3. WebSocket házi feladat

Dr. Simon Balázs (sbalazs@iit.bme.hu), BME IIT, 2023.

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 Maven vagy egy .NET Core webalkalmazásban kell megvalósítani. A két technológia közül szabadon lehet választani.

A webalkalmazás neve a következő: **WebSocket_NEPTUN**

A szolgáltatást a következő URL-en kell publikálni:

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

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 változókban* tároljuk a szükséges adatokat! Például a nyitott Session-öket is egy statikus listában érdemes tárolni.

2.1 Maven választása esetén

A szolgáltatást egy ugyanolyan Maven webalkalmazásban kell megvalósítani, mint amilyen a tutorialban szerepel. A **pom.xml**-nek kötelezően a csatolt **service/pom.xml**-nek kell lennie, de a NEPTUN szót le kell cserélni benne a saját Neptun-kódra.

A forrásfájlok készülhetnek Java vagy Kotlin nyelven is.

2.2 .NET Core választása esetén

A szolgáltatást egy ugyanolyan ASP.NET Core webalkalmazásban kell megvalósítani, mint amilyen a tutorialban szerepel. A projektfájlnak kötelezően a csatolt **service/WebSocket_NEPTUN.csproj**-nak kell lennie, de a fájl nevében a NEPTUN szót le kell cserélni a saját Neptun-kódra.

2.3 Működés

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

A szolgáltatás a következő üzeneteket kaphatja a klienstől:

Művelet	Példa üzenet	Leírás

Moziterem inicializálása	<pre>{
	  "type": "initRoom",
	  "rows": 10,
	  "columns": 20
	}</pre>
		A mozitermet a **rows**-ban megadott számú sorral és a **columns**-ban megadott számú oszloppal inicializálja. Az érvényes sorok 1-től a **rows**-ban megadott értékig futnak, az érvényes oszlopok 1-től a **columns**-ban megadott értékig futnak.
		
		Ha a **rows** vagy **columns** értéke nem pozitív egész, akkor hibát küld vissza (lásd az **error** típusú üzenet a kliensnél).

Moziterem méretének lekérdezése	<pre>{"type":"getRoomSize"}</pre>	Visszaküldi a moziterem méretét (lásd a **roomSize** típusú üzenet a kliensnél).
Székek állapotának lekérdezése	<pre>{"type":"updateSeats"}</pre>	Egymásután visszaküldi minden egyes szék állapotát (lásd a **seatStatus** típusú üzenet a kliensnél).

Szék zárolása	<pre>{
	  "type": "lockSeat",
	  "row": 3,
	  "column": 11
	}</pre>
		Zárolja a **row** sorú és **column** oszlopú széket. Eredményként visszaküldi a zárolás azonosítóját (lásd a **lockResult** típusú üzenet a kliensnél), valamint az új szék státuszát (lásd a **seatStatus** típusú üzenet a kliensnél). Az új szék státuszát minden nyitott Session-ben vissza kell küldeni, hogy minden kliens lássa az eredményt.
		
		Amennyiben a szék nem szabad (már zárolt vagy foglalt), vagy a sor-oszlop érték hibás, akkor hibát küld vissza (lásd az **error** típusú üzenet a kliensnél).

Zárolás feloldása	<pre>{
	  "type": "unlockSeat",
	  "lockId": "lock5"
	}</pre>
		Feloldja a **lockId** azonosítójú zárolást. Eredményként visszaküldi a szabaddá tett szék státuszát (lásd a **seatStatus** típusú üzenet a kliensnél). A szék státuszát minden nyitott Session-ben vissza kell küldeni, hogy minden kliens lássa az eredményt.
		
		Amennyiben nincs ilyen azonosítójú zárolás, akkor hibát küld vissza (lásd az **error** típusú üzenet a kliensnél).

Zárolás lefoglalása	{
	"type": "reserveSeat",
	"lockId": "lock7"
	}
		Lefoglalja a **lockId** azonosítójú zárolásban zárolt széket. Eredményként visszaküldi a lefoglalttá vált szék státuszát (lásd a **seatStatus** típusú üzenet a kliensnél). A szék státuszát minden nyitott Session-ben vissza kell küldeni, hogy minden kliens lássa az eredményt.
		
		Amennyiben nincs ilyen azonosítójú zárolás, akkor hibát küld vissza (lásd az **error** típusú üzenet a kliensnél).

3 A kliens

A kliensnek egy HTML-ben futó JavaScript alkalmazásnak kell lennie. A HTML-t be kell ágyazni a szerver webalkalmazásba. A kiindulási HTML kódot a jelen dokumentumhoz csatolt **client/cinema.html** fájl adja. Miután az alkalmazás elindult, 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

Moziterem mérete	{
	"type": "roomSize",
	"rows": 10,
	"columns": 20
	}
		A **getRoomSize** műveletre válaszként érkezik, és megadja az éppen aktuális szerveren tárolt terem sorainak (**rows**) és oszlopainak (**columns**) számát.
		
		Ennek az üzenetnek a megérkezése után le kell kérdezni az összes szék állapotát (**updateSeats**), és a székeket ki kell rajzolni a canvas-re.

Egy szék státusza	{
	"type": "seatStatus",
	"row": 4,
	"column": 5,
	"status": "free"
	}
		Frissítésként jövő üzenet a szervertől. Megadja az adott sorban (**row**) és oszlopban (**column**) szereplő szék státuszát (**status**). A státusz a következő értékeket veheti fel:
		• **free**: szabad
		• **locked**: zárolt
		• **reserved**: foglalt
		
		Ha a kliens bármikor ilyen üzenetet kap, azonnal ki kell rajzolnia a canvas-re a széket a megfelelő státusszal.

Zárolás eredménye	<pre>{
	  "type": "lockResult",
	  "lockId": "lock45"
	}</pre>
		Zárolása válaszként érkező üzenet a szervertől. Megadja a zárolás eredményeként előálló azonosítót (**lockId**). Ezt az azonosítót lehet foglalásra felhasználni.

Hiba	<pre>{
	  "type": "error",
	  "message": "Seat is not free."
	}</pre>
		Valamilyen művelet esetén visszaküldött hibaüzenet. A **message**-ben visszaadott üzenetet egy **alert** JavaScript hívással jelenítse meg a kliens.

4 Segítség a megoldáshoz

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 moziterem aktuális állapota jelenjen meg.

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

Előfordulhatnak olyan API részek, amelyek előadáson nem lettek részletesen kifejtve (pl. JSON sorosítás csomag használata), illetve a JavaScript nyelv sem szerepelt az alapképzésben. A hiányzó ismeretek elsajátítása is a feladat egyik kihívása.

Vigyázzunk, hogy többszálú esetben is jól működjön az alkalmazásunk, mert lehet, hogy két kliens egyszerre próbál meg csatlakozni!

4.1 Tippek Java esetén

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

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, új kapcsolat nyitásakor pedig az új **Session** begyűjthető.

4.2 Tippek C# esetén

A JSON sorosításhoz célszerű saját Encoder/Decoder-t írni. Lehet használni System.Text.Json névteret vagy a Json.NET csomagot is.

Előfordulhat, hogy egy üzenet hatására több üzenettel is kell válaszolni. Ez nem oldható meg egy egyszerű **return** utasítással. Ilyenkor a **WebSocket** típusú paramétert is használni kell, és azon keresztül

tetszőlegesen sok üzenet visszaküldhető, új kapcsolat nyitásakor pedig az új **WebSocket** objektum begyűjthető.

5 Beadandók

Beadandó egyetlen ZIP fájl. Más tömörítési eljárás használata tilos!

A ZIP fájl gyökerében egyetlen könyvtárnak kell lennie, a szolgáltatás alkalmazása:

- **WebSocket_NEPTUN:** a szolgáltatás Maven vagy .NET Core projektje teljes egészében

A szolgáltatásnak a fent specifikált JSON üzeneteket kell támogatnia. Más formátum/elnevezés használata tilos!

A megoldásnak a telepítési leírásban meghatározott környezetben kell fordulnia és futnia, a **pom.xml**-ek illetve a **csproj** fájlok tartalmát a Neptun-kód módosításán felül megváltoztatni tilos!

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 nagybetűkkel!

5.1 Maven választása esetén

Mielőtt a projektet becsomagolnánk a ZIP-be, a projekten belül adjuk ki az **mvn clean** parancsot! Ez a parancs törli a **target** alkönyvtárat, amely elég nagy is lehet és egyébként sem lesz figyelembe véve a kiértékelés során.

A ZIP kibontása után a szolgáltatásnak parancssorból fordulnia kell a következő parancs kiadásával:

```
...\WebSocket_NEPTUN>mvn package
```

A szolgáltatásnak parancssorból települnie kell a szerverre következő parancs kiadásával:

```
...\WebSocket_NEPTUN>mvn wildfly:deploy
```

5.2 .NET Core választása esetén

Mielőtt a projektet becsomagolnánk a ZIP-be, a projekten belül adjuk ki a **dotnet clean** parancsot! Ez a parancs törli a **bin** és **obj** alkönyvtárakat, amelyek elég nagyok is lehetnek és egyébként sem lesznek figyelembe véve a kiértékelés során.

A ZIP kibontása után a szolgáltatásnak parancssorból fordulnia kell a következő parancs kiadásával:

```
...\WebSocket_NEPTUN>dotnet build
```

A szolgáltatásnak parancssorból futnia kell a következő parancs kiadásával:

```
...\WebSocket_NEPTUN>dotnet run
```