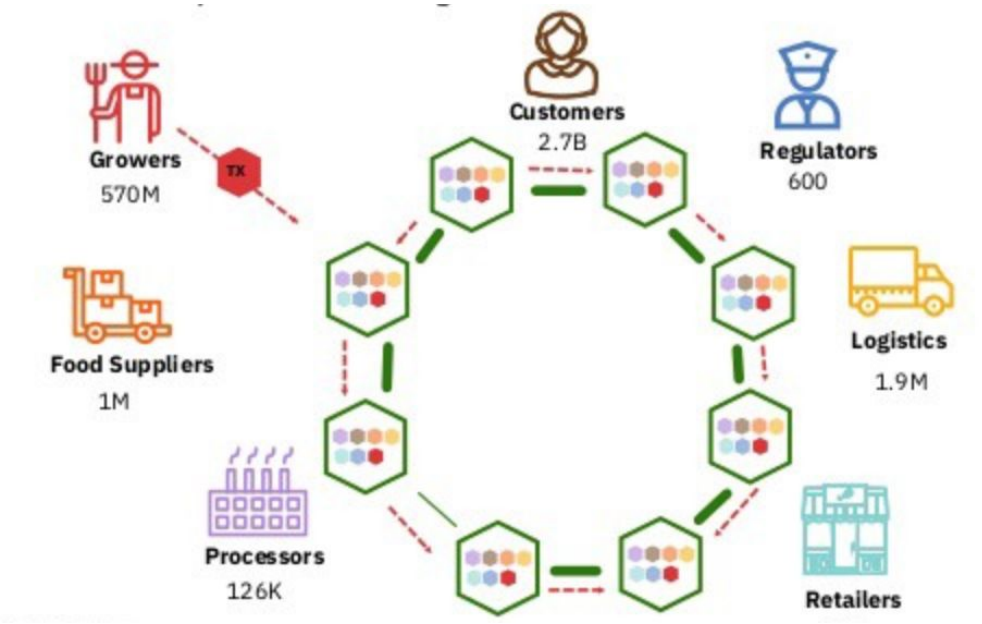


# Projekt: Food Chain

***Abstrakt***

*Implementovat zjednodušenou implementaci block chain a nad ní pak realizovat systém pro sledování toku potravin od jejich vypěstování, přes jejich procesování, skladování, distribuci, až po prodej zákazníkovi. Součástí celého ekosystému jsou Entity jako Zemědělec nebo Farmář, Zpracovatel, Sklad, Prodejce, Distribuce, Zákazník, které tvoří ekosystém bodů přes které tečou potraviny (např. Entita Potravina, Surovina). Plus použijte další vhodné entity, které se vám hodí pro realizaci funkčních požadavků.*

*Cílem ekosystému - řetězec vzniku, zpracování a prodeje potravin je zaručit, aby v každém bodě systému bylo možné dohledat, kterými body potravina prošla, co se s ní v každém bodu dělo a zároveň, aby nebylo možné tyto informace někým zmanipulovat. Tedy aby bylo tedy možné zabránit situacím, že maso z Polska je vydáváno s certifikátem masa z Rakouska, nešlo upravit dobu a teplotu při jaké bylo maso skladováno atd.*

***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*.

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.

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)
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).
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.
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

***Nefunkční požadavky***

- Není požadována autentizace ani autorizace
- Aplikace nemá GUI. Aplikace komunikuje s uživatelem pomocí command line, případně výpisů do souboru
- V systému neexistuje centrální autorita, přes kterou by se prováděly transakce. Nicméně aplikace má běžet pouze v jedné JVM, tedy není potřeba distribuované aplikace. Aplikaci není nutné řešit ve více threadech. Stačí v jednom threadu, kdy se postupně aplikují všechny transakce provedené jednotlivými parties v rámci aktuálního kroku diskrétní simulace (vysvětleno níže).
- Aplikaci pište tak, aby byly dobře skryté metody a proměnné, které nemají být dostupné ostatním třídám. Vygenerovný javadoc by měl mít co nejméně public metod a proměnných.
- Namísto PKI infrastruktury, public a private klíčů, šifrování/dešifrování pracujte pouze s abstrakcí - např. Jednoduchá java třída reprezentující klíč (PRK, PUK) uživatele,

metoda které např. podepíše obsah - ta pouze nastaví na obsahu flag podepsáno a jakým klíčem atd.
- Reporty jsou generovány do textového souboru
- Konfigurace ekosystému může být vytvořena přímo kódem z třídy nebo externího souboru (např json)

***Vhodné design patterny***

- Použijte alespoň 7 design patternů, které jsme probírali na přednáškách a dávají zde smysl

***Požadované výstupy***

- Realizace života takového ekosystému pro dvě různé konfigurace ekosystému (alespoň 10 parties, 5 potravin). Realizaci proveďte pomocí tzv. *Diskrétní simulace*:
    - Diskrétní krok 0:
        - *Party A vyšle požadavek*
        - *Party B reaguje na požadavek party A předáním potraviny na party C*
        - *Party D předává potravinu na party C a zároveň vysílá požadavek*
        - *Party E reaguje jako první na požadavek party D a začíná procesovat potravinu*
    - Diskrétní krok 1:
        - *Party C předává potravinu na party A*
        - *Party E předává potravinu na party C*
        - *Party E předává potravinu na party D => detekován double spending problém … atd*
- Design ve formě class diagramů a stručného popisu jak chcete úlohu realizovat
- Public API - Javadoc vygenerovaný pro funkce, kterými uživatel pracuje s vaším software - tedy public funkce ve třídách, které jsou v balíčcích (package) určených pro uživatele tohoto software
- Alespoň 20% tříd public API pokryto unit testy

***Hodnocení***

- Použití design patternů, čistota softwarového designu aplikace a public API (40%)
- Množství naimplementovaných funkčních požadavků (50%)
- Vlastní inovace - nový funkční požadavek, který jste přidali nebo zajímavý softwarový design (10%)

***Logistika***

- Úloha se vypracovává ve dvou - třech studentech (ideálně ze stejného cvičení), přičemž je požadováno, aby každá osoba napsala přibližně stejně komplexní část aplikace a rozuměla i částem, které nepsala (bude kontrolováno v gitu). Semestrální úloha může být znovu otevřena u zkoušky.
- Odevzdání je do konce semestru. Ideálně před vánoci, maximálně konec semestru.
- Minimálně dva týdny před odevzdáním práce je potřeba cvičícímu ukázat design vaší aplikace. Pokud by byl design špatný, došlo k odchýlení od zadání nebo naopak budete chtít poradit, tak bude konzultace provedena na cvičení, případně mimo cvičení po dohodě s cvičícím.