#### **A képen szöveg, Grafika, Betűtípus, Grafikus tervezés látható Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.Premontrei Szakgimnázium és Technikum Keszthely**

**„Gumizz, Gumiszervíz webshop” alkalmazás  
Készítette: Séfer Tamás, Csordás Patrik  
13.A/2**

**2025**

Tartalom

[Témaválasztás 4](#_Toc195298396)

[Az alkalmazás funkciói: 4](#_Toc195298397)

[1. Időpontfoglalás kezelése 4](#_Toc195298398)

[2. Raktárkészlet kezelése 4](#_Toc195298399)

[3. Webshop funkció 4](#_Toc195298400)

[4. Felhasználói fiókok kezelése 4](#_Toc195298401)

[Rendszerarchitektúra és Működési Folyamatok 5](#_Toc195298402)

[Használati Példák és Forgatókönyvek 5](#_Toc195298403)

[Adatbázis Struktúra 6](#_Toc195298404)

[Kapcsolatok összegzése 9](#_Toc195298405)

[Relációs adatmodell: 11](#_Toc195298406)

[Frontend Részletes Leírás 11](#_Toc195298407)

[Állapotkezelés és Kontextusok 12](#_Toc195298408)

[Backend API-k és Kommunikáció 12](#_Toc195298409)

[Tesztelés és Hibakezelés 13](#_Toc195298410)

[Felhasználói Élmény és UX szempontok 13](#_Toc195298411)

[Frontend Komponensek Részletes Elemzése 13](#_Toc195298412)

[MyGaragesPage komponens 13](#_Toc195298413)

[ProfilePage komponens 15](#_Toc195298414)

[RegisterForm komponens 18](#_Toc195298415)

[LoginForm komponens 20](#_Toc195298416)

[Homepage komponens 23](#_Toc195298417)

[ShopPage komponens 27](#_Toc195298418)

[Checkout komponens 30](#_Toc195298419)

[GarageAppointmentSchedulePage komponens 33](#_Toc195298420)

[Fontosabb backend elemek leírása 35](#_Toc195298421)

[UsersController elemzése 35](#_Toc195298422)

[InventoryController elemzése 37](#_Toc195298423)

[AppointmentsController elemzése 38](#_Toc195298424)

[Backend tesztelés 41](#_Toc195298425)

[AppointmentsController tesztjei 41](#_Toc195298426)

[InventoryController tesztjei 44](#_Toc195298427)

[UsersController tesztjei 46](#_Toc195298428)

[Frontend tesztelés 48](#_Toc195298429)

[Jövőbeli Fejlesztések és Skálázhatóság 48](#_Toc195298430)

[Összegzés 49](#_Toc195298431)

# Témaválasztás

A szakmai vizsgafeladatunk egy webes alkalmazás fejlesztése, amely egy gumiszerviz működését segíti elő. A projekt célja egy valós problémára adott hatékony, piacképes megoldás kidolgozása. Alapvető elképzelésünk az volt, hogy a gumiszervíz műhelyek által kínált szolgáltatásokat egy rendszerben integráljuk, ezáltal megkönnyítsük a végfogyasztók számára a szolgáltatások keresését, foglalását és igénybevételét, valamint a szervíztulajdonosok számára ezek kezelését. A projekt kiválasztásakor fontos szempont volt, hogy ilyen funkcionalitású szoftverrel még nem találkoztunk a piacon.

# Az alkalmazás funkciói:

## 1. Időpontfoglalás kezelése

Az alkalmazás egyik legfontosabb funkciója az időpontfoglalás rendszerezett kezelése. A felhasználó képes lesz az adott szervizbe időpontot foglalni egy intuitív felület segítségével. A tulajdonos saját kezelőfelületén látja az általa kapott foglalásokat, valamint lehetősége van azokat átütemezni vagy módosítani, ha szükséges.

## 2. Raktárkészlet kezelése

A rendszer lehetővé teszi a szervizek számára a saját raktárkészletük nyilvántartását. A szervíz tulajdonosai kezelhetik a készletet, amely tartalmazza a különböző gumiabroncsokat (évszakok szerint: téli, nyári, négyévszakos), valamint egyéb alkatrészeket, mint pl. csavarokat, szelepeket stb. Az adatok egy MySQL alapú adatbázisban kerülnek tárolásra.

## 3. Webshop funkció

A felhasználók böngészhetnek a rendelkezésre álló termékek között, szűrési lehetőségekkel (pl. méret, évszak). Az általuk kiválasztott termékek felszereltetésére is lehetőség van az időpontfoglalás során.

## 4. Felhasználói fiókok kezelése

* Az ügyfelek regisztrálhatnak és saját felhasználói fiókot hozhatnak létre.
* A szerviz tulajdonosa admin jogokkal rendelkezik, amely lehetővé teszi a készlet- és munkabeosztás-kezelést.
* A felhasználók e-mailes értesítéseket kapnak időpontjaikról, rendeléseikről és egyéb fontos információkról.

# Rendszerarchitektúra és Működési Folyamatok

Az alkalmazás egy modern, kliens-szerver architektúrán alapul, amelyben a backend Node.js segítségével biztosítja az üzleti logikát és az adatok kezelését. A React.js alapú frontend Vite fejlesztőkörnyezetben épül, amely gyors és hatékony fejlesztést tesz lehetővé. Az adatok kezeléséhez a Sequelize ORM-et használjuk, amely MySQL adatbázishoz kapcsolódik. A rendszer fejlesztése és tesztelése során Docker konténerizációt alkalmazunk, míg az adatbáziskezeléshez Beekeeper Studio szoftvert használunk. A verziókezelést és csapatmunkát a Git kliensprogram és GitHub segítségével végezzük, míg a teljes fejlesztési folyamatot Visual Studio Code környezetben hajtjuk végre. Fejlesztésünkben figyelembe vettük, hogy olyan technológiákat használjunk, amelyek a Képzési és Kimeneti Követelmények-ben található dokumentumban fel voltak tűntetve. Ennek megfelelően alkalmaztunk MVC-t(model-view-controller) és REST API-t is.

# Használati Példák és Forgatókönyvek

1. **Ügyfél időpontfoglalása** – Az ügyfél kiválasztja a kívánt szervizt és a megfelelő időpontot, majd megerősíti a foglalását. A rendszer automatikusan értesítést küld az időpontról. Emellett a felhasználó profiljában nyomon követheti a foglalásokat. A szerviz tulajdonosának lehetősége van a foglalt időpontot jóváhagyni, valamint törölni, ha szükséges.
2. **Webshop vásárlás** – Az ügyfél böngészi a rendelkezésre álló termékeket, kiválasztja a kívánt méretet és típust, majd időpontot foglal a felszerelésre.
3. **Szerviztulajdonos funkciói** – A szerviztulajdonosok ellenőrizhetik és kezelhetik a beérkező időpontfoglalásokat a szervizeikhez, valamint optimalizálhatják a munkabeosztást. Emellett módosíthatják a feltöltött készletet (pl. képek feltöltése, leírás módosítása stb.) vagy a szerviz adatait (pl. Nyitvatartás, elérhetőség stb.).

Adatbázis Struktúra  
Az adatbázis tervezése során célunk egy jól strukturált és könnyen skálázható rendszer létrehozása volt, amely biztosítja a gumiszerviz hatékony működését. Az alábbiakban bemutatjuk az adatbázis egyes tábláit és azok kapcsolatait.

**Felhasználók (users)**

**Szerepe:**

* Tárolja az alkalmazás felhasználóit, akik lehetnek vásárlók (customer), garázstulajdonosok (garage\_owner) vagy adminok (admin).
* Minden felhasználónak egyedi e-mail címe és szerepköre van.

**Kapcsolatai:**

* Egy garázsnak lehet egy tulajdonosa (garages.owner\_id → users.id).
* Egy felhasználó rendelhet termékeket (orders.user\_id → users.id).
* Egy felhasználó foglalhat időpontot (appointments.user\_id → users.id).
* Egy felhasználó tarthat tételeket a kosárban (cart.user\_id → users.id).

**Garázsok (garages)**

**Szerepe:**

* Tárolja az egyes autószervizeket és azok elérhetőségeit.
* Minden garázs egy tulajdonoshoz (felhasználóhoz) van kötve.

**Kapcsolatai:**

* Egy garázs a tulajdonosán keresztül kapcsolódik a users táblához (garages.owner\_id → users.id).
* Egy garázs készletet tart (inventory.garage\_id → garages.id).
* Egy garázshoz be lehet jelentkezni időpontfoglalásra (appointments.garage\_id → garages.id).
* Egy rendelés egy adott garázs szolgáltatásait tartalmazza (orders.garage\_id → garages.id).

**Készlet (inventory)**

**Szerepe:**

* Tárolja a garázsok raktárkészletét
* Tartalmaz egy vehicle\_type mezőt, amely segít a járműtípus szerinti szűrésben.

**Kapcsolatai:**

* Egy adott garázshoz tartozik (inventory.garage\_id → garages.id).
* A webshopban megjelenik, ha a felhasználó kiválaszt egy garázst.
* A kosárhoz kapcsolódik (cart\_items.product\_type = 'inventory').
* A rendelési tételek között is szerepel (order\_items.product\_type = 'inventory').

**Kosár (cart)**

**Szerepe:**

* Tárolja a felhasználók kosarát, amely egy adott garázshoz kötött.
* A felhasználó mindig csak egy garázs kínálatából vásárolhat egyszerre.

**Kapcsolatai:**

* Egy adott felhasználóhoz tartozik (cart.user\_id → users.id).
* Egy adott garázshoz kapcsolódik (cart.garage\_id → garages.id).
* A cart\_items táblában tárolódnak az egyes kosártételek.

**Kosár tételek (cart\_items)**

**Szerepe:**

* Tárolja a kosárba helyezett tételeket, pl. készletben lévő termék.
* Meghatározza, hogy hány darabot rendelt a felhasználó.

**Kapcsolatai:**

* Egy adott kosárhoz tartozik (cart\_items.cart\_id → cart.id).
* A product\_type mező alapján hivatkozhat inventory táblákra.

**Rendelések (orders)**

**Szerepe:**

* Tárolja a felhasználók által véglegesített rendeléseket.
* Tartalmazza a teljes árat és a rendelés állapotát.

**Kapcsolatai:**

* Egy adott felhasználóhoz tartozik (orders.user\_id → users.id).
* Egy adott garázshoz kapcsolódik (orders.garage\_id → garages.id).
* A rendelés tételei az order\_items táblában tárolódnak.
* Egy időpontfoglalás egy adott rendeléshez kapcsolódik (appointments.order\_id → orders.id).

**Rendelési tételek (order\_items)**

**Szerepe:**

* Tárolja a megrendelésekhez tartozó egyes tételeket, termékeket.
* Tartalmazza a rendelési mennyiséget és az egységárat.

**Kapcsolatai:**

* Egy rendeléshez kapcsolódik (order\_items.order\_id → orders.id).
* A product\_type mező alapján hivatkozhat inventory táblákra.

**Időpontfoglalások (appointments)**

**Szerepe:**

* Tárolja az időpontfoglalásokat, amelyeket a felhasználók a checkoutnál választanak.
* Összekapcsolja a felhasználót, a garázst és a rendelést.

**Kapcsolatai:**

* Egy adott felhasználóhoz tartozik (appointments.user\_id → users.id).
* Egy adott garázshoz kapcsolódik (appointments.garage\_id → garages.id).
* Egy adott rendeléshez kötött (appointments.order\_id → orders.id).

**Garázs Időbeosztás (GarageScheduleSlot)**

**Szerepe:**

* Tárolja a garázsok időbeosztását és a rendelkezésre álló időpontokat.
* Meghatározza, hogy egy garázs mikor fogad ügyfeleket a hét különböző napjain.
* Lehetővé teszi a foglalási rendszer működését azáltal, hogy definiálja a foglalható időintervallumokat.
* Korlátozza az egy időpontban fogadható ügyfelek számát a max\_bookings mezővel.

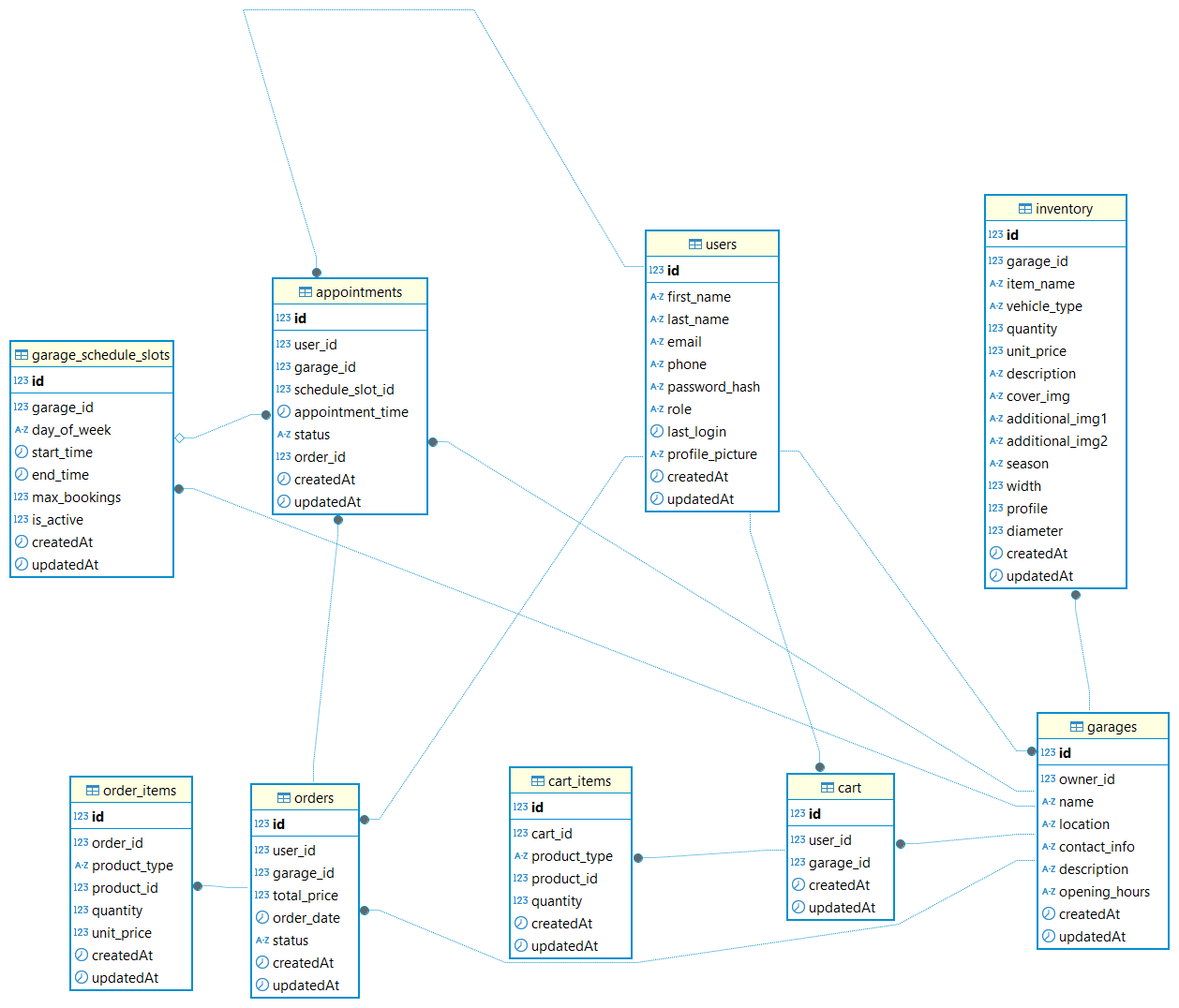
**Kapcsolatai:**

* Egy adott garázshoz tartozik (garage\_schedule\_slots.garage\_id → garages.id)
* Időpontfoglalások hivatkozhatnak rá (appointments.schedule\_slot\_id → garage\_schedule\_slots.id)

## Kapcsolatok összegzése

|  |  |  |
| --- | --- | --- |
| **Tábla** | **Kapcsolódik hozzá** | **Kapcsolat típusa** |
| **users** | garages, orders, cart, appointments | egy-a-többhöz |
| **garages** | users, inventory, orders, appointments | egy-a-többhöz |
| **inventory** | |  | | --- | |  |  |  | | --- | | garages, cart\_items, order\_items | | egy-a-többhöz |
| **cart** | |  | | --- | |  |  |  | | --- | | users, garages, cart\_items | | egy-a-többhöz |
| **cart\_items** | cart, inventory | egy-a-többhöz |
| **orders** | users, garages, order\_items, appointments | egy-a-többhöz |
| **order\_items** | orders, inventory | egy-a-többhöz |
| **appointments** | users, garages, orders | egy-a-többhöz |
| **GarageScheduleSlot** | garages | egy-a-többhöz |

## Relációs adatmodell:

****

Az adatbázis relációs szerkezete biztosítja a konzisztenciát, valamint lehetővé teszi a gyors adatlekérdezéseket és módosításokat. Az egyes rekordok között létrejövő kapcsolatok biztosítják a rendszer hatékony és átlátható működését.

# Frontend Részletes Leírás

Az alkalmazás frontendje modern, reszponzív webes felületet biztosít, amely React.js és Tailwind CSS technológiákra épül. A vizuális megjelenéshez és az interaktivitáshoz a Framer Motion animációs könyvtár is felhasználásra került.

Főbb oldalak és funkciók:

- Főoldal: Interaktív nyitóoldal, kiemelt termékekkel és szervizajánlással.

- Webshop: Garázshoz kötött terméklista szűrési lehetőségekkel.

- Termékoldal: Részletes termékinformáció és vásárlási lehetőség.

- Bejelentkezés/Regisztráció: Modál formában megjelenő hitelesítés.

- Profil: Foglalások, rendelések megtekintése.

- Admin felületek: Garázs és készlet kezelése, időpontok és rendelések adminisztrációja.

- Checkout: Időpontfoglalással egybekötött rendelés véglegesítése.

# Állapotkezelés és Kontextusok

Az alkalmazás globális állapotkezeléséhez React Context API-t használunk:

- ThemeContext: Világos/sötét mód támogatás.

- CartContext: Kosár állapotának kezelése (termékek, mennyiség, garázs kapcsolás).

# Backend API-k és Kommunikáció

A frontend Axios könyvtár segítségével kommunikál a Node.js alapú backenddel. A legfontosabb végpontok:

- /inventory – Terméklista lekérése.

- /garages – Elérhető szervizek.

- /login és /register – Hitelesítés.

- /cart, /cart/items – Kosárműveletek.

- /appointments – Időpontfoglalás kezelése.

- /orders – Rendelések mentése és listázása.

# Tesztelés és Hibakezelés

A rendszer automatikusan kijelentkezteti a felhasználót, ha a JWT token lejárt.

Betöltési folyamat alatt animáció jelenik meg ('Betöltés...').

Hálózati hibák konzolban kerülnek naplózásra, a jövőben felhasználói értesítésekkel bővíthető.

# Felhasználói Élmény és UX szempontok

- Animációk használata a betöltések és interakciók során (Framer Motion).

- Teljes reszponzivitás (Tailwind CSS).

- Egyszerű navigáció, azonnali kosár-elérés és kiemelt ajánlatok.

# Frontend Komponensek Részletes Elemzése

## MyGaragesPage komponens

A MyGaragesPage komponens a garázstulajdonosok saját garázsainak kezelésére szolgáló felület, amely a következő funkciókat biztosítja:

**Fő funkcionalitások:**

* **Garázsok listázása**: A komponens megjelenít minden olyan garázst, amelynek a bejelentkezett felhasználó a tulajdonosa. Ezek a garázsok a szervertől API-hívással érkeznek.
* **Új garázs hozzáadása**: A felhasználó az oldal tetején található gombbal egy űrlapot nyithat meg, ahol megadhatja az új garázs részleteit.
* **Garázsok kezelése**: Minden megjelenített garázshoz több kezelőgomb tartozik:
  + A képen szöveg, képernyőkép, szoftver, Weblap látható

    Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.Készlet kezelése: Átirányít a garázs készletének kezelőfelületére
  + Rendelések: Átirányít a garázshoz tartozó rendelések oldalára
  + Szerkesztés: A garázs adatainak módosítása
  + Időpontok: A garázshoz tartozó foglalások megtekintése és kezelése

**Technikai megvalósítás:**

* A komponens React Hooks-ra épül (useState, useEffect), amelyek segítségével kezeli az állapotokat és életciklus-eseményeket.
* Axios segítségével végez API-hívásokat a backend felé.
* A felhasználói élmény fokozására a Framer Motion animációs könyvtárat használja.
* A felhasználói fiók hitelesítését a localStorage-ból kinyert JWT tokennel végzi.
* Reszponzív felület Tailwind CSS segítségével, amely különböző képernyőméretekhez igazodik.

**Állapotkezelés:**

* garages: A felhasználó garázsait tárolja
* loading: Betöltési állapot jelzésére
* error: Hibaüzenetek tárolására
* showAddGarageForm: Az új garázs űrlap megjelenítésének vezérlésére
* newGarage: Az új garázs adatainak tárolására
* searchQuery: Keresési lekérdezés tárolására

**Kontextusok használata:**

* ThemeContext: Világos/sötét mód beállításához
* CartContext: Kosár állapotának kezeléséhez

**Komponens folyamat:**

1. Oldal betöltésekor ellenőrzi a felhasználói jogosultságokat
2. Ha a felhasználó nem garázstulajdonos, átirányítja a főoldalra
3. Lekéri a garázsok listáját a szerverről
4. Szűri a garázsokat a felhasználói azonosító alapján
5. Megjeleníti a garázsokat vagy a megfelelő üzenetet, ha nincsenek garázsok

## ProfilePage komponens

**Alapstruktúra és funkciók**

A komponens egy komplex felhasználói profil felületet biztosít, amely a következő főbb funkciókat tartalmazza:

* Felhasználói adatok megjelenítése és szerkesztése
* Profilkép kezelése (feltöltés, megjelenítés)
* Jelszóváltoztatás
* Fiók törlése
* Felhasználói tevékenységek nyomon követése (rendelések, időpontfoglalások)

**Állapotkezelés**

A komponens számos React állapotot használ a különböző funkcionalitások kezeléséhez:

* Felhasználói adatok tárolása (profileData)
* Szerkesztési mód jelzése (editMode)
* Modálablakok megjelenítése (showPasswordModal, showDeleteModal)
* Jelszóadatok és validáció (passwordData, validations)
* Felhasználói profilkép (profilePicture)
* Üzenetkezelés különböző műveletekhez (message, passwordMessage, deleteMessage)
* Felhasználói rendelések és időpontfoglalások (userOrders, userAppointments)

**API kommunikáció**

A komponens számos API-végponttal kommunikál:

1. Felhasználói adatok lekérése (/api/users/${userId})
2. Profilkép feltöltése (/api/users/${parsedUserData.id}/profile-picture)
3. Profil frissítése (/api/users/${parsedUserData.id})
4. Jelszó módosítása (/api/users/${parsedUserData.id}/change-password)
5. Fiók törlése (/api/users/${parsedUserData.id})
6. Rendelések lekérése (/orders/user/${userId})
7. Időpontfoglalások lekérése (/appointments/user/${userId})

**Kontextus használat**

A komponens több React kontextust használ:

* ThemeContext - sötét/világos mód kezelésére
* CartContext - kosár kezelésére (kijelentkezéskor)

**UI/UX elemek**

* Reszponzív elrendezés (mobile-first megközelítéssel)
* Animációk a felhasználói élmény javítására (framer-motion könyvtárral)
* Modálablakok jelszómódosításhoz és fiók törléséhez
* Feltételesen megjelenő UI elemek (pl. szerkesztési mód)
* Visszajelzések a felhasználói műveletekről (üzenetek)
* Theme-alapú megjelenítés (sötét/világos mód támogatás)

**Biztonsági funkciók**

1. Jelszó komplexitás validálása:
   * Minimum 8 karakter hosszúság
   * Nagybetűk megléte
   * Számok megléte
   * Speciális karakterek megléte
2. Fiók törlés megerősítése ("TÖRLÉS" szó beírásával)
3. Token-alapú hitelesítés az API kérésekhez

**Főbb komponens szekciók**

1. **Fejléc és navigáció**: A felhasználó navigációs lehetőségeit biztosítja
2. **Profil adatok kártya**: Felhasználó alapadatait és profilképét jeleníti meg
3. **Személyes adatok szekció**: Név, email, telefonszám megjelenítése/szerkesztése
4. **Fiókbeállítások szekció**: Fiók létrehozásának dátuma, utolsó bejelentkezés
5. **Aktivitás szekció**: Időpontfoglalások és rendelések megjelenítése
6. **Modális ablakok**: Jelszómódosításhoz és fiók törléséhez

**Technikai jellemzők**

* Funkcionális komponens alapú megközelítés
* React hookokat használ (useState, useEffect, useRef, useNavigate)
* Formázott dátum- és időmegjelenítés (formatDate, formatTime segédfüggvényekkel)
* Feltételesen alkalmazott CSS osztályok
* Animációk a framer-motion könyvtárral
* Lokális tárolás (localStorage) felhasználói adatok és token tárolására

**Potenciális fejlesztési területek**

1. **Állapotkezelés optimalizálása**: A túl sok useState hook refaktorálása egyetlen useReducer-re.
2. **Hibakezelés**: A try-catch blokkok ismétlődő kódjainak kiszervezése.
3. **API hívások centralizálása**: Az API hívások egy közös modulba kiszervezhetők, a kód duplikáció elkerülése érdekében.

## RegisterForm komponens

**Általános áttekintés**

A komponens egy animált modal ablakban megjelenő regisztrációs űrlapot tartalmaz magyar nyelven, amely támogatja a sötét/világos módot, alapos jelszó validációval rendelkezik, és a háttérben egy API-t hív meg a regisztrációhoz.

**Fő funkciók és jellemzők**

**1. State kezelés**

* Számos state változót használ:
  1. formData: A felhasználói adatok tárolására
  2. validations: Jelszó komplexitási validációkhoz
  3. error és success: Hibaüzenetek és sikeres műveletek jelzésére
  4. localIsOpen és egyéb UI-vezérlő state-ek

**2. Form validáció**

* Részletes jelszó validáció:
  1. Minimum hossz (8 karakter)
  2. Számjegyek jelenléte
  3. Speciális karakterek ellenőrzése
  4. Nagybetű ellenőrzés
* Név validáció (minimum 2 karakter hosszúságú)
* Jelszavak egyezésének ellenőrzése

**3. UI jellemzők**

* Framer Motion animációk a modal nyitás/zárás effektekhez
* Téma érzékelés (sötét/világos mód)
* Jelszó mutatás/elrejtés funkció
* Interaktív jelszó követelmény ellenőrzőlista visszajelzéssel
* Felhasználói szerep kiválasztási lehetőség (customer/garage\_owner)

**4. API integráció**

* fetch API használata a regisztrációs adatok küldéséhez
* Hibakezelés és sikeres regisztráció kezelése

**5. Egyéb funkciók**

* Modál ablak zárása és átirányítás bejelentkezési ablakra
* Görgetés blokkolása, amikor a modál nyitva van (jobb UX érdekében)
* Feltételes CSS osztályok a dark/light módhoz

**Technikai részletek**

**Használt könyvtárak és komponensek**

* React hooks (useState, useEffect)
* Framer Motion animációkhoz
* Egyéni useTheme hook a téma kezelésére
* UI komponensek (pl. Button)

**Validációs logika**

A komponens részletes validációs logikával rendelkezik, különösen a jelszavakra vonatkozóan:

* A validatePasswordComplexity függvény ellenőrzi a jelszó erősségét
* Valós idejű visszajelzés az input mezők változásakor

**Felhasználói élmény optimalizálása**

* Görgetés letiltása modális ablak megjelenítésekor
* Animációk a modál nyitásához/zárásához
* Vizuális visszajelzés a jelszavak egyezésénél

**API kommunikáció**

* RESTful API végponttal kommunikál (http://localhost:3000/api/register)
* POST kérés a regisztrációs adatokkal
* Hibakezelés és sikeres regisztráció esetén átirányítás

**Érdekes pontok/Megfigyelések**

1. A komponens magyar nyelvű felhasználói felületet kínál
2. A garage\_owner szerepkör választási lehetőség autószerviz-specifikus alkalmazásra utal
3. Gondosan kezeli a görgetést a modális ablak működése során
4. A jelszóerősség-ellenőrzés vizuálisan visszajelez
5. Autofill stílusok kezelése mindkét színsémában

## LoginForm komponens

**Alapvető struktúra és funkciók**

A LoginForm egy modális ablakként jelenik meg, amelynek láthatóságát az isOpen prop szabályozza. Fő funkciói:

* Felhasználói bejelentkezés kezelése API hívással
* Űrlapadatok kezelése (email, jelszó)
* Hibaüzenetek és sikeres bejelentkezés visszajelzése
* Animált megjelenés és eltűnés
* Sötét/világos téma támogatása

**Fontosabb funkcionális részletek**

1. **State kezelés**: Több állapotot kezel:

* formData: A bejelentkezési adatok tárolása
* error és success: Visszajelzés állapota
* localIsOpen: Az animáció vezérléséhez
* showPassword: Jelszó láthatóságának kapcsolója

1. **API kommunikáció**: A handleSubmit függvény HTTP POST kérést küld a bejelentkezési adatokkal a szerver felé.
2. **Animációk**: A Framer Motion könyvtárat használja az elegáns megjelenéshez és eltűnéshez.
3. **Témaváltás**: A useTheme hookban lévő darkMode állapottól függően módosítja a megjelenést.
4. **Navigáció**: Sikeres bejelentkezés után a React Router navigate hookját használja a kezdőlapra irányításhoz.

**Technikai megoldások**

**Használt könyvtárak és hook-ok**

* **React hooks**: useState az állapotkezeléshez
* **React Router**: useNavigate az oldalak közötti navigációhoz
* **Framer Motion**: Animációk kezeléséhez
* **Egyedi hook**: useTheme a sötét/világos téma kezeléséhez

**Animációk**

A komponens két szinten használ animációt:

* A háttér (backdrop) fokozatos megjelenése/elhalványulása
* A modális ablak tartalmának méretezése és pozícionálása

**Űrlapkezelés**

* A bemeneteket a handleChange függvény kezeli, amely az állapotot frissíti
* A handleSubmit függvény végzi a validálást és a bejelentkezési kísérletet
* Biztonsági funkció: jelszó láthatóságának kapcsolása

**Stílusok**

* Tailwind CSS osztályokat használ a stílusokhoz
* A sötét/világos téma váltáshoz dinamikusan módosítja a CSS osztályokat
* Különleges kezelés az auto-fill inputokhoz, hogy a sötét témában is megfelelően működjenek

**UX megfontolások**

1. **Visszajelzések**:

* Hibaüzenetek jelennek meg sikertelen bejelentkezés esetén
* Sikeres bejelentkezés után megerősítő üzenet jelenik meg
* Animált átmenetek az állapotváltozások között

1. **Akadálymentesség**:

* A jelszóbeviteli mező típusa váltható a tartalom megjelenítéséhez/elrejtéséhez
* A gombokhoz ikonok tartoznak a jobb érthetőség érdekében

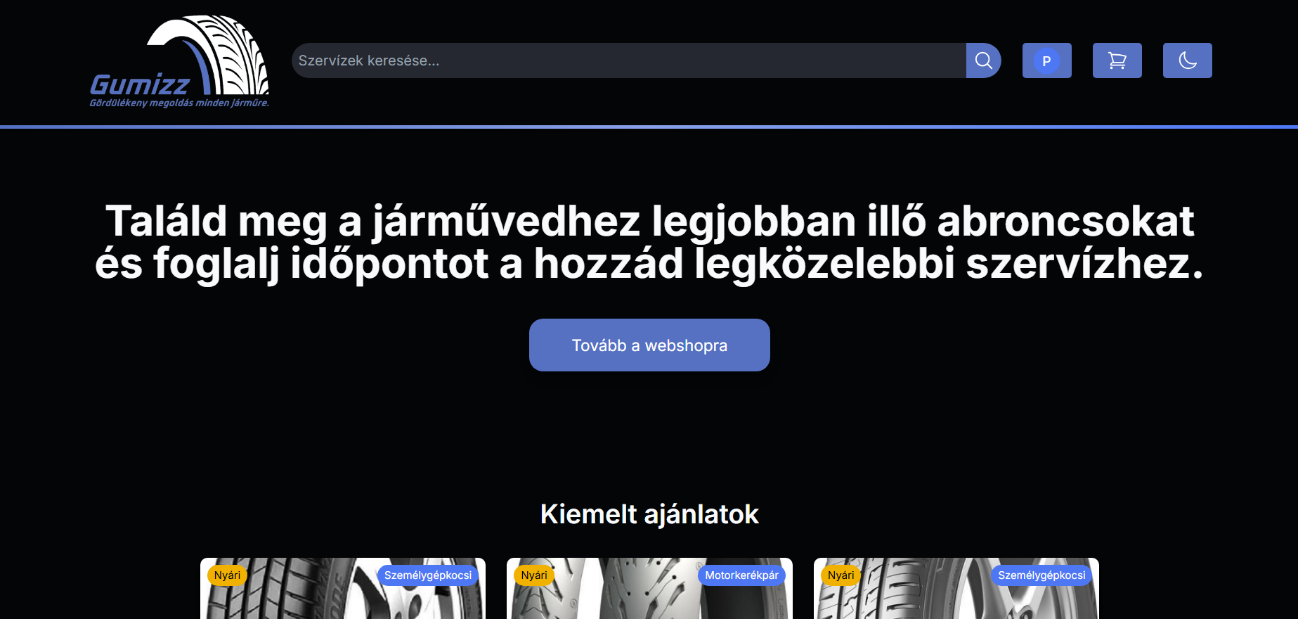
1. **Felhasználói folyamat**:

* Egyszerű váltás a regisztrációs űrlapra
* A sikeres bejelentkezés után automatikus átirányítás
* A modal egyszerűen bezárható

**Területek, amelyek fejleszthetők**

1. **Hibakezelés**: A catch blokk ugyan naplózza a hibát a konzolra, de a felhasználó számára csak általános hibaüzenetet jelenít meg.
2. **Validáció**: Az űrlap alapvető validációt használ (required attribútumok), de összetettebb validáció nincs beépítve.
3. **Hozzáférhetőség**: Nincsenek kifejezett ARIA attribútumok vagy billentyűzetes navigációs kezelők.
4. **Biztonsági megfontolások**: A jelszó helyi állapotban tárolódik, ami a legjobb gyakorlatok szerint történik, de nincs automatikus időkorlát a bejelentkezéshez.

## Homepage komponens

**Főbb jellemzők és funkciók**

**Importok és függőségek**

* **React és hook-ok**: useState, useEffect
* **UI komponensek**: Button, Header, CartSidebar, ProductCard, Footer
* **Kiegészítő könyvtárak**: framer-motion (animációkhoz), axios (HTTP kérésekhez), react-router-dom (navigációhoz)
* **Context rendszerek**: ThemeContext, CartContext

**Állapotkezelés**

A komponens számos állapotot kezel:

* searchQuery: keresési kifejezés tárolása
* featuredProducts: kiemelt termékek listája
* loading: betöltési állapot jelzője
* garages: szervizek/garázsok listája
* isCartOpen: kosár megjelenítésének állapota

**API interakciók**

* **Termékadatok lekérése**: A useEffect hook-on belül az axios segítségével lekéri a termékeket és a garázsokat a backend API-tól
* **Kosárműveletek**: Termékek hozzáadása a kosárhoz API-kérésekkel

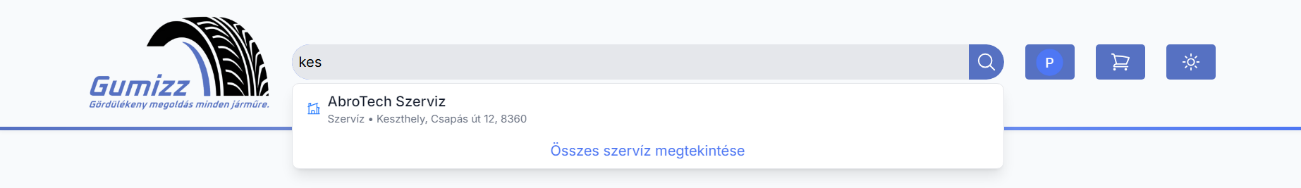
**Felhasználói élmény**

* **Témamód kezelés**: világos és sötét téma támogatása
* **Animációk**: framer-motion könyvtárral implementált animációk
* **Betöltési állapotok kezelése**: placeholder elemek a betöltés során

**Autentikáció**

* **Bejelentkezés/regisztráció**: Képességek felhasználói belépésre és regisztrációra
* **Kijelentkezés**: Felhasználó kijelentkeztetése és a kosár adatainak törlése

**Komponens felépítése**

**Header**

* Navigációs sáv
* Keresőmező
* Bejelentkezési/regisztrációs gombok
* Kosár gomb

**Fő tartalom szakaszok**

1. **Bevezető banner**: Nagybetűs szlogen és call-to-action gomb a webshopra
2. **Kiemelt ajánlatok**: 3 véletlenszerűen kiválasztott termék megjelenítése
3. **Szolgáltatások**: 3 különböző szolgáltatás rövid bemutatása
4. **Vásárlói vélemények**: Felhasználói visszajelzések

**Footer**

* Lábrész komponens

**Technikai megvalósítás részletei**

**Inicializálás és adatbetöltés**

A komponens betöltődésekor:

1. Az oldal tetejére görgeti a megjelenítést
2. Betölti a termékeket és a garázsokat
3. Inicializálja a kosarat, ha a felhasználó be van jelentkezve

**Kosár kezelése**

* addToCart: Új termék hozzáadása a kosárhoz
* Kosáridebar megjelenítése sikeres hozzáadás után
* Kosáradatok törlése kijelentkezéskor

**Navigáció**

* Termékre kattintáskor navigáció a részletes terméklapra
* "Tovább a webshopra" gomb a shop oldalra irányít

**Feltételes megjelenítés**

* Betöltési állapotok kezelése (loading skeleton)
* Különböző megjelenítés sötét vagy világos témában

**Kód elemzés**

**Erősségek**

* **Jól strukturált**: Logikusan csoportosított funkciók és UI elemek
* **Responsive design**: Reszponzív elrendezés különböző képernyőméretekhez
* **Felhasználói élmény**: Animációk, betöltési állapotok, téma váltás
* **Hatékony adatkezelés**: API kérések és helyi állapot kombinálása

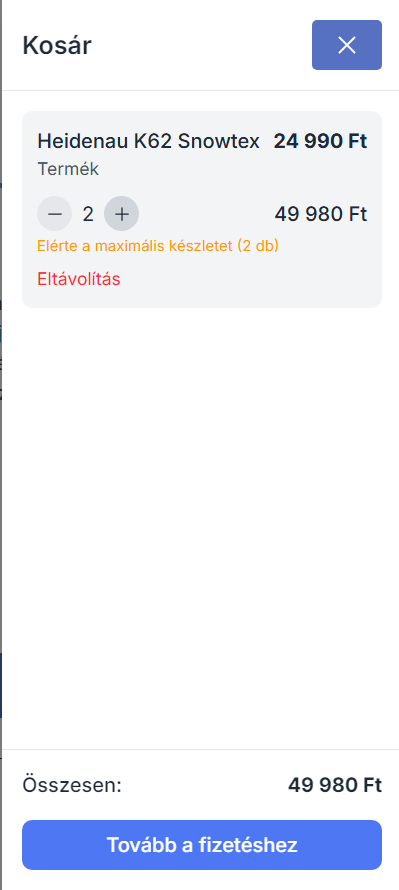
**Lehetséges fejlesztési területek**

* **Teljesítmény optimalizáció**: A több useEffect hook egyesítése bizonyos esetekben
* **Hibakezelés**: Hibaüzenetek megjelenítése adatbetöltési problémáknál
* **Komponens felosztás**: A több, mint 300 soros komponens kisebb alkomponensekre bontása
* **Hardcoded értékek**: A "<http://localhost:3000>" URL és más fix értékek környezeti változókba kiemelése

## ShopPage komponens

**Fő funkciók**

1. **Termék megjelenítés** - Reszponzív grid elrendezésben, betöltési animációkkal
2. **Több szintű szűrés**:
   * Szervíz/garázs alapján
   * Járműtípus szerint (személygépkocsi, motorkerékpár, teherautó)
   * Évszak szerint (téli, nyári, négyévszakos)
   * Gumiabroncs méret szerint (szélesség, profil, átmérő)
   * Keresés név és egyéb adatok alapján
3. **A képen képernyőkép, szöveg, szoftver, Multimédiás szoftver látható

   Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.Kosár kezelés** - Oldalsávként megjelenő kosár, termékek hozzáadása
4. **URL paraméter kezelés** - A szűrési feltételek megjelennek az URL-ben, így könnyen megoszthatók
5. **Világos/sötét mód támogatás**

**Props**

* setIsLoginOpen - Bejelentkezési ablak megnyitása
* setIsRegisterOpen - Regisztrációs ablak megnyitása
* isLoggedIn - Bejelentkezési állapot
* userData - Felhasználói adatok
* handleLogout - Kijelentkezés kezelése

**Adatforrások**

A komponens két fő API végpontról kér le adatokat:

* /garages - Elérhető szervízek listája
* /inventory - Termékek listája

**Szűrés működése**

A komponens fenntart egy eredeti terméklistát, majd a kiválasztott szűrők alapján egy useEffect hook-ban állítja elő a szűrt terméklistát. A szűrők állapotának változásakor automatikusan frissül az URL és a megjelenített termékek listája.

**Felhasználói interakciók**

* Termékre kattintás - navigálás a termék részletes oldalára
* "Kosárba" gomb - termék hozzáadása a kosárhoz (bejelentkezett felhasználóknak)
* Szűrők ki/bekapcsolása - azonnal frissíti a látható terméklistát
* Aktív szűrők törlése - egyenként vagy egyszerre mind

**Felhasználói élmény**

A komponens számos funkciót tartalmaz a jobb felhasználói élmény érdekében:

* Betöltési animációk (skeleton loader)
* Aktív szűrők látható címkékként jelennek meg
* A szűrők egyszerűen törölhetők
* Keresési eredmények üres állapot kezelése

## Checkout komponens

**Általános felépítés**

A Checkout komponens egy modern, reszponzív fizetési és időpontfoglalási oldalt implementál, amely:

* Két fő szekcióból áll: egy kosár összesítőből és egy űrlapból
* Sötét és világos témát támogat
* Animációkat használ a felhasználói élmény javítására
* Validációkat tartalmaz a felhasználói hibák elkerülésére
* API-kal kommunikál az adatok tárolásához és lekéréséhez

**Főbb funkciók**

**1. Kosár kezelés**

A komponens megjeleníti a felhasználó kosarában lévő termékeket, azok mennyiségét és összesített árát. Ha a kosár üres, a felhasználót automatikusan átirányítja a webáruházba.

**2. Fizetési adatok kezelése**

Bekéri a felhasználó elérhetőségi adatait (név, email, telefonszám), amelyeket előre kitölt, ha a felhasználó be van jelentkezve. A rendszer csak készpénzes fizetést támogat, amelyet a felhasználó az átvételkor teljesít.

**3. Szerviz és időpontfoglalás**

Lehetőséget biztosít a felhasználónak, hogy:

* Válasszon a rendelkezésre álló szervizek közül
* Válasszon egy dátumot a naptárból
* Az adott napra elérhető időpontok közül foglaljon időpontot

**4. Dinamikus adatbetöltés**

A komponens API-hívásokkal tölti be:

* A szervizek listáját
* Az adott szervizre és napra vonatkozó szabad időpontokat

**5. Rendelés leadása**

A rendelés véglegesítésekor a komponens:

* Összeállítja a megrendelés adatait
* Létrehozza az időpontfoglalást (ha a felhasználó választott időpontot)
* A rendelés sikeres leadása után átirányítja a felhasználót a sikeres fizetés oldalra

**Technikai megvalósítás**

A komponens számos modern webtechnológiát és React-megoldást alkalmaz:

**Használt library-k és technológiák**

* React hook-ok az állapotkezeléshez
* Context API a globális állapotok (kosár, téma) kezeléséhez
* React Router a navigációhoz
* Axios az API-kommunikációhoz
* Framer Motion az animációkhoz
* Date-fns a dátumok kezeléséhez
* Tailwind-szerű osztályokat használ a stílusokhoz

**Állapotkezelés**

A komponens összetett állapotkezelést valósít meg: nyilvántartja a felhasználói adatokat, az időpontfoglaláshoz kapcsolódó információkat, a betöltési állapotokat és a hibaüzeneteket.

**Felhasználói visszajelzés**

A komponens visszajelzést ad a felhasználónak:

* Betöltési indikátorokat jelenít meg az adatok lekérésekor
* Hibaüzeneteket mutat, ha valami nem sikerül
* Vizuális megerősítést ad, amikor a felhasználó kiválaszt egy időpontot

**Reszponzív design**

A komponens különböző képernyőméretekre optimalizált elrendezést biztosít, a mobileszközöktől az asztali számítógépekig.

## GarageAppointmentSchedulePage komponens

**Főbb funkciók**

1. **Felhasználói jogosultságkezelés**

* Csak garage\_owner szerepkörrel rendelkező felhasználók használhatják
* Csak a saját garázsuk foglalásait és beosztását kezelhetik

1. **Háromféle nézet**

* Beosztás nézet (schedule): heti időpontok beállítása
* Heti naptár (week): vizuális áttekintés a foglalásokról
* Foglalások lista (bookings): részletes foglalási információk

1. **Időpontkezelés funkciók**

* Új időpontok felvétele meghatározott kapacitással
* Időpontok törlése
* Időpontok másolása egyik napról a másikra
* Átfedések ellenőrzése az időpontok között

1. **Foglalások kezelése**

* Foglalások áttekintése időpont szerint
* Foglalások státuszának követése (függőben, megerősítve, teljesítve, lemondva)
* Részletes információk megtekintése egy adott időpontra vonatkozóan

1. **Felhasználói élmény**

* Reszponzív dizájn (mobil és asztali nézet)
* Sötét/világos mód támogatása
* Animációk a frissebb UI élmény érdekében (Framer Motion használata)
* Tooltipek a gombokon a jobb használhatóság érdekében

**Technológiai megvalósítás**

* React funkcionális komponensek és hook-ok használata
* React Router a navigációhoz
* Axios az API-kérések kezeléséhez
* Kontextusok használata (ThemeContext, CartContext)
* date-fns könyvtár a dátumkezeléshez
* Modularizált komponensek (különálló komponensek a különböző nézetekhez)

**Adatkezelés**

* A garázsadatok, foglalások és időpontok lekérése REST API-n keresztül
* Lokális state-ek a felhasználói felület állapotának kezeléséhez
* Token alapú autentikáció a backend kérésekhez

**Biztonsági funkciók**

* JWT token alapú hitelesítés
* Felhasználói jogosultságellenőrzés (csak a garázs tulajdonosa férhet hozzá)
* Hibakezelési mechanizmusok

**Kiemelendő megoldások**

* Többféle nézet az időpontok kezeléséhez, ami rugalmas használatot tesz lehetővé
* Időpontok másolása a különböző napok között, ami megkönnyíti a rendszeres beosztások beállítását
* Átfedések automatikus ellenőrzése az időpontok között
* A komponens figyelembe veszi a valódi használati eseteket, mint például a foglalások maximális számának beállítása

# Fontosabb backend elemek leírása

## UsersController elemzése

**Főbb funkciók a kontrollerben:**

1. **createUser**: Új felhasználó létrehozása

* Ellenőrzi a kötelező mezőket (vezetéknév, keresztnév, email, szerep, jelszó)
* Titkosítja a jelszót bcrypt segítségével
* Létrehozza a felhasználói rekordot
* Regisztrációs e-mailt küld

1. **authenticateUser**: Felhasználói bejelentkezés kezelése

* Ellenőrzi a bejelentkezési adatokat
* Összehasonlítja a jelszót a tárolt hash-sel
* Frissíti az utolsó bejelentkezés időpontját
* JWT tokent generál a sikeres bejelentkezéshez

1. **getAllUsers**: Az összes felhasználó listázása

* A jelszó hash-ek nélkül adja vissza a felhasználókat

1. **getUserById**: Egy felhasználó adatainak lekérése azonosító alapján
2. **updateUser**: Felhasználói adatok frissítése

* Jogosultságkezelést tartalmaz (csak admin vagy a saját profil módosítható)
* Jelszóváltoztatás esetén új hash-t generál

1. **changePassword**: Célzott jelszóváltoztatási funkció

* Ellenőrzi a jelenlegi jelszót
* Jogosultságkezelést tartalmaz
* Az új jelszót hash-eli és menti

1. **deleteUser**: Felhasználó törlése

* Jogosultságkezelést tartalmaz a törléshez

1. **uploadProfilePicture**: Profilkép feltöltése

* Jogosultságkezelést tartalmaz
* Ellenőrzi a feltöltött fájlt
* Menti a képútvonalat a felhasználóhoz

**Biztonsági elemek:**

* Jelszavak hash-elése bcrypt segítségével
* JWT authentikáció
* Jogosultságellenőrzés (admin vs. normál felhasználó)
* Hibaüzenetek nem adnak ki érzékeny információt

## InventoryController elemzése

**Felépítés és Funkciók**

A controller a következő fő funkciókat tartalmazza:

1. **getAllItems** - Leltári elemek listázása különböző szűrési, rendezési lehetőségekkel:
   * Szűrés garázsok, járműtípusok, árak, évszakok, gumiabroncs-méretek szerint
   * Keresés terméknév alapján
   * Rendezés különböző szempontok szerint
2. **getItemById** - Adott azonosítójú termék lekérdezése
3. **createItem** - Új termék létrehozása:
   * Adatok validálása
   * Garázsok ellenőrzése
   * Képfeltöltés kezelése
4. **updateItem** - Termék frissítése:
   * Validáció
   * Képek kezelése (feltöltés, törlés)
5. **deleteItem** - Termék törlése (beleértve a képek törlését is)
6. **getItemsByGarageId** - Adott garázs termékeinek lekérdezése
7. **updateItemQuantity** - Termék mennyiségének frissítése

**Technikai Jellemzők**

A kódban a következő technikai megoldások figyelhetők meg:

1. **Sequelize ORM** használata adatbázis-műveletekhez
   * Kapcsolódó modellek: Inventory és Garage
   * Különböző lekérdezési feltételek és operátorok (pl. Op.like, Op.gte)
2. **Fájlkezelés**
   * Kép feltöltések kezelése
   * Régi képek törlése frissítéskor
3. **Hibakezelés**
   * Sequelize hibák elkapása (pl. egyedi kulcsok megsértése)
   * Validációs hibák kezelése
   * Próbálkozás-elkapás minták minden művelethez
4. **Adatvalidáció**
   * Paraméterek ellenőrzése (pl. számok, enumerációk)
   * Létező garázsok ellenőrzése
   * Kötelező mezők ellenőrzése

**Erősségek**

1. Részletes validáció és hibaüzenetek
2. Jól strukturált kód
3. Fájlkezelés biztonságos implementációja
4. Rugalmas szűrési és rendezési lehetőségek

## AppointmentsController elemzése

**1. Általános cél**

A kontroller az időpontfoglalási funkciók kezelését végzi egy gumiszerviz webalkalmazás backend oldalán. A következő fő funkciókat valósítja meg:

* Időpontok lekérése (összes, vagy ID, illetve felhasználó alapján)
* Időpont létrehozása validációval
* Időpont módosítása validációval és opcionális e-mail küldéssel
* Időpont törlése
* Jogosultságkezelés a felhasználói időpontok lekérdezésénél

**2. Függőségek**

* Appointments: az időpontokat kezelő modell
* GarageScheduleSlot: időablakokat kezelő modell
* User, Garage: kapcsolódó entitások
* Op: Sequelize operátorok
* emailService: e-mail küldés időpont visszaigazolásához

**3. Függvények elemzése**

**getAllAppointments**

* Lekéri az összes időpontot az adatbázisból.
* Csak a legfontosabb mezőket adja vissza (id, user\_id, stb.).
* Egyszerű listázó funkció, tipikusan admin vagy garázs tulajdonos számára.

**getAppointmentById**

* Egyetlen időpontot keres ID alapján.
* 404-es hibát ad, ha nincs ilyen rekord.
* Egyszerű lekérdező funkció.

**createAppointment**

* Új időpontot hoz létre.
* Ellenőrzi, hogy minden kötelező mező meg van-e adva (user\_id, garage\_id, appointment\_time, order\_id).
* Ha van schedule\_slot\_id:
  + Lekéri a slotot, ellenőrzi, hogy a megadott garázshoz tartozik-e.
  + Ellenőrzi, hogy az időpont a megfelelő napra esik-e, és időben beleesik-e a megadott slotba.
  + Megszámolja a meglévő foglalásokat az adott slotban, és elutasítja a foglalást, ha elérte a maximumot.
* Siker esetén létrehozza az új rekordot.

**updateAppointment**

* Módosít egy meglévő időpontot.
* Ha változik a schedule\_slot\_id, akkor újra lefutnak a foglalási szabályok:
  + Slot létezésének és garázshoz tartozásának ellenőrzése
  + Nap és időintervallum validáció
  + Foglaltsági szint ellenőrzése
* Ha a státusz „confirmed” -re változik, e-mailt küld a felhasználónak.
* Hibakezeléssel védi a teljes folyamatot, beleértve az e-mail küldés esetleges hibáját is.

**deleteAppointment**

* Törli az időpontot ID alapján.
* Ha nem található a rekord, 404-es hibát ad.
* Siker esetén visszaad egy sikeres törlés visszajelzést.

**getAppointmentsByUserId**

* Lekéri egy felhasználó összes időpontját.
* Ellenőrzi, hogy a lekérő az adott felhasználó saját időpontjait nézi-e (auth check).
* A válasz tartalmazza a garázs és slot információkat is (include-al).
* Rendezi az eredményt idő szerint visszafelé.

**4. Erősségek**

* Részletes validáció minden időpont műveletnél.
* Jó hibakezelés: 400-as, 403-as, 404-es és 500-as válaszok.
* Slotokhoz kötött logika megfelelően implementálva (nap, idő, foglaltság).
* Jogosultságkezelés a felhasználói időpont lekérdezésnél.
* E-mail küldés csak akkor, ha valóban megerősítés történik.

# Backend tesztelés

A backend tesztek célja, hogy ellenőrizzék a szerveroldali logika helyes működését, és megelőzzék a hibák éles rendszerbe kerülését. A projektben Jest keretrendszerrel készült egységtesztek futnak, amelyek mockolják az adatbázis-modelljeinket és külső szolgáltatásokat. A tesztek lefedik az alapvető vezérlőműveleteket, kezelik a hibákat, és biztosítják, hogy a válaszok megfeleljenek az elvárt működésnek.

## AppointmentsController tesztjei

**Általános beállítások**

* A tesztmodulban minden adatbázismodellt (Appointments, GarageScheduleSlot, User, Garage) és külső szolgáltatást (emailService, sequelize) mockoltak.
* A beforeEach részben létrejön egy alap req és res objektum, amelyek minden teszthez újrainicializálódnak.
* A jest.clearAllMocks() gondoskodik arról, hogy az előző tesztek mellékhatásai ne befolyásolják az aktuális futást.

**1. getAllAppointments**

**Pozitív teszt:**

* Sikeres esetben visszaadja az összes időpontot (Appointments.findAll visszatér egy tömbbel).
* Ellenőrzi, hogy meghívásra került-e a findAll, és a válaszban az elvárt adatok szerepelnek.

**Negatív teszt:**

* Hibakezelés tesztelve: ha a findAll hibát dob, 500-as státuszkódot és hibát tartalmazó választ ad vissza.

**2. getAppointmentById**

**Pozitív teszt:**

* Ha az időpont létezik, findByPk segítségével lekérdezi, és visszaadja az adatokat.

**Negatív teszt:**

* Ha az adott id-hez nem tartozik időpont, akkor 404-es státuszt és hibaüzenetet ad vissza.

**3. createAppointment**

**Pozitív tesztek:**

* Ha minden kötelező mező megvan (user\_id, garage\_id, appointment\_time, order\_id), és nincs megadva schedule\_slot\_id, az időpont létrejön.
* Ha schedule\_slot\_id is szerepel:
  + A slotot lekérdezi (GarageScheduleSlot.findByPk).
  + Megszámolja a meglévő foglalásokat a slothoz (Appointments.count).
  + Ha a max\_bookings értéket nem haladja meg, létrehozza az időpontot.

**Negatív tesztek:**

* Ha hiányoznak mezők a kérésből, 400-as válasz jön, benne a „Missing required fields” üzenettel.
* Ha a slot már teljesen lefoglalt, a válasz 400-as státusz, és egy hibaüzenet, miszerint a slot betelt.

**4. updateAppointment**

**Pozitív teszt:**

* Az időpontot findByPk segítségével lekérdezi.
* Frissíti a státuszt (appointment.update).
* Ha a státusz értéke confirmed, akkor:
  + Lekéri a felhasználót (User.findByPk) és a szervizt (Garage.findByPk).
  + Meghívja az emailküldő szolgáltatást (mockolt sendAppointmentConfirmationEmail).

**Negatív teszt:**

* Ha az időpont nem található, akkor 404-es válasz érkezik.

**5. deleteAppointment**

**Pozitív teszt:**

* Lekérdezi az időpontot, majd meghívja a destroy metódust, amivel törli az adatbázisból.
* Siker esetén üzenetet küld vissza, hogy a törlés megtörtént.

**Negatív teszt:**

* Ha az időpont nem található, 404-es válasz és hibaüzenet érkezik.

**6. getAppointmentsByUserId**

**Pozitív teszt:**

* Ha az adott felhasználó létezik, és azonos a belépett felhasználóval, akkor visszaadja az időpontjait.
* Meghívja a User.findByPk és Appointments.findAll metódusokat a lekérdezéshez.

**Negatív tesztek:**

* Ha a felhasználó nem található, 404-es válasz jön vissza.
* Ha a belépett felhasználó nem egyezik a kért userId-val, akkor 403-as válasz érkezik, „Unauthorized access” üzenettel.

## InventoryController tesztjei

**Általános struktúra és módszertan**

A teszt modern JavaScript tesztelési gyakorlatokat követ:

* Jest mock-ok használata a külső függőségek (model-ek, FS műveletek) izolálására
* A describe/it struktúra a tesztek logikus csoportosítására
* A beforeEach hook használata a közös teszt előkészítésekre
* Hibaesetek és peremfeltételek alapos lefedése

**Lefedett funkciók**

A tesztkészlet az alábbi controller funkciókat fedi le:

1. **getAllItems** - Leltári elemek lekérdezése különböző szűrési feltételekkel
2. **getItemById** - Egy elem részletes adatainak lekérdezése
3. **createItem** - Új leltári elem létrehozása
4. **updateItem** - Meglévő elem módosítása
5. **deleteItem** - Elem törlése
6. **getItemsByGarageId** - Adott garázshoz tartozó elemek lekérdezése
7. **updateItemQuantity** - Készletmennyiség gyors frissítése

**Tesztelési lefedettség**

Minden funkcióra alapos teszteket írtak, amelyek tartalmazzák:

* Sikeres művelet tesztelése a várható eredménnyel
* Validációs hibák kezelésének ellenőrzése
* Hiányzó erőforrások (nem létező elem, garázs) esetének kezelése
* Szerveroldali hibák megfelelő kezelése

**Kiemelendő részek**

**Fájlkezelés tesztelése**

A rendszer képfájlok kezelését is végzi, és a tesztek részletesen ellenőrzik:

* A régi képek törlését az FS modulon keresztül
* Az új képek megfelelő tárolását
* Az útvonalak helyes kezelését

**Szűrési és keresési funkciók**

Az getAllItems végpont összetett szűrési lehetőségeket támogat:

* Garázs szerinti szűrés
* Járműtípus szerinti szűrés
* Ár intervallum szerinti szűrés
* Szöveges keresés az elemek nevében

**Validáció**

A controller megfelelően validálja a bemeneti adatokat:

* Kötelező mezők ellenőrzése
* Enumeráció típusú értékek validálása (pl. járműtípusok)
* Numerikus értékek ellenőrzése

## UsersController tesztjei

**A tesztfájl felépítése és jellemzői**

A tesztek a usersController következő funkcióit vizsgálják:

* Felhasználó létrehozása (createUser)
* Felhasználó hitelesítése/bejelentkezés (authenticateUser)
* Felhasználó lekérdezése azonosító alapján (getUserById)
* Felhasználó adatainak frissítése (updateUser)
* Jelszó módosítása (changePassword)
* Felhasználó törlése (deleteUser)
* Profilkép feltöltése (uploadProfilePicture)
* Összes felhasználó lekérdezése (getAllUsers)

A tesztkód jól strukturált, minden végpont tesztelésére külön describe blokkot használ, és ezeken belül különböző teszteseteket vizsgál.

**Mockolási stratégia**

A tesztek hatékony izolációja érdekében a fájl számos függőséget mockol:

* Users modell: az adatbázis-műveleteket végző függvényeket (create, findOne, stb.)
* bcryptjs: a jelszókezelési funkciókat (hash, compare)
* jsonwebtoken: a JWT token generálást
* emailService: a regisztrációs e-mail küldés funkcióját

Minden teszt előtt a mockok alaphelyzetbe állnak a jest.clearAllMocks() hívással.

**Tesztesetek minősége**

A tesztek három alapvető esetet vizsgálnak minden funkciónál:

1. **Sikeres eset**: amikor minden paraméter helyes és a művelet sikeresen végrehajtódik
2. **Validációs hibák**: amikor hiányos vagy érvénytelen adatokat kap a kontroller
3. **Hibakezelés**: amikor a függőségek hibát dobnak

Külön tesztesetek vizsgálják a jogosultságkezelést is azoknál a funkcióknál, ahol ez releváns (pl. csak admin vagy a saját felhasználó módosíthat adatokat).

**Erősségek**

1. **Alapos lefedettség**: Minden kontroller funkció minden lehetséges útvonalát és hibakezelési ágát teszteli.
2. **Jól strukturált tesztek**: A describe és it blokkok logikusan tagolják a teszteket.
3. **Részletes ellenőrzések**: Nem csak a visszaadott HTTP státuszkódot, hanem a válasz tartalmát is ellenőrzi.
4. **Mockolt függőségek**: A külső rendszerektől való függetlenség biztosított.
5. **Előkészítések használata**: A beforeEach segítségével minden teszt tiszta állapotból indul.

# Frontend tesztelés

A frontend tesztelését mi manuálisan végeztük el. Ennek során alaposan átvizsgáltuk az alkalmazás különböző funkcióit, interakcióit és a felhasználói élményt a böngészőben. A manuális tesztelés során figyeltünk arra, hogy a felhasználói felület megfelelően jelenjen meg minden eszközön, és hogy minden interaktív elem – mint például gombok, űrlapok, szűrők – a kívánt módon működjön. Ezen kívül ellenőriztük, hogy az alkalmazás gyorsan reagáljon a felhasználói műveletekre, és megfelelő hibaüzeneteket jelenítsen meg szükség esetén.

# **Jövőbeli Fejlesztések és Skálázhatóság**

A szoftver jövőbeli bővítési lehetőségei közé tartozhatnak például a következők:

* **Mobilalkalmazás fejlesztése**, hogy a felhasználók könnyebben és gyorsabban érhessék el a szolgáltatásokat akár útközben is. Az alkalmazás lehetőséget biztosítana a gyors időpontfoglalásra, értesítések fogadására és a rendelési előzmények nyomon követésére.
* **Új szervizek könnyebb integrálása**, amely lehetővé tenné, hogy a rendszer rugalmasan bővíthető legyen, és további műhelyek is csatlakozhassanak. Ezzel biztosítható lenne a rendszer országos méretű skálázása, valamint az adminisztrációs lehetőségek bővítése.
* **Online fizetési lehetőségek integrálása**, amely lehetővé teszi az ügyfelek számára, hogy előre rendezzenek bizonyos szolgáltatásokat, csökkentve ezzel a helyszíni fizetésekből adódó várakozási időt.
* **Ügyfélszolgálati chat funkció beépítése**, amely gyors kapcsolatfelvételt tenne lehetővé a felhasználók és a szervizek között, ezáltal csökkentve az ügyfélkezelési időt és növelve az elégedettséget.
* **Automatikus karbantartási emlékeztetők küldése**, hogy a felhasználók időben értesüljenek arról, mikor lenne érdemes gumicserére vagy más szolgáltatásra időpontot foglalni. Ez javíthatja az ügyfélmegtartást és ösztönözheti az ismételt szolgáltatásokat.
* **Kiterjesztett statisztikai és elemzési funkciók**, amelyek segítenék a szervizek működésének optimalizálását és a felhasználói igények jobb megértését.

# Összegzés

A dokumentációban bemutatott lépések, valamint a különböző funkciók megvalósítása és tesztelése révén átfogó képet kaptunk a projekt részleteiről és működéséről. A feladatok során használt technológiai eszközök (React, Node.js, Jest) mind hozzájárultak a hatékony fejlesztéshez és teszteléshez. A tesztelési folyamatok eredményei alapján az alkalmazás megbízhatósága és stabilitása biztosítottnak tűnik. A hibák és problémák gyorsan beazonosíthatóak és orvosolhatóak, így a jövőbeli fejlesztések is gördülékenyebben haladhatnak. A projekt során szerzett tapasztalatok segítettek abban, hogy mélyebb megértést nyerjünk a fejlesztési és tesztelési ciklusok összetettségéről, valamint a felhasználói élmény javításának fontosságáról. Továbbá tisztában lettünk egymás erősségeivel és gyengeségeivel is, ezzel jobban megismerve önmagunkat és sok tapasztalatot szereztünk a fejlesztés során. Összességében a projekt fejlesztése alatt sok kihívással álltunk szemben mind technológia, mind tudás szempontjából, azonban így is igyekeztük hozzánk mérve a legtöbbet kihozni a vizsgamunkánkból.