
# KIV/DS – Semestrální práce

## 1. Samostatná práce - Sekvencer

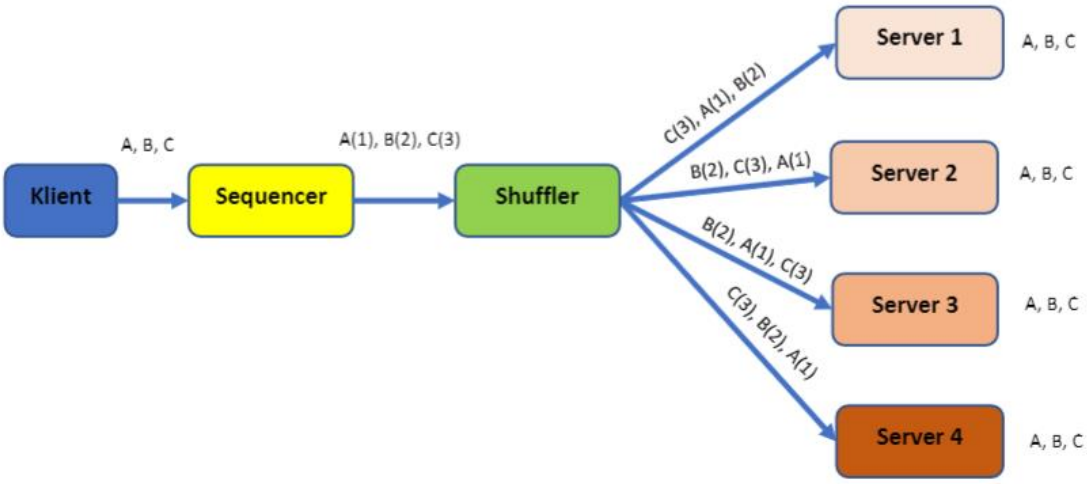
**Martin Kantořík**
**A18N0090P**
Narozen 18. 8. 1994
kantorik@students.zcu.cz

# Obsah

# Zadání

Implementujte distribuovanou bankovní aplikaci, která využívá sekvencer k řazení příchozích požadavků. Popis chodu aplikace:

- všechny servery začínají se stejným zůstatek na účtu 5.000.000
- klient náhodně generuje částku v intervalu a náhodně vybere operaci CREDIT či DEBIT a pošle operaci s částkou sekvenceru. Klient celkem vygeneruje volitelný počet operací (např. číslo předané z příkazové řádky)
- sekvencer přiřadí jednoznačný identifikátor (rostoucí sekvence celých čísel) zprávě klienta a pošle zprávu „shuffleru“, který každou zprávu pošle na všechny bankovní servery (1 až N, kde N > 3) v náhodném pořadí a zároveň, změní jejich pořadí.
- bankovní servery se musí vypořádat se zprávami, které chodí mimo původní pořadí, aby operace nad jejich lokálním účtem byly provedeny ve správném pořadí tak, jak je poslal klient.
- Po ukončení činnosti klienta a zpracování všech operací, musí být v případě správné funkce bankovních serverů na všech serverech stejný zůstatek.
- K vytvoření infrastruktury použijte nástroje Vagrant (+ VirtualBox), k instalaci, konfiguraci a nasazení aplikace využijte jeho „shell provisioning”, volitelně lze použít nástroj Ansible.
- Aplikaci můžete implementovat v Jazyce Java nebo Python s využitím již existujícího software, který vám usnadní implementaci jednotlivých modulů a jejich vzájemnou komunikaci.
- Rozhraní serverů Sequence, Shuffler a Server N implementujte jako REST API. Popis všech 3 rozhraní ve formátu OpenAPI bude součástí zdrojových kódů. Data přenášejte ve formátu JSON.
- Zdrojové kódy udržujte a publikujte v repozitáři [https://gitlab.kiv.zcu.cz/](https://gitlab.kiv.zcu.cz/)



# Implementace

K implementaci tohoto distribuovaného systému jsem použil Framework Spark v jazyce Java, který usnadňuje vytvoření jednoduchého API. Systém tvoří čtyři samostatné části, které vychází z obrázku v zadání. Každá část komunikuje se sousedem pomocí API, přes které se posílají zprávy ve formátu JSON. Každá zpráva dostane odpověď tvořená třídou Status, která obsahuje informace o úspěchu či neúspěchu daného úkonu. Popis API je podrobně popsán pomocí přiložené OpenAPI dokumentace.

## Klient

Klient má za úkol vygenerovat dostatek plateb, které se poté posílají do sekvenceru. Klient má dva způsoby na generování. První způsob je, že se přes API definuje, kolik plateb se má vygenerovat a klient je vytvoří a odešle. Druhý způsob je nekonečné vytváření plateb, dokud se uživatel nerozhodne tuto funkci zastavit.

Jednotlivé třídy v této službě jsou ClientAPIHandler, GeneratingPayment a SendingMessages. První třída ClientAPIHandler vytváří přístupové API pro volání funkcí. Třída GeneratingPayment vytváří jednotlivé platby a poslední třída slouží k odesílání plateb do sekvenceru.

## Sekvencer

Sekvencer má za úkol přiřadit jednotlivým platbám jejich ID, které určuje pořadí, v jakém se mají vykonávat. Následně se platby s tímto číslem posílají do Shuffleru.

Zde se nacházejí dvě třídy, GeneratingWholePayment a SendingMessages. První třída přidává do přijatých plateb jejich číslo sekvence, které se přiřadí podle toho kolikátá zpráva právě přišla a následně se předá platba druhé třídě, která ji odešle, jakmile to bude možné.

## Shuffler

Shuffler přijímá jednotlivé platby a po jejich zamíchání je náhodně odešle na jednotlivé banky. Tato služba má čtyři třídy, APIShuffler, PreparePayment, SendPayment a Shuffling. První třída opět pouze připravuje API a definuje reakci na jednotlivé zprávy. Druhá třída ve vteřinových intervalech odebírá příchozí platby z pole, zamíchá je a následně přidává do třídy SendPayment s jednotlivými bankami. Vteřinové rozestupy jsou tu kvůli čekání na nahromadění zpráv, aby bylo co zamíchat. Po takovémto přidání jsou platby ještě jednou zamíchány i s ostatními zprávami ve třídě SendPayment.

Ve třetí třídě SendPayment se tedy uchovávají jednotlivé instance bank, které udržují jednotlivé platby, které mají být odeslány na onu banku. Zde se také posílají platby, kdy se vybírá vždy náhodná banka, která ještě má platbu k odeslání.

Poslední třída Shuffling slouží jen k udržení příchozích plateb, ze sekvenceru před jejich zpracováním.

## Banka

Banka má jedinou činnost, průběžné provádění plateb, tak jak přichází. Mechanismus je proveden tak, že se jednotlivé platby ukládají do pole, které je následně seřazeno, protože je ale nutno udržet pořadí plateb, tak je zde proměnná, která slouží pro kontrolování, zda se provádí správná platba. Pro předcházení zbytečného seřazování pole se ještě před přidáním platby do pole zkontroluje, zda se nemá rovnou i vykonat, pokud ano tak se vykoná. Následně se zkontroluje, jestli tato platba nebyla blokující a nelze vykonat některé další platby.

## Core package

V tomto balíčku se nachází třídy, které používají všechny nebo aspoň většina služeb. Kvůli předcházení duplikace kódu, byly tyto funkčnosti přeneseny do vlastního balíčku.

### Enum Address
Zde jsou definovány jednotlivé API adresy jednotlivých služeb.

### Configuration
V této třídě se zpracovává konfigurace jednotlivých služeb. To znamená, kontrolování IP adres a portů a následně je i službě nastaví.

### Logger
Tato třída obsahuje obecný logger pro aplikaci.

### Status
Třída status slouží jako odpověď na každou zprávu služby. Obsahuje status výkonu akce a infomace o jejím průběhu.

### Bank
Tato třída slouží pro vytvoření instance jednotlivých bank. Obsahuje url, ip, port, metodu přístupu a následně seznam zbylých plateb.

### Enum ConMethod
Tento enum definuje HTTP metody.

### Connection
Zde je zabalená komunikace pomocí REST API. Pomocí této třídy se zprávy odesílají a přijímají.

### ClientPayment
Tato třída slouží pro vytvoření instancí platby z klienta.

### Payment
Třída, která se vytváří v sekvenceru.

### Enum PaymentType
Enum definující typy platby – CREDIT a DEBIT.

## Vagrant
Vagrantfile obsahuje konfiguraci virtuálních strojů. Je zde nadefinováno pět bank a poté shuffler, klient a sekvencer po jednom. Nejdříve se vytváří shuffler, poté pět bank, protože se mohou již připojit k vytvořenému shuffleru. Následuje sekvencer, který se již může také napojit na shuffler a nakonec se vytvoří klient, který se automaticky spustí a vytvoří prvních sto plateb, které se provedou.

Je zde vytvořen trigger, který spouští služby automaticky při zadání příkazu *vagrant up*.

## Vlastnosti řešení
Bankám stačí přidat adresu shuffleru a ony se samy připojí, jakmile je shuffler dostupný. Při jeho odpojení a znovu spuštění se banky automaticky do jedné minuty znovu registrují.

Shuffler ani žádná jiná služba si neuchovává platby pro budoucí platby, proto je nutné mít banky připojené dříve, než se začne platit. Pokud se tato podmínka nesplní narazí se na omezení aplikace.

## Omezení implementace
Hlavní omezení aplikace je, že sekvencer a banky musí být synchronizované, kvůli hlídání sekvence platby, která se má provést. Banky při spuštění očekávají první sekvenci rovno jedné, ale pokud

sekvencerem prošly nějaké platby, když banka ještě nebyla připojena tak se žádné platby v bance nikdy neprovedou, protože sekvencer už bude mít dávno jiné sekvenční číslo. V tomto případě je nutné restartovat jak banky, tak sekvencer, aby se obnovilo počítadlo plateb a vyčistily se platby i v bance.

# Použití aplikace

Musí být zprovozněn Vagrant s Virtualboxem. Poté stačí jít do složky s Vagrantfilem a zadat příkaz *vagrant up*. Tento příkaz vytvoří celý systém a spustí ho. Následně lze používat API popsané v OpenAPI dokumentaci, která je přiložená. Vagrantfile je uložen ve složce *out.*

## Adresy a parametry služeb

- Klient
  - Adresa: 10.0.1.40
  - Port: 8080
  - Parametry: IP adresa, port, IP sekvenceru, port sekvenceru, počet plateb (nepovinné)
    - Pokud počet plateb není zadán, spustí se nekonečné generování
  - Příklad spuštění: c*lient.jar 10.0.1.40 8080 10.0.1.30 8080 100*
- Banky
  - Adresa: 10.0.1.11-15
  - Port: 8080
  - Parametry: IP adresa, port, IP adresa shuffleru, port shuffleru
  - Příklad spuštění*: bank.jar 10.0.1.11 8080 10.0.1.20 8080*
- Shuffler
  - Adresa: 10.0.1.20
  - Port: 8080
  - Parametry: IP adresa, port
  - Příklad spuštění: *shuffler.jar 10.0.1.20 8080*
- Sekvencer
  - Adresa: 10.0.1.30
  - Port: 8080
  - Parametry: IP adresa, port, IP adresa shuffleru, port shuffleru
  - Příklad spuštění*: sequencer.jar 10.0.1.30 8080 10.0.1.20 8080*

# Závěr

Semestrální práce byla zajimavá, jelikož jsem se na ní naučil pracovat s Vagrantem, Sparkem a Kotlinem. Navíc jsem si zprovoznil WSL na Windows, ke kterému bych se také normálně nedostal. Kvůli problémům s přihlášením do školního Gitlabu jsem byl nucen použít normální Github. K zadání mám jedinou výtku a to, že by se mělo jednat spíš o systém s nějakým výpočtem například náhodně generovaným s násobením a dělením, aby opravdu záleželo na pořadí zpracování.

Projekt je dostupný na adrese: https://github.com/BobrZlosyn/sequencer