

# 3. WebSocket házi feladat

---

Simon Balázs (sbalazs@iit.bme.hu), BME IIT, 2015.

A továbbiakban a NEPTUN szó helyére a saját Neptun-kódot kell behelyettesíteni, csupa nagybetűkkel.

## 1 A feladat leírása

A feladat egy mozi jegyfoglalási rendszerének elkészítése, és ennek WebSocket szolgáltatásként való publikálása.

## 2 A szolgáltatás

A szolgáltatást egy Eclipse dinamikus webalkalmazásban kell megvalósítani.

A webalkalmazás neve a következő: **WebSocket\_NEPTUN**

A szolgáltatást a következő URL-en kell publikálni:

`ws://localhost:8080/WebSocket_NEPTUN/cinema`

A szolgáltatásnak JSON formátumban kell kommunikálnia a külvilággal. A sorokat és az oszlopokat pozitív egész számok jelölik.

A szolgáltatás a következő üzeneteket kaphatja a kientől:

Művelet	Példa üzenet	Leírás
Moziterem inicializálása	<pre>{   "type": "initRoom",   "rows": 10,   "columns": 20 }</pre>	A mozitermet a <b>rows</b> -ban megadott számú sorral és a <b>columns</b> -ban megadott számú oszloppal inicializálja. Az érvényes sorok 1-től a <b>rows</b> -ban megadott értékig futnak, az érvényes oszlopok 1-től a <b>columns</b> -ban megadott értékig futnak.  Ha a <b>rows</b> vagy <b>columns</b> értéke nem pozitív egész, akkor hibát küld vissza (lásd az <b>error</b> típusú üzenet a kliensnél).
Moziterem méretének lekérdezése	<pre>{"type": "getRoomSize"}</pre>	Visszaküldi a moziterem méretét (lásd a <b>roomSize</b> típusú üzenet a kliensnél).
Székek állapotának lekérdezése	<pre>{"type": "updateSeats"}</pre>	Egymásután visszaküldi minden egyes szék állapotát (lásd a <b>seatStatus</b> típusú üzenet a kliensnél).

<b>Szék zárolása</b>	<pre>{   "type": "lockSeat",   "row": 3,   "column": 11 }</pre>	<p>Zárolja a <b>row</b> sorú és <b>column</b> oszlopú széket. Eredményként visszaküldi a zárolás azonosítóját (lásd a <b>lockResult</b> típusú üzenet a kliensnél), valamint az új szék státuszát (lásd a <b>seatStatus</b> típusú üzenet a kliensnél). Az új szék státuszát minden nyitott Session-ben vissza kell küldeni, hogy minden kliens lássa az eredményt.</p> <p>Amennyiben a szék nem szabad (már zárolt vagy foglalt), vagy a sor-oszlop érték hibás, akkor hibát küld vissza (lásd az <b>error</b> típusú üzenet a kliensnél).</p>
<b>Zárolás feloldása</b>	<pre>{   "type": "unlockSeat",   "lockId": "lock5" }</pre>	<p>Feloldja a <b>lockId</b> azonosítójú zárolást. Eredményként visszaküldi a szabaddá tett szék státuszát (lásd a <b>seatStatus</b> típusú üzenet a kliensnél). A szék státuszát minden nyitott Session-ben vissza kell küldeni, hogy minden kliens lássa az eredményt.</p> <p>Amennyiben nincs ilyen azonosítójú zárolás, akkor hibát küld vissza (lásd az <b>error</b> típusú üzenet a kliensnél).</p>
<b>Zárolás lefoglalása</b>	<pre>{   "type": "reserveSeat",   "lockId": "lock7" }</pre>	<p>Lefoglalja a <b>lockId</b> azonosítójú zárolásban zárolt széket. Eredményként visszaküldi a lefoglalttá vált szék státuszát (lásd a <b>seatStatus</b> típusú üzenet a kliensnél). A szék státuszát minden nyitott Session-ben vissza kell küldeni, hogy minden kliens lássa az eredményt.</p> <p>Amennyiben nincs ilyen azonosítójú zárolás, akkor hibát küld vissza (lásd az <b>error</b> típusú üzenet a kliensnél).</p>

A szolgáltatásnak adatokat kell tárolnia az egyes hívások között. Egy valódi alkalmazás esetén az adatok tárolására adatbázist kéne használni. **A házi feladatban a könnyebbség kedvéért statikus változóiban tároljuk a szükséges adatokat!** Például a nyitott Session-öket is egy statikus listában érdemes tárolni.

### 3 A kliens

A kliensnek egy HTML-ben futó JavaScript alkalmazásnak kell lennie. A HTML-t be kell ágyazni a szerver webalkalmazásba, vagyis az Eclipse dinamikus webprojektjében szokásos **WebContent** könyvtárban kell elhelyezni. A kiindulási HTML kódot a jelen dokumentumhoz csatolt **cinema.html** fájl adja. Miután az alkalmazás feltelepült a szerverre, a HTML-nek az alábbi URL-en elérhetőnek kell lennie:

`http://localhost:8080/WebSocket_NEPTUN/cinema.html`

A kliens a következő üzeneteket kaphatja a szolgáltatástól:

Művelet	Példa üzenet	Leírás
<b>Moziterem mérete</b>	<pre>{   "type": "roomSize",   "rows": 10,   "columns": 20 }</pre>	<p>A <b>getRoomSize</b> műveletre válaszként érkezik, és megadja az éppen aktuális szerveren tárolt terem sorainak (<b>rows</b>) és oszlopainak (<b>columns</b>) számát.</p> <p>Ennek az üzenetnek a megérkezése után le kell kérdezni az összes szék állapotát (<b>updateSeats</b>), és a székeket ki kell rajzolni a canvas-re.</p>
<b>Egy szék státusza</b>	<pre>{   "type": "seatStatus",   "row": 4,   "column": 5,   "status": "free" }</pre>	<p>Frissítésként jövő üzenet a szervertől. Megadja az adott sorban (<b>row</b>) és oszlopban (<b>column</b>) szereplő szék státuszát (<b>status</b>). A státusz a következő értékeket veheti fel:</p> <ul style="list-style-type: none"><li>• <b>free</b>: szabad</li><li>• <b>locked</b>: zárolt</li><li>• <b>reserved</b>: foglalt</li></ul> <p>Ha a kliens bármikor ilyen üzenetet kap, azonnal ki kell rajzolnia a canvas-re a széket a megfelelő státusszal.</p>
<b>Zárolás eredménye</b>	<pre>{   "type": "lockResult",   "lockId": "lock45" }</pre>	<p>Zárolása válaszként érkező üzenet a szervertől. Megadja a zárolás eredményeként előálló azonosítót (<b>lockId</b>). Ezt az azonosítót lehet foglalásra felhasználni.</p>
<b>Hiba</b>	<pre>{   "type": "error",   "message": "Seat is not free." }</pre>	<p>Valamilyen művelet esetén visszaküldött hibaüzenet. A <b>message</b>-ben visszaadott üzenetet egy <b>alert</b> JavaScript hívással jelenítse meg a kliens.</p>

## 4 Segítség a megoldáshoz

A JSON sorosításhoz célszerű saját Encoder/Decoder-t írni. Lehet használni JAXB sorosítást is, de lehet használni a **javax.json** csomagban található dinamikus objektumkezelést is.

Az üzenetek **type** mezője alapján lehet megkülönböztetni, hogy milyen műveletet is kellene végrehajtani. Ez a mező minden üzenetben szerepel.

Előfordulhat, hogy egy üzenet hatására több üzenettel is kell válaszolni. Ez nem oldható meg egy egyszerű **return** utasítással. Ilyenkor a **Session** típusú paramétert is használni kell, és annak a **getBasicRemote()** függvényén keresztül elért objektum segítségével tetszőlegesen sok üzenet visszaküldhető. Itt azonban ügyelni kell arra, hogy a szerver osztályra felannotált **Encoder** csak a **return**-ben visszaadott objektumra hívódik meg, a **Session**-ön keresztül küldött objektumra nem! Így ebben az esetben az **Encoder**-t manuálisan meg kell hívni.

A HTML kliens jelenleg csak egy vázat biztosít. A gombokra be kell kötni a megfelelő eseménykezelő függvényeket. A **TODO** szót érdemes keresni, ezeken a helyeken kell változtatni a HTML kódon. A HTML kód tartalmazza a székek kirajzolásához szükséges függvényeket, és ezeket célszerű felhasználni, hogy a képernyőn mindig a mozi terem aktuális állapota jelenjen meg.

Előfordulhatnak olyan API részek, amelyek előadásán nem lettek részletesen kifejtve (pl. **javax.json** csomag használata), illetve a JavaScript nyelv sem szerepelt az alapképzésben. A hiányzó ismereteknek egyénileg kell utánajárni.

## 5 Beadandók

Beadandó egyetlen ZIP fájl. Más tömörítési eljárás használata tilos!

A ZIP fájl gyökerében egyetlen könyvtárnak kell lennie, az Eclipse webalkalmazás könyvtárának:

- **WebSocket\_NEPTUN**: a szolgáltatás Eclipse alkalmazás teljes egészében, amely egyben tartalmazza a kliensként funkcionáló cinema.html-t is

A lefordított class fájlokat nem kötelező beadni.

A szolgáltatásnak a fent specifikált JSON üzeneteket kell támogatnia. Az üzenetek formátumának megváltoztatása tilos!

A megoldásnak a telepítési leírásban meghatározott környezetben kell fordulnia és futnia, más JAR illetve 3rd party könyvtár nem használható!

Fontos: a dokumentumban szereplő elnevezéseket és kikötéseket pontosan be kell tartani!

Még egyszer kiemelve: a NEPTUN szó helyére mindig a saját Neptun-kódot kell behelyettesíteni, csupa nagybetűvel!