FoodChain

Tarnopolskyi Dmytro, Utemissov Adilbek, Shukatka Nazariy

Popis:

Cílem této semestrální práce bylo implementace systému pro sledování toku potravin od jejich vypěstování, přes jejich procesování, skladování, distribuci, až po prodej zákazníkovi na základě zjednodušené technologie blockchain.
Součástí celého ekosystému jsou Entity jako Zemědělec nebo Farmář, Zpracovatel, Sklad, Prodejce, Distribuce, Zákazník, které tvoří ekosystém bodů přes které tečou potraviny (např. Entita FoodItem). Kromě hlavních entit náš systém obsahuje další třídy, jako Milk, Meat, Cheese, Butter, Eggs, což jsou různé typy entity FoodItem.
Další třídou je Chanel, což je Kanál, který spojuje parties, operace a má také svůj vlastní řetěz událostí. Tato třída by měla umět přijmout požadavek, na který pak bude příslušná party reagovat.

API:

Uživatel si muže udělat konfiguraci podle sebe v souboru Configuration.java, kde nastaví budget každé party.

V třidě RequestController máme public static metodu validateCommand, která zpracovává požadavek uživatele.

JavaDoc:

JavaDoc celého systému máme vygenerovaný a uložený ve složce documentation/JavaDoc.

Jak to funguje?

FoodChain systém, který jsme naimplementovali, se skládá ze 6 party, které mohou mezi sebou komunikovat. Jsou to: Grower, FoodSupplier, Processors, Retailer, Logistics a Customer.
Komunikace mezi party probiha pomoci požadavků, které jsou popsané níže.
Každý požadavek musí projit validaci, která kontroluje spravnost požadavku a podminky požadavku, například jestli si zvolit požadavek typu “send 10 eggs from processors to retailer” a processors tento FoodItem u sebe nemá, tak vyhodí chybu “Party has not enough food!”. Jestli chcete FoodItem vratit v opačném směru, například “send 10 eggs from processors to foodsupplier” tak vyhodí jinou chybu “You can't do requests between those parties! Please try again”.
Jestli chcete program zastavit tak musíte napsat na příkazový řádek “exit”.
Z každého požadavku se tvoří transakce, která v sobě obsahuje informace odkud, kam a co bylo předáno. Tato informace se hashuje a přidává hash předchozí transakce. Každá transakce tvoří blok, ze kterých se skládá BlockChain. Tak že systém je realizován pomocí velmi zjednodušené block chain platformy.

Všechna funkční požadavky jsou podle nás splněné a program je úplně funkční.

Vlastní inovace:

1. Vytvořit si můžete kolik chcete FoodItemu najednou, což znamená, že není potřeba aby jeden FoodItem prošel celý systém před tím než se vytvoří další FoodItem.
2. Není předdefinovaný scénář protože aplikace se řidí pomoci příkazů, které uživatel piše do přikazové řadky. Je to vhodné na testování všech případů v rámci jednoho běhu aplikaci.

Komunikace s aplikaci:

Pro komunikace s aplikaci používame příkazový řádek, který se objeví po spuštění. Máme k dispozici 3 druhy požadavku:

1. GROW <count> <foodName> (patří groweru a vytvořenou potravinu pridáva jemu)
2. SEND <count> <foodName> FROM <partyName> TO <partyName> (posílání potraviny od jedné party k jiné)
3. <partyName> INFO (vypisuje informaci o party, právě jaký má počet každé potraviny a kolik má peněz)

, kde count - je počet litrů, kilogramů nebo kusů potraviny, kterou chceme vytvořit,
partyName - je název potraviny, kterou chceme vytvořit.

Seznam použitých návrhových vzorů:

- Strategy

Tento návrhový vzor je vhodným řešením v našem případě, kdy máme objekt, jež během své existence mění své vnitřní chování tak, že nabývá různých stavů jako GrowerState, FoodSuplierState atd, což říká o tom, na jaké etapě FoodChainu se nachází objekt. Na jeho stavu budou zaležet jeho parametry - cena, délka chránění atd.

- Factory

Pro vytvoření instance bude existovat rozhraní FoodItem s definovanou metodou pro vytvoření instance objektu. Vlastní implementace přípravy instance je pak skryta v metodě pro vytváření a nezatěžuje detaily vlastní kód, který instanci potřebuje.

- Bridge

Bridge je návrhový vzor pro strukturu objektů. Používá se, když chceme oddělit abstrakci od její implementace tak, aby se obě mohly měnit nezávisle. Klient posléze využije některou z implementací nepřímo prostřednictvím abstrakce. V našem případě jsme to použili při vytvaření instanci FoodItem typu Meat. Rozdělili jsme Meet na různé druhy, jako Pork, Chicken a Beef.

- Singleton

Singleton je návrhový vzor, který garantuje,že objekt je jedinečný a poskytuje globální přístup k němu. V našem případě je použit tento vzor na vytvoření instance každé party. Každá party může objekt FoodItem přijímat, zpracovávat a odesílat mezi sebou. Tento vzor garantuje existence a práce s jedinečným exemplářem.

- Observer

Návrhový vzor Observer ukazuje, že při změně stavu globálního objektu FoodItem (přechodu z jedné entity na jinou) může reagovat na tyto změny každá entita. Při změně stavu FoodItem bude informovat třídy které jsou spojené s ním, volá se metodu tyto třídy.

- Template method (šablonová metoda)

Návrhový vzor Template Method definuje kostru algoritmu, tedy jeho jednotlivé kroky. Potomci poté kroky implementují a představují zaměnitelné algoritmy. Použili jsme tento návrhový vzor při implementaci Strategy, když máme abstraktní třídu Strategy s metodou changeToNextState() a každa strategie, jako GrowerStrategy, FoodSupplierStrategy atd. tuto metodu implementuje u sebe.

- State pattern (stavový návrhový vzor)

Návrhový vzor řeší problém jak změnit chování objektu, jestliže se změní jeho vnitřní stav. Po změně chování se objekt jeví jako instance jiné třídy. Realizovali jsme to v podobě LogisticsStates, tzn, že když se potravina (FoodItem) nachází na etapě Logistics tak může nabývat jeden z třech stavu: OrderedState (defaultní stav), BoxedState (FoodItem byl zabalen do krabice) nevo DeliveredState(potravina se poslala zákazníkovi).