PD CAR WEBSITE – Fejlesztői dokumentáció

### by: Pór Dániel Endre NEPTUN: TMNRCM

# Bevezető

A weboldal egy autó kölcsönző alkalmazást valósít meg. Az autóbérlő tulajdonosa (adminisztrátor) képes új autókat felvenni, amit a felhasználók kibérelhetnek. A kibérlést a felhasználó megtekintheti és befejezheti az oldalon. Az adminisztrátor minden bérlést lát és törölhet, csak úgy mint minden autót is.

# Technológiai áttekintő

A weboldal front-end-jét az Angular framework segítségével valósítottam meg, ami egy REST API-s back-end-et szólít meg ami Springboot-al valósítottam meg. A spring-es backend a beépített Spring data-s eszközökkel éri el a relációs adatbázist. A relációs adatbázisom PostgreSQL.

A backend fejlesztéséhez az IDEA intellij fejlesztő környezetet használtam, az Angular frontend fejlesztéséhez a VS Code-ot. Az adatbázis felügyeléséhez a pgAdmin-t használtam.

# Adatbázis

Az adatbázist a Postgresql szolgálja ki az 5432-es porton. Az alkalmazás indításakor mindig újrainicializálódnak a táblák. Alapvetően két autó kerül be, és egy szuperfelhasználó USER és ADMIN jogokkal. Három tábla van az adatbázisban: User ami a felhasználó entitásokat tartalmazza, Car, ami az autókat tartalmazza és egy összekötő tábla a Rent, mely a User és a Car kulcsát tartalmazza idegenkulcsként és egy timestamp-et ami a bérlés időpontját tartalmazza. A User és Rent között több-egy kapcsolat van, hiszen egy felhasználó, több autót is bérelhet. A Car és Rent közt egy-egy kapcsolat van, egy autóhoz egy bérlés tartozhat.

#### User séma:

* *id*: A User egyedi azonosítója
* *username*: A User felhasználóneve
* *user\_email*: A felhasználó email címe
* *active:* Aktív-e a felhasználó
* *password\_hash:* A jelszót tartalmazó mező
* *authority:* Milyen felhaszálói csoporthoz tartozik a felhasználó. Ez lehet ROLE\_ADMIN ha adminisztrátor és ROLE\_USER ha felhasználó.

#### Car séma:

* *car\_id:* A Car egyedi azonosítója
* *car\_name:* Az autó neve
* *car\_color:* Az autó színe
* *image\_url*: Az autóhoz tartozó képnek az URL-je

#### Rent séma:

* *rent\_id*: A bérlés azonosítója
* *fk\_car:* Az autó egyedi azonosítója idegenkulcsként.
* *fk\_user* A User egyedi azonosítója idegenkulcsként.
* *rent\_ts:* A bérlés időpontja

# Back-end

A back-end-et Springboot-al valósítottam meg. Egy REST API-s kezelő felületet valósítottam meg. A back-end projekt struktúrája a következő:

->pd.cars.cars: A root.

->controller: Ebben találhatók az endpoint-ok definíciói.

->model: Ebben találhatók az entitások amit az adatbázisban is szerepelnek

->repository: Repository interface-ek amik leírják a spring-nek hogyan kell elérni az adatokat az adatbázisban

->security: Az endpointok biztonságát implementáló osztályok.

->service: Az entitásokhoz tartozó kiszolgáló osztályok.

A back-end követi a Spring MVC tervezési mintát.

# Controller

Az endpointokat két osztályban valósítottam meg: A CarWesbiteController és a UserResouce-ban. Az előbbiben az autóbérléshez szükséges endpointok vannak megvalósítva a másikban az auntenikációhoz szükségesek.

### CarWebsiteController

Minden itteni endpoint elé alapvetően az api/car-rental út vezet.

* GET /cars: Visszatér az adatbázisban található összes autóval
* GET /cars/{id}: Visszatér egy bizonyos id-val rendelkező autóval
* GET /rents: Visszatér az összes bérléssel ha a bejelntkezett felhasználó admin, egyébként meg a felhasználó bérléseivel.
* POST cars/{id}: Létrehoz egy bérlést egy adott autóra az épp bejelentkezett felhasználónak az adott azonosítójú autóra.
* GET isrented/{id}: Megmondja, hogy az adott azonosítójú autó ki van-e bérelve.
* DELETE /cars/{id}: Kitörli az adott azonosítójú autót. (A front-end biztosítja, hogy ezt csak admin jogosultság legyem)
* POST cars/: Létrehoz egy autót a paraméterként kapott request szerint.
* DELETE /rents/{id}: Kitörli az adott azonosítójú bérlést.

### UserResource

* POST /register: Létrehoz egy felhasználót.
* GET /user: Visszatér az épp bejelentkezett felhasználóval.

Illetve egy /auth is tartozik ide, ami a Security-ben lett implementálva.

# Service osztályok

A Service osztályok az egyes entitások adatbázis elérését segítik, és az entitásokon elvégezhető műveleteket tartalmazza. Injektálva van beléjük a megfelelő Repository osztály interface-ek amik segítségével hajtanak végre módosításokat (CRUD) az adatbázisban lévő táblájukon. Alapvetően implementálnak egy save és findAll metódust, ami általában az Repository által kínált függvényt használja. A Rent service-ben lehetőség van autó illetve felhasználó szerint is keresni az adatbázisban.

# Biztonság (security)

A backend biztonságát Spring Security segítségével és a JWT (Json Web Token) referencia szerint implementáltam. Az alap konfiguráció a SecurityConfiguration-ben található. Itt be van állítva, hogy az autentikáció állapot nélküli (ez feltétele a JWT autentikációhoz), a CORS úgy lett konfigurálva, hogy a front-end-től fogadjon csak kéréseket a back-end. Alapvetően az auth és register endpointok elérhetők, amin keresztül lehet tokent szerezni. Két filter van létrehozva: a JwtUsernameAndPasswordAuthenticationFilter és a JwtTokenAuthenticationFilter. Az előbbi a bejelentkezést segíti, a másik meg annak az ellenőzését, hogyha már létezik token, akkor hitelesítse azt. A JWT generáló, tároló és azon műveleteket végző részt a JwtCookieStore osztály valósítja meg.

Bejelentkezéskor kérés jön az api/auth címre, ilyenkor a request-ben érkeznek a bejelelentkezéshe szükséges adatok. A UserCredentials osztály mint egy DTO-ként működik ami a jelszó felhasználónév párost tartalmazza. Az autentikációt a CustomAuthenticationProvider segíti, mely megnézi, hogy az adott felhasználó benne van az adatbázisban, és a az adott jelszó megegyezik. Amennyiben ez teljesül a successfulAuthentication-ben a JwtCookieStore létrehoz egy token-t, amit visszaküld a response-ban. A token-t a bögésző sütiként tárolja. Amikor kéréseket intéz a front-end a JwtAuthetnicationFilter doFilter metódusa megnézi a böngésző által küldött sütiben szerepel-e le nem járt hiteles süti. Amennyiben igen, engedélyezi a kéréseket az endpoint-ok felé.

# Felhasználói körök

#### ROLE\_ADMIN:

Az adminisztrátori felhasználói csoport. Az alkalmazás indításakor egy ilyen jön létre. Megtekintheti az összes bérlést, kitörölhet és felvehet új autókat az adatbázisba.

#### ROLE\_USER:

Sima felhasználói csoport. Regisztráció után megtekintheti az autókat, tetszőleges mennyiségben intézhet bérlést.

# Front-end

A front-end-et az Angular framework-el implementáltam. A REST-API-val való kommunikációt a restapi.service-ben található publikus metódusokkal kommunikál a frontend. Itt minden endpoint-nak van egy megfelelő metódusa. Emellett definiálva vannak az entitásoknak megfelelő osztályok: Car, Rent, és egy DTO az AuthRequest, ami az autentikációhoz szükséges információkat szállítja.

## Frontend komponensek

Login komponens: A bejelentkezésért felelős komponens. Miután a felhasználó megadta a belépési adatait, a login-ra kattintva a back-end felé autentikációs kérést küld. Amennyiben sikeres lett az autentikáció, a Home komponensre irányítja a felhasználót. A Login mellett lehetőség van a Register gombra kattintva regisztrálni ilyenkor az oldal átirányít a Register komponensre.

Register komponens: Itt új felhasználó regisztrálhat. Meg kell adnia a felhasználó nevét és jelszót és egy email címet. A frontend ellenőrzi, hogy a megadott mezők megfelelően legyenek kitöltve. Ha a felhasználónév vagy email cím már regisztrálva van, akkor ezt a backend jelzi, és a frontend a jelzést interpretálva tájékoztatja erről a felhasználót.

Home komponens: Itt láthatjuk az összes adatbázisban lévő autót, ha ki van bérelve, ha nincs kibérelve. A komponens inicializálója egy kérést küld a backend felé, ami visszatér az adatbázisban lévő autókkal, és ezeket megjeleníti egy táblában. Itt a felhasználó, ha egy autó melletti Rent-re kattint, akkor rákerül a Carform komponensre, ahol további adatokat láthat az autóról. A View Rent gomb a Rentform-ra irányítja a felhasználót, ahol a bérléseket láthatja a felhaszálói csoportja szerint