MOVIES

Vizsgaremek

Helyfoglalási rendszer

A vizsgaremek egy, elsősorban mozik számára készült helyfoglalási rendszer.

Kenyó László, Szöllősi Norbert, Balogh Csaba

Vizsgaremek

Movies

A vizsgaremek egy, elsősorban mozik számára készült helyfoglalási rendszer, mely a következő igények kiszolgálására valósult meg:

- Interaktív webes felület, amin megjelennek a mozik filmkínálatai.
- Interaktív felület, ahol a vendégek könnyen, intuitív módon tudják eldönteni, hogy mely helyekre szeretnének jegyet foglalni.
- Asztali és mobil készüléken is jól látható, kezelhető felület.
- Regisztrációs és autentikációs felület.
- Nyomon követhető foglalási rendszer, ahol a felhasználó és az adminisztrátor átlátható képet kap a foglalás állapotáról.
- Adminisztrációs felület, ahol az adminisztrátor joggal rendelkező felhasználók könnyedén hozzáadhatnak és törölhetnek adatokat.
- Méretezhetőség, hatékonyság, kis- és nagyméretű mozik számára.

A projekt gerincét két fejlesztői környezet adja:

- **Django-rest-framework**, rest API backend
- **Vite-React** JavaScript frontend könyvtár

A forráskövető rendszer a **GIT**, melyet a **github.com** oldalon használtunk.

- generatív, nagy nyelvi modellek:
  - **Microsoft Copilot**
  - **Github Copilot**
- kódszerkesztő:
  - **Visual Studio Code**

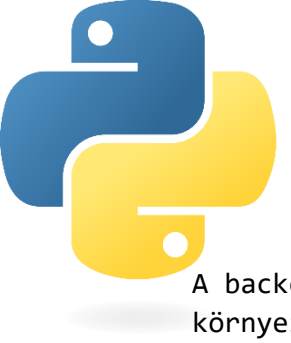
A működés alapelve

Minden ülést egy frontenden megjelenő apró négyzet reprezentál. A háttérben a frontend a backend által küldött **JSON** adatmodellt interpretálja a weboldal megjelenítésekor. A *json* tartalma egy *dictionary*, ahol a *kulcsok stringek*, pontosabban a *seat* szó + ülések száma, pl. “*seat_003*”. A *dictionary* értékei *boolean* változók. Kétféle foglalás van. Az előfoglalt ülés értéke *false*, a foglalt ülés értéke *true*. A szabad ülés értéke *null*, pontosabban nem képezi részét a dictionary-nek. Ez is azt a célt szolgálja, hogy minél kisebb mennyiségű adatot kelljen a frontend és

a backend között továbbítani. A frontend a terem kapacitásának függvényében hoz létre változókat, melyek nevei megegyeznek a backendről származó dictionary kulcsaival (pl. “seat_043”). Amikor a kulcshoz tartozó érték true, az azonos nevű változó értéke 2 lesz, ha false, akkor pedig 1. Ha pedig az érték null, tehát nem található a dictionary-ben, akkor a változó értéke 0 lesz. A frontend egy tömböt tölt fel ezekkel a változókkal. A megjelenítés folyamán minden változóra jut egy HTML elem (div) melyek színes négyzetekként jelölik az ülések foglaltságát. A 0 értékű változó eleme zöld, tehát a szabad ülés zöld színű. Az 1 értékű változó eleme piros színű, ez az előfoglalt ülés. A 2 értékű változóhoz tartozó elem színe szürke, ez reprezentálja a foglalt ülést. Az ülések szimbólumai, egy, a moziterem kialakítására nagyban hasonlító felülnézetes térképen jelennek meg a képernyőn. Az ülések sorrendje és elhelyezkedése hűen tükrözi a valóságot. Ezen a felületen megtalálhatóak egyéb fontos elemek is, például a mozivászon, a bejárat, de az átjárók is. Az ülések gombokként működnek, ha a leendő vendég rákattint a neki tetsző zöld ülésre, akkor az piros színű lesz, és a hozzá kapcsolódó változó értéke 0-ról 1-re változik. Amennyiben a vendég meggondolja magát, és újra rákattint a piros ülésre, akkor az újra zöld lesz, a változó értéke pedig visszaáll nullára. A szürke ülésekre nem lehet kattintani, hiszen azokat már lefoglalta valaki. Amikor a vendég a foglalás gombra kattint, a frontend interpretálja a változókat a következő módon:

- Az 1 értékű változók alapján létrehoz egy dictionaryt, ezt **HTTP PATCH** kéréssel elküldi a backendnek. Itt is megvalósul az elv mely szerint csak a szükséges adatokat küldjük el.
- A backend a http kérés alapján megkeresi a Venue példányt és foglaltra változtatja a helyeket.
- A frontend egy **HTTP POST** kérést is küld a backendre, az pedig létrehoz egy Reservation példányt.

A másik fontos mozzanat a foglalás törlése. A frontend a megfelelő backend API-végpontról lekéri a felhasználó foglalásait, és megjeleníti azokat egy aloldalon. Amikor a felhasználó az adott foglalás törlése mellett dönt, akkor a frontend egy **HTTP DELETE** kérést küld, a backend a foglaláshoz kötődő helyeket pedig „felszabadítja” majd törli a foglalás példányt. Így a helyek újra felszabadulnak a többi vevő számára.



Backend

A backend alapja a **Python3** nyelven íródott **Django** fejlesztési környezetben, illetve a következő Python könyvtárak segítségével:

- **Django rest-framework**
  - API-végpontok kialakításához, és az adatok szerializációjához
- **Pillow**
  - képkezelő könyvtár
- **django-cors-headers**
  - Cross-Origin Resource Sharing (CORS) headerek hozzáadása a kérésekhez, a biztonságos források eléréséért
- **JSON Web Tokens**
  - kódolt token párokat használó könyvtár az autentikációhoz

Modellek

A projekt a következő osztály alapú adatmodellekre épül:

- **Show**
  - Maga a **film**
  - Tartalma:
    - a film címe
    - a film plakátja
    - korhatára
    - játékidő (percben)
    - rövid leírása
    - **Venue**: a következő adatmodell many-to-one jelleggel kapcsolódik a show modellhez, tehát egy filmhez több Venue kötődhet, de egy Venue csak egy filmhez kötődhet.

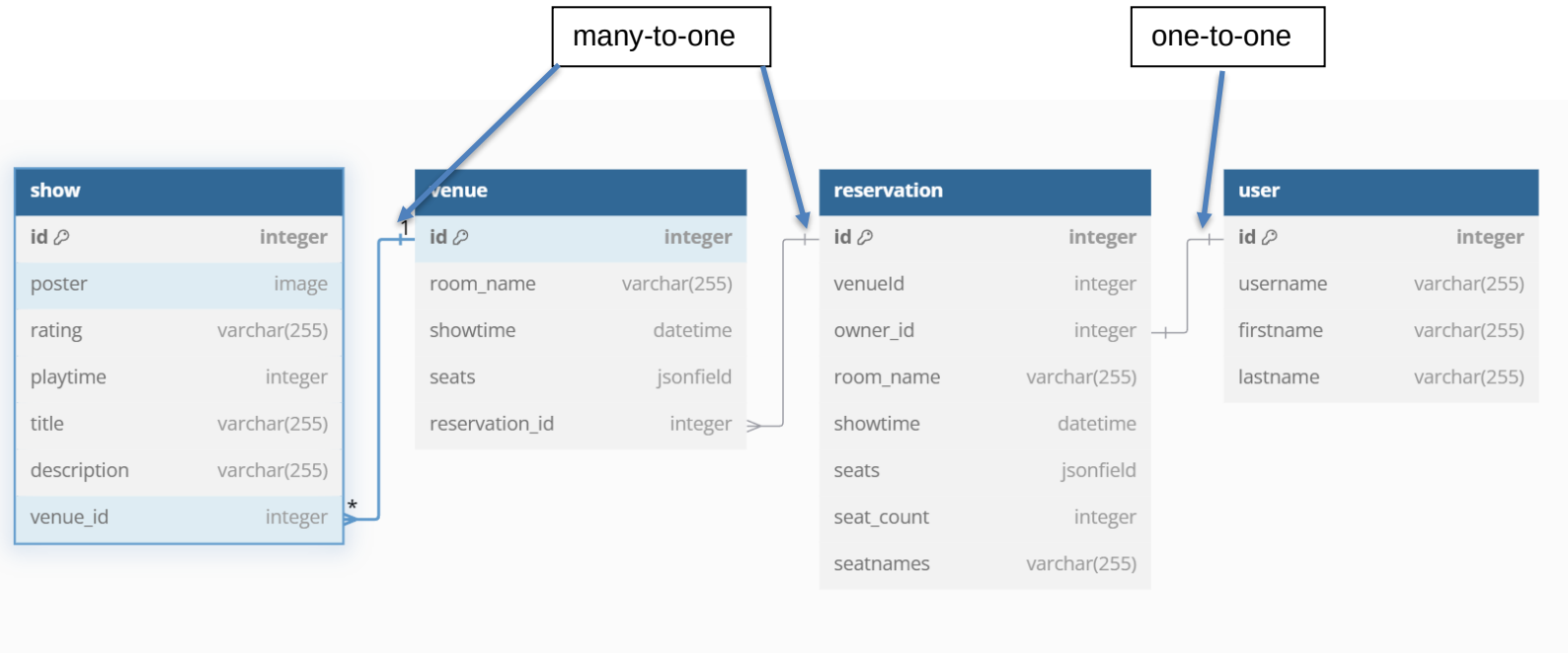
- **Venue**
  - A **vetítést** reprezentáló adatmodell
  - Tartalma:
    - a film címe
    - a terem neve
    - a vetítés időpontja
    - a modell gerincét adó jsonfield adatmező, mely a foglalt és előfoglalt helyeket szimbolizálja
    - a terem stílusa (a frontenden van szerepe) **RoomStyleDict**, one-to-many kapcsolattal, tehát egy Venue-hoz egy stílus kapcsolódhat, de egy stílus bármennyi Venue-hoz kapcsolódhat
    - **Reservation:** „foglalás”, a következő adatmodell many-to-one jelleggel, tehát egy Venue-hoz kötődhet sok foglalás, de egy foglalás csak egy Venue-hoz kötődhet

- **Reservation**
  - A **foglalást** reprezentáló adatmodell
  - Tartalma:
    - a Venue id-ja
    - a film címe
    - a felhasználó, aki a foglalást végezte, egy foglaláshoz csak egy felhasználó kötődhet
    - a terem neve
    - a vetítés időpontja
    - az ülések (JSONField)
    - a foglalt ülések száma
    - a lefoglalt ülések nevei, úgy mint 3. sor, 7. szék stb.

- **CustomUser**
  - a **felhasználó**
  - tartalma:
    - keresztnév
    - vezetéknév
    - felhasználónév
    - email cím
    - jelszó (hash digest)

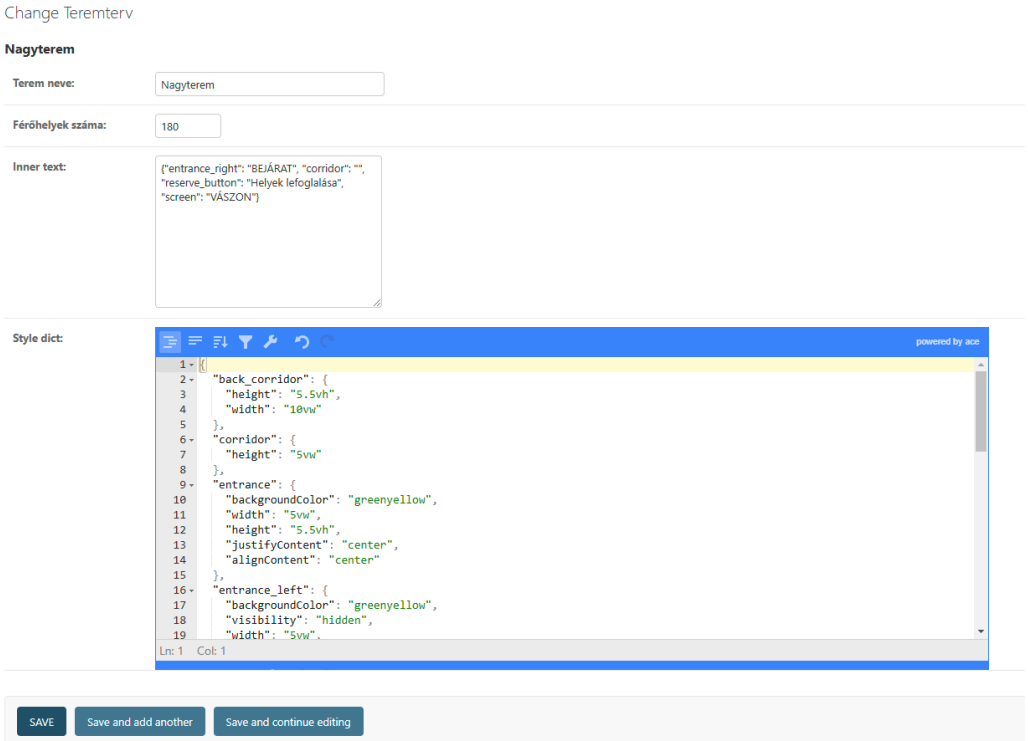
- **RoomStyleDict**
  - egy modell, arra a célra, ha a termek megjelenésén akarunk változtatni anélkül, hogy a frontenden módosítást végeznénk, illetve új termeket is létrehozhatunk vele
  - tartalma:
    - terem neve
    - férőhelyek száma
    - belső szöveg (pl. „vászon”, „bejárat”), dictionary
    - style_dict, lényegében egy dictionary, amit a frontend felhasznál a moziterem elemeinél, lényegében a CSS-t váltja ki pl:
      - {inner_text[’entrance_left’]}
      - ez azért lehet célszerű, mert így az ügyfél közvetlenül a backendről tud újfajta termet hozzáadni a mozihoz, illetve átalakítani azokat

Relációk

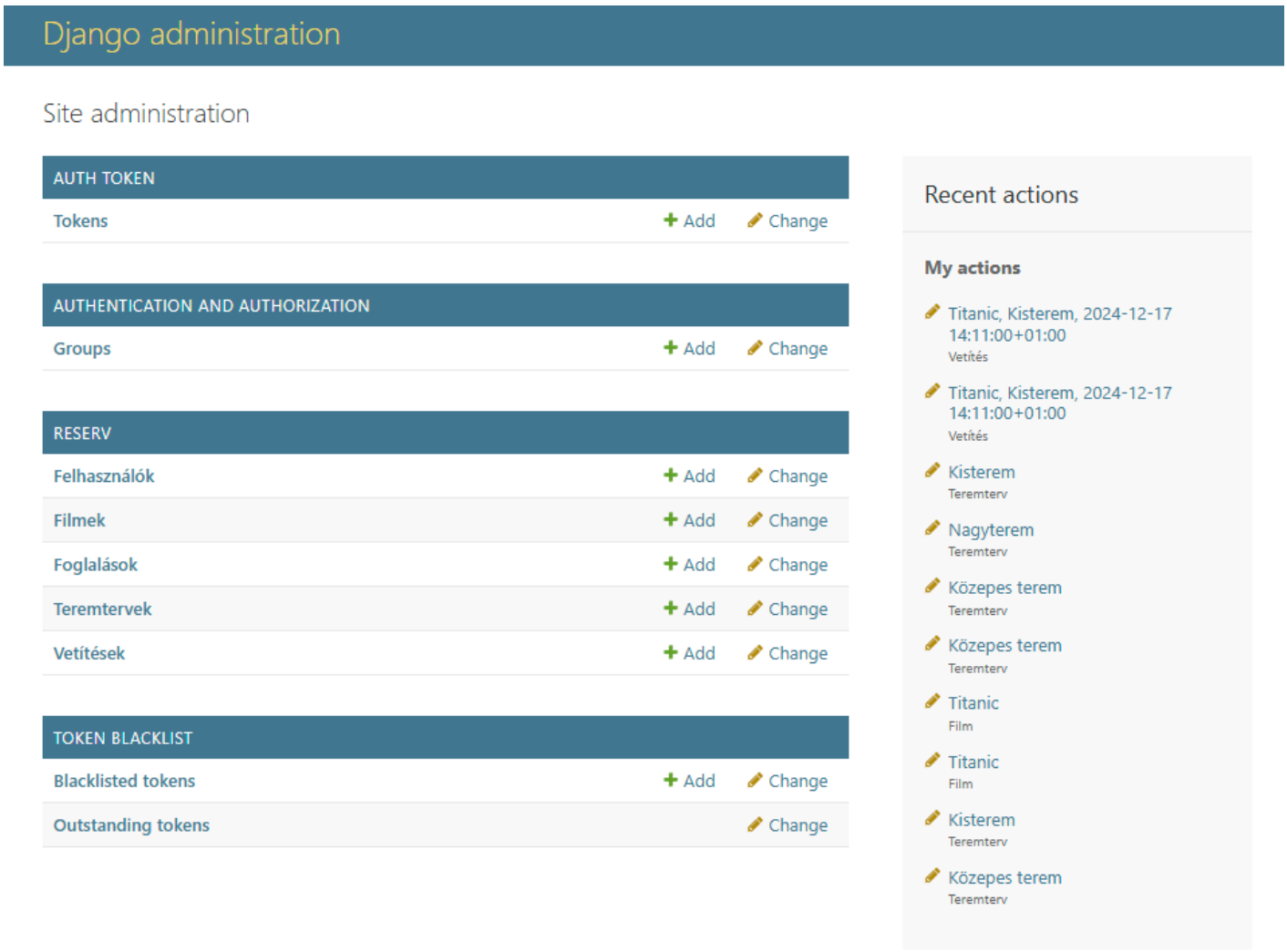


Adminisztrációs felület

A csapat a Django által biztosított jól bevált admin felületet használta ki. A backenden a ModelAdmin osztály segítségével regisztráltuk a modelleket.
A fent említett RoomStyleDict szerkesztése nehezen átlátható, ezért egy widgetet is meghívtunk, melynek neve JSONEditorWidget. Ezt a widgetet egy formba helyezve és a formot meghívva regisztráltuk a ModelAdmin segítségével.

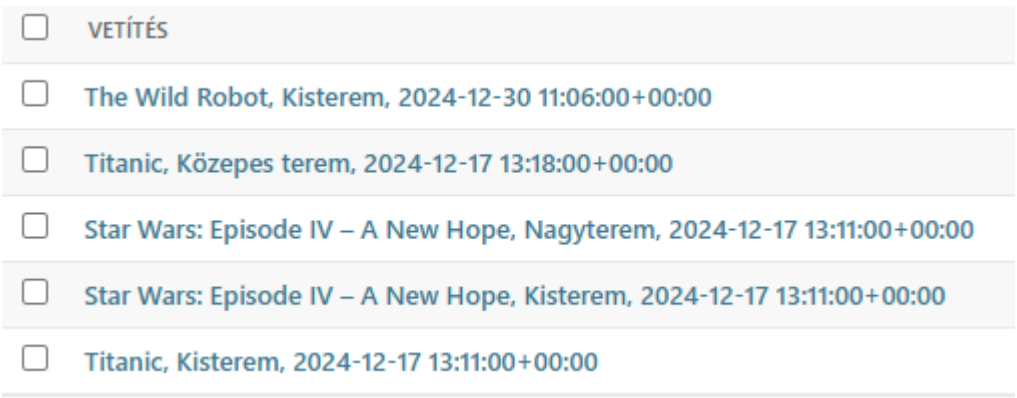


A Meta osztály segítségével magyar nyelvű egyes- és többesszámú nevekkel is elláttuk a modelleket.



Az osztályok __str__(self) metódusával pedig megoldottuk, hogy az objektumok jól felismerhetőek, megkülönböztethetőek legyenek:
Venue:

```
def __str__(self) -> str:
        return self.title+', '+self.room_name+', '+str(self.showtime)
```



A ModelAdmin segítségével kihagytuk a hosszas mezőket a list displayből:

```
class ReservationAdmin(admin.ModelAdmin):
    list_display=['owner','title','room_name','seat_count','showtime']
    pass
```

Funkciók

A backend elsősorban osztály alapú nézeteket használ. Ezeket a nézeteket (attribútumaikat, metódusaikat) írják felül a rest-framework generic nézetei. Összetettebb esetekben szükség volt a nézetek metódusait is felülírni. Az alábbiakban erre hozok fel egy példát:

```
class ReservDestroy(generics.DestroyAPIView):
    permission_classes =[IsAuthenticated]
    queryset=Reservation.objects.all()
    serializer_class=ReservSerializer
    def destroy(self, request, *args, **kwargs):
            instance: object = self.get_object()
            venue: object = Venue.objects.get(id=instance.venueId)
            venue=seat_liberator2(instance,venue)
            print('venue: ',venue.seats)
            print('instance',instance.seats)
            self.perform_destroy(instance)
            venue.save()
            return Response(status=status.HTTP_204_NO_CONTENT)
    def perform_destroy(self, instance):
        instance.delete()
```

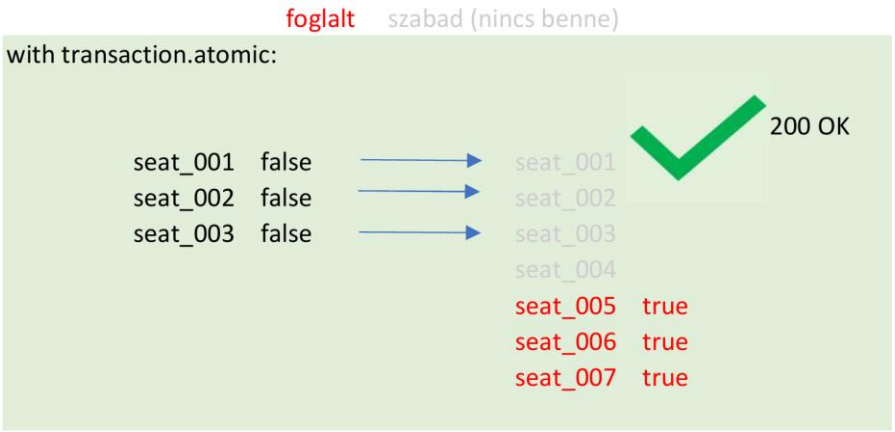
Ezen esetben a foglalás törléséért felelős nézetet látjuk. A destroy metódus itt nem csak a példányt törli, hanem megkeresi a vetítés példányt, majd a két JSONField adatmezőt egymáshoz vetve törli a foglalásban szereplő székeket a vetítés adataiból, így szabaddá téve azokat. A foglalás példány törlése csak ezután történik meg.

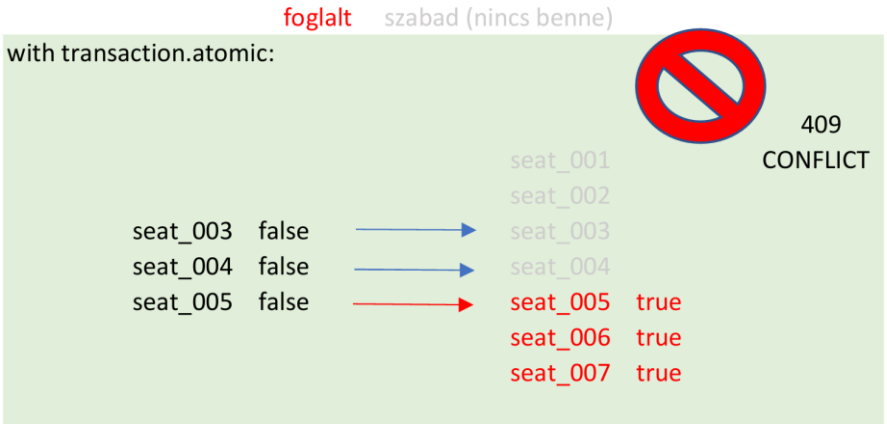
A kódban látható a „seat_liberator2” függvény. A kód átláthatósága végett az ülésekkel kapcsolatos függvényeket egy saját magunk által írt modulban, a „seathandler” azaz „ülés kezelő” modulban írtuk meg.

```
from .seathandler.seathandler import reserv_data_maker,venue_data_dict_maker,\
                                     venue_data_updater2,seat_liberator2,\
                                     validate
```

Helyfoglalás

A bevezetőben is taglaltak szerint, a backend a frontendtől kap egy PATCH http kérést, amit a partial_update metódus kezel. Mivel várhatóan egyszerre több ember is foglalhat jegyet ugyanarra a filmre, fontos felkészülni az esetleges konfliktusokra is. Ezt a Django transaction modullal oldottuk meg, azon belül is az atomic() metódussal. Ennek a metódusnak a lényege, hogy a try blokkban levő folyamatok vagy egységesen (atomikusan) lefutnak, vagy visszaállnak a belépési pont előtti állapotra. A seathandler.validate() függvény ellenőrzi, hogy az előfoglalt helyek nincsenek-e már benne a vetítés üléseiben, tehát nem foglaltak-e. Amennyiben nincsenek, úgy ezeket a változókat hozzáadja a vetítés JSONFieldjéhez, true értékkel, és 200-as OK http kóddal jelzi a frontendnek, hogy rendben ment a foglalás, majd a serializer elvégzi a módosításokat. Ha valamelyik széket már lefoglalták, akkor http 409-es CONFLICT üzenetet küld a frontendnek.

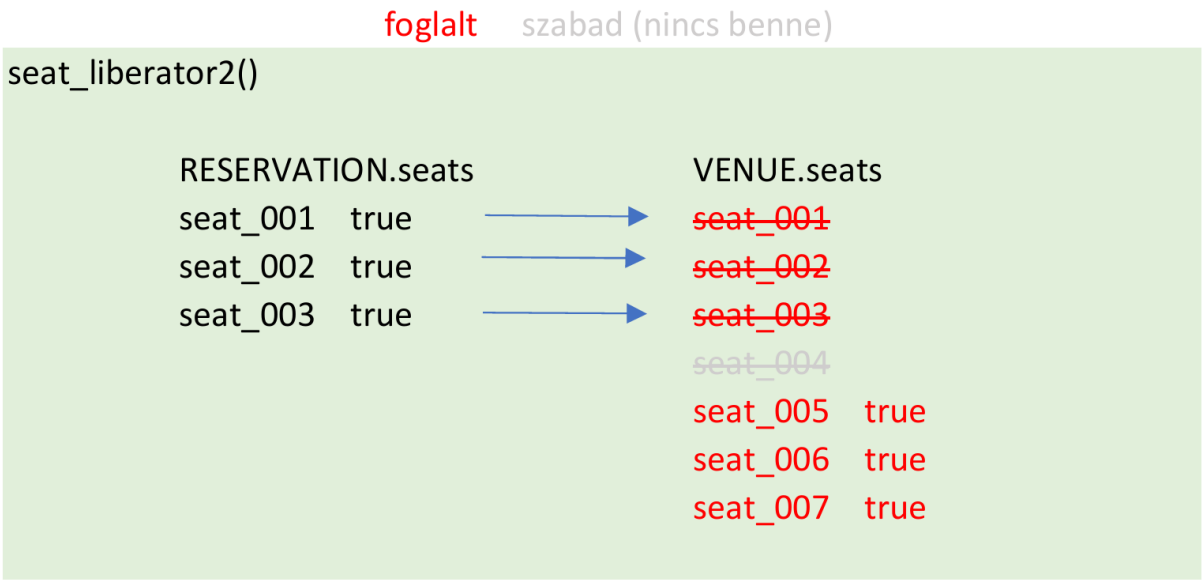




Foglalás törlése

A törlést a rest-framework generics.DestroyAPIView nézetével oldottuk meg. Az ezen nézetet használó URL-re csak DELETE http kérést lehet küldeni. Természetesen csak autentikált kérést fogad el a nézet. Itt a seathandler modul seat_liberator2 függvénye végzi el a helyek felszabadítását a következő módon:

- a Django a nézet példány self.get_object() metódusával megkeresi a Reservation azaz a foglalás példányt
- a Model osztály Venue.objects.get() metódussal megkeresi az érintett Venue, azaz a vetítés példányt
- összeveti a foglalás üléseit a vetítés üléseivel, kitörölve az érintett üléseket a vetítés JSONFieldjéből
- végül törli a foglalás példányt



Adminisztratív funkciók

A használat megkönnyítése céljából a frontend oldalról is lehetővé tettük az olyan funkciókat, mint a

- film hozzáadása
- vetítés hozzáadása adott filmhez
- film törlése
- vetítés törlése

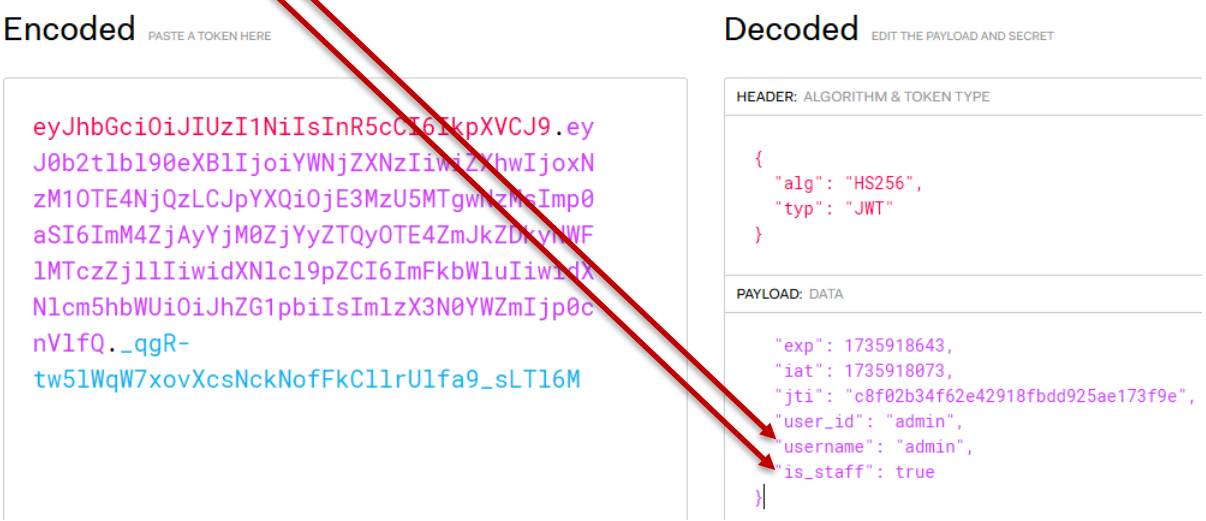
Így olyan .is_staff attribútummal rendelkező felhasználó is elvégezheti ezeket a műveleteket, akik valamilyen okból nem férnek hozzá a Django adminisztrációs felületéhez.
Ez akkor is célravezető lehet, ha a backend külön domainen helyezkedik el. Ezeket a funkciókat is a rest framework generic nézeteivel oldottuk meg.

Az alkalmazás autentikációját a **JSON Web Token** könyvtár segítségével oldottuk meg. Ez a megközelítés két tokenből álló párt használ, az *access* és a *refresh* tokent. Az access token szükséges minden olyan http kéréshez, ami autentikációhoz kötött (permission_classes =[IsAuthenticated]). Az access token rövid ideig érvényes, emiatt a visszaélésre alkalmas idő is kisebb. A refresh token élettartama hosszabb, ezt akár hetekre is eltárolhatja a böngésző. A biztonság érdekében a backend „**rotálja**” a refresh tokent, tehát amikor a frontend egy új access tokent kér, akkor nem csak az access token, hanem a refresh token is megújul. Ez azért létfontosságú, mert ha nem így lenne, akkor egy jogosulatlanul megszerzett refresh tokennel korlátlan számú access tokent lehetne kérni. Az elhasznált tokeneket a Django feketelistára teszi, így megelőzve a jogtalan „újrahasznosítást”.

```python
SIMPLE_JWT = {
    'USER_ID_FIELD': 'username',
    'USER_ID_CLAIM': 'user_id',
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=5),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
    'ROTATE_REFRESH_TOKENS': True,
    'BLACKLIST_AFTER_ROTATION': True,
    'AUTH_COOKIE_HTTP_ONLY': True,
    'AUTH_COOKIE_SAMESITE': 'Lax',
}
```

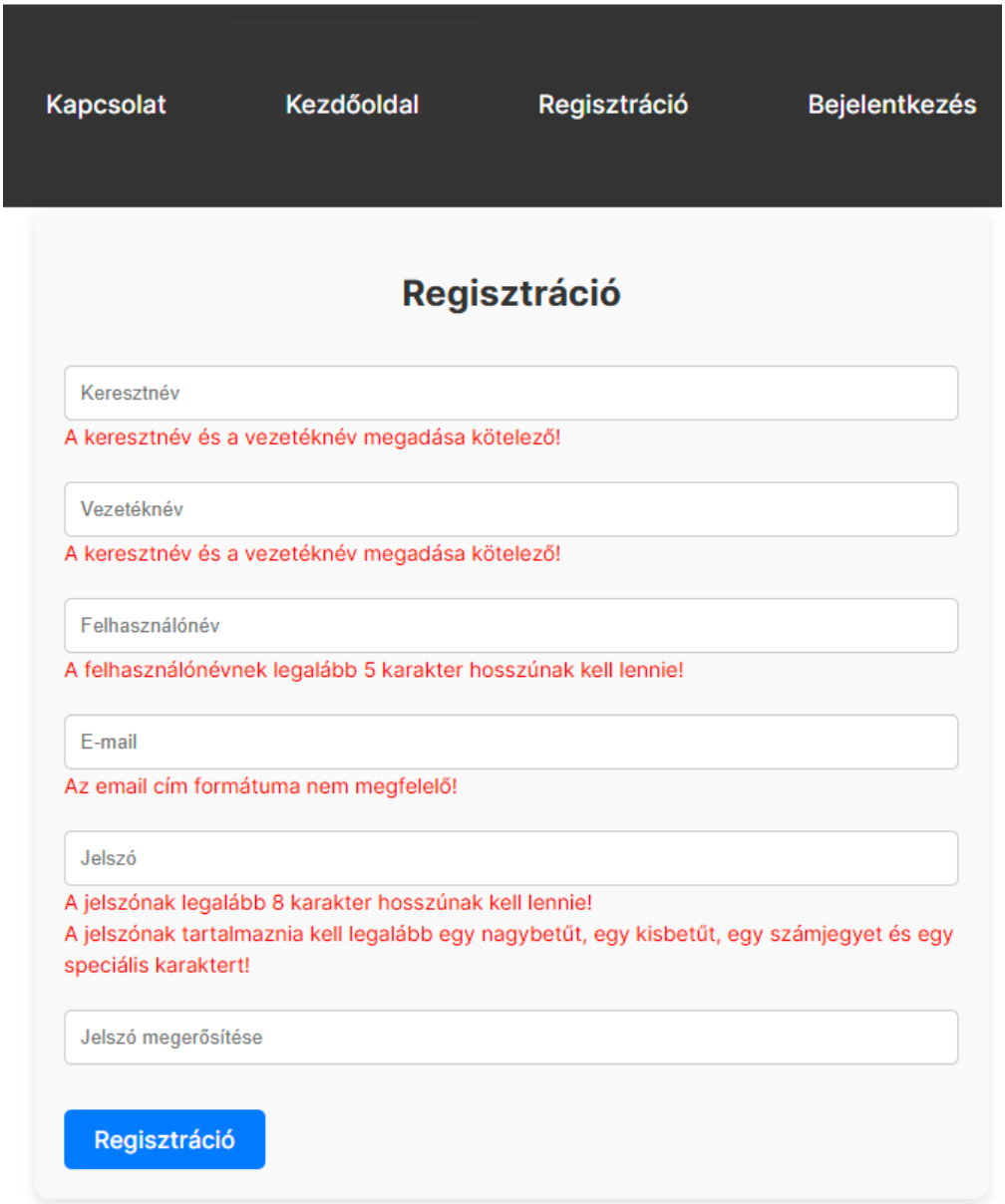
A frontenden szintén a JSON Web Token könyvtár JavaScript megfelelője dekódolja a tokent. A JWT könyvtár lehetővé teszi a Django számára, hogy „személyre szabjuk” a tokeneket, ezért a tokenek részét képezi a felhasználónév, és az is_staff státusz is, így ezeket az adatokat nem kell más helyeken elmenteni.

```python
class CustomTokenObtainPairSerializer(TokenObtainPairSerializer)
:
    @classmethod
    def get_token(cls, user):
        token = super().get_token(user)
        # Add custom claims
        token['username'] = user.username
        token['is_staff'] = user.is_staff
        return token
```



{token_type: 'refresh', exp: 1736004473, iat: 1735918073, jti: 'a6a20fc15ea74e6b8d4a5bc2d5141a88', user_id: 'admin', …}
exp: 1736004473
iat: 1735918073
is_staff: true
jti: "a6a20fc15ea74e6b8d4a5bc2d5141a88"
token_type: "refresh"
user_id: "admin"
username: "admin"

A frontenden a biztonság elsősorban a regisztrációnál a jelszóbiztonság kikényszerítésében összpontosul.

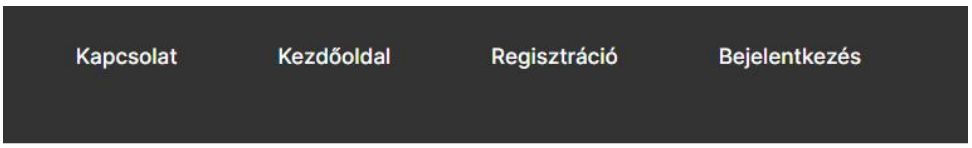


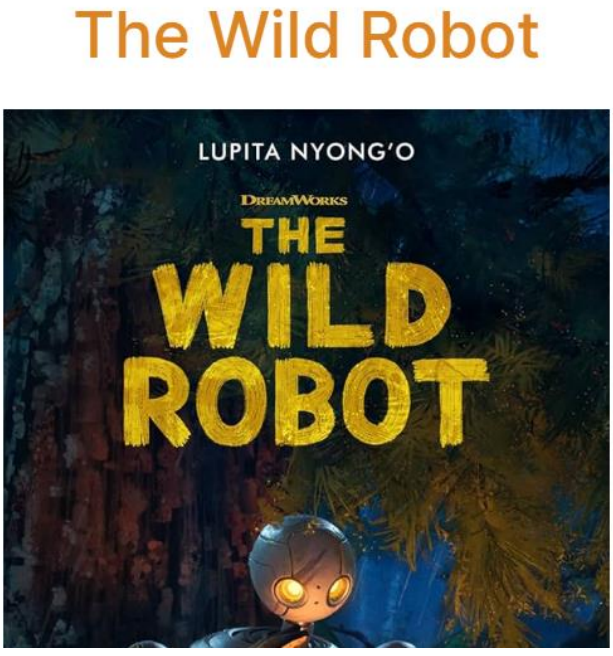
Frontend

A frontend alapja egy, a Node.js futtatási környezetben futó Vite fejlesztési szerver, amin a React nevű, nyílt forráskódú JavaScript könyvtárat használtuk.
Tekintettel a jó prezentálhatóságra, az áttekinthetőségre és az időkorlátokra, a letisztult, minimalista dizájn mellett döntöttünk.

A frontend felépítése a következő:

- navigációs sáv:
  - bejelentkezve
    - Kapcsolat
    - Kezdőoldal
    - Üdvözlő szöveg
    - Foglalások
    - Kijelentkezés
  - kijelentkezve
    - Kapcsolat
    - Kezdőoldal
    - Regisztráció
    - Bejelentkezés



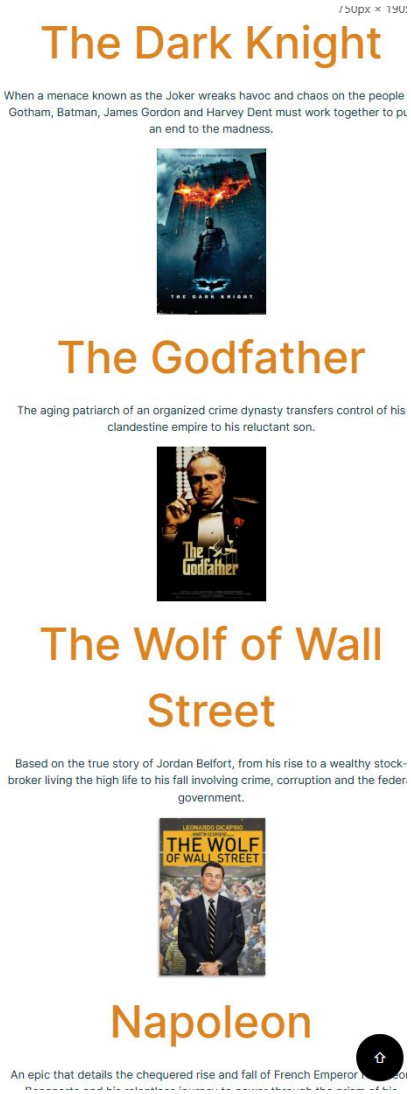


A kezdőoldal tagoltsága

A kezdőoldal tartalma kizárólag a műsoron levő filmekből áll, tehát vetítéseket nem tartalmaz. Ez részben az áttekinthetőséget szolgálja, részben pedig egyfajta pagináció, mely megakadályozza a nagy mennyiségű adat lekérését.
A kényelem kedvéért elhelyeztünk egy gombot a jobb alsó sarokban, amire kattintva felgördül a viewport az oldal tetejére.

A kezdőoldal kettő fő részből áll:

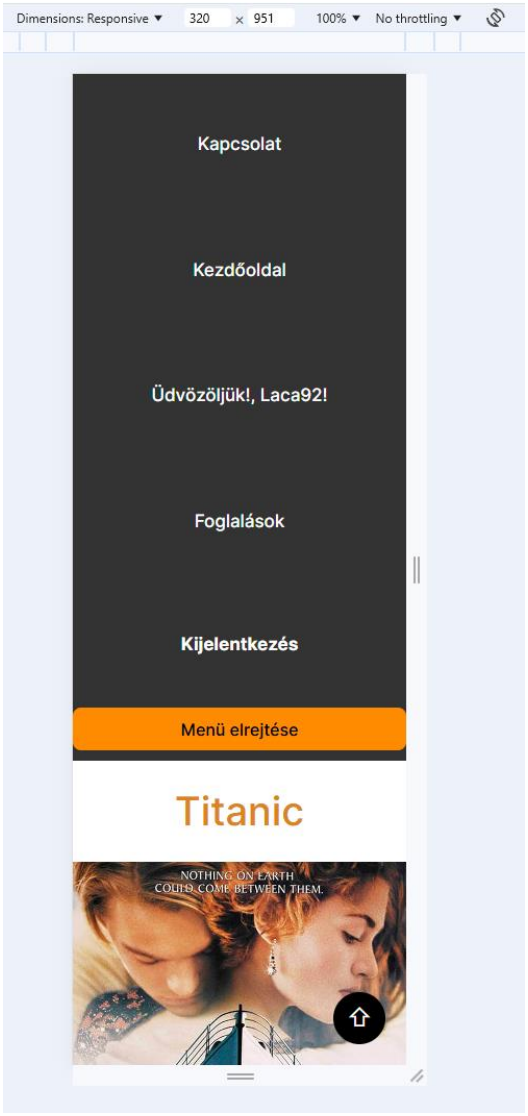
- diavetítés
- filmlista



Reszponzivitás

Az oldal szinte kizárólag **relatív** mértékegységeket használ, így mindig jól igazodik a viewport méretéhez.
A React **useEffect** funkciójának segítségével a frontend mindig tisztában van a viewport méretével, így, ha annak magassága, avagy szélessége 600 pixel alá csökken, akkor megváltozik a navigációs sáv felépítése, sőt a

helytakarékosság érdekében megjelenik egy gomb, mellyel „összecsukhatjuk” a menüt.

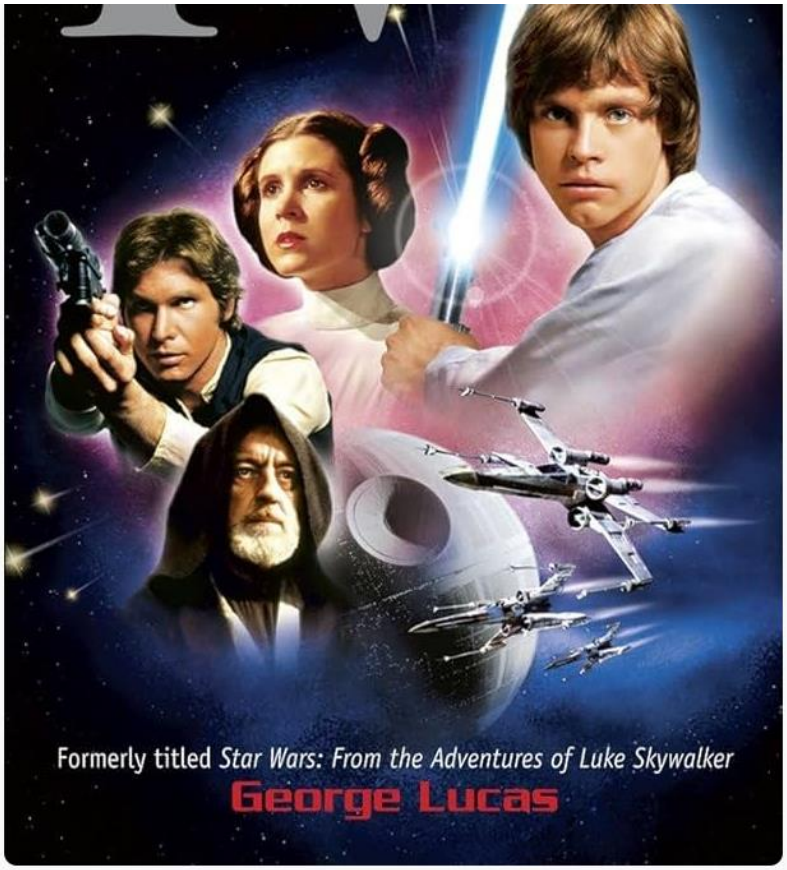


A CSS **@media** dekorátorának köszönhetően feltételekhez kötötten könnyen meg lehet változtatni adott attribútumokat, míg a többi attribútumot nem kell újra megadni.

```
@media (max-width: 600px) or (max-height: 600px) {
  .poster {
   min-width: 80vw;

  }
  .nav {
      display: flex;
      flex-direction: column;
```

Helyfoglalás

A filmcímre kattintva a következő aloldalon az elérhető vetítéseket látjuk, és eldönthetjük, hogy hány helyet szeretnénk lefoglalni.



Korhatár: 12 éven felülieknek.

Műsoridő: 121 perc

Vetítések

Foglalja le a helyeket

Hány helyet szeretne lefoglalni? 1

A terem neve: Kisterem

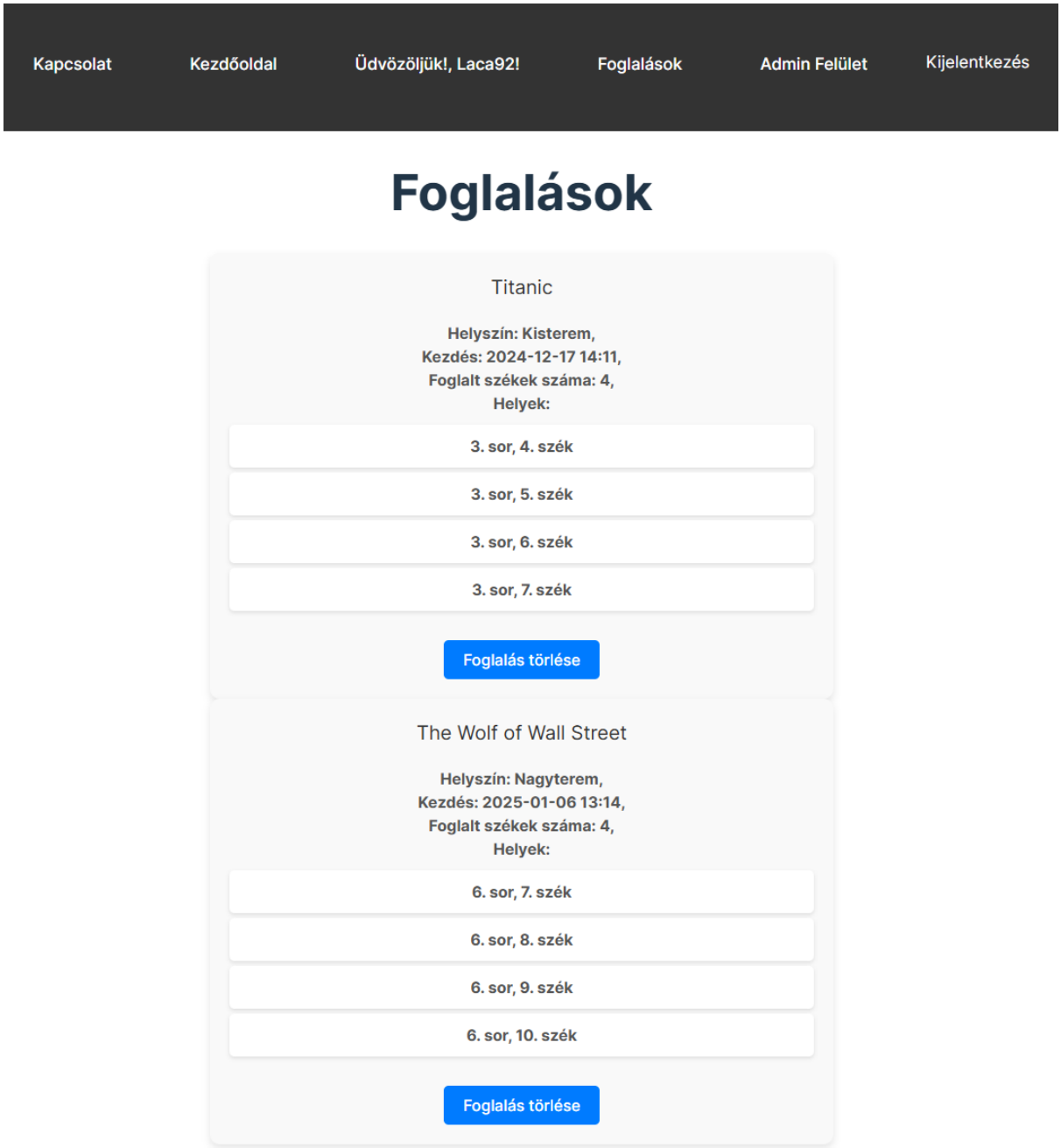
Vetítés kezdete: 2024. 12. 17. 14:11:00

A terem neve: Nagyterem

Vetítés kezdete: 2024. 12. 17. 14:11:00

Ezek után a megjelenő térképen a helyeket kell kiválasztanunk. Az oldal addig nem enged foglalni; amíg a kiválasztott számú helyet le nem foglaljuk.
Sikeres foglaláskor az ablak window.alert buborékkal közli velünk a foglalás megtörténtét. Ellenkező esetben a hiba jellegéről kapunk buborékot, és frissíteni kell az oldalt.

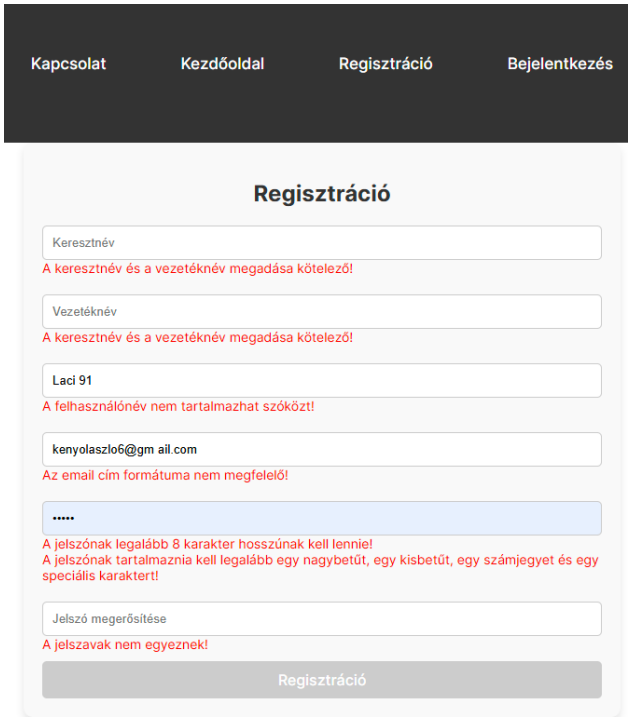
A navigációs sáv foglalások menüpontjára kattintva meggyőződhetünk róla, hogy a foglalásunk valóban sikeres volt-e.



Ha a foglalás törlése mellett döntünk, akkor az adott gombra kattintva megtehetjük azt, a foglalás törlődik a helyek pedig újra felszabadulnak.

Regisztráció

A biztonság érdekében a regisztrációnál regular expressiont illetve proptypes használatával felépített függvény ellenőrzi a jelszó erősségét és az adatok helyes formátumát, például az email címet.
Regisztráció előtt a frontend elküldi az email címet és a felhasználónevet egy külön erre a célra megírt nézetnek, amely választ küld aszerint, hogy a felhasználónév és az email cím foglalt-e már. Sikertelen regisztráció esetén window.alert buborék formájában figyelmeztetést kap a felhasználó. Sikeres regisztráció esetén a frontend be is jelentkezteti a felhasználót.

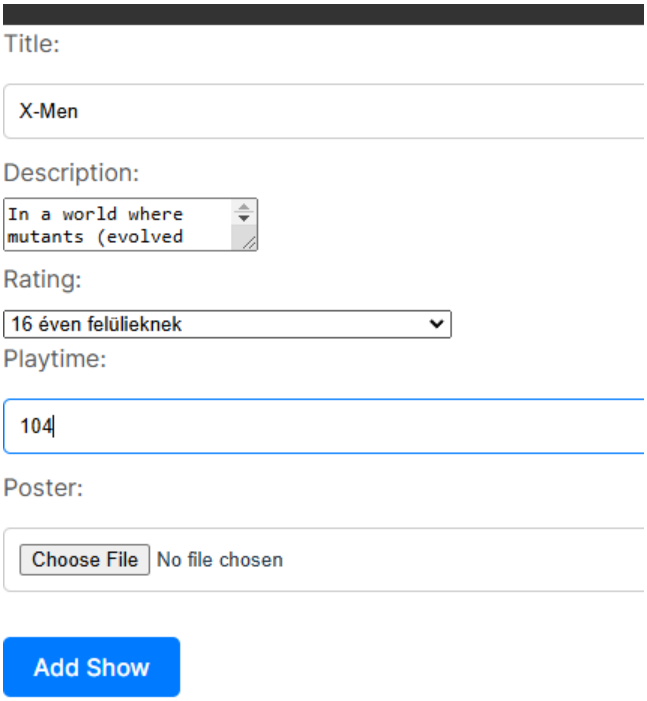


```
export const ValidateInputs = (firstname, lastname, email, username, password, confPW, setProblem, setButtonDisabled) => {
    let valid = true;
    let problemText = '';
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    const passRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[^\w\s])[A-Za-z\d\s\W]+$/;

    if (firstname.length < 1 || lastname.length < 1) {
        problemText += 'A keresztnév és a vezetéknév megadása kötelező!\n';
        valid = false;
    }
    if (username.length < 5) {
        problemText += 'A felhasználónévnek legalább 5 karakter hosszúnak kell lennie!\n';
        valid = false;
    }
    if (/\s/.test(username)) {
        problemText += 'A felhasználónév nem tartalmazhat szóközt!\n';
        valid = false;
    }
    if (!emailRegex.test(email)) {
        problemText += 'Az email cím formátuma nem megfelelő!\n';
        valid = false;
    }
    if (password.length < 8) {
        problemText += 'A jelszónak legalább 8 karakter hosszúnak kell lennie!\n';
        valid = false;
    }
    if (!passRegex.test(password)) {
        problemText += 'A jelszónak tartalmaznia kell legalább egy nagybetűt, egy kisbetűt,';
        problemText+= 'egy számjegyet és egy speciális karaktert!\n';
        valid = false;
    }
    if (password !== confPW) {
        problemText += 'A jelszavak nem egyeznek!\n';
        valid = false;
    }

    setProblem(problemText);
    setButtonDisabled(!valid);
    return valid;
}
```

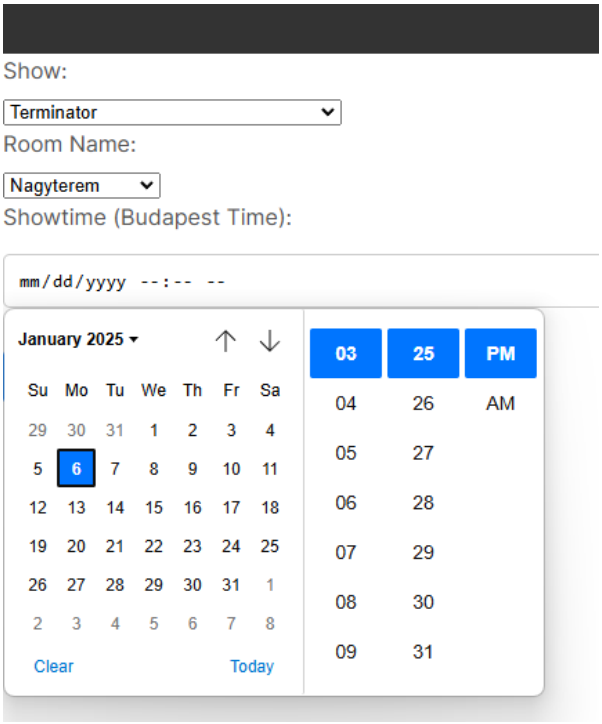
Adminisztratív funkciók

A frontend a személyre szabott tokeneknek köszönhetően eldönti, hogy adminisztrátor-e a felhasználó, és a menüt eszerint jeleníti meg. Az admin felületen a felhasználó mindig az API végpontokról lekért friss információt látja. A négy funkció:

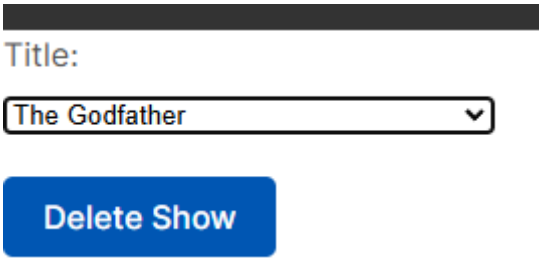
- Film hozzáadása
- Vetítés hozzáadása
- Film törlése
- Vetítés törlése



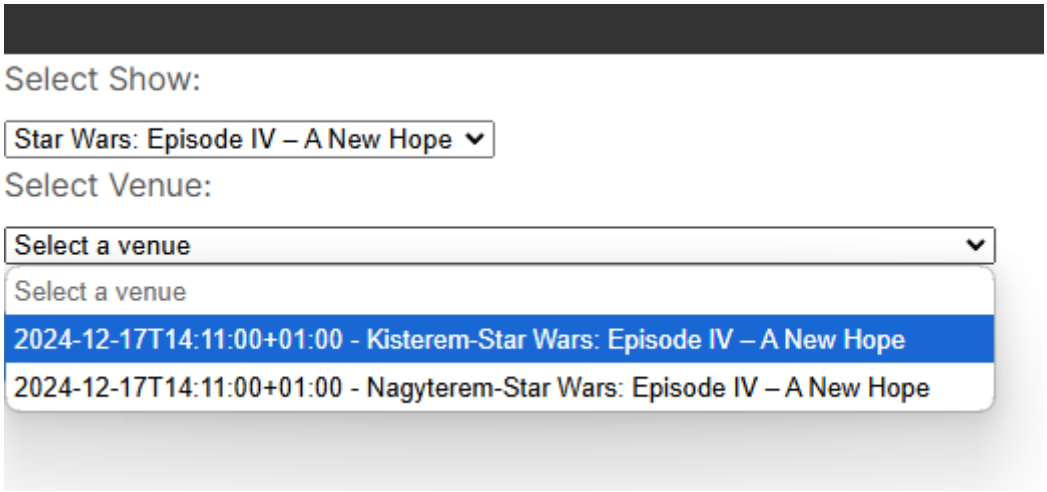
Film hozzáadása



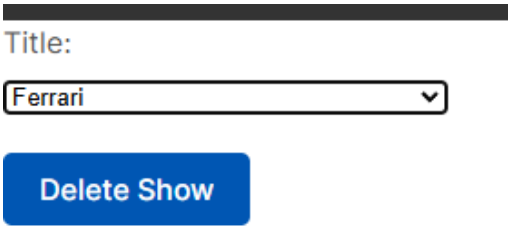
Vetítés hozzáadása



Film törlése

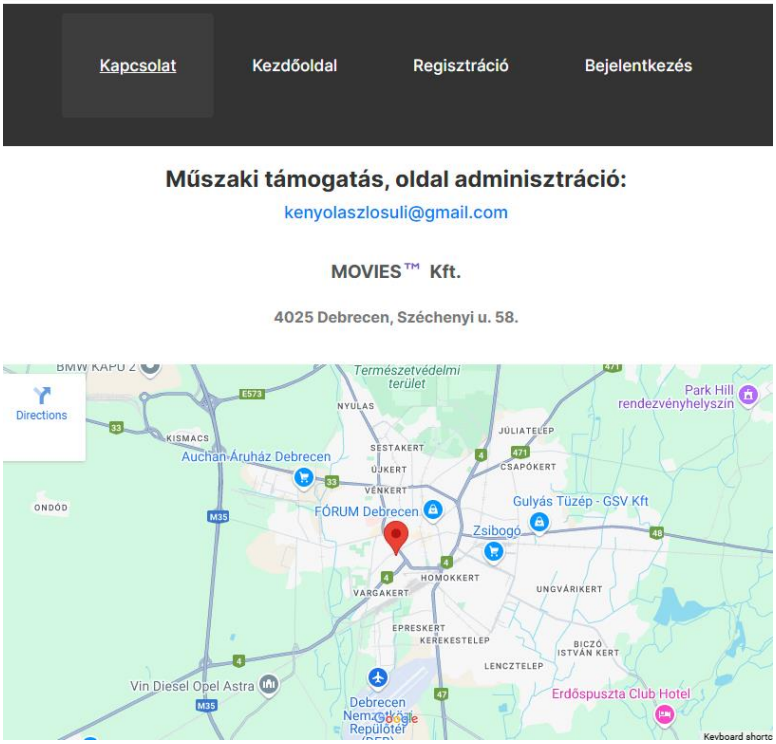


Vetítés törlése



Film törlése

A kapcsolattartást megkönnyítendő, az adott mozi egy kapcsolat oldalt is létrehozhat az alkalmazásban.



Köszönjük a megtisztelő figyelmet!

Tartalom


LIBERATE
SEATS