# Adatbázis dokumentáció:

## ’actings’ táblázat:

Adatkapcsolótáblázat az ’actors’ és ’games’ táblázathoz. Azt mutatja, hogy melyik játékokban milyen színészek játszottak, kiegészítve ezt a színész által eljátszott szereppel.

CREATE TABLE `actings` (

  `id` int(10) UNSIGNED NOT NULL,

  `role` varchar(100) DEFAULT NULL COMMENT 'A színész által eljátszott karakter',

  `GameId` int(10) UNSIGNED DEFAULT NULL,

  `ActorId` int(10) UNSIGNED DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A szénészek és játékokat kapcsolja össze kiegészítve a színész által eljátszott szereppel';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A role a színész által eljátszott karakter, karakterlánc, maximum 100 karakter, lehet üres is, mert van, olyan eset mikor a színész fel van tűntetve, de nincs tisztázva a szerepe. A GameId játék azonosítója a games táblázatból, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. Az ActorId színész azonosítója az actors táblázatból, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat.

ALTER TABLE `actings`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `Actings\_ActorId\_GameId\_unique` (`GameId`,`ActorId`),

  ADD KEY `ActorId` (`ActorId`);

Az id oszlopot elsődleges kulccsá tesszük. Megadjuk, hogy egy színész játék kapcsolat egyedi legyen, így nem ismétlődhet. Az ActorId-hez hozzárendeljük, hogy kulcs legyen.

ALTER TABLE `actings`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

ALTER TABLE `actings`

  ADD CONSTRAINT `actings\_ibfk\_1` FOREIGN KEY (`GameId`) REFERENCES `games` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT `actings\_ibfk\_2` FOREIGN KEY (`ActorId`) REFERENCES `actors` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

Az id-nek megadjuk, hogy az értékét automatikusan növelje. A GameId-hez és ActorId-hez hozzárendeljük, hogy hogy idegen kulcsként működjenek, előbbinél a games táblázat id oszlopának, az utóbbi esetén az actors tábla id oszlopához. Megadjuk, hogy ha bármely két szülő rekord elsődleges kulcsát megváltoztatják, a változtatást ez a tábla kövesse le, és változtassa meg az idegen kulcs értékét az új értékre. Ha egy szülő rekord törlésre kerül, akkor az összekötő rekord is kerüljön törlésre.

## ’actors’ táblázat:

Tárolja a játékokban megjelenő színészek nevét, illetve egy képet róluk.

CREATE TABLE `actors` (

  `id` int(10) UNSIGNED NOT NULL,

  `firstName` varchar(100) DEFAULT NULL COMMENT 'A színész keresztneve',

  `lastName` varchar(100) DEFAULT NULL COMMENT 'A színész vezetékneve',

  `profilePicture` varchar(255) DEFAULT NULL COMMENT 'A színészről tárolt kép elérési útvonala'

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A színészek adatait tárolja';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A firstName a színész keresztneve, ami karakterlánc, maximum 100 karakter, lehet üres. A lastName a színész vezetékneve, ami karakterlánc, maximum 100 karakter, lehet üres. A profilePicture a színész profilképének az elérési útvonala, ami, karakterlánc, maximum 255 karakter, lehet üres.

ALTER TABLE `actors`

  ADD PRIMARY KEY (`id`);

Megadjuk az id-nek, hogy elsődleges kulcs legyen.

ALTER TABLE `actors`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=62;

Az id-nek megadjuk, hogy az értékét automatikusan növelje.

## ‘agerating’ táblázat:

Korhatár besorolás a játékokhoz, mivel több intézet foglalkozik ezzel, ezért egy játéknak lehet több, akár különböző besorolása az intézményektől.

CREATE TABLE `ageratings` (

  `id` int(10) UNSIGNED NOT NULL,

  `rating` varchar(10) NOT NULL COMMENT 'A korhatár besorolás',

  `institution` varchar(4) NOT NULL COMMENT 'A korhatár besorolást intéző intézmény',

  `url` varchar(255) NOT NULL COMMENT 'A korhatár besorolás képének az elérési útvonala'

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='Korhatár besorolások adatait tartalmazó tábla';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A rating a korhatár besorolás, ami karakterlánc, maximum 10 karakter, nem lehet üres, mert kötelező adat. Az institution a korhatár besorolást adó szervezet, ami karakterlánc, maximum 4 karakter, nem lehet üres, mert kötelező adat. A GameId a játék azonosítója a ’games’ táblázatból, ami egész szám, nem lehet üres, mert kötelező adat.

ALTER TABLE `ageratings`

  ADD PRIMARY KEY (`id`);

Megadjuk, hogy az id legyen elsődleges kulcs.

ALTER TABLE `ageratings`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

Az azonosítónak megadjuk, hogy az értékét automatikusan növelje.

## ’creations’ táblázat:

A ’creators’ és ’games’ táblázatot köti össze kiegészítve ezt azzal, hogy a fejlesztő milyen területen járult hozzá a játék elkészítéséhez.

CREATE TABLE `creations` (

  `id` int(10) UNSIGNED NOT NULL,

  `field` varchar(50) DEFAULT NULL COMMENT 'A személy munkaterületét írja le',

  `GameId` int(10) UNSIGNED DEFAULT NULL,

  `CreatorId` int(10) UNSIGNED DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='Összeköti a készítőket az általuk részt vett játékokkal és hogy milyen téren vett részt';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A field azt mutatja, hogy a készítő milyen területen dolgozott az adott játékon, karakterlánc, maximum 50 karakter, lehet üres az olyan esetek miatt, ahol nem tisztázott a személy szerepe. A GameId a játék azonosítója a ’games’ táblázatból, egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A CreatorId a készítő azonosítója a ’creator’ táblázatból, egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, kötelező adat.

ALTER TABLE `creations`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `Creations\_CreatorId\_GameId\_unique` (`GameId`,`CreatorId`),

  ADD KEY `CreatorId` (`CreatorId`);

Megadjuk, hogy az azonosító legyen az elsődleges kulcs. Egy adott játék és fejlesztő kapcsolat egyedi, így nem ismétlődhet. Végül kulcsokká alakítjuk a GameId és CreatorId oszlopokat.

ALTER TABLE `creations`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

ALTER TABLE `creations`

  ADD CONSTRAINT `creations\_ibfk\_1` FOREIGN KEY (`GameId`) REFERENCES `games` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT `creations\_ibfk\_2` FOREIGN KEY (`CreatorId`) REFERENCES `creators` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

Az id-nek megadjuk, hogy az értékét automatikusan növelje. Megkötéseket hozunk létre, hogy a GameId oszlop a games tábla id oszlopának az idegen kulcsa legyen. Hasonlóan a CreatorId oszlop pedig a creators tábla id oszlop idegen kulcsa legyen. Megadjuk, hogy ha bármely két szülő rekord elsődleges kulcsát megváltoztatják, a változtatást ez a tábla kövesse le, és változtassa meg az idegen kulcs értékét az új értékre. Ha egy szülő rekord törlésre kerül, akkor az összekötő rekord is kerüljön törlésre.

## ’creators’ táblázat:

A játék fejlesztőit tartalmazza, és hogy milyen területen dolgoznak, bele értve a producereket, rendezőket és zeneszerzőket.

CREATE TABLE `creators` (

  `id` int(10) UNSIGNED NOT NULL,

  `firstName` varchar(100) DEFAULT NULL COMMENT 'A készítő keresztneve',

  `lastName` varchar(100) DEFAULT NULL COMMENT 'A készítő vezetékneve'

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A játékokban részvevő készítőket tárolja';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A firstName a személy keresztneve, ami karakterlánc, maximum 100 karakter, lehet üres. A lastName a személy vezetékneve, ami karakterlánc, maximum 100 karakter, lehet üres.

ALTER TABLE `creators`

  ADD PRIMARY KEY (`id`);

ALTER TABLE `creators`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

Az id oszlopot elsődleges kulcsra alakítjuk, majd megadjuk, hogy az értékét automatikusan növelje.

## ’favourites’ táblázat:

A felhasználók által kedvencekbe elmentett játékok.

CREATE TABLE `favourites` (

  `id` int(10) UNSIGNED NOT NULL,

  `GameId` int(10) UNSIGNED DEFAULT NULL,

  `UserId` int(10) UNSIGNED DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A felhasználók által a kedvencekbe mentett játékaikat tárolja';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A UserId a felhasználó azonosítója a users tábla id oszlopából, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A GameId a játék azonosítója a games tábla id oszlopából, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat.

ALTER TABLE `favourites`

  ADD PRIMARY KEY (`id`),

  ADD KEY `GameId` (`GameId`),

  ADD KEY `UserId` (`UserId`);

Az id-nek megadjuk, hogy elsődleges kulcs. A UserId és GameId oszlopokat kulcsokká alakítjuk.

ALTER TABLE `favourites`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

ALTER TABLE `favourites`

  ADD CONSTRAINT `favourites\_ibfk\_1` FOREIGN KEY (`GameId`) REFERENCES `games` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT `favourites\_ibfk\_2` FOREIGN KEY (`UserId`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

Az id-nek megadjuk, hogy az értékét automatikusan növelje. Megkötésekben megadjuk, hogy a UserId idegen kulcs legyen a users tábla id oszlopának, illetve a GameId is idegen kulcs legyen a games tábla id oszlopának. Megadjuk, hogy ha bármely két szülő rekord elsődleges kulcsát megváltoztatják, a változtatást ez a tábla kövesse le, és változtassa meg az idegen kulcs értékét az új értékre. Ha egy szülő rekord törlésre kerül, akkor az összekötő rekord kerüljön törlésre.

## ’gamepictures’ táblázat:

Ez a tábla a játékokhoz tartozó képeknek az elérési útját tárolják játék betekintésére szánt képeket.

CREATE TABLE `gamepictures` (

  `id` int(10) UNSIGNED NOT NULL,

  `url` varchar(255) NOT NULL COMMENT 'A képernyőkép elérési útvonala',

  `GameId` int(10) UNSIGNED DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A játékok képernyőképeinek az elérési útvonalait tárolja';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. Az url a kép elérési útvonala, ami karakterlánc, maximum 255 karakter, nem lehet üres, mert kötelező adat. A GameId a játék azonosítója a ’games’ táblából, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat.

ALTER TABLE `gamepictures`

  ADD PRIMARY KEY (`id`),

  ADD KEY `GameId` (`GameId`);

Az id-nek megadjuk, hogy elsődleges kulcs. A GameId oszlopot kulccsá alakítjuk.

ALTER TABLE `gamepictures`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

ALTER TABLE `gamepictures`

  ADD CONSTRAINT `gamepictures\_ibfk\_1` FOREIGN KEY (`GameId`) REFERENCES `games` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

Az id-nek megadjuk, hogy az értékét automatikusan növelje. Megkötésekben megadjuk, hogy a GameId idegen kulcs legyen a games tábla id oszlopának. Megadjuk, hogy ha a szülő rekord rekord törlésre kerül, akkor az összekötő rekord is kerüljön törlésre, ha a szülő rekord elsődleges azonosítója megváltozik, az összekötő rekord kövesse le a változást.

## ’games’ táblázat:

Ez a fő táblázat, itt tárolódnak a játék egyedi adatai.

CREATE TABLE `games` (

  `id` int(10) UNSIGNED NOT NULL,

  `gameTitle` varchar(350) NOT NULL COMMENT 'A játék címe',

  `altGameTitle` varchar(350) DEFAULT NULL COMMENT 'A játék esetleges alternatív címe',

  `description` varchar(255) NOT NULL COMMENT 'Rövid leírás a játékról',

  `release` date NOT NULL COMMENT 'A játék legelső megjelenési dátuma',

  `boxart` varchar(255) DEFAULT NULL COMMENT 'A játék borító képének az elérési útvonala',

  `promoArt` varchar(255) DEFAULT NULL COMMENT 'Egy olyan széles felbontású kép a játékról',

  `controllerSupport` int(10) UNSIGNED DEFAULT NULL COMMENT 'A játék kontroller támogatásának száma, a NULL azt jelenti hogy nem releváns, a 0 hogy nincs támogatás, az 1 hogy csak xbox típusú kontrollert támogat, a 2 hogy az xbox mellet a playstation részleges támogatást kap, a 3 hogy xbox és playstation kontrollert is teljesen támogat',

  `crossplatform` int(10) UNSIGNED DEFAULT NULL COMMENT 'A crossplay támogatás száma, a NULL azt jelenti hogy nem releváns, a 0 hogy nincs, az 1 hogy részleges támogatás, a 2 hogy teljes támogatás',

  `crossPlatformException` varchar(255) DEFAULT NULL COMMENT 'Azt magyarázza hogy a részleges crossplay támogatást mit jelent az adott játék esetén'

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A játékok egyedi adatait tárolja';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A gameTitle a játék neve, ami karakterlánc, maximum 350 karakter, nem lehet üres, mert kötelező adat. Az altGameTitle a játék alternatív neve, ami jöhet onnan, hogy más régióban más nevet vagy számozást kapott, illetve egy kisegítő, ha az eredeti neve megegyezik egy másik játék nevével, karakterlánc típusú, maximum 350 karakter, lehet üres. A description a játék rövid leírása, ami karakterlánc, maximum 255 karakter, nem lehet üres, mert kötelező adat. A release a játék legelső megjelenési ideje, ami dátum típusú, nem lehet üres, mert kötelező adat. A boxart a játék borítóképének az elérési útvonala, ami karakterlánc, maximum 255 karakter, lehet üres. A promoart a játékhoz tartozó nagyobb méretű promóciós képe, ami karakterlánc, maximum 255 karakter, lehet üres. A controllerSupport azt mutatja, hogy támogatja egy kontroller használatot számítógépen, előjel nélküli, mert nem lehet negatív, egész szám, ami ezeket az eseteket jelöli: a NULL, hogy nem releváns, mert pl PC-re nem jelent meg vagy hiányzik, 0, vagyis nincs natív kontroller támogatás. 1, vagyis csak Xbox kontroller támogatása van. 2, vagyis Playstation kontroller esetén nincs teljes támogatás, ami során a kontroller működik, de nem a megfelelő gombkiosztás jelenik meg. 3, vagyis Xbox és Playstation kontrollert egyaránt támogat. A crossplatform azt mutatja meg hogy támogatja e, hogy a játékokat különböző platformokon egymással tudjanak-e játszani, előjel nélküli, mert nem lehet negatív, egész szám, ami ezeket az eseteket tárolja: NULL, nem releváns, mert a játéknak nincs több játékos módja, 0, vagyis nincs crossplatfrom. 1, részleges támogatás van. 2, vagyis konzolok és számítógépek között egyaránt. A crossPlatformException az elöbbi esetet bővití ki, ha szükséges, a félreértéseket elkerülve, karakterlánc típusú, maximum 255 karakter, lehet üres.

ALTER TABLE `games`

  ADD PRIMARY KEY (`id`);

ALTER TABLE `games`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

A games táblának az id oszlopát elsődleges kulcsnak adjuk meg. Majd megadjuk, hogy értékét automatikusan növelje.

## ’gamesawards’ táblázat:

Összekötő tábla a games és awards tábla között, ami a díjakat a játékokkal összeköti, hozzáadva hogy melyik évben jelölték a díjra és hogy megnyerte-e.

CREATE TABLE `gamesawards` (

  `id` int(10) UNSIGNED NOT NULL,

  `year` int(10) UNSIGNED NOT NULL COMMENT 'Az év amikor a játékot a díjra jelölték',

  `result` tinyint(1) NOT NULL DEFAULT 0 COMMENT 'Az eredmény, ahol a 0 azt jelenti hogy a játékot csak jelölték a díjra, az 1 hogy megnyerte a díjat',

  `GameId` int(10) UNSIGNED DEFAULT NULL,

  `AwardId` int(10) UNSIGNED DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A játékokat köti össze azokkal a díjakkal, amelyekre jeölték és az adott jelölésre specifikus adatokkal kiegészítve';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A year a díjazás éve, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A result a díjazás eredménye, ami logikai, ha igaz, akkor a játék megnyerte a díjat, ha hamis, akkor csak jelölve volt, alapértelmezetten hamis. A GameId a játék azonosítója a ’games’ táblából, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. Az AwardId a díj azonosítója az ’awards’ táblából, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat.

ALTER TABLE `gamesawards`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `Gamesawards\_AwardId\_GameId\_unique` (`GameId`,`AwardId`),

  ADD KEY `AwardId` (`AwardId`);

A games táblának az id oszlopát elsődleges kulcsnak adjuk meg. Megadjuk, hogy egy játék és díj összekötés egyedi legyen, így nem ismételhető. Az AwardId oszlopot kulcsá alakítjuk.

ALTER TABLE `gamesawards`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

ALTER TABLE `gamesawards`

  ADD CONSTRAINT `gamesawards\_ibfk\_1` FOREIGN KEY (`GameId`) REFERENCES `games` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT `gamesawards\_ibfk\_2` FOREIGN KEY (`AwardId`) REFERENCES `awards` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

Az id-nek megadjuk, hogy az értékét automatikusan növelje. Megkötésekben megadjuk, hogy a GameId idegen kulcs legyen a games tábla id oszlopának, illetve az AwardId legyen az award tábla id oszlopának az idegen kulcsa. Megadjuk, hogy ha bármely két szülő rekord elsődleges kulcsát megváltoztatják, a változtatást ez a tábla kövesse le, és változtassa meg az idegen kulcs értékét az új értékre. Ha egy szülő rekord törlésre kerül, akkor az összekötő rekord is kerüljön törlésre.

## ’gameslanguages’ táblázat:

Összekötő tábla a ’games’ és ’languages’ között, hozzá adva hogy a játékhoz tartozik e szinkron az adott nyelven.

CREATE TABLE `gameslanguages` (

  `id` int(10) UNSIGNED NOT NULL,

  `dub` tinyint(1) NOT NULL DEFAULT 0 COMMENT 'Az adott nyelvhez tartozik-e szinkron, a 0 nemet jelent, az 1 az igent',

  `GameId` int(10) UNSIGNED DEFAULT NULL,

  `LanguageId` int(10) UNSIGNED DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A játékokat köti össze a játék által támogatott nyelvekkel kiegészítve, hogy tartozik-e hozzá szinkron';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A dub azt mutatja, hogy tartozik-e a játékhoz szinkron is, ami logikai, ha igaz, akkor a játék az adott nyelven szinkronizálva is van, ha hamis, akkor csak felirat érhető el, nem lehet üres, kötelező adat, alapvetően hamis. A GameId a játék azonosítója a ’games’ táblázatból, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A LanguageId a nyelv azonosítója a ’languages’ táblázatból, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat.

ALTER TABLE `gameslanguages`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `Gameslanguages\_LanguageId\_GameId\_unique` (`GameId`,`LanguageId`),

  ADD KEY `LanguageId` (`LanguageId`);

Az id-nek megadjuk, hogy elsődleges kulcs. Megadjuk, hogy egy játék és nyelv összekötés egyedi eset legyen, így nem ismétlődhet. A LanguageId oszlopot kulcsá alakítjuk.

ALTER TABLE `gameslanguages`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

ALTER TABLE `gameslanguages`

  ADD CONSTRAINT `gameslanguages\_ibfk\_1` FOREIGN KEY (`GameId`) REFERENCES `games` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT `gameslanguages\_ibfk\_2` FOREIGN KEY (`LanguageId`) REFERENCES `languages` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

Az id-nek megadjuk, hogy az értékét automatikusan növelje. Megkötésekben megadjuk, hogy a GameId idegen kulcs legyen a games tábla id oszlopának, illetve az LanguageId legyen az languages tábla id oszlopának az idegen kulcsa. Megadjuk, hogy ha bármely két szülő rekord elsődleges kulcsát megváltoztatják, a változtatást ez a tábla kövesse le, és változtassa meg az idegen kulcs értékét az új értékre. Ha egy szülő rekord törlésre kerül, akkor az összekötő rekord is kerüljön törlésre.

## ’gamesplatforms’ táblázat:

Összekötő tábla a ’games’ és az ’platforms’ között, további információt nem tartalmaz.

CREATE TABLE `gamesplatforms` (

  `id` int(10) UNSIGNED NOT NULL,

  `GameId` int(10) UNSIGNED DEFAULT NULL,

  `PlatformId` int(10) UNSIGNED DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A játékokat köti össze azokkal a platformokkal amelyeken megjelentek';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A GameId a játék azonosítója a ’games’ táblázatból, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A PlatformId a platform azonosítója a ’platforms’ táblázatból, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat.

ALTER TABLE `gamesplatforms`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `Gamesplatforms\_PlatformId\_GameId\_unique` (`GameId`,`PlatformId`),

  ADD KEY `PlatformId` (`PlatformId`);

Az id-nek megadjuk, hogy elsődleges kulcs. Megadjuk, hogy a játék és platform összeköttetés egyedi legyen, így nem ismétlődhet. A PlatformId oszlopot kulccsá alakítjuk.

ALTER TABLE `gamesplatforms`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

ALTER TABLE `gamesplatforms`

  ADD CONSTRAINT `gamesplatforms\_ibfk\_1` FOREIGN KEY (`GameId`) REFERENCES `games` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT `gamesplatforms\_ibfk\_2` FOREIGN KEY (`PlatformId`) REFERENCES `platforms` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

Az id-nek megadjuk, hogy az értékét automatikusan növelje. Megkötésekben megadjuk, hogy a GameId idegen kulcs legyen a games tábla id oszlopának, illetve az PlatformId legyen az platforms tábla id oszlopának az idegen kulcsa. Megadjuk, hogy ha bármely két szülő rekord elsődleges kulcsát megváltoztatják, a változtatást ez a tábla kövesse le. Ha egy szülő rekord törlésre kerül, akkor az összekötő rekord is kerüljön törlésre.

## ’gametags’ táblázat:

Összekötő tábla a ’games’ és az ’tags’ között, további információt nem tartalmaz

CREATE TABLE `gamestags` (

  `id` int(10) UNSIGNED NOT NULL,

  `GameId` int(10) UNSIGNED DEFAULT NULL,

  `TagId` int(10) UNSIGNED DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A játékokat köti azokkal a címkékkel amelyek a játékra illenek';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A GameId a játék azonosítója a ’games’ táblázatból, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A TagId a tag azonosítója a ’tags’ táblázatból, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat.

ALTER TABLE `gamestags`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `Gamestags\_TagId\_GameId\_unique` (`GameId`,`TagId`),

  ADD KEY `TagId` (`TagId`);

Az id-nek megadjuk, hogy elsődleges kulcs. Megadjuk, hogy egy játék és címke kapcsolat egyedi legyen, így nem ismétlődhet. A TagId oszlopot kulccsá alakítjuk.

ALTER TABLE `gamestags`

  ADD CONSTRAINT `gamestags\_ibfk\_1` FOREIGN KEY (`GameId`) REFERENCES `games` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT `gamestags\_ibfk\_2` FOREIGN KEY (`TagId`) REFERENCES `tags` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

Az id-nek megadjuk, hogy az értékét automatikusan növelje. Megkötésekben megadjuk, hogy a GameId idegen kulcs legyen a games tábla id oszlopának, illetve az TagId legyen az tags tábla id oszlopának az idegen kulcsa. Megadjuk, hogy ha bármely két szülő rekord elsődleges kulcsát megváltoztatják, a változtatást ez a tábla kövesse le, és változtassa meg az idegen kulcs értékét az új értékre. Ha egy szülő rekord törlésre kerül, akkor az összekötő rekord is kerüljön törlésre.

## ’languages’ táblázat:

Ez a tábla a különböző nyelveket tartalmaz, amelyek játékok esetén előfordulnak.

CREATE TABLE `languages` (

  `id` int(10) UNSIGNED NOT NULL,

  `language` varchar(25) NOT NULL COMMENT 'A nyelv neve'

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A játékokban előforduló nyelveket tárolja';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A language a nyelv neve, ami karakterlánc, maximum 25 karakter, nem lehet üres, mert kötelező adat.

ALTER TABLE `languages`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `language` (`language`);

ALTER TABLE `languages`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=20;

Az id-nek megadjuk, hogy elsődleges kulcs és hogy a language egyedi adat legyen, szóval két megegyező nevű rekord ne legyen lehetséges. Végül megadjuk, hogy az id az értékét automatikusan növelje.

## ’pcspecs’ táblázat:

A játék minimum és ajánlott gépigényét tárolja. Az eszközök általános rövidítéseit használja, és ami a minimumhoz tartozik, ott szerepel a „min” szó.

CREATE TABLE `pcspecs` (

  `id` int(10) UNSIGNED NOT NULL,

  `minOp` varchar(50) NOT NULL COMMENT 'A minimum rendszer követelményben meghatározott operációs rendszer',

  `minCpu` varchar(100) NOT NULL COMMENT 'A minimum rendszer követelményben meghatározott processzor',

  `minRam` varchar(10) NOT NULL COMMENT 'A minimum rendszer követelményben meghatározott memória',

  `minGpu` varchar(100) NOT NULL COMMENT 'A minimum rendszer követelményben meghatározott videókártya',

  `minDirectx` varchar(20) DEFAULT NULL COMMENT 'A minimum rendszer követelményben meghatározott DirectX verziója',

  `op` varchar(50) DEFAULT NULL COMMENT 'Az ajánlott rendszer követelményben meghatározott operációs rendszer',

  `cpu` varchar(100) DEFAULT NULL COMMENT 'Az ajánlott rendszer követelményben meghatározott processzor',

  `ram` varchar(10) DEFAULT NULL COMMENT 'Az ajánlott rendszer követelményben meghatározott memória',

  `gpu` varchar(100) DEFAULT NULL COMMENT 'Az ajánlott rendszer követelményben meghatározott videókártya',

  `directx` varchar(20) DEFAULT NULL COMMENT 'Az ajánlott rendszer követelményben meghatározott DirectX verzió',

  `storage` varchar(10) DEFAULT NULL COMMENT 'Az rendszer követelményben meghatározott háttártáron szükséges szabad hely',

  `sidenote` varchar(255) DEFAULT NULL COMMENT 'Egyéb rendszerkövetelmény',

  `GameId` int(10) UNSIGNED DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A játékok rendszer követelményeit tárolja';

Az id, az azonosító, egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, kötelező adat. A minop az a minimum szükséglet operációsrendszer terén, karakterlánc, maximum 50 karakter, nem lehet üres, kötelező adat. A mincpu a minimum szükséglet a processzor terén, karakterlánc, maximum 100 karakter, nem lehet üres, kötelező adat. A minram a minimum szükséglet a memória terén, karakterlánc, maximum 10 karakter, nem lehet üres, kötelező adat. A mingpu a minimum szükséglet a videókártya terén, karakterlánc, maximum 100 karakter, nem lehet üres, kötelező adat. A mindirectx a minimum szükséglet a directx verzió terén, karakterlánc, maximum 20 karakter, nem lehet üres, kötelező adat. Az op az ajánlott szükséglet operációsrendszer terén, karakterlánc, maximum 50 karakter. A cpu az ajánlott szükséglet a processzor terén, karakterlánc, maximum 100 karakter. A ram az ajánlott szükséglet a memória terén, karakterlánc, maximum 10 karakter. A gpu az ajánlott szükséglet a videókártya terén, karakterlánc, maximum 100 karakter. A directx az ajánlott szükséglet a directx verzió terén, karakterlánc, maximum 20 karakter. A storage a szükséges szabad tárhely, karakterlánc, maximum 10 karakter, nem lehet üres, kötelező adat. A sidenote az egyéb megjegyzéseket tartalmazza, karakterlánc, maximum 255 karakter. A GameId a játék azonosítója a ’games’ táblából, egész szám, előjel nélküli, mert nem lehet negatív.

ALTER TABLE `pcspecs`

  ADD PRIMARY KEY (`id`),

  ADD KEY `GameId` (`GameId`);

Az id-nek megadjuk, hogy elsődleges kulcs. A GameId oszlopot kulcsra alakítjuk.

ALTER TABLE `pcspecs`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=17;

ALTER TABLE `pcspecs`

  ADD CONSTRAINT `pcspecs\_ibfk\_1` FOREIGN KEY (`GameId`) REFERENCES `games` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

Az id-nek megadjuk, hogy az értékét automatikusan növelje. Megkötésekben megadjuk, hogy a GameId idegen kulcs legyen a games tábla id oszlopának és hogy ha a szülő rekord törlésre kerül akkor gyermek rekord is törlődjön, ha megváltozik az elsődleges kulcs, akkor kövesse a változtatást.

## ’platforms’ táblázat:

Ez a táblázat tartalmazza játék platformokat/konzolokat tartalmaz.

CREATE TABLE `platforms` (

  `id` int(10) UNSIGNED NOT NULL,

  `platform` varchar(40) NOT NULL COMMENT 'A platform neve'

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='Különböző platformokat tárol, amelyek hivatalosan támogatják a játékok futtatását';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A GameId a játék azonosító a ’games’ táblából, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A platform mező a platform, avagy konzol neve, ami karakterlánc, maximum 40 karakter, nem lehet üres, mert kötelező adat.

ALTER TABLE `platforms`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `platform` (`platform`);

ALTER TABLE `platforms`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

Az id-nek megadjuk, hogy elsődleges kulcs és hogy a platform egyedi adat legyen, szóval két megegyező nevű rekord ne legyen lehetséges. Végül megadjuk, hogy az id az értékét automatikusan növelje.

## ’rating’ táblázat:

A játékok értékeléseit tartalmazza, amit a regisztrált felhasználók adtak.

CREATE TABLE `ratings` (

  `id` int(10) UNSIGNED NOT NULL,

  `positive` tinyint(1) NOT NULL DEFAULT 0 COMMENT 'Az értékelés, ha 0 az negatív értékelést jelent míg az 1 pozitívat',

  `GameId` int(10) UNSIGNED DEFAULT NULL,

  `UserId` int(10) UNSIGNED DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A játékok felhasználói értékeléseit tárolja';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A positive a felhasználó értékelése, ami logikai, ahol a hamis azt jelzi, hogy a felhasználó negatívan értékelte, az igaz meg hogy pozitívan értékelte, nem lehet üres, mert kötelező adat. A GameId a játék azonosítója a ’games’ táblázatból, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A UserId a felhasználó azonosítója a ’users’ táblázatból, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat.

ALTER TABLE `ratings`

  ADD PRIMARY KEY (`id`),

  ADD KEY `GameId` (`GameId`),

  ADD KEY `UserId` (`UserId`);

Az id-nek megadjuk, hogy elsődleges kulcs. A GameId és UserId oszlopokat kulcsokká alakítjuk.

ALTER TABLE `ratings`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

ALTER TABLE `ratings`

  ADD CONSTRAINT `ratings\_ibfk\_1` FOREIGN KEY (`GameId`) REFERENCES `games` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT `ratings\_ibfk\_2` FOREIGN KEY (`UserId`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

Az id-nek megadjuk, hogy az értékét automatikusan növelje. Megkötésekben megadjuk, hogy a GameId idegen kulcs legyen a games tábla id oszlopának, illetve az UserId legyen az users tábla id oszlopának az idegen kulcsa. Mindkét esetben megadjuk, hogy ha a szülő rekord törlésre kerül, akkor az összekötő táblában a rekord törlődjön, ha a szülő rekord elsődleges kulccsa megváltozik, akkor kövesse le a változtatást.

## ’reviews’ táblázat:

A felhasználók által írt kritikákat tárolja egy specifikus játék esetén.

CREATE TABLE `reviews` (

  `id` int(10) UNSIGNED NOT NULL,

  `title` varchar(100) NOT NULL COMMENT 'A kritika címe',

  `content` text NOT NULL COMMENT 'A kritika törzs szövege',

  `date` date NOT NULL COMMENT 'A kritika létrehozásának a dátuma',

  `UserId` int(10) UNSIGNED DEFAULT NULL,

  `GameId` int(10) UNSIGNED DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A felhasználók által írt játék kritikákat tárolja';

Az id a kritika azonosítója, egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, kötelező adat. A title a kritika címe, karakterlánc, maximum 100 karakter, nem lehet üres, kötelező adat. A content a kritika törzs szövege, szöveg típusú, nem lehet üres, mert kötelező adat. A date a kritika létrehozásának dátuma, nem lehet üres, kötelező adat. A UserId a felhasználó azonosítója a users táblázatból, egész szám, előjel nélküli, mert nem lehet negatív. A GameId a játék azonosítója a games táblázatból, egész szám, előjel nélküli, mert nem lehet negatív szám.

ALTER TABLE `reviews`

  ADD PRIMARY KEY (`id`),

  ADD KEY `UserId` (`UserId`),

  ADD KEY `GameId` (`GameId`);

Az azonosítót elsődleges kulccsá alakítjuk. A UserId és GameId oszlopokat kulcsokká alakítjuk.

ALTER TABLE `reviews`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

ALTER TABLE `reviews`

  ADD CONSTRAINT `reviews\_ibfk\_1` FOREIGN KEY (`UserId`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT `reviews\_ibfk\_2` FOREIGN KEY (`GameId`) REFERENCES `games` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

Az azonosítónak megadjuk, hogy az értékét automatikusan növelje. Megadjuk, hogy a UserId idegen kulcsként funkcionáljon a users tábla id oszlopának, a GameId pedig a games tábla id oszlopának. Mindkét idegen kulcs esetén, ha a szülő rekord törlésre kerül, akkor az összekötő tábla rekordja törlődjön, és ha a szülő rekordok elsődleges kulcsát megváltoztatják, az összekötő tábla kövesse le változást.

## ’studios’ táblázat:

A játék fejlesztő és kiadó stúdiókat tárolja és ezeknek a logóját.

CREATE TABLE `studios` (

  `id` int(10) UNSIGNED NOT NULL,

  `name` varchar(50) NOT NULL COMMENT 'A stúdió neve'

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='Különböző játék stúdiókat tárol';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A name a cég neve, ami karakterlánc, maximum 50 karakter, nem lehet üres, mert kötelező adat.

ALTER TABLE `studios`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `name` (`name`);

ALTER TABLE `studios`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=30;

Az id-nek megadjuk, hogy elsődleges kulcs és hogy a name egyedi adat legyen, szóval két megegyező nevű rekord ne legyen lehetséges. Végül megadjuk, hogy az id az értékét automatikusan növelje.

## ’studiosgames’ táblázat:

Összeköti a ’games’ és a ’studios’ táblázatot azonosító alapján, kiegészítve ezt azzal, hogy az adott játék esetén fejlesztő és/vagy kiadói szerepet töltöttek be.

CREATE TABLE `studiosgames` (

  `id` int(10) UNSIGNED NOT NULL,

  `isDeveloper` tinyint(1) NOT NULL DEFAULT 0 COMMENT 'A stúdió részt vett-e a játék fejlesztésében, a 0 nemet, az 1 igent jelent',

  `isPublisher` tinyint(1) NOT NULL DEFAULT 0 COMMENT 'A stúdió részt vett-e a játék kiadásában, a 0 nemet, az 1 igent jelent',

  `GameId` int(10) UNSIGNED DEFAULT NULL,

  `StudioId` int(10) UNSIGNED DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='Összeköti a játékokat a hozzájuk tartozó stúdiókkal kiegészítve a stúdió szerepével';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A GameId a játék azonosítója, a ’games’ táblázatból kerül ide az ’id’ oszlop, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A StudioId a stúdió azonosítója a ’studios’ táblázatból, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. Az isDeveloper azt mutatja, hogy a stúdió fejlesztette-e a játékot, ami logikai, nem lehet üres, mert kötelező adat, alapvetően hamis. Az isPublisher azt mutatja, hogy a stúdió kiadta-e a játékot, ami logikai, nem lehet üres, mert kötelező adat.

ALTER TABLE `studiosgames`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `Studiosgames\_StudioId\_GameId\_unique` (`GameId`,`StudioId`),

  ADD KEY `StudioId` (`StudioId`);

Az id-nek megadjuk, hogy elsődleges kulcs. A StudioId oszlopot kulccsá alakítjuk.

ALTER TABLE `studiosgames`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

ALTER TABLE `studiosgames`

  ADD CONSTRAINT `studiosgames\_ibfk\_1` FOREIGN KEY (`GameId`) REFERENCES `games` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

  ADD CONSTRAINT `studiosgames\_ibfk\_2` FOREIGN KEY (`StudioId`) REFERENCES `studios` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

Az id-nek megadjuk, hogy az értékét automatikusan növelje. Megkötésekben megadjuk, hogy a GameId idegen kulcs legyen a games tábla id oszlopának, illetve az StudioId legyen az users tábla id oszlopának az idegen kulcsa. Megadjuk, hogy ha bármely két szülő rekord elsődleges kulcsát megváltoztatják, a változtatást ez a tábla kövesse le, és változtassa meg az idegen kulcs értékét az új értékre. Ha egy szülő rekord törlésre kerül, akkor az összekötő rekord is kerüljön törlésre.

## ’tags’ táblázat:

A játékokhoz tartozó címkék, avagy ’tag’-ek, ami a játékhoz illenek.

CREATE TABLE `tags` (

  `id` int(10) UNSIGNED NOT NULL,

  `tag` varchar(40) NOT NULL COMMENT 'A címke neve'

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='Különböző címkéket tárol amivel a játékokat lehet besorolni';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A tag a játék címkéje, ami stílust, illetve műfajt jelöl, karakterlánc típusú, maximum 40 karakter, nem lehet üres, mert kötelező adat.

ALTER TABLE `tags`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `tag` (`tag`);

ALTER TABLE `tags`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=0;

Az id-nek megadjuk, hogy elsődleges kulcs és hogy a tag egyedi adat legyen, szóval két megegyező nevű rekord ne legyen lehetséges. Végül megadjuk, hogy az id az értékét automatikusan növelje.

## ’users’ táblázat:

Információk a felhasználókról.

CREATE TABLE `users` (

  `id` int(10) UNSIGNED NOT NULL,

  `username` varchar(30) NOT NULL COMMENT 'A felhasználó által megadott becenév',

  `password` varchar(255) NOT NULL COMMENT 'A felhasználó jelszava titkosítva',

  `email` varchar(255) NOT NULL COMMENT 'A felhasználó email címe',

  `admin` tinyint(1) NOT NULL DEFAULT 0 COMMENT 'A felhasználó admin státusza, ahol a 0 azt jelenti hogy csak általános felhasználó, míg az 1 azt jelenti hogy admin',

  `creation` date NOT NULL COMMENT 'A felhasználó fiókjának létrehozásának a dátuma'

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci COMMENT='A felhasználók adatait tárolja';

Az id az azonosító, ami egész szám, előjel nélküli, mert nem lehet negatív, nem lehet üres, mert kötelező adat. A username a felhasználó neve, ami karakterlánc, maximum 30 karakter, nem lehet üres, mert kötelező adat. A password a felhasználó jelszava titkosítva, ami karakterlánc, maximum 255 karakter, nem lehet üres, mert kötelező adat. Az email mező a felhasználó email címe, ami karakterlánc, maximum 255 karakter, nem lehet üres, mert kötelező adat. Az admin azt mutatja, hogy a felhasználó rendelkezik-e admin jogokkal, ami logikai, nem lehet üres, mert kötelező adat. A creation a profil létrehozásának időpontja, ami dátum típusú, nem lehet üres, mert kötelező adat.

ALTER TABLE `users`

  ADD PRIMARY KEY (`id`),

  ADD UNIQUE KEY `username` (`username`),

  ADD UNIQUE KEY `email` (`email`);

Az id-nek megadjuk, hogy elsődleges kulcs. Az email és username oszlopok egyedinek állítjuk be, hogy ne engedjen két ugyanolyan rekordot a táblába.

ALTER TABLE `users`

  MODIFY `id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=;

Megadjuk, hogy az id az értékét automatikusan növelje.

# Backend dokumentáció

## HTTP kérések

|  |  |  |  |
| --- | --- | --- | --- |
| URL | Metódus | Magyarázat | Státusz kódok |
| / | GET | A főoldalon megjelennek a hamarosan megjelenő játékok, a nemrég megjelent játékok, továbbá 15 db játékot a következő kategóriákban: lövöldöző, kaland, szerepjáték, versenyzős és stratégia. | 200 – Adatok sikeresen lekérdezve  500 – Ismeretlen hiba |
| PUT | A főoldalon egy működő kereső mező, ami a játékokat tud keresni az eredeti, illetve alternatív nevével is, a keresendő szöveget a frontend elküldi, amit a backend feldolgozz. | 200 – Keresés eredményei lekérve  500 – Ismeretlen hiba |
| /game/:gameId | GET | Az URL-ben megadott paraméterben szereplő szám alapján az azonos számú azonosítójú játék minden hozzá tartozó adatait kéri le. | 200 – Adatok sikeresen lekérdezve  404 – Az URL-ben megadott azonosítóhoz nincs játék  500 – Ismeretlen hiba |
| POST | Egy bejelentkezett felhasználó játék értékelését menti el. A felhasználót a frontend által elküldött token alapján határozzuk meg. A játék azonosítóját az URL paraméteréből nyerjük ki. Az értékelést a frontend küldi el egy body-ban, amiben egy „isPositive” logikai változó van, ahol a false negatív, a true pedig pozitív értékelést jelent, végül elmentjük az adatbázisban. | 201 – Az értékelés sikeresen lementve 400 - Hiányzik a játék azonosítója  401 – Nem engedélyezett a játék értékelés, mert a felhasználó nincs bejelentkezve  404 – Az URL-ben megadott azonosítóhoz nincs játék  409 – A felhasználó már értékelte a játékot  500 – Ismeretlen hiba |
| PUT | Egy bejelentkezett felhasználó kedvencekbe helyezheti az adott játékot. A felhasználót a frontend által elküldött token alapján határozzuk meg. A játék azonosítóját az URL paraméteréből nyerjük ki. Végül elmentjük a kapcsolatot az adatbázisban. | 201 – Az kedvencekbe tétel sikeresen lementve  401 – Nem engedélyezett a játék kedvencekbe tétele, mert a felhasználó nincs bejelentkezve  404 – Az URL-ben megadott azonosítóhoz nincs játék  409 – A felhasználó már a kedvencekbe helyezte a játékot  500 – Ismeretlen hiba |
| /registration | POST | A felhasználó regisztrációját kezeli, az adatokat a felhasználók adják meg amit a frontend továbbít a backend-nek, a backend ellenőrzi az adatok formátumát, hiba esetén hibát dob, ha átment az ellenőrzéseken akkor az adatbázisban elmenti a felhasználó adatait. | 201 – Az adatbázisba sikeresen feljuttattuk az új felhasználó adatokat  400 – Hiba a regisztrációs adatokkal  409 – Már van az adatbázisban olyan felhasználó, akinek ilyen felhasználóneve vagy email címe van.  500 – Ismeretlen hiba |
| /login | POST | A felhasználó bejelentkezését kezeli, az adatokat a felhasználó adja meg, amelyet a frontend közvetít a backend-nek. Ellenőrizzük, hogy van-e ilyen felhasználó az adatbázisban, ha van akkor létrehozzunk egy tokent, és a válaszban visszaküldjük. | 200 – Sikeres bejelentkezés, ehhez a felhasználó kap egy token-t  400 – Hiányzik valamilyen adat vagy üres  404 – Nem létezik ilyen adatokkal felhasználó  500 – Ismeretlen hiba |
| /myprofile | GET | Egy bejelentkezett felhasználó adatait kéri le. A felhasználót a frontend által elküldött token alapján határozzuk meg. | 200 – Adatok sikeresen lekérdezve  401 – A felhasználó nincs bejelentkezve  500 – Ismeretlen hiba |
| POST | Egy bejelentkezett felhasználó megtudja változtatni a felhasználónevét, az email címét és jelszavát. A felhasználót a frontend által elküldött token alapján határozzuk meg. Az adatokat a backend ellenőrzi, hogy a megfelelő formátumban van-e, ha igen akkor az adatbázisban megváltoztatjuk az adatokat. | 201 – Az adat(ok) sikeresen változtatása  400 – Probléma az adatok formátumával  409 – Már ilyen adat vagy adatokkal már van felhasználó  500 – Ismeretlen hiba |
| /favourites | GET | Egy bejelentkezett felhasználó a kedvencekben elmentett játékait kéri le az adatbázisból. A felhasználót a frontend által elküldött token alapján határozzuk meg. | 200 – Adatok sikeresen lekérve  401 – A felhasználó nincs bejelentkezve  500 – Internal Server Error |
| POST | Egy bejelentkezett felhasználó eltávolíthatja a kedvencekben mentett játékát. A felhasználót a frontend által elküldött token alapján határozzuk meg. A játék azonosítóját a frontend küldi el. | 200 – A kapcsolat sikeresen törölve  400 – A játék azonosítója nincs megadva  401 – A felhasználó nincs bejelentkezve  404 – A megadott játék azonosítóhoz nincs játék hozzáadva  500 – Internal Server Error |
| /game/:gameId/review | POST | Egy bejelentkezett felhasználó rövid kritikáját kezeli. Kritikát csak az a felhasználó írhat, aki értékelte a játékot. A felhasználót a frontend által elküldött token alapján határozzuk meg. A játék azonosítóját az URL paraméteréből nyerjük ki. A kritika egy címből (title) és egy törzs szövegből (content) áll, amiket a frontend küldd el és a backend ellenőrzi. Ha jó formátumban van, akkor elmentjük a kritikát az adatbázisban. | 201 – Sikeres kritika létrehozás  400 – Hiba a kapott adatokkal  401 – A felhasználó nincs bejelentkezve  404 – Nincs ilyen azonosítójú játék  409 – A felhasználó már írt kritikát  500 – Ismeretlen hiba |
| DELETE | Egy felhasználó kritikáját törli. Kritikát a saját bejelentkezett felhasználó írója tudja törölni, illetve admin jogokkal rendelkező felhasználó. A felhasználót a frontend által elküldött token alapján határozzuk meg. A játék azonosítóját az URL paraméteréből nyerjük ki. A kritika azonosítóját a frontend küldi el reviewId néven. | 200 – A kritika sikeresen törölve.  401 –A felhasználó nincs bejelentkezve, vagy nincs jogosultsága a törléshez  400 – Hiányzik a játék azonosító vagy a kritika azonosítója  404 – Nincs ilyen azonosítójú játék vagy kritika  500 – Ismeretlen hiba |

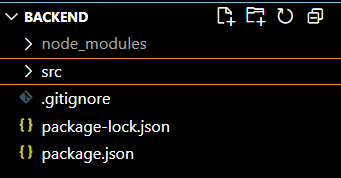
# Eszközök

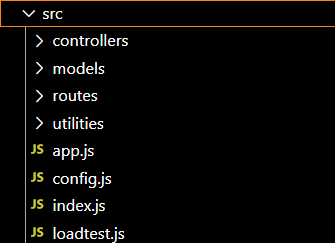
A backend Node.js segítségével lett létrehozva az Express keretrendszerrel, továbbá a Sequelize ORM-et használjuk az adatbázis eléréséhez, ehhez telepítve van mellé a Mariadb csomag is, hogy megadjuk az adatbázis dialektusát. A Cors csomag lehetővé teszi, hogy kapcsolatot alakítsunk ki a frontend-el.

Az oldal token-eket alkalmaz a felhasználók kezelésére, ehhez a Jose könyvtár JWE token-eket alkalmazunk, ami az alapvető JWT-hez képest további titkosítással van ellátva. A felhasználók jelszavának titkosításához a Bcrypt csomagot használjuk. A leo-profanity egy olyan csomag, amely eszközöket ad ahhoz, hogy angol nyelvű káromkodásokat szűrjünk ki egy megadott szövegből.

A Nodemon csomagot használjuk, hogy megkönnyítse a fejlesztési oldalt azzal, hogy a szerver minden változtatás mentése után újra induljon, illetve biztosít tiszta kilépést hiba esetén, ez a csomag fejlesztési függőséként van telepítve. Szintén fejlesztési függőségként adjuk meg a k6 csomagot, amivel a terhelés tesztet tudunk végrehajtani.

## Mappa struktúra

A backend mappában három fájl található, a package.json és a package-lock.json olyan fájlok amik egy Node.js projekt megkezdésekor létre jönnek és meta adatokat tartalmaznak, a .gitignore-t arra használjuk hogy a git commit-ok és push-ok során a fájlban megadott mappák és fájlokat ignorálja, jelen esetben itt csak a node modules mappa van megadva, mert az itt lévő viszonylag sok helyet foglalnak és a tartalmuk az npm install paranccsal gyorsan létrehozhatóak. Itt található még kettő mappa, a node modules a telepített csomagokat tartalmazza, az src mappa a forrás fájlokat tartalmazza.

Az src mappa további négy mappát tartalmaz, A controllers mappában lévő fájlok kezelik a felhasználó által kezdeményezett kéréseket, ezek metódusokra vannak bontva. A models mappa az adatbázis tábláinak a Sequelize modelljeit tartalmazza, minden tábla egy külön fájl. A routes mappa a controllers mappában lévő metódusokat társítja egy URL-hez és http metódussal, majd ezt exportálja. Az utilities olyan metódusokat tartalmaz, amelyek a projektben több helyen felvannak használva, így csökkentve a redundanciát és átláthatóbb lesz a kód, a különböző fájlok valamilyen csomagra specifikusak, például a bcrypt.methods.js a bcrypt csomaghoz tartozó műveleteket és változókat tartalmazza. A mappa tartalmaz még négy fájlt, az app.js segít létrehozni a http szervert és összesíti, hogy milyen kérésekre milyen választ adjon, de ezek pontos meghatározása nem itt történik. A config.js-ben a Sequelize csomag konstansa van megadva, ami tartalmazza az adatbázis beállítását. Az index.js a szoftver fő fájlja, avagy belépési pontja, itt határozzuk meg a modellek kapcsolatait és ha minden sikeres lefut és hozzá fér az adatbázishoz és szinkronizálta a modelleket, akkor elindítja a szervert. A loadtest.js a terhelés tesztet tartalmazza.

## Backend komponensek működése

Itt a backend-en az általunk létrehozott fájlokon megyünk át, hogy mit és hogyan csinálunk. Itt nem megyünk bele a loadtest.js tartalmába, ehhez a terhelés tesztnél térünk vissza.

### Az src mappa fájljai

**app.js működése**

// Be importáljuk a szükséges csomagokat

import express from "express";

import cors from "cors"

import jweMethods from "./utilities/jwe.methods.js";

import RegistRouter from "./routes/regist.router.js";

import MainPageRouter from "./routes/mainpage.router.js";

import LoginRouter from "./routes/login.router.js";

import GamepageRouter from "./routes/gamepage.router.js";

import MyprofileRouter from "./routes/myprofile.router.js";

import FavouritesRouter from "./routes/favourites.router.js";

import ReviewRouter from "./routes/review.router.js";

Először beimportáljuk a szükséges csomagokat, az express segítségével hozzuk létre a http szerver alkalmazását, a cors-al lehetővéve tesszük a kommunikációt a frontend-el, a jweMethods-ból egy middleware-t használunk fel, hogy új token-t hozzunk létre egy még aktív felhasználónak, az azt követő csomagok úgynevezett Router konstansokat tartalmazzák, hogy milyen URL címre milyen metódusra milyen műveletet hajtson végre az oldal.

// Létrehozzunk egy konstanst az express segítségével, amivel a szervert fogjuk kezelni

const app = express();

// A frontend elérése a 4200-as porton és engedélyezzük, hogy a frontend hozzá férjen az Authorization fejléchez

app.use(cors({

    origin: "http://localhost:4200",

    allowedHeaders: ["Authorization", "Content-type"],

    exposedHeaders: ["Authorization"]

}));

Itt létrehozzuk az app konstanst az expresss csomag segítségével. Megadjuk, hogy ez az app használja a cors csomagot, itt megadjuk továbbra, hogy milyen cím érheti el a backend-et, illetve létrehozzuk az Authorization fejlécet a http válaszokhoz, és azt is megkell adnunk, hogy a frontend elérje ezt a fejlécet. Ez a fejléc tartalmazza az új token-t egy még aktív felhasználónak.

// Engedélyezzük json típusú válaszokat

app.use(express.json());

// A router-ek elérését adjuk meg, illetve egy middleware a token hosszabbításra

app.use("/", jweMethods.ExntendingToken, MainPageRouter);

app.use("/", RegistRouter);

app.use("/", LoginRouter);

app.use("/", jweMethods.ExntendingToken, GamepageRouter);

app.use("/", jweMethods.ExntendingToken, MyprofileRouter);

app.use("/", jweMethods.ExntendingToken, FavouritesRouter);

app.use("/", jweMethods.ExntendingToken, ReviewRouter);

// Ha egy olyan metódust kapunk egy útvonalra, amit nem támogatunk, akkor ezt az üzenetet adjuk vissza

app.use((req, res) => {

    res.status(405).json({

        error: true,

        message: "Method not allowed!"

    });

    return;

});

// Exportáljuk, hogy az index.js-ben felhasználjuk

export default app;

Megadjuk, hogy az szerverünk engedélyezze a json formátumú kéréseket és válaszokat. Majd megadjuk, hogy a szerverünk használja az importált Router konstansokat, amik már tartalmazzák a szükséges infókat, így itt már csak megadjuk, hogy az oldalunk használja őket, de előtte rendelje hozzá a jweMethods-ból az ExtendingToken middleware-t, ami a megadott útvonalakon ellenőrzi és ha kell ad új útvonalat. A RegistRouter és LoginRouter-hez ezt nem adjuk hozzá, az előző a regisztrációhoz az utóbbi a bejelentkezéshez tartozik, ezekhez egy bejelentkezett felhasználó már nem fér hozzá kijelentkezés nélkül így felesleges lenne hozzá rendelni. Ez után megadjuk a szervernek, hogy a Router-ekben megadott útvonalakban, ha olyan metódust használunk, ami nincs megadva, akkor küldje el a http 405-ös kódú hiba üzenetet. Végül exportáljuk a szervert, hogy felhasználjuk az index.js-ben.

**config.js működése**

// Az adatbázist külön konstansként mentjük majd exportáljuk, mert sok különböző fájlban fel van használva

import { Sequelize } from "sequelize";

// Megadjuk az adatbázis adatait az eléréséhez

const sequelize = new Sequelize({

    username: 'root',

    password: '',

    dialect: "mariadb",

    // Így a createdAt és updatedAt táblák automatikus létrehozását kikapcsoljuk, mert nincs rá szükségünk

    define: {

        timestamps: false,

    },

    database: 'game\_knowledge',

    host: 'localhost'

});

export default sequelize;

Először importáljuk a Sequelize szükséges csomagját, majd létrehozunk egy sequelize nevű konstanst ezzel a csomaggal, ami az adatbázisunkat jelképezi a bakcend-en. Megadjuk az egy alapértelmezett felhasználó nevét és jelszavát, ami alapvetően tartalmaz a phpmyadmin, és megadjuk az ehhez tartozó adatbázis dialektusát. Utána letiltjuk, hogy a Sequelize létrehozza az alapértelmezett ’createdAt’ és ’updatedAt’ táblákat, mert nem szeretnénk ezeket tárolni minden táblánál. A ’createdAt’ oszlop azt tartalmazná, hogy mikor jött létre egy rekord, erre két esetben lenne szükségünk, de egyszerűbb letiltani itt, hogy létre jöjjön mindenhol majd manuálisan megadni abban a két esetben. Az ’updatedAt’ oszlop azt tartalmazná, hogy a rekordot mikor módosították utoljára, de ezt sehol se szeretnénk tárolni. Ezután megadjuk az adatbázis nevét, majd a host-ot. Végül exportáljuk, hogy felhasználjuk az index.js-nél és a modellek létrehozásánál.

**index.js működése**

import sequelize from "./config.js";

import app from "./app.js";

// Minden modelt be importálunk ide

import Acting from "./models/acting.js";

import Actor from "./models/actor.js"

import Agerating from "./models/agerating.js";

import Award from "./models/award.js";

import Creation from "./models/creation.js";

import Creator from "./models/creator.js"

import Favourite from "./models/favourite.js";

import Game from "./models/game.js";

import Gamepicture from "./models/gamepicture.js";

import Gamesagerating from "./models/gamesagerating.js";

import Gamesaward from "./models/gamesaward.js";

import Gameslanguage from "./models/gameslanguage.js";

import Gamesplatform from "./models/gamesplatform.js";

import Gamestag from "./models/gamestag.js";

import Language from "./models/language.js";

import Pcspec from "./models/pcspec.js";

import Platform from "./models/platform.js";

import Rating from "./models/rating.js";

import Review from "./models/review.js";

import Studio from "./models/studio.js";

import Studiosgame from "./models/studiosgame.js";

import Tag from "./models/tag.js";

import User from "./models/user.js";

Elsősorban importáljuk az előzőleg meghatározott app és a sequelize-t. Ezután importáljuk az összes tábla modellt, hogy beállítsuk a táblák kapcsolatait.

// Létrehozzuk az adatbázis kapcsolatokat

// Színész - Játék kapcsolat

Game.belongsToMany(Actor, {

    through: Acting

});

Actor.belongsToMany(Game, {

    through: Acting

});

// Korhatár - Játék kapcsolat

Game.belongsToMany(Agerating, {

    through: Gamesagerating

});

Agerating.belongsToMany(Game, {

    through: Gamesagerating

});

// Díj - Játék kapcsolat

Game.belongsToMany(Award, {

    through: Gamesaward

});

Award.belongsToMany(Game, {

    through: Gamesaward

});

// Készítők - Játékok kapcsolat

Game.belongsToMany(Creator,{

    through: Creation

});

Creator.belongsToMany(Game, {

    through: Creation

});

// Kedvencek Felhasználók - Játékok kapcsolat

Game.hasMany(Favourite);

Favourite.belongsTo(Game);

User.hasMany(Favourite);

Favourite.belongsTo(User);

// Játék - Játék képek kapcsolat

Game.hasMany(Gamepicture, {

    onDelete: 'CASCADE',

});

Gamepicture.belongsTo(Game);

// Nyelv - játék kapcsolat

Game.belongsToMany(Language, {

    through: Gameslanguage

});

Language.belongsToMany(Game, {

    through: Gameslanguage

});

// Gépigény - Játék kapcsolat

Game.hasOne(Pcspec);

Pcspec.belongsTo(Game);

// Platform - Játék kapcsolat

Game.belongsToMany(Platform, {

    through: Gamesplatform

});

Platform.belongsToMany(Game, {

    through: Gamesplatform

});

// Értékelések - Játékok kapcsolat

Game.hasMany(Rating);

Rating.belongsTo(Game);

// Értékelések - Felhasználók kapcsolat

User.hasMany(Rating);

Rating.belongsTo(User);

// Studió - Játék kapcsolat

Game.belongsToMany(Studio, {

    through: Studiosgame

});

Studio.belongsToMany(Game, {

    through: Studiosgame

});

// Címke - Játék kapcsolat

Game.belongsToMany(Tag, {

    through: Gamestag,

});

Tag.belongsToMany(Game, {

    through: Gamestag,

});

// Kritika - Felhasználó kapcsolat

User.hasMany(Review);

Review.belongsTo(User);

// Kritika - Játék kapcsolat

Game.hasMany(Review);

Review.belongsTo(Game);

Itt megadjuk, hogy milyen kapcsolatban állnak a táblák egymással. Amikor két tábla között kettő belongsToMany-t használunk akkor sok a sokhoz kapcsolatot hozunk létre, ezeknél a through paraméterben megadjuk, hogy melyik a két tábla összekötő táblája. Amikor két tábla között hasMany és belongsTo-t használunk akkor egy a sokhoz kapcsolatot hozunk létre, ahol hasMany parancs előtt lévő tábla egy rekordjához több, a parancs utáni táblának rekordja tartozik, illetve a belongsTo parancs előtt lévő táblánál megadjuk, hogy függ a parancs után megadott táblától. Amikor két táblánál hasOne és belongsTo-t használunk akkor egy az egyhez kapcsolatot hozunk létre, ahol a hasOne parancs előtti táblánál egy rekordhoz megadjuk, hogy a parancs utáni táblából csak egy rekord tartozik, majd, hogy a belongsTo parancs előtti táblától függ a parancs utáni tábla.

// Konstansként elmentjük a backend port számát

const PORT = 3000;

// Egy csatkalozás kisérlet után, ha sikeres akkor elindítjuk a szervert

await sequelize.authenticate()

    .then(() => {

        console.log("The test connection to the server was succesfull!");

        // Szinkronizáljuk a modeleket az adatbázissal

        sequelize.sync()

        .then(() => {

            console.log("The database sync was succesfull!");

            app.listen(PORT, () => {

                console.log(`The backend server is running on: https://localhost:${PORT}/`);

            });

        })

        .catch((error) => {

        console.log("The database sync failed", error);

        });

    })

    .catch((error) => {

        console.error("Can't connect to the server!\n" + error);

        sequelize.close();

    });

Először elmentjük, hogy a szerver milyen porton fusson a PORT konstansba, hisz ennek értéke nem változik a futás során. Ezután a megnézzük, hogy tudunk egy csatlakozni az adatbázisunkhoz, ha nem akkor írja ki a hibát, és álljon le a program, ha igen akkor szinkronizáljuk a programban meghatározott modelleket az adatbázisban lévő táblákkal, ha ez nem sikerül akkor írja ki a hiba okát, ha sikerül akkor induljon el a http szerverünk a meghatározott porton.

### Controller fájlok

Ezek a fájlok mind a controllers mappában találhatóak és minden fájl nevében benne van a controller szó. Mindegyik fájlra igaz export default {} kapcsos zárójeleiben írunk, hogy könnyen exportáljuk az hozzájuk tartozó Router-hoz. Az export default-ban metódusok vannak írva, ezek nevében benne van, hogy milyen metódussal érjük el. Minden metódust try catch párban írunk meg hogy ha a futás során valami hiba történik akkor azt kezeljük, és egy 500-as kódú választ adjunk.

**favourites.controller.js működése**

Ez a fájl a controllers mappában található. Ebben a fájlban a kedvencek kezelésére való metódusokat tartalmazza.

import Favourite from "../models/favourite.js";

import Game from "../models/game.js";

import jweMethods from "../utilities/jwe.methods.js";

Itt be importáljuk a fájlban használt modelleket és jweMethods-t.

Az első metódus, amit megnézünk a FavouritesGetController, ez felel egy bejelentkezett felhasználó kedvenceibe lementett játékainak az adatai lekérését.

FavouritesGetController: async(req, res) => {

        try {

            // Lekérjük a felhasználó azonosítóját, és hiba üzenetet küldünk, ha ez nem sikerül, mert nem bejelentkezett felhaszánó

            // nem fér hozzá ehhez a funkcióhoz

            const userId = await jweMethods.GetUserId(req);

            if (userId === undefined) {

                res.status(401).json({

                    error: true,

                    message: "The token is empty or faulty!"

                });

                return;

            }

Konstansban elmentjük a felhasználó azonosítóját, amit a jweMethods-ból a GetUserId metódusával kérünk le, ha nem sikerült lekérni az arra utal, hogy nincs megadva token vagy hibás, erre hiba üzenetet küldünk.

// A felhasználó azonosítója alapján megkeressük az összes kedvencekbe elmentett játékokat, majd visszaadjuk a válaszban

            const favourites = await Game.findAll({

                attributes: ["id", "gameTitle", "boxart"],

                include: {

                    attributes: [],

                    where: { UserId: userId},

                    model: Favourite

                }

            });

            res.status(200).json({

                error: false,

                message: "User's favourite games are fetched!",

                favourites: favourites

            });

            return;

        }

Ha sikerült a felhasználó azonosítóját lekérni, akkor egy konstansba elmentjük azoknak a játékoknak a megjelenítésre kívánt adatait, amik a megadott felhasználó kedvenceikbe el van mentve. Végül a válasz üzenetben egy 200-as kóddal elküldjük a konstans tartalmát.

A következő metódus a FavouritesDeleteController, ami egy bejelentkezett felhasználó kedvencébe elmentett játékot törli ki a kedvenceiből.

// Ez a metódus a kitörli a bejelentkezett felhasználó kedvencekbe mentet játékát,

    // hogy melyik játékot, azt a body-ban kell megadni

    FavouritesDeleteController: async(req, res) => {

        try {

            // Lekérjük a felhasználó azonosírtóját, és hiba üzenetet küldünk, ha ez nem sikerül, mert nem bejelentkezett felhaszánó

            // nem fér hozzá ehhez a funkcióhoz

            const userId = await jweMethods.GetUserId(req);

            if (userId === undefined) {

                res.status(401).json({

                    error: true,

                    message: "The token is empty or faulty!"

                });

                return;

            }

Konstansban elmentjük a felhasználó azonosítóját, amit a jweMethods-ból a GetUserId metódusával kérünk le, ha nem sikerült lekérni az arra utal, hogy nincs megadva token vagy hibás, erre hiba üzenetet küldünk.

// Ha nem kapjuk meg a játék azonosítóját, akkor erre hiba üzenetet küldünk

            const gameId = req.body.gameId;

            if (!gameId) {

                res.status(400).json({

                    error: true,

                    message: "The game's id is missing!"

                });

                return;

            }

            // Ha nem számot kapunk értékül, akkor hibát adunk rá

            if (isFinite(gameId) === false) {

                res.status(400).json({

                    error: true,

                    message: "The game's id is not a number!"

                });

                return;

            }

Itt lementjük a kérés body-ban lévő játék azonosítóját, majd ellenőrizzük, hogy sikerült-e, ha nem akkor egy 400-as kódú választ adunk. Ezután megnézzük, hogy a lementett paraméter valóbán szám-e, ha nem akkor egy 400-as kódú választ küldünk.

// Ellenőrizzük, hogy a játék, amelyet törölni szeretnénk az tényleg le volt-e mentve a felhasználónál,

            // ha nem, akkor erre hibát dobunk

            const favouriteToDelete = await Favourite.findOne({

                where: {

                    GameId: gameId,

                    UserId: userId

                }

            });

            if (!favouriteToDelete) {

                res.status(404).json({

                    error: true,

                    message: "The game was not saved in the favourites!"

                });

                return;

            }

Itt megkeressük az adott játék és felhasználó kapcsolatot, a lekért felhasználó és játék azonosító segítségével. Ha nincs ilyen kapcsolat akkor egy 404-es kódú választ küldünk.

// Ha nem volt probléma az ellenőrzések során, akkor kitöröljük a játékot a felhasználó kedvenceiből, és a válaszban

            // megerősítjük, hogy sikeres volt

            await favouriteToDelete.destroy();

            res.status(200).json({

                error: false,

                message: "The game is removed from the user's favourites!",

            });

            return;

Ha minden ellenőrzésen sikeres átment, akkor töröljük a játékos és játék kapcsolatát, majd egy http 200-as kódú válasszal megerősítjük a sikeres törlést.

gamepage.controller.js működése

import Game from "../models/game.js";

import Gamepicture from "../models/gamepicture.js";

import Studio from "../models/studio.js";

import Studiosgame from "../models/studiosgame.js";

import Rating from "../models/rating.js";

import Agerating from "../models/agerating.js";

import Tag from "../models/tag.js";

import Gamestag from "../models/gamestag.js";

import Gamesaward from "../models/gamesaward.js";

import Award from "../models/award.js";

import Gameslanguage from "../models/gameslanguage.js";

import Language from "../models/language.js";

import Gamesplatform from "../models/gamesplatform.js";

import Platform from "../models/platform.js";

import Acting from "../models/acting.js";

import Actor from "../models/actor.js";

import Creation from "../models/creation.js";

import Creator from "../models/creator.js";

import Pcspec from "../models/pcspec.js";

import Favourite from "../models/favourite.js";

import jweMethods from "../utilities/jwe.methods.js";

import Gamesagerating from "../models/gamesagerating.js";

import Review from "../models/review.js";

import User from "../models/user.js";

Itt importáljuk az összes a fájlban felhasznált adatbázis modellt.

Az fájl első metódusa a GamepageGetController ami az URL paraméterében megadott játék adatait kéri le, amiket szeretnénk megjeleníteni az oldalon.

// Ez a metódus egy adott játéknak az összes adatát lekérdezi

    GamepageGetController: async (req, res) => {

        // Az url paramétere tartalmazza a játék azonosítóját, ezt elmentjük és megkeressük az adott játékot

        try {

            const gameId = Number(req.params.gameId?.trim());

            if (!gameId || isFinite(gameId) === false) {

                res.status(400).json({

                    error: true,

                    message: "There's no game id!"

                });

                return;

            }

Lementjük a paraméterben megadott számot és megbizonyosodunk közben, hogy a megfelelő formátumban legyen. Ez után ellenőrizzük, hogy sikeresen megkaptuk a paraméter értékét és hogy biztos szám-e, ha nem akkor egy 400-as kódú választ adunk.

// Megkeressük az URL-ben megadott számhoz tartozó játékot

            const game = await Game.findOne({

                where: {

                    id: gameId

                }

            });

            // Ha az adatbázisban nincs az URL-ben megadott számhoz tartozó játék akkor erre hiba üzenetet küldünk

            if (!game) {

                res.status(404).json({

                    error: true,

                    message: "There's no game with this id!"

                });

                return;

            }

Megkeressük a megadott azonosítóhoz tartozó játékot a játék azonosítója alapján, ha nem találunk ehhez azonosítóval játékot, akkor egy 404-es kódú hibát küldünk.

// Lementjük az adott játékhoz tartozó képeket

            const pictures = await Gamepicture.findAll({

                attributes: ['url'],

                where: {

                    gameid: gameId

                }

            });

Ha találtunk az azonosítóhoz játékot, akkor elkezdjük a játék adatait lekérni, kezdve a játékhoz tartozó képernyőképeket.

// Megkeressük, hogy az adott játékhoz milyen stúdiók vettek részt a fejlesztésben

            const developers = await Studio.findAll({

                attributes: ["id", "name"],

                include: {

                    where: {id: gameId},

                    attributes: [],

                    model: Game,

                    through: {

                        attributes: [],

                        model: Studiosgame,

                        where: {isDeveloper: true}

                    }

                },

                raw: true

            });

            // Megkeressük, hogy az adott játékhoz milyen stúdiók vettek részt a kiadásában

            const publishers = await Studio.findAll({

                attributes: ["id", "name"],

                include: {

                    where: {id: gameId},

                    attributes: [],

                    model: Game,

                    through: {

                        attributes: [],

                        model: Studiosgame,

                        where: {isPublisher: true}

                    }

                },

                raw: true

            });

Itt lekérdezzük a játék fejlesztőit és kiadóit, a játék és stúdió kapcsolatát a Studiosgame-ben van meghatározva, egy isDeveloper egy logikai értéket tároló oszlop, ahol az igaz azt jelenti, hogy a stúdió a megadott játék fejlesztője, az isPublisher is egy logikai értéket tároló oszlop, ahol az igaz azt jelenti, hogy a stúdió a játék kiadója.

// Megkeressük és kiszámoljuk, hogy a játék értékelése hány százalékban pozitív

            const positiveRatings = await Rating.count({where: {GameId: gameId, positive: true}});

            const allRatings = await Rating.count({where: {GameId: gameId}});

            const rating = Math.round((positiveRatings / allRatings) \* 100)

Először lementjük, hogy az adott játéknál mennyi pozitív értékelés van, utána meg hogy összesen mennyi értékelés van. Ezek segítségével kiszámoljuk a rating konstansban, hogy az értékelések hány százaléka pozitív, ezt pedig matematikailag kerekítjük.

// Megkeressük, hogy az adott játékhoz milyen korhatár besorolások tartoznak és azok képeit elmentjük

            const agerating = await Agerating.findAll({

                attributes: ["url"],

                include: {

                    where: { id: gameId },

                    model: Game,

                    through: {

                        model: Gamesagerating,

                        attributes: [],

                    },

                    attributes: [],

                },

                raw: true

            });

            // Megkeressük az adott játékhoz tartozó műfajokat

            const genres = await Tag.findAll({

                attributes: ["tag"],

                include: {

                    where: { id: gameId},

                    model: Game,

                    through: {

                        model: Gamestag,

                        attributes: [],

                    },

                    attributes: [],

                },

                raw: true

            });

Ezeknél a lekérdezéseknél a játék azonosítóját használjuk, hogy játék korhatár besorolásait, majd a játék címkéit.

 // Megkeressük, hogy az adott játékot milyen díjakra jelölték, amiket nem nyert el végül

            const nominations = await Award.findAll({

                attributes: ["organizer", "name"],

                include: {

                    where: { id: gameId},

                    model: Game,

                    through: {

                        model: Gamesaward,

                        where: { result: false },

                        attributes: ["year"]

                    },

                    attributes: []

                },

                raw: true

            });

            // Megkeressük, hogy az adott játékot milyen díjakra jelölték, amiket végül megnyert

            const wins = await Award.findAll({

                attributes: ["organizer", "name"],

                include: {

                    where: { id: gameId},

                    model: Game,

                    through: {

                        model: Gamesaward,

                        where: { result: true },

                        attributes: ["year"]

                    },

                    attributes: []

                },

                raw: true

            });

Ezeknél a lekérdezéseknél a ugyanabban a táblában keressünk az játék azonosító alapján, azt hogy csak jelöléseket vagy nyeréseket keresünk a Gamesaward modellben a result logikai értéket tároló oszlopban dől el, az igaz nyerést jelent, a hamis csak jelölést.

// Megkeressük, hogy az adott játék milyen nyelveket támogat

            const languages = await Language.findAll({

                attributes: ["language"],

                include: {

                    where: { id: gameId},

                    model: Game,

                    through: {

                        model: Gameslanguage,

                        attributes: ["dub"]

                    },

                    attributes: []

                },

                raw: true

            });

            // Megkeressük, hogy az adott játék milyen platformokon elérhető

            const platforms = await Platform.findAll({

                attributes: ["platform"],

                include: {

                    where: { id: gameId },

                    model: Game,

                    through: {

                        model: Gamesplatform,

                        attributes: []

                    },

                    attributes: []

                },

                raw: true

            });

Itt a játékokhoz tartozó nyelveket, majd a platformokot mentjük le a játék azonosítója alapján.

// Megkeressük, hogy az adott játékban milyen színészek vettek részt és hogy milyen szerepben

            const actors = await Actor.findAll({

                attributes: ["firstName", "lastName", "profilePicture"],

                include: {

                    where: { id: gameId },

                    model: Game,

                    through: {

                        model: Acting,

                        attributes: ["role"]

                    },

                    attributes: []

                },

                raw: true

            });

            // Megkeressük, hogy az adott játék fejlesztésében kik vettek részt és milyen szerepben

            const creators = await Creator.findAll({

                attributes: ["firstName", "lastName"],

                include: {

                    where: { id: gameId},

                    model: Game,

                    through: {

                        model: Creation,

                        attributes: ["field"]

                    },

                    attributes: []

                },

                raw: true

            });

Ennél a két kérésnél személyeket kérünk le, az elsőnél a színészeket mentjük el, beleértve, hogy milyen szerepe volt az adott játéknál. Majd a készítőket kérjük le, beleértve, hogy milyen területen dolgoztak a játéknál. Ezekhez a játék azonosítóját használjuk.

// Megkeressük, hogy az adott játék gépigényét

            const pcspec = await Pcspec.findOne({

                attributes: ["minop", "mincpu", "minram", "mingpu", "mindirectx", "op", "cpu", "ram", "gpu", "directx", "storage", "sidenote"],

                where: {

                    GameId: gameId

                }

            });

            // Megkeressük, hogy az adott játékhoz az oldal felhasználói milyen kritikákat írtak, illetve azt is hogy milyen értékelést

            // adtak a játékra

            const reviews = await Review.findAll({

                attributes: ["id", "title", "content", "date"],

                where: {

                    GameId: gameId

                },

                include: {

                    model: User,

                    attributes: ["username"],

                    include: {

                        where: {

                            GameId: gameId

                        },

                        model: Rating,

                        attributes: ["positive"]

                    }

                },

                order: [["date", "DESC"]],

                raw: true,

            });

Itt az játék azonosítója alapján lekérjük a játék gépigényét, ezután a játékhoz írt kritikákat kérjük le, beleértve a kritika íróját és a hogy hogyan értékelte, ezeket a létrehozásuk ideje alapján rendezzük, kezdve a legújabbtól.

// Ha minden lekérdezés sikeres, akkor minden információt elküldünk egy válaszban

            res.status(200).json({

                error: false,

                message: "Data fetch was successfull!",

                datas: {

                    title: game.gameTitle,

                    altTitle: game.altGameTitle,

                    description: game.description,

                    boxart: game.boxart,

                    gallery: pictures,

                    release: game.release,

                    developers: developers,

                    publishers: publishers,

                    rating: rating,

                    agerating: agerating,

                    genres: genres,

                    controllerSupport: game.controllerSupport,

                    crossplatform: game.crossplatform,

                    crossPlatformException: game.crossPlatformException,

                    awards: {

                        nominations: nominations,

                        wins: wins,

                    },

                    languages: languages,

                    platforms: platforms,

                    actors: actors,

                    creators: creators,

                    pcspec: pcspec,

                    reviews: reviews

                }

            });

            return;

Végül egy http 200-as válaszban megadunk minden lekért adatot.

A GamepagePostController egy bejelentkezett felhasználó a paraméterben megadott játék kedvencekbe mentésért kezeli.

//  Ez a metódus feladata hogy egy bejelentkezett felhasználó elmentse a játékokat a kedvencekbe

    GamepagePostController: async (req, res) => {

        try {

            // Lekérjük a felhasználó azonosítóját a további műveletekhez

            const userId = await jweMethods.GetUserId(req);

            // Ha nem sikerült a felhasználó azonosítóját lekérni, akkor egy hiba üzenetet küldünk vissza

            if (userId === undefined) {

                res.status(401).json({

                    error: true,

                    message: "The user is not logged in or the token is faulty!"

                });

                return;

            };

            // Az URL-ben szereplő játék azonosított lementjük a megfelelő formátumban

            const gameId = Number(req.params.gameId?.trim());

            // Ha a játék azonosító hiányzik vagy nem egy szám, akkor egy hiba üzenetet küldünk vissza

            if (!gameId || isFinite(gameId) === false) {

                res.status(400).json({

                    error: true,

                    message: "The games's id is missing!"

                });

                return;

            }

            // Megnézzük, hogy az URL-ben megadott számhoz tartozik-e játék, ha nem akkor egy hiba üzenetet adunk vissza

            const gameExist = await Game.findOne({

                where: {

                    id: gameId

                }

            });

            if (!gameExist) {

                res.status(404).json({

                    error: true,

                    message: "There's no game with this id!"

                });

                return;

            }

Először a megnézzük, hogy a van-e azonosító az URL paraméterében, majd a ellenőrizzük hogy a kapott azonosítő tényleg szám, és megnézzük hogy van-e ilyen azonosítóval játék az adatbázisban.

// Ellenőrizzük, hogy a felhasználó hozzá adta-e már a játékot a kedvencekbe,

            // ha igen, akkor egy hiba üzenetet küldünk vissza

            const conflict = await Favourite.findOne({

                where: {

                    UserId: userId,

                    GameId: gameId

                }

            });

            if (conflict) {

                res.status(409).json({

                    error: true,

                    message: "The user has already added the game to favourites!"

                });

                return;

            }

            // Ha minden ellenőrzésen átment, akkor a játékot elmentjük a kedvencekbe, és egy megerősítő üzenetet küldünk róla

            await Favourite.create({

                UserId: userId,

                GameId: gameId

            });

            res.status(201).json({

                error: false,

                message: "The game has been successfuly added to favourites!"

            });

            return;

Itt ellenőrizzük, hogy nincs-e már a felhasználónál elmentve az adott játék, ha nincs akkor elmentjük a felhasználóhoz a játékot a Favourite modellben.

A GamepagePutController-nél egy bejelentkezett felhasználó játék értékelését mentjük el.

// Ez a metódus egy bejelentkezett felhasználónál teszi lehetővé, hogy értékelje a játékot, ehhez a kérés body-ban lévő

    // logikai értéket veszi alapul, ahol a false negatív értékelést, míg a true pozitív értékelést jelent

GamepagePutController: async (req, res) => {

        // Lekérjük a felhasználó azonosítóját a további műveletekhez

        const userId = await jweMethods.GetUserId(req);

        // Ha nem sikerült a felhasználó azonosítóját lekérni, akkor egy hiba üzenetet küldünk vissza

        if (userId === undefined) {

            res.status(401).json({

                error: true,

                message: "The user is not logged in or the token is faulty!"

            });

            return;

        };

        // Az URL-ben szereplő játék azonosított lementjük a megfelelő formátumban

        const gameId = Number(req.params.gameId?.trim());

        // Ha a játék azonosító hiányzik vagy nem egy szám, akkor egy hiba üzenetet küldünk vissza

        if (!gameId || isFinite(gameId) === false) {

            res.status(400).json({

                error: true,

                message: "The games's id is missing!"

            });

            return;

        }

        // Lekérjük a játék adatait az megadott azonosító alapján

        const existingGame = await Game.findOne({

            attributes: ["gameTitle", "release"],

            where: {

                id: gameId

            }

        });

        // Ha az adott azonosítóhoz nem tartozik játék, akkor erre egy hiba üzenetet küldünk

        if (!existingGame) {

            res.status(404).json({

                error: true,

                message: "There isn't any game with this id!"

            });

            return;

        };

Először meghatározzuk a felhasználó azonosítóját, majd a paraméterben megadott játék azonosítóját elmentjük, és ellenőrizzük, hogy mindkettőt megkaptuk. Utána megnézzük, hogy van-e ilyen azonosítóval játék.

// Először lementjük a játék megjelenési dátumot a megfelelő dátum formában, majd lementjük a jelenlegi dátumot,

        // utána megnézzük, hogy az adott játék megjelent, ha nem, akkor a felhasználó nem tudja értékelni

        // és ez esetben egy hiba üzenetet adunk

        const gamesReleaseDate = new Date(existingGame.release);

        const currentDate = new Date();

        if (gamesReleaseDate > currentDate) {

            res.status(401).json({

                error: true,

                message: "You can't rate a game when it's not released yet!"

            });

            return;

        }

        // Ellenőrizzük, hogy a felhasználó értékelte-e már a játékot, ha igen, akkor ne tudja ezt megismételni és erre hiba üzenetet

        // adunk

        const conflict = await Rating.findOne({

            where: {

                UserId: userId,

                GameId: gameId

            }

        });

        if (conflict) {

            res.status(409).json({

                error: true,

                message: "The user has already rated the game!"

            });

            return;

        }

        // Lementjük a body-ban lévő értéket, ha ez valamilyen formában üres, akkor hiba üzenetet adunk

        const isPositive = req.body.isPositive;

        if (isPositive === "" || isPositive === null || isPositive === undefined) {

            res.status(400).json({

                error: true,

                message: "The rating is missing!"

            });

            return;

        }

        // Ellenőrizzük, hogy a body-ban lévő érték ténylegesen logikai érték, ha nem, akkor egy hiba üzenetet küldünk

        if (isPositive !== true && isPositive !== false) {

            res.status(400).json({

                error: true,

                message: "The rating is not a boolean!"

            });

            return;

        }

        // Ha minden ellenőrzésen átment, akkor elmentjük a felhasználó értékelését és egy megerősítő választ adunk

        await Rating.create({

            GameId: gameId,

            positive: isPositive,

            UserId: userId

        });

res.status(201).json({

            error: false,

            message: "The rating has been saved!"

        })

        return;

Itt megnézzük, hogy az adott játék megjelent-e, ha nem akkor ne tudja értékelni a felhasználó a játékot. Majd megnézzük, hogy a felhasználó értékelte-e már a játékot, ha nem akkor lementjük a body-ból kapott játék értékelést, majd ellenőrizzük, hogy tényleg logikai érték. Ha eddig minden sikeres, akkor elmentjük az értékelést és elküldjük a megerősítő választ.

**login.controller.js működése**

Ebben a fájlban egy metódus található, amiben a felhasználó bejelentkezését kezeljük.

import jwtHandler from "../utilities/jwe.methods.js";

import bcryptMethods from "../utilities/bcrypt.methods.js";

import User from "../models/user.js";

Itt beimportáljuk a szükséges metódusokat és modellt.

A fájlban lévő egyetlen metódus a LoginPostController.

// A bejelentkezést kezelő metódus

    LoginPostController: async (req, res) => {

        try {

            // Mivel ezeknek az értéke nem változik, konstansként mentjük el hogy tovább dolgozzunk vele

            const {username: loginUsername, password: loginPassword} = req.body;

        // Ha bármelyik mező üres, akkor az ahhoz megfelelő hiba kódot és üzenetet küldjük

            if (!loginUsername || !loginPassword) {

                res.status(400).json({

                    error: "true",

                    message: "Not every field was filled!"

                });

                return;

            }

            // Konstansként elmentjük azt a felhasználót az adatbázisból amelyik neve megegyezik a megadottal

            const correctUser = await User.findOne({

                where: {username: loginUsername}

            });

            // Ha nincs teljes egyezés, akkor hiba üzenetet küldünk

            if (!correctUser) {

                res.status(400).json({

                    error: "true",

                    message: "The username or password is incorrect!"

                });

                return;

            }

            // Itt ellenőrizzük, hogy a felhasználó titkosított jelszava visszafejtve megegyezik-e a felhasználóval megadott jelszóval

            const correctPassword = bcryptMethods.Comparing(loginPassword, correctUser.password);

            // Ha a jelszó helyesen van megadva akkor a felhasználóhoz készül egy token és beengedi az oldalra

            // Más esetben hiba üzenetet küld, hogy a felhasználónév vagy a jelszó helytelen

            if (correctPassword === true) {

                const token = await jwtHandler.CreatingToken(correctUser.id, correctUser.username, correctUser.email);

                res.setHeader("Authorization", `Bearer ${token}`);

                res.status(200).json({

                    error: "false",

                    message: "Succefull login",

                    isAdmin: correctUser.admin,

                    token: token

                });

                return;

            }

            else {

                res.status(400).json({

                    error: "true",

                    message: "The username or password is incorrect!"

                });

                return;

            }

Itt először elmentjük a kapott felhasználónevet és a jelszót. Ellenőrizzük, hogy ténylegesen megkaptuk-e az adatokat. Ezek után megnézzük, hogy van-e ilyen névvel felhasználó az adatbázisban, majd megnézzük, hogy a megadott jelszó megegyezik-e az adatbázisban lévő titkosított jelszóval, ehhez a bcrypt egy metódusát használjuk. Ha a jelszó is megegyezik akkor létrehozunk egy token-t amit be teszünk az Authorization fejlécbe és elküldjük a sikeres válaszban is.

**A mainpage.controller.js magyarázata**

Ebben a fájlban két metódus található, az egyik a weboldal fő oldalához tartozik, a másik a navigációs sávhoz tartozik, amely minden oldalon elérhető.

import { Op } from "sequelize";

import Game from "../models/game.js";

import Tag from "../models/tag.js";

import Gamestag from "../models/gamestag.js";

Itt beimportáljuk a szükséges funkciót és modelleket.

A fájl első metódusa a MainpageGetController, ami a főoldalhoz tartozó lekérdezéseket kezeli.

    MainpageGetController: async(req, res) => {

        // Ez a főoldalhoz kéri le az adatbázisból az adatokat, így egyből try catch párban kezdjük a kódot

        try {

            // Először azokat a játékokat kérjük le, amelyek az elkövetkező 12 hónapban megjelennek

            const currentDate = new Date();

            const oneYearForward = new Date();

            oneMonthForward.setMonth(currentDate.getMonth() + 12);

            const upcomingGames = await Game.findAll({

                attributes: ["id", "promoArt", "release"],

                where: {

                    release: {

                        [Op.between]: [currentDate, oneYearForward]

                    }

                }

            });

            // Most azokat a játékokat kérjük le, amelyek 12 hónapon belül jelentek meg

            const oneYearBack = new Date();

            oneMonthBack.setMonth(currentDate.getMonth() - 12);

            const newReleaseGames = await Game.findAll({

                attributes: ["id", "gameTitle", "boxart"],

                where: {

                    release: {

                        [Op.between]: [oneYearBack, currentDate]

                    }

                }

            });

            // Itt 5 játék kategóriát küldünk le, amik: Shooter, Adventure, RPG, Racing, Strategy

            // Egy kategóriához maximum 15 játék tartozik és ABC sorrendben jelennek meg, képpekkel együtt

            // 15 Shooter/lövöldözős játék

            const shooters = await Game.findAll({

                attributes: ["id", "gameTitle", "boxart"],

                limit: 15,

                where: {release: {

                    [Op.lte]: currentDate

                }},

                include: [{

                        model: Tag,

                        through: Gamestag,

                        where: { tag: "Shooter" },

                        attributes: [],

                        required: true,

                    }],

            });

            // '15 Adventure/Kaland játék

            const adventures = await Game.findAll({

                attributes: ["id", "gameTitle", "boxart"],

                limit: 15,

                where: {release: {

                    [Op.lte]: currentDate

                }},

                include: [{

                        model: Tag,

                        through: Gamestag,

                        where: { tag: "Adventure" },

                        attributes: [],

                        required: true,

                    }],

            });

            // 15 RPG/Szerep játék

            const rpgs = await Game.findAll({

                attributes: ["id", "gameTitle", "boxart"],

                limit: 15,

                where: {release: {

                    [Op.lte]: currentDate

                }},

                include: [{

                        model: Tag,

                        through: Gamestag,

                        where: { tag: "RPG" },

                        attributes: [],

                        required: true,

                    }],

            });

            // 15 Racing/Versenyzős játék

            const racings = await Game.findAll({

                attributes: ["id","gameTitle", "boxart"],

                limit: 15,

                where: {release: {

                    [Op.lte]: currentDate

                }},

                include: [{

                        model: Tag,

                        through: Gamestag,

                        where: { tag: "Racing" },

                        attributes: [],

                        required: true,

                    }]

            });

            // 15 Strategy/Stratégiai játékok

            const strategies = await Game.findAll({

                attributes: ["id", "gameTitle", "boxart"],

                limit: 15,

                where: {release: {

                    [Op.lte]: currentDate

                }},

                include: [{

                        model: Tag,

                        through: Gamestag,

                        where: { tag: "Strategy" },

                        attributes: [],

                        required: true

                    }],

            });

            // Ha minden rendben futott le, akkor a válaszban megadjuk a lementett értékeket

            res.status(200).json({

                error: false,

                message: "Game datas successfully fetched!",

                datas: {

                    upcomingGames,

                    newReleaseGames,

                    shooters,

                    adventures,

                    rpgs,

                    racings,

                    strategies

                }

            });

            return;

Az elején meghatározzuk a jelenlegi dátumot, majd az egy évvel későbbi dátumot, így lekérjük azokat a játékokat, amelyek megjelenítési a maira rá egy évvel jelenik meg. Ez után meghatározzuk a mostani dátumtól egy évvel ezelőtti dátumot, lekérjük azokat a játékokat, amelyek az elmúlt egy évben jelentek meg, ezekután öt címke alapján kérünk le játékokat, maximum 15-öt. A játékoknak csak azokat az adatait kérjük le, amit az oldalon szeretnénk megjeleníteni. A végén a válaszban minden megadott játék adatait elküldjük.

A MainpagePutController metódus a keresősáv működéséért felel, ez a weboldalon a navigációs sávon található, így minden oldalon elérhető.

// Ez a metódus a kereső sávért felel

    MainpagePutController: async (req, res) => {

        try {

            // Le mentjük a kapott keresést és a felesleges keresést elkerülve ellenőrizzük, hogy a nem-e üres, de alapvetően ez nem számít

            // hibának, így a 200-as kódot küldjük vissza, de megüzenjük, hogy ez üres

            const search = req.body.search;

            if (!search) {

                res.status(200).json({

                    error: false,

                    message: "It is an empty search!"

                });

                return;

            }

            // Az átláthatóság érdekében egy új változóban formázzuk a keresés szövegét az adatbázishoz való lekérdezéshez

            const formattedSearch = "%" + String(search).toLowerCase() + "%";

            // A szükséges információkat kérjük le csak, a lekérdezés érvényes a játék alternatív nevére is, ezután 200-as kóddal

            // visszaadjuk az eredményt/eredményeket

            const game = await Game.findAll({

                attributes: ["id", "gameTitle", "altGameTitle", "release", "boxart"],

                where: {[Op.or]:

                    [

                        {gameTitle: {[Op.like]: formattedSearch}},

                        {altGameTitle: {[Op.like]: formattedSearch}}

                    ]

                }

            });

            // Ha minden helyesen futott le, akkor visszaadjuk a talált eredményeket

            res.status(200).json({

                error: false,

                message: "Successful search!",

                game: game

            });

        }

Először elmentjük a body-ban kapott keresendő szöveget, ha ez üres akkor válaszban elküldjük, hogy ez egy üres keresés volt, ha van szöveg akkor formázzuk, hogy az SQL keresésben bármilyen szöveg lehet előtte és utána és kisbetűsre alakítjuk. Majd elküldünk egy kérést, ahol a formázott szöveggel a játék alap és alternatív címe alapján is keressen, majd az eredményt küldje el a válaszban.

**A myproflie.controller.js magyarázata**

Ebben a fájlban egy bejelentkezett felhasználó fiókjához tartozó három metódust tartalmaz.

import jweMethods from "../utilities/jwe.methods.js";

import validationMethods from "../utilities/validation.methods.js";

import bcryptMethods from "../utilities/bcrypt.methods.js";

import User from "../models/user.js";

Itt három metódust importálunk be és egy modellt.

Az első metódus a MyProfileGetController egy bejelentkezett felhasználó adatinak a lekérdezést kezeli.

 MyprofileGetController: async(req, res) => {

        try {

            // Lekérjük a bejelentkezett felhasználó azonosítóját, ha nincs, akkor hiba üzenetet küldünk

            const userId = await jweMethods.GetUserId(req);

            if (userId === undefined) {

                res.status(401).json({

                    error: true,

                    message: "The token is missing or faulty!"

                });

                return;

            }

            // Az azonosító alapján megkeressük a felhasználót

            const user = await User.findOne({

                attributes: ["username", "email", "admin", "creation"],

                where: {

                    id: userId

                }

            });

            // Végül a válaszban visszaadjuk a felhasználó adatait

            res.status(200).json({

                error: false,

                message: "The user's data are successfully fetched!",

                user: user

            });

            return;

        }

Először lekérdezzük a felhasználó azonosítóját, majd megkeressük az alapján a felhasználót az adatbázisban, végül a válaszban elküldjük a felhasználó adatait.

    // Ez a metódus a bejelentkezett felhasználó adatainak a változtatásait kezeli

    MyprofilePostController: async(req, res) => {

        try {

            // Lekérjük a bejelentkezett felhasználó azonosítóját, ha nincs, akkor hiba üzenetet küldünk

            const userId = await jweMethods.GetUserId(req);

            if (userId === undefined) {

                res.status(401).json({

                    error: true,

                    message: "The token is missing or faulty!"

                });

                return;

            }

            // Konstansként lementjük a req.body-ban lévő értékékeket, illetve létrehozunk egy objektumot, amely leköveti hogy

            // a felhasználó milyen adatokat változtat és hogy mire

            const {username, email, password, passwordAgain} = req.body;

            const changes = new Object();

            // Egy változóban llenőrizzük hogy történik-e változtatás

            let empty = true;

            // A lekért azonosító alapján lekérjük a felhasználó adatait

            const user = await User.findOne({where: {id: userId}, attributes: ["username", "password", "email"]});

            // A felhasználó nevet ellenőrizzük, hogy meglett-e adva, ha nem, akkor megyünk a következő adatra

            if (username) {

                // Ha a megadott felhasználó név ugyanaz, mint az eredeti, akkor hiba üzenetet küldünk

                if (username === user.username) {

                    res.status(409).json({

                        error: "true",

                        message: "The username is the same as the original!"

                    });

                    return;

                }

                // Ha a megadott névvel már létezik felhasználó, akkor hiba üzenetet küldünk

                const conflictingUsername = await User.findOne({where: {username: username}});

                if (conflictingUsername) {

                    res.status(409).json({

                        error: "true",

                        message: "There's already an user with this username!"

                    });

                    return;

                }

                else {

                    // Megnézzük, hogy a felhasználónév formátuma helyes, ha nem akkor hiba üzenetet küldünk

                    if (validationMethods.CheckUsername(username) === false) {

                        res.status(400).json({

                            error: "true",

                            message: "The username is not in the correct length or contains space!"

                        });

                        return;

                    }

                    // Megnézzük, hogy a felhasználónév tartalmaz-e káromkodást, ha igen, akkor hiba üzenetet küldünk

                    if (validationMethods.CheckProfanity(username) === true) {

                        res.status(400).json({

                            error: "true",

                            message: "The username contains profanity!"

                        });

                        return;

                    }

                }

                // Ha minden ellenőrzésen átment, akkor megadjuk, hogy a változtatás történt, szóval nem lesz üres kérés, illetve

                // elmentjük a változtatást

                empty = false;

                changes.username = username;

            };

            // Ha az email cím meg van adva akkor ellenőrizzük az adatot, ha nincs, akkor tovább haladunk a következő adatra

            if (email) {

                // Ellenőrizzük, hogy az megadott új email nem-e egyezik a régivel, ha igen, erre hiba üzenetet küldünk

                if (email === user.email) {

                    res.status(409).json({

                        error: true,

                        message: "The email is the same as the original!"

                    });

                    return;

                }

                // Megnézzük, hogy az új email cím nem-e egyezik már egy adatbázisban lévő email címmel,

                // ha igen, erre hibaüzenetet küldünk

                const conflictingEmail = await User.findOne({where: {email: email}});

                if (conflictingEmail) {

                    res.status(409).json({

                        error: "true",

                        message: "There's already an user with this email address!"

                    });

                    return;

                }

                // Ha minden ellenőrzésen átment, akkor megadjuk, hogy a változtatás történt, szóval nem lesz üres kérés, illetve

                // elmentjük a változtatást

                empty = false;

                changes.email = email;

            };

            // Megnézzük, hogy a jelszó mezőbe kaptunk-e eredményt, ha nem, akkor tovább haladunk

            if (password) {

                // Ha hiányzik a jelszó megerősítés, akkor hiba üzenetet küldünk

                if (!passwordAgain) {

                    res.status(400).json({

                        error: "true",

                        message: "The password confirmation is empty!"

                    });

                    return;

                }

                else {

                    // Megnézzük, hogy a jelszó megegyezik-e a megerősítéssel, ha nem erre hiba üzenetet küldünk

                    if (password === passwordAgain) {

                        // Megnézzük, hogy a jelszó a megfelelő formátumban van, ha nem akkor hiba üzenetet küldünk

                        if (validationMethods.CheckPassword(password) === true) {

                            // Megnézzük, hogy a új jelszó megegyezik-e a régi jelszóval, ha igen, akkor erre egy hibát küldünk

                            if (bcryptMethods.Comparing(password, user.password) === true) {

                                res.status(400).json({

                                    error: "true",

                                    message: "The password is the same as the original!"

                                });

                                return;

                            }

                            // Ha minden ellenőrzésen átment, akkor megadjuk, hogy a változtatás történt, szóval nem lesz üres kérés, illetve

                            // elmentjük a változtatást

                            else {

                                empty = false;

                                changes.password = bcryptMethods.Hashing(password);

                            }

                        }

                        else {

                            res.status(400).json({

                                error: "true",

                                message: "The password is in incorrect form!"

                            });

                            return;

                        }

                    }

                    else {

                        res.status(400).json({

                            error: "true",

                            message: "The passwords don't match!"

                        });

                        return;

                    }

                }

            }

            else {

                // Ha a jelszó mező üres, de a megerősítésnek van tartalma akkor erre hibát adunk

                if (passwordAgain) {

                    res.status(400).json({

                        error: "true",

                        message: "The password field is empty!"

                    });

                    return;

                }

            }

            // Ha végül metódus úgy fut let hogy semmilyen változtatás sem történik, vagyis minden mező üres, akkor hibát küldünk rá

            if (empty) {

                res.status(400).json({

                    error: true,

                    message: "Every field is empty!"

                });

                return;

            }

            // Ha minden ellenőrzésen átment, akkor a felhasználó azonosítója alapján megváltoztatjuk a felhasználó adatait

            await User.update(changes, {where: {id: userId}});

            // Végül visszaküldjük, hogy a metódus sikeresen lefutott

            res.status(201).json({

                error: false,

                message: "Datas has been updated!"

            });

            return;

        }

Lekérjük a felhasználó azonosítóját, majd ellenőrizzük, hogy megkaptuk-e. Ezekután lementjük a body-ban kapott értékeket, emellé létrehozzuk egy objektumot, amiben lekövetjük az adatváltoztatásokat, ha átmentek az ellenőrzéseken, és egy logikai változót, amiben azt tároljuk, hogy történik-e adat változtatás. Ezek után jönnek az adatok ellenőrzései, amik során megnézzük, hogy nem-e ugyanaz, mint az eredeti, nincs-e már ilyen adat az adatbázisban (kivéve a jelszónál), a formátumnak megfelelnek-e és nem tartalmaznak káromkodást. Ha nem kaptunk semmilyen adatot akkor a fent meghatározott logikai érték igaz marad, amire egy 400-as hibát adunk. Ha volt adatváltoztatás és átment az ellenőrzéseken, akkor azt az objektumba elmentjük olyan formátumban, hogy azt használva frissíteni tudjuk a felhasználó adatait.

A MyprofileDeleteController metódus egy bejelentkezett felhasználó fiók törlését kezeli.

// Ez a metódus egy bejelentkezett felhasználó fiókjának a törlését intézi

    MyprofileDeleteController: async(req, res) => {

        try {

            // Lekérjük a bejelentkezett felhasználó azonosítóját, ha nincs, akkor hiba üzenetet küldünk

            const userId = await jweMethods.GetUserId(req);

            if (userId === undefined) {

                res.status(401).json({

                    error: true,

                    message: "The token is missing or faulty!"

                });

                return;

            }

            // Ha az ellenőrzéseken sikeresen átment, akkor a töröljük a felhasználót, majd visszaküldjük a megerősítő választ,

            // hogy sikerült a törlés

            await User.destroy({

                where: {

                    id: userId

                }

            });

            res.status(200).json({

                error: false,

                message: "The user profile has been deleted!"

            });

            return;

        }

        catch (error) {

            console.log(error);

            res.status(500).json({

                error: true,

                message: "Something went wrong fetching the user's data!"

            });

            return;

        }

    }

Lekérjük a felhasználó adatait, majd megnézzük, hogy megkaptuk-e. Majd a kapott azonosító alapján töröljük a felhasználót és visszaküldünk egy megerősítő választ.

**regist.controller.js működése**

Ebben a fájlban csak egy metódus található, ami a felhasználó regisztrációját kezeli.

import User from '../models/user.js';

import { Op } from 'sequelize';

import bcryptMethods from '../utilities/bcrypt.methods.js';

import validationMethods from '../utilities/validation.methods.js';

Itt importáljuk szükséges modellt és metódusokat.

A RegistPostMetódus kezeli egy felhasználó regisztrációját. ehhez az adatokat a kérés body-ból kapja meg.

 RegistPostController: async (req, res) => {

        // Try catch párban vannak írva a következő kódok, hogy az esetleges adatbázis csatlakozási hibákat kezelni tudjuk

        try {

            // Konstansként elmentjük azokat az adatokat, amelyeket bekérünk a felhasználótól, mert ezek nem változnak

            const { username: registUsername, password: registPassword, passwordAgain: registPasswordAgain, email: registEmail} = req.body;

            // Ha a valamelyik mező nincs kitöltve akkor 400-as hibával visszaküldi, hogy hiba történt,

            // mert nem volt minden mező kitöltve és ezeket kötelező kitölteni

            if (!registUsername || !registPassword || !registEmail || !registPasswordAgain) {

                res.status(400).json({

                    error: "true",

                    message: "Not every field was filled!"

                });

                return;

            }

            // Ellenőrzi, hogy a bekért adatoknál van-e már egyező az adatbázisban

            const conflictingDatas = await User.findOne({

                where: {[Op.or]:

                    [

                        {email: registEmail},

                        {username: registUsername},

                    ]

                }

            });

            // Ha van egyező akkor küldjön vissza egy 409 http kódot és egy üzenetet, hogy valamelyik már létezik

            // Csak a felhasználónév és az email cím egyedi oszlop, ezért csak ezeket ellenőrizzük

            if (conflictingDatas) {

                res.status(409).json({

                    error: "true",

                    message: "There's already an user with this username or email address!"

                });

                return;

            }

            // A bekért adatokat a backend-en ellenőrizzük

            // Először a felhasználónév helyes méretét ellenőrizzük, majd hiba üzenetet ad, ha nem megfelelő tartományban van.

            // Mivel a bekért adat hiába karakterláncként értelmezi alapvetően,

            // de típuskényszerítéssel biztosabban elérhető, hogy a szükséges String metódusok elérhetőek legyenek

            if (validationMethods.CheckUsername(registUsername) === false) {

                res.status(400).json({

                    error: "true",

                    message: "The username is not in the correct length or contains space!"

                });

                return;

            }

            if (validationMethods.CheckProfanity(registUsername) === true) {

                res.status(400).json({

                    error: "true",

                    message: "The username contains profanity!"

                });

                return;

            }

            // Ha a jelszó megerősítés sikertelen a felhasználótól akkor hiba üzenetet küldd az esetre

            if (registPassword !== registPasswordAgain) {

                res.status(400).json({

                    error: "true",

                    message: "The passwords don't match!"

                });

                return;

            }

            // Itt a jelszó helyes formátumát ellenőrzi, miszerint legalább 10 karakter, maximum 30 lehet, tartalmazzon legalább

            // egy nagy betűt és legalább egy számot.

            // Mivel az ellenőrzések mind a jelszó formátumára vonatkozik, ezért a hiba üznenet mindegyikre ugyan az

            if (validationMethods.CheckPassword(registPassword) === false) {

                res.status(400).json({

                    error: "true",

                    message: "The password is in incorrect form!"

                });

                return;

            }

            // Ha minden ellenőrzésen átment akkor azt jelenti, hogy a felhasználót készek vagyunk elmenteni az adatbázisban

            // Magát a jelszót nem mentjük el egyenesen, hanem előtte a bcrypt segítségével titkosítjuk, az egyszerűség kedvéért a

            // bcrypt-methods.js fájlban megvan írva egy nyílvános metódus a titkosításhoz

            const hashedPassword = bcryptMethods.Hashing(registPassword);

            // A jelenlegi dátumot mentjük el, ezt egyenesen el lehet menteni a sequelize model DATEONLY típusába

            const date = new Date();

            // A felhasználó létrehozása és az adatbázisban elmentése egyben a megfelelő adatokkal,

            // majd a 201-es http kóddal megerősítjük, hogy sikeres volt a regisztráció

            const newUser = await User.create({

                username: registUsername,

                password: hashedPassword,

                email: registEmail,

                creation: date,

            });

            res.status(201).json({

                error: "false",

                message: "User succesfully created!",

                data: newUser,

            });

            return;

        }

        catch (error) {

            // Az egyetlen validáció, ami nem kézzel írtunk meg az az email ellenőrzése, és ugyanakkor csak az email címnél használunk

            // sequelize validációt, ezért, ha a hiba a sequelize validációhoz kötődik, biztos, hogy az email címmel volt probléma, ekkor

            // visszaküldjük a megfelelő hibakódot és választ

            if (error.name === "SequelizeValidationError") {

                res.status(400).json({

                    error: "true",

                    message: "The email is in incorrect form!"

                });

                return;

            }

Elmentjük a kérés body-ban kapott adatokat, majd ellenőrizzük, hogy ténylegesen megvan-e minden kért adat. Ezután megnézzük, hogy az adatbázisban nincs-e azonos nevű vagy email című felhasználó. Majd ellenőrizzük az adatok formátumát, ha ez is rendben volt, akkor titkosítjuk a jelszót. Végül létrehozzuk a felhasználót a megadott adatokkal és küldünk egy megerősítő választ. Az email cím formátumának ellenőrzését a Sequelize validátor végzi, ha az hibát dob, akkor tudjuk, hogy a hiba az emailcím formátumával van.

**review.controller.js magyarázata**

Ebben a fájlban kettő metódus található, mindegyik a bejelentkezett felhasználó kritika kezelésével kapcsolatos.

import Game from "../models/game.js";

import Rating from "../models/rating.js";

import Review from "../models/review.js";

import User from "../models/user.js";

import jweMethods from "../utilities/jwe.methods.js";

import validationMethods from "../utilities/validation.methods.js";

Importáljuk a szükséges modelleket és metódusokat.

Az első metódus a ReviewPutController, ami a kérés body-ban megadott adatok alapján hoz létre kritikát.

    // Felhasználó kritika írását kezeli

    ReviewPutController: async (req, res) => {

        try {

            // Lekérjük a bejelentkezett felhasználó azonosítóját, ha nincs, akkor hiba üzenetet küldünk

            const userId = await jweMethods.GetUserId(req);

            if (userId === undefined) {

                res.status(401).json({

                    error: true,

                    message: "The token is missing or faulty!"

                });

                return;

            }

            // Az URL-ben szereplő játék azonosított lementjük a megfelelő formátumban

            const gameId = Number(req.params.gameId?.trim());

            // Ha a játék azonosító hiányzik vagy nem egy szám, akkor egy hiba üzenetet küldünk vissza

            if (!gameId || isFinite(gameId) === false) {

                res.status(400).json({

                    error: true,

                    message: "There's no game id!"

                });

                return;

            }

            // Ellenőrizzük, hogy az URL-ben megadott számhoz ténylegesen tartozik egy játék, ha nem akkor egy hiba üzenetet küldünk

            const game = await Game.findOne({

                attributes: ["id"],

                where: {

                    id: gameId

                }

            });

            if (!game) {

                res.status(404).json({

                    error: true,

                    message: "There's no game with this id!"

                });

                return;

            }

            // Ellenőrizzük, hogy a felhasználó írt-e már egy kritikát a játékról, ha igen, akkor hiba üzenetet küldünk

            const reviewExist = await Review.findOne({

                attributes: ["id"],

                where: {

                    UserId: userId,

                    GameId: gameId

                }

            });

            if (reviewExist) {

                res.status(409).json({

                    error: true,

                    message: "The user already wrote a review!"

                });

                return;

            }

            // Ellenőrizzük, hogy a felhasználó értékelte-e a játékot már, ha nem, akkor hiba üzenetet küldünk

            const userRated = await Rating.findOne({

                where: {

                    UserId: userId,

                    GameId: gameId

                }

            });

            if (!userRated) {

                res.status(400).json({

                    error: true,

                    message: "The user didn't rate the game before!"

                });

                return;

            }

            // Konstansként lementjük a kritika címét és törzs szövegét

            const { title, content } = req.body;

            // Ha hiányzik a cím, akkor hiba üzenetet küldünk

            if (!title) {

                res.status(400).json({

                    error: true,

                    message: "Title is missing!"

                });

                return;

            }

            // Ha a cím kevesebb mint 5 karakter, akkor hiba üzenetet küldünk

            if (title.length < 5) {

                res.status(400).json({

                    error: true,

                    message: "Title is less than 5 character!"

                });

                return;

            }

            // Ha a cím több mint 100 karakter, akkor hiba üzenetet küldünk

            if (title.length > 100) {

                res.status(400).json({

                    error: true,

                    message: "Title is more than 100 character!"

                });

                return;

            }

            // Ha a törzs szöveg hiányzik, akkor hiba üzenetet küldünk

            if (!content) {

                res.status(400).json({

                    error: true,

                    message: "Content is missing!"

                });

                return;

            }

            // Ha a törzs szöveg tartalma kevesebb mint 50 karakter, akkor hiba üzenetet küldünk

            if (content.length < 50) {

                res.status(400).json({

                    error: true,

                    message: "Content is less than 50 character!"

                });

                return;

            }

            // Ha a törzs szöveg tartalma több mint 5000 karakter, akkor hiba üzenetet küldünk

            if (content.length > 5000) {

                res.status(400).json({

                    error: true,

                    message: "Content is more than 5000 character!"

                });

                return;

            }

            // Ha a cím vagy a törzs szöveg káromkodást tartalmaz, akkor hiba üzenetet küldünk

            if ((validationMethods.CheckProfanity(title) === true) || validationMethods.CheckProfanity(content) === true) {

                res.status(400).json({

                    error: true,

                    message: "The review contains profanity!"

                });

                return;

            }

            // Lementjük a jelenlegi dátumot, hogy felhasználjuk a kritika mentésére

            const currentDate = new Date();

            // Ha minden ellenőrzésen átment, akkor létrehozza a kritikát a megadott értékekkel

            await Review.create({

                title: title,

                content: content,

                date: currentDate,

                GameId: gameId,

                UserId: userId

            });

            // Visszaküldjük, hogy a metódus sikeresen lefutott

            res.status(201).json({

                error: false,

                message: "Review has been shared!"

            });

            return;

        }

Lekérjük a felhasználó azonosítóját, majd ellenőrizzük, hogy sikerült-e. Ezután a paraméterből lekérjük a játék azonosítóját, és ellenőrizzük, hogy van-e ilyen azonosítóval játék. Ezekután megnézzük, hogy a felhasználó értékelte a játékot, mert csak az írhat kritikát, aki értékelte az adott játékot. Utána lementjük a kérés body-ból a kritika címét és a tartalmát. Ellenőrizzük, hogy az adatokat megkaptuk-e és hogy a megfelelő hosszban vannak és nem tartalmaznak káromkodást. Végül elmentjük a jelenlegi dátumot és létrehozzuk a kritikát.

A ReviewDeleteController egy felhasználó kritikájának a törlését kezeli, azt hogy melyik kritikát a kell törölni, azt a kérés body-ban kell törölni.

    // A felhasználó kritika törlését kezeli

    ReviewDeleteController: async (req, res) => {

        try {

            // Lekérjük a bejelentkezett felhasználó azonosítóját, ha nincs, akkor hiba üzenetet küldünk

        const userId = await jweMethods.GetUserId(req);

        if (userId === undefined) {

            res.status(401).json({

                error: true,

                message: "The token is missing or faulty!"

            });

            return;

        }

        // Egy változóban elmentjük, hogy a felhasználónak van-e jogosultsága törölni a kritikát, ez alapvetően hamis,

        // a további ellenőrzések változtathatják az értékét

        let isUserAuthorized = false;

        // Megnézzük, hogy a felhasználó létezik-e az adatbázisban

        const isUserAdmin = await User.findOne({

            where: {

                id: userId

            }

        });

        // Ha a felhasználó létezik, akkor megnézzük, hogy van-e admin státusza, ha igen akkor jogosult a kritika törlésére

        if (isUserAdmin.admin === true) {

            isUserAuthorized = true

        }

        // A játék azonosított mentjük az URL-ből amegfelelő formátumra

        const gameId = Number(req.params.gameId?.trim());

        // Elelnőrizzük hogy ténylegesen megkaptuk a játék azonosított és hogy az szám-e

        if (!gameId || isFinite(gameId) === false) {

            res.status(400).json({

                error: true,

                message: "There's no game id!"

            });

            return;

        }

        // Az URL-ben lévő szám alapján megkeressük a játékot, ha nincs ehhez a számhoz tartozó játék, akkor hiba üzenetet küldünk

        const game = await Game.findOne({

            attributes: ["id"],

            where: {

                id: gameId

            }

        });

        if (!game) {

            res.status(404).json({

                error: true,

                message: "There's no game with this id!"

            });

            return;

        }

        // Lementjük az értékelés azonosítóját, amit a body-ban kellene megkapnunk

        const reviewId = req.body.reviewId;

        // Ha nem kaptuk meg az értékelés azonosítóját akkor hiba üzenetet adunk

        if (!reviewId || isFinite(reviewId) === false) {

            res.status(400).json({

                error: true,

                message: "Review id is missing!"

            });

            return;

        }

        // Ellenőrizzük, hogy a megadott azonosítóhoz ténylgesen tartozik-e egy kritika, ha nem akkor hiba üzenetet dobunk

        const reviewExist = await Review.findOne({

            where: {

                id: reviewId

            }

        });

        if (!reviewExist) {

            res.status(404).json({

                error: true,

                message: "There's no review with this id!"

            });

            return;

        }

        // Megnézzük, hogy a kritika írója megegyezik-e a törlést kezdeményező felhasználóval, ha igen akkor jogosult a törlésre

        if (reviewExist.UserId === userId) {

            isUserAuthorized = true;

        }

        // Ha a felhasználó jogosult a törlésre akkor kitöröljük a kritikát és egy megerősítő válasz adunk, ha nem jogosult rá,

        // akkor egy hiba üzenetet adunk rá

        if (isUserAuthorized === true) {

            await Review.destroy({

                where: {

                    id: reviewId

                }

            });

            res.status(200).json({

                error: false,

                message: "The review has been deleted!"

            });

            return;

        }

        else {

            res.status(401).json({

                error: true,

                message: "You are not authorized to do this!"

            });

            return;

        }

        }

Lekérjük a felhasználó azonosítóját és ellenőrizzük, hogy sikerült-e. Létrehozunk egy logikai változót, amiben lekövetjük, hogy a felhasználó jogosult-e a kritika törlésére. Lekérjük a felhasználó adatait az azonosító segítségével, és megnézzük, hogy admin-e, ha igen akkor jogosult a kritika törlésére és a logikai változót igazra állítjuk. Majd lekérjük a játék azonosítóját az URL paraméteréből, és ellenőrizzük, hogy van-e ilyen azonosítójú játék. Majd a kérés body-ból lekérjük a kritika azonosítóját és megnézzük, hogy van-e ilyen azonosítójú kritika. Ezután ellenőrizzük, hogy a felhasználó azonosítója megegyezik-e a kritika írójának azonosítójával, ha igen akkor jogosult a kritika törlésére. Végül ellenőrizzük a logikai értéket, és ha igaz, akkor töröljük a kritikát.

### Models fájlok

Ebben a mappában az adatbázis tábla modelljei találhatóak. Ezeknek a felépítése követi az adatbázis dokumentációban meghatározott táblákét, ezért itt csak egy modellt nézünk meg hogy hogyan lett létrehozva általánosságban. Ehhez az actor.js fájlt nézzük meg.

import sequelize from "../config.js";

import { Model, DataTypes } from "sequelize";

class Actor extends Model {};

Actor.init ({

            id: {

                type: DataTypes.INTEGER.UNSIGNED,

                allowNull: false,

                autoIncrement: true,

                primaryKey: true,

            },

            firstName: {

                type: DataTypes.STRING(100),

                comment: "A színész keresztneve"

            },

            lastName: {

                type: DataTypes.STRING(100),

                comment: "A színész vezetékneve",

            },

            profilePicture: {

                type: DataTypes.STRING,

                comment: "A színészről tárolt kép elérési útvonala"

            },

        },

        {

            sequelize,

            modelName: "Actor",

            comment: "A színészek adatait tárolja"

        });

export default Actor;

Az elején importáljuk az config.js-ben meghatározott Sequlize adatbázis konstanst, majd a DataTypes-t amivel egy tábla adattípusát tudjuk megadni, és végül a Model osztályt. Ezzel a megoldással egy Model típusú osztályt hozunk létre az általunk megadott névvel, ezek után az első kapcsos zárójelben megadjuk a táblák neveit, majd azok típusát, illetve egyéb megkötéseket. A második zárójelben megadjuk, hogy kapcsolja a sequlize nevű adatbázis konstansukhoz, majd a biztonság kedvéért itt is megadjuk a modell nevét. A legvégén exportáljuk a modellt.

### Router fájlok

Ezeket a fájlokat a routes mappában tároljuk, ezekben írjuk meg hogy milyen címen milyen metódusnál melyik controller fájl fusson le. Ezek felépítése nagyon hasonló, ezért itt csak egy esetet nézünk meg, a favourites.router.js fájlt.

import { Router } from "express";

import favouritesController from "../controllers/favourites.controller.js";

const FavouritesRouter = new Router();

FavouritesRouter.get("/favourites", favouritesController.FavouritesGetController);

FavouritesRouter.delete("/favourites", favouritesController.FavouritesDeleteController);

export default FavouritesRouter;

Először is importáljuk a Router-t az express-ből, majd a router fájlal megegyező kezdő nevű controller-t. Létrehozunk egy konstans olyan névvel, amivel azonosítani tudjuk és létrehozzuk a Router segítségével. Ezekután megadjuk, hogy a Router konstansunknak, hogy milyen metódus esetét kezelje, majd a zárójelben karakterláncként adjuk meg a címet, ehhez nem kell tenni a htttp://localhost:3000 részt, csak az ezutáni részt, majd ezután adjuk meg a controller fájlt, és azt, hogy melyik metódusát rendeljük hozzá. Ezt minden tervezett útvonalon megcsináljuk. Végül exportáljuk a konstansunkat.

### Az utilities mappa

Ez mappa három fájlt tartalmaz, amelyek egy bizonyos csomaghoz specifikus metódusokat tartalmaz, amiket a controller fájloknál használunk fel, illetve van egy middleware amit az app.js-nél használunk.

**bcrypt.methods.js fájl**

import bcrypt from "bcrypt";

const salts = 10;

// A jelszó titkosítás és összehasonlítást külön fájlban definiáljuk majd exportáljuk

export default {

    Hashing: (password) => {

        return bcrypt.hashSync(password, salts);

    },

    Comparing: (plainPassword, hashedPassword) => {

        return bcrypt.compareSync(plainPassword, hashedPassword);

    },

};

Ebben a fájlban a bcrypt csomaghoz tartozó metódusokat kezeljük. Először a salts-ban megadjuk a titkosításhoz szükséges általunk meghatározott számhoz. Majd export default-ban beírva létrehozunk két metódust. A Hashing paraméterében egy jelszót várunk el, eredményül a bcrypt metódussal titkosított jelszót küldjük vissza. A Comparing metódus paraméterében egy egyszerű jelszót és egy titkosított jelszót várunk el. Összehasonlítjuk, hogy az egyszerű jelszó megegyezik-e a titkosított jelszó eredeti formájával, eredményül egy logikai értéket küldünk vissza, ahol az igaz azt jelenti, hogy megegyezik, a hamis, hogy nem.

**jwe.methods.js fájl**

Ebben a fájlban a jose könyvtár token kezeléseivel kapcsolatos három metódus található.

import { compactDecrypt, EncryptJWT} from "jose";

import crypto from "crypto";

import User from "../models/user.js";

// A token titkosításához használt kulcs egy random szám, Uint8Array típusú, mert a titkosításhoz ilyen típusú objektum kell

// Ez a kulcs minden szerver indításnál új, amely tovább növeli a biztonságot

const securekey = new Uint8Array(32);

crypto.randomFillSync(securekey);

Beimportáljuk a jose könyvtárból szükséges funkciókat, majd egy random érték generálásához szükséges modult és a User modellt. Ezután minden metódust export default-ban írunk meg.

CreatingToken: async (loginId, loginUsername, loginEmail) => {

        const payload = {

            id: loginId,

            username: loginUsername,

            email: loginEmail,

        };

        // Itt jön létre a token, a payload a felhasználó adatokat használja, 1 óráig érvényes a token, titkosított fejléce van,

        // a titkosítás a fájl elején létrehozott kulccsal történik

        // Végül visszaadja a token-t

        const token = await new EncryptJWT(payload)

            .setExpirationTime("1 hour")

            .setProtectedHeader({ alg: "dir", enc: "A256GCM"})

            .encrypt(securekey)

        return token;

    },

Ebben a metódusban egy felhasználó token-jét hozzuk létre, ehhez szükséges az azonosítója, a felhasználóneve és email címe. Ezekből létrehozunk egy objektumot payload néven. Ezután létrehozzuk a tokent-t egy konstansba a jose könyvtár egy funkciójával, aminek beállítjuk a titkosításait és azt hogy csak egy óráig legyen érvényes, ez után elküldjük a token-t.

// Ez a metódus egy middleware-ként szolgál, amely a felhasználónak új token-t ad, ha az előző még aktív, de hamarosan lejár

    ExntendingToken: async (req, res, next) => {

        // A következő sorok a biztonság kedvéért try catch párban vannak írva hogy a felmerülő hibákat kezelni tudjuk

        try {

            // A tokent lekérdezzük, majd visszafejtjük és lemenetjük a nyers adatokat

            const currentToken = req.headers['authorization']?.split(' ')[1];

            // Ha nincs token, azt jelenti, hogy felhasználó nincs bejelentkezve, ilyenkor nem tudunk token-t cserélni

            if (!currentToken) {

                return next();

            }

            // Megnézzük, hogy a token érvényes-e, ha nem, akkor nem cserélünk token-t

            let validToken = true;

            const decodedToken = await compactDecrypt(currentToken, securekey)

                .catch(() => {

                    validToken = false;

                });

            if (validToken === false) {

                return next();

            }

            const currentPayload = JSON.parse(decodedToken.plaintext.toString("utf8"));

            // A token lejárati idejét és a jelenlegi dátumut lementjük egy azonos formátumban, majd kiszámoljuk, hogy mennyi idő van hátra

            // a token élettartalmából

            const expire = currentPayload.exp \* 1000;

            const currentDate = new Date();

            const timeLeft = expire - currentDate;

            // Ha a maradék idő kevesebb mint 30 perc akkor a felhasználónak adunk egy új token-t,

            // majd tovább lépünk, más esetben visszaküldjük, hogy még nincs szükség új tokenre

            if (timeLeft < 1800000) {

                const newToken = await new EncryptJWT(currentPayload)

                    .setExpirationTime("1 hour")

                    .setProtectedHeader({ alg: "dir", enc: "A256GCM"})

                    .encrypt(securekey);

                res.setHeader("Authorization", `Bearer ${newToken}`);

            }

            return next();

        }

        catch (error) {

            console.log(error);

            return res.status(500).json({

                error: true,

                message: "Something went wrong when checking the validity of the token!",

            });

        }

    },

Ebben a middleware-ben a token hosszabbítását érjük el. Ehhez lekérjük a meglévő token-t, ha nincs akkor tovább engedjük. Visszafejtjük a token-t és kinyerjük belőle az adatokat. majd megnézzük, hogy mennyi ideje van hátra, ha kevesebb mint 30 perc, akkor a token-ből szerzett adatok segítségével új token-t hozunk létre, amit az Authorization fejlécbe teszünk és hozzá csatoljuk azt ezt következő válaszhoz.

// Ez a metódus a token-ből kinyeri a felhasználó azonosítóját

    GetUserId: async (req) => {

        // A token lekérjük, de utána ellenőrizzük, hogy van e tényleges token, ha nincs akkor visszaküldünk egy undefined

        // eredményt, amit a fő program majd kezel

        const token = req.headers['authorization']?.split(' ')[1];

        if (!token) {

            return undefined;

        }

        let id = 0;

        // Dekódoljuk a token-t majd a megfelelő formátumba hozzuk

        await compactDecrypt(token, securekey)

            .then(async(decodedToken) => {

                const currentPayload = JSON.parse(decodedToken.plaintext.toString("utf8"));

                // Ellenőrizzük, hogy az adatbázisban van ilyen felhasználó

                const isInTheDatabase = await User.findOne({

                    attributes: ["id", "username"],

                    where: {

                        id: currentPayload.id,

                        username: currentPayload.username

                    }

                });

                // Ha nincs, akkor a undefined értéket küldünk vissza

                if (!isInTheDatabase) {

                    id = undefined

                }

                // Ha az ellenőrzésen átment, akkor elmentjük a felhasználó azonosítóját

                id = currentPayload.id;

            })

            .catch((error) => {

                console.log(error);

            });

        // Ha a token visszafejtése nem sikerült, az azért van, mert a token lejárt, vagy egyéb okok miatt már nem aktív,

        // ilyenkor szintén undefined-ot küldünk vissza, ha érvényes a token, akkor visszaküldjük az azonosítót

        if (id === 0) {

            return undefined;

        }

        else {

            return id;

        }

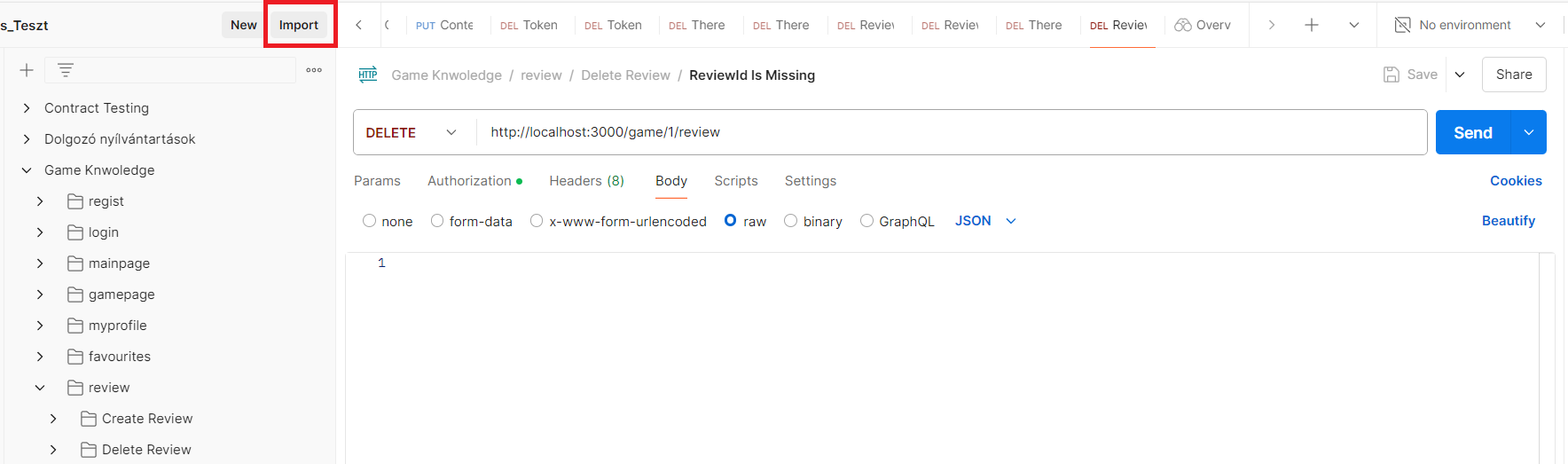
    }

Ebben a metódusban a felhasználó azonosítóját nyerjük ki a token-ből. Ha bármi hiba történik közben akkor egy undefined értéket küldünk vissza, más esetben a token-ből származó azonosított. Futás során teszteljük hogy tényleg létezik-e ilyen azonosítójú felhasználó az adatbázisban.

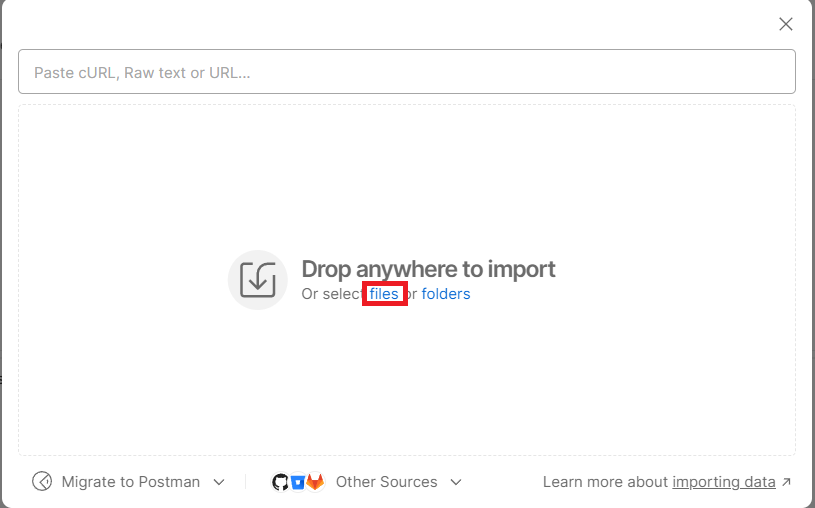
## Backend teszt Postman-nel

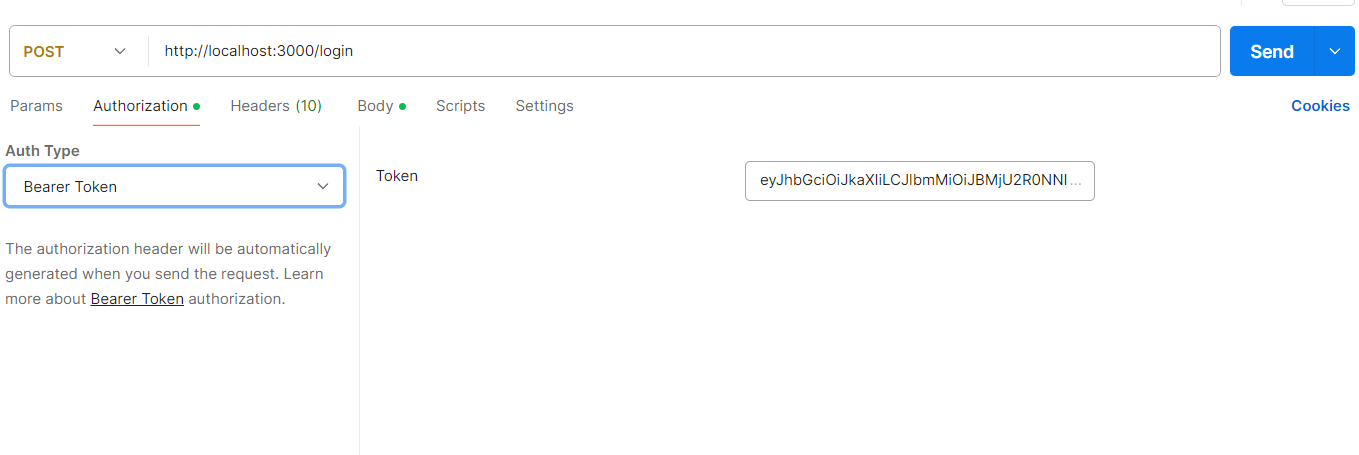
Az API teszteket a Postman alkalmazással tesztelhetjük, így importálhatjuk a ’Game Knwoledge.postman\_collection.json’ fájlt a Postman-be:

Nyissuk meg a Postman alkalmazást és a bal felső sarokban kattintsunk az Import gombra.



Ez bedob egy új ablakot, ahol kattintsunk kékkel írt files-ra majd válasszuk ki a ’Game Knwoledge.postman\_collection.json’.



Ezzel be importáltuk a teszteket, itt hét mappát találhatunk, a mappák neve utal a hozzá tartozó backend fájlra. Ahhoz, hogy biztosítsuk, hogy az elvárt eredményeket kapjuk érdemes egy frissen importált adatbázist használni, és egy tesztnél nincs meghatározva, hogy hányszor futtassuk le, akkor csak egyszer tegyük meg, mert megtörténhet, hogy adatváltoztatások vagy törlések miatt bizonyos funkciók már nem úgy működnek ahogy elvártuk. Bizonyos funkciókhoz egy bejelentkezett felhasználóra van szükség, ehhez a ’login’ mappán belül a ’Login’ mappában található három sikeres bejelentkezés eset: ’Successfull Login With Admin User’,’ Successfull Login With General User’ és ’Successfull Login For Test User’ (de ez esetén később megváltoztatjuk az adatait, azután nem fog működni ez a bejelentkezés). Ezek a sikeres bejelentkezések biztosítanak egy token-t a válasz body-ban, illetve a fejléceknél az ’Authorization’ fejlécében. A token-t tartalmazza a felhasználó adatait és ha egy Postman kérés Authorization fülén, itt az Auth Type-on kiválasztjuk a Bearer Token opciót és a token-t beillesztjük a megadott mezőbe akkor hozzáférhetünk azokhoz a funkciókhoz, amelyekhez be kell jelentkezni előtte.

Fontos megjegyezni, hogy egy token élettartalma egy óra, ezután a token már nem érvényes, tartsuk észbe, ha nem várt hibát kapunk egy kérés során. A backend rendelkezik egy token újító funkcióval, ha a token élettartalma kevesebb mint harminc perc, akkor a ’login’ és ’regist’ mappán belüli kérések kivételével, mindegyik kérésre adhat egy új token-t, ami a válasz fejléc Authorization-nél található. Ha valamelyik kérés során olyan hiba történik amire nem számítottunk akkor HTTP 500-as kódú hibát kapunk.

### ’regist’ teszt mappa

A ’regist’ mappában kettő további mappa található, mindkettő a kéréseket a [HTTP://localhost:3000/registration](http://localhost:3000/registration) küldjük.

A Registration mappában a regisztrációs tesztek érhetőek el POST metódussal.

* Correct/Already Exist – Ez a teszt sikeresen létrehoz egy új felhasználót az adatbázisban, hisz minden adat a megfelelő formátumban van, erre így egy 201-es HTTP kódot kapunk, a válaszban láthatjuk, hogy a jelszót sikeresen titkosította. Ha még egyszer lefuttatjuk akkor hibát kapunk hisz ezekkel az adatokkal már van felhasználó az adatbázisban, a HTTP kód ez esetben 409.
* New User for Test – Ez a teszt sikeresen létrehoz egy új felhasználót az adatbázisban, hisz minden adat a megfelelő formátumban van, erre így egy 201-es HTTP kódot kapunk. Ha még egyszer lefuttatjuk akkor hibát kapunk hisz ezekkel az adatokkal már van felhasználó az adatbázisban, a HTTP kód ez esetben 409. A célunk ezzel a felhasználóval, hogy később felhasználjuk az adat változtatás funkcióhoz.
* Empty Body – Ez a teszt egy üres body-t küldd a backend-nek, mivel az adatok hiányoznak, ezért HTTP 400-as kódú hibát kapunk vissza.
* Missing Username – Ez a teszt egy alapvetően helyes formátumú regisztrációt küldene el, de hiányzik a felhasználónév, ezért HTTP 400-as kódú hibát kapunk vissza.
* Missing Email - Ez a teszt egy alapvetően helyes formátumú regisztrációt küldene el, de hiányzik az email cím, ezért HTTP 400-as kódú hibát kapunk vissza.
* Missing Password - Ez a teszt egy alapvetően helyes formátumú regisztrációt küldene el, de hiányzik a jelszó, ezért HTTP 400-as kódú hibát kapunk vissza.
* Missing Password Confirmation - Ez a teszt egy alapvetően helyes formátumú regisztrációt küldene el, de hiányzik a jelszó megerősítő, ezért HTTP 400-as kódú hibát kapunk vissza.
* Username Too Long – Ebben a tesztben a felhasználónév túllépi a maximum 30 karakter megengedett hosszt, ezért HTTP 400-as kódú hibát kapunk vissza.
* Username Too Short – Ebben a tesztben a felhasználónév nem éri el a minimum 5 karater hosszt, ezért HTTP 400-as kódú hibát kapunk vissza.
* Username Contains Space – Ebben a tesztben a felhasználónév szóközt tartalmaz, amit nem engedünk meg a felhasználónévben, ezért HTTP 400-as kódú hibát kapunk vissza.
* Username Is a Profanity – Ebben a tesztben a felhasználónév egy angol káromkodást, ezt nem engedjük a felhasználónévnek, ezért HTTP 400-as kódú hibát kapunk vissza.
* Email is Missing: @ - Ebben a tesztben az email címből hiányzik a @ jel, ami azt jelenti, hogy ez nem lehet egy hiteles email cím, ezért HTTP 400-as kódú hibát kapunk vissza.
* Email Missing: dot . – Ebben a tesztben az email címből hiányzik a @ jel utáni pont, ez azt jelenti, hogy ez nem lehet egy hiteles email cím, ezért HTTP 400-as kódú hibát kapunk vissza.
* Passwords Don't Match – Ebben a tesztben a jelszó és a jelszó megerősítő nem egyezik, ezért HTTP 400-as kódú hibát kapunk vissza.
* Password Too Short – Ebben a tesztben a jelszó nem éri el a minimum 10 karaktert, ezért HTTP 400-as kódú hibát kapunk vissza.
* Password Too Long – Ebben a tesztben a jelszó meghaladja a 30 karaktert, ezért HTTP 400-as kódú hibát kapunk vissza.
* No Uppercase In The Password – Ebben a tesztben a jelszó nem tartalmaz nagy betűs karaktert, ezért HTTP 400-as kódú hibát kapunk vissza.
* No Number In The Password – Ebben a tesztben a jelszó nem tartalmaz számot, ezért HTTP 400-as kódú hibát kapunk vissza.
* Empty Datas – Ebben a tesztben az adatok értéke üres, ezért HTTP 400-as kódú hibát kapunk vissza.
* Username Is Empty – Ebben a tesztben a felhasználónév értéke üres, ezért HTTP 400-as kódú hibát kapunk vissza.
* Email Is Empty - Ebben a tesztben az email cím értéke üres, ezért HTTP 400-as kódú hibát kapunk vissza.
* Password is Empty - Ebben a tesztben a jelszó értéke üres, ezért HTTP 400-as kódú hibát kapunk vissza.
* Password Confirmation Is Empty – Ebben a tesztben a jelszó megerősítő értéke üres, ezért HTTP 400-as kódú hibát kapunk vissza.
* Username Is Null - Ebben a tesztben a felhasználónév értéke null, ezért HTTP 400-as kódú hibát kapunk vissza.
* Email Is Null - Ebben a tesztben az email cím értéke null, ezért HTTP 400-as kódú hibát kapunk vissza.
* Password Is Null – Ebben a tesztben a jelszó értéke null, ezért HTTP 400-as kódú hibát kapunk vissza.
* Password Confirmation Is Null - Ebben a tesztben a jelszó megerősítő értéke null, ezért HTTP 400-as kódú hibát kapunk vissza.

A ’regist’ mappában a Method Not Allowed mappán belül teszteljük, hogy ha olyan metódusokkal próbáljuk elérni az adott címet, amit nem támogat.

* PUT – Ebben a tesztben PUT metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* PATCH – Ebben a tesztben PATCH metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* DELETE – Ebben a tesztben DELETE metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* GET – Ebben a tesztben GET metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.

### ’login’ teszt mappa

Ebben a mappában további két mappa található, mindkettőben a kéréseket a [HTTP://localhost:3000/login](http://localhost:3000/login) címre küldi.

A ’Login’ mappában a bejelentkezéshez tartozó tesztek tartózkodnak, amiket POST metódussal érünk el:

* Successfull Login With Admin User – Ebben a tesztben egy sikeres bejelentkezés található egy admin jogosultsággal rendelkező felhasználóval, a válaszban láthatjuk, hogy az ’isAdmin’ értéke igaz. A válaszban továbbá láthatjuk, hogy megkaptuk a felhasználó token-t, amit a felhasználó azonosításához használunk. Ez a token megtalálható a válasz fejléc ’Authorization’ értékénél. A sikeres választ HTTP 200 kóddal kapjuk meg.
* Successfull Login With General User - Ebben a tesztben egy sikeres bejelentkezés található egy átlagos jogosultsággal rendelkező felhasználóval, a válaszban láthatjuk, hogy az ’isAdmin’ értéke hamis. A válaszban továbbá láthatjuk, hogy megkaptuk a felhasználó token-t, amit a felhasználó azonosításához használunk. Ez a token megtalálható a válasz fejléc ’Authorization’ értékénél. A sikeres választ HTTP 200 kóddal kapjuk meg.
* Successfull Login For Test User - Ebben a tesztben egy sikeres bejelentkezés található egy admin jogosultsággal rendelkező felhasználóval, a válaszban láthatjuk, hogy az ’isAdmin’ értéke igaz. A válaszban továbbá láthatjuk, hogy megkaptuk a felhasználó token-t, amit a felhasználó azonosításához használunk. Ez a token megtalálható a válasz fejléc ’Authorization’ értékénél. A sikeres választ HTTP 200 kóddal kapjuk meg, de csak akkor lesz sikeres, ha a regisztrációs tesztnél futtattuk a New User for Test nevű tesztet.
* Username Is Missing – Ebben a tesztben a felhasználónév hiányzik, ezért HTTP 400-as kódú hibát kapunk.
* Password Is Missing - Ebben a tesztben a jelszó hiányzik, ezért HTTP 400-as kódú hibát kapunk.
* Username Is Empty - Ebben a tesztben a felhasználónév üres, ezért HTTP 400-as kódú hibát kapunk.
* Password Is Empty - Ebben a tesztben a jelszó üres, ezért HTTP 400-as kódú hibát kapunk.
* Username Is Null - Ebben a tesztben a felhasználónév NULL, ezért HTTP 400-as kódú hibát kapunk.
* Password Is Null - Ebben a tesztben a felhasználónév NULL, ezért HTTP 400-as kódú hibát kapunk.
* Username Is Incorrect - Ebben a tesztben ilyen felhasználónévvel nincs az adatbázisban felhasználó, ezért HTTP 400-as kódú hibát kapunk.
* Password Is Incorrect - Ebben a tesztben van ilyen felhasználónévvel felhasználó az adatbázisban, de a megadott jelszó nem egyezik az adatbázisban lévővel, ezért HTTP 400-as kódú hibát kapunk.

A login mappán belül a Method Not Allowed mappában találhatunk olyan teszteket, amikkel a nem támogatott metódusokat teszteljük a megadott címen.

* GET – Ebben a tesztben GET metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* PUT – Ebben a tesztben PUT metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* PATCH – Ebben a tesztben PATCH metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* DELETE – Ebben a tesztben DELETE metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.

### ’mainpage’ teszt mappa

A ’mainpage’ mappán belül további három mappa található, amelyekben a tesztek a [HTTP://localhost:3000/](http://localhost:3000/) címre kerülnek küldésre.

A ’Mainpage Fetch’ mappában egy darab teszt található, amit GET metódussal tudunk elérni.

* Fetching Games On Mainpage – Ez a teszt azokat az adatokat kéri le, amelyeket a weboldal fő oldalára szeretnénk megjeleníteni, a sikeres lekérdezést HTTP 200-as kóddal kapjuk meg.

A ’Search’ mappában a weboldal keresőmezőjére tartozó tesztek tartoznak, amiket PUT metódussal érünk el. A szöveget, amelyet a kereséshez használunk a kérés body-ban kapjuk meg. A keresést a játék neve és alternatív neve alapján is keres.

* Valid Search – Ebben a tesztben egy sikeres keresés eredményét látjuk, ahol azokat a játékokat adja a válaszban eredményül, ami tartalmazza a body-ban lévő search eredményének a szövegét, kis- és nagybetűtől függetlenül. Az eredményben a játékoknál azokat az adatokat kérjük le, amelyeket a keresés eredményénél szeretnénk megjeleníteni. A válasz HTTP kódja 200.
* Text Is Missing – Ebben a tesztben hiányzik a keresés szövege, erre nem adunk hiba kódot, de a válaszban tudatjuk, hogy ez egy üres keresés. A válasz HTTP kódja 200.
* No Text – Ebben a tesztben a keresés szövege üres, erre nem adunk hiba kódot, de a válaszban tudatjuk, hogy ez egy üres keresés. A válasz HTTP kódja 200.

A ’Method Not Allowed’ mappa olyan teszteket tartalmaz, amelyek a megadott címre nem támogatott metódussal küld kéréseket.

* POST - Ebben a tesztben POST metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* PATCH - Ebben a tesztben PATCH metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* DELETE - Ebben a tesztben DELETE metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.

### ’gamepage’ teszt mappa

A ’gamepage’ mappán belül négy további mappa található, amelyekben a minden teszt a [HTTP://localhost:3000/game/:gameId](http://localhost:3000/game/:gameId) címre kerül elküldésre. Ezek a tesztek a paraméterben megadott számhoz tartozó játékra specifikusak.

A ’Gamepage Fetching’ mappában egy konkrét játék adatainak a lekérdezésére való teszteket tartalmazza a GET metódussal.

Successfull Fetch – Ez a teszt lekérdezi a paraméterben megadott számhoz megegyező azonosítóval rendelkező játét adatait, amelyet a weblapon szeretnénk megjeleníteni. A sikeres lekérdezést HTTP 200-as kóddal jelezzük és válasz tartalmazza a játék adatait.

Game With This Id Does Not Exist – Ebben a tesztben a paraméterben megadott számhoz nem tartozik játék, ezért egy HTTP 404-es kódú hibát kapunk.

A ’Rating’ mappán belüli tesztek a paraméterben megadott számhoz tartozó játék értékelés létrehozását teszteli egy bejelentkezett felhasználó által PUT metódussal. A felhasználó értékelést a válasz body-ban kapjuk az ’isPositive’-ból, ami egy logikai értéknek kell lennie.

* Successfull Positive Rating / Conflict – Ennél a tesztnél használjunk egy általános felhasználó token-jét, és ha egyszer lefuttatjuk akkor egy HTTP 201 kódú választ kapunk, hogy a pozitív értékelést sikeresen lementettük. Ha még egyszer lefuttatjuk akkor egy HTTP 409-es kódú hibát kapunk, hogy a felhasználó már értékelte a játékot.
* Successfull Negative Rating / Conflict - Ennél a tesztnél használjunk egy általános felhasználó token-jét, és ha egyszer lefuttatjuk akkor egy HTTP 201 kódú választ kapunk, hogy a negatív értékelést sikeresen lementettük. Ha még egyszer lefuttatjuk akkor egy HTTP 409-es kódú hibát kapunk, hogy a felhasználó már értékelte a játékot.
* The Game's Not Released Yet - Ennél a tesztnél használjunk egy általános felhasználó token-jét, és ha lefuttatjuk akkor egy HTTP 401-es kódú hibát kapunk, mert olyan játékot próbálunk értékelni, ami még nem jelent meg.
* No Token Provided – Ennél a tesztnél ne használjunk token-t, ha lefuttatjuk a tesztet akkor egy HTTP 401-es kódú hibát kapunk, mert hiányzik a token.
* Faulty Token – Ennél a tesztnél, mér meg van adva egy hibás token, ha lefuttatjuk a tesztet akkor egy HTTP 401-es kódú hibát kapunk, mert a token hibás.
* No Rating Value – Ennél használjunk egy általános felhasználó token-jét, ha lefuttatjuk a tesztet akkor egy HTTP 400-es kódú hibát kapunk, mert a kérés body-ban az értékelés értéke üres.
* Missing Value - Ennél használjunk egy általános felhasználó token-jét, ha lefuttatjuk a tesztet akkor egy HTTP 400-es kódú hibát kapunk, mert a kérés body-ból hiányzik az értékelés.
* Rating's Value Is Not a Boolean - Ennél használjunk egy általános felhasználó token-jét, ha lefuttatjuk a tesztet akkor egy HTTP 400-es kódú hibát kapunk, mert a kérés body-ban nem egy logikai értéket kaptunk.
* Game With This Id Does Not Exist - Ennél a használjunk egy általános felhasználó token-jét, ha lefuttatjuk a tesztet akkor egy HTTP 404-es kódú hibát kapunk, mert a paraméterben lévő számhoz nem tartozik játék.

Az ’Add To Favourites’ mappában lévő tesztek egy bejelentkezett felhasználó a paraméterben megadott számhoz tartozó játék kedvencekbe tételét ellenőrzi.

* Successfully Added To Favourites / Conflict - Ennél használjunk egy általános felhasználó token-jét, ha egyszer lefuttatjuk a tesztet akkor egy HTTP 201-es kódú választ kapunk, hogy a játékot sikeresen lementettük a kedvencekbe. Ha még egyszer lefuttatjuk a tesztet, akkor egy HTTP 409-es kódú hibát kapunk, miszerint a felhasználó már a kedvencekbe tette a játékot.
* Game Does Not Exist - Ennél használjunk egy általános felhasználó token-jét, ha lefuttatjuk akkor egy 404-es kódú hibát kapunk, mert a paraméterben olyan szám van megadva, amihez nem tartozik játék.
* No Token Provided – Ennél a tesztnél ne használjunk tokent, ha lefuttatjuk a tesztet, akkor egy HTTP 401-es kódú hibát kapunk, mert a hiányzik a token.
* Faulty Token – Ennél a tesztnél már megvan adva egy hibás token, ha ha lefuttatjuk a tesztet, akkor egy HTTP 401-es kódú hibát kapunk, mert a token hibás.

A ’Method Not Allowed’ mappában lévő tesztesetekkel teszteljük, hogy ha nem támogatott metódusokkal próbáljuk elérni a megadott címet.

PATCH - Ebben a tesztben PATCH metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.

DELETE - Ebben a tesztben DELETE metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.

### ’myproflie’ teszt mappa

A ’myprofile’ mappában négy további mappa található. Ezek a tesztek egy bejelentkezett felhasználó adatainak lekérését, illetve módosítását kezeli. A kéréseket a HTTP://localhost:3000/myprofile címre küldjük. A felhasználót a token alapján azonosítjuk.

A ’Fetch User Data’ mappában lévő tesztesetek a bejelentkezett felhasználó adatainak lekérését teszteli GET metódussal.

* Successfull Fetch – Ehhez használjuk ’Successfull Login For Test User’-ban lévő felhasználó token-jét, ha lefuttatjuk akkor sikeres választ kapunk HTTP 200-as kóddal, ami tartalmazza a felhasználó adatait.
* No Token Provided – Ehhez ne használjunk token-t, ha így lefuttatjuk a tesztet, akkor HTTP 401-es kódú hibát kapunk, mert nincs token megadva.
* Faulty Token – Ennél a teszt esetnél már meg van adva hibás, ha így lefuttatjuk a tesztet, akkor HTTP 401-es kódú hibát kapunk, mert a megadott token hibás.

A ’Change User Data’ mappában a bejelentkezett felhasználó adatainak a változásaival kapcsolatos tesztek találhatóak. Minden ebben a mappában lévő teszthez használjuk a ’Successfull Login For Test User’ tesztből származó token-t. A címet POST metódussal érjük el. Jegyezzük meg hogy ha sikeresen megváltoztatjuk a felhasználó nevet vagy a jelszót, akkor a bejelentkezéshez használt teszt már nem fog működni a jelenlegi formájában.

* Username Is a Profanity – Ennél a tesztnél a felhasználónevet egy angol káromkodásra próbáljuk megváltoztatni, ezt nem engedélyezzük, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Username Is Too Short – Ennél a tesztnél a megadott felhasználónév nem éri el az 5 karakter hosszúságot, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Username Is Too Long - Ennél a tesztnél a megadott felhasználónév túllépi a 30 karakter hosszúságot, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Username Contains a Space - Ennél a tesztnél a felhasználónévben szóköz található, ezt nem engedélyezzük, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Username Is Same as Before – Ennél a tesztnél a megadott felhasználónév megegyezik az eredeti névvel, mivel ez egy felesleges változtatás lenne, ezért egy HTTP 409-as kódú hiba üzenetet küldünk.
* Username Is Already Taken – Ennél a tesztnél a megadott felhasználónév megegyezik az adatbázisban egy meglévő felhasználó nevével, mivel nem engedjük, hogy két ugyanolyan nevű felhasználó legyen, ezért egy HTTP 409-as kódú hiba üzenetet küldünk.
* Email Is Missing: @ - Ennél a tesztnél a megadott email címből hiányzik a @ jel, ami azt jelenti, hogy ez egy nem hiteles email cím, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Email Is Missing: Dot . - Ennél a tesztnél a megadott email címből hiányzik a @ jel utáni pont, ami azt jelenti, hogy ez egy nem hiteles email cím, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Email Is The Same As Before - Ennél a tesztnél a megadott email cím megegyezik az eredeti email címmel, mivel ez egy felesleges változtatás lenne, ezért egy HTTP 409-as kódú hiba üzenetet küldünk.
* Email Is Already Taken - Ennél a tesztnél a megadott email cím megegyezik az adatbázisban egy meglévő felhasználó email címével, mivel nem engedjük, hogy két ugyanolyan email című felhasználó legyen, ezért egy HTTP 409-as kódú hiba üzenetet küldünk.
* Passwords Don't Match – Ennél a tesztnél a megadott jelszó és jelszó megerősítés nem egyezik, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Password Same As Before – Ennél a tesztnél a megadott jelszó megegyezik az eredeti jelszóval, mivel ez egy felesleges változtatás lenne, ezért egy HTTP 409-as kódú hiba üzenetet küldünk.
* Password Does Not Contain a Number – Ennél a tesztnél a megadott jelszóban nincs szám, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Password Does Not Contain Uppercase Letter - Ennél a tesztnél a megadott jelszóban nincs nagybetűs karakter, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Password Is Too Short - Ennél a tesztnél a megadott jelszó nem éri el a minimum 10 karakter hosszt, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Password Is Too Long - Ennél a tesztnél a megadott jelszó túllépi a maximum 30 karakter hosszt, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Password's Missing Confirmation – Ennél a tesztnél meg van adva a jelszó, de hiányzik a megerősítés, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Only Password Confirmation Given - Ennél a tesztnél nincs megadva jelszó, de a megerősítés meg van adva, ezért egy HTTP 400-as kódú hiba üzenetet küldünk.
* Successfull Username Change – Ennél a tesztnél sikeresen megváltoztatjuk a felhasználó nevét, erre egy HTTP 201-es kódú választ kapunk.
* Successfull Email Change Copy – Ennél a tesztnél sikeresen megváltoztatjuk a felhasználó email címét, erre egy HTTP 201-es kódú választ kapunk.
* Successfull Password Change - Ennél a tesztnél sikeresen megváltoztatjuk a felhasználó jelszavát, erre egy HTTP 201-es kódú választ kapunk.
* Successfull Change Every Data - Ennél a tesztnél sikeresen megváltoztatjuk a felhasználó összes adatát egyszerre, erre egy HTTP 201-es kódú választ kapunk.

A ’Delete User’ mappában a felhasználó törlésével kapcsolatos tesztek találhatóak, illetve található egy bejelentkezést a ’Successfull Login For Test User’-ból lévő felhasználóhoz, aminél lefuttattuk a ’Successfull Change Every Data’ tesztet, ezt továbbra is a POST metódussal érjük el a [HTTP://localhost:3000/login](http://localhost:3000/login) címen, ezt leszámítva a további teszteket a [HTTP://localhost:3000/myprofile](http://localhost:3000/myprofile) címen érjük el DELETE metódussal.

* Login For Changed Test User – Ennél a tesztnél bejelentkezünk a teszt felhasználóval, hogy a további tesztet az innen szerzett token-nel hajtsuk végre.
* Successfull Deleting User / User Doesn't Exist – Ha megadtuk a teszt felhasználó token-jét, és egyszer lefuttattjuk, akkor a felhasználó törlésre kerül az adatbázisból, erre HTTP 200-as kódú választ kapunk. Ha még egyszer lefuttatjuk akkor HTTP 404-es kódú hibát kapunk, mert a felhasználó már nem létezik.
* Faulty Token – Ennél a tesztnél már megvan adva egy hibás token, ha lefuttatjuk akkor HTTP 401-es kódú hibát kapunk, mert a token hibás.
* No Token Provided - Ennél a tesztnél nincs megadva token, ha lefuttatjuk akkor HTTP 401-es kódú hibát kapunk, mert a token hiányzik.

A ’Method Not Allowed’ mappában lévő tesztekkel a nem támogatott metódusokat ellenőrizzük.

* PUT - Ebben a tesztben PUT metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* PATCH - Ebben a tesztben PATCH metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.

### ’favourites’ teszt mappa

Ebben a mappában három további mappa található, ezekben a mappákban minden kérés a [HTTP://localhost:3000/favourites](http://localhost:3000/favourites) címre kerül elküldésre. A kérések egy bejelentkezett felhasználó kedvencekbe mentett játékait kezeli, a tesztekhez használjuk a ’Successfull Login With Admin User’ tesztből származó admin felhasználó token-jét, kivéve ahol a teszt esetnél más van írva.

A ’Fetch Favourites’ mappában a bejelentkezett felhasználó kedvencekbe elmentett játékait kéri le GET metódussal.

* Fetch User's Favourites – Ebben a tesztben a bejelentkezett felhasználó kedvenceikbe elmentett játékait kéri, ha lefuttatjuk akkor egy HTTP 200-as kódú válasz tartalmazza játékok megjelenítésre kívánt adatait.
* Faulty Token – Ebben a tesztben már meg van adva egy hibás token, ha lefuttatjuk akkor HTTP 401-es kódú hibát kapunk, mert a token hibás.
* No Token Provided - Ennél a tesztnél nincs megadva token, ha lefuttatjuk akkor HTTP 401-es kódú hibát kapunk, mert a token hiányzik.

A ’Delete Favourite’ mappában egy bejelentkezett felhasználó a kedvencekből való törlés tesztjeit tartalmazza. Azt, hogy melyik játékot kell törölni azt a kérés body-ban a ’gameId’ értékében kell megadni a játék azonosítóját.

* Game Is Not a Favourite – Ennél a tesztnél egy olyan játék azonosítója van megadva, aminek a megadott felhasználónak nincs a kedvencekbe mentve, ha lefuttattjuk akkor HTTP 404-es kódú hibát kapunk.
* GameId Is Not a Number - Ennél a tesztnél a játék azonosítójának nem egy szám van megadva, ezért, ha lefuttattjuk akkor HTTP 400-as kódú hibát kapunk.
* Empty GameId - Ennél a tesztnél a játék azonosítója üres, ezért, ha lefuttattjuk akkor HTTP 400-as kódú hibát kapunk.
* GameId Is Null - Ennél a tesztnél a játék azonosítójának NULL érték van megadva, ezért, ha lefuttattjuk akkor HTTP 400-as kódú hibát kapunk.
* Missing GameId - Ennél a tesztnél a játék azonosítója hiányzik, ezért, ha lefuttattjuk akkor HTTP 400-as kódú hibát kapunk.
* Successfull Delete – Ennél a tesztnél van ilyen játék elmentve az adott felhasználó kedvenceinél, ezért, ha lefuttattjuk akkor HTTP 200-as kódú választ kapunk a sikeres törlésről.
* Faulty Token – Ebben a tesztben már meg van adva egy hibás token, ha lefuttatjuk akkor HTTP 401-es kódú hibát kapunk, mert a token hibás.
* No Token Provided - Ennél a tesztnél nincs megadva token, ha lefuttatjuk akkor HTTP 401-es kódú hibát kapunk, mert a token hiányzik.

A ’Method Not Allowed’ mappában lévő tesztekkel a nem támogatott metódusokat ellenőrizzük.

* POST - Ebben a tesztben POST metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* PUT - Ebben a tesztben PUT metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* PATCH - Ebben a tesztben PATCH metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.

### ’review’ teszt mappa

Ebben a mappában további három mappa található, ezekben a bejelentkezett felhasználó egy játékról írt kritikájához tartozó tesztjeit tartalmazzák. Ezekhez a tesztekhez használjuk a ’Successfull Login With Admin User’-ból származó admin felhasználó és ’Successfull Login With General User’-ból származó általános felhasználó token-jét. Az adott teszteknél meg van adva, hogy milyen típusú felhasználó token-t alkalmazzunk. Ebben a mappában minden teszt során a [HTTP://localhost:3000/game/:gameId/review](http://localhost:3000/game/:gameId/review) címet érjük el. Az URL-ben lévő paraméterbe az adott játék azonosítója kerül.

A ’Create Review’ mappában egy bejelentkezett felhasználó kritika létrehozását teszteljük egy adott játéknál, ehhez használjuk az admin jogosultsággal rendelkező felhasználó tokone-jét, de egy kritikát bármilyen bejelentkezett felhasználó létre tud hozni. Ahhoz, hogy valaki kritikát írjon, először értékelnie kell az adott játékot. A kéréshez a body-ban küldeni kell egy ’title’-t, ami a kritika címét kell tartalmaznia és egy ’content’-et ami a kritika törzsszövegét kell tartalmaznia. A megadott címet PUT metódussal érjük el.

* Content Contains Profanity – Ebben a tesztben egy kritikát próbálunk létrehozni, de a törzsszöveg káromkodást tartalmaz, ezt nem engedélyezzük, ezért HTTP 400-as kódú hibát kapunk.
* Title Contains Profanity – Ebben a tesztben egy kritikát próbálunk létrehozni, de a cím káromkodást tartalmaz, ezt nem engedélyezzük, ezért HTTP 400-as kódú hibát kapunk.
* Content Is Null - Ebben a tesztben egy kritikát próbálunk létrehozni, de a törzsszöveg NULL értékű, ezért HTTP 400-as kódú hibát kapunk.
* Content Is Empty - Ebben a tesztben egy kritikát próbálunk létrehozni, de a törzsszöveg üres, ezért HTTP 400-as kódú hibát kapunk.
* Content Is Missing - Ebben a tesztben egy kritikát próbálunk létrehozni, de a törzsszöveg hiányzik, ezért HTTP 400-as kódú hibát kapunk.
* Content Is Too Short - Ebben a tesztben egy kritikát próbálunk létrehozni, de a törzsszöveg nem éri el a minimum 50 karaktert, ezért HTTP 400-as kódú hibát kapunk.
* Content Is Too Long - Ebben a tesztben egy kritikát próbálunk létrehozni, de a törzsszöveg túllépi a maximum megengedett 5000 karaktert, ezért HTTP 400-as kódú hibát kapunk.
* Title Is Null - Ebben a tesztben egy kritikát próbálunk létrehozni, de a cím NULL értékű, ezért HTTP 400-as kódú hibát kapunk.
* Title is Empty - Ebben a tesztben egy kritikát próbálunk létrehozni, de a cím üres, ezért HTTP 400-as kódú hibát kapunk.
* Title Is Missing - Ebben a tesztben egy kritikát próbálunk létrehozni, de a cím hiányzik, ezért HTTP 400-as kódú hibát kapunk.
* Title Is Too Long - Ebben a tesztben egy kritikát próbálunk létrehozni, de a cím túllépi a maximum megengedett 100 karaktert, ezért HTTP 400-as kódú hibát kapunk.
* Title Is Too Short - Ebben a tesztben egy kritikát próbálunk létrehozni, de a cím nem éri el a minimum 5 karaktert, ezért HTTP 400-as kódú hibát kapunk.
* Faulty Token – Ebben a tesztben már meg van adva egy hibás token, ha lefuttatjuk akkor HTTP 401-es kódú hibát kapunk, mert a token hibás.
* No Token Provided - Ennél a tesztnél nincs megadva token, ha lefuttatjuk akkor HTTP 401-es kódú hibát kapunk, mert a token hiányzik.
* There's No Game With This Id – Ebben a tesztben az URL paraméterében olyan szám van megadva, amihez nem tartozik vele megegyező azonosítójú játék, ha lefuttatjuk akkor HTTP 404-es kódú hibát kapunk.
* Successfull Creation / Already Reviewed – Ha egyszer lefuttattjuk akkor sikeresen létrehozzunk egy kritikát az adott játékhoz, erre HTTP 201-es kódú választ kapunk. Ha még egyszer lefuttattjuk akkor HTTP 409-es kódú hibát kapunk, mert a felhasználó már írt egy kritikát az adott játékról.
* The User Did Not Rate The Game – Ebben a tesztben egy olyan játékhoz próbálunk kritikát írni, amit a megadott felhasználó még nem értékelt, ezért HTTP 400-as kódú hibát kapunk.

A ’Delete Review’ mappában a kritikák törléséhez tartozó tesztek tartózkodnak. Egy kritikát csak annak írója vagy admin jogosultságokkal rendelkező felhasználó tudja törölni. Azt, hogy melyik kritikát akarjuk törölni azt a kérés body-ban a ’reviewId’ tartalmazza, amiben a kritika azonosítójának kell lennie. A címet a DELETE metódussal érjük el.

* ReviewId Is Missing – Ehhez a teszthez használjuk az admin jogosultsággal rendelkező felhasználó token-jét, ha lefuttattjuk akkor HTTP 400-as kódú hibát kapunk, mert hiányzik a kritika azonosítója.
* ReviewId Is Null - Ehhez a teszthez használjuk az admin jogosultsággal rendelkező felhasználó token-jét, ha lefuttattjuk akkor HTTP 400-as kódú hibát kapunk, mert a kritika azonosítója NULL.
* ReviewId Is Empty - Ehhez a teszthez használjuk az admin jogosultsággal rendelkező felhasználó token-jét, ha lefuttattjuk akkor HTTP 400-as kódú hibát kapunk, mert a kritika azonosítója üres.
* There's No Review With This Id - Ehhez a teszthez használjuk az admin jogosultsággal rendelkező felhasználó token-jét, ha lefuttattjuk akkor HTTP 404-es kódú hibát kapunk, mert a megadott reviewId-hoz nem tartozik kritika.
* There's No Game Id - Ehhez a teszthez használjuk az admin jogosultsággal rendelkező felhasználó token-jét, ha lefuttattjuk akkor HTTP 400-as kódú hibát kapunk, mert az URL paraméteréből hiányzik a játék azonosítója.
* There's No Game With This Id - Ehhez a teszthez használjuk az admin jogosultsággal rendelkező felhasználó token-jét, ha lefuttattjuk akkor HTTP 404-es kódú hibát kapunk, mert az URL paraméterében megadott számhoz nem tartozik játék ilyen azonosítóval.
* Faulty Token – Ebben a tesztben már meg van adva egy hibás token, ha lefuttatjuk akkor HTTP 401-es kódú hibát kapunk, mert a token hibás.
* No Token Provided - Ennél a tesztnél nincs megadva token, ha lefuttatjuk akkor HTTP 401-es kódú hibát kapunk, mert a token hiányzik.
* Successful Delete With Admin / No Review With This Id - Ehhez a teszthez használjuk az admin jogosultsággal rendelkező felhasználó token-jét, ha egyszer lefuttattjuk akkor sikeresen töröljük a megadott kritikát és HTTP 200-as kódú választ kapunk. A törlés sikeres, mert admin jogosultsággal bármilyen kritikát törölhetünk. Ha még egyszer lefuttattjuk akkor HTTP 404-es hibát kapunk, mert a ’reviewId’-ban megadott kritika nem létezik.
* Successful Delete By The Author / No Review With This Id - Ehhez a teszthez használjuk az említett általános jogosultsággal rendelkező felhasználó token-jét, ha egyszer lefuttattjuk akkor sikeresen töröljük a megadott kritikát és HTTP 200-as kódú választ kapunk. A törlés sikeres, mert az a felhasználó törölte a kritikát, akitől származik. Ha még egyszer lefuttattjuk akkor HTTP 404-es hibát kapunk, mert a ’reviewId’-ban megadott kritika nem létezik.
* Delete Is Not Authorized - Ehhez a teszthez használjuk az említett általános jogosultsággal rendelkező felhasználó token-jét, lefuttattjuk akkor HTTP 401-es hibát kapunk, mert a megadott felhasználónak nincs admin jogosultsága és nem a kritika írója.

A ’Method Not Allowed’ mappában lévő tesztekkel a nem támogatott metódusokat ellenőrizzük.

* GET - Ebben a tesztben GET metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* POST - Ebben a tesztben POST metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.
* PATCH - Ebben a tesztben PATCH metódussal próbáljuk elérni a címet, de ezt backend nem támogatja ezért HTTP 405-ös kódú hibát küldünk.

## Terhelés teszt

A terhelés tesztben azt teszteljük, hogy a backend szerver mennyi kérést tud kezelni. Ehhez a teszthez a k6 nevű csomagot használjuk, először indítsuk el a XAMPP Control Panel-t és indítsuk el az Apache és MySQL-t, majd egy terminal-ban futtassuk a szervert az ’npm run dev’ paranccsal, ezután nyissunk meg egy új terminal-t majd futtassuk le az ’npm run loadtest’ parancsot a terhelés teszt elkezdéséhez. A teszteket a <http://localhost:3000/game/1> címen hajtottuk végre, ami az oldalunkon a legnagyobb lekérdezést tartalmazza, ha GET metódussal férünk hozzá. A teszteket egy fejlesztői gépen futtattuk le, ami egy ’AMD Ryzen 3 7320U with Radeon Graphics 2.40 GHz’ processzorral rendelkezik, és 8 GB memóriával. Ezen gépen futott még az adatbázis a backend szerverrel. Amikor lefuttatjuk a terhelés tesztet, akkor a ’loadtest.js’ fájlt indítjuk el.

// Importáljuk a teszthez szükséges k6 csomagot

import http from 'k6/http';

import { sleep } from 'k6';

// Itt beállítjuk a tesztet

export let options = {

    // A vus-ban megadjuk, hogy hány virtuális felhasználóval végezzük a tesztet

    vus: 200,

    // A duration-ben megadjuk, hogy mennyi ideig tartson a teszt

    duration: '180s'

};

export default function () {

    // Itt megadjuk, hogy milyen metódussal érjük el melyik címet

    http.get('http://localhost:3000/game/1');

    // Megadjuk, hogy egy virtuális felhasználó kérése után várjon egy másodpercet

    sleep(1);

}

Ha szeretnénk változtatni hogy hány virtuális felhasználóval akarunk tesztelni, akkor azt a ’vus’ változóban adjuk meg az általunk kívánt számot, illetve a teszt ideje is változtatható a ’duration’ változónál.

A legoptimálisabb teszt eredményben 150 virtuális felhasználó másodpercenként küldött kérést a címre 180 másodpercig. A teszt során nem volt hiba és az átlagos válaszidő 1,63 másodperc volt.

A legnagyobb hiba mentes teszt eredményben 200 virtuális felhasználó másodpercenként küldött kérést a címre 180 másodpercig. A teszt során nem volt hiba és az átlagos válaszidő 2,55 másodperc volt.

Alacsony hiba aránnyal rendelkező teszt eredményében 250 virtuális felhasználó másodpercenként küldött kérést a címre 180 másodpercig. A teszt során 7 kérés került visszautasításra és az átlagos válaszidő 3,49 másodperc volt.

A terhelést nem bíró teszt eredményében 300 virtuális felhasználó másodpercenként küldött kérést a címre 180 másodpercig. A teszt során 68 kérés került visszautasításra és az átlagos válaszidő 4,34 másodperc volt.

Fontos megjegyezni, hogy az tesztek eredményei különböző számítógépeken eltérhetnek, mert befolyásolja a processzor és a memória mérete.