**ENDPOINTOK**

**Authentikáció:**

**Regisztrálás**http://localhost:8080/auth/register

Json formában megkapja a user adatokat majd az addNewUser service segítségével feltölti adatbázisba, majd visszaküldi az accesstokent, refreshtokent json formában.

UserRegisterModel-t használ hogy elkerülje az esetleges adminisztrátori jog megszerzését. Így hiába regisztráláskor "admin": true szerepel a json-ben, átlagos felhasználóként regisztrálja. Regisztrálás alatt leelenőrzi, hogy szerepel e már az adatbázisban egy felhasználó az adott email címmel, ha igen visszaküldi a 409-es ***CONFLICT*** hibakódot. Ha valamelyik adattagot is kihagyjuk akkor a 400-as ***BAD REQUEST*** hibakódot dobja vissza a szerver.  
  
Regisztráláshoz szükséges JSON formátum:

POST http://localhost:8080/auth/register  
*Content-Type*: application/json  
  
{  
 "first\_name": "Elek",  
 "last\_name": "Teszt",  
 "email": "tesztelek2@gmail.com",  
 "password": "Adminadmin1"  
}

**Bejelentkezés**  
<http://localhost:8080/auth/login>

Json formában megkapja a user adatokat majd a findUserByEmailAndPasswordservice segítségével ellenőrzi az adatbázisban, ha sikeres visszaküldi az accesstokent és a refreshtoken json típusban. Ha viszont sikertelen akkor a tokenek helyett null és a status code 403 ***FORBIDDEN.*** Bejelentkezéshez elengedthetetlen az email cím és egy jelszó.

Belentkezéshez szükséges JSON formátum:

POST http://localhost:8080/auth/login  
*Content-Type*: application/json  
  
{  
 "email": "tesztelek@gmail.com",  
 "password": "Adminadmin1"  
}

**Admin-login**  
<http://localhost:8080/auth/admin-login>

Ugyan az mint a /login egy extra lépéssel. Csak akkor küldi vissza, ha a felhasználó admin.

**Refresh**  
<http://localhost:8080/auth/refresh>

Ha egy request eredménye 451-es hibakódot dob akkor a frontend küld egy újabb requestet ami erre az end pointra érkezik, body-ban egy refreshTokennel.

A refresh tokenből kinyeri az email címet a getEmailFromRefreshToken service-el majd ellenőrzi adatbázisban a létezését, azután generál egy accessTokent.

Majd visszaküldi az accessTokent és a refreshTokent json típusban. Ha lejárt/érvénytelen a refreshToken vagy a felhasználó nem létezik akkor a tokenek helyett null és ***BAD\_REQUEST*** státusz.

Access token frissítéséhez szükséges JSON formátum:

POST http://localhost:8080/auth/refresh  
*Content-Type*: application/json  
  
{  
 "refreshToken": "<refresh token>"  
}

***Felhasználó kezelés***

**Felhasználói adatok lekérdezése**  
<http://localhost:8080/user/data>

Headerben megkapja az access token-t. Ellenőrzi a token érvényességét, lejárat esetén **451**-es hibakódot küld vissza. Kinyeri a token-ből az e-mail címet. Email alapján kikeresi a felhasználó adatait majd visszaküldi a felhasználói adatokat a jelszót leszámítva egy UserResponseDto formában. Ha nem sikerül azonosítani a felhasználót akkor válaszként a 401-es ***UNAUTHORIZED*** státuszt küldi.

GET http://localhost:8080/user/data   
*Authorization*: Bearer <access token>

**Összes felhasználó lekérdezése**  
<http://localhost:8080/user/get-all>

Minden usert visszaküld feltéve ha a felhasználó aki kéri admin joggal rendelkezik (Ezt a tokenből ellenőrzi). Ellenőrzi az admin jogot sikeres ellenőrzés után visszaküldi a user listát egy http status ok-al. Ha a felhasználó nem admin akkor visszaküldi a 403-as ***FORBIDDEN*** státuszt.

GET http://localhost:8080/user/get-all   
*Authorization*: Bearer <access token>

**Felhasználó törlése**  
[http://localhost:8080/user/{userId}](http://localhost:8080/user/%7buserId%7d)

Megkapja a törölni kívánt felhasználó azonosítóját paraméterként és egy access token-t a header-ben. Megnézi, hogy az access token még érvényes -e ha nem akkor küld egy ***451***-es kódot. Ezután ellenőrzi, hogy a felhasználó admin jogosultsággal rendelkezik -e. Ezek után törli a kiválasztott felhasználót az adatbázisból. Ha sikeres akkor vissza kapja az üzenetet, hogy "Felhasználó sikeresen törlésre került" és egy HttpStatus OK-t, ha viszon nem rendelkezik admin jogosultsággal akkor visszaküldi a ***FORBIDDEN*** status-t egy üzenettel

"A hozzáféréshez admin jogosultság szükséges!".

DELETE http://localhost:8080/user/2  
*Authorization*: Bearer

**Felhasználói adatok frissítése**  
[http://localhost:8080/user/{userId}](http://localhost:8080/user/%7buserId%7d)

Paraméterként megkapja a módosítani kívánt felhasználó azonosítóját. Headerben a User adatokat és az access token-t. Ha érvényes a token leelenőrzi, hogy a sima felhasználó a saját adatát szeretné -e módosítani. Ha nem akkor visszadob egy hibaüzenetet (***FORBIDDEN***), ha a felhasználó admin akkor van jogosultsága módosítani bármely felhasználót. Nem kötelező minden adatot megadni csak azt, amit módosítani szeretnénk! Amikor a sima user a saját adatait módosítja akkor nem tudja megváltoztatni az admin jogát mivel a tokenből ellenőrzi a jogosultságot és eldönti melyik metódust hívja meg. Ha nincs jogosultság visszadobja a FORBIDDEN-t. Ha nem találja a felhasználót akkor ***NOT FOUND***, ha sikeres akkor OK.

PUT http://localhost:8080/user/3  
*Content-Type*: application/json  
*Authorization*: Bearer <access token>  
  
{  
 "email": "tesztelek3@gmail.com"  
}

***Pizzákkal kapcsolatos endpointok***

**Összes pizza lekérése**<http://localhost:8080/pizza/get-all>

Kilistázza az összes pizzát és elküldi JSON formában, ehhez nem kér semmi token-t mivel bejelentkezés nélkül is kilistázható a web oldalon.

GET http://localhost:8080/pizza/get-all

**Pizza hozzáadása**  
<http://localhost:8080/pizza/add-new>

Megkapja a pizza adatait body-ban és egy access token-t a headerben, ha a token leját **451**-es hibakódot küld vissza. Token érvényesség ellenőrzést követően ellenőrzi, hogy a felhasználó admin jogosultsággal rendelkezik -e, ha nem visszadobja a 403-as ***FORBIDDEN*** hibakódot. Ezt követően hozzáadja az adatbázishoz és visszaküli a **OK** response code-ot. Ha létezik az adatbázisban visszaküldi a ***CONFLICT*** (409) status code-ot.

POST http://localhost:8080/pizza/add-pizza  
*Content-Type*: application/json  
*Authorization*: Bearer <access token>  
  
{  
 "name": "Húsimádó2",  
 "price": "2230",  
 "description": "A világ egyik legnépszerűbb pizzája a pepperónis, amelyet Kínától, az Egyesült Államokon át Belgiumig minden pizzériában tudunk rendelni.",  
 "picture": "https://i.imgur.com/pK3cuaw.png",  
 "available": false  
}

**Pizza módosítása**  
[http://localhost:8080/pizza/{pizzaId}](http://localhost:8080/pizza/%7bpizzaId%7d)

Paraméterként megkapja a pizza azonosítóját, headerben a módosított pizza adatait és egy access token-t. Ellenőrzést követően feltölti a módosított pizzát az adatbázisba és visszaküld egy 200-as kódot (**OK**), ha az azonosító nem található akkor egy 404-es (**NOT FOUND**) hiba kódot. Ha az accessToken lejárt akkor egy **451**-es kódot. Nem kötelező minden adatot megadni.

PUT http://localhost:8080/pizza/6  
*Content-Type*: application/json  
*Authorization*: Bearer <access token>  
  
{  
 "name": "Sonkássss",  
 "price": "1765",  
 "description": "Asd",  
 "picture": "xd"  
}

***Rendeléssel kapcsolatos endpointok***

**Rendelés lekérdezése felhasználó azonosító alapján**  
<http://localhost:8080/order/get-orders>

Headerben megkapja az access token-t majd ellenőrzi hogy érvényes -e. Ha lejárt a token vissza küld egy **451**-es hibakódot. Ha sikeres volt az azonosítás akkor elküldi az adott felhasználóhoz tartozó rendeléseket (azonosítót a tokenből szedi ki). Ha viszont valamilyen oknál fogva sikertelen eredménye lenne akkor ***INTERNAL\_SERVER\_ERROR*** hibaüzenetet dobja vissza.

GET http://localhost:8080/order/get-orders  
*Authorization*: Bearer <access token>

**Összes új rendelés lekérdezése**  
<http://localhost8080/order/get-new-orders>

Minden új rendelést visszaküld ha érvényes az access token illetve ha a felhasználó admin.  
Ha a token lejárt **451**-es hibakódot küld vissza. Ha a felhasználó nem admin akkor ***FORBIDDEN*** hibakódot. Ha viszont valamilyen oknál fogva sikertelen eredménye lenne akkor ***INTERNAL\_SERVER\_ERROR*** hibaüzenetet dobja vissza.

GET http://localhost:8080/order/get-new-orders  
*Authorization*: Bearer <access token>

**Új rendelés felvétele**  
<http://localhost:8080/order/add-order>

Megkapja Json formában az orderDto típust, ami egy tartózkodási hely, telefonszám és a pizza azonosítók listája. Leellenőrzi a token érvényességét, majd kiszedi belőle a felhasználó email címét és az alapján szerzi meg a felhasználó azonosító kódját. A pizzáknak az árát összesíti és feltölti a felhasználó id-t, tartózkodási helyet, feltöltés dátumát, összesített árat, telefonszámot és egy olyan változót hamisként, hogy készen van -e. Minden adat megadása kötelező, különben ***BAD\_REQUEST*** hibaüzenetet dob vissza.

POST http://localhost:8080/order/add-order  
*Content-Type*: application/json  
*Authorization*: Bearer <access token>  
  
{  
 "location": "Budapest, Thököly út 48-54, 1146",  
 "phoneNumber": "06305489346",  
 "pizzaIds": [ 1, 4, 5, 5, 5 ]  
}

**Rendelés módosítása**  
[http://localhost:8080/order/{orderId}](http://localhost:8080/order/%7borderId%7d)

A rendeléseknek csak a státuszát lehet módosítani adatvédelmi szempontból!

Paraméterként megkapja a rendelés azonosítót, headerből az accessTokent és a bodyból az új order-t és az alapján módosítja az adatbázist. Persze itt is rendelkezni kell érvényes access token-el és admin jogosultsággal. Ha a token lejárt visszadobja a **451**-es hibakódot. Ha a felhasználó nem admin akkor visszadobja a 403-as ***FORBIDDEN*** hibakódot.

PUT http://localhost:8080/order/1  
*Content-Type*: application/json  
*Authorization*: Bearer <access token>  
  
{  
 "ready": "true"  
}