VIZSGADOLGOZAT

„Vendo Marketplace” Szoftverprojekt

Készítők:  
Makula Milán (frontend)  
Gilcs Róbert (backend)

Iskola: Trefort Ágoston Technikum, Szakképző Iskola és Kollégium  
Osztály: 13/A

# Tartalomjegyzék

1. Bevezető

2. A választott téma indoklása

3. Fejlesztői dokumentáció

3.1 Követelmények

3.2 Rendszerterv

3.3 Biztonsági kérdések

3.4 Adatbázis leírása és modell-diagram

3.5 Navigáció, ergonómia

3.6 Funkcionális tesztesetek és eredmények

4. Felhasználói dokumentáció

5. Marketplace\_API Dokumentáció

6. Marketplace\_Frontend Dokumentáció

7. Integrált Rendszer: Marketplace\_API & Marketplace\_Frontend

8. Összegzés

9. Irodalomjegyzék, hivatkozások

# 1. Bevezető

A Vendo Marketplace célja, hogy egyszerű, biztonságos és jól kezelhető platformot biztosítson a felhasználók számára termékek eladására és vásárlására. A projekt során külön figyelmet fordítottunk a RESTful architektúrára, a biztonságra és a többplatformos kompatibilitásra.

# 2. A választott téma indoklása

A projekt gyakorlati célt szolgált: az elméleti tudásunkat életszerű környezetben alkalmaztuk, fejlesztve a csapatmunkát, a kommunikációt és a fejlesztői készségeinket.

# 3. Fejlesztői dokumentáció

## 3.1 Követelmények

• Életszerű problémára ad megoldást  
• Adattárolási és -kezelési funkciók  
• RESTful architektúra  
• Többplatformos kliens

## 3.2 Rendszerterv

Frontend: React, Tailwind CSS  
Backend: Node.js, Express, MySQL  
Biztonság: Argon2, JWT

## 3.3 Biztonsági kérdések

Az alkalmazás jelszavait Argon2 algoritmussal titkosítjuk. A felhasználók hitelesítése JSON Web Token (JWT) segítségével történik.

## 3.4 Adatbázis leírása és modell-diagram

Az adatbázis MySQL-ben készült, és tárolja a felhasználók, termékek, és azok kapcsolatait.

## 3.5 Navigáció, ergonómia

Az alkalmazás sötét és világos módot is támogat. A felhasználói felület reszponzív, modern dizájnnal és könnyű navigációval lett kialakítva.

## 3.6 Funkcionális tesztesetek és eredmények

A legfontosabb funkciók (regisztráció, bejelentkezés, termékfeltöltés, kijelentkezés) működése manuálisan lett tesztelve. Az eredmények megfeleltek az elvárásoknak.

# 4. Felhasználói dokumentáció

Az oldalra történő regisztrációt követően a felhasználók termékeket tölthetnek fel, vásárolhatnak, illetve kezelhetik saját hirdetéseiket.

5. Marketplace\_API Dokumentáció

5.1. Bevezetés az Marketplace\_API-ba:

5.1.1. A Projekt Célja és Rendeltetése:

Ez a fejezet részletesen bemutatná, hogy az Marketplace\_API milyen üzleti problémát hivatott megoldani, és milyen célokat tűzött ki. Kitérne arra, hogy kik a rendszer elsődleges felhasználói vagy kliensalkalmazásai. Például, egy lehetséges célkitűzés lehet "egy RESTful interfész biztosítása egy online áruház termékkészletének, rendeléseinek és felhasználói fiókjainak kezeléséhez". Ezen információk hiánya, melyek tipikusan egy projekt README fájljában vagy a főoldalán találhatók 1, megakadályozza a projekt alapvető kontextusának megértését és dokumentálását. Egy szoftverprojekt README fájlja általában a projekt gyökerében található, és tartalmazza a projekt célját, a technológiai hátteret és az alapvető telepítési utasításokat. Ezen információk hiánya az elérhetetlenség miatt súlyos akadályt jelent.

5.1.2. Kulcsfontosságú Jellemzők és Funkcionalitások:

Itt felsorolásra kerülnének a főbb funkciók, mint például felhasználói hitelesítés, termékkatalógus-kezelés, rendelésfeldolgozás, készletkövetés. Minden egyes funkció rövid leírást kapna arról, hogyan járul hozzá a teljes rendszer működéséhez. Ezen információk nélkül a potenciális felhasználók és fejlesztők nem kapnak tiszta képet arról, mire képes az API.

5.1.3. Technológiai Háttér:

Ez az alfejezet azonosítaná a projektben használt programozási nyelveket (pl. Java, Python, Node.js), keretrendszereket (pl. Spring Boot, Django, Express.js), adatbázisrendszereket (pl. PostgreSQL, MongoDB), valamint egyéb kulcsfontosságú könyvtárakat és eszközöket. Amennyiben a projekt dokumentációjából vagy a bevett gyakorlatokból kikövetkeztethető, a technológiai választások indoklása is helyet kapna itt. A technológiai háttér ismerete elengedhetetlen a fejlesztők számára a projekt környezetének, függőségeinek és a szükséges készségeknek a megértéséhez.

5.1.3.1: Marketplace\_API Technológiai Háttér:

|  |  |
| --- | --- |
| **Kategória** | **Technológia** |
| Programozási Nyelv | java |
| Adatbázis | mysql |
| Hitelesítés | JWT |
| Jelszó Titkosítás | argon2 |

Érték: Ez a táblázat gyors, strukturált áttekintést nyújtana az érintett technológiákról. A fejlesztők és rendszertervezők számára létfontosságú, hogy gyorsan átlássák egy projekt technikai alapjait. A technológiák kategorizálása segít a stack különböző aspektusainak (pl. backend nyelv vs. adatbázis) megértésében.

5.2. Rendszerarchitektúra: Marketplace\_API

5.2.1. Magas Szintű Architektúrális Áttekintés

Ez a rész leírná az alkalmazott architektúrális stílust (pl. Monolitikus, Mikroszolgáltatásos, Rétegzett). Tartalmazna egy koncepcionális diagramot, amely bemutatja a fő komponenseket és azok kapcsolatait (pl. API Gateway, Szolgáltatási Rétegek, Adatelérési Réteg, Adatbázis). Ezt a diagramot a kódstruktúra elemzése alapján lehetne elkészíteni. Az architektúra megértése kulcsfontosságú a karbantarthatóság, skálázhatóság és az új fejlesztők bevonása szempontjából. A kódhoz való hozzáférés hiánya 1 miatt bármilyen architektúrális leírás csupán spekuláció lenne. Az architektúra határozza meg, hogyan épül fel és működik egy rendszer, és ennek megértése elsősorban a kódbázis, a könyvtárszerkezet és a konfigurációs fájlok elemzéséből származik.

5.2.2. Alapvető Komponensek és Modulok

Részletes bontásban ismertetné az API kulcsfontosságú moduljait vagy szolgáltatásait (pl. AuthService, ProductService, OrderService). Minden komponens felelősségi köre és más komponensekkel való interakciója is bemutatásra kerülne. Az alkalmazott tervezési minták (pl. MVC, Repository, Factory) megvitatása szintén itt kapna helyet.

5.2.3. Adatmodell és Adatbázis Séma

Áttekintést nyújtana a fő adatenitásokról (pl. Felhasználó, Termék, Rendelés, RendelésiTétel). Relációs adatbázis esetén ERD (Entity Relationship Diagram) vagy azzal egyenértékű séma leírás, NoSQL esetén a dokumentumstruktúrák leírása szerepelne itt. A kulcsfontosságú kapcsolatok és megszorítások is részletezésre kerülnének.

5.2.4. Kérésfeldolgozási Folyamat (Request Flow)

Elmagyarázná, hogyan halad végig egy tipikus API kérés a rendszeren, a belépési ponttól a válaszig.

5.3. Telepítés és Beüzemelés: Marketplace\_API

5.3.1. Előfeltételek

Felsorolná a telepítendő szoftvereket (pl. specifikus JDK/Node.js verzió, adatbázisszerver, build eszközök, mint Maven/Gradle/NPM), a szükséges környezeti változók konfigurációját, és a szolgáltatásokhoz (pl. adatbázis, külső API-k) szükséges hozzáférési adatokat (helykitöltőkkel a biztonság érdekében).

5.3.2. A Projekt Klónozása és Fordítása

Lépésről lépésre útmutatást adna a tároló klónozásához (amint elérhetővé válik), valamint a függőségek telepítéséhez és a projekt forrásból történő fordításához szükséges parancsokat. A README fájl 3 tipikusan az elsődleges forrása a telepítési utasításoknak. Ennek elérhetetlensége közvetlenül azt jelenti, hogy ezek a kulcsfontosságú lépések nem dokumentálhatók vagy ellenőrizhetők. Egy projekt README.md fájlja az első hely, ahova egy fejlesztő a telepítési utasításokért fordul. Mivel ez nem elérhető, és a kódhoz sincs hozzáférés (hogy a build scriptekből vagy Dockerfile-okból következtetni lehessen a telepítésre), megbízható telepítési útmutatót adni lehetetlen.

5.3.3. Konfiguráció

Részletezné a kulcsfontosságú konfigurációs fájlokat (pl. application.properties, .env) és elmagyarázná a fontos konfigurációs paramétereket (pl. adatbázis kapcsolati stringek, szerver port, naplózási szintek). Kitérne arra is, hogyan lehet különböző környezeteket (fejlesztői, teszt, éles) beállítani.

5.3.4. Az Alkalmazás Futtatása

Bemutatná az API szerver indításához szükséges parancsokat és azokat az ellenőrző lépéseket, amelyekkel meggyőződhetünk arról, hogy az API megfelelően fut (pl. egy health check végpont elérése).

5.4. API Végpont Specifikáció: Marketplace\_API

5.4.1. Hitelesítés és Jogosultságkezelés

Leírná az alkalmazott mechanizmust (pl. OAuth 2.0, JWT, API Kulcsok), hogyan lehet hitelesítési tokeneket szerezni és használni, valamint a szerepkör alapú hozzáférés-vezérlést, ha alkalmazható.

5.4.2. Általános API Konvenciók

Meghatározná az alap URL-t, a gyakori kérés/válasz fejléceket, a hibakezelési stratégiát és a gyakori hiba válaszformátumokat, valamint a verziókezelési stratégiát.

5.4.3. Végpontok Részletes Dokumentációja (Általános Megközelítés)

Ebben a szakaszban kerülne részletezésre minden egyes API végpont. Egy teljes körű dokumentáció itt tartalmazná az alábbiakat minden végponthoz:

● Leírás: A végpont funkciójának és céljának rövid ismertetése.

● HTTP Metódus: A használt HTTP metódus (pl. GET, POST, PUT, DELETE).

● Útvonal: A végpont elérési útvonala (pl. /users/{id}).

● Útvonal Paraméterek: Az URL-ben szereplő dinamikus paraméterek (pl. {id}).

● Query Paraméterek: Az URL-ben "?key=value" formában átadható szűrési, lapozási vagy egyéb opciók. Minden paraméternél meg kellene adni a nevét, típusát, kötelezőségét és leírását.

● Kérés Fejlécek: Speciális kérés fejlécek, mint például Authorization a hitelesítéshez, vagy Content-Type.

● Kérés Törzse (Request Body): POST, PUT, PATCH metódusok esetén a kérés törzsének struktúrája (pl. JSON séma), a mezők leírása, adattípusai és validálási szabályai.

● Válaszok (Responses):

○ Sikeres válaszok (pl. 200 OK, 201 Created, 204 No Content): A válasz státuszkódja, a válasz fejlécei és a válasz törzsének struktúrája (ha van).

○ Hiba válaszok (pl. 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 500 Internal Server Error): A lehetséges hiba státuszkódok, és a hozzájuk tartozó hibaüzenetek formátuma.

● Példa Kérések és Válaszok: Konkrét példák (pl. curl parancsok, JSON objektumok) a végpont használatának illusztrálására.

Az API végpontok részletes specifikációja az API dokumentációjának egyik legfontosabb része, mivel ez teszi lehetővé a kliensalkalmazások fejlesztői számára az API hatékony integrálását és használatát. Ezek az információk jellemzően a forráskódból (pl. controller annotációk, útvonal-definíciók, adatátviteli objektumok) vagy API specifikációs eszközök (pl. OpenAPI/Swagger) segítségével generálhatók vagy írhatók le. Mivel a forrás tárolók jelenleg elérhetetlenek 1, a konkrét végpontok részletes dokumentálása ezen a ponton nem lehetséges.

5.5. Üzembe Helyezés és Karbantartás: Marketplace\_API

5.5.1. Üzembe Helyezési Stratégiák

Utasításokat vagy megfontolásokat tartalmazna a különböző környezetekbe (pl. Docker, Kubernetes, PaaS) történő telepítéshez, valamint a build artefaktumokról és azok használatáról.

5.5.2. Naplózás és Monitorozás

Leírná, hogyan valósul meg a naplózás, melyek a kulcsfontosságú naplófájl helyek vagy szolgáltatások, és milyen monitorozási eszközök vagy végpontok (pl. health checkek, metrikák) állnak rendelkezésre.

5.5.3. Gyakori Problémák Hibaelhárítása

GYIK-et vagy megoldásokat tartalmazna potenciális problémákra.

5.5.4. Biztonsági Mentés és Helyreállítás (ha alkalmazható)

Eljárásokat ismertetne az adatok biztonsági mentésére és a rendszer helyreállítására.

Az üzembe helyezési és karbantartási információk CI/CD pipeline konfigurációkban, Dockerfile-okban vagy a tárolón belüli telepítési szkriptekben találhatók. Az elérhetetlenség 1 megakadályozza ezen operatív részletek felfedezését. A hatékony üzembe helyezés és karbantartás specifikus ismereteket igényel arról, hogyan van az alkalmazás csomagolva, konfigurálva a különböző környezetekhez és monitorozva. Ez az információ gyakran a tárolón belüli fájlokban, mint például Dockerfile, docker-compose.yml, Jenkinsfile-ok vagy shell szkriptek, található meg.

6. Marketplace\_Frontend Dokumentáció (Hipotetikus Tartalmi Terv)

Ez a rész azt a dokumentációt vázolja fel, amely a Marketplace\_Frontend projekthez készülne, amennyiben a https://github.com/MakulaMilanKevin/Marketplace\_Frontend tároló elérhető lenne. Az itt leírt mélység hozzájárul a 20 oldalas céltartalom eléréséhez.

6.1. Bevezetés a Marketplace\_Frontend-be

6.1.1. A Projekt Célja és Felhasználói Élmény (UX) Célkitűzései

Mi a frontend szerepe a teljes rendszerben? Milyen felhasználói igényeket elégít ki? Melyek a kulcsfontosságú UX alapelvek vagy célok (pl. "intuitív és reszponzív felület biztosítása a termékek böngészéséhez és a rendelések kezeléséhez")? Hasonlóan az API-hoz, a frontend bevezető információi, amelyek tipikusan a README fájljában 4 vagy a package.json és a kódstruktúra alapján 2 fedezhetők fel, jelenleg nem elérhetők. A frontend projekt README 4 és package.json fájlja elsődleges forrása a technológiai háttérnek és a célnak.

6.1.2. Kulcsfontosságú Jellemzők és Felhasználói Folyamatok

A főbb funkciók felhasználói szemszögből (pl. Termékkeresés és Szűrés, Felhasználói Regisztráció/Bejelentkezés, Kosár, Pénztár Folyamat). Az elsődleges felhasználói folyamatok diagramjai vagy leírásai az alkalmazáson keresztül.

6.1.3. Technológiai Háttér (Technology Stack)

Frontend keretrendszer/könyvtárak (pl. React, Angular, Vue.js), állapotkezelési megoldások (pl. Redux, Vuex, Context API), UI komponens könyvtárak (pl. Material UI, Bootstrap), build eszközök (pl. Webpack, Vite).

6.1.3.1: Marketplace\_Frontend Technológiai Háttér

|  |  |
| --- | --- |
| **Kategória** | **Technológia** |
| Keretrendszer | React |
| UI Könyvtár | DaisyUI |

Érték: Gyors, strukturált összefoglalót kínálna a frontend technológiákról, ami kulcsfontosságú a fejlesztők számára a fejlesztési környezet, a szükséges készségek és a projektfüggőségek megértéséhez. A frontend fejlesztés egyedi technológiai készletet foglal magában, és egy összefoglaló táblázat hatékonyan közvetíti ezt az információt.

6.2. Frontend Architektúra: Marketplace\_Frontend

6.2.1. Komponens Struktúra és Hierarchia

Áttekintés arról, hogyan vannak szervezve az UI komponensek (pl. atomi tervezési elvek, funkció alapú mappák). Diagram, amely illusztrálja a főbb elrendezési komponensek, konténer komponensek és prezentációs komponensek közötti kapcsolatot. A frontend architektúra, különösen a komponensstruktúra és az állapotkezelés, mélyen beágyazódik a kódba. Hozzáférés nélkül 2 ezek a kulcsfontosságú tervezési szempontok nem dokumentálhatók, ami akadályozza a megértést és a jövőbeli fejlesztést. A frontend architektúra (komponensek, állapot, útválasztás) a kód és annak szervezése által definiált.

6.2.2. Állapotkezelési Megközelítés

A választott állapotkezelési megoldás részletes magyarázata. Hogyan kezelik a globális és lokális állapotot. Az állapotfrissítések adatfolyama.

6.2.3. Útválasztási Stratégia (Routing)

Hogyan történik a navigáció és a nézetek kezelése (pl. React Router, Vue Router használatával). Kulcsfontosságú útvonalak és a hozzájuk tartozó nézetek/komponensek. Védett útvonalak és hitelesítéskezelés a frontend oldalon.

6.2.4. API Interakciós Réteg

Hogyan kommunikál a frontend az Marketplace\_API-val. Szolgáltatási rétegek, hook-ok vagy funkciók, amelyek felelősek az API hívásokért. Adattranszformációs vagy hibakezelési stratégiák az API válaszokhoz.

6.3. Fejlesztői Környezet Beállítása: Marketplace\_Frontend

6.3.1. Előfeltételek

Node.js/npm/yarn verziókövetelmények. Bármilyen globális CLI eszköz, ami szükséges.

6.3.2. Klónozás, Telepítés és Konfiguráció

Parancsok a tároló klónozásához. npm install vagy yarn install utasítások. Környezeti változók (.env fájlok) az API alap URL-ekhez stb. A telepítési utasítások létfontosságúak minden fejlesztő számára. A README 4 vagy a package.json szkriptek tartalmaznák ezeket. Elérhetetlenségük 2 azt jelenti, hogy egy új fejlesztő el sem tudja kezdeni a munkát a projekten. A frontend projekt README.md 4 és package.json fájlja szabványos helye a fejlesztői telepítési utasításoknak (pl. telepítési, indítási, build szkriptek).

6.3.3. A Fejlesztői Szerver Futtatása

npm start vagy yarn dev parancsok. Az alkalmazás elérése böngészőben (pl. localhost:8000).

6.3.4. Build Készítése Éles Környezethez

npm run build vagy yarn build parancsok. A build kimenetének leírása (pl. fájlok a dist vagy build mappában).

6.4. Kulcs Komponensek és UI Útmutató: Marketplace\_Frontend

6.4.1. UI Komponens Könyvtár Áttekintése

Ha egy harmadik féltől származó könyvtárat (pl. Material UI) széles körben használnak, rövid megjegyzés annak integrációjáról. Az egyedi készítésű, megosztott komponensek áttekintése.

6.4.2. Főbb Egyedi Komponensek Részletes Leírása

Ebben a részben kerülnének bemutatásra a projektben definiált főbb, újrafelhasználható egyedi UI komponensek. Minden ilyen komponens esetében a dokumentációnak tartalmaznia kellene:

● Cél és funkcionalitás: A komponens rendeltetésének és működésének leírása.

● Prop-ok (bemenetek): A komponens által elfogadott paraméterek (properties), azok neve, típusa, leírása, és esetleges alapértelmezett értéke.

● Kibocsátott események (kimenetek): Amennyiben a komponens eseményeket bocsát ki (pl. egy gomb kattintása), ezeknek a neve és az általuk hordozott adatok leírása.

● Slot-ok vagy gyermekelemek használata: Ha a komponens lehetővé teszi más elemek beágyazását (pl. slot-okon keresztül), ennek módja és korlátai.

● Használati példák: Rövid kódpéldák, amelyek bemutatják a komponens tipikus használatát.

● Vizuális megjelenítés: Ideális esetben képernyőképek vagy illusztrációk a komponens kinézetéről.

A frontend vizuális és interaktív elemeit a komponensei határozzák meg. Kódhozzáférés nélkül 2 a komponensstruktúra, azok prop-jai és stílusozási megközelítései ismeretlenek maradnak, ami lehetetlenné teszi az UI dokumentációját. A frontend komponensek (pl. React, Vue, Angular komponensek) definiálják az UI-t és a felhasználói interakciókat. Ezen komponensek megértéséhez szükség van a forráskódjuk (JSX, TSX, Vue fájlok stb.) és a kapcsolódó CSS vagy stílusdefiníciók vizsgálatára. Mivel a forrás tárolók jelenleg elérhetetlenek 2, a konkrét komponensek részletes dokumentálása ezen a ponton nem lehetséges.

6.4.3. Témázási és Stílusozási Irányelvek

Hogyan közelítik meg a stílusozást (pl. CSS Modulok, Styled-components, Tailwind CSS). Téma változók (színek, tipográfia, térközök). Irányelvek új stílusok létrehozásához vagy a meglévők módosításához.

6.5. Tesztelés és Minőségbiztosítás: Marketplace\_Frontend

6.5.1. Tesztelési Stratégiák

Alkalmazott teszttípusok (pl. Egységteszt, Integrációs teszt, Végponttól-végpontig teszt). Használt tesztelési keretrendszerek és könyvtárak (pl. Jest, React Testing Library, Cypress, Playwright).

6.5.2. Tesztek Futtatása

Parancsok a különböző tesztcsomagok végrehajtásához. Tesztjelentések megtekintése.

6.5.3. Kódlefedettség

Információ arról, hogyan mérik és jelentik a kódlefedettséget, ha elérhető.

A tesztelési konfigurációk, szkriptek és meglévő tesztek mind a tárolón belül találhatók. Az elérhetetlenség 2 azt jelenti, hogy a projekt minőségbiztosítási megközelítése nem határozható meg. A tesztelés a szoftverminőség kritikus része. A tesztfájlok (pl. \*.test.js, \*.spec.ts), tesztelési konfigurációk (pl. jest.config.js) és tesztfuttató szkriptek (a package.json-ben) a projekt tárolójában helyezkednek el.

7. Integrált Rendszer: Marketplace\_API & Marketplace\_Frontend Interakció (Hipotetikus Tartalmi Terv)

Ez a szakasz azt részletezné, hogyan tervezték a két projektet, az Marketplace\_API-t és a Marketplace\_Frontend-et, hogy együttműködjenek, feltételezve, hogy egy összetartozó alkalmazást alkotnak. Ez mindkét tárolóból származó információkat igényel.

7.1. Backend-Frontend Interakció és Adatfolyam

7.1.1. Teljes Rendszerdiagram

Egy magas szintű diagram, amely bemutatja a Marketplace\_Frontend kommunikációját az Marketplace\_API-val, és potenciálisan más szolgáltatásokkal vagy adatbázisokkal.

7.1.2. Kulcsfontosságú API Integrációs Pontok

A főbb frontend funkciók/felhasználói folyamatok leképezése az általuk használt specifikus Marketplace\_API végpontokra (pl. "A felhasználói bejelentkezés a frontenden a POST /auth/login hívást intézi az API-n," "A termékmegjelenítő oldal a frontenden a GET /products/{id} hívást intézi az API-n").

7.1.3. Adatszinkronizáció és Konzisztencia

Stratégiák a frontend állapotának a backend adatokkal való konzisztenciájának fenntartására. Optimista frissítések, gyorsítótárazás vagy valós idejű frissítések kezelése (ha van ilyen).

7.1.4. Hitelesítési Folyamat a Rétegek Között

Hogyan használják a frontend által megszerzett hitelesítési tokeneket az API kérésekben. Munkamenet-kezelési megfontolások, ha alkalmazható.

Az integrált rendszer alapfeltevése mindkét részének megértésén nyugszik. Mivel mindkét tároló elérhetetlen 1, kölcsönhatásuk dokumentálása duplán lehetetlen. Ez a szakasz megköveteli annak megértését, hogy a frontend mely API végpontokat hívja, és hogyan történik az adatcsere. Ehhez hozzáférés szükséges a frontend kódjához (az API hívás implementációinak megtekintéséhez) és az API dokumentációjához/kódjához (a rendelkezésre álló végpontok megértéséhez).

# 8. Összegzés

8.1. A Dokumentációs Terv Összefoglalása

Röviden összefoglalásra kerül az Marketplace\_API és a Marketplace\_Frontend számára felvázolt átfogó dokumentációs struktúra. Ez a terv alapul szolgálhat a tényleges dokumentáció elkészítéséhez, amint a szükséges forrásanyagok rendelkezésre állnak.

8.2. A Kritikus Akadály Ismételt Hangsúlyozása: Tárolók Elérhetetlensége

Nyomatékosítani kell, hogy a felvázolt dokumentációs terv végrehajtása teljes mértékben a megadott GitHub tárolókhoz való hozzáférés biztosításától függ. Az 1 és 2 hivatkozások igazolják a jelenlegi elérhetetlenséget. Enélkül a hozzáférés nélkül a dokumentáció nem tudja betölteni alapvető funkcióját: a szoftver pontos és megbízható leírását. A továbblépéshez elengedhetetlen a felhasználó közreműködése a tárolókhoz való hozzáférés biztosításában, például a linkek ellenőrzésével, érvényes linkek megadásával, vagy privát tárolók esetén hozzáférési engedélyekkel.

A Vendo Marketplace projekt során teljes értékű, biztonságos és skálázható alkalmazást hoztunk létre, amely alkalmas mobil és asztali használatra egyaránt.

# 9. Irodalomjegyzék, hivatkozások

### JSON Web Tokens (JWT):

### Bevezetés: <https://jwt.io/introduction>

### Könyvtárak: <https://jwt.io/libraries>

### Argon2 (Jelszó Hashelő Algoritmus):

* + Specifikáció és információk: <https://www.cryptolux.org/index.php/Argon2> (lásd)

### Referencia implementáció (GitHub): <https://github.com/P-H-C/phc-winner-argon2>

### Go implementáció (példa): <https://pkg.go.dev/golang.org/x/crypto/argon2>

### DaisyUI (Tailwind CSS Komponens Könyvtár):

### Bevezetés és dokumentáció: <https://daisyui.com/docs/intro/>

### Figma könyvtár: <https://daisyui.com/blog/figma/>

### React (JavaScript Könyvtár UI Építéséhez):

### Hivatalos weboldal (új): <https://react.dev/>

### Korábbi weboldal (legacy): <https://legacy.reactjs.org/>

### React Router DOM (Kliensoldali Útválasztás Reacthez):

### Hivatalos weboldal: <https://reactrouter.com/home>

### NPM csomag: <https://www.npmjs.com/package/react-router-dom>