Technikusi képesítés neve:
Szoftverfejlesztő és -tesztelő

VIZSGAREMEK

Todo App

Lipák Tibor — Konzulens

Hargitai Norbert Viktor, Puchele András — 13 / A

Budapest, 2025.

Tartalomjegyzék

Az oldal bemutatása

Témaválasztás

A vizsgaremek témájául egy teendőkezelő alkalmazás (Todo App) elkészítését választottuk, mivel egy olyan gyakorlati problémára kívántunk megoldást nyújtani, amely a mindennapi életben is hasznos lehet. A teendőkezelő alkalmazások népszerűsége és gyakorlati haszna inspirált bennünket: szerettünk volna egy olyan rendszert létrehozni, amely egyszerűen használható, ugyanakkor számos hasznos funkcióval rendelkezik. A célunk egy olyan webalkalmazás fejlesztése volt, amely lehetővé teszi a felhasználók számára teendőik rendszerezését, kezelését, valamint az alkalmazás kinézetének testreszabását egyedi skinek segítségével. A projekt során fontos szempont volt, hogy a rendszer biztonságos legyen, ezért olyan funkciókat is implementáltunk, mint a kétfaktoros hitelesítés (2FA) és a jelszó-visszaállítás.

A fejlesztés során nagy hangsúlyt fektettünk arra, hogy a felhasználói felület (UI) intuitív és esztétikus legyen, miközben a backend logika biztosítja az adatok biztonságos kezelését és tárolását. A projekt lehetőséget adott arra, hogy a szoftverfejlesztés és -tesztelés területén szerzett ismereteinket gyakorlatban is alkalmazzuk, például adatbázis-tervezés, API-készítés, valamint modern webes technológiák (HTML, CSS, JavaScript, PHP) használata terén.

A weboldal funkciója

A Todo App egy olyan webalkalmazás, amely lehetővé teszi a felhasználók számára, hogy teendőiket rendszerezzék és kezeljék egy modern, testreszabható felületen keresztül. Az alkalmazás főbb funkciói a következők:

- **Regisztráció és bejelentkezés:** A felhasználók létrehozhatnak egy fiókot, majd bejelentkezhetnek a rendszerbe. A regisztráció során a jelszavak titkosítva kerülnek tárolásra (bcrypt algoritmus használatával). A bejelentkezésnél a felhasználók választhatják a kétfaktoros hitelesítést (2FA), amely egy további biztonsági réteget biztosít.
- **Teendők kezelése:** A felhasználók új teendőket hozhatnak létre, amelyekhez címet, dátumot és kezdési időt adhatnak meg. A teendők egy táblázatos nézetben jelennek meg, ahol a felhasználók kipipálhatják őket, ha elkészültek, de csak akkor, ha a kezdési idő már elérkezett. A teendők törölhetők, és a teljesített teendőkért a felhasználók érméket (coins) kapnak, amelyeket később skinek vásárlására használhatnak fel.
- **Skinek és testreszabás:** A felhasználók különböző skineket választhatnak ki, amelyek megváltoztatják az oldal kinézetét. A skinek megvásárolhatók érmékkel, amelyeket a teendők teljesítésével lehet megszerezni. A kiválasztott skin a selected_skins táblában kerül tárolásra, és a megfelelő CSS fájl betöltésével alkalmazódik az oldalon.
- **Profil és fiókkezelés:** A felhasználók megtekinthetik és szerkeszthetik profiladataikat (név, email), valamint kezdeményezhetik fiókjuk törlését. A fiók törlése egy visszaigazoló emailben található link segítségével véglegesíthető, amely biztosítja, hogy illetéktelenek ne tudják törölni a fiókot.
- **Biztonsági funkciók:** Az alkalmazás támogatja a kétfaktoros hitelesítést (2FA), amely során a felhasználók egy 6 jegyű kódot kapnak emailben, amit meg kell adniuk a bejelentkezéshez. Emellett a rendszer jelszó-visszaállítási lehetőséget kínál, és kezeli a megbízható eszközöket, hogy a 2FA-val rendelkező felhasználóknak ne kelljen minden bejelentkezéskor kódot megadniuk.
- **Értesítések és visszajelzések:** Az alkalmazás különböző üzeneteket jelenít meg a felhasználó számára, például sikeres bejelentkezésről, hibákról (pl. "Sikertelen bejelentkezés!"), vagy arról, ha egy teendő még nem pipálható ki a kezdési idő miatt (pl. "Nem pipálható ki 14:00:00-ig!").

Az oldal jövője

A Todo App fejlesztése során számos ötlet merült fel, amelyekkel a jövőben tovább bővíthetjük az alkalmazást. A terveink között szerepelnek az alábbiak:

- **Teendők kategorizálása és szűrése:** A felhasználók számára lehetővé tesszük, hogy címkéket adjanak a teendőkhöz, így könnyebben rendszerezhetik azokat. Például egy teendőhöz hozzárendelhető a "munka", "személyes" vagy "sürgős" címke, és a teendők szűrhetők lesznek ezek alapján.

- Mobilalkalmazás fejlesztése: Az alkalmazást szeretnénk mobilplatformokra is eljuttatni, például egy natív Android és iOS alkalmazás formájában. Ez lehetővé tenné a felhasználók számára, hogy bárhol, bármikor hozzáférjenek teendőikhez.

- További skinek és testreszabási lehetőségek: Új skineket szeretnénk hozzáadni, valamint lehetőséget biztosítani a felhasználóknak, hogy saját színsémákat állítsanak be. Például egy színválasztóval a felhasználók maguk állíthatnák be a háttérszínt vagy a gombok színét.

- Admin felület létrehozása: Egy adminisztrátori felületet tervezünk, amely lehetővé teszi a rendszergazdák számára a felhasználók kezelését, például fiókok törlését, skinek hozzáadását, vagy a teendők monitorozását.

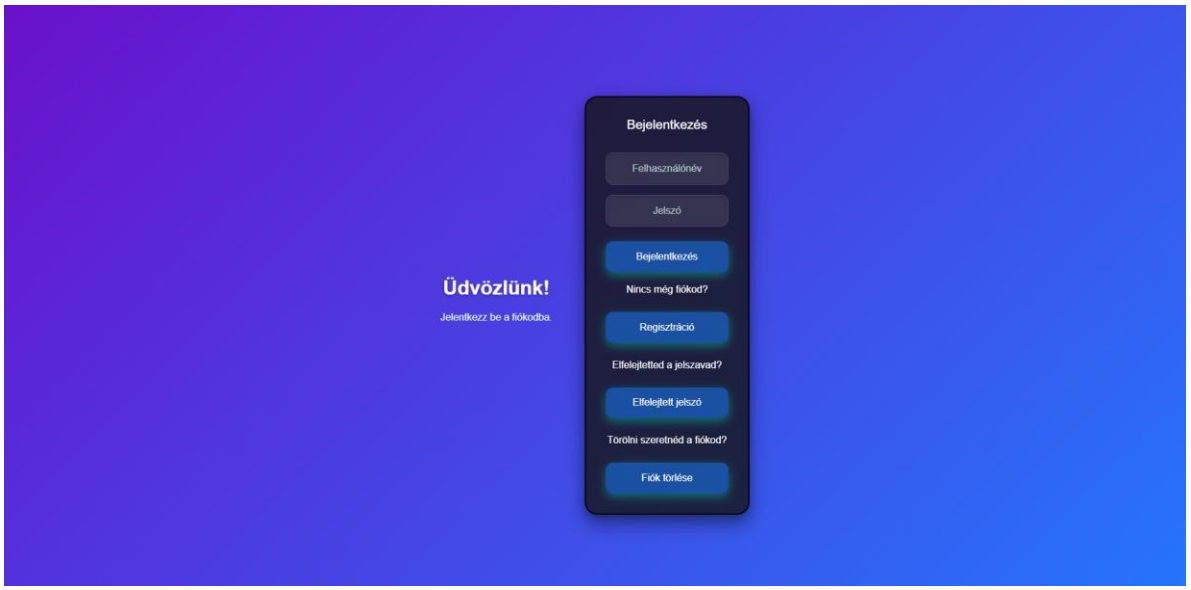
- Többnyelvű támogatás: Az alkalmazást több nyelvre is lefordítanánk, hogy nemzetközi közönség számára is elérhető legyen. Ez magában foglalná a magyar mellett például az angol, német és spanyol nyelvi támogatást.

Felhasználói felület bemutatása

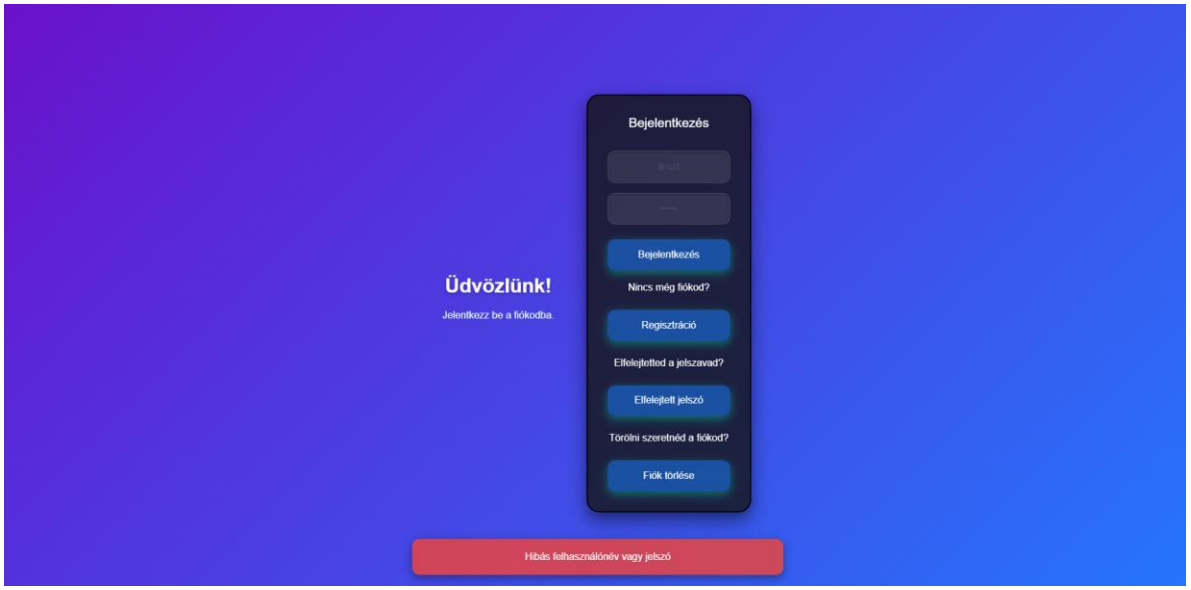
Bejelentkezés

A bejelentkezési felület (index.html) a felhasználók számára a rendszerbe való belépést biztosítja. Az oldalon egy űrlap található, amelyben a felhasználónak meg kell adnia a felhasználónevét és jelszavát. A bejelentkezés gomb megnyomásakor a rendszer ellenőrzi az adatokat, és ha azok helyesek, a felhasználót átirányítja a teendők kezelésére szolgáló oldalra (todo.html). Sikertelen bejelentkezés esetén egy piros háttérrel rendelkező üzenet jelenik meg a "Sikertelen bejelentkezés!" szöveggel. Ha a felhasználó engedélyezte a kétfaktoros hitelesítést, akkor a bejelentkezés után egy 6 jegyű kód megadására szolgáló oldalra (verify_2fa.html) kerül átirányításra, amelyet emailben kap meg.

1. ábra: Bejelentkezési felület



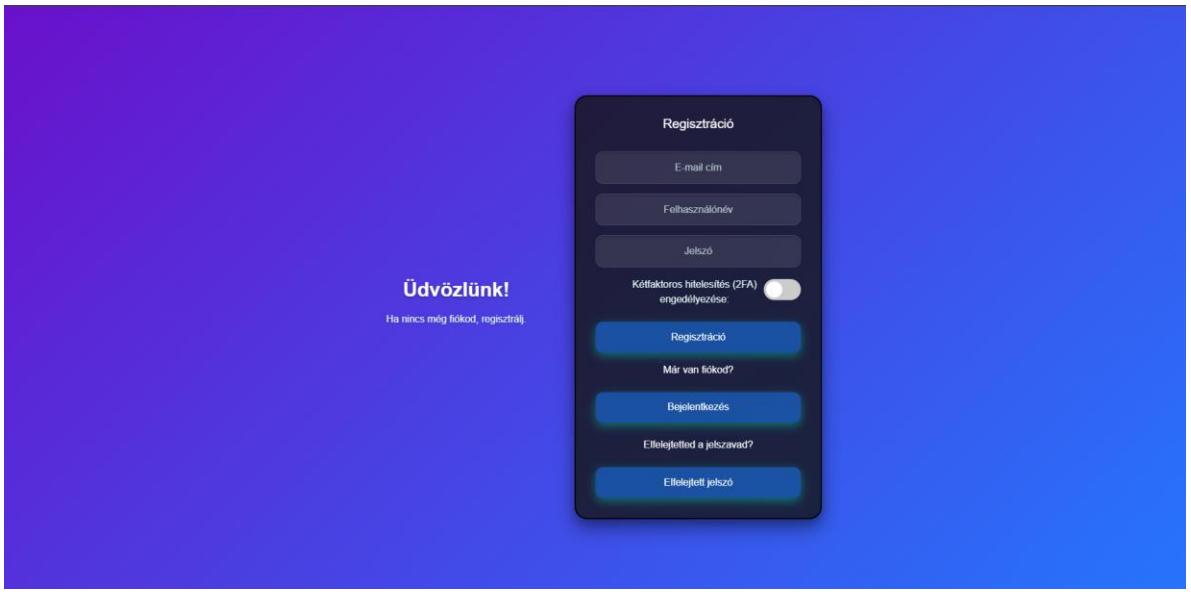
2. ábra: Bejelentkezés sikertelen üzenet



Regisztráció

A regisztrációs felület (registration.html) lehetővé teszi új felhasználók számára, hogy fiókot hozzanak létre a Todo App rendszerében. Az oldalon egy űrlap található, amelyben a felhasználónak meg kell adnia az email címét, felhasználónevét, jelszavát, valamint egy opciót a kétfaktoros hitelesítés (2FA) engedélyezésére. A jelszó titkosítva kerül tárolásra az adatbázisban a bcrypt algoritmus segítségével, amely biztosítja, hogy a jelszavak ne legyenek könnyen visszafejthetők. Sikeres regisztráció után a felhasználó egy emailt kap egy verifikációs kóddal, amelyet meg kell adnia a regisztráció véglegesítéséhez. Ezt követően átirányításra kerül a bejelentkezési felületre (index.html).

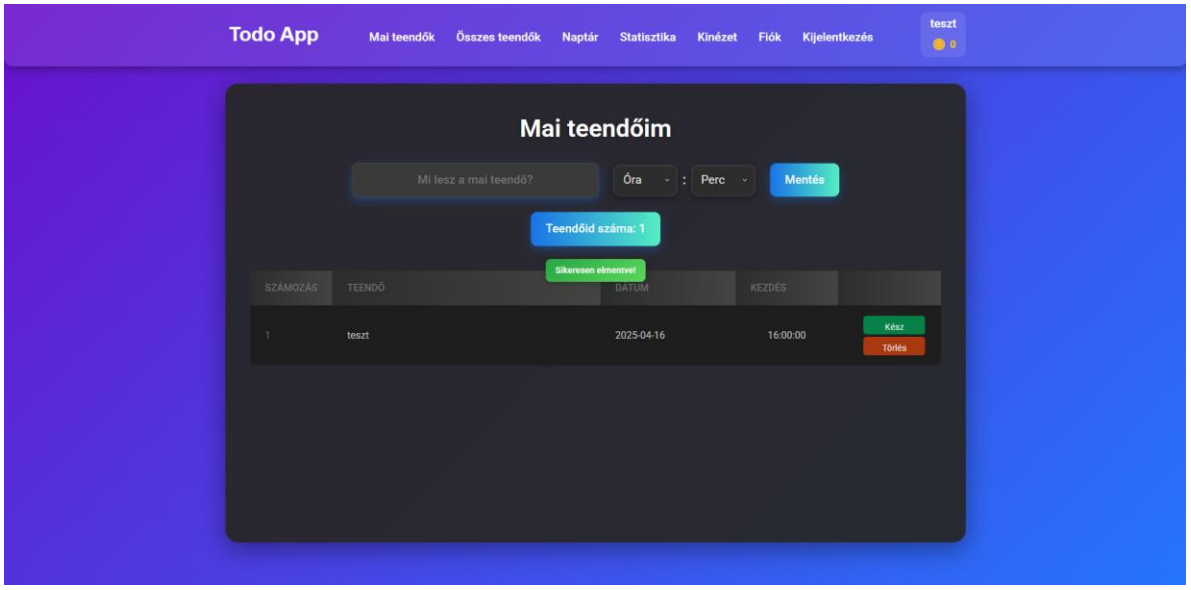
3. ábra: Regisztrációs felület



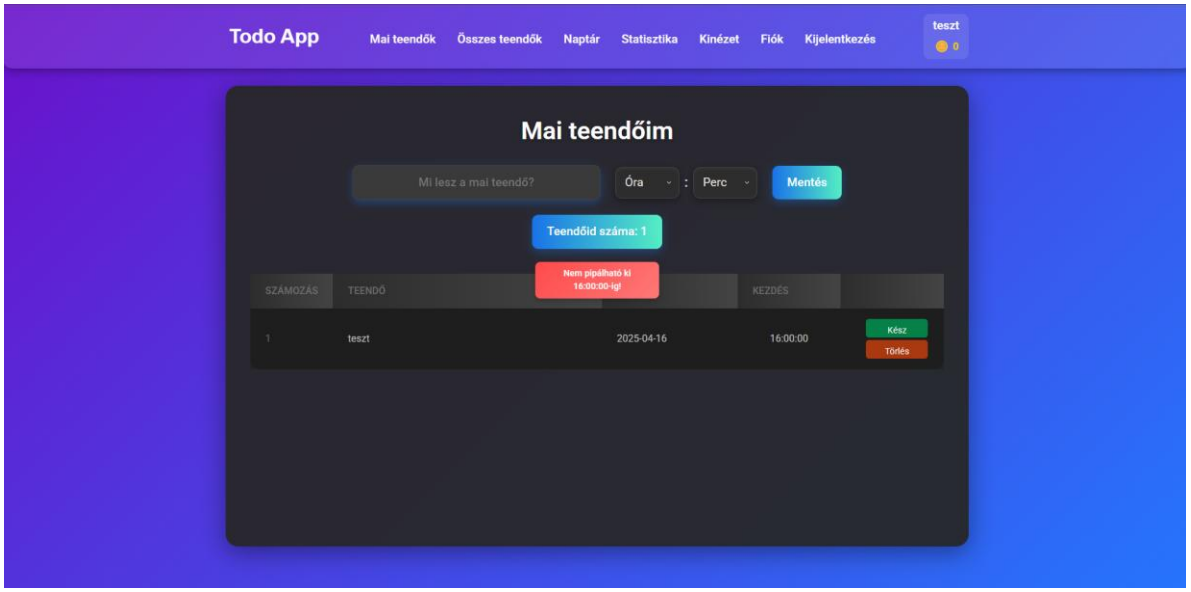
Teendők kezelése

A teendők kezelése a todo.html oldalon történik, amely a rendszer központi része. Az oldalon a felhasználók egy űrlap segítségével új teendőt hozhatnak létre, amelyhez megadhatnak egy címet, dátumot és kezdési időt. A teendők egy táblázatos formában jelennek meg, ahol az egyes sorok tartalmazzák a teendő sorszámát, címét, dátumát, kezdési idejét, valamint a műveleti gombokat (kipipálás, visszavonás, törlés). A teendők csak akkor pipálhatók ki, ha a kezdési idő már elérkezett; ellenkező esetben egy piros üzenet jelenik meg, például: "Nem pipálható ki 14:00:00-ig!". A teljesített teendőkért a felhasználók érméket kapnak, amelyeket a skinek vásárlására használhatnak fel.

4. ábra: Teendők kezelése felület



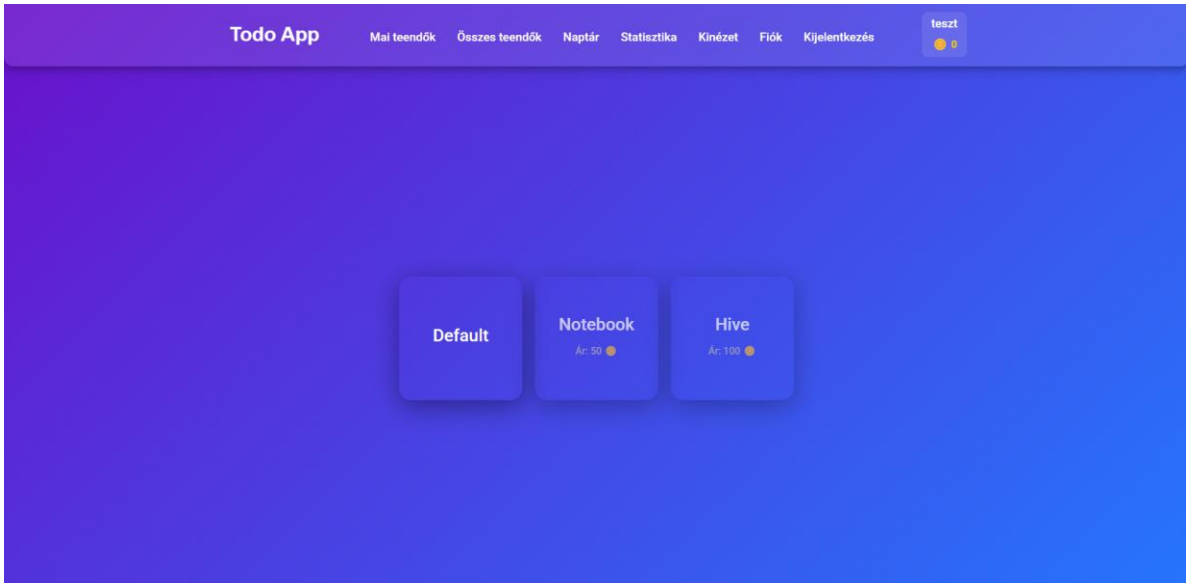
5. ábra: Teendő nem kipipálható üzenet

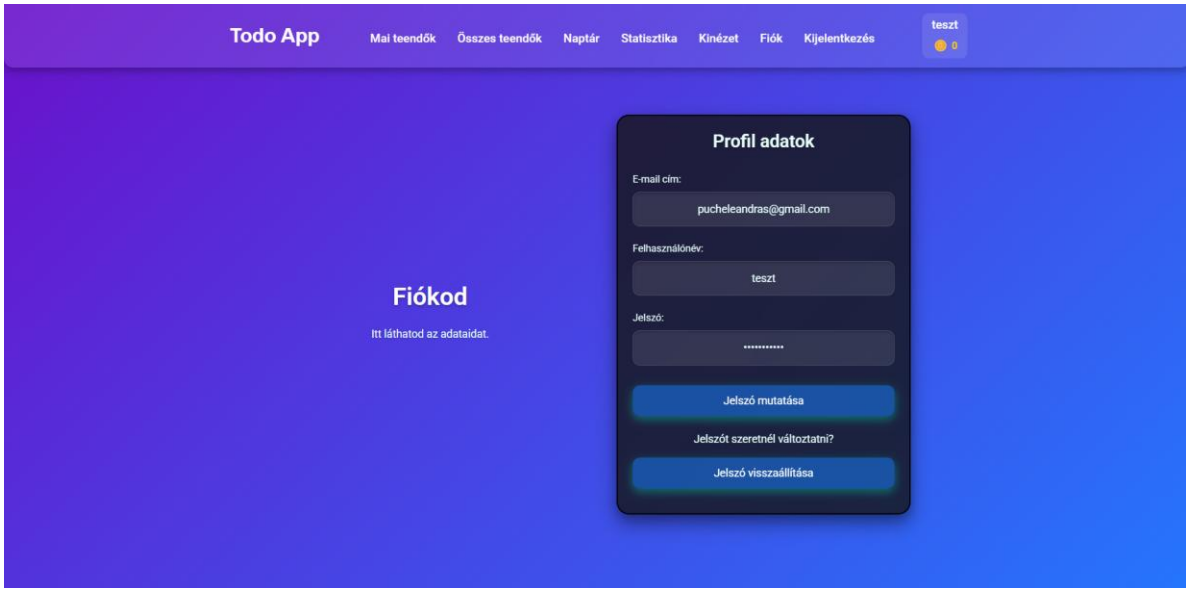


Profil és skinek

A profil oldalon (account.html) a felhasználók megtekinthetik adataikat, például a nevüket, email címüket, és a jelszavukat. Az oldalon lehetőség van a jelszó megváltoztatására is, amely során a rendszer ellenőrzi a jelszó erősségét (legalább 8 karakter, kis- és nagybetű, szám, speciális karakter). A skinek kezelése a skins.html oldalon történik, ahol a felhasználók különböző skineket választhatnak ki, amelyek megváltoztatják az oldal kinézetét. A skinek megvásárolhatók érmékkel, és a kiválasztott skin a selected_skins táblában kerül tárolásra. A skinek alkalmazása során a rendszer a megfelelő CSS fájlt tölti be az oldal betöltésekor.

6. ábra: Profil és skinek kiválasztása

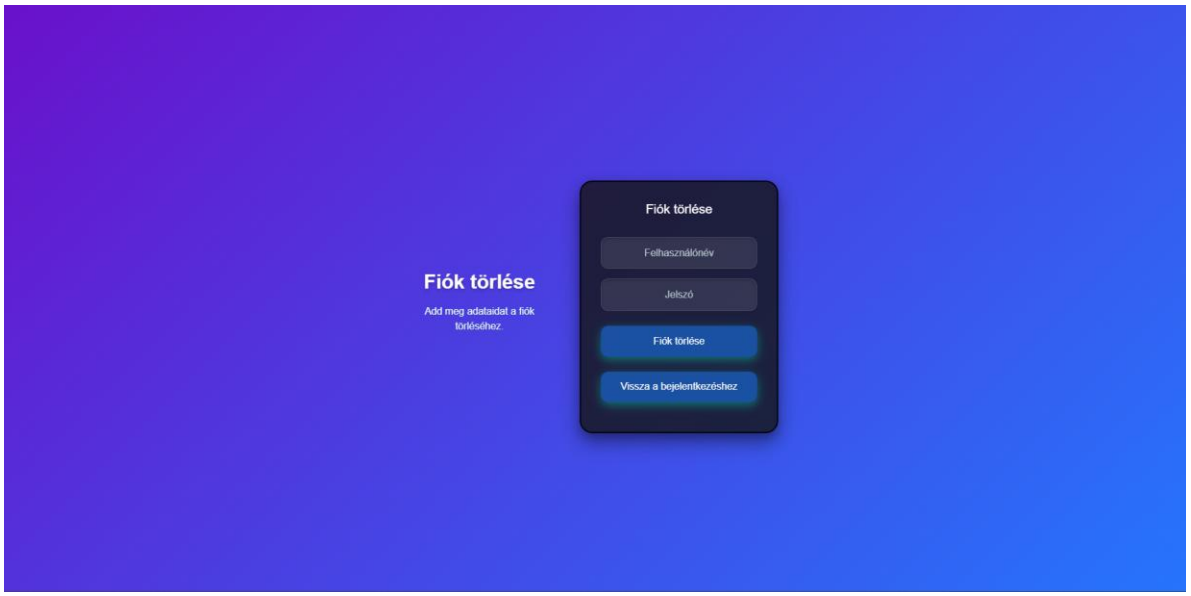




Fiók törlése

A fiók törlési kérelem a delete_account.html oldalon keresztül kezdeményezhető a bejelentkezési felületről. A felhasználó egy gomb megnyomásával kérheti a fiókja törlését, amelynek során a rendszer egy visszaigazoló emailt küld ki. Az emailben található linkre kattintva a felhasználó véglegesítheti a törlést, amely során a felhasználó adatai törlődnek az adatbázisból. A törlési kérelem adatai a delete_requests táblában tárolódnak, amely biztosítja, hogy a törlési folyamat nyomon követhető legyen, és illetéktelenek ne tudják törölni a fiókot.

7. ábra: Fiók törlési kérelem felület



Fejlesztői dokumentáció

Bejelentkezés fejlesztői dokumentáció

A bejelentkezés funkcionalitása két fő részre oszlik: a frontend (index.html és todo.js) kezeli a felhasználói felületet és a kliensoldali logikát, míg a backend (login.php) az autentikációt végzi és biztosítja a szerveroldali adatkezelést.

Frontend (index.html és todo.js):

Az index.html oldalon egy űrlap található, amelyben a felhasználó megadja a felhasználónevét és jelszavát. Az űrlap elküldésekor az todo.js fájlban definiált login függvény egy AJAX kérést küld a login.php fájlnak a POST metódussal. A kérés JSON formátumban tartalmazza a felhasználónév és jelszó adatokat. A válaszban a rendszer egy JSON objektumot küld vissza, amely jelzi, hogy a bejelentkezés sikeres volt-e, valamint tartalmazza a felhasználó adatait (pl. user_ID, token, username, coins). Ezek az adatok a localStorage-ban tárolódnak, hogy a munkamenet során a felhasználó azonosítható legyen, és az alkalmazás különböző oldalain (pl. todo.html, account.html) használhatók legyenek. Sikertelen bejelentkezés esetén egy piros háttérrel rendelkező üzenet jelenik meg a "Sikertelen bejelentkezés!" szöveggel.

Ha a felhasználó engedélyezte a kétfaktoros hitelesítést (2FA), a login.php egy session ID-t küld vissza, és a felhasználó átirányításra kerül a verify_2fa.html oldalra, ahol meg kell adnia az emailben kapott 6 jegyű kódot. A 2FA kód a two_factor_codes táblában tárolódik, alapértelmezés szerint 10 percig érvényes az egyszeri kód, míg a tartós (persistent) kód 30 napig érvényes.

Az todo.js fájlban a check függvény ellenőrzi, hogy van-e már érvényes token a localStorage-ban. Ha igen, akkor a rendszer automatikusan átirányít a todo.html oldalra anélkül, hogy a felhasználónak újra be kellene jelentkeznie. Ez biztosítja a zökkenőmentes felhasználói élményt, miközben a biztonságot is fenntartja a token alapú autentikációval.

Backend (login.php):

A login.php fájl a szerveroldali logikát kezeli a bejelentkezéshez. A következő lépéseket végzi:

- Adatellenőrzés: A beérkező POST kérésből kinyeri a felhasználónevet és jelszót, majd ellenőrzi, hogy a felhasználónév létezik-e a users táblában egy SELECT SQL lekérdezéssel.

- Jelszó-ellenőrzés: Ha a felhasználónév létezik, a password_verify függvény segítségével ellenőrzi, hogy a megadott jelszó megegyezik-e a tárolt titkosított jelszóval (amelyet a password_hash függvény bcrypt algoritmusa titkosít).

- 2FA ellenőrzés: Ha a felhasználó engedélyezte a 2FA-t (a users tábla two_factor_enabled mezője 1), akkor a rendszer ellenőrzi, hogy az aktuális eszköz megbízható-e a trusted_devices táblában. Ha az eszköz nem megbízható, a rendszer két 6 jegyű kódot generál (egy egyszeri és egy tartós kódot), amelyeket a two_factor_codes táblába ment. Ezután egy session ID-t generál, amelyet a login_sessions táblában tárol, és egy emailt küld a felhasználónak a kódokkal a PHP Mailer segítségével.

- Token generálás: Ha a 2FA nem szükséges, vagy a 2FA kód sikeresen ellenőrizve lett, a rendszer egy egyedi tokent generál (a bin2hex(random_bytes(16)) függvény segítségével), amelyet a logins táblába ment a login_token mezőbe. A login_state mezőt 1-re állítja, jelezve, hogy a felhasználó be van jelentkezve.

- Válasz küldése: A login.php egy JSON választ küld vissza, amely tartalmazza a sikeres bejelentkezés esetén a felhasználó adatait (pl. user_ID, token, username, coins), vagy hiba esetén egy hibaüzenetet (pl. "Hibás felhasználónév vagy jelszó").

A login.php fájlban külön figyelmet fordítottunk a biztonsági szempontokra, például a bemenetek megfelelő tisztítására (real_escape_string használata az SQL injekciók ellen), valamint a hibakezelésre (pl. adatbázis hibák naplózása).

Regisztráció fejlesztői dokumentáció

A regisztráció hasonlóan két részre oszlik: a frontend (registration.html és reg.js) kezeli a felhasználói felületet, míg a backend (registration.php) az adatbázisba történő mentést és az emailes verifikációt végzi.

Frontend (registration.html és reg.js):

A registration.html oldalon egy űrlap található, amelyben a felhasználó megadja az email címét, felhasználónevét, jelszavát, valamint egy jelölőnégyzet segítségével engedélyezheti a 2FA-t. Az űrlap elküldésekor a reg.js fájlban definiált register függvény egy AJAX kérést küld a registration.php fájlnak a POST metódussal. A kérés JSON formátumban tartalmazza az űrlap adatait. A válasz alapján a rendszer üzenetet jelenít meg a felhasználónak: sikeres regisztráció esetén egy zöld háttérrel rendelkező üzenet jelenik meg ("Sikeres regisztráció!"), majd a felhasználó átirányításra kerül a bejelentkezési felületre (index.html). Sikertelen regisztráció esetén (pl. ha az email már létezik) egy piros üzenet jelenik meg a megfelelő hibaüzenettel.

A regisztráció során a felhasználó egy verifikációs kódot kap emailben, amelyet egy külön mezőben kell megadnia a registration.html oldalon (egy dinamikusan megjelenő codeInput div-ben). A kód ellenőrzése a verifyCode függvény segítségével történik, amely szintén egy AJAX kérést küld a registration.php fájlnak.

Backend (registration.php):

A registration.php fájl a következő lépéseket végzi:

- Adatellenőrzés: Ellenőrzi, hogy az email cím már létezik-e a users táblában egy SELECT lekérdezéssel. Ha igen, egy hibaüzenetet küld vissza ("Ez az email cím már regisztrálva van!").

- Jelszó titkosítás: A jelszót a password_hash függvény segítségével titkosítja a bcrypt algoritmus használatával, majd előkészíti az adatbázisba történő mentésre.

- Felhasználó mentése: Egy INSERT INTO SQL paranccsal beszúrja az új felhasználót a users táblába, a user_email, user_name, user_pw, two_factor_enabled és coins mezők kitöltésével (a coins alapértelmezetten 0).

- Verifikációs kód küldése: Egy 6 jegyű verifikációs kódot generál, amelyet a two_factor_codes táblába ment, és emailben elküldi a felhasználónak a PHP Mailer segítségével. Az email tartalmazza a kódot és egy üdvözlő üzenetet.

- Kód ellenőrzése: Amikor a felhasználó megadja a verifikációs kódot, a registration.php ellenőrzi, hogy a kód érvényes-e a two_factor_codes táblában. Ha igen, a felhasználó fiókja aktiválódik, és a kód törlésre kerül a táblából.

A registration.php fájlban szintén figyelmet fordítottunk a biztonsági szempontokra, például az SQL injekciók elleni védelemre és a hibakezelésre. Az email küldés során a PHP Mailer könyvtárat használtuk, amely biztosítja az SMTP alapú email küldést.

Teendők kezelése fejlesztői dokumentáció

A teendők kezelése a todo.html és todo.js fájlokban történik a frontend oldalon, míg a backendet az api/entities/todos mappában található PHP fájlok biztosítják.

Frontend (todo.html és todo.js):

A todo.html oldalon egy űrlap található, ahol a felhasználó új teendőt adhat fel. Az űrlap tartalmaz egy szövegmezőt a teendő címéhez (title), egy dátumválasztót (date), egy időválasztót (start_time), valamint egy "Mentés" gombot. Az űrlap elküldésekor a todo.js fájlban definiált saveTodo függvény egy AJAX kérést küld a POST metódussal az API-nak (api/?token=...&userid=...&entity=todos), amely a teendőt a todos táblába menti.
A teendők egy táblázatos formában jelennek meg, ahol minden sor tartalmazza a teendő sorszámát (id), címét (title), dátumát (date), kezdési idejét (start_time), valamint a műveleti gombokat (kipipálás, visszavonás, törlés). A todoState függvény kezeli a teendők állapotának változtatását (kipipálás vagy visszavonás). Ez a függvény ellenőrzi, hogy a teendő kipipálható-e a kezdési idő alapján: ha a jelenlegi időpont korábbi, mint a start_time, akkor egy piros üzenet jelenik meg a save-message div-ben, például: "Nem pipálható ki 14:00:00-ig!". Ha a teendő kipipálható, akkor egy PATCH kérés küldésével a completed mező értéke 1-re módosul, és a felhasználó érméket kap (a rewarded mező 1-re állítása után).
A deleteTodo függvény a teendő törlését végzi egy DELETE kéréssel az API-n keresztül, amely a todos táblából törli a megfelelő rekordot.

Backend (api):

Az API a todos táblával kommunikál a következő módon:

- POST kérés: Új teendőt szúr be a todos táblába egy INSERT INTO SQL paranccsal, amely tartalmazza a user_id, title, date, start_time, completed (alapértelmezetten 0) és rewarded (alapértelmezetten 0) mezőket.

- PATCH kérés: A teendő állapotát módosítja (pl. completed mező 1-re állítása). Ha a teendő először lesz kipipálva (rewarded mező 0-ról 1-re vált), a felhasználó érméket kap, amelyeket a users tábla coins mezőjében tárolunk.

- DELETE kérés: Törli a teendőt a todos táblából egy DELETE SQL paranccsal, a user_id és a teendő id alapján.

Az API-ban külön figyelmet fordítottunk a jogosultságkezelésre: minden kérésnél ellenőrizzük a token érvényességét a logins táblában, hogy csak az adott felhasználó férhessen hozzá a saját teendőihez.

Skinek kezelése fejlesztői dokumentáció

A skinek kezelése a skins.html oldalon történik a frontend oldalon, míg a backendet a skins.php fájl biztosítja.

Frontend (skins.html és skins.js):

A skins.html oldalon a felhasználók megtekinthetik a rendelkezésre álló skineket, amelyek a skins táblából származnak. A skinek adatai (pl. skin_name, css_file, price) egy táblázatos formában jelennek meg, ahol látható, hogy az adott skin megvásárolható-e (a listable mező alapján), valamint hogy a felhasználó már megvásárolta-e (az unlocked_skins tábla alapján). A felhasználó kiválaszthat egy skint, amely a selected_skins táblában kerül tárolásra.

A skins.js fájlban a fetchSkins függvény egy GET kérést küld a skins.php fájlnak, amely lekéri a skineket és a felhasználó adatait (pl. coins, már megvásárolt skinek). A purchaseSkin függvény kezeli a skin vásárlását: ellenőrzi, hogy a felhasználónak van-e elég érméje, és ha igen, egy POST kérést küld a skins.php fájlnak a vásárlás végrehajtására. A selectSkin függvény a skin kiválasztását végzi, amely a selected_skins táblába menti a választást.

Backend (skins.php):

A skins.php fájl a következő lépéseket végzi:

- GET kérés: Lekéri a skins táblából a skineket, valamint az unlocked_skins és selected_skins táblák alapján ellenőrzi, hogy a felhasználó mely skineket vásárolta meg, és melyik skin van kiválasztva. A válasz JSON formátumban tartalmazza a skinek adatait és a felhasználó érme-egyenlegét.

- POST kérés (vásárlás): Ellenőrzi, hogy a felhasználónak van-e elég érméje a skin megvásárlásához. Ha igen, beszúr egy rekordot az unlocked_skins táblába, és levonja az érméket a users tábla coins mezőjéből egy UPDATE SQL paranccsal.

- POST kérés (kiválasztás): Frissíti a selected_skins táblát a kiválasztott skin azonosítójával (skinID) és a felhasználó azonosítójával (userID), valamint a kiválasztás dátumával (select_date).

- A skins.php fájlban szintén biztosítjuk a jogosultságkezelést a token ellenőrzésével, valamint figyelmet fordítunk az adatbázis tranzakciók integritására (pl. az érmék levonása és a skin feloldása egy tranzakcióban történik).

Fiók törlése fejlesztői dokumentáció

A fiók törlése a delete_account.html és delete_account.php fájlokban történik.

Frontend (delete_account.html és delete_account.js):

A delete_account.html oldalon egy gomb található, amelyre kattintva a felhasználó kezdeményezheti a fiókja törlését. A kérés egy AJAX kéréssel kerül elküldésre a delete_account.php fájlnak a delete_account.js fájlban definiált requestDelete függvény segítségével. A válasz alapján egy üzenet jelenik meg: sikeres kérelem esetén a felhasználó egy zöld üzenetet kap ("Törlési kérelem elküldve!"), és egy emailt kap a törlés véglegesítéséhez szükséges linkkel. A linkre kattintva a felhasználó egy külön oldalon (pl. confirm_delete.html) véglegesítheti a törlést.

Backend(delete_account.php):

A delete_account.php fájl a következő lépéseket végzi:

- Token ellenőrzés: Ellenőrzi, hogy a felhasználó be van-e jelentkezve a logins tábla alapján.

- Törlési kérelem mentése: Egy egyedi tokent generál, amelyet a delete_requests táblába ment a felhasználó azonosítójával (user_id) és a kérelem időpontjával (created_at) együtt.

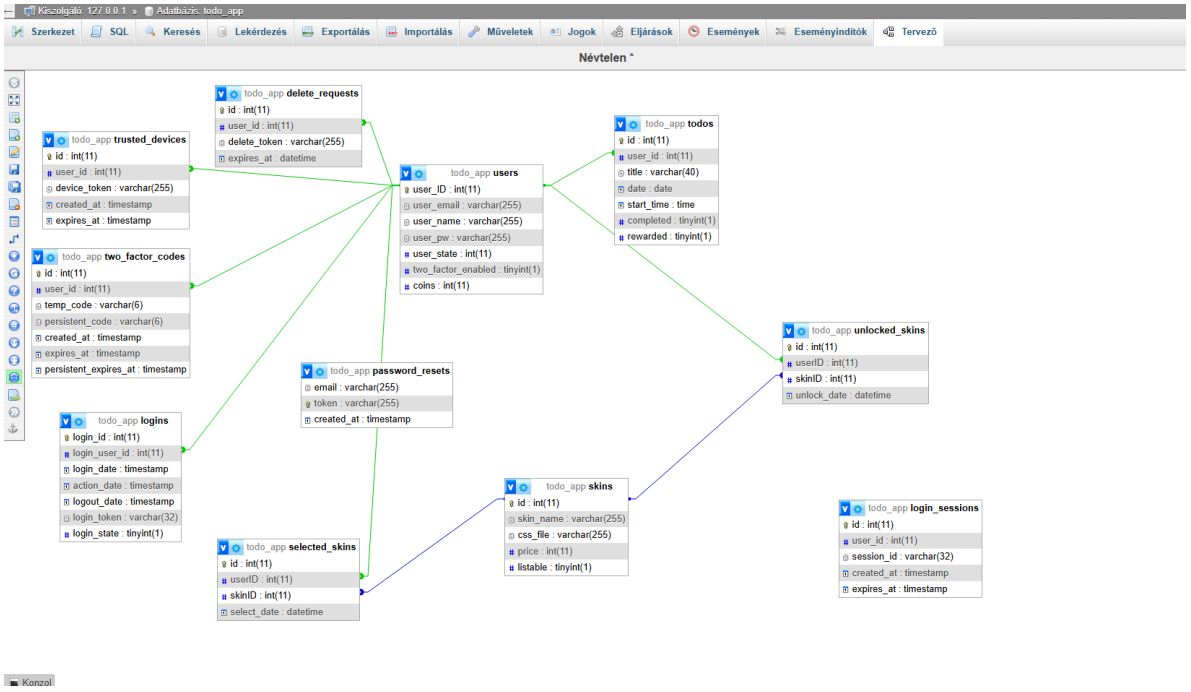
- Email küldése: Egy emailt küld a felhasználónak a PHP Mailer segítségével, amely tartalmazza a törlés véglegesítéséhez szükséges linket. A link a generált tokent is magában foglalja (pl. confirm_delete.html?token=...).

- Törlés véglegesítése: Amikor a felhasználó a linkre kattint, a confirm_delete.php fájl ellenőrzi, hogy a token érvényes-e a delete_requests táblában. Ha igen, törli a felhasználó adatait a users táblából egy DELETE SQL paranccsal, valamint az összes kapcsolódó adatot a többi táblából (pl. todos, logins, selected_skins) a foreign key constraint-ek segítségével.

A delete_account.php fájlban biztosítjuk, hogy a törlési folyamat biztonságos legyen, és csak az adott felhasználó tudja véglegesíteni a törlést.

Adatbázis

8. ábra: Adatbázis tervezői nézet



Az adatbázis a következő táblákat tartalmazza:

- **users**: A regisztrált felhasználók adatait tárolja. Mezők: user_ID (egyedi azonosító, auto increment), user_email (email cím), user_name (felhasználónév), user_pw (titkosított jelszó), user_state (felhasználó állapota), two_factor_enabled (2FA engedélyezve-e, 0 vagy 1), coins (érme egyenleg, alapértelmezetten 0).

- **todos**: A teendőket tárolja. Mezők: id (egyedi azonosító, auto increment), user_id (külkulcs a users táblára), title (teendő címe), date (dátum), start_time (kezdési idő), completed (teljesítve-e, 0 vagy 1), rewarded (érte kapott-e már jutalmat, 0 vagy 1).

- **skins**: A rendelkezésre álló skineket tárolja. Mezők: id (egyedi azonosító, auto increment), skin_name (skin neve), css_file (a skinhez tartozó CSS fájl neve), price (ár érmékben), listable (megjeleníthető-e a listában, 0 vagy 1).

- **selected_skins**: A felhasználók által kiválasztott skineket tárolja. Mezők: id (egyedi azonosító, auto increment), userID (külkulcs a users táblára), skinID (külkulcs a skins táblára), select_date (kiválasztás dátuma).

- **unlocked_skins**: A felhasználók által megvásárolt skineket tárolja. Mezők: id (egyedi azonosító, auto increment), userID (külkulcs a users táblára), skinID (külkulcs a skins táblára), unlock_date (feloldás dátuma).

- **logins**: A bejelentkezési eseményeket tárolja. Mezők: login_id (egyedi azonosító, auto increment), login_user_id (külkulcs a users táblára), login_date (bejelentkezés dátuma), action_date (utolsó aktivitás dátuma), logout_date (kijelentkezés dátuma), login_token (egyedi token), login_state (bejelentkezve-e, 0 vagy 1).

- **login_sessions**: Az aktív bejelentkezési munkameneteket tárolja, például a 2FA sessionöket. Mezők: id (egyedi azonosító, auto increment), user_id (külkulcs a users táblára), session_id (egyedi session azonosító), created_at (létrehozás dátuma), expires_at (lejárati idő).

- **password_resets**: A jelszó-visszaállítási kérelmeket tárolja. Mezők: email (felhasználó email címe), token (egyedi token), created_at (létrehozás dátuma).

- **trusted_devices**: A megbízható eszközöket tárolja. Mezők: id (egyedi azonosító, auto increment), user_id (külkulcs a users táblára), device_token (eszköz azonosító, az IP cím és böngésző user agent hash-e), created_at (létrehozás dátuma), expires_at (lejárati idő, alapértelmezetten 30 nap).

- **two_factor_codes**: A kétfaktoros hitelesítési kódokat tárolja. Mezők: id (egyedi azonosító, auto increment), user_id (külkulcs a users táblára), temp_code (egyszeri 6 jegyű kód), persistent_code (tartós 6 jegyű kód), created_at (létrehozás dátuma), expires_at (egyszeri kód lejárati ideje), persistent_expires_at (tartós kód lejárati ideje).
- **delete_requests**: A fiók törlési kérelmeket tárolja. Mezők: id (egyedi azonosító, auto increment), user_id (külkulcs a users táblára), token (egyedi token), created_at (létrehozás dátuma).

Az adatbázisban foreign key constraint-eket használtunk a táblák közötti kapcsolatok biztosítására, például a todos tábla user_id mezője a users tábla user_ID mezőjére hivatkozik. A constraint-ek biztosítják, hogy ha egy felhasználó törlésre kerül, akkor a hozzá kapcsolódó rekordok (pl. teendők, skinek) is automatikusan törlődjenek.

Testreszabás és profil

Profil

A profil oldalon (account.html) a felhasználók megtekinthetik és szerkeszthetik adataikat, például a nevüket (user_name). Valamint láthatják az érméik (coins) számát, amelyeket a teendők teljesítésével gyűjtöttek. Az oldalon lehetőség van a jelszó megváltoztatására is, amely során a rendszer ellenőrzi a jelszó erősségét: a jelszónak legalább 8 karakterből kell állnia, tartalmaznia kell kis- és nagybetűt, számot, valamint speciális karaktert. A profil szerkesztése után a módosítások mentésre kerülnek az adatbázisban a users táblában egy UPDATE SQL paranccsal.

Profil fejlesztői dokumentáció

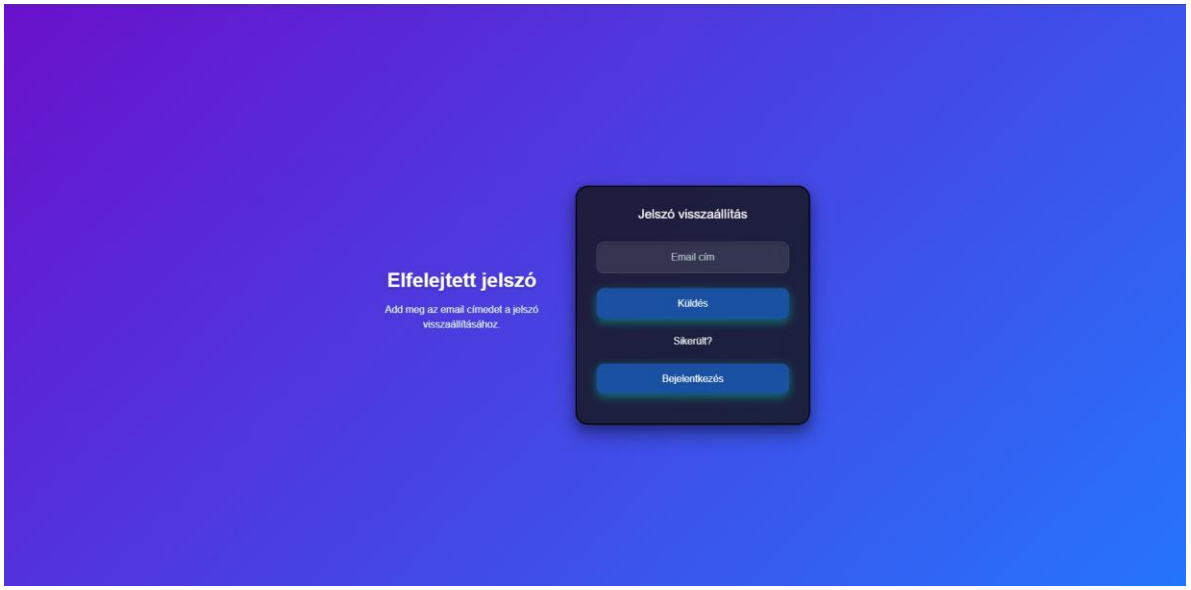
Frontend (account.html és account.js):

Az account.html oldalon egy űrlap található, amelyben a felhasználó adatai (név, email) szerkeszthetők. Az űrlap betöltésekor a fetchProfile függvény egy GET kérést küld a profile.php fájlnak, amely lekéri a felhasználó adatait a users táblából a user_ID alapján, és kitölti az űrlapot. A jelszó megváltoztatására egy külön gomb szolgál, amely egy modális ablakot nyit meg, ahol a felhasználó megadhatja az új jelszót és annak megerősítését. A changePassword függvény ellenőrzi a jelszó erősségét kliensoldalon, majd egy POST kérést küld a change_password.php fájlnak.

Backend (profile.php és change_password.php):

- **profile.php:** A GET kérésre JSON formátumban visszaküldi a felhasználó adatait a users táblából a user_ID alapján. A POST kérésre frissíti a users táblát a kapott adatokkal (pl. user_name, user_email) egy UPDATE SQL paranccsal.
- **change_password.php:** Ellenőrzi, hogy a felhasználó be van-e jelentkezve a logins tábla alapján. Ha igen, titkosítja az új jelszót a password_hash függvény segítségével, majd frissíti a users tábla user_pw mezőjét egy UPDATE SQL paranccsal.

9. ábra: Jelszó megváltoztatása



Jelszó-visszaállítás

Elfelejtett jelszó

Az elfelejtett jelszó funkció a forgot_password.html oldalon érhető el, amely a bejelentkezési felületről ("Elfelejtetted a jelszavad?" link) navigálható. A felhasználónak meg kell adnia az email címét, amelyre a rendszer egy jelszó-visszaállítási linket küld. A link egy egyedi tokent tartalmaz, amely a password_resets táblában tárolódik. A linkre kattintva a felhasználó egy új oldalon (reset_password.html) adhatja meg az új jelszavát.

Elfelejtett jelszó fejlesztői dokumentáció

Frontend (forgot_password.html és reset_password.html):

- A forgot_password.html oldalon egy űrlap található, ahol a felhasználó megadhatja az email címét. Az űrlap elküldésekor a requestPasswordReset függvény egy AJAX kérést küld a forgotpassword.php fájlnak a POST metódussal. A válasz alapján egy üzenet jelenik meg: sikeres kérés esetén egy zöld üzenet ("Jelszó-visszaállítási kérelem elküldve!"), sikertelen kérés esetén egy piros üzenet (pl. "Ez az email cím nem létezik!").
- A reset_password.html oldalon a felhasználó megadhatja az új jelszavát és annak megerősítését. A resetPassword függvény kliensoldalon ellenőrzi, hogy a jelszó megfelel-e a követelményeknek (legalább 8 karakter, kis- és nagybetű, szám, speciális karakter), majd egy AJAX kérést küld a resetpassword.php fájlnak a POST metódussal.

Backend (forgot_password.php és reset_password.php):

- forgotpassword.php: Ellenőrzi, hogy az email cím létezik-e a users táblában egy SELECT lekérdezéssel. Ha igen, generál egy egyedi tokent (pl. bin2hex(random_bytes(16))), amelyet a password_resets táblába ment a token mezőbe, az email címmel és a létrehozás időpontjával (created_at) együtt. Ezután egy emailt küld a felhasználónak a PHP Mailer segítségével, amely tartalmazza a jelszó-visszaállítási linket (pl. reset_password.html?token=...).

- resetpassword.php: Ellenőrzi, hogy a token érvényes-e a password_resets táblában egy SELECT lekérdezéssel. Ha a token érvényes, az új jelszót titkosítja a password_hash függvény segítségével, majd frissíti a users tábla user_pw mezőjét egy UPDATE SQL paranccsal. A token ezután törlésre kerül a password_resets táblából, hogy ne legyen újra felhasználható.**20. ábra: Jelszó-visszaállítási link**

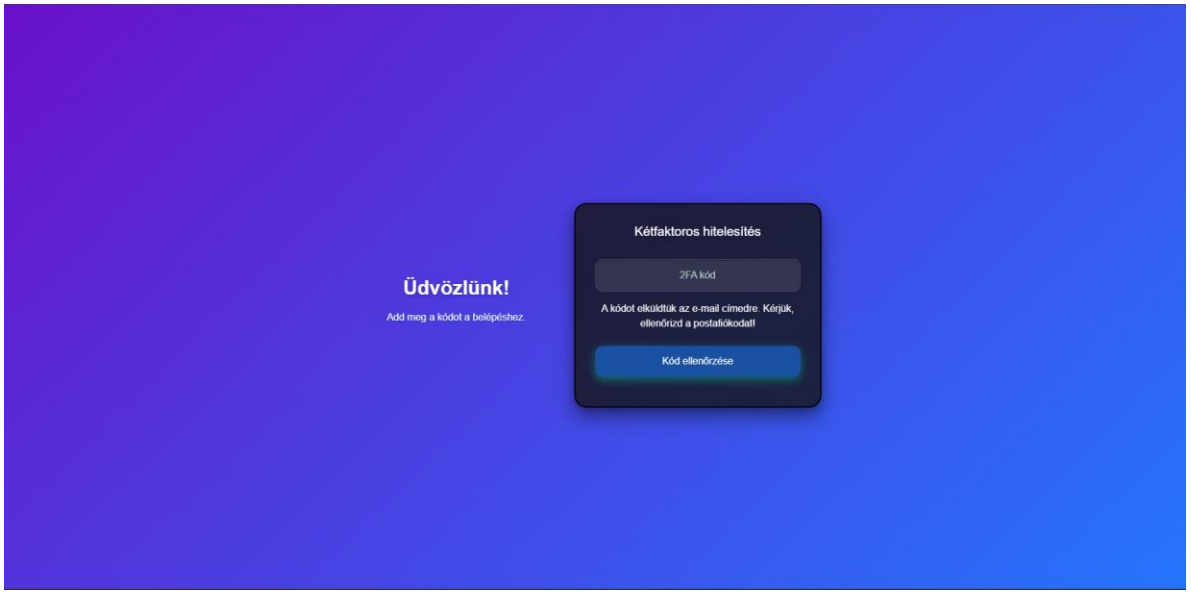
Kétfaktoros hitelesítés (2FA)

Kétfaktoros hitelesítés

A kétfaktoros hitelesítés (2FA) a bejelentkezés során aktiválható, ha a felhasználó engedélyezte ezt a funkciót a regisztráláskor. Bejelentkezéskor, miután a felhasználó megadta a felhasználónevét és jelszavát, a rendszer

két 6 jegyű kódot küld emailben: egy egyszeri kódot, amely 10 percig érvényes, és egy tartós kódot, amely 30 napig érvényes. A kódokat a felhasználónak a verify_2fa.html oldalon kell megadnia. Ha a kód helyes, a rendszer bejelentkezteti a felhasználót, és átirányítja a todo.html oldalra.

10. ábra: 2FA kód megadása



Kétfaktoros hitelesítés fejlesztői dokumentáció

Frontend (verify_2fa.html és verify_2fa.js):

A verify_2fa.html oldalon egy űrlap található, ahol a felhasználó megadhatja a kapott 6 jegyű kódot. Az űrlap elküldésekor a verify_2fa.js fájlban definiált verifyCode függvény egy AJAX kérést küld a login.php fájlnak a POST metódussal, amely a kódot és a session ID-t tartalmazza. Ha a kód helyes, a rendszer bejelentkezteti a felhasználót, és a localStorage-ba menti a kapott tokent. Ezután a felhasználó átirányításra kerül a todo.html oldalra. Ha a kód helytelen, egy piros üzenet jelenik meg ("Érvénytelen kód!").

Backend (login.php):

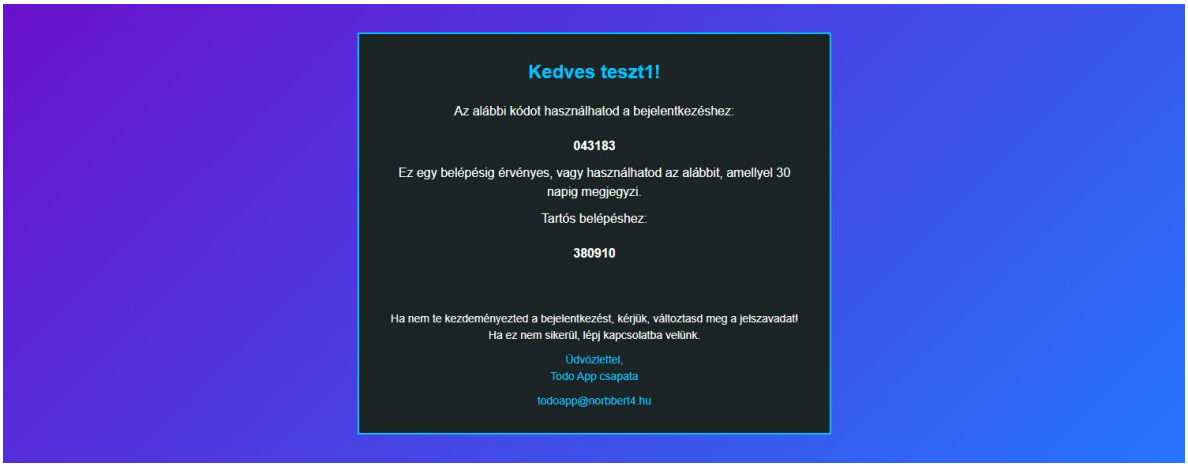
A login.php fájl a 2FA-val kapcsolatos logikát a következő módon kezeli:

- Kód generálás: Ha a felhasználó engedélyezte a 2FA-t, a rendszer két 6 jegyű kódot generál (egyszeri és tartós), amelyeket a two_factor_codes táblába ment a temp_code, persistent_code, expires_at (10 perc) és persistent_expires_at (30 nap) mezőkbe.

- Email küldés: A kódokat egy emailben elküldi a felhasználónak a PHP Mailer segítségével. Az email HTML formátumban van, és tartalmazza mindkét kódot.

- Kód ellenőrzés: Amikor a felhasználó megadja a kódot a verify_2fa.html oldalon, a login.php ellenőrzi, hogy a kód létezik-e a two_factor_codes táblában, és hogy még nem járt-e le (expires_at vagy persistent_expires_at alapján). Ha az egyszeri kódot használta, a rendszer bejelentkezteti a felhasználót, és az expires_at mezőt NOW()-ra állítja, hogy a kód ne legyen újra felhasználható. Ha a tartós kódot használta, a rendszer a felhasználó eszközét megbízhatóként jelöli meg a trusted_devices táblában, és 30 napig nem kér 2FA kódot az adott eszközről történő bejelentkezéskor.

11. ábra: 2FA kód emailben



Megbízható eszközök

A megbízható eszközök funkció lehetővé teszi, hogy a felhasználók bizonyos eszközöket (pl. böngészőket) megbízhatóként jelöljenek meg, így ezekről történő bejelentkezéskor nem kell 2FA kódot megadniuk. A funkció úgy érhető el, hogy a 2FA bejelentkezéskor a felhasználó a 30 napos kódot adja meg. Ebben az esetben a rendszer az eszköz azonosítóját (az IP cím és a böngésző user agent hash-ét) elmenti a trusted_devices táblába, és 30 napig érvényesíti. A megbízható eszközök listája a profil oldalon megtekinthető, és a felhasználó törölheti őket, ha szükséges.

2FA hibajavítás és optimalizálás

A fejlesztés során egy jelentős hibát észleltünk a 2FA működésében: ha a felhasználó az egyszeri kódot használta a bejelentkezéshez, majd kijelentkezett, a 30 napos kód nem volt újra használható, mert a rendszer nem generált új kódokat. Ez a probléma abból adódott, hogy

a login.php fájl nem törölte a régi kódokat a two_factor_codes táblából, így a kliensoldali logika nem tudta megfelelően kezelni a 30 napos kódot.

Hibajavítás lépései:

1. Régi kódok törlése: Módosítottuk a login.php fájlt, hogy minden új 2FA kérés esetén törölje a felhasználóhoz tartozó régi kódokat a two_factor_codes táblából egy DELETE SQL paranccsal, mielőtt új kódokat generálna. Ez biztosítja, hogy a 30 napos kód mindig friss legyen, és kijelentkezés után is működjön.

2. Új kódok generálása: Az új kódok generálása során biztosítjuk, hogy az expires_at és persistent_expires_at mezők megfelelően legyenek beállítva (10 perc, illetve 30 nap).

3. Kliensoldali logika javítása: A verify_2fa.js fájlban javítottuk a hibakezelést, hogy a felhasználó egyértelmű visszajelzést kapjon, ha a kód érvénytelen vagy lejárt. Ehhez try-catch blokkokat adtunk a fetch kérés köré, és részletesebb hibaüzeneteket jelenítünk meg.

Optimalizálás:

- Email sablon javítása: Az email sablont átláthatóbbá és reszponzívvá tettük, hogy mobil eszközökön is jól olvasható legyen. A HTML sablonban CSS stílusokat használtunk a szebb megjelenés érdekében.

- Adatbázis lekérdezések optimalizálása: A two_factor_codes táblában lévő lekérdezéseket indexekkel optimalizáltuk, például a user_id mezőre indexet hoztunk létre, hogy a lekérdezések gyorsabbak legyenek.

- Session kezelés: A login_sessions tábla expires_at mezőjét szigorúbban kezeljük, hogy a lejárt sessionök automatikusan törlődjenek egy cron job segítségével, így csökkentve az adatbázis terhelését.

Ez a hibajavítás biztosította, hogy a 2FA funkció megbízhatóan működjön, és a felhasználók zökkenőmentesen használhassák mind az egyszeri, mind a tartós kódot.

Fejlesztési folyamat és tesztelés

Fejlesztési folyamat

A Todo App fejlesztése agilis módszertan szerint zajlott, amely lehetővé tette a rugalmas iterációkat és a folyamatos visszajelzések beépítését. A projektet a következő főbb fázisokra bontottuk:

1. **Tervezés (1-2. hét)**:

- A projekt ötletelésével kezdtünk, amelynek során meghatároztuk az alkalmazás főbb funkcióit (regisztráció, bejelentkezés, teendők kezelése, skinek, 2FA).
- Elkészítettük az adatbázis vázlatát (ER diagram), és definiáltuk a táblák közötti kapcsolatokat.
- A felhasználói felületet (UI) papíron vázoltuk fel.

2. **Fejlesztés (3-8. hét)**:

- **Backend fejlesztés**: Először a szerveroldali logikát készítettük el (PHP fájlok az api mappában), amely az adatbázis műveleteket kezeli. Kezdtük a registration.php és login.php fájlokkal, majd fokozatosan bővítettük a többi végponttal (pl. todos, skins).
- **Frontend fejlesztés**: Párhuzamosan dolgoztunk a frontend oldalon, az index.html, registration.html és todo.html oldalakkal kezdve. A JavaScript fájlokban (pl. todo.js) AJAX kéréseket használtunk az API-val való kommunikációhoz.
- **Biztonsági funkciók**: A 2FA és a jelszó-visszaállítás implementálása a 6. héten történt, miután az alapvető funkcionalitás stabilan működött.
- **Skinek és testreszabás**: A skinek kezelését a 7. héten valósítottuk meg, amely során különböző CSS fájlokat készítettünk a skinekhez (pl. default.css, dark.css).

3. **Hibajavítás és optimalizálás (9-10. hét)**:

- A 2FA-val kapcsolatos hibát a 9. héten azonosítottuk és javítottuk.
- Optimalizáltuk az adatbázis lekérdezéseket és a kliensoldali logikát, például a todo.js fájlban a teendők állapotváltozásának kezelését.
- A felhasználói felületet finomhangoltuk, például a regisztrációs oldalon a 2FA jelölőnégyzet mellett lévő szöveg láthatóságát javítottuk a CSS módosításával.

4. **Dokumentáció és véglegesítés (11-12. hét)**:

- Elkészítettük a vizsgaremek dokumentációját, amely tartalmazza az alkalmazás bemutatását, a felhasználói felület leírását, a fejlesztői dokumentációt, valamint a tesztelési eredményeket.
- Végső teszteket végeztünk az alkalmazáson, hogy biztosítsuk a hibamentes működést.

A fejlesztés során a GitHub-ot használtuk verziókezelésre, a GitHub Desktop segítségével kezeltük a commitokat és a branch-eket. A csapatmunkát a Discord segítségével koordináltuk, ahol napi szinten egyeztettünk a feladatokról és a problémákról.

Tesztelés

A Todo App tesztelése több szinten zajlott, hogy biztosítsuk a rendszer stabilitását és helyes működését.

Egységtesztek (Unit Tests)

Az egységtesztek célja az volt, hogy a rendszer egyes komponenseit külön-külön ellenőrizzük, biztosítva, hogy azok önállóan helyesen működjenek. A backend és frontend kódok tesztelésére különböző eszközöket és módszereket használtunk.

- Backend tesztek: A backend logika tesztelésére a PHPUnit keretrendszert használtuk, amely lehetővé tette a PHP kódok automatizált tesztelését. A tesztek során különös figyelmet fordítottunk az autentikációs, adatbázis-műveleti és biztonsági funkciókra. Néhány konkrét tesztpélda:

    - Bejelentkezés (login.php): Teszteltük, hogy a login.php helyesen kezeli-e a helyes és helytelen felhasználónév-jelszó párosokat. Például egy helytelen jelszóval történő bejelentkezés során ellenőriztük, hogy a válasz JSON objektum a

megfelelő hibaüzenetet tartalmazza-e: {"success": false, "error": "Hibás felhasználónév vagy jelszó"}. Egy másik teszt során azt vizsgáltuk, hogy a rendszer helyesen generál-e tokent, és azt a logins táblába menti-e, ha a bejelentkezés sikeres.

- 2FA kódok kezelése: A login.php fájlban teszteltük, hogy a kétfaktoros hitelesítési kódok helyesen generálódnak-e (6 jegyű számok), és hogy az expires_at és persistent_expires_at mezők megfelelően vannak-e beállítva (10 perc, illetve 30 nap). Egy külön tesztben ellenőriztük, hogy a lejárt kódok nem használhatók fel: például egy 10 perc után lejárt temp_code esetén a rendszer elutasítja a bejelentkezést, és a megfelelő hibaüzenetet küldi vissza: {"success": false, "error": "Érvénytelen 2FA kód vagy lejárt"}.
- Skin vásárlás (skins.php): A skins.php fájlban teszteltük, hogy a skin vásárlása során a tranzakció helyesen zajlik-e le. Például ellenőriztük, hogy ha a felhasználónak nincs elég érméje, a vásárlás elutasításra kerül, és a users tábla coins mezője nem változik. Egy másik tesztben azt vizsgáltuk, hogy sikeres vásárlás esetén az érmék levonása és az unlocked_skins táblába történő beszúrás egy tranzakcióban történik-e, így elkerülve az adatbázis inkonzisztenciát.

- Frontend tesztek: A JavaScript kódok tesztelésére kezdetben manuális teszteket végeztünk, majd később a Jest keretrendszert használtuk az AJAX kérések és a kliensoldali logika automatizált tesztelésére. Néhány konkrét tesztpélda:

  - Teendők mentése (todo.js): A todo.js fájlban teszteltük, hogy a saveTodo függvény helyesen küldi-e a POST kérést az API-nak, és a válasz alapján megfelelően frissíti-e a teendők táblázatot a todo.html oldalon. Egy mock API-t használtunk, amely JSON választ szimulált, így ellenőrizhettük, hogy a függvény helyesen kezeli-e a sikeres és sikertelen válaszokat. Például egy sikertelen válasz esetén (pl. {"success": false, "error": "Érvénytelen token"}) a rendszer egy piros üzenetet jelenít meg: "Hiba történt a teendő mentése során!".
  - 2FA kód ellenőrzés (verify_2fa.js): A verify_2fa.js fájlban teszteltük, hogy a verifyCode függvény helyesen kezeli-e a különböző válaszokat a login.php-tól. Egy mock API-val szimuláltunk helyes és helytelen kódokat, valamint hálózati hibákat. Például egy helytelen kód esetén a függvény egy piros üzenetet jelenít meg: "Érvénytelen kód!", míg hálózati hiba esetén: "Hiba történt a kód ellenőrzése során: Network Error".

Integrációs tesztek

Az integrációs tesztek során a rendszer különböző komponenseinek együttműködését vizsgáltuk, például a frontend és backend közötti kommunikációt, valamint az adatbázis-műveletek helyes működését.

- **Regisztráció és bejelentkezés integráció**: Teszteltük, hogy egy újonnan regisztrált felhasználó sikeresen be tud-e jelentkezni. A teszt során először a registration.php fájl segítségével regisztráltunk egy új felhasználót, majd a login.php fájl segítségével bejelentkeztünk. Ellenőriztük, hogy a users táblában helyesen tárolódnak-e az adatok (pl. titkosított jelszó), és hogy a logins táblában megjelenik-e a bejelentkezéshez generált token.

- **Teendők kezelése és érmék**: Egy integrációs teszt során vizsgáltuk, hogy egy teendő kipipálása után a felhasználó helyesen kapja-e meg az érméket. A teszt során először egy teendőt hoztunk létre a todos táblában, majd kipipáltuk a todo.js segítségével egy PATCH kéréssel. Ellenőriztük, hogy a todos tábla completed és rewarded mezői 1-re frissülnek-e, valamint hogy a users tábla coins mezője megfelelően növekedik-e (+1 érme teendőnként).

- **2FA és megbízható eszközök**: Teszteltük, hogy a 2FA helyesen működik-e egy nem megbízható eszközről történő bejelentkezéskor. A teszt során először bejelentkeztünk egy olyan felhasználóval, akinek engedélyezve van a 2FA, majd a tartós kódot használtuk a bejelentkezéshez. Ellenőriztük, hogy a trusted_devices táblában megjelenik-e az eszköz, és hogy a következő bejelentkezés során a rendszer nem kér-e 2FA kódot.

Felhasználói tesztek (UI/UX tesztek)

A felhasználói felület tesztelésére valós felhasználókat vontunk be, akik különböző forgatókönyveket próbáltak ki az alkalmazásban. A tesztek során a következő szempontokra fókuszáltunk:

- **Bejelentkezés és regisztráció**: A felhasználók visszajelzése alapján a bejelentkezési és regisztrációs felület intuitív volt, de néhányan jelezték, hogy a 2FA jelölőnégyzet melletti szöveg nem látható megfelelően a regisztrációs oldalon. Ezt a hibát a CSS módosításával javítottuk.

- **Teendők kezelése**: A felhasználók pozitívan értékelték a teendők táblázatos megjelenítését és a műveleti gombokat (kipipálás, törlés), de néhányan jelezték, hogy a "Nem pipálható ki …-ig!" üzenet túl kicsi betűmérettel jelenik meg. Ezt a problémát a save-message div betűméretének növelésével oldottuk meg a style.css fájlban.
- **Skinek és testreszabás**: A skinek kiválasztása és alkalmazása zökkenőmentesen működött, de a felhasználók javasolták, hogy a skinek előnézete legyen látható a vásárlás előtt. Ezt a funkciót a jövőbeli fejlesztések közé felvettük.

Teljesítménytesztek:

A teljesítménytesztek során az alkalmazás válaszidejét és az adatbázis terhelését vizsgáltuk. Például teszteltük, hogy a todo.html oldal betöltése és a teendők listázása mennyi időt vesz igénybe 1000 teendő esetén. Az eredmények alapján optimalizáltuk a todos tábla lekérdezéseit: indexet hoztunk létre a user_id mezőre, és a lekérdezésekben korlátoztuk a visszaadott rekordok számát (LIMIT 100) egy lapozási rendszer bevezetésével.

Biztonsági tesztek

A biztonsági tesztek során az alábbi forgatókönyveket vizsgáltuk:

- **SQL injekció:** Próbáltunk SQL injekcióval hozzáférni az adatbázishoz, például a bejelentkezési űrlapon keresztül. A real_escape_string használata és a paraméterezett lekérdezések (prepared statements) biztosították, hogy az injekciók ne legyenek sikeresek.

- **Token hamisítás:** Teszteltük, hogy egy hamis tokennel történő API kérés elutasításra kerül-e. A logins tábla alapján a rendszer minden kérésnél ellenőrzi a token érvényességét, így a hamis tokenek nem férhetnek hozzá a felhasználó adataihoz.

- **2FA kód újrafelhasználás:** Ellenőriztük, hogy egy egyszeri 2FA kód újrafelhasználható-e a lejárati időn belül. A hibajavítás után (lásd a fejezetet) a kódok helyesen törlődnek a használat után, így az újrafelhasználás nem lehetséges.

Források és segédeszközök

Források

Az alkalmazás fejlesztése során a következő forrásokat használtuk fel:

- Bootstrap dokumentáció: A Bootstrap keretrendszer dokumentációja segített a reszponzív és esztétikus felhasználói felület kialakításában. Forrás: https://getbootstrap.com/docs/.
- jQuery dokumentáció: A jQuery-t az AJAX kérések kezelésére és a DOM manipulációra használtuk. A dokumentáció segített a helyes szintaxis és legjobb gyakorlatok alkalmazásában. Forrás: https://api.jquery.com/.
- PHP dokumentáció: A PHP hivatalos dokumentációja alapvető forrás volt a szerveroldali logika fejlesztéséhez, például a password_hash és password_verify függvények helyes használatához. Forrás: https://www.php.net/manual/en/.
- MySQL dokumentáció: Az adatbázis-műveletekhez (pl. indexek, tranzakciók) a MySQL dokumentációját használtuk. Forrás: https://dev.mysql.com/doc/.

Segédeszközök

A fejlesztés során a következő segédeszközöket használtuk:

- PHP Mailer: Az email küldéshez a PHP Mailer könyvtárat használtuk, amely biztosítja az SMTP alapú email küldést. Forrás: https://github.com/PHPMailer/PHPMailer.
- PHPUnit: A backend egységtesztek futtatásához a PHPUnit keretrendszert használtuk, amely lehetővé tette a PHP kódok automatizált tesztelését. Forrás: https://phpunit.de/index.html.
- Jest: A JavaScript kódok egységteszteléséhez a Jest keretrendszert használtuk, amely különösen hasznos volt az AJAX kérések mockolására. Forrás: https://jestjs.io/.

Használt programok

- PHP: A szerveroldali logika fejlesztéséhez.
- HTML: A weboldalak struktúrájának kialakításához.
- JavaScript: A kliensoldali logika és interaktivitás biztosításához.
- CSS: A felhasználói felület stílusozásához.
- Bootstrap: Reszponzív és esztétikus dizájn létrehozásához.
- jQuery: AJAX kérések és DOM manipuláció kezelésére.
- AJAX: Aszinkron kommunikáció a frontend és backend között.
- XAMPP: Helyi fejlesztőkörnyezet biztosításához (Apache és MySQL szerver).
- GitHub: Verziókezeléshez és a csapatmunkához.
- Visual Studio Code: Forráskód szerkesztéséhez.
- Github Desktop: A GitHub repository kezelésére és commitok készítésére.