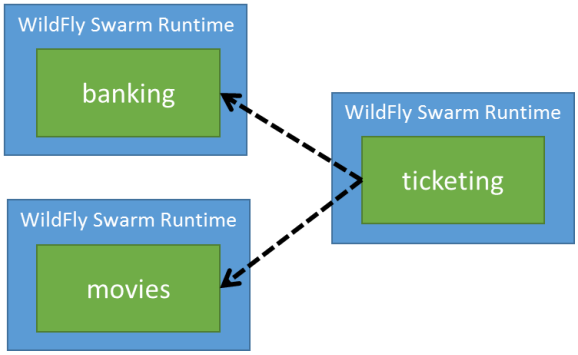
# 4. Mikroszolgáltatások házi feladat

*Dr. Simon Balázs (sbalazs@iit.bme.hu), BME IIT, 2017.*

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

## 1 A feladat leírása

A feladat egy mozi jegyvásárló rendszerének kialakítása mikroszolgáltatásokból. A rendszer architektúrája a következő:




A szerver három mikroszolgáltatásból épül fel:

- **movies**: egy filmek adatbázisát kezelő erőforrás alapú REST szolgáltatás
- **banking**: banki műveleteket támogató RPC alapú REST szolgáltatás
- **ticketing**: jegyvásárlást támogató RPC alapú REST szolgáltatás

Minden mikroszolgáltatás egy önálló mikroszerveren fut. A jegyvásárló szolgáltatás a banki és a mozi szolgáltatásokra épít.

## 2 A mozi szolgáltatás

A szolgáltatást egy WildFly Swarm Maven alkalmazásban kell megvalósítani. A pom.xml-nek kötelezően a csatolt **movies/pom.xml**-nek kell lennie, de a NEPTUN szót le kell cserélni benne a saját Neptun-kódra.

Az alkalmazás neve a következő: **Microservices_NEPTUN_Movies**, a főosztály pedig: **program.Program**.

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!

A szolgáltatás báziscíme a következő URL:

**`http://localhost:8081`**

A szolgáltatás funkcionalitása pontosan ugyanaz, mint a 2. REST házi feladatban szereplő szolgáltatásé. Pontosan ugyanazokat a hívásokat kell támogatnia a fenti báziscímtől számítva. A szolgáltatás megvalósításához szükséges üzenetek formátumát a csatolt **movies/movies.proto** fájl írja le.

A szolgáltatás tehát a következő hívásokat támogatja:

- **`GET /movies`**
  - bemenet HTTP body: nincs
  - kimenet HTTP body: **`MovieList`**
- **`GET /movies/{id}`**
  - bemenet HTTP body: nincs
  - kimenet HTTP body: **`Movie`**
- **`POST /movies`**
  - bemenet HTTP body: **`Movie`**
  - kimenet HTTP body: **`MovieId`**
- **`PUT /movies/{id}`**
  - bemenet HTTP body: **`Movie`**
  - kimenet HTTP body: nincs
- **`DELETE /movies/{id}`**
  - bemenet HTTP body: nincs
  - kimenet HTTP body: nincs
- **`GET /movies/find?year={year}&orderby={field}`**
  - bemenet HTTP body: nincs
  - kimenet HTTP body: **`MovieIdList`**

# 3 A banki szolgáltatás

A szolgáltatást egy WildFly Swarm Maven alkalmazásban kell megvalósítani. A pom.xml-nek kötelezően a csatolt **banking/pom.xml**-nek kell lennie, de a NEPTUN szót le kell cserélni benne a saját Neptun-kódra.

Az alkalmazás neve a következő: **Microservices_NEPTUN_Banking**, a főosztály pedig: **program.Program**.

A szolgáltatás báziscíme a következő URL:

**`http://localhost:8082`**

A szolgáltatás interfészét a következő pszeudo-kód reprezentálja:

```
interface Banking
{
    bool ChargeCard(string cardNumber, int amount);
}
```

A **ChargeCard** függvény egy hitelkártya terhelését szimulálja. A **cardNumber** egy tetszőleges szöveg, az **amount** pedig egy tetszőleges egész szám. A függvény akkor térjen vissza igaz értékkel, ha az **amount** értéke pozitív, és a **cardNumber** páros sok karakter hosszú. Egyébként a függvény hamis értékkel térjen vissza.

A szolgáltatást RPC alapú REST szolgáltatásként kell megvalósítani. A be- és kimenő üzenetek formátumát a csatolt **banking/banking.proto** írja le. A szolgáltatás függvényeit a következő POST kérésekkel lehet meghívni a fenti báziscímtől számítva:

- **`POST /banking/ChargeCard`**
  - bejövő üzenet HTTP body-ban: **`ChargeCardRequest`**
  - kimenő üzenet HTTP body-ban: **`ChargeCardResponse`**

# 4 A jegyvásárlási szolgáltatás

A szolgáltatást egy WildFly Swarm Maven alkalmazásban kell megvalósítani. A pom.xml-nek kötelezően a csatolt **ticketing/pom.xml**-nek kell lennie, de a NEPTUN szót le kell cserélni benne a saját Neptun-kódra.

Az alkalmazás neve a következő: **Microservices_NEPTUN_Ticketing**, a főosztály pedig: **program.Program**.

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!

A szolgáltatás báziscíme a következő URL:

**`http://localhost:8080`**

A szolgáltatás interfészét a következő pszeudo-kód reprezentálja:

```
struct Movie
{
    int id;
    string title;
}

struct Ticket
{
    int movieId;
    int count;
}

interface Ticketing
{
    Movie[] GetMovies(int year);
    bool BuyTickets(int movieId, int count, string cardNumber);
    Ticket[] GetTickets();
}
```

A **GetMovies** függvény visszaadja az adott év (**year** paraméter) filmjeinek listáját a filmek címei szerint sorba rendezve. Ehhez a mozi szolgáltatás **`GET /movies/find?year={year}&orderby={field}`** hívását lehet igénybe venni az azonosítók listájának lekéréséhez, valamint a **`GET /movies/{id}`** hívást lehet felhasználni a filmek címeinek kinyeréséhez. Az így összekombinált eredményekből kapott filmek listáját adja vissza a **GetMovies** függvény.

A **BuyTickets** függvény egy adott azonosítójú (**movieId** paraméter) filmre vásárol **count** darab jegyet a banki szolgáltatás segítségével. A vásárlás összege **10*count**, vagyis egy jegy 10 pénzegységbe kerül. A hitelkártyaszámot (**cardNumber** paraméter) változtatás nélkül tovább kell adni a banki szolgáltatás **ChargeCard** függvényének. A **BuyTickets** függvény visszatérési értéke ugyanaz, mint a továbbhívott banki szolgáltatás **ChargeCard** függvényének visszatérési értéke.

A **Ticketing** szolgáltatásnak folyamatosan nyilván kell tartania, hogy az egyes filmekre hány jegyet sikerült eddig eladni. Amennyiben egy **BuyTickets** hívás esetén a vásárlás sikeres volt (a **ChargeCard** függvény igazzal tért vissza), a **Ticketing** szolgáltatásnak hozzá kell adnia az eladott jegyek darabszámát az adott filmhez tartozó számlálóhoz.

A **GetTickets** függvény visszaadja a jegyeladási statisztikák adatait. A visszaadott listában pontosan azoknak a filmeknek kell szerepelnie, amelyekre legalább egy jegyet már sikerült eladni. A lista egyes elemei (**Ticket** objektum) tartalmazzák a film azonosítóját (**movieId**), valamint az eddig eladott jegyek darabszámát (**count**).

A szolgáltatást RPC alapú REST szolgáltatásként kell megvalósítani. A be- és kimenő üzenetek formátumát a csatolt **ticketing/ticketing.proto** írja le. A szolgáltatás függvényeit a következő POST kérésekkel lehet meghívni a fenti báziscímtől számítva:

- **`POST /ticketing/GetMovies`**
  - bejövő üzenet HTTP body-ban: **`GetMoviesRequest`**
  - kimenő üzenet HTTP body-ban: **`GetMoviesResponse`**
- **`POST /ticketing/BuyTickets`**
  - bejövő üzenet HTTP body-ban: **`BuyTicketsRequest`**
  - kimenő üzenet HTTP body-ban: **`BuyTicketsResponse`**
- **`POST /ticketing/GetTickets`**
  - bejövő üzenet HTTP body-ban: **`GetTicketsRequest`**
  - kimenő üzenet HTTP body-ban: **`GetTicketsResponse`**

Fontos: a saját mozi és banki szolgáltatás bázis URL-jét tilos beleégetni a jegyvásárló szolgáltatás kódjába! Ehelyett a csatolt **ticketing/project-defaults.yml** konfigurációs fájl alapján kell konfigurálni a hívott szolgáltatások bázis URL-jét, amelyet a tutorial szerint az **src/main/resources** könyvtárban kell elhelyezni. A két hívott szolgáltatás URL-jének konfigurációja így néz ki:

```
microservices:
    movies:
        url: http://localhost:8081
    banking:
        url: http://localhost:8082
```

A konfigurációban szereplő értékek a főprogramban a Swarm objektumtól kérdezhetők le:

```
String moviesUrl=swarm.configView().resolve("microservices.movies.url").getValue();
String bankingUrl=swarm.configView().resolve("microservices.banking.url").getValue();
```

## 5 Üzenetformátumok

Minden szolgáltatásnak egyszerre kell támogatnia a JSON és Protocol Buffer formátumokat is! Az egyes formátumok esetén a MIME típusok a következők:

- JSON: **application/json**
- Protocol Buffers: **application/x-protobuf**

A proto üzenetek JSON sorosításához a **protobuf-java-util** csomag használható, amelyet a csatolt **pom.xml**-ek függőségként már tartalmaznak.

(A JSON üzeneteknek nem a 2. REST házi feladatban megadottnak kell lenniük, hanem az itt definiált proto üzeneteket kell sorosítani a bináris változat mellett JSON formátumban is.)

## 6 A kliens

Beadandó klienst nem kell készíteni. Saját célra készíthető kliens, de azt nem kell beadni.

## 7 Beadandók

Beadandó egyetlen ZIP fájl. Más tömörítési eljárás használata tilos!

A ZIP fájl gyökerében három könyvtárnak kell lennie, a három szolgáltatás alkalmazásának:

- **Microservices_NEPTUN_Movies:** a mozi szolgáltatás Maven projektje teljes egészében
- **Microservices_NEPTUN_Banking:** a banki szolgáltatás Maven projektje teljes egészében
- **Microservices_NEPTUN_Ticketing:** a jegyvásárlási szolgáltatás Maven projektje teljes egészében

A lefordított class fájlokat (target alkönyvtár) nem kötelező beadni.

Minden projektnek parancssorból önállóan fordulnia kell az adott projekt főkönyvtárában kiadott **`mvn package`** parancs segítségével.

Minden szolgáltatásnak parancssorból önállóan el kell indulnia az adott projekt főkönyvtárában kiadott **`mvn wildfly-swarm:run`** parancs segítségével.

A szolgáltatásoknak a csatolt proto fájlok által leírt üzeneteket kell támogatnia, a proto fájlok megváltoztatása tilos. Más formátum vagy elnevezés használata tilos!

A megoldásnak a telepítési leírásban meghatározott környezetben kell fordulnia és futnia, a **pom.xml**-ek 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!