

TOLNA VÁRMEGYEI SZC
Apáczai Csere János Technikum
és Kollégium

Vizsgaremek

Készítették:

- **Nyári Milán Bence**
- **Péter Édua**
- **Niedermayer Dávid Károly**

Pécs

2025

TOLNA VÁRMEGYEI SZC
Apáczai Csere János Technikum
és Kollégium

Szoftverfejlesztő és -tesztelő
A szakma azonosító száma: 506131203

Vizsgaremek

WareTech

Készítették:

- **Nyári Milán Bence**
- **Péter Édua**
- **Niedermayer Dávid Károly**

Tartalomjegyzék

• Projekt ismertetése, projekt dokumentáció	6
• Bevezetés	6
• Cél	6
• Jövőbeli tervek	6
• Technológiák	7
Használt nyelvek	7
1. Frontend	7
2. Backend	7
Használt technológiák	7
1. Git	7
2. Figma	7
3. Jira.....	7
4. Postman.....	8
5. Discord	8
• Adatbázis.....	8
• Fejlesztési modell	11
Scrum	11
• Tesztelés	12
Használt technológiák	12
Tesztelés fajtái.....	12
Jelentőség.....	12
• Program működésének leírása.....	13
Munkásként bejelentkezve	13
Adminisztrátorként bejelentkezve	13
• Fejlesztői dokumentáció	14
Fejlesztési technológiák	14
HTTP protokollok	14
1. GET.....	14
2. POST	14
3. DELETE	14
A rétegek szerepköre a projektben	14
1. Config réteg.....	14
2. Controller réteg.....	15
3. Model réteg	15

4. Service réteg	16
A projektben való rétegek felépítése	16
1. Config réteg.....	16
2. Controller réteg.....	17
3. Modell réteg	17
4. Service réteg	17
Frontend alapos leírása	18
Könyvtárstruktúra	18
Alapstruktúrája	18
Komponensek	18
Stílusok	19
Reszponzivitás	19
AuthService	19
Modulok és szolgáltatások.....	19
Angular modulok	19
Frontenden lévő főbb funkciók és annak megoldásai.....	19
• Lokális szerver futása.....	21
• Csapatmunka	21
Szerepek	21
• <i>Nyári Milán Bence</i>	21
• <i>Péter Édua</i>	21
• <i>Niedermayer Dávid Károly</i>	22
• Munkafelosztás	23
• Jövőbeli tervek	23
Teszt Dokumentáció.....	24
USER OSZTÁLY	25
SHELFS OSZTÁLY:	32
STORAGE OSZTÁLY:	37
PALLET OSZTÁLY:	40
ITEMS OSZTÁLY:	44
INVENTORY MOVEMENT OSZTÁLY:	46
MOVEMENT REQUEST OSZTÁLY:.....	47
SELENIUM TESZTEK:.....	52
Telepítési útmutató	53
Rendszerkövetelmények.....	54

Hardver követelmények.....	54
A minimális ajánlott rendszerkövetelmények:	54
Szoftver Követelmények	54
A program telepítése	54
Figma live design dokumentáció	61

- **Projekt ismertetése, projekt dokumentáció**

- **Bevezetés:**

Projektünk célja egy modern, webalapú raktárkezelő alkalmazás létrehozása, amely megoldja a raktárkészlet nyilvántartásának és mozgatásának problémáit. Sok vállalat számára kihívást jelent a raktári készletek átlátható kezelése és a pontos helymeghatározás. Alkalmazásunk ezt a problémát célozza: egyszerűbbé, hatékonyabbá és átláthatóbbá teszi a logisztikai folyamatokat. A rendszer lehetővé teszi a raktárak, polcok, raklapok és termékek kezelését, miközben valós idejű adatokat biztosít a dolgozók és vezetők számára.

- **Cél:**

Az alkalmazás célja, hogy digitalizálja és optimalizálja a raktárkezelést. A rendszer támogatja a raktárosokat, logisztikai vezetőket és menedzsereket a készlet mozgásának nyomon követésében, a raktári helyek hatékony kihasználásában és a készletszintek valós idejű monitorozásában. A cél egy felhasználóbarát, rugalmas platform, amely csökkenti a hibázási lehetőségeket, növeli az átláthatóságot és javítja a raktározási hatékonyságot.

- **Jövőbeli tervek:**

A weboldal jelenlegi verziója szilárd alapot nyújt, de számos előremutató fejlesztést tervezünk, hogy a rendszer még sokoldalúbb legyen. Célunk a platform integrálása külső rendszerekkel, például más logisztikai vagy készletkezelő szoftverekkel, hogy rugalmasan alkalmazkodjon a jövőbeli igényekhez és támogassa a bővülő műveleteket. Tervezzük, hogy a rendszer képes legyen két, akár földrajzilag különálló raktár egyidejű kezelésére, valós idejű adatszinkronizációval és egységes munkafolyamatokkal. Szeretnénk továbbfejleszteni a mobilalkalmazás élményét illetve implementálni egy teljesen mobilos funkciót ami lehetővé teszi a barcode beolvasását, hogy a raktári dolgozók intuitív és hatékony felületen végezhessek feladataikat, akár útközben is. Egy fejlett analitikai modul bevezetése is a terveink között szerepel, amely részletes betekintést nyújt a készletmozgások és a dolgozók teljesítményének mintázataiba, elősegítve a stratégiai döntéshozatalt. A felhasználói visszajelzések alapján folyamatosan finomhangolnánk a rendszert, hogy lépést tartson a raktári környezet változó követelményeivel.

- **Technológiák**

Adatbázis oldalon:

Az adatbázisunk egy relációs adatbázis, amit MySQL-re építettünk. A MySQL azért jó, mert megbízható, és segít, hogy a weboldal gyorsan tudjon dolgozni az adatokkal, miközben minden biztonságban van.

Használt nyelvek

1. Frontend

- Angular
- HTML
- CSS
- TypeScript

2. Backend

- Java

Használt technológiák

1. Git

- Arra jó, hogy a kód változásait rendszerezetten tárolja, így mindig lehet tudni, mi történt, és ha valami elromlik, vissza lehet állítani egy korábbi verziót. Tökéletes csapatmunkához, mert mindenki egyszerre dolgozhat anélkül, hogy káosz lenne.

2. Figma

- Arra jó, hogy a weboldal kinézetét előre meg lehessen tervezni, mintha egy digitális rajztáblán dolgoznánk. Lehetővé teszi, hogy a dizájn könnyen megosztható és szerkeszthető legyen, így a felület már az elejétől szép és átgondolt.

3. Jira

- Ez egy tökéletes eszköz a feladatok rendszerezésére. Segít, hogy mindenki lássa, mit kell megcsinálni, mi a legfontosabb, és ki mit csinál, így a munka gördülékeny és átlátható.

4. Postman

- Ez egy olyan eszköz, ami az API-k tesztelésére szolgál. Segít, hogy a weboldal háttérben futó részei, például az adatokat kérő vagy küldő funkciók, hibátlanul működjenek, és gyorsan kiderüljön, ha valami nem stimmel.

5. Discord

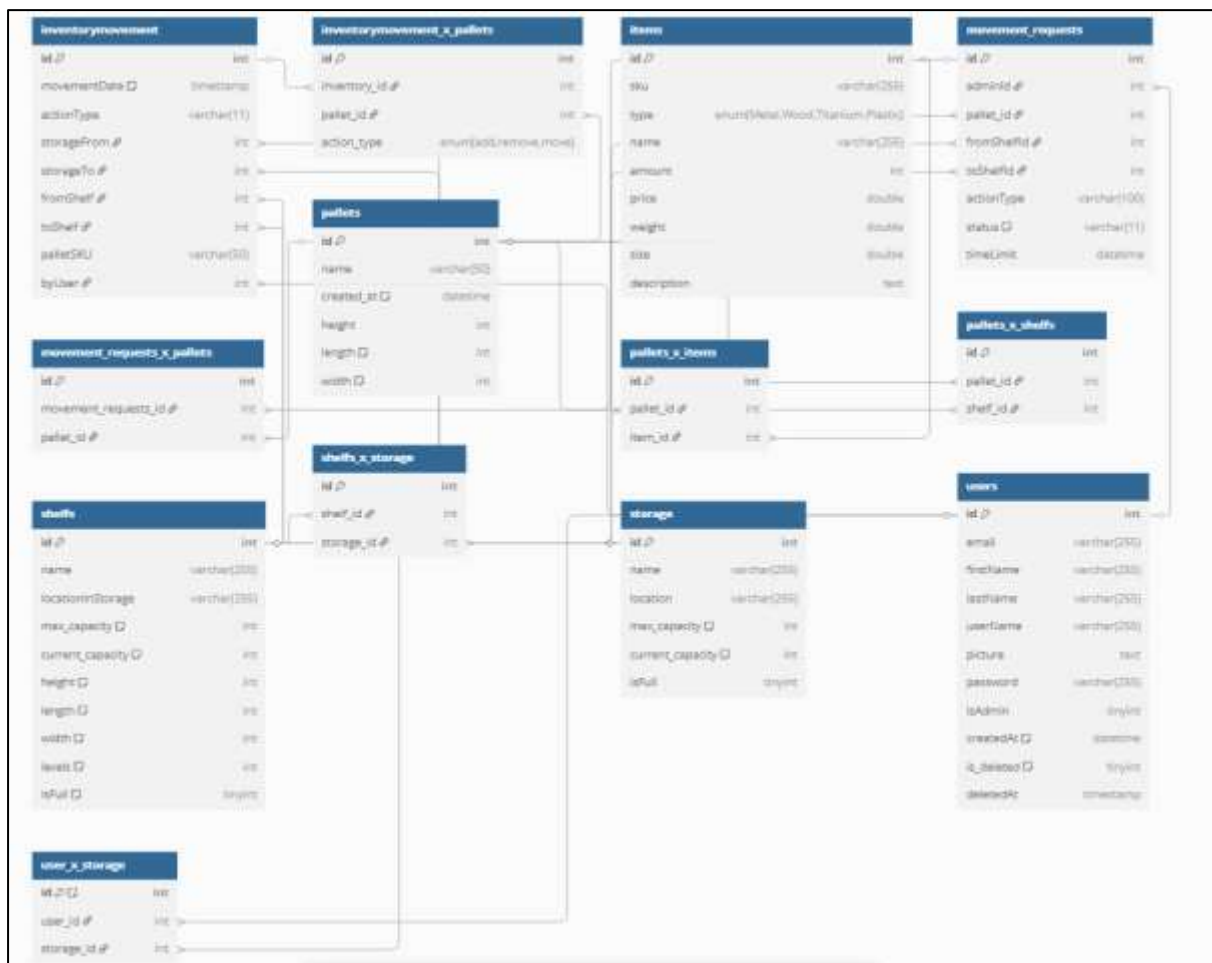
- Ezen eszköz tökéletes a csapattal való kommunikáció fenntartására, hibák esetén képernyőmegosztással is lehet akár segítséget kérni.

• **Adatbázis**

Adatbázis leírása

1. Az adatbázis lényege, hogy minden fontos adatot rendszerezetten, átláthatóan tudjunk tárolni. Egy jól megtervezett adatbázis segít abban, hogy az alkalmazás gyorsan megtalálja, amit keres, és gördülékenyen működjön. Legyen szó termékekről, felhasználókról vagy épp feladatokról, az adatbázis mindent egy helyen tart nyilván, és úgy rendezi el, hogy az adatok kapcsolódni tudjanak egymáshoz.
2. Az ilyen rendszerek akkor igazán hasznosak, ha sokféle információval dolgozunk, és fontos, hogy ezek naprakészen, pontosan elérhetőek legyenek. Például nyomon tudjuk követni, ki mit csinált, mikor történt egy változás, vagy hogy valamiből mennyi van készleten.

3. A modern alkalmazásokban az adatbázis nemcsak háttérben futó „adat-tároló”, hanem aktív része a rendszernek: támogatja a döntéseket, segít a műveletek automatizálásában, és lehetővé teszi, hogy az egész rendszer skálázható, átlátható és megbízható legyen.



A képen a rendszer adatbázisának vázlata látható. Ez mutatja meg, hogy milyen adatokat tárolunk, és hogyan kapcsolódnak egymáshoz. Látszanak benne a táblák (mint például felhasználók vagy termékek), azok adatai, a köztük lévő kapcsolatok, és az is, melyik adat alapján lehet egyértelműen beazonosítani egy sort (elsődleges kulcs), illetve melyik adat mutat egy másik táblára (idegen kulcs).

Az adatbázis minden része különböző célokra alkalmas:

1. Felhasználói adatok kezelése

- A users tábla tárolja a felhasználók alapadatait, mint például e-mail cím, vezetéknév, keresztnév, felhasználónév, jelszó és jogosultsági szint (admin-e vagy sem). A felhasználók egy vagy több raktárhoz (storage) is kapcsolódhatnak a user_x_storage kapcsolótáblán keresztül, ami segít nyomon követni, ki melyik raktárhoz tartozik.

2. Készlet és tárolási egységek

- A storage ,shelves és pallets táblák írják le a raktárak struktúráját. A storage tábla a raktárakat, a shelves a polcokat, a pallets pedig a raklapokat tartalmazza. A polcok és raktárak közötti kapcsolatot a shelves_x_storage, míg a raklapok és polcok közötti kapcsolatot a pallets_x_shelves tábla kezeli. Minden ilyen elemnél figyelve van a kapacitás, telítettség és méretadatok is.

3. Termékek kezelése

- Az items tábla tartalmazza a termékek adatait, például nevüket, típusukat (pl. fém, fa, műanyag), méretüket, árukat és mennyiségüket. A pallets_x_items kapcsolótábla írja le, hogy melyik raklapon milyen termékek találhatók.

4. Mozgások nyomon követése

- Az inventorymovement és inventorymovement_x_pallets táblák kezelik a raktáron belüli mozgásokat. Ezek rögzítik például, hogy mikor, hova és mit mozgattott egy felhasználó. A mozgáshoz tartozik dátum, típus (pl. áthelyezés, kivétel), forrás- és célraktár vagy polc, valamint a mozgattott raklap azonosítója.

5. Mozgási kérelmek

- A movement_requests és movement_requests_x_pallets táblák segítségével lehet mozgási kérelmeket létrehozni, például amikor egy dolgozó javasolja egy raklap áthelyezését. A rendszer kezeli ezek státuszát, időkorlátját, és azt, hogy ki kérte a mozgást, valamint melyik polcra hova történne az áthelyezés.

6. Adatkapcsolatok és integritás

- A kapcsolatok idegen kulcsokon keresztül valósulnak meg, ami biztosítja az adatok pontosságát és összhangját. A sok-sok kapcsolat (például egy raklap több terméket is tartalmazhat, egy raktár több polcot is) segít abban, hogy minden adat jól szervezetten legyen kezelve.

A képen látható adatbázisvázlat bemutatja, hogyan kapcsolódnak egymáshoz az egyes táblák, és hogyan lehet ezekkel hatékonyan működtetni egy raktárkezelő rendszert.

• Fejlesztési modell

Scrum

1. Sprint alapú fejlesztés

- A projekt során a Scrum módszertant alkalmaztuk, amelyben négy hetes sprintekben dolgoztunk. Minden sprint elején meghatároztuk, milyen feladatokat végzünk el, majd a végén kiértékeljük az elkészült munkát. Ez a megközelítés segített abban, hogy folyamatosan haladjunk, és fokozatosan építsük fel a rendszer különböző részeit.

2. Munkaidő naplózása

- A fejlesztés alatt minden csapattag naplózta, hogy mennyi időt töltött egy-egy feladattal. Ez nemcsak az átláthatóságot növelte, hanem segített jobban beosztani az időnket, és látni, mely feladatok igényeltek több energiát, így könnyebben tudtunk tervezni a következő sprintekben.

3. Csapatmunka és kommunikáció

- A csapatmunka kulcsfontosságú volt a projekt során. Rendszeres megbeszéléseken egyeztettünk az aktuális feladatokról, haladásról és esetleges akadályokról. Mindenki tudta, ki min dolgozik, és szükség esetén segítettük egymást. A közös tervezés és visszatekintés segített abban, hogy együtt fejlődjünk és hatékonyabban működjünk.

- **Tesztelés**

Használt technológiák

- Manuális tesztelés – Opera
- Unit tesztelés – JUnit
- Automatizált tesztelés – Selenium

Tesztelés fajtái

1. Manuális tesztelés

- A manuális tesztelés során kézzel próbáltuk ki az alkalmazás funkcióit. Teszteltük a felhasználói felület működését, az egyes gombokat, űrlapokat és a navigációt. Ezzel ellenőriztük, hogy az oldal könnyen kezelhető, logikusan felépített, és minden funkció a vártnak megfelelően működik.

2. Unit tesztelés

- A unit tesztelés során külön-külön teszteltük az egyes funkciókat és metódusokat. Ez segített abban, hogy már a fejlesztés közben észrevegyük az esetleges hibákat, és stabilan tartsuk a kódot. A tesztek automatizáltak voltak, így gyorsan és többször is le tudtuk futtatni őket. Ehhez JUnit-ot használtunk.

3. Automatizált tesztelés (Selenium)

- Selenium segítségével automatizált böngészőteszteket készítettünk. Ez lehetővé tette, hogy szimuláljuk, ahogyan egy felhasználó kattintgat és adatokat visz be az néhány oldalon. A Selenium tesztek gyorsan lefuttatták a gyakori műveleteket amiket le akartunk tesztelni, és visszajelezték, ha valami nem úgy működött, ahogy kellett volna.

Jelentősség

1. Manuális tesztelés

- Segített a felhasználói élmény ellenőrzésében, és abban, hogy az oldal logikusan használható legyen és zökkenőmentesen.

2. Unit tesztelés

- Biztosította, hogy az egyes részek külön-külön is jól működjenek, és a fejlesztések ne rontsák el a már meglévő funkciókat.

3. Selenium tesztek

- Gyors, automatikus módon ellenőrizték pár oldalunk működését, ezzel időt spóroltunk és megbízhatóbbá tettük a rendszert.

• **Program működésének leírása**

Az oldal két felhasználói profilt alkalmaz:

- Munkás
- Műszakvezető / Adminisztrátor

A bejelentkezés során a felhasználók jogosultsága ellenőrizve van ezáltal belépés után az adott felhasználó ezek alapján kapja meg az oldal funkciókat.

Munkásként bejelentkezve

1. Saját profil megtekintése
2. Raktári tárgyak listájának megtekintése
3. Raktár áttekinő hozzáférés (Storage Overview)
4. Raklapok kezelése
5. Feladatok (Movement requestek) teljesítése

Munkás feladata:

- Raklapok kezelése (Hozzáadás, törlés, áthelyezés)
- Mozgatási feladatok teljesítése (Movement request)
- Tárolt tárgyak és raktár – polcstuktúra áttekintése

Adminisztrátorként bejelentkezve

Főbb funkciói: (munkás jogosultságán felül)

1. Adminisztrátori kezelő panel
2. Felhasználók és további adminisztrátorok regisztrálása
3. Raktár és polcstruktúra bővítése (Raktárak és polcok hozzáadása és törlése)
4. Mozgatási feladatok kiadása
5. Tárgyak hozzáadása és törlése

Adminisztrátor feladata:

- Teljeskörű raktár és felhasználókezelés
- Raktárak és polcok létrehozása
- Tárgyak aktualizálása és karbantartása
- Mozgatási feladatok kiadása

- **Fejlesztői dokumentáció**

Fejlesztési technológiák

- Az alkalmazás adatkezelési háttérét MySQL alapú relációs adatbázis adja, ami lehetővé teszi az adatok strukturált, biztonságos és gyors tárolását, ezzel is növelve a rendszer megbízhatóságát.
- A szerveroldali logika megvalósításához JAX-RS technológiát használtunk, ami egy szabványos Java EE REST API megoldás. Előnyei közé tartozik a megjegyzésekkel történő egyszerű konfigurálhatóság és a keretrendszer-függetlenség, amely megkönnyíti az integrációt más Java EE technológiákkal.
- A frontend fejlesztés során az Angular keretrendszert választottuk, ami modern, dinamikus és felhasználóbarát felületet biztosít, ami segíti az élményszerű felhasználói élményt és a frontend fejlesztésének könnyítését.

HTTP protokollok

1. GET

- Az adatbázisból való adatok lekérdezésére használt protokoll, könnyen tesztelhetőek (pl: Postman).

2. POST

- A használt adatok küldésére a szerver felé használatos, például tárgyak kitöltött űrlapjának feltöltéséhez.
- A POST kérés adatai a kérés testében vannak JSON formátumban.

3. DELETE

- Az adatbázisból való adatok törlésére használt protokoll.

A rétegek szerepköre a projektben

1. Config réteg

a) Szerepe:

- A config réteg kezeli az alkalmazás konfigurációs beállításait.

b) Funkciói:

- **Konfigurációkezelés:** A Config réteg tartalmazza azon osztályokat amik az alkalmazásaink konfigurációit kezelik, a CORS filtert és a JWT tokenjeinket ez a réteg kezeli.
- **Betöltés és inicializálás:** Betölti a használt konfigurációkat a megfelelő osztályokba, és inicializálja az alkalmazás konfigurációját

2. Controller réteg

a) Szerepe:

- A Controller csomag kezeli a kliensoldalról érkező HTTP kéréseket, és irányítja azok feldolgozásának menetét.

b) Funkciói:

- **HTTP kérések fogadása:** A *Controller* réteg felelős a kliensoldalról érkező HTTP kérések fogadásáért. Ez lehet GET, POST, PUT, DELETE stb., attól függően, milyen műveletet kíván végrehajtani a felhasználó.
- **Adatok feldolgozása:** A kérésekből kinyeri a szükséges adatokat – például útvonalváltozókat, lekérdezési paramétereket vagy a kérés törzsét – és előkészíti azokat a további feldolgozásra.
- **Válasz összeállítása és visszaküldése:** A *Service* rétegből visszakapott eredményeket HTTP válaszként állítja össze, majd visszaküldi azokat a kliensnek a megfelelő státusz kódokkal és választessel.
- **Alapszintű hibakezelés:** Kezeli a gyakori hibákat – például érvénytelen bemenetet vagy hiányzó erőforrást – és ezekhez illeszkedő hibaválaszokat biztosít.

3. Model réteg

a) Szerepe:

- A Modell réteg felelős az alkalmazásban használt adatok felépítésének és tárolási struktúrájának meghatározásáért.

b) Funkciói:

- **Adatmodellek definiálása:** A Modell csomag tartalmazza azokat az osztályokat, amelyek az alkalmazás különféle adatait reprezentálják, például felhasználók, tárgyak, raktárak, polcok stb.

Ezek az osztályok képezik az adatbázis és az alkalmazás közötti kapcsolat alapját.

- **Adatok validálása:** Gyakran itt kap helyet az alapvető validációs logika is, amely biztosítja, hogy az adatok megfeleljenek az előírt formátumnak és üzleti szabályoknak, még mielőtt elmentésre vagy további feldolgozásra kerülnének.
- **Kapcsolatok kezelése:** A modellek gyakran tartalmazznak annotációkat vagy logikát az entitások közötti kapcsolatok (pl. egy-az-egyhez, egy-a-többhöz) kezelésére, amely elősegíti az ORM rendszerek (pl. JPA) működését.

4. Service réteg

a) Szerepe:

- A Service réteg felelős a logika kezeléséért és annak végrehajtásához

b) Funkciói:

- **Adatok kezelése:** A Service osztályok kapcsolatban állnak az adatbázissal vagy más adatforrásokkal, hogy szükség esetén lekérdezzék, frissítsék vagy töröljék az adatokat.
- **Üzleti folyamatok irányítása:** A Service réteg tartalmazza azokat az osztályokat, amelyek az alkalmazás működéséhez szükséges logikai műveleteket hajtják végre, mint például adatok lekérdezése, manipulálása és üzleti szabályok érvényesítése.

A projektben való rétegek felépítése

1. Config réteg

a) CORS (Cors Filter) osztály

- Ez az osztály a Cross-Origin Resource Sharing (CORS) beállításainak kezeléséért felelős a backend oldalon. Lehetővé teszi vagy blokkolja a kereséseket különböző eredetű forrásokból, ezzel biztosítva az alkalmazás biztonságát és szabályozva a külső hozzáféréseket.

b) Token osztály (JWT)

- Ez az osztály lehetővé teszi a tokenek kezelését, például a JWT (JSON Web Token) létrehozását és érvényesítését.

2. Controller réteg

- **ApplicationConfig:** A globális alkalmazásbeállításokat kezeli
- **InventorymovementController:** Az alkalmazás készletmozgásait kezeli, például a termékek, például a raklapáthelyezéseket.
- **ItemController:** Kezeli az egyes termékekkel kapcsolatos kéréseket, például termékek létrehozása, frissítése és lekérdezése.
- **MovementRequestController:** A készletmozgások kérésének kezeléséért felelős.
- **PalletController:** A raklapokkal kapcsolatos műveletek kezelését végzi.
- **ShelfController:** Kezeli a polcokkal kapcsolatos műveleteket, például létrehozást, módosítást.
- **StorageController:** A raktárok kezelését végzi, például raktár hozzáadás, törlés, módosítás.
- **UserController:** Felhasználók kezelését végzi, beleértve a regisztrációt és felhasználói funkciókat.

3. Modell réteg

- **Inventorymovement:** A készletmozgások adatait tároló modell.
- **Items:** A termékeket reprezentáló modell.
- **Material:** Az anyagok típusait és jellemzőit tartalmazó modell.
- **MovementRequests:** A készletmozgásokat reprezentáló modell.
- **Pallets:** A raklapok adatait tároló modell.
- **Shelfs:** A polcokat leíró modell.
- **Storage:** A raktárhelyiségek adatait tartalmazó modell.
- **Users:** A rendszer felhasználóit reprezentáló modell.

4. Service réteg

- **InventorymovementService:** A készletmozgásokhoz kapcsolódó függvények validálását tartalmazza.
- **ItemService:** A termékekhez kapcsolódó függvények validálását tartalmazza.

- **MovementRequestService:** Az áthelyezési kérésekhez kapcsolódó függvények validálását tartalmazza.
- **PalletService:** A raklapokhoz kapcsolódó függvények validálását tartalmazza.
- **ShelfService:** A polcokhoz kapcsolódó függvények validálását tartalmazza.
- **StorageService:** A raktárakhoz kapcsolódó függvények validálását tartalmazza
- **UserService:** A felhasználókhöz kapcsolódó függvények validálását tartalmazza

Frontend alapos leírása

Könyvtárstruktúra

- A projektünk rendezett és átlátható, könnyű navigálást biztosít.

Alapstruktúrája

- **app:** A projekt főkönyvtára amiben megtalálhatóak a service továbbá a komponens fájlok.
- **src:** Tartalmazza az alkalmazás forrásfájljait.
- **assets:** Ebben találhatóak a projektben használt állandó fájlok. (képek, ikonok, stb.)

Komponensek

Az projektünk oldalait és ismétlődő részeit komponensekre bontottuk:

- **Navbar:** A navigációs menüt megjelenítő komponens, ami tartalmazza az összes oldalhoz való hozzáférés gombjait.
- **Item-list:** A tárgylistát megjelenítő komponens, ez tartalmazza a tárgyakat és azoknak adatait.
- **Login:** Ez a komponens felelős a felhasználó bejelentkezésének kezelésére és annak megjelenítésére.
- **Admin-panel:** Ezen komponens az összes adminisztrátori funkciót megjeleníti és kezeli azokat.
- **Storage-management:** A raktár áttekintő oldaláért felelős, a raktárak és polcok megjelenítését kezeli ezen felül készletszint ellenőrzést is folytat.

- **Profile:** Ezen komponens feladata a felhasználók adatainak megjelenítését és a felhasználónév vagy jelszóváltás funkciójának kezelése.

Stílusok

- A projektünk stílusát CSS és Bootstrap használatával valósítottuk meg, igyekeztünk mindennek egyedi stílust adni a szebb összkép érdekében. A Bootstrap segített az oldal reszponzivitásának elérésében.

Reszponzivitás

- Az alkalmazásunk úgy terveztük hogy 720px-ig minimum reszponzív legyen, ami annyit tesz hogy minden oldal alkalmazkodik az aktuális képernyőmérethez.

AuthService

- Az AuthService osztály az alkalmazás felhasználó autentikációs és autorizációs folyamataiért felelős. Ez a szolgáltatás végzi a bejelentkezéshez szükséges adatküldést a backend felé, valamint a felhasználói adatok és a JWT token lokális tárolását és törlését. Ezáltal megakadályozható az illetéktelen belépés.

Modulok és szolgáltatások

- A projektben modulokat és szolgáltatásokat használunk egyszerűbb kezelhetőségi szempontokból.

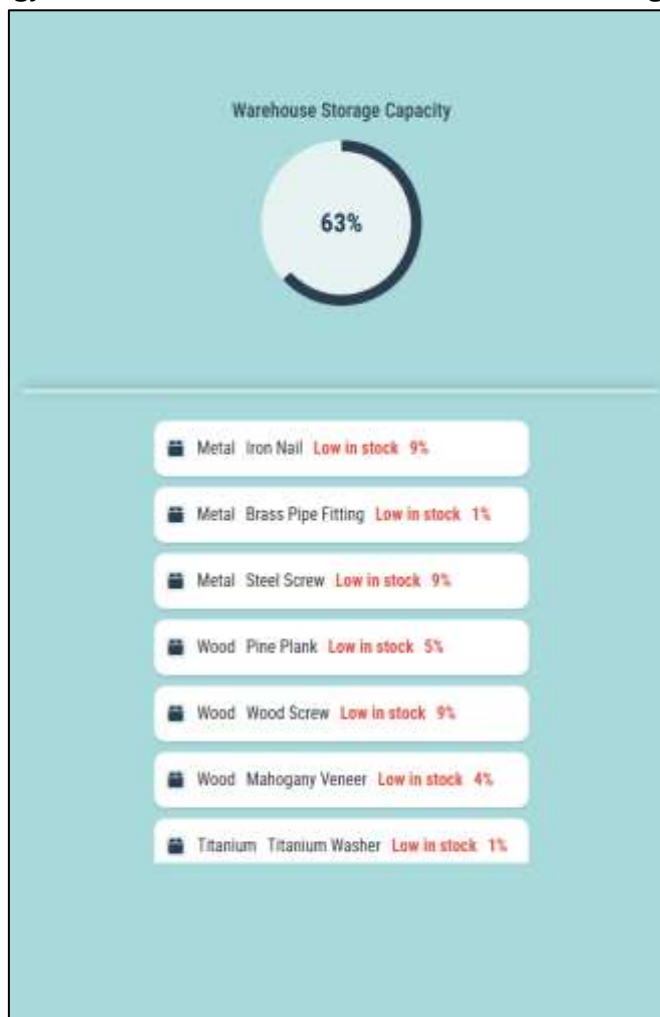
Angular modulok

- **ReactiveFormsModule:** Lehetővé teszi reaktív űrlapok létrehozását és kezelését, amelyek jól skálázhatók és könnyen validálhatók.
- **HttpClientModule:** Az HTTP kommunikációhoz szükséges modult biztosítja, lehetővé téve például adatok lekérését vagy küldését a backend felé.
- **FormsModule:** A hagyományos, template-alapú űrlapok kezelését támogatja, például kétirányú adatközléssel

Frontenden lévő főbb funkciók és annak megoldásai

- A loadInitialData gyorsan összeszedi az itemeket, polcokat és palettákat az API-ból. Ezekből szűrhető listák készülnek, így könnyen megtalálod, amit keresel.

- Az oldal ReactiveFormsModule-t használ ezért könnyebben kezelhetőek a regisztrációk és minden form-al kapcsolatos tevékenység, ezáltal könnyebben megoldható a hibakezelés ezeknél.
- Gyors keresések és filterezések. A frontend valós idejű keresést és szűrést használ ezáltal az oldalnak nem kell frissülnie egy tárgy keresése során vagy egy filterezés során.
- Folyamatos adat megfigyelés van implementálva ezáltal minden új adat valós időben kerül megjelenítésre.
- A getCapacityByShelfUsage lehetővé teszi számunkra, hogy valós időben kiszámítható legyen bármelyik raktárnak a telítettsége és az összes raktár telítettsége egybevonva amit frondenen kiszámolunk és megjelenítünk.



- A getPalletsWithShelfs lekéréssel lekérjük az adatokat és frondenden és megvizsgáljuk annak mennyiségét, ezáltal kiszámolhatóvá tesszük és felhasználhatjuk egy low stock figyelő funkcióhoz.

- A profilok és adminisztrátori panel előhozásához modal-ok lettek használatbavéve ami openAdminModal, closeAdminModal-al(user ugyan így van megoldva) nyílik meg és zárul be. Ezek a funkciók külön service rétegen vannak kezelve.
- A profile ablakban megjelenő adatok is valós idejűek, mivel felhasználónév és jelszóváltoztatás elérhető(passwordChangeByUserId, usernameChangeByUserId) a modalban ezért Two-way data binding van használva ami az azonnali adatfrissítést és megjelenítést lehetővé teszi az oldal újratöltése nélkül ami elősegíti a felhasználók élményének megtartását az oldal használat.

• Lokális szerver futása

- A projekt teljes működését tesztelesekkel és futtatásokkal validáltuk. A rendszer minden komponense, a háttérben futó backend és adatbázis funkciókat is tesztek alá vetettük és az ezek közötti kapcsolatokat is. Az eredmények szerint az alkalmazásunk minden része hatékonyan és párhuzamosan működik.

• Csapatmunka

Szerepek

- **Nyári Milán Bence**

Vállalt területek a projekt során:

1. Project manager (Projektmenedzser)
 - Feladata a projekt elkészítésének irányítása és felügyelése, kommunikáció fenntartása.
2. Adatbázisfejlesztő
 - Feladata a projekt adatbázisának a megtervezése, struktúrájának felépítése és az adatbázisterv megvalósítása.

- **Péter Édua**

Vállalt területek a projekt során:

1. Frontend fejlesztő

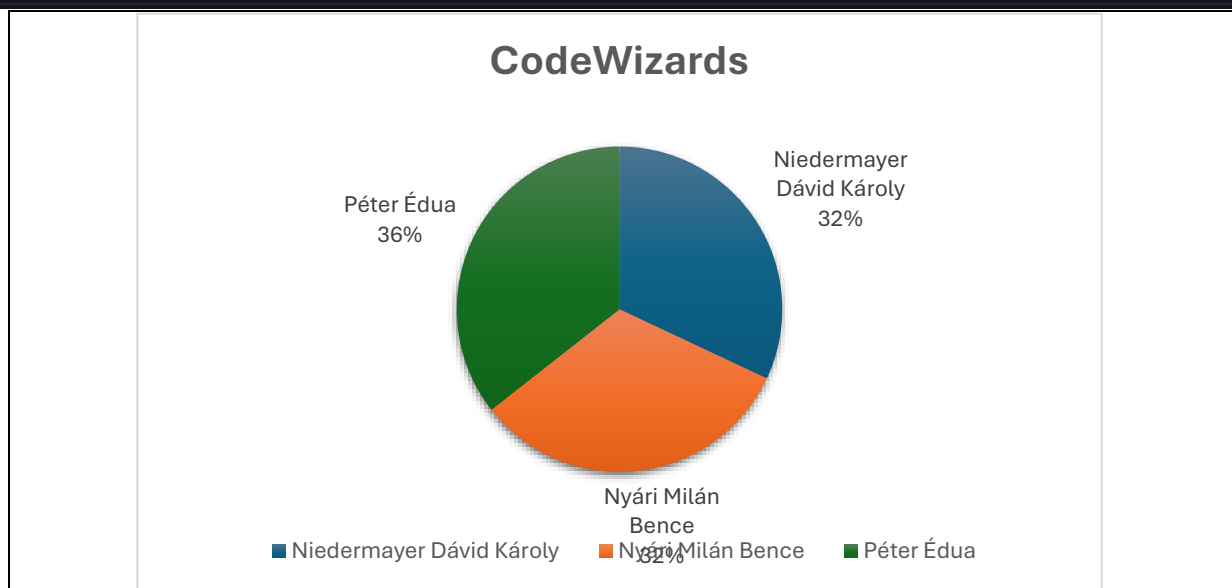
- Feladata a projekt teljes frontendjének a megtervezése annak felépítése és fejlesztése volt.
 - Implementálta a megtervezett felhasználói funkciókat és megvalósította a felhasználói felületet.
2. Manuál tesztelő
 - Feladata a fejlesztőcsapat értesítése a tesztelési eredményekről, esetleges problémák felmerüléséről.
 - Ellenőrizte hogy az alkalmazás megfelelő-e a követelményeknek.

- **Niedermayer Dávid Károly**

1. Backend fejlesztő
 - Felelős volt a háttérben futó rendszer kialakításáért és implementációjáért.
 - A szerveroldali funkciókat és logikákat megvalósította.
 - Biztosította a rendszer hatékonyságát és megbízhatóságát.
2. Unit tesztelő
 - Feladata a backend funkcióinak és osztályainak a tesztelése
 - A kód megbízhatóságának és működésének biztosítása érdekében implementálta és futtatta a unit teszteket.
3. Automatizált tesztelő
 - Feladata az alkalmazás néhány oldalának az automatizált tesztelése.
 - Implementálta a Seleniumos automatizált tesztelést és futtatta azt több adattal is az oldal funkcióinak tesztelése és annak hatékonyságának ellenőrzésének érdekében.

- **Munkafelosztás**

Branch	Updated	Check status	Behind	Ahead	Pull request
Frontend	5 hours ago		45	5	
Backend	5 hours ago		45	1	
Database	last month		01	0	



- **Jövőbeli tervek**

A rendszer funkcionalitásának bővítése érdekében a jelenlegi *adminisztrátor* és *felhasználó* szerepkörökön túl további pozíciók, például műszakvezető és raktári dolgozó bevezetését tervezzük, hogy a raktári munkafolyamatok teljes spektrumát lefedjük.

Tervben van a vonalkódolvasás integrálása az űrlapokba és az egész rendszerbe, lehetővé téve az itemek gyors és pontos azonosítását.

A készletmozgások hatékonyabb nyomon követése érdekében készletjelentések alapján automatikus rendelési javaslatokat, valamint a készletfogyás tendenciáit szemléltető vizualizációkat (diagramokat) kívánunk beépíteni a rendszerbe.

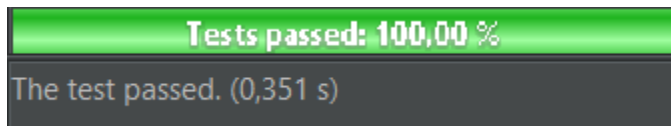
Teszt Dokumentáció

Készítette:
Niedermayer Dávid Károly

USER OSZTÁLY

Ez a Junit teszt a User osztályon végez egy get user by id http kérést, ellenőrizve, hogy egy érvényes ID-vel lekért felhasználó adatai helyesen jelennek-e meg, beleértve az emailt, és hogy a HTTP státuszkód 200 legyen.

Jó teszt: Egy get user by id az adatbázisban létező id-vel.



Hozzá tartozó kódrészlet:

```
@Test
public void testGetUserById_withValidId_returnsUser() {

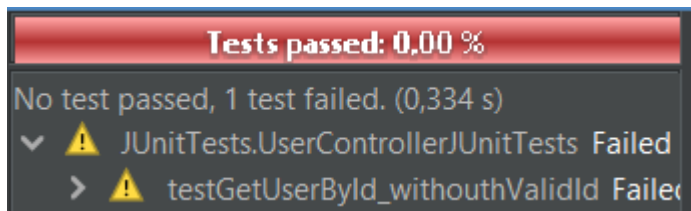
    Response response = client.target(BASE_URI)
        .path("getUserById")
        .queryParams("id", 1)
        .request(MediaType.APPLICATION_JSON)
        .get();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertTrue(responseBody.has("id"), "A válaszban szerepelnie kell az 'id' mezőnek");
    assertEquals("user@example.com", responseBody.getString("email"), "Az emailnek meg kell egyeznie");

    response.close();
}
```

Rossz teszt: A get user by id az adatbázisban nem létező id-vel



Hozzá tartozó kódrészlet:

```
@Test
public void testGetUserById_withouthValidId() {

    Response response = client.target(BASE_URI)
        .path("getUserById")
        .queryParams("id", 9999)
        .request(MediaType.APPLICATION_JSON)
        .get();

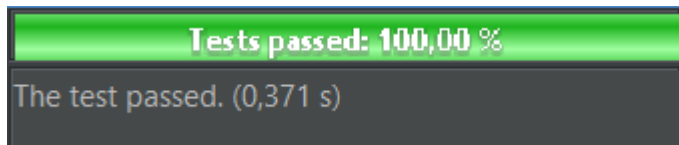
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertTrue(responseBody.has("id"), "A válaszban szerepelnie kell az 'id' mezőnek");
    assertEquals("user9999@example.com", responseBody.getString("email"), "Az emailnek meg kell egyeznie");

    response.close();
}
```

A testRegisterUser_withValidData_returnsSuccess teszt a POST /user/registerUser végpontot ellenőrzi, várva a 200-as státuszkódot és "success" választ érvényes adatokra.

Jó teszt mivel az adatok még nem léteztek az adatbázisban:



```
@Test
public void testRegisterUser_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("email", "test.elak@example.com")
        .put("firstName", "Testt")
        .put("lastName", "Elak")
        .put("userName", "testtelek")
        .put("picture", "base64image")
        .put("password", "Test123!@#");

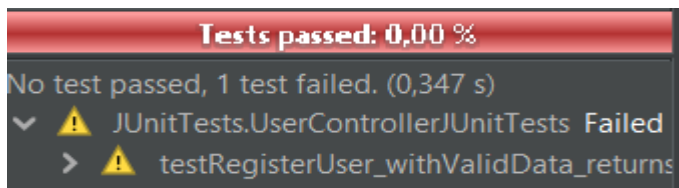
    Response response = client.target(BASE_URI)
        .path("registerUser")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("status"), "A státusz 'success' kell legyen");
    assertEquals(200, responseBody.getInt("statusCode"), "A státusz kódnak 200-nak kell lennie");

    response.close();
}
```

Rossz teszt mivel az adatbázisban már létezik ez a felhasználó:



```
@Test
public void testRegisterUser_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("email", "test.elak@example.com")
        .put("firstName", "Testt")
        .put("lastName", "Elak")
        .put("userName", "testtelek")
        .put("picture", "base64image")
        .put("password", "Test123!@#");

    Response response = client.target(BASE_URI)
        .path("registerUser")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

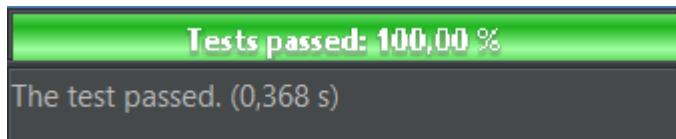
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("status"), "A státusz 'success' kell legyen");
    assertEquals(200, responseBody.getInt("statusCode"), "A státusz kódnak 200-nak kell lennie");

    response.close();
}
```

A testRegisterAdmin_withValidData_returnsSuccess teszt a POST /user/registerAdmin végpontot ellenőrzi, várva a 200-as státuszkódot és "success" választ érvényes admin adatokra.

Jó teszt mivel az adatok még nem léteztek az adatbázisban:



```
@Test
public void testRegisterAdmin_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("email", "admin@example.com")
        .put("firstName", "Admin")
        .put("lastName", "Főnök")
        .put("userName", "adminfo")
        .put("picture", "base64image")
        .put("password", "Admin123!@#");

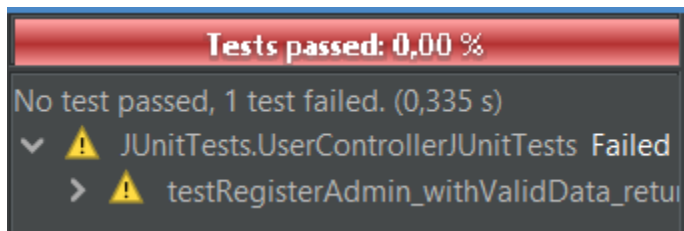
    Response response = client.target(BASE_URI)
        .path("registerAdmin")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("status"), "A státusz 'success' kell legyen");
    assertEquals(200, responseBody.getInt("statusCode"), "A státusz kódnak 200-nak kell lennie");

    response.close();
}
```

Rossz teszt mivel az adatbázisban már létezik ez a felhasználó:



```
@Test
public void testRegisterAdmin_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("email", "admin@example.com")
        .put("firstName", "Admin")
        .put("lastName", "Főnök")
        .put("userName", "adminfo")
        .put("picture", "base64image")
        .put("password", "Admin123!@#");

    Response response = client.target(BASE_URI)
        .path("registerAdmin")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("status"), "A státusz 'success' kell legyen");
    assertEquals(200, responseBody.getInt("statusCode"), "A státusz kódnak 200-nak kell lennie");

    response.close();
}
```

A testGetAllUsers_withExistingUsers_returnsUserList teszt a GET /user/getAllUsers végpontot ellenőrzi, várva a 200-as státuszkódot és a nem üres felhasználói listát tartalmazó választ.

Jó teszt:

Tests passed: 100,00 %

The test passed. (0,368 s)

```
@Test
public void testGetAllUsers_withExistingUsers_returnsUserList() {
    Response response = client.target(BASE_URI)
        .path("getAllUsers")
        .request(MediaType.APPLICATION_JSON)
        .get();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertTrue(responseBody.has("users"), "A válaszban szerepelnie kell a 'users' tömbnek");
    assertTrue(responseBody.getJSONArray("users").length() > 0, "A felhasználók listája nem üres");

    response.close();
}
```

A testPasswordChangeByUserId_withValidData_returnsSuccess teszt a PUT /user/passwordChangeByUserId végpontot ellenőrzi, várva a 200-as státuszkódot és "success" választ érvényes jelszóváltoztatási adatokra.

Jó teszt jó adatok esetén:

Tests passed: 100,00 %

The test passed. (0,362 s)

```
@Test
public void testPasswordChangeByUserId_withValidData_returnsSuccess() {
    // Jelszó módosítása
    JSONObject requestBody = new JSONObject()
        .put("userId", 2)
        .put("oldPassword", "password123")
        .put("newPassword", "New123!@#");

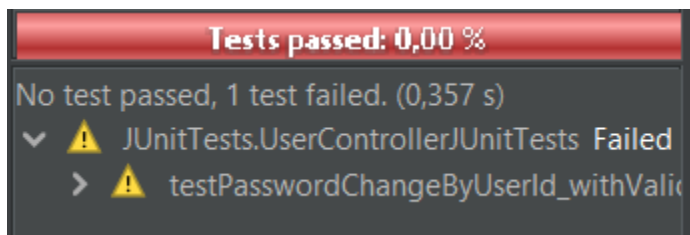
    Response response = client.target(BASE_URI)
        .path("passwordChangeByUserId")
        .request(MediaType.APPLICATION_JSON)
        .put(Entity.json(requestBody.toString()));

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("status"), "A státusz 'success' kell legyen");
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");

    response.close();
}
```

Rossz teszt hibás jelszó esetén:



```
@Test
public void testPasswordChangeByUserId_withValidData_returnsSuccess() {

    // Jelszó módosítása
    JSONObject requestBody = new JSONObject()
        .put("userId", 2)
        .put("oldPassword", "Rozserpassword123")
        .put("newPassword", "Haw123!@#");

    Response response = client.target(BASE_URI)
        .path("passwordChangeByUserId")
        .request(MediaType.APPLICATION_JSON)
        .put(Entity.json(requestBody.toString()));

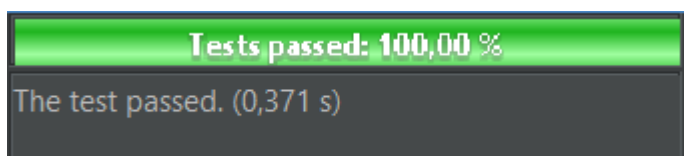
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("status"), "A státusz 'success' kell legyen");
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");

    response.close();
}
```

A testUsernameChangeByUserId_withValidData_returnsSuccess teszt a PUT /user/usernameChangeByUserId végpontot ellenőrzi, várva a 200-as státuszkódot, "success" választ és az új felhasználónevet érvényes adatokra.

Jó teszt jó id esetén:



```
@Test
public void testUsernameChangeByUserId_withValidData_returnsSuccess() {

    // Felhasználó módosítása
    JSONObject requestBody = new JSONObject()
        .put("userId", 3)
        .put("newUsername", "ujtest");

    Response response = client.target(BASE_URI)
        .path("usernameChangeByUserId")
        .request(MediaType.APPLICATION_JSON)
        .put(Entity.json(requestBody.toString()));

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("status"), "A státusz 'success' kell legyen");
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertEquals("ujtest", responseBody.getJSONObject("result").getString("username"), "A felhasználónévnek meg kell változnia");

    response.close();
}
```

Rossz teszt rossz id esetén:

Tests passed: 0,00 %

No test passed, 1 test failed. (0,361 s)

- JUnitTests.UserControllerJUnitTests Failed
 - testUsernameChangeByUserId_withValidData_returnsSuccess()

```

@Test
public void testUsernameChangeByUserId_withValidData_returnsSuccess() {
    // Felhasználó módosítása
    JSONObject requestBody = new JSONObject();
    .put("userId", 1000)
    .put("newUsername", "u{test}");

    Response response = client.target(BASE_URI)
        .path("usernameChangeByUserId")
        .request(MediaType.APPLICATION_JSON)
        .put(Entity.json(requestBody.toString()));

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("status"), "A státusz 'success' kell legyen");
    assertEquals(200, responseBody.getInt("statusCode"), "A státusz kódjának 200-nak kell lennie");
    assertEquals("u{test}", responseBody.getJSONObject("result").getString("username"), "A felhasználónév meg kell változzon");

    response.close();
}

```

A `testDeleteUser_withValidId_returnsSuccess` teszt a `DELETE /user/deleteUser` végpontot ellenőrzi, várva a 200-as státuszkódot és "success" választ egy érvényes felhasználó ID törlésére.

Jó teszt:

Tests passed: 100,00 %

The test passed. (0,371 s)

```

@Test
public void testDeleteUser_withValidId_returnsSuccess() {
    // Felhasználó törlés
    Response response = client.target(BASE_URI)
        .path("deleteUser")
        .queryParam("id", 4)
        .request(MediaType.APPLICATION_JSON)
        .delete();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("result"), "A válasz 'success' kell legyen");

    response.close();
}

```

Rossz teszt mivel a user már törölve lett:

Tests passed: 0,00 %

No test passed, 1 test failed. (0,549 s)

▼ ⚠ JUnitTests.UserControllerJUnitTests Failed
 > ⚠ testDeleteUser_withValidId_returnsSuccess

```
@Test
public void testDeleteUser_withValidId_returnsSuccess() {

    // Felhasználó törlése
    Response response = client.target(BASE_URI)
        .path("deleteUser")
        .queryParams("id", 4)
        .request(MediaType.APPLICATION_JSON)
        .delete();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

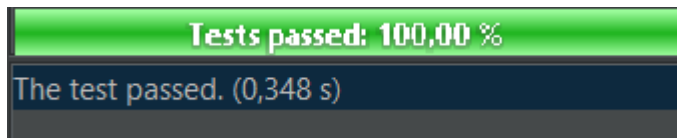
    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("result"), "A válasz 'success' kell legyen");

    response.close();
}
```

SHELFS OSZTÁLY:

A `testAddShelfToStorage_withValidData_returnsSuccess` teszt a `POST /shelves/addShelfToStorage` végpontot ellenőrzi, várva a 201-es státuszkódot, "Shelf successfully added to storage" üzenetet és a helyes polcnevet érvényes adatokra.

Jó teszt:



```
@Test
public void testAddShelfToStorage_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("storageId", 1)
        .put("shelfName", "TestShelf 1")
        .put("locationIn", "Aisle 2");

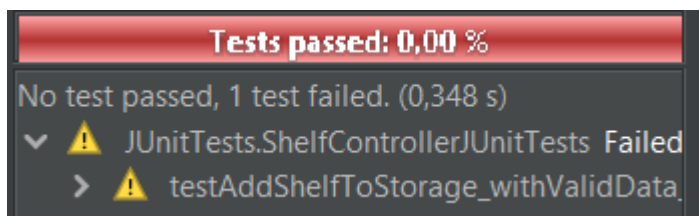
    Response response = client.target(BASE_URI)
        .path("addShelfToStorage")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(201, response.getStatus(), "A státuskódnak 201-nek kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(201, responseBody.getInt("statusCode"), "A státuskódnak 201-nek kell lennie");
    assertEquals("Shelf successfully added to storage", responseBody.getString("message"), "A válaszbannak helyes kell legyen");
    assertEquals("TestShelf 1", responseBody.getString("shelfName"), "A polc neve helyes kell legyen");

    response.close();
}
```

Rossz teszt rossz id esetén:



```
@Test
public void testAddShelfToStorage_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("storageId", 100)
        .put("shelfName", "TestShelf 1")
        .put("locationIn", "Aisle 2");

    Response response = client.target(BASE_URI)
        .path("addShelfToStorage")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

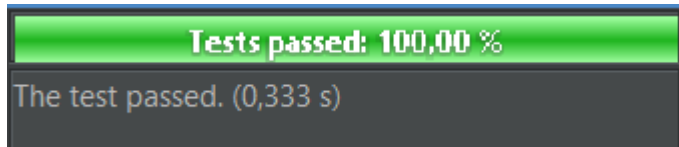
    assertEquals(201, response.getStatus(), "A státuskódnak 201-nek kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(201, responseBody.getInt("statusCode"), "A státuskódnak 201-nek kell lennie");
    assertEquals("Shelf successfully added to storage", responseBody.getString("message"), "A válaszbannak helyes kell legyen");
    assertEquals("TestShelf 1", responseBody.getString("shelfName"), "A polc neve helyes kell legyen");

    response.close();
}
```


A `testGetAllShelves_withExistingShelves_returnsShelfList` teszt a `GET /shelves/getAllShelves` végpontot ellenőrzi, várva a 200-as státuszkódot és a nem üres polclista választ.

Jó teszt:



```
@Test
public void testGetAllShelves_withExistingShelves_returnsShelfList() {
    Response response = client.target(BASE_URI)
        .path("getAllShelves")
        .request(MediaType.APPLICATION_JSON)
        .get();

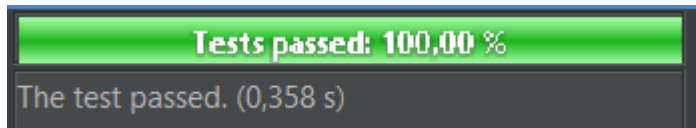
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertTrue(responseBody.has("shelves"), "A válaszban szerepelnie kell a 'shelves' tömbnek");
    assertTrue(responseBody.getJSONArray("shelves").length() > 0, "A polcok listája nem üres");

    response.close();
}
```

A `testGetShelfsById_withValidId_returnsShelf` teszt a `GET /shelfs/getShelfsById` végpontot ellenőrzi, várva a 200-as státuszkódot, az ID mezőt és a "Shelf a" nevet egy érvényes polc ID lekérdezésére.

Jó teszt:



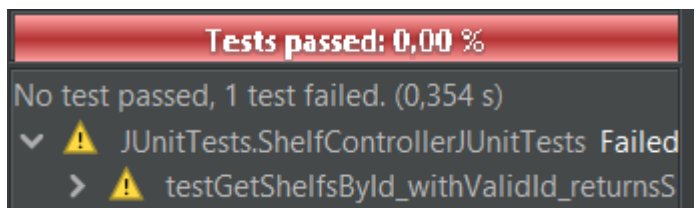
```
@Test
public void testGetShelfsById_withValidId_returnsShelf() {
    Response response = client.target(BASE_URI)
        .path("getShelfsById")
        .queryParams("id", 1)
        .request(MediaType.APPLICATION_JSON)
        .get();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertTrue(responseBody.has("id"), "A válaszban szerepelnie kell az 'id' mezőnek");
    assertEquals("Shelf a", responseBody.getString("name"), "A polc nevének meg kell egyeznie");

    response.close();
}
```

Rossz teszt mivel nem létező id lett megadva:



```
@Test
public void testGetShelfsById_withValidId_returnsShelf() {
    Response response = client.target(BASE_URI)
        .path("getShelfsById")
        .queryParams("id", 100)
        .request(MediaType.APPLICATION_JSON)
        .get();

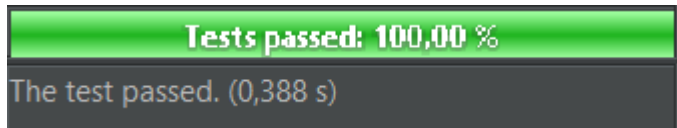
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertTrue(responseBody.has("id"), "A válaszban szerepelnie kell az 'id' mezőnek");
    assertEquals("Shelf a", responseBody.getString("name"), "A polc nevének meg kell egyeznie");

    response.close();
}
```

A `testGetShelvesByStorageId_withValidStorageId_returnsShelves` teszt a `GET /shelves/getShelvesByStorageId` végpontot ellenőrzi, várva a 200-as státuszkódot, nem üres polclistát és a "StorageShelf" nevet egy érvényes tároló ID-re.

Jó teszt:



```
@Test
public void testGetShelvesByStorageId_withValidStorageId_returnsShelves() {
    Response response = client.target(BASE_URI)
        .path("getShelvesByStorageId")
        .queryParam("storageId", 1)
        .request(MediaType.APPLICATION_JSON)
        .get();

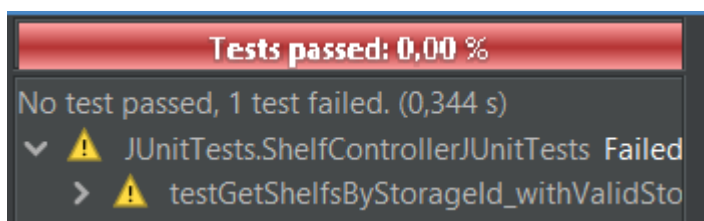
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertTrue(responseBody.has("shelves"), "A válaszban szerepelnie kell a 'shelves' tömbnek");
    assertTrue(responseBody.getJSONArray("shelves").length() > 0, "A polcok listája nem üres");

    JSONArray shelves = responseBody.getJSONArray("shelves");
    JSONObject firstShelf = shelves.getJSONObject(0);
    assertEquals("Shelf B", firstShelf.getString("shelfName"), "A polc nevének meg kell egyeznie");

    response.close();
}
```

Rossz teszt rossz storage id esetén:



```
@Test
public void testGetShelvesByStorageId_withValidStorageId_returnsShelves() {
    Response response = client.target(BASE_URI)
        .path("getShelvesByStorageId")
        .queryParam("storageId", 100)
        .request(MediaType.APPLICATION_JSON)
        .get();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

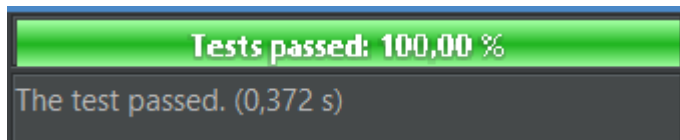
    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertTrue(responseBody.has("shelves"), "A válaszban szerepelnie kell a 'shelves' tömbnek");
    assertTrue(responseBody.getJSONArray("shelves").length() > 0, "A polcok listája nem üres");

    JSONArray shelves = responseBody.getJSONArray("shelves");
    JSONObject firstShelf = shelves.getJSONObject(0);
    assertEquals("Shelf B", firstShelf.getString("shelfName"), "A polc nevének meg kell egyeznie");

    response.close();
}
```

A `testDeleteShelfFromStorage_withValidId_returnsSuccess` teszt a DELETE `/shelves/deleteShelfFromStorage` végpontot ellenőrzi, várva a 200-as státuszkódot és "success" választ egy érvényes polc ID törlésére.

Jó teszt:



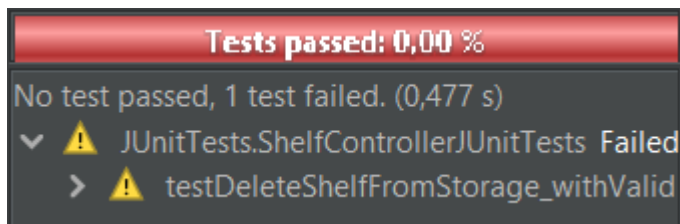
```
@Test
public void testDeleteShelfFromStorage_withValidId_returnsSuccess() {
    Response response = client.target(BASE_URI)
        .path("deleteShelfFromStorage")
        .queryParams("id", 12)
        .request(MediaType.APPLICATION_JSON)
        .delete();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("result"), "A válasz 'success' kell legyen");

    response.close();
}
```

Rossz teszt nem létező ID esetén:



```
@Test
public void testDeleteShelfFromStorage_withValidId_returnsSuccess() {
    Response response = client.target(BASE_URI)
        .path("deleteShelfFromStorage")
        .queryParams("id", 12)
        .request(MediaType.APPLICATION_JSON)
        .delete();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

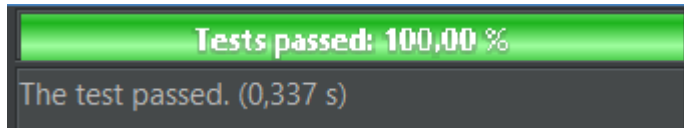
    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("result"), "A válasz 'success' kell legyen");

    response.close();
}
```

STORAGE OSZTÁLY:

testAddStorage_withValidData_returnsSuccess: Ellenőrzi, hogy a POST /storage/addStorage végpont 201-es státuszkóddal és sikerüzenettel hozzáad egy új tárolót érvényes adatokkal.

Jó teszt:



```
@Test
public void testAddStorage_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("storageName", "TestStorage")
        .put("location", "Building A");

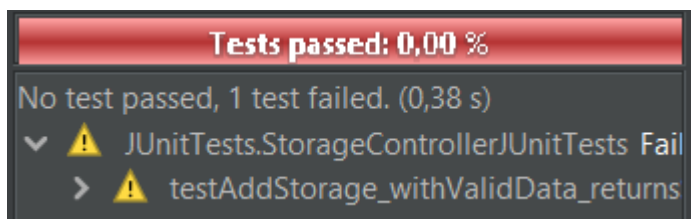
    Response response = client.target(BASE_URI)
        .path("addStorage")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(201, response.getStatus(), "A státusznak 201-nek kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertEquals("Storage successfully added", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");
    assertEquals("TestStorage", responseBody.getString("storageName"), "A tároló neve helyes kell legyen");

    response.close();
}
```

Rossz teszt:



```
@Test
public void testAddStorage_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("storageName", "")
        .put("location", "");

    Response response = client.target(BASE_URI)
        .path("addStorage")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

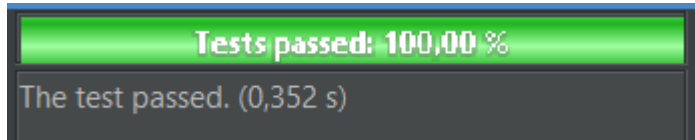
    assertEquals(201, response.getStatus(), "A státusznak 201-nek kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertEquals("Storage successfully added", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");
    assertEquals("TestStorage", responseBody.getString("storageName"), "A tároló neve helyes kell legyen");

    response.close();
}
```

testGetAllStorages_withExistingStorages_returnsStorageList: Ellenőrzi, hogy a GET /storage/getAllStorages végpont 200-as státuszkóddal és nem üres tárolólistával tér vissza meglévő tárolók esetén.

Jó teszt:



```
STest
public void testGetAllStorages_withExistingStorages_returnsStorageList() {
    Response response = client.target(BASE_URI)
        .path("getAllStorages")
        .request(MediaType.APPLICATION_JSON)
        .get();

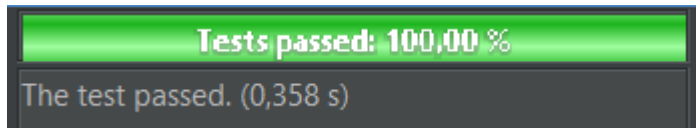
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertTrue(responseBody.has("Storages"), "A válaszban szerepelnie kell a 'Storages' tömbnek");
    assertTrue(responseBody.getJSONArray("Storages").length() > 0, "A tárolók listája nem üres");

    response.close();
}
```

testDeleteStorageById_withValidId_returnsSuccess: Ellenőrzi, hogy a DELETE /storage/deleteStorageById végpont 200-as státuszkóddal és sikerüzenettel töröl egy létező tárolót érvényes ID alapján.

Jó teszt:



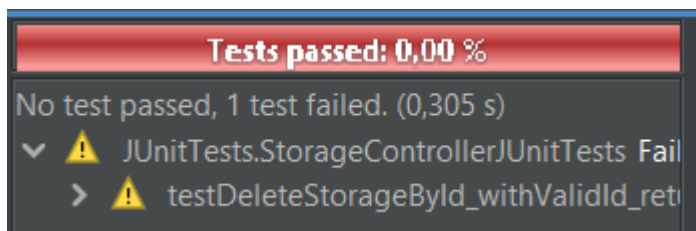
```
@Test
public void testDeleteStorageById_withValidId_returnsSuccess() {
    // Feltevésezzük, hogy létezik egy tároló ID-val 1
    Response response = client.target(BASE_URI)
        .path("deleteStorageById")
        .queryParam("id", 1)
        .request(MediaType.APPLICATION_JSON)
        .delete();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertEquals("Storage successfully deleted", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");

    response.close();
}
```

Rossz teszt:



```
@Test
public void testDeleteStorageById_withValidId_returnsSuccess() {
    // Feltevésezzük, hogy létezik egy tároló ID-val 1
    Response response = client.target(BASE_URI)
        .path("deleteStorageById")
        .queryParam("id", 9999)
        .request(MediaType.APPLICATION_JSON)
        .delete();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

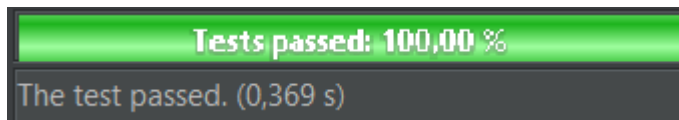
    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertEquals("Storage successfully deleted", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");

    response.close();
}
```


PALLET OSZTÁLY:

testAddPalletToShelf_withValidData_returnsSuccess: Ellenőrzi, hogy a POST /pallet/addPalletToShelf végpont 201-es státuszkóddal és sikerüzenettel hozzáad egy új raklapot egy polchoz érvényes adatokkal.

Jó teszt:



```
@Test
public void testAddPalletToShelf_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("skuCode", "ITEM001")
        .put("shelfId", 1)
        .put("height", 80);

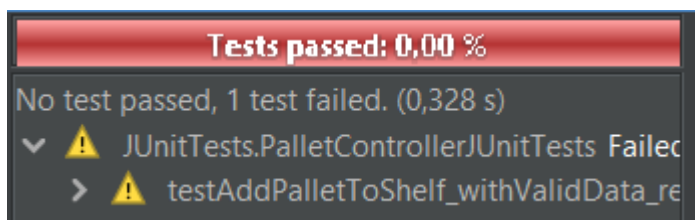
    Response response = client.target(BASE_URI)
        .path("addPalletToShelf")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(201, response.getStatus(), "A státuszhat 201-nek kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(201, responseBody.getInt("statusCode"), "A státuszkódhat 201-nek kell lennie");
    assertEquals("Pallet successfully added to shelf", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");
    assertEquals("ITEM001", responseBody.getString("skuCode"), "A SKU kód helyes kell legyen");

    response.close();
}
```

Rossz teszt:



```
@Test
public void testAddPalletToShelf_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("skuCode", "FAKED001")
        .put("shelfId", 1)
        .put("height", 80);

    Response response = client.target(BASE_URI)
        .path("addPalletToShelf")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

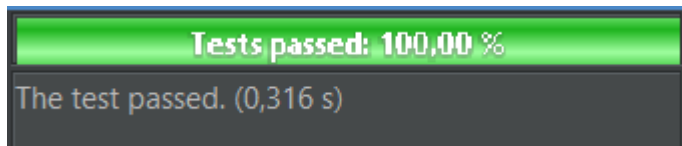
    assertEquals(201, response.getStatus(), "A státuszhat 201-nek kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(201, responseBody.getInt("statusCode"), "A státuszkódhat 201-nek kell lennie");
    assertEquals("Pallet successfully added to shelf", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");
    assertEquals("FAKED001", responseBody.getString("skuCode"), "A SKU kód helyes kell legyen");

    response.close();
}
```


testGetPalletsById_withValidId_returnsPallet: Ellenőrzi, hogy a GET /pallet/getPalletsById végpont 200-as státuszkóddal és a megfelelő raklap adataival tér vissza egy érvényes ID esetén.

Jó teszt:



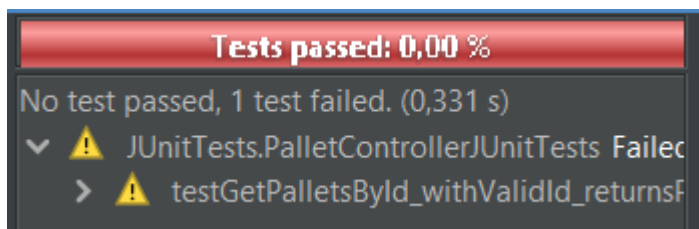
```
@Test
public void testGetPalletsById_withValidId_returnsPallet() {
    Response response = client.target(BASE_URI)
        .path("getPalletsById")
        .queryParams("id", 5)
        .request(MediaType.APPLICATION_JSON)
        .get();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertTrue(responseBody.has("id"), "A válaszban szerepelnie kell az 'id' mezőnek");
    assertEquals(5, responseBody.getInt("id"), "A raklap ID-jának meg kell egyeznie");

    response.close();
}
```

Rossz teszt:



```
@Test
public void testGetPalletsById_withValidId_returnsPallet() {
    Response response = client.target(BASE_URI)
        .path("getPalletsById")
        .queryParams("id", 1)
        .request(MediaType.APPLICATION_JSON)
        .get();

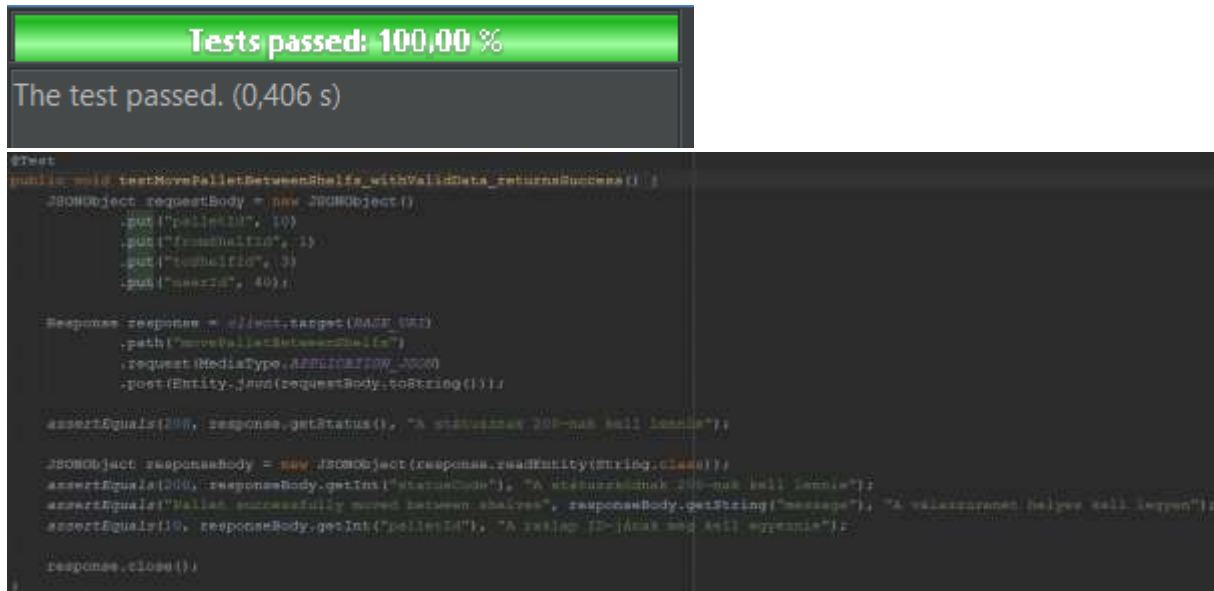
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertTrue(responseBody.has("id"), "A válaszban szerepelnie kell az 'id' mezőnek");
    assertEquals(1, responseBody.getInt("id"), "A raklap ID-jának meg kell egyeznie");

    response.close();
}
```

testMovePalletBetweenShelves_withValidData_returnsSuccess: Ellenőrzi, hogy a POST /pallet/movePalletBetweenShelves végpont 200-as státuszkóddal és sikerüzenettel áthelyez egy raklapot két polc között érvényes adatokkal.

Jó teszt:



```
@Test
public void testMovePalletBetweenShelves_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("palletId", 10)
        .put("fromShelfId", 1)
        .put("toShelfId", 3)
        .put("userId", 40);

    Response response = client.target(BASE_URI)
        .path("/pallet/movePalletBetweenShelves")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertEquals("Raklap sikeresen áthelyezve a polcra", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");
    assertEquals(10, responseBody.getInt("palletId"), "A raklap ID-jének meg kell egyeznie");

    response.close();
}
```

Rossz teszt:



```
@Test
public void testMovePalletBetweenShelves_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("palletId", 1)
        .put("fromShelfId", 1)
        .put("toShelfId", 100)
        .put("userId", 40);

    Response response = client.target(BASE_URI)
        .path("/pallet/movePalletBetweenShelves")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

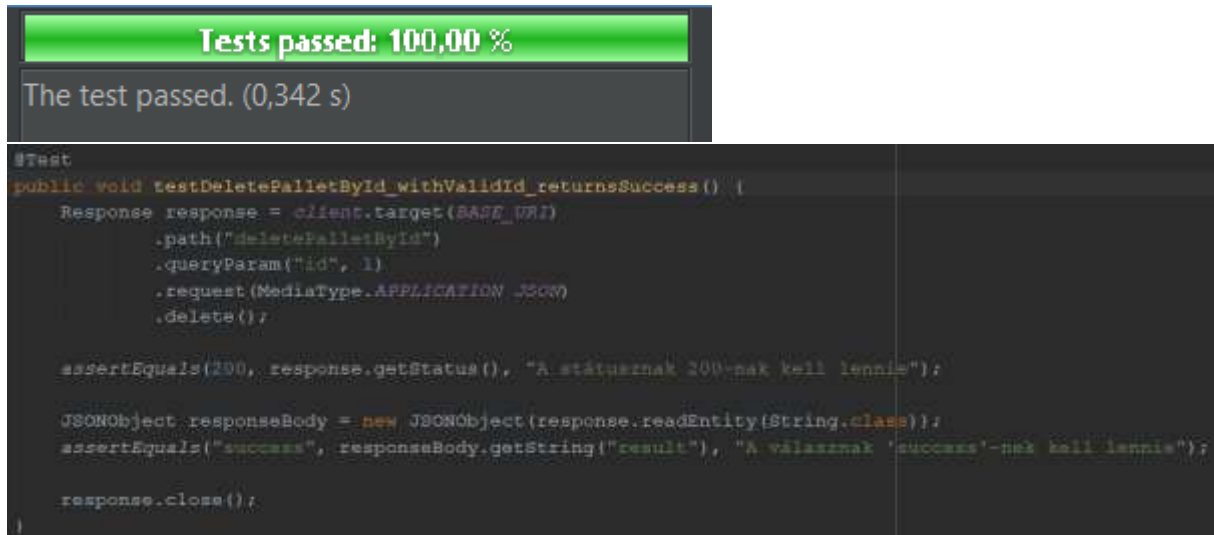
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertEquals("Raklap sikeresen áthelyezve a polcra", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");
    assertEquals(1, responseBody.getInt("palletId"), "A raklap ID-jének meg kell egyeznie");

    response.close();
}
```

testDeletePalletById_withValidId_returnsSuccess: Ellenőrzi, hogy a DELETE /pallet/deletePalletById végpont 200-as státuszkóddal és sikerüzenettel töröl egy létező raklapot érvényes ID alapján.

Jó teszt:



```
#Test
public void testDeletePalletById_withValidId_returnsSuccess() {
    Response response = client.target(BASE_URI)
        .path("deletePalletById")
        .queryParams("id", 1)
        .request(MediaType.APPLICATION_JSON)
        .delete();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("result"), "A válasznak 'success'-nek kell lennie");

    response.close();
}
```

Rossz teszt:



```
#Test
public void testDeletePalletById_withValidId_returnsSuccess() {
    Response response = client.target(BASE_URI)
        .path("deletePalletById")
        .queryParams("id", 1000)
        .request(MediaType.APPLICATION_JSON)
        .delete();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

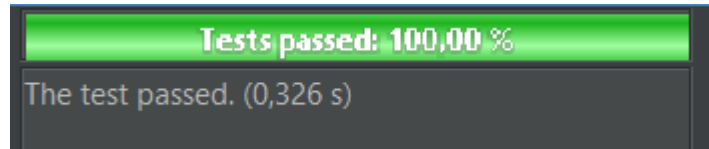
    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals("success", responseBody.getString("result"), "A válasznak 'success'-nek kell lennie");

    response.close();
}
```

ITEMS OSZTÁLY:

testAddItem_withValidData_returnsSuccess: Ellenőrzi, hogy a POST /items/addItem végpont 201-es státuszkóddal és sikerüzenettel hozzáad egy új terméket érvényes adatokkal.

Jó teszt:



```
@Test
public void testAddItem_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("sku", "TESTSKU123")
        .put("type", "wood")
        .put("name", "Test item")
        .put("amount", 10)
        .put("price", 99.99)
        .put("weight", 5.5)
        .put("size", 1.0)
        .put("description", "Test description");

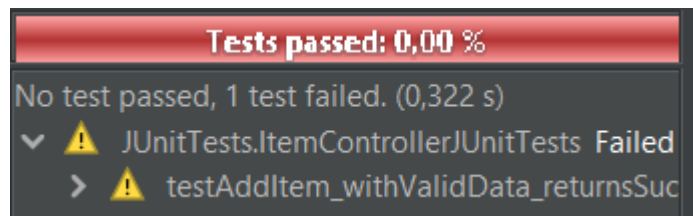
    Response response = client.target(BASE_URI)
        .path("addItem")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(201, response.getStatus(), "A státusznak 201-nek kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(201, responseBody.getInt("statusCode"), "A státuszkódnak 201-nek kell lennie");
    assertEquals("Item successfully added", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");
    assertEquals("TESTSKU123", responseBody.getString("sku"), "A SKU kód helyes kell legyen");

    response.close();
}
```

Rossz teszt:



```
@Test
public void testAddItem_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("sku", "TESTSKU123")
        .put("type", "BAITTER")
        .put("name", "Test item")
        .put("amount", 10)
        .put("price", 99.99)
        .put("weight", 5.5)
        .put("size", 1.0)
        .put("description", "Test description");

    Response response = client.target(BASE_URI)
        .path("addItem")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

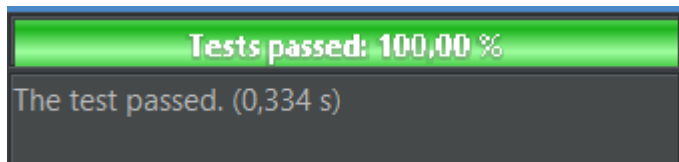
    assertEquals(201, response.getStatus(), "A státusznak 201-nek kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(201, responseBody.getInt("statusCode"), "A státuszkódnak 200-nek kell lennie");
    assertEquals("Item successfully added", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");
    assertEquals("TESTSKU123", responseBody.getString("sku"), "A SKU kód helyes kell legyen");

    response.close();
}
```

testGetItemList_withExistingItems_returnsItemList: Ellenőrzi, hogy a GET /items/getItemList végpont 200-as státuszkóddal és nem üres terméklistával tér vissza meglévő termékek esetén.

Jó teszt:



```
@Test
public void testGetItemList_withExistingItems_returnsItemList() {
    Response response = client.target(BASE_URI)
        .path("getItemList")
        .request(MediaType.APPLICATION_JSON)
        .get();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

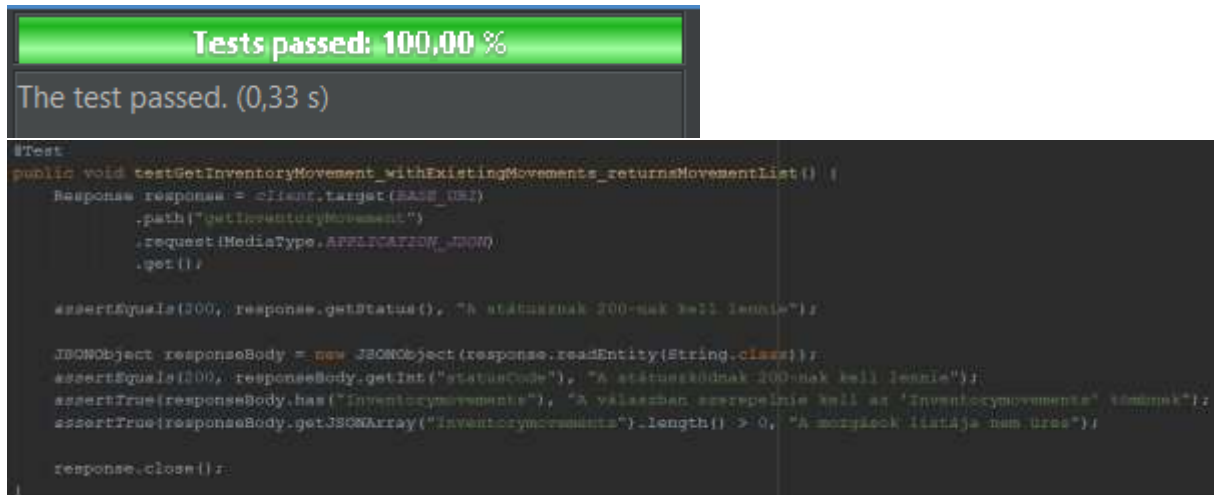
    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertTrue(responseBody.has("items"), "A válaszban szerepelnie kell az 'items' tömbnek");
    assertTrue(responseBody.getJSONArray("items").length() > 0, "A termékek listája nem üres");

    response.close();
}
```

INVENTORY MOVEMENT OSZTÁLY:

testGetInventoryMovement_withExistingMovements_returnsMovementList: Ellenőrzi, hogy a GET /inventorymovement/getInventoryMovement végpont 200-as státuszkóddal és nem üres mozgáslistával tér vissza meglévő mozgások esetén.

Jó teszt:



```
@Test
public void testGetInventoryMovement_withExistingMovements_returnsMovementList() {
    Response response = client.target(BASE_URI)
        .path("getInventoryMovement")
        .request(MediaType.APPLICATION_JSON)
        .get();

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

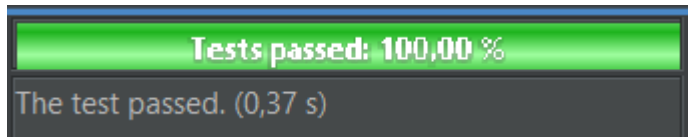
    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertTrue(responseBody.has("inventorymovements"), "A válaszban szerepelnie kell az 'inventorymovements' táblának");
    assertTrue(responseBody.getJSONArray("inventorymovements").length() > 0, "A mozgások listája nem üres");

    response.close();
}
```

MOVEMENT REQUEST OSZTÁLY:

testGetMovementRequests_withExistingRequests_returnsRequestList: Ellenőrzi, hogy a GET /movementrequests/getMovementRequests végpont 200-as státuszkóddal és nem üres kérelem listával tér vissza meglévő kérések esetén.

Jó teszt:



```
@Test
public void testGetMovementRequests_withExistingRequests_returnsRequestList() {
    Response response = client.target(BASE_URI)
        .path("getMovementRequests")
        .request(MediaType.APPLICATION_JSON)
        .get();

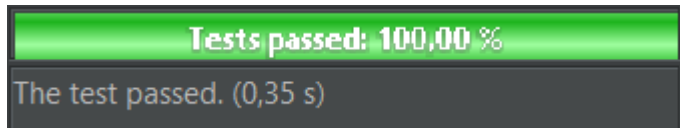
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie!");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie!");
    assertTrue(responseBody.has("MovementRequests"), "A válaszban szerepelnie kell a 'MovementRequests' tételnek");
    assertTrue(responseBody.getJSONArray("MovementRequests").length() > 0, "A kérések listája nem üres");

    response.close();
}
```

testCreateAddMovementRequest_withValidData_returnsSuccess: Ellenőrzi, hogy a POST /movementrequests/createAddMovementRequest végpont 200-as státuszkóddal és sikerüzenettel létrehoz egy új hozzáadási kérelmet érvényes adatokkal.

Jó teszt:



```
@Test
public void testCreateAddMovementRequest_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("adminId", 1)
        .put("palletId", 10)
        .put("toPalletId", 4)
        .put("timeLimit", "2023-04-20 12:00:00");

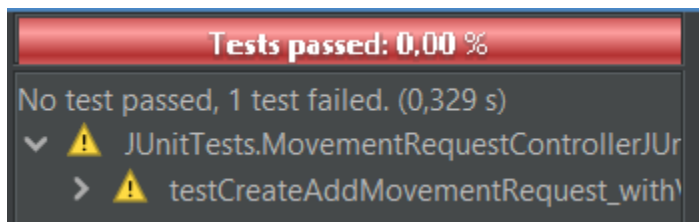
    Response response = client.target(BASE_URI)
        .path("createAddMovementRequest")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertEquals("Add movement request created successfully", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");

    response.close();
}
```

Rossz teszt:



```
@Test
public void testCreateAddMovementRequest_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("adminId", 1000)
        .put("palletId", 1000)
        .put("toPalletId", 400)
        .put("timeLimit", "2023-04-20 12:00:00");

    Response response = client.target(BASE_URI)
        .path("createAddMovementRequest")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

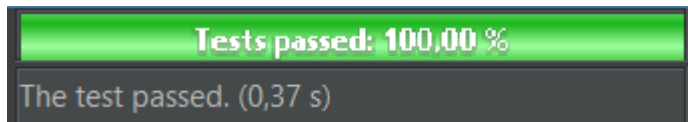
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertEquals("Add movement request created successfully", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");

    response.close();
}
```


testCreateMoveMovementRequest_withValidData_returnsSuccess: Ellenőrzi, hogy a POST /movementrequests/createMoveMovementRequest végpont 200-as státuszköddel és sikerüzenettel létrehoz egy új mozgatósi kérelmet érvényes adatokkal, továbbá ellenőrzi a visszaadott adatokat.

Jó teszt:



```
@Test
public void testCreateMoveMovementRequest_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("adminId", 11)
        .put("palletId", 4)
        .put("fromShelfId", 2)
        .put("toShelfId", 8)
        .put("timestamp", "2023-04-20 12:00:00");

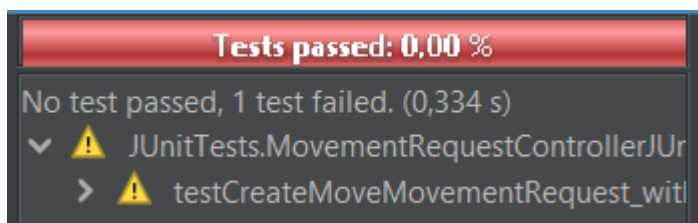
    Response response = client.target(BASE_URI)
        .path("createMoveMovementRequest")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státusz kódnak 200-nak kell lennie");
    assertEquals("Move movement request created successfully", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");
    assertEquals(1, responseBody.getInt("adminId"), "Az adminId helyes kell legyen");
    assertEquals(4, responseBody.getInt("palletId"), "A palletId helyes kell legyen");
    assertEquals(2, responseBody.getInt("fromShelfId"), "A fromShelfId helyes kell legyen");
    assertEquals(8, responseBody.getInt("toShelfId"), "A toShelfId helyes kell legyen");

    response.close();
}
```

Rossz teszt:



```
@Test
public void testCreateMoveMovementRequest_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("adminId", 110000)
        .put("palletId", 4)
        .put("fromShelfId", 2)
        .put("toShelfId", 8)
        .put("timestamp", "2023-04-20 12:00:00");

    Response response = client.target(BASE_URI)
        .path("createMoveMovementRequest")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

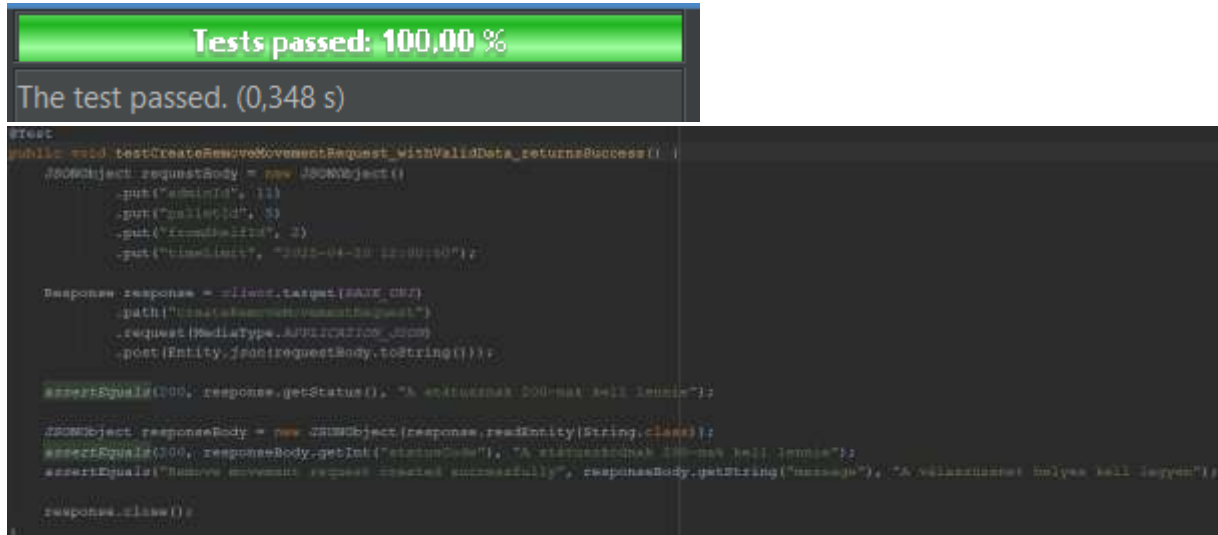
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státusz kódnak 200-nak kell lennie");
    assertEquals("Move movement request created successfully", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");
    assertEquals(110000, responseBody.getInt("adminId"), "Az adminId helyes kell legyen");
    assertEquals(4, responseBody.getInt("palletId"), "A palletId helyes kell legyen");
    assertEquals(2, responseBody.getInt("fromShelfId"), "A fromShelfId helyes kell legyen");
    assertEquals(8, responseBody.getInt("toShelfId"), "A toShelfId helyes kell legyen");

    response.close();
}
```

testCreateRemoveMovementRequest_withValidData_returnsSuccess: Ellenőrzi, hogy a POST /movementrequests/createRemoveMovementRequest végpont 200-as státuszkóddal és sikerüzenettel létrehoz egy új eltávolítási kérelmet érvényes adatokkal.

Jó teszt:



```
Test
Tests passed: 100,00 %
The test passed. (0,348 s)

@Test
public void testCreateRemoveMovementRequest_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("adminId", 1)
        .put("galleryId", 3)
        .put("roomId", 2)
        .put("timestamp", "2023-04-20 12:00:00");

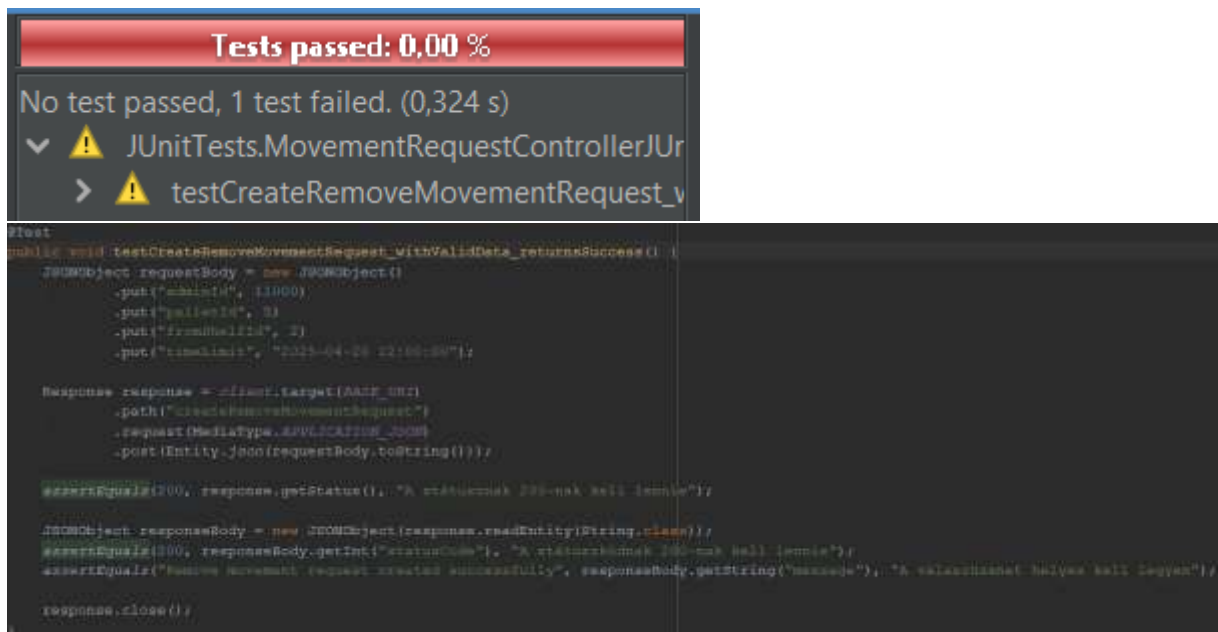
    Response response = client.target(BASE_URI)
        .path("createRemoveMovementRequest")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertEquals("Remove movement request created successfully", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");

    response.close();
}
```

Rossz teszt:



```
Test
Tests passed: 0,00 %
No test passed, 1 test failed. (0,324 s)
▼ ⚠ JUnitTests.MovementRequestControllerJUR
  > ⚠ testCreateRemoveMovementRequest_v

@Test
public void testCreateRemoveMovementRequest_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("adminId", 1000)
        .put("galleryId", 3)
        .put("roomId", 2)
        .put("timestamp", "2023-04-20 12:00:00");

    Response response = client.target(BASE_URI)
        .path("createRemoveMovementRequest")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

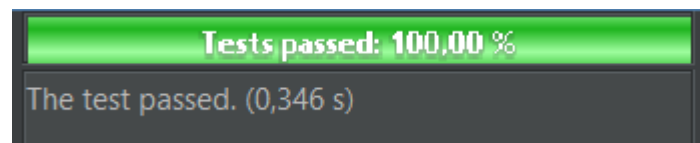
    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státuszkódnak 200-nak kell lennie");
    assertEquals("Remove movement request created successfully", responseBody.getString("message"), "A válaszüzenet helyes kell legyen");

    response.close();
}
```

testCompleteMovementRequest_withValidData_returnsSuccess: Ellenőrzi, hogy a POST /movementrequests/completeMovementRequest végpont 200-as státuszköddal és sikerüzenettel teljesít egy létező kérelmet érvényes adatokkal, továbbá ellenőrzi a visszaadott azonosítókat.

Jó teszt:



```
@Test
public void testCompleteMovementRequest_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("movementRequestId", 6)
        .put("userId", 12);

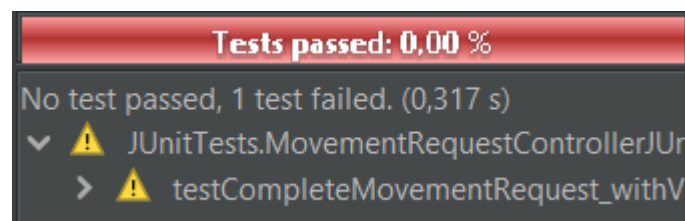
    Response response = client.target(BASE_URI)
        .path("completeMovementRequest")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státusz kódnak 200-nak kell lennie");
    assertTrue(responseBody.getString("message").contains("successfully completed"), "A válaszüzenet helyes kell legyen");
    assertEquals(6, responseBody.getInt("movementRequestId"), "A movementRequestId helyes kell legyen");
    assertEquals(12, responseBody.getInt("userId"), "A userId helyes kell legyen");

    response.close();
}
```

Rossz teszt:



```
@Test
public void testCompleteMovementRequest_withValidData_returnsSuccess() {
    JSONObject requestBody = new JSONObject()
        .put("movementRequestId", 6)
        .put("userId", 12000);

    Response response = client.target(BASE_URI)
        .path("completeMovementRequest")
        .request(MediaType.APPLICATION_JSON)
        .post(Entity.json(requestBody.toString()));

    assertEquals(200, response.getStatus(), "A státusznak 200-nak kell lennie");

    JSONObject responseBody = new JSONObject(response.readEntity(String.class));
    assertEquals(200, responseBody.getInt("statusCode"), "A státusz kódnak 200-nak kell lennie");
    assertTrue(responseBody.getString("message").contains("successfully completed"), "A válaszüzenet helyes kell legyen");
    assertEquals(6, responseBody.getInt("movementRequestId"), "A movementRequestId helyes kell legyen");
    assertEquals(12000, responseBody.getInt("userId"), "A userId helyes kell legyen");

    response.close();
}
```

SELENIUM TESZTEK:

A testLogIn teszt sikeres bejelentkezést ellenőriz helyes felhasználónév és jelszó megadásával.

```
@Test
public void testLogIn() throws InterruptedException {
    Thread.sleep(2000);
    WebElement username = driver.findElement(By.className("login-input"));
    username.sendKeys("asd");

    WebElement password = driver.findElement(By.className("password-input"));
    password.sendKeys("asd");

    driver.findElement(By.className("signIn")).click();
    Thread.sleep(2000);
    String actualResult = driver.findElement(By.tagName("p")).getText();
    String expectedResult = "MAIN";
    Assert.assertEquals(actualResult, expectedResult);
}
```

Tests passed: 100.00 %

The test passed. (0,346 s)

Telepítési útmutató

Készítette:

Nyári Milán Bence

Rendszerkövetelmények

Hardver követelmények

Mivel az alkalmazás egy webalapú rendszer, nincs szükség speciális hardverre a futtatásához. Bármilyen eszközről elérhető és használható, amely képes modern webböngészők futtatására.

A minimális ajánlott rendszerkövetelmények:

- Bármilyen tablet, okostelefon, számítógép
- Stabil internetkapcsolat

Ezenkívül javasolt, hogy az eszköz a legfrissebb böngészőverzióval rendelkezzen a teljebb teljesítmény és felhasználói élmény érdekében.

Szoftver Követelmények

A webes alkalmazás támogatja az alábbi böngészőket:

- Opera GX
- Google Chrome
- Microsoft Edge

Az alkalmazás elérhető és használható, Androidon és Windows alapú eszközökön.

A program telepítése

Fejlesztői Eszközök és Keretrendszerek

A modern szoftvertervezés és fejlesztés során az alkalmazások hatékonysága és funkcionalitása a megfelelő fejlesztői eszközök és keretrendszerek kiválasztásával kezdődik.

Négy kulcsfontosságú eszköz és keretrendszer:

- MAMP
- NetBeans, JDK, Wildfly
- Visual Studio Code
- Angular

Programok telepítése

MAMP Telepítése:

1. Letöltés

- Látogasd meg a MAMP hivatalos letöltési oldalát:
- <https://www.mamp.info/en/downloads/>
- Töltsd le a megfelelő verziót a Windows vagy macOS rendszeredhez.
- Verziószám: 5.0.6



2. Telepítési folyamat

- Nyisd meg a letöltött telepítőfájlt és kövesd az egyszerű lépéseket.
- Válaszd ki a telepítési könyvtárat és telepítsd a szükséges komponenseket.

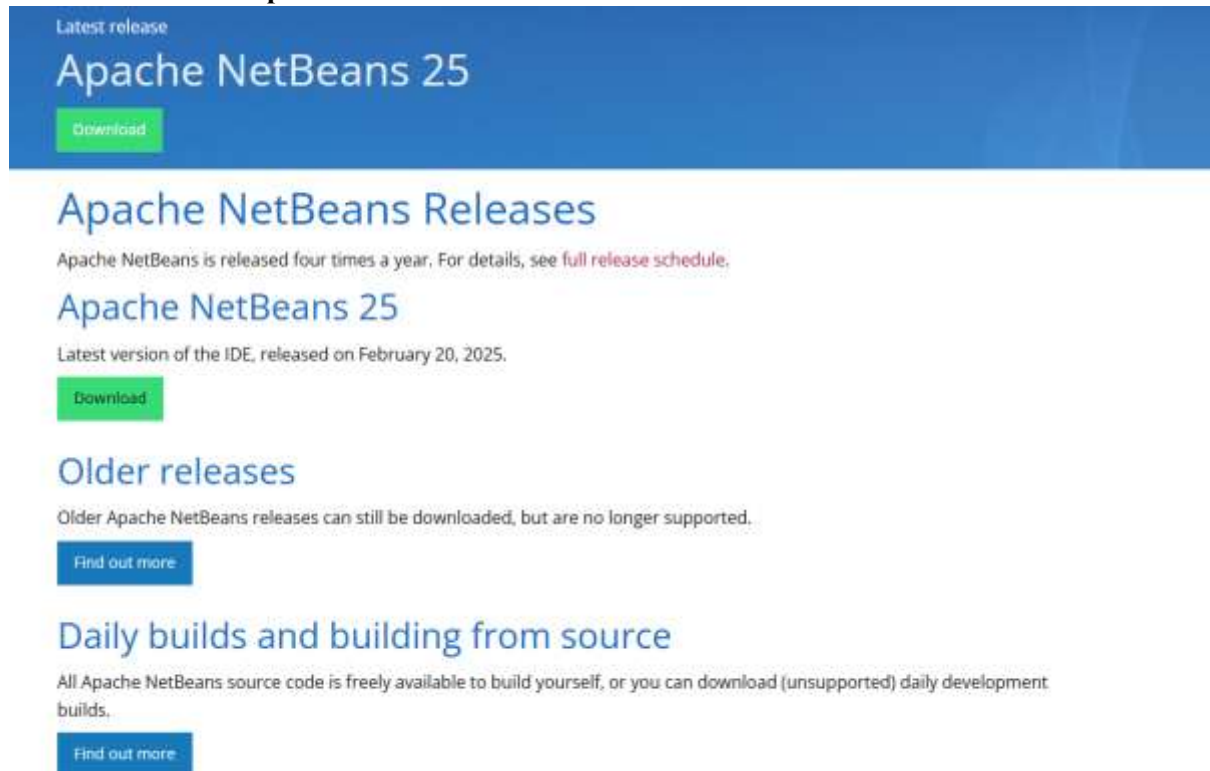
3. Program indítása

- Telepítés után indítsd el a MAMP-ot
- Indításkor vagy automatikusan elindulnak a serverek vagy meg kell nyomni a “Start Servers” gombot.

NetBeans, JDK Telepítése WildFly webszerverrel:

1. Netbeans letöltése

- Látogasd meg a NetBeans hivatalos oldalát a letöltéshez: <https://netbeans.apache.org/front/main/download/index.html>
- Válaszd ki a legfrissebb verziót.
- Verziószám: Apache NetBeans 25



2. Telepítési folyamat

- Nyisd meg a letöltött NetBeans telepítőfájlt és az egyszerű lépéseket követve telepítsd le azt.
- Telepítés során válaszd ki azokat az opciókat amik támogatják a Java EE és a WildFly fejlesztést

3. JDK Telepítése

- Győződj meg róla, hogy a NetBeans számára szükséges JDK (Java Development Kit) már telepítve van a rendszeredben.
- Ha nincs telepítve akkor töltsd le és telepítsd a JDK 21-es verzióját a JDK hivatalos oldaláról: <https://www.oracle.com/java/technologies/downloads/#java21>
- Verziószám: JDK 21

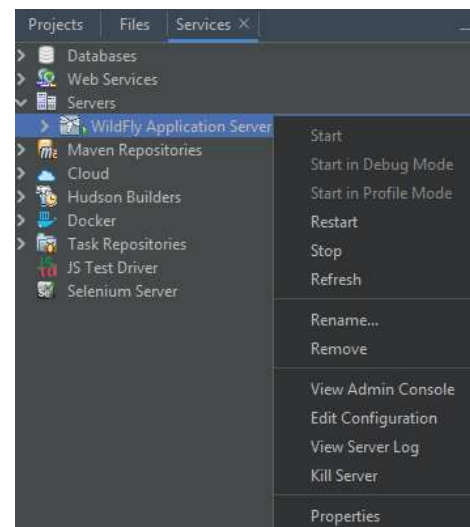
4. WildFly Telepítése

- Látogasd meg a WildFly hivatalos letöltési oldalát: <https://www.wildfly.org/downloads/>
Töltsd le a 26.1.1.Final verziót és csomagold ki az elérési útvonaladra

- Verziószám: 26.1.1.Final

5. NetBeans és WildFly Integráció

- Indítsd el a NetBeans IDE-t.
- Nyisd meg a “Services” fült avagy ablakot, majd a “Servers” mappára kattints jobb gombbal.
- Válaszd a “Add Server” lehetőséget, ezután majd válaszd ki a WildFly alkalmazás szerver verzióját.
- Konfiguráld a szükséges beállításokat, és a NetBeans készen áll a WildFly alkalmazások fejlesztésére.



6. WildFly Management Console Fiók létrehozása

- Nyisd meg a parancssort, majd navigáld a WildFly telepítési könyvtárába.
- A bin mappában található add-user.bat(Windows) vagy add-user.sh(Linux), futtatásával indítsd el a felhasználó létrehozási programot.
- Kövesd a program lépéseit és hozz létre egy felhasználót a WildFly Management Console-hoz.

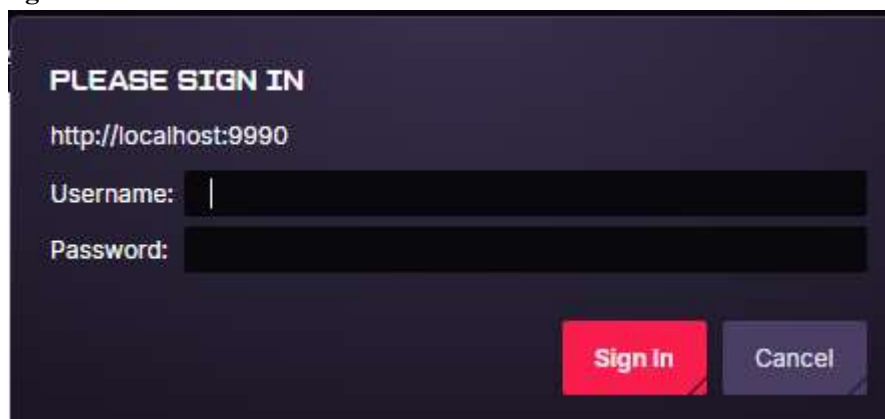
```
C:\wildfly\bin>add-user.bat

What type of user do you wish to add?
a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
(a):

Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the existing property files.
Username : admin
```

7. Belépés a WildFly Management Console-ba

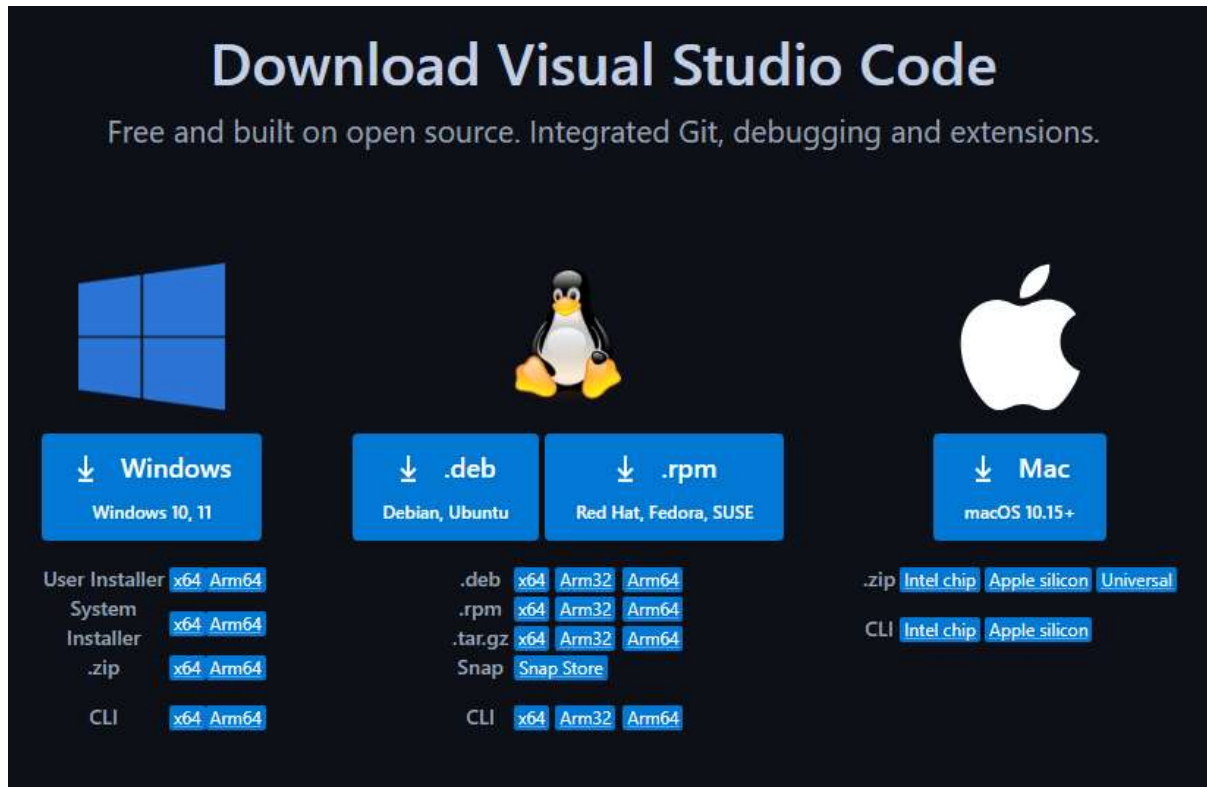
- Indítsd el a WildFly alkalmazás szerveret a netbeans-ben majd látogasd meg a következő címet:
<http://localhost:9990/>
- Használd az előzőleg létrehozott felhasználóneved és jelszavad a Wildfly Management Console eléréséhez.



Visual Studio Code Telepítése

1. Letöltés

- Látogasd meg a Visual Studio Code hivatalos oldalát a letöltéshez:
<https://code.visualstudio.com/download>
- Válaszd ki a neked megfelelő verziót.
- Verziószám: 1.99.3



2. Telepítés

- Nyisd meg a letöltött telepítő fájlt és kövesd az egyszerű lépéseket a sikeres telepítéshez.
- Válaszd ki a kívánt beállításokat(pl. Asztali Ikonok, PATH beállítások, Társítási beállítások)

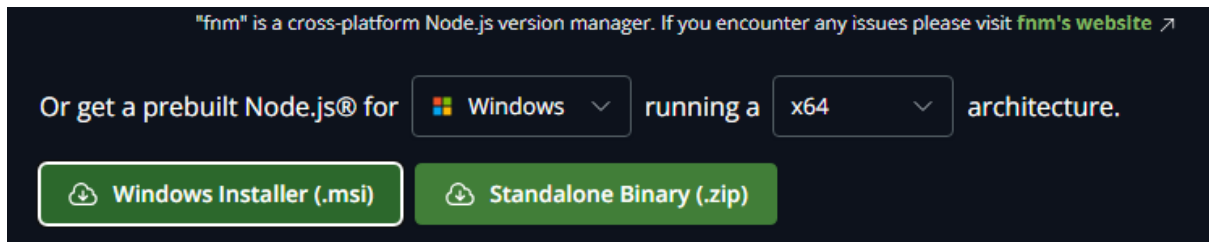
3. Indítás

- Telepítés után csak indítsa el az alkalmazást.

Angular Projekt megnyitása a VS Code segítségével

1. Node.js telepítése

- Az Angular fejlesztéséhez először telepíteni kell a Node.js-t
- Töltsd le és telepítsd fel a legfrissebb Node.js verziót a hivatalos oldaláról: <https://nodejs.org/en/download>
- Verziószám: 22.9.0



2. Angular CLI Telepítése

- Nyisd meg a parancssort vagy terminált.
- Futtasd a következő parancsot a Node.js csomagkezelővel:
`npm install -g @angular/cli`
- Ez telepíti az Angular Command Line Interface-t a rendszeredre

3. Angular Projekt Megnyitása

- Nyisd meg a meglévő munkaterületet a VS Code-ban.
- Nyisd meg a beépített terminált a view menüből, majd válaszd a "Terminal" opciót
- Győződj meg róla hogy command prompt terminált nyiss és ne powershell-t
- Navigálj a projektmappába és a böngészőben való megnyitáshoz írd be az alábbi parancsot: "ng s -o" vagy "ng serve -o"
- Ez a parancs indítja el a fejlesztői szerveret, és megnyitja a projektet a böngészőben, a terminált nyitva kell tartani különben megszakad a server futása.

Programok indítása

Most, hogy minden telepítve lett, Node.js, VS Code, Angular CLI, MAMP, NetBeans, Wildfly és a konfigurálások is megtörténtek, minden készen áll ahhoz hogy elindítsd a programot.

1. Indítsd el a MAMP-ot

- Ez automatikusan elindítja a servereket

2. Indítsd el a NetBeans, WildFly szervert
 - Nyisd meg a NetBeanst
 - Navigálj a projekt mappához és nyisd meg a meglévő projektet.
 - A NetBeans “Services” menüjében indítsd el a WildFly szerveredet
3. Visual Studio Code
 - Nyisd meg a VS Code-ot
 - Navigálj a projekt mappájába vagy nyisd meg a meglévő projektet
4. Angular
 - A VS Code termináljában navigálj a projektmappába
 - Futtasd az `ng serve -o` parancsot a szerver elindításához és a projekt megnyitásához a böngészőben

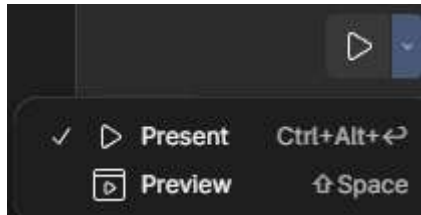
Figma live design dokumentáció

1. Projekt Megnyitása Figma-ban:

- Nyisd meg a Figma alkalmazást a számítógépeden
- Nyisd meg a projektet

2. Prezentációs mód elindítása

- A Figma jobb felső sarkában található „Play” gombra kattints



3. Interaktív Böngészés

- Kattints interaktív elemekre hogy láthatsz az interakciókat és az átmeneteket.

Figma design link:

<https://www.figma.com/design/WkO7hDDf0pFi6iLsfhiQri/Project?node-id=0-1&t=xSYNaSDX3GbnZwWC-1>