagy regisztrálni.

Ha a felhasználó elfelejtette a jelszavát, akkor megadhat egy emailt, és ha megadta, egy linket küldünk rá, amely egy külön oldalra navigálja, ahol új jelszót állíthat be. Ekkor kap egy különleges token, amely 1 óráig van életben, szóval 1 link 1 óráig elérhető.

A bejelentkezés után a felhasználó tokenet kap, amely meghatározott ideig érvényes. Ennek lejártá után automatikusan kijeltkeztetjük, és újból be kell jelentkeznie, hogy új tokenet kaphasson, ezt a **getOrCreateToken** függvénnyel tudjuk megkapni, vagy létrehozni, lekérdezni a **getExistingToken** függvénnyel, amelynek paraméterben meg kell adjuk a userId-t. Új token létrehozni **bin2hex** függvénnyel tudunk, amely 32 bitből fog állni, és a lejárási ideje 30 nap elteltével jár le. Szóval ha bejelentkezik valaki, és egy idő után bezárja az oldalt, és majd megint megnyitja be lesz jelentkezve, amíg nem törlődik a tokene, vagy nem jelentkezik ki magától. Ez egy kényelmi funkció, amely elég biztonságos, ugyanis **hexadecimal secure token** az egyik legbiztonságosabb token fajta.

LOGOUT

A kijelentkezés a logout.php-ba van definiálva, a **SessionManager** osztályba, a **destroySession** függvénybe. **Session** próbálunk indítani, illetve, ha van eltárolni egy tömbbe, majd ezeket megsemmisítjük **session_destroy()** függvény segítségével. Kitérlünk minden cookie-t a **deleteCookie** függvénnyel, ez fontos, hiszen itt töröljük a meglévő tokenet is, és a beállított nevet, amikor pl. bejelentkezőnk, így nem tárolunk már fölösleges adatokat benne. A **redirect** függvénybe definiált url-re vezényeljük át a felhasználót, ilyenkor az url-be láthatjuk a *?sessionEnded=true*-t amely egyben beállítja az **isLoggedIn** változót false-ra, szóval innentől nincs bejelentkezve a felhasználó, tevékenységi köre korlátozva lett.

CHECK TOKEN

A token ellenőrzés nagyon fontos, hogy esetleges bugokat, illetve azonosításokat oldal váltásakor, frissítéskor kiszűrjünk. A **TokenAuthenticator** osztályban történik a felhasználó **userId**-hoz rendelt token ellenőrzése, egy **validateToken** függvénnyel. Beállítunk egy usersession-t, amennyiben rendelkezik tokennel a felhasználó, azonban szinte mindig csak a tokent ellenőrizzük, hiszen abba van minden információ, amely a beazonosításhoz kell számunkra. Ezeket az **\$authenticator** változóban mindig eltároljuk és bármikor tudunk felhasználót azonosítani, vagy lekérdezni.


ORDER VALIDATION ÉS PROCESS

Rendelés leadásakor form-ot kell ugyan úgy kitöltenünk, és az itt levő adatokat validáljuk. Nehéz a pontos validáció, azonban karakter hosszokkal tudjuk vizsgálni, hogy adott-e meg várost, illetve nevet, címet, vagy irányítószámot. A payment-et egy lenyíló listával tudjuk kiválasztani, jelenleg csak utánvét érhető el, ezért csak ezt tudjuk kiválasztani. Ha rendelkezünk kuponnal, akkor tudunk kupont érvényesíteni, de csak akkor rendelkezhetünk kuponnal, ha be vagyunk jelentkezve és esetlegesen nyertünk a naponta 1x játszható játékunkban. Validálás után az **order_process.php**-ba történnek meg az adatok táblába rendelése, tehát insertálása a stmt-s módszerrel, így itt is védve vagyunk az **SQL injection**-től. A kosárhoz hozzáadott termékeket decode-oljuk így megkapjuk az itemeket amelyeket foreachel végig járunk és a könyvek Id-ből megtudjuk a pontos árat. Ha van kuponunk akkor érvényesítés után az összegünkben levonódik, és a beérkezésnél már a pontos árat látjuk, ezt nyilván kilistázva az adminok látják.

ADMIN OLDAL

Az admin oldal teljes mértékben a backend kezelésére épül. Könyveket tudunk feltölteni, egy form segítségével megadjuk a könyv adatait, és ezek beillesztésre kerülnek. Fontos, egy könyvhöz több kategóriát is tudunk rendelni. A könyv **ISBN** száma **UNIQUE** és a könyv id-je **PRIMARY KEY** lesz. A könyvek képeit, amelyek file-ok, neveket tárolunk el, ehhez a **bin2hex 8 bájtos** nevet generálunk, amely 16 karakter lesz nyilvánvalóan. Ez azért kell, mivel a képet is feltöltjük, azonban az azonosítása ezáltal sokkal egyszerűbb lesz, hiszen nem valami random file-t kell keresni, hanem egy file-hoz rendelt „*karaktersorozat*”.

A kategória feltöltés szinte teljesen ugyan ez, azonban itt csak a kategória nevét kell megadnunk, és ezáltal nyitunk egy teljesen új kategóriát, amelyhez szinte azonnal tudunk is könyveket rendelni.

Könyv törléséhez egy **ISBN**-t kell megadnunk, amely által megkeressük a **bookId**-ját és a cover-jét, és töröljük, ezáltal a többi adat is törlésre fog kerülni, azonban a képet külön is törölnünk kell, ugyanis az teljesen máshogy van ugye letárolva, lásd. a fentebbi könyvfeltöltés leírását. Ha könyvet törölünk akkor a könyvet a kategóriából is ki kell töröljük, ugyanis külön táblákhoz is hozzá lett adva, és ott se szerepeljen ha már eltávolítottuk.

Kategória törlése név alapján történik, ha a kategóriában nincsenek könyvek, akkor csak a delete-et végre hajtjuk, azonban ha szerepelnek benne könyvek előbb a könyveket kell kitöröljük, aztán majd a kategóriát. Ha megszűnik egy kategória, akkor a könyveket belőle ki kell törölnünk, de más kategóriákban ha szerepel egy könyv akkor onnan értelemszerűen nem. Minden hozzáadás és törlés ad visszajelzést, hogy sikeres, vagy sikertelen a művelet.

Az **orders.php** fájlban az adminoknak kilistázásra kerül minden megrendelés, már pontosan kiszámított árral, tehát a kupon használat már érvényesítve van ilyenkor. Ez azért fontos mert a könyveket mi adjuk fel, és ezeket a rendeléseket számon tudjuk itt tartani, amely elősegítheti a további könyvelést.

KÍVÁNSÁGLISTA

A kívánságlistához adáshoz készítettünk egy **wishlist managert**, amely felelős a könyvek leképzésért a kívánságlistába. A könyveket bookId alapján a `/bookshop/web/api/wishlist.php` útvonalra adjuk hozzá, és ezután frissítjük a whistlistet az adott felhasználónál. Ugye ez a funkció csak bejelentkezett tagoknak érhető el, és a frissítés „update” azért is kell, mert eltároljuk ezeket a könyveket az adatbázisba, minden felhasználónak külön. Persze tudunk innen is törölni, ekkor a bookId alapján kerül törlésre a kívánságlistából a könyv. Használunk egy **findIndex** függvényt, amely a keresett könyv book_id mezőjét hasonlítja össze a bookId eredeti mezőjével, ugye külön-külön táblában vannak ezért kell fetchelni is, hogy az útvonalakkal tudjunk egyszerűbben dolgozni. A **whislits.js**-nek a funkciója az, hogy a Modal-al közvetlenül működjön, tehát a felhasználó lássa az interakciókat, ha hozzáad egy könyvet jelenjen meg ott, lássa az eddigi hozzáadott könyveket stb., illetve ide vannak meghívva a törlés, hozzáadás és frissítés függvények, amelyek szerves részét képezik az interakciók zavartalan működésének. Az **updateWishListModal**, ahogy a neve is jelzi a Modalt frissíti, minden esetben törlés, hozzáadásnál, dinamikusan, tehát nem kell újratöltenünk az oldalt, hogy lássuk a változást. Ehhez a wishlistmanagernek foreach-el meg kell keresse azt a könyvet a fetch-elt útvonalon, ezután feltételt vizsgálunk és megjelenítjük **innerHTML** segítségével.

RemoveFromWishlist függvény pedig könyv (bookId) alapján töröl, illetve a hozzáadás addEventListenerrel figyelve van, és ha click true értéket kap hozzáadja a wishlisthez.

CART

A kosárhoz adás ugyan olyan alapon történik mint a kívánságlista, azonban az itt hozzáadott termékeket nem tároljuk el, ugyanis ez egy elég kellemetlen funkció lenne, és sok felhasználónak biztosan zavaró, hogy a kosárba adott termékek mentődnek és ha bejelentkezünk pl. 15 nap múlva akkor ugyan úgy benne lennének a kosárban. Itt csak a modal-hoz rendeljük hozzá a könyvet, **addEventListener**rel figyeljük a clicket, és ennek ütemében nyitunk eventet. A **toggleCheckButton** beszédes elnevezés a kifizetéshez visz minket, ahol adatok megadása után megrendelni tudjuk az adott könyvet.

A JÁTÉK

GRAFIKAI RÉSZ

A játék betöltésekor fent az ablak balsarkába megjelenik egy **Fájl** menü amit ha lenyitnak akkor megjelenik egy menü opció, amivel kilehet lépni a játékból ez alatt lesz egy utasítás, ami megkéri a játékost, hogy válaszson 3 kártyát. A játék fő részén az alkalmazás betölt **36** db kártyát amik gombként funkcionálnak és ha rákattint a játékos akkor felfordul és kiírja az eredmény amit sorsolt és ha eléri a 3 felfordítást vagy ha nyer mielőtt eléri a 3 kártya limitet, akkor megjelenik egy ablak ami közli a felhasználóval, hogy nem tud több kártyát megfordítani.

Amikor a játékos el szeretné indítani a játékot úgy, hogy az nap már játszott akkor ennek megfelelően megjelenik egy ablak ami közli, hogy most nem tud játszani és leállítja önmagát a játék.

FUNKCIONÁLIS RÉSZ

Form.cs

Ebben a fájlban szerepelnek a **responszibilitásért** felelős kódok, az adatbázisból lekérdezett adatokra válaszoló funkciók, a Fájl menüben lévő kilépés gomb működtető kód, és a kártyák betöltésért felelős funkciók.

KARTYAK.CS

Itt szerepel az összes adottságuk a kártyáknak, **szám generátor** ami 0 és 10 között sorsol egy számot és ha 1 vagy 2 lenne akkor megszorozza 10-el és azt adja vissza nyereménynek, a képbetöltését egy külön funkcióba lehet megtalálni, hogy könnyebb legyen elküldeni az elérési

útvonalát más funkciókba, és van még a kattintásra reagáló funkció amiben szerepel az, hogy ha 3 kártyát fordít fel akkor ne engedje tovább játszani a felhasználót, felfordításért felelős kódrészletek ahol a generátort meghívva dönti el, hogy nyert-e vagy sem.

DATABASE.CS

Ennek a fájlnek a nagyrészt Laurents Meyer „Pomelo” írta, aki azt oldotta meg, hogy a **C#** program direkt tudjon kommunikálni **MySQL** adatbázissal tehát a játék bárholnan eltudná érni az adatbázist és nem kell a htdocs mappába helyezni mint a php fájlokat.

A maradék kódrészletek azt csinálják, hogy példányosítják azt a táblát ahol dolgozik adatokkal a program, és lekérdezéseket hajtanak végre amire a **Form.cs**-ben kapnak választ és ha elég helyen megfelel az elvárt adatoknak akkor tud játszani a felhasználó és ha lejátszotta akkor feltölti a szükséges adatokat.

RESZPONZIVITÁS

USER INTERFACE

A weboldalunk fejlesztése során az egyik legfontosabb szempontunk a **reszponzivitás** volt, azaz hogy az oldal minden eszközön, legyen szó asztali gépről, tabletről vagy okostelefonról, hibátlanul és esztétikusan jelenjen meg. Ennek érdekében több technológiát és módszert is alkalmaztunk, melyek közül kiemelkedik a **Bootstrap** keretrendszer használata. A Bootstrap lehetővé tette számunkra, hogy előre definiált osztályokkal gyorsan és hatékonyan alakítsuk ki a felület struktúráját, a navigációt, a gombokat, valamint az egyéb UI-elemeket. A dropdown menük, navigációs sávok és rács-alapú elrendezések mind keretrendszer segítségével lettek implementálva.

Mobil nézet esetén a felület automatikusan alkalmazkodik a kisebb képernyőhöz. Például a fő navigációs sáv átalakul egy kattintásra lenyíló menüvé (*hamburger menü*), amely lefelé kinyílvá jeleníti meg a menüpontokat. Ez a megoldás nemcsak helytakarékos, hanem vizuálisan is letisztult, és a felhasználók számára jól érthető.

A tartalom is ehhez igazodik: amikor a lenyíló lista megjelenik, a mögötte lévő tartalom automatikusan lejjebb csúszik, így semmi nem takaródik el, megőrizve a vizuális egyensúlyt. A képek vízszintes görgetősávba kerülnek, amely egyedileg designolt, így eltér az alapértelmezett böngésző-görgetőtől – ez modern, esztétikus megjelenést kölcsönöz a galéria jellegű elemeknek.

A különböző tartalmak, mint például a kártyás elrendezések, dinamikusan alkalmazkodnak a kijelző méretéhez. A rendszer automatikusan több sorba rendezi őket, ha nem férnek ki egy sorba – ez a megközelítés leginkább **CSS Grid** rendszerként írható le. Ez biztosítja, hogy az elrendezés minden eszközön áttekinthető és konzisztens maradjon.

Egy másik fontos **UI**-funkció a keresőmező viselkedése. 1200 pixelnél kisebb képernyőszélesség esetén a keresőmező kitágul, átfedve a logó helyét, amely ekkor automatikusan eltűnik. Ez az elrendezés segít a felhasználónak a teljes keresésre koncentrálni, és javítja a szövegolvasás átláthatóságát.

Az adminisztrációs felület szintén reszponzív grid-rendszerre épül, lehetővé téve a tartalmak könnyű kezelését akár mobiltelefonról is. Ez különösen hasznos lehet a jövőbeli adminisztrátorok számára, akik gyors módosításokat végezhetnek útközben. Ugyanakkor fontos megemlíteni, hogy nagy mennyiségű adatbevitel vagy tartalomkezelés esetén továbbra is érdemes PC-t vagy laptopot használni a gyorsabb és hatékonyabb munkavégzés érdekében.

USER EXPERIENCE

A weboldal egyik kiemelkedő erőssége a **dinamikus működés**, amelyet igyekeztünk minden eszközön megőrizni. Célunk az volt, hogy a felhasználó már az első pillanattól kezdve gördülékeny és zökkenőmentes élményt kapjon, függetlenül attól, hogy milyen platformról látogatja meg az oldalt. Ehhez elengedhetetlen volt a frontend és a backend közötti hatékony kommunikáció kialakítása.

Ebben kulcsszerepet játszik az általunk alkalmazott **AJAX (Asynchronous JavaScript and XML)** technológia, amely lehetővé teszi, hogy az oldal bizonyos részei újratöltés nélkül frissüljenek. Az AJAX egyfajta „közvetítő réteggént” működik a kliens és a szerver között, így a felhasználó valós idejű visszajelzést kap az interakciói után – például keresésnél, űrlap beküldésnél vagy tartalom betöltésnél.

Ennek eredményeképpen a rendszer gyors válaszokat ad, nincs szükség az oldal teljes újratöltésére, ami jelentősen javítja a felhasználói élményt. A válaszidők optimalizálásával elértük, hogy az oldal gyors és gördülékeny interakciókat biztosít, minimalizálva a várakozási időt. A dinamikus tartalombetöltés és az azonnali vizuális visszajelzések (pl. loading animációk, validációs üzenetek) növelik az átláthatóságot és segítik a felhasználót a navigációban.

A **UX** kialakításánál szem előtt tartottuk, hogy ne csak esztétikus, hanem intuitív és felhasználóbarát felületet nyújtsunk. Kiemelt figyelmet fordítottunk arra is, hogy a navigáció

egyszerű és következetes legyen, a műveletek pedig egyértelmű visszajelzéseket adjanak (pl. gombnyomás után színváltozás, animáció vagy szöveges üzenet).

RESTFUL ÉS REST APIS ÚTVONALAK

RESTFUL API

A **RESTful API** lehetővé teszi, hogy a modern webes fejlesztések során teljes mértékben kihasználjuk a **HTTP** metódusokat, mint például a **GET, POST, PUT, DELETE** és más, szabványos műveleteket. Webáruházunk felépítésénél tudatosan törekedtünk arra, hogy az API-k kialakítása megfeleljen a RESTful elveknek, vagyis minden végpont egy adott erőforrásra (például könyv, kategória vagy kívánságlista) mutasson, és a hozzájuk tartozó műveletekhez megfelelő HTTP metódust rendeljünk. Ez azt jelenti, hogy például a könyvek lekérésére a **GET** /api/books útvonalat használjuk, míg egy új könyv létrehozására a **POST** /api/books végpont szolgál. Egy konkrét könyv törlésére a **DELETE** /api/books/{id} végpont felel, és hasonló elvet alkalmaztunk a kívánságlisták és admin funkciók esetében is. A kívánságlista kezelésénél főként **PUT** és **DELETE** metódusokat alkalmazunk a hozzáadásra és eltávolításra. Minden RESTful API válasza egységesen **JSON** formátumban érkezik, amely könnyen feldolgozható, strukturált adatot biztosít a frontend számára. Ez a megközelítés hozzájárul a webshop dinamikus működéséhez, a könnyű bővíthetőséghez, valamint a karbantartható és átlátható rendszer kialakításához.

REST API

A RESTful megoldások mellett **REST API**-szerű, de nem teljesen REST elveket követő végpontokat is alkalmaztunk. Ezek leginkább olyan PHP fájlokra mutatnak, amelyek lekérdezéseket hajtanak végre paraméterek alapján. Ilyen például az *info_backend.php?bookId=123* hívás, amely egy konkrét könyv adatait szolgáltatja. Ezek az URL-ek ugyan szintén HTTP-n keresztül kommunikálnak és JSON-t adnak vissza, de nem tekinthetők RESTful-nak, mivel nem erőforrás-orientált módon épülnek fel, és gyakran a fájl neve tartalmazza a művelet jellegét is. A REST API-jellegű hívásokat főként egyszerűbb, célzott feladatokhoz alkalmaztuk, ahol nem volt szükség a teljes REST struktúra kialakítására.

A két megközelítést tudatosan különválasztottuk: a **RESTful API** végpontokat egy külön api nevű mappában helyeztük el, míg a REST-alapú, fájlra mutató megoldások maradtak a hagyományos PHP fájlstruktúrában. Fontos szempont volt azonban, hogy a visszaküldött adat

minden esetben JSON formátumban történjen, ezzel biztosítva az egységes adatkezelést és feldolgozást a frontend oldalon.

TESZTELÉS

INDEX TESZT

A teszteléshez **Selenium web-drivert** használtunk. Az **index.js** tesztelése fontos, mert backend és frontend részeket is tartalmaz, továbbá fetchelést útvonalak összeköttetését és ajax kommunikációt is. A selenium web-driverje csak **szimulálni** képes, szóval csak egy szimulált környezetet állítottunk fel csakis erre a célra. Ellenőrizzük a kosarat, és a kívánságlistát is, amelyben elemeket teszünk, és ennek megfelelően megnézzük, hogy ezek az elemek betöltődnek-e, illetve műveletek elvégezhetőek-e velük. Továbbá a **localStorage**-os elemeket is lerendereljük, ezzel pontosan láthatjuk, hogy van-e bennük valami, és hogy milyen információk kaptak ott helyet. Az index-be 3 tesztesetet szimuláltunk le, az első, hogy tudunk-e **törölni** a kosárból szimulált környezetben persze, a második, hogy **hozzáaadni** tudunk-e, így a localStorage-hoz adódik, és a modal-ban csak megjelenítődnek ezek az item-ek, a mi esetünkbe könyvek. A harmadik, a tesztadatok **hozzáadása a kosárhoz**. Itt csak a kosárhoz adogattunk, azonban a kívánságlista is szinte ugyan az, szóval elég csak az egyik részét tesztelni, a másik is működni fog, ha ez is. Minden esetet **Log**-olunk, ami annyit takar, hogy minden eredményt eltárolunk egy .txt file-ba, és így pontosan látható bárki számára aki a teszteket ellenőrzi, hogy melyik helyesen futott le, és melyek azok amelyek esetlegesen megbuktak.

REGISTRATION TESZT

Az előzőleg használt **selenium webdrivert** használtuk itt is, csináltunk konstanként egy emailt és egy telefonszámot, és a driver segítségével betöltöttük a formba, majd regisztráltunk. Email címbe beletettük a dátumot, tehát ez mindig más lesz, azonban a telefonszám azonos, szóval csak 1 teszt lett helyes, a többi ahogy vártuk bukott, hiszen a telefonszám nem lehet azonos. Regisztrálás után átdobott a logination.php-re tehát az átirányítás is sikeres és minden esetet lelogoltunk, majd a driver futtatása befejeződött és leállt. Egy aszinkron függvényt használtunk csak, a **testRegistration()**-t, amelyet csak meg kell hívni, adat változtatáskor újra, bármikor tesztelhető persze.

LOGIN TESZT

A bejelentkezésnél is ugyan ezt a fajta szimulációs tesztkörnyezetet használtuk, ahol meg tudunk adni egy regisztrált emailt és jelszót, és meg tudjuk nézni hogy be tudunk-e ezáltal jelentkezni, vagy nem. Ha be tudunk akkor sikeres és tovább is enged, ha sikertelen kiírja, hogy mi miatt nem sikerült, és esetleg mit kell tennünk, hogy sikeres legyen. Minden esetet a txt-be dokumentálunk, és bármennyi teszt eset létrehozható! A teszteket a **testLogin** függvénybe deklaráljuk és meghívása után le is fut a driver és azonnal látjuk az eredményt.

JÁTÉK TESZT

A tesz **Nunit**-ban lett letesztelve amivel függőségben is van magával a játékkal mert olyan funkciót tesztel ami megnézi, hogy melyik elérési útvonalon van képnek megfelelő útvonal.

FELHASZNÁLÓI DOKUMENTÁCIÓ

WEBSHOPUNK CÉLJA

A webshopunk célja, hogy egy olyan helyet teremtsünk a felhasználóinknak, hogy biztonságosan, modern funkciókkal ellátott weboldalon tudjanak könyveket böngészni, és vásárolni. Nyújtunk játékot, melyben a felhasználók **értékes kuponokat** nyerhetnek, amelyeket vásárláskor fel tudnak használni. Továbbá fontos célunk, hogy növeljük az olvasás iránt rajongó személyek könyvvásárlási leltárát, ahol tudnak normál, papíros könyveket, vagy e-könyveket vásárolni. A mai modernebb világban sokan e-könyveket olvasnak, hiszen mondhatni környezetbarátabb, azonban vannak akik a régi papíros módszert részesítik előnyben, mi pedig helyet teremtünk mind a két fajtának.

HASZNÁLATI IGÉNYEK

Weboldalunkat a webről, szinte bármilyen hardver eszközről elérhetik, amelyen szerepel valamilyen webböngésző. Minden az adatbázisból indul ki, és köztes kommunikációval oldunk meg, régebbi elavultabb eszközön maximum lassabb lesz egy kicsivel a weboldal, azonban hiba nélkül betöltődik minden. **Domain-t** nem vásároltunk, ezért localhoston érhető el a weboldal, ezért kell egy **Xampp** nevű szoftver és az adatbázisunk, hogy teljesen böngészni tudjunk.

A WEBSHOP KÖRNYEZETÜNK FELÁLLÍTÁSA

A webshop környezetünk felállítása kicsit bonyolultan hangozhat, azonban ez egyáltalán nincs így, a látszat sokszor csal. Először is az **Apache**-ot konfiguráljuk. A windows gomb mellett található keresőgombot nyomja meg, és másolja be a következőt oda: `C:\xampp\apache\conf\httpd.conf` amennyiben nem a **C** meghajtón van a xampp, úgy annak

megfelelően módosítsa, pl. **D**:, **E**:, **F**: stb. Nyomjon egy entert, ha mindent jól csinált egy txt nyílt meg, amelynek lapozzon le a legaljára és másolja be a következőt:

Alias /bookshop/web C:/Users/{felhasznalo}/Documents/book-shop-website/web

<Directory C:/Users/{felhasznalo}/Documents/book-shop-website/web>

Options Indexes FollowSymLinks MultiViews

Require all granted

AllowOverride None

Order allow,deny

Allow from all

</Directory>

Alias /bookshop/php C:/Users/{felhasznalo}/Documents/book-shop-website/src

<Directory C:/Users/{felhasznalo}/Documents/book-shop-website/src>

Options Indexes FollowSymLinks MultiViews

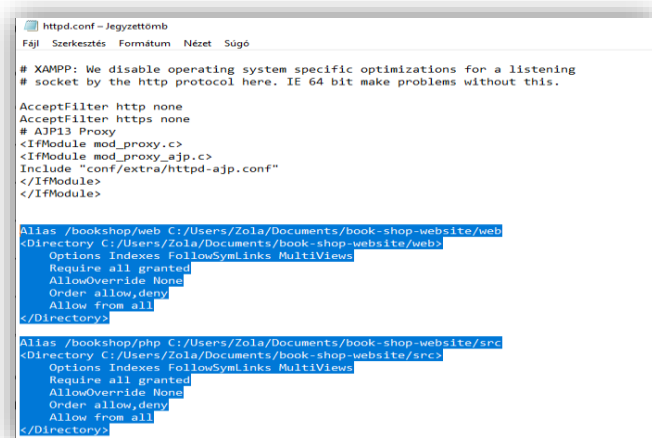
Require all granted

AllowOverride None

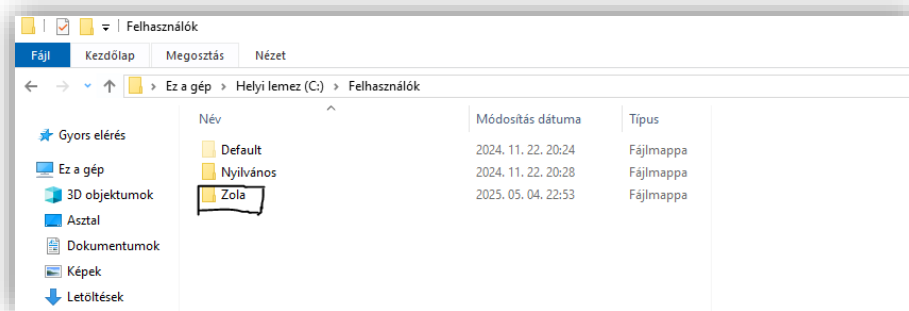
Order allow,deny

Allow from all

</Directory>



Ahova *{felhasznalo}* van írva, oda írja át a számítógépen belépett felhasználó nevet. Ezt a C meghajtón a **Users** mappában találja meg, ha esetleg nem tudta.



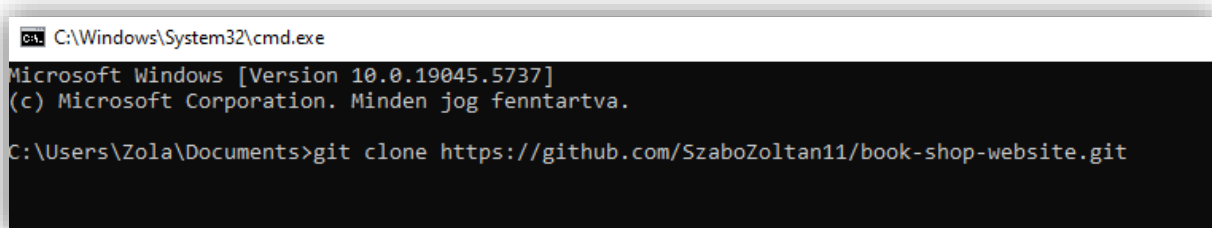
A webshopunk megtekintése után, érdemes törölni a **htp.d.conf**-ból az általunk bemásolásra kért kód részletet, ugyanis más projektek megtekintésénél komplikációk léphetnek fel.

Mit is csináltunk eddig? Az útvonalak ezáltal más helyről érhetőek el, ezt a későbbiekben látni lehet, illetve az „src” mappa „php” -ként lett átnevezve, könnyebb beazonosítás érdekében, azonban a funkciója még mindig source lesz.

Klónozza le a projektunket a Dokumentumok mappába, fontos hogy ide, ne máshova, mert ez az útvonal van kijelölve, nem kell a htdocs-ba csakis a **Dokumentumok** mappába. Ezt úgy lehet, hogy nyissa meg a dokumentumok mappát:



Kattintson a dokumentumok után és írja be simán hogy „cmd”, majd enter és egy cmd fog megnyílni.



Másolja be ezt utána: `git clone https://github.com/SzaboZoltan11/book-shop-website.git`

Majd nyomjon egy entert, és a webshopunk klónozva lett a dokumentumok mappába. Kérjük nyissa meg a mappát és lesz benne egy `konyvwebaurhaz.sql`, ezt importálja be új adatbázisként mysql-be természetesen.

Fontos, hogy mielőtt beimportálja az általunk „konyvwebaruhaz.sql”-t hozza létre az adatbázist, az alábbi parancs bemásolásával:

```
CREATE DATABASE konyvwebaruhaz
```

```
CHARACTER SET utf8
```

```
COLLATE utf8_hungarian_ci;
```

Ha ezzel megvan, akkor mentse a httpd.conf fájlt, vagy ha hamarabb elmentette és bezárta rendben van. Kérem indítsa el a **Xamppot**, majd az **Apache-t** és a **Mysql-t**.

Kérem másolja be az alábbi weboldal elérhetőségét a webböngésző **URL** részébe:
<http://localhost/bookshop/web/>

Ha betöltődött, és látja a könyveket, akkor sikeresen felállította a környezetet.


Első indítás előtt a játékot rebuildelni kell vagy egyszer le kell futatni Visual Studio 2022-ben, hogy az **exe** fájl létrejöjjön.

Fontos, hogy a következő **bat** fájl futatásához rendszergazdai jog kell (tehát jobb click és rendszergazdagént indítsa el), mert ez a fájl fogja az operációs rendszernek regisztrálni, hogy nem vírus az exe fájl ami játék exe fájlja és így eltudja indítani a weboldalról a játékot.

Ezután van egy register.reg bat fájl a „desktop” mappán belül amiben a felhasználónak megfelelően átkell írni a "Users\\" utáni felhasználót, és ez után le kell futatni a **bat** fájlt.

Windows Registry Editor Version 5.00

```
[HKEY_CLASSES_ROOT\game]
"URL Protocol"=""
[HKEY_CLASSES_ROOT\game\shell]
[HKEY_CLASSES_ROOT\game\shell\open]
[HKEY_CLASSES_ROOT\game\shell\open\command]
@="cmd /K \"cd /D C:\\Users\\xyzfelhasználó\\Documents\\book-shop-website\\desktop\\
```



A játék indítása közben megjelenik egy **cmd** amit ne zárjon be, mert a játék azzal nézzi, hogy melyik felhasználó játszik jelenleg és ha bezárja akkor a játék is bezárul, de ha már lejátszotta a jelenlegi kört és még mindig nyitva van akkor zárja be nyugodtan.

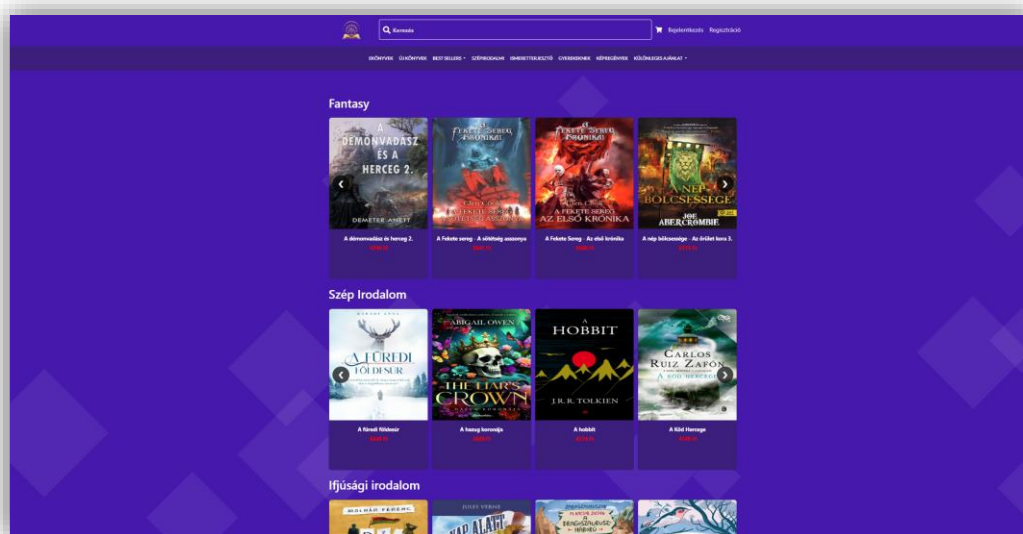
A játékot ellehet érni a felhasználói név rákattintása után lenyíló menüpontból ha bevan jelentkezve egy fiókba.

ÁLLAPOTOK

Ha nincs bejelentkezve, akkor korlátozott funkciókkal rendelkezik, pl. nem tudja használni a kívánságlista funkciót, és kuponokat sem tud érvényesíteni, azonban a böngészés és vásárlás továbbra is elérhető.


Ha a felhasználó regisztrált és bejelentkezett, akkor bővülnek a funkciói, tudja használni a kívánságlistát, és játszani is tudj, melyel értékes kuponokat gyűjthet.

Ha adminként van jelen, akkor admin oldal is elérhetővé válik, és könyveket tud felvenni, törölni stb. ezeket az állapotokat az **\$isAdmin** és az **\$isLogged** in változókkal figyeljük.



KEZDŐ OLDAL

REGISZTRÁCIÓ ÉS BEJELENTKEZÉS



Bejelentkezés

Email

Jelszó

[Jelszó megjelenítése](#)


☐ Emlékezz rám

Fiók létrehozásával elfogadod a Bookshop [Adatvédelmi Tájékoztatóját](#) és [Felhasználási Feltételeit](#).

Bejelentkezés

Nincs még fiókod? [Regisztráld itt](#)

[Elfelejtetted a jelszavad?](#)



Hozz létre egy fiókot a gyors vásárláshoz, speciális kuponkódokhoz, kívánságlistákhoz és rendelési előzményekhez.

Regisztráció

Család név:

Kereszt név:

Email

Telefonszám

Jelszó

Jelszó megerősítés

[Jelszó megjelenítése](#)

☐ Legalább 8 karakter

☐ Legalább egy szám (0-9) és egy speciális karakter

☐ Szeretnék hírleveleket és e-maileket kapni a Bookshop.org-tól (új könyvekről, szerzőkről és akciókról).


Fiók létrehozásával elfogadod a Bookshop [Adatvédelmi Tájékoztatóját](#) és [Felhasználási Feltételeit](#).

Regisztráció

Vagy [Bejelentkezés meglévő vásárlóként](#)

EGY KÖNYV SAJÁT KILISTÁZÁSA

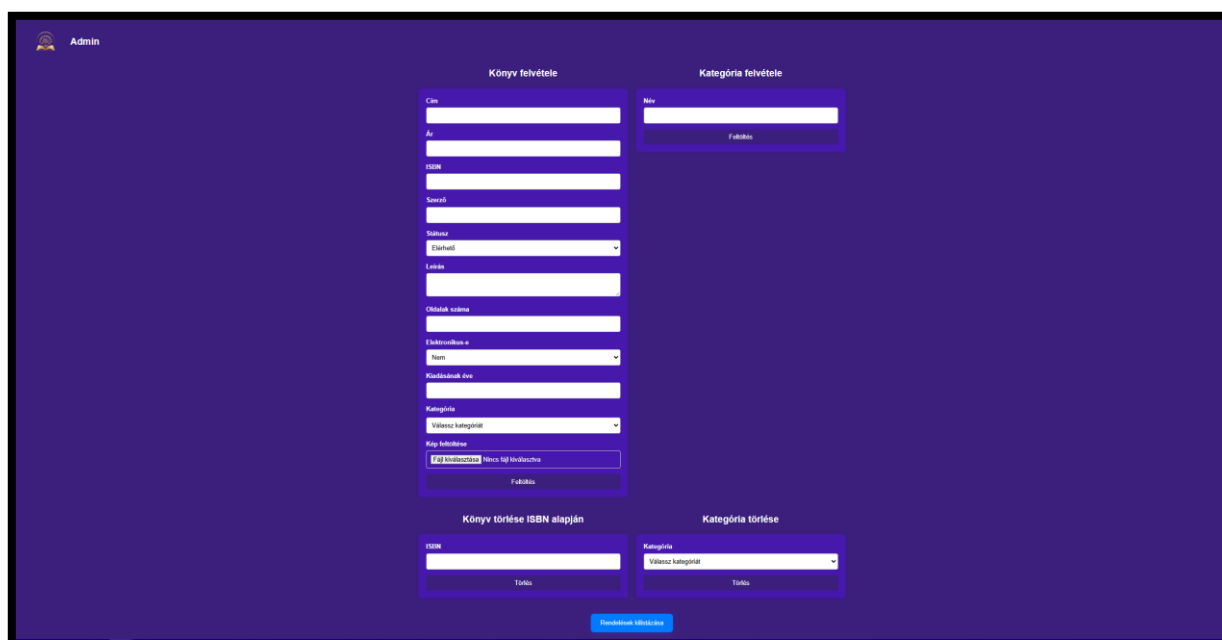
Vissza a földalra



Kosárba Kívánságlistába

Cím:	A démonvadász és herceg 2.
Ár:	4 740 Ft
Leírás:	<p>A démonok karmaiból épphogy csak megmenekültek - Thomas herceg máglyúsan fellángoló démonról kardja mentette meg őt, de a veszély korántsem múlt el. A Sötét Komédiás küldetés befejezése még nagyobb fenyegetést hoz rájuk, és a király hitetlen halálával Thomasból korondatlan, menekülőre kényszerülő király válik. A birodalom sorsa most az ő kezében van. Thomas és Ana újra kellenek - nemcsak azért, hogy sereget toborozzanak, hanem hogy meggyűjtsék a lángoló kard titkait és különleges kapcsolatuk rejtélyét. Ahogy a végül háború árnya egyre közelebb ér, egy fés prófécia beteljesedése is fenyegeti közös jövőjüket. De a felmerülő rejtélyek sötét árnyakat vetnek rájuk. Valóban elkerülhetetlen a háború? Miért áll Ana és Thomas kapcsolatuk közepén éppen ez a misztikus kard? És hogyan bátorít meg a két fiatal a rájuk nehezedő nyomásból, amikor még egymást is alig ismerik? Képesek lesznek megmenteni a világot, miközben saját szívüket is meg kell övniük? Vagy a prófécia örökre megváltoztatja a végzetüket? A Démonvadász</p>
Szerző:	Demeter Anett
Oldal szám:	312
Kiadás éve:	2024
ISBN:	9789633862689

ADMIN OLDAL



The screenshot displays the Admin interface with a dark blue background. At the top left, there is a logo and the word "Admin". The main content area is divided into four sections:

- Könyv felvétele**: A form for adding a new book. It includes input fields for Cím, Ár, ISBN, Szerző, Képlet, Elérhető (dropdown), Leírás, and a text area for Utóiratok száma. There is also a dropdown for Kiadások száma and a dropdown for Kategória. At the bottom, there are checkboxes for "Kijelölés" and "Nincs kijelölés", and a "Feltöltés" button.
- Kategória felvétele**: A form for adding a new category. It includes an input field for Név and a "Feltöltés" button.
- Könyv törlése ISBN alapján**: A form for deleting a book by ISBN. It includes an input field for ISBN and a "Törölés" button.
- Kategória törlése**: A form for deleting a category. It includes a dropdown for "Válassz kategóriát" and a "Törölés" button.

At the bottom center, there is a "Rendelések törlése" button.

ÖSSZEFOGLALÁS

Lényegében megvalósítottunk majdnem minden funkciót amelyet terveztünk, itt értjük az egész webshopot körülvevő logikát, az oldalakat, al-oldalakat, és funkciókat. A jövőben biztosan lesznek még változtatások, és tervezzük is folytatni, és véglegesíteni egy domain-en. Az útvonalakban még biztosan lesznek változások, hiszen ezek localhoston működnek, de egy teljesen más adatbázis környezetben kicsit máshogy lesznek elérhetőek. A játékot egy kis extrának terveztük, azonban egy elég nagy részének nötte ki magát, sok mindent megtanultunk a projekt közbeni dolgozás során, azonban az elejétől a végéig a tervünk szerint haladtunk, és egy kicsit túl is lőttünk a célunkon jó értelemben. Elég nagy részét kitettük könyvek keresgélésével, és felvételével amelyet Bence végzett és több mint 300 könyvet sikerült felvennie az adatbázisba, amely hatalmas munka volt. A Front- és Backend közben megérthettük, mennyire más azonban annyira inkább együtt működik a kettő, két külön terület, azonban mind a kettő elkészítése mégis nagyon jól sikerült szerintünk és sokat tanultunk. Az összedolgozás nem volt számunkra idegen, az említettem, hogy felállítottunk egy környezetet, ahol szinte mindenhol tudtunk használni, így egyszerű volt, egymást könnyen megértettük és logikusan építettük fel a teljes projektet, szóval nem voltak nagy kudarcok e téren.

TOVÁBBI CÉLKITŰZÉSEINK

Ahogy említettem, szeretnénk a továbbiakban beüzemelni a weboldalt és működtetni is. Szeretnénk fejleszteni a weboldalt mégjobban, hogy egyre jobb, és jobb legyen, több funkciót,

több könyvet is tervezünk hozzáadni, több fizetési módszert és kategóriát is. A telefonos UI-n is tervezünk még finomítani, hiszen használható és elég jó, modern, azonban van még rajta még mit csiszolni, de elégedettek vagyunk az eddigi munkánkkal. Törekedtünk végig a tiszta kódot tartani, azonban vannak olyan részek ahol ezen még lehetne változtatni, hogy jobb legyen. Az admin felületen akarunk könyv módosítást is betenni, hogy lehessen egyszerűen árat módosítani, illetve ilyen eventeket, pl. akciók, promóciókat akarunk, ezeket ilyen heti rendszerességgel, hogy növeljük majd a felhasználók számát és ez egy jó esemény lehet erre.

IRODALOM JEGYZÉK

Az általunk használt szakirodalomak, illetve a felhasznált composer és egyéb mechanikák listája.

Pomelo githubjának a neve: lauxjpn

PHPMailer composer a csatolt githubon is elérhető, azonban composer paranccsal is egyszerűen futtatható: *composer require phpmailer/phpmailer*

- <https://dev.mysql.com/doc/>
- <https://developer.mozilla.org>
- <https://www.w3schools.com>
- <https://www.php.net/manual/en/>
- <https://getbootstrap.com/>
- <https://www.blackbox.ai/>
- <https://stackoverflow.com>
- <https://github.com>
- <https://www.javascript.info/>
- <https://github.com/lauxjpn>
- <https://github.com/PHPMailer/PHPMailer/blob/master/composer.json>