Ondřej Bureš a Phuong Dong Cu

**Jak simulaci spustit?** Stačí v IDEA spustit spring aplikaci, následovně až se aplikace načte stačí v prohlížeči zadat http://localhost:8080. Načte se úvodní stránka, ze které je možné se dostat na config stránku *(pro zobrazení i stavů jako zkažení potravin doporučujeme nastavení customerů pouze na 10 a dobu simulace na minimálně 20 a více týdnů).* Po odkliknutí startu program začne provádět simulaci. Následně po simulaci se načte stránka Reports, kde jsou kompletní výpisy, ze stránky se dá přesměrovat na stránky animals, crops, inovices, kde jsou separátně rozdělené výpisy pro větší přehlednost.

Systém jsme pojali po svém a vydali se experimentální cestou velké rozsáhle simulace, která při každém spuštění bude mít úplně jiný průběh. Snažili jsme se, aby věci, které chybí, byly nahrazeny komplexitou simulace. Pro detail komplexity uvedu například nezbytnost starat se o plodiny, krmit zvířata, kažení surovin, různé strategie nákupu, vznik a splátky dluhů, převážení zásob, zpracování zvířat tak, jak by tomu bylo i ve skutečnosti, doplňování krmiva farmáři, či množství vzniklých produktů a jejich typů.

Níže uvádíme, náš pohled a poznámky k naší realizaci funkčních požadavků

Funkční požadavky:

F1. Hlavní entity se kterými pracujeme je *Zemědělec nebo Farmář, Zpracovatel, Sklad, Prodejce, Distribuce, Zákazník, Potravina.*

*F2.* Jednotlivé strany (parties) si v ekosystému předávají potraviny a za potraviny/operace s nimy si platí penězy. Každá strana s potravinou provede některé operace. Každá operace má parametry - např. operace *Skladování* - parametry: délka 4 dny, teplota 12 stupňů, vlhkost …..

Operace provedená s určitými parametry a určitou Party = *Transakce*.

**Poznámka:** Transakce realizujeme sloučením s requesty, kdy vyvolání requestu začne provádět celou transakci operací. Pro zrušení transakce v našem případě standardně není důvod. Řeší se to pomocí toho, že se nestane nic nebo vznikne při převodu dluh, k pozdější úhradě.

F3. Systém je realizován pomocí velmi zjednodušené block chain platformy. Platforma je zjednodušená pro realizaci funkčních požadavků z tohoto seznamu. Každá Transakce s parametry a identifikací party, která je provedla, je zaznamenána a odkazuje se na předchozí Transakci. Již provedené Transakce a jejich parametry nelze zpětně upravovat. **Poznámka:** Již provedené transakce nelze zpětně upravovat, provedené operace jsou zaznamenány v logu nebo v exportovaných reportech. Nevyužíváme block chain platformu a a neodkazujeme se na předchozí transakce (historie je zaznamenána v reportu). Naše pojetí chainu je předávání si potravin mezi jednotlivými parties.

F4. V každém bodě životního cyklu potraviny lze získat věrohodné informace o tom, přes jaké parties potravina prošla a jaké operaci s ní provedly. Systém umožňuje tyto informace vygenerovat ve formě reportu (textový soubor)

**Poznámka:** Tyto informace jsou dohledatelné v reportu, pod konkrétním UUID. Jsou vygenerovány reporty pro zvířata, plodiny a faktury.

F5. Systém realizuje tzv. *Kanály.* Kanál spojuje parties, operace a má také svůj vlastní řetěz událostí. Příkladem je chanel na výměnu zeleniny, channel na maso, a channel pro platby. Každá operace má nastaveno v jakém channel se může provést, stejně tak Party má definováno v jakých kanálech je účastníkem (participant).

**Poznámka:** Jednotlivé kanály jsme pojali alternativně, kde slouží jako transakční brány pro komunikaci mezi jednotlivými parties procesního řetězce.

F6. Systém detekuje tzv. *double spending problém*. Tuto detekci odsimulujete na vámi zvoleném scénáři, např. kdy se zpracovatel snaží prodat tu samou potravinu 2x pod stejným certifikátem (dokládající původ potraviny).

F7. Systém detekuje pokus o zpětnou změnu do řetězce událostí. Tuto detekci odsimulujete na vámi zvoleném scénáři.

F8. Zpracování potraviny u party může být realizováno stavovým automatem - tedy potravina prochází různými stavy než může být předána další party. Mezi jednotlivými diskrétními kroky simulace (vysvětleno níže) může dojít pouze k jednomu přechodu mezi stavy.

F9. Do kanálů lze rozesílat požadavky, které obdrží Parties, které jsou účastníky v daných kanálech. Např. požadavek *Chtěl bych 150kg masa*. Parties na požadavky mohou a nemusí reagovat. Parties se mohou registrovat/odregistrovat buď do/z celého kanálu nebo na typ/z typu požadavku v rámci kanálu.

**Poznámka:** Máme vygenerováno četné množství různých parties. Parties mezi sebou komunikují přes channely. Situace je taková, že retailer, nemůže ani při zájmu prodat maso, které nemá. Jindy naopak pěstitel nemůže obhospodařit pole, pokud nemá uhrazené faktury, …

F10. Ze systému je možné vygenerovat následující reporty:

- *Parties report:* Jaké parties se podílely na procesování daných potravin, jak dlouho se u nich potraviny zdržely, jakou marži si party aplikovala na vstupní cenu
- *Food chain report* (viz funkční požadavek výše): pro potravinu se vypíše přes jaké parties prošla a jaké operace s jakými parametry s ní provedla
- *Security report:* Jaké parties se snažily podvrhnout původ potravin nebo provést tzv. Double spending a kolikrát.
- *Transaction report* - pro každý diskrétní krok ekosystému vypíše transakce které byly provedeny, kým a kolik měla každá party v ekosystému peněz a potravin

**Poznámka:** Transaction report negenerujeme, ale popis diskrétních kroků, je kompletně loggován

Nefunkční požadavky-úpravy:

Aplikace nemá GUI. Aplikace komunikuje s uživatelem pomocí command line, případně výpisů do souboru

**Poznámka:** Z důvodů dlouhé prokrastinační chvíle a chuti trochu experimentovat, jsme toto porušili a reporty je možné zobrazit pomocí springové web aplikace. Přes kterou se i potvrzují konfigurační podmínky. Konzolový proces je vidět v logu.

Použité design patterny:

- Observer
- Facade: v několikavrstvém odstínění. Z mainu aplikace je defaultně k dispozici pouze run simulation a print stats. Simulation pak má přístupy pouze k channelům a editačním metodám pro přidání uživatelem vytvořené entity.
- Dependency injection
- Visitor: zpracovává data z observeru
- Strategy: použitá na dvou místech pro customery a pro vytváření produktů
- State: provázaný celou aplikací, závislý na konkrétním UUID
- Iterator
- Factory
- Singleton: v podobě ve které to Kotlin umožňuje přes klíčové slovo object, většina singletonů se nachází ve statics
- Lambda funkce (map, reduce a filter nikoli), v datovém generátoru

Osobní shrnutí:

Jako standardně jsme po vyhotovění semestrálky přišli na spoustu věcí, které měli být řešeny jinak. Nyní bychom všechny typy fází potravin a produktů tahali jako jeden item a tím bychom i mohli smysluplněji využít State a spousta věcí by byla intuitivnější. Nicméně i tak jsme s výslednou prací spokojeni. V porovnání se semestrálkou z Javy je výsledná aplikace zase o úroveň dál. Vyloženě pochválit bych chtěl naši dvojici za experiment z Kotlinem, kdy jsme do toho šli opravdu s nulovou znalostí. Také i když je to věc navíc, které byla přímo zmíněna, že se nemá udělat, jsem rád, že jsme přidali i FE, protože mi osobně přijde, že se to tím o něco málo více blíží nějaké reálné aplikaci.