Relatório

PROJECTO SEGURANÇA 2018/2019 (SEGURANCA2018-P2G1)

CARLOS OLIVEIRA 88702

DANIEL MADAÍL 77729

Introdução:

Para este trabalho, foi pedida a construção de um sistema de leilões, composto por clientes, e dois servidores. Estes dois servidores serviriam para administrar e servir de repositório para as transações necessárias para a execução de vários leilões.
O objectivo deste trbalho foi a implementação de medidas de segurança variadas, desde cifragem e decifragem de campos de informação nos pacotes de dados transferidos entre os componentes do sistema, bem como durante o seu armazenamento; Assinaturas digitais dos dados que relatavam as transações e o percurso que estes tomaram durante as trocas de dados entre tais sistemas; bem como Autenticação do utilizador perante os servidores e dos servidores perante o utilizador.

Para este trabalho, foram tomadas várias decisões que, ultimamente, afectaram e comprometeram o trabalho final. Estas decisões e suas consequências tornar-se-ão explícitas ao longo do relatório.

Numa primeira parte, explicamos como funciona o nosso projecto e a sua estrutura, bem como o fluxo de informação dentro e entre os componentes do sistema.

De seguida, uma parte que explica os processos e resultados das decisões tomadas ao longo da realização do trabalho.

Decisões tomadas

Linguagem a usar:

Como qualquer projecto, o planeamento é um primeiro passo importante para garantir a conclusão atempada e correcta deste. Tal como é referido inúmeras vezes no contexto educacional e empresarial, a linguagem a usar num projecto tem de ser a linguagem mais adequada para resolver o problema. É também referido que, em caso de dúvida, uma linguagem com a qual se esteja mais familiar irá permitir que se expedite o trabalho, ao eliminar o tempo de aprendizagem necessário para a linguagem em específico.
Por estarmos mais confortáveis a usar *Java*, decidimos que esta seria a linguagem mais adequada para um projecto de complexidade superior como este.

Para esta decisão pesaram também factores a favor como a segurança de *Java* (devido ao uso da *JVM* e maior adopção pela comunidade informática), capacidade de programar de forma orientada a objectos (que se revelou bastante importante, devido à complexidade do sistema a desenvolver) e a maior quantidade de recursos disponíveis online, bem como uma maior adopção por parte dos grupos da unidade curricular, podendo assim usufruir de uma maior base de ajuda para casos específicos de *Java.*

Contudo, outros factores também foram importantes para esta escolha, neste caso negativos, como é o caso da disponibilidade de IDE's para *Java*. Esta linguagem dispõem de três IDE's que são relevantes. NetBeans(agora pertencente à fundação Apache, e não se encontra actualizada e estável o suficiente para um projecto desta envergadura), Eclipse (que já mostra a sua idade e, por recomendação de terceiros, não aparentava ser a melhor escolha) e IntelliJ IDEA (que é brilhante no que toca a ajudas de pequena dimensão e gestão do projecto e, aparentemente, a melhor opção de entre os três.

Estando decidido assim a linguagem a usar, decidimos então optar por usar o IntelliJ IDEA que se tornou num grande obstáculo ao avanço do projecto quando este estava numa fase crítica.

IDE (Integrated Development Environment)

Como não existe bom e barato, segundo ditados populares, as nossas dúvidas caíram por completo quando este software se mostrou incompatível com o percurso que estávamos a tomar no projecto – webapps e applets com API's REST.

Apenas a meio do projecto, enquanto estávamos a passar do papel para a realidade, é que o IDE nos informou que grande parte das funcionalidades que queríamos usar, parte do Java EE, se encontrava disponível apenas para utilizadores pagos. Com uma anualidade de cerca de 500€, rapidamente se tornou óbvio que teríamos de usar outro IDE para continuar o desenvolvimento do projecto. O problema instalou-se quando nos apercebemos que as implementações de Java EE na concorrência não era sustentável e o projecto era difícil de importar para estes.
Decidimos não investir mais tempo na procura de um outro IDE, e manter o IDEA por razões logísticas e de *workflow*. Como nos permitia produzir mais código de forma consistente por unidade de tempo, não o quisemos abandonar.

Transmissão de Dados

Numa primeira fase, queríamos dispor dos serviços UDP, que se podiam montar com o recurso ao Java EE. Mas como referido anteriormente, não foi uma escolha que se concretizasse.

Decidimos então mudar para um conceito que é sólido e fácil de implementar em Java: sockets. Tentámos usar websockets, mas mais uma vez, era exclusivo da versão paga do nosso IDE.

Decidimos, depois de descobrirmos que a Apache disponibiliza bibliotecas para esta linguagem capazes de criar servidores HTTP e HTTPS, optar pelo uso de GETs e POSTs para a transmissão de dados e comandos entre os vários componentes do sistema.

O uso de HTTP para a transmissão de dados, apesar de ser um conceito com o qual estávamos apreensivos dado ser novidade para nós, provou ser relativamente fácil de implementar, sendo que os maiores problemas prenderam-se com o tipo de informação que estava a ser dada a estes métodos e não tanto pela transmissão destes.

Numa fase inicial, como não estávamos a conseguir usar os métodos GET, decidimos usar apenas POSTs para envio de informação por forma acelerar o processo de desenvolvimento. Mais tarde, conseguímos usar os GETs para comunicação de comandos ou para pedidos de listagens.

Concordámos que foi uma boa decisão, não só pela facilidade de implementação, como também, por ser a única alternativa dada a situação em que nos encontrávamos com os IDEs.

Um outro ponto bastante importante para a escolha destes métodos é, como deve ser óbvio, a interoperabilidade entre sistemas. Isto significa que, caso fosse necessário usar outra linguagem, ambiente de desenvolvimento, sistema operativo ou hardware diferente, era possível interligar os vários componentes e aproveitar código. Num contexto educacional é bastante útil, mas acreditamos que num contexto mais concreto e real, seria uma mais valia enorme.

Segurança

Como não foi possível a conclusão atempada do projecto, não foi possível implementar as funções básicas do sistema de leilões, pelo que não foi possível por isso, implementar muitas medidas de segurança. As medidas implementadas reduzem-se a cifragem e decifragem dos campos sensíveis, dependendo das definições de cada leilão, presentes nas *Bids*. Foi também implementado código para a leitura dos cartões de cidadão portugueses, que retira o número de documento para ser usado ao longo do programa como a identidade do autor do leilão. Contudo, como não usámos os certificados e os sistemas de autenticação presentes no cartão e bibliotecas correspondentes, acabámos por não usar estes para a assinatura das transações dos clientes. O uso de HTTPS foi preparado, tendo sido criados os certificados necessários por parte do servidor (e que se encontram na raíz do projecto), bem como a entidade certificadora. Contudo, não foi possível converter atempadamente estes certificados em *keystores*, compatíveis com a linguagem *Java* e os métodos que esta disponibiliza de origem.

Blockchain

Quanto à *blockchain*, esta foi preparada já de início como forma de poupar tempo de desenvolvimento, para ser a base do repositório. Decidímos optar por usar *ArrayList<Block>* para armazenamento de cada uma. É um sistema de armazenamento por listas que é semelhante ao princípio do blockchain, e é fácil de usar. Dado que, muitos dos casos de utilização, não requeriam acessos aleatórios ao esquema de dados, foi tomada esta decião. Como quase todos os casos de uso necessitavam ou de uma iteração completa da blockchain ou da leitura apenas do bloco inicial ou final, esta tornou-se a melhor escolha.

Para armazenar as blockchains, unidade de cada leilão, foi escolhido o uso de dois *HashMap<String, ArrayList<Block>>.* Um para os leilões em aberto, a decorrer, e outro para os leilões que terminaram, e que possuem os dados de cada autor de leilão*, leilão* e das *bids*. Decidímos usar HashMaps porque estes permitem uma pesquisa por um leilão específico de forma aleatória, pois a chave usada para cada um dos valores (blockchains) era o identificador único de cada leilão.

JSON

Para permitir uma troca de objectos entre os componentes, rapidamente se tornou óbvio que o uso de JSON iria tornar tudo mais acessível. Decidimos usar e concordamos que foi uma boa maneira de enviar mensagens estruturadas de e para os servidores. Uma das vantagens é a existência de métodos que tornavam possível a serialização e desserialização de dados de forma trivial e análoga ao que se usava para objectos Java. A possibilidade de transformar um objecto proprietário do nosso projecto, num objecto JSON e depois passar para uma String e reverter de volta para um objecto é poderosa. Como a implementação de métodos para trabalhar com este formato seria possivelmente mais trabalhosa que o próprio projecto, decidimos usar uma biblioteca de JSON, OpenSource e gratuita da Google – GSON. Isto é importante, pois significa suporte, e maior segurança dado ser não só opensource mas também um *standard.* Foi escolhida também por funcionar muito bem com *JavaScript*, linguagem que ainda estava em possibilidade de ser escolhido para a interface do Cliente.

Swing

Para o desenvolvimento da interface gráfica do utilizador Cliente, decidimos que o uso da ferramenta incluída para tal, no IDE que escolhemos, que seria a melhor opção – devido à integração rápida e simples com o nosso projecto. O software incluído no IDE foi rapidamente desmascarado como nada mais que um instrumento de turtura. O grau de personalização das interfaces é atróz e de é, de longe, a pior experiência que alguma vez tivemos com software do género. Este coloca o editor de GUIs do NetBeans num pedestal.
A decisão de incluir um GUI no programa de cliente foi tomada pois era uma maneira mais visual e simples de introdução de dados, bem como de *debug*. Como a lógica do programa era toda executada no ficheiro principal, sem nenhuma adaptação especial para trabalhar em sintonia com a interface, o seu impacto no esforço inicial de desenvolvimento foi mínimo. Contudo, conforme a complexidade do projecto aumentava, também aumentava a necessidade de uma interface gráfica para a introdução de dados de forma rápida e eficaz, bem como para a leitura destes.

UUID

Uma menção honrosa é feita para a libraria *java.util.UUID*, que permitiu a criação de identificadores únicos e universais de cada um dos leilões e de cada uma das *Bids.* Esta é uma escolha que é fácil, pois é um método popular e com razão. A possibilidade de colisão é negligível, e não permite que se descubram ID's dos objectos por estes identificados devido à sua aleatoriedade.

Cartão de Cidadão

As ferramentas disponibilizadas pelo governo para o uso do Cartão de Cidadão provaram-se úteis e fáceis de implementar. Em apenas 1 hora, foi possível colocar a identificação do utilizador do Cartão de Cidadão nas transações efectuadas pelo Cliente. De salientar, que 55 destes minutos prenderam-se com o facto de que não é explícito o local onde se devem colocar as bibliotecas referidas e, apesar da definição do *java.library.paths* em todos os locais possíveis tanto manualmente como através do IDE, recorrendo a variáveis de ambiente de trabalho do sistema operativo, variáveis de execução do *Java* e a definição de múltiplas maneiras diferentes deste caminho em código executado em *Runtime.*

Sistema Operativo

Uma pequena nota tem de ser tomada quanto à escolha do sistema operativo para o ambiente de desenvolvimento: A biblioteca da Sun do PKCS#11 existe em quase todos os sistemas operativos, menos no mais usado a nível mundial: Microsoft Windows de 64 bits. Para contrariar isto é possível o uso limitado da biblioteca de 32 bits para Windows ou o uso de um *Wrapper.*

Estrutura do projecto

Este projecto contempla 3 módulos, cada um com a sua função no sistema de leilões.

Estes são: O Cliente, o *Manager* ou Administrador e o Repositório.

O objectivo deste sistema era permitir a realização de leilões de forma segura.

Cliente

O cliente é o programa que possui Interface Gráfica, e é o programa com o qual mais se interage. Neste programa é possível listar todos os leilões que estejam a decorrer ou que tenham decorrido, bem como os dados que neles se encontram, como valores de bids e identificação dos autores. É possível criar os leilões e as *Bids.*

Este programa implementa as funções, descritas a mais pormenor nos anexos, de enviar pedidos POST e GET para ambos os servidores conforme o que é pedido pelo utilizador, através da interface gráfica deste.

Manager

O *Manager* é um módulo que ajuda na gestão e verificação das transações relacionadas com os leilões. Este é responsável por criar os leilões tendo em conta as definições criadas pelos clientes e de passar as informações necessárias ao Repositório para iniciar um leilão. Esta aplicação cria e pára os leilões (a mando ou quando o tempo de cada uma acaba), executa a verificação das *Bids* que o Repositório lhe manda para serem verificadas antes da sua adição à blockchain correspondente. Quando um leilão é criado, esta aplicação lê as definições de cada leilão antes de ser criado (como por exemplo, os campos a serem cifrados antes de serem submetidos ao repositório) tanto como a cifragem dos campos relevantes para cada tipo de leilão de cada *Bid.*

Repositório

Aqui são guardadas todas as transações válidas dos leilões. Quando um cliente cria uma *Bid,* é este o ponto de chegada. Com recurso a uma *blockchain*, o repositório garante assim confiança nos dados ali presentes, para que todos os clientes que participem possam atestar de tal. O uso desta técnica, permite também, em conjunto com *proof-of-work* e *cryptopuzzles*, manter um custo não completamente negligível para possíveis atacantes. Como se trata apenas de uma aplicação virada para o utilizador, estes criptopuzzles são de dificuldade acessível, não podendo ser comparados com os usados em moedas virtuais.

Pontos focais dos módulos.

Seguem-se alguns exemplos de código e esquemas que foram implementados e que são de interesse para este relatório:

```
public AuctionClient() {
    POSTButton.addActionListener((e) → {

            Gson gson = new Gson();

            Settings settings = new Settings(
                    Double.parseDouble(settingsValue.getText()),
                    Double.parseDouble(settingsMinimum.getText()),
                    Double.parseDouble(settingsMaximum.getText()),
                    settingsBidder.isSelected(),
                    settingsBidValue.isSelected(),
                    Integer.parseInt(settingsTime.getText()),
                    settingsAuthor.isSelected()
            );
            String auctionSettings = gson.toJson(settings);
            Auction auction = new Auction( id: "tbd",
                    ccNumber,
                    auctionProduct.getText(),
                    auctionDescription.getText(),
                    auctionSettings);

            String stringAuction = gson.toJson(auction);
            System.out.println(stringAuction);
            Message message = new Message( type: "Auction",  action: "addAuction", stringAuction);
            String stringMessage = gson.toJson(message);
            System.out.println(stringMessage);
            try {
                String result = postMessage(stringMessage, ManagerPort);
                System.out.println(result);
                editorPane1.setText(editorPane1.getText() + "\n\n" + result);
            } catch (Exception en) {
            }

    });
```

Figura 1 - Criação de um leilão, lado do Cliente

Na Figura 1 é possível observar o bloco de código responsável por criar um Objecto da classe Auction. Numa primeira parte, são obtidos os valores introduzidos na GUI e colocados nas variáveis respectivas para a criação do objecto da classe Settings. Este objecto é depois serializado usando JSON para ser introduzido no objecto Auction. Note-se aqui, que o id deste auction é “tbd” (To Be Determined). Escolhemos aplicar o código para identificar o leilão (e as licitações também) no módulo de administração. Primeiramente, porque o objectivo inicial era ter uma identificação numérica simples e, que para tal, necessitava ser executada toda no mesmo módulo para que pudesse ser única. Esta ideia manteve-se e não foi revertida pois se a identidade for dada num módulo mais seguro que o Cliente, quer por obfuscação quer por distância física ou lógica. Como estes dados não estão cifrados nem assinados digitalmente, esta foi uma medida para impedir que possíveis Clientes simplesmente modificassem os binários do seu programa antes ou durante a sua execução para obter vantagem. Outro ponto a salientar é o uso de um Objecto Mensagem. Isto permitiu troca de mensagens ainda mais estruturada, visto que não são só os dados (leilões e

licitações) que se encontram estruturados mas também os comandos que os acompanham e que, sozinhos, também formam os comandos GET…

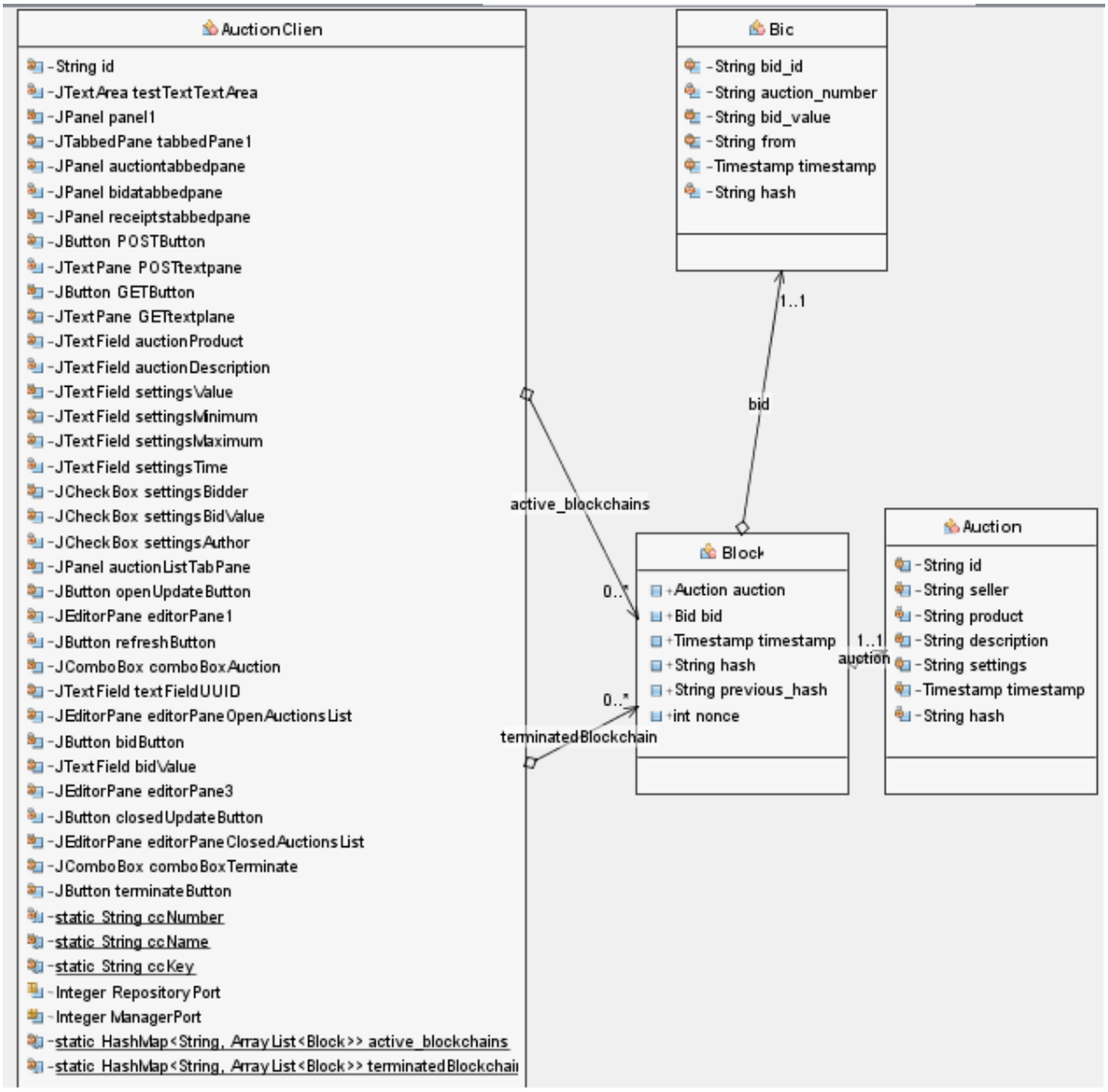
Um outro ponto que gostaríamos de salientar é o facto de, para eliminarmos necessidade de pedidos de informação aos servidores, por parte dos clientes, os repositórios são copiados para os restantes módulos e actualizados conforme necessário. Os repositórios são transferidos por inteiro e, para além de servir para distribuição dos repositórios para maior confiança, também libertam o módulo de repositório visto que as operações de abertura de cada camada dos repositórios (De repositório, para *Blockchain*, para bloco, para leilão, para poder ler as *Settings*) são executadas no módulo que as requiriu, evitando assim conflitos de recursos no repositório e maior velocidade.

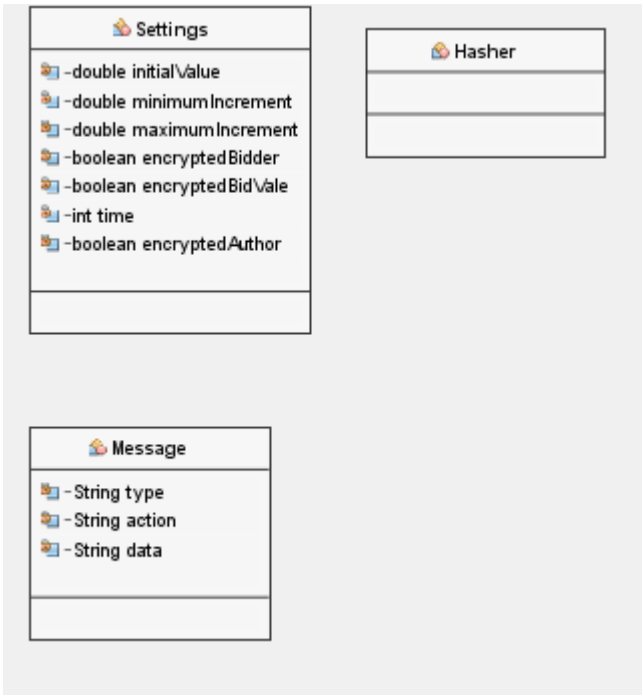
```
public Auction(String id, String seller, String product, String description, String settings)
{
    this.id = id;
    this.seller = seller;
    this.product = product;
    this.description = description;
    this.settings = settings;
    this.timestamp = new Timestamp(System.currentTimeMillis()); // TO DO: CHANGE THIS TO THE
    this.hash = calculateHash();
}
public String calculateHash(){
    String output = Hasher.hashSHA256( input: id+seller+product+settings+
            (timestamp));
    return output;
}
```

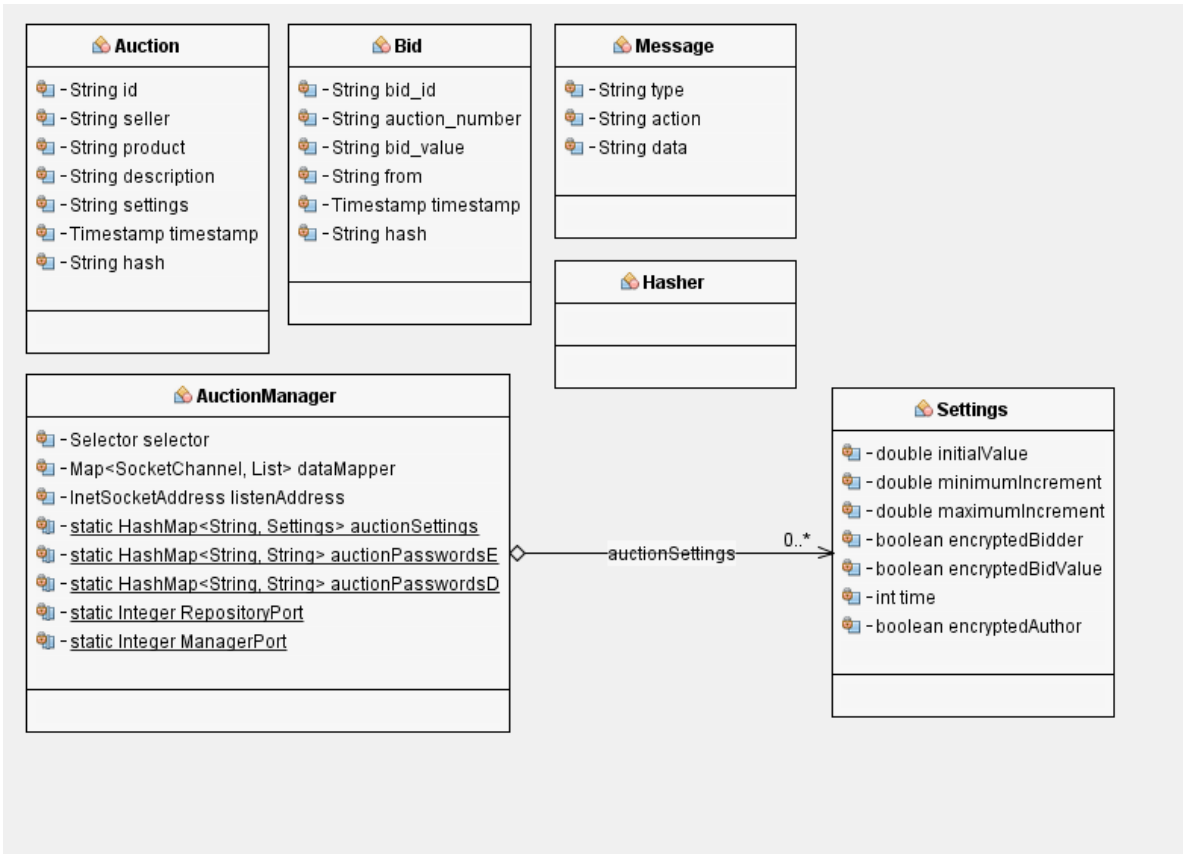
Figura 2 - Código do Objecto Auction

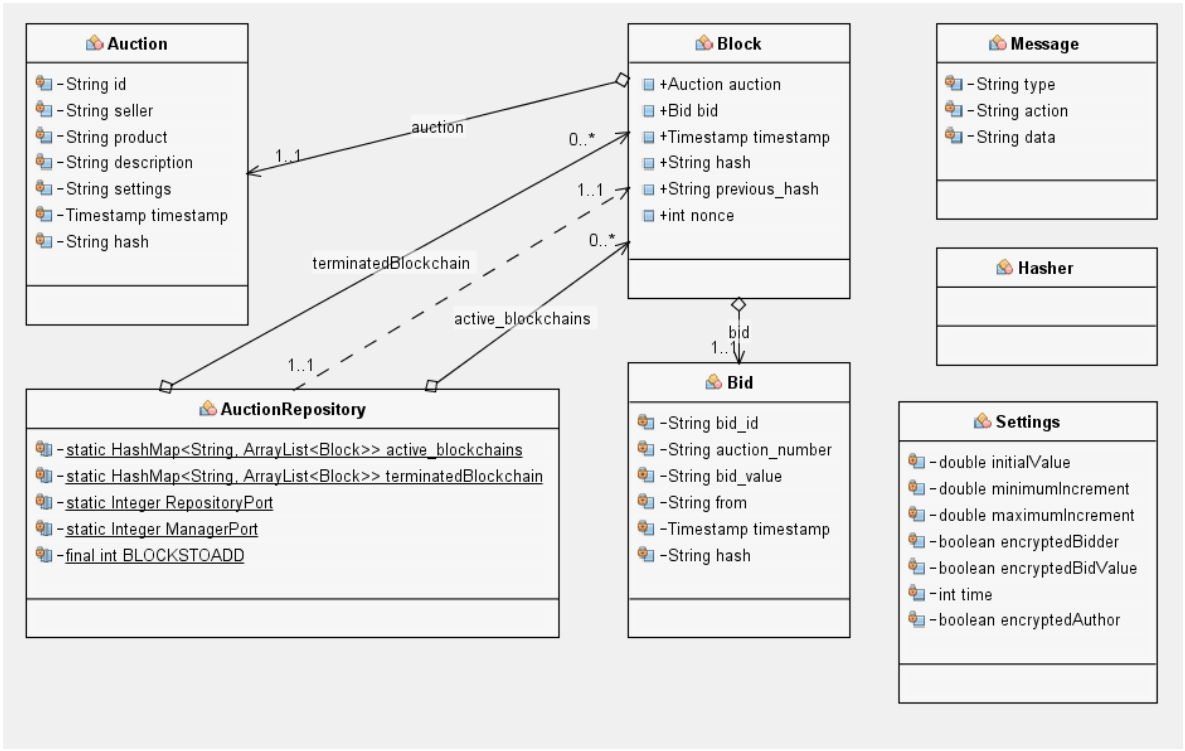
Na Figura 2 pode-se ver que para aumentar a segurança ligeiramente, cada objecto Auction e Bid é finalizado com um *hash*. Isto foi implementado por motivos de segurança, enquanto o transporte e assinatura digital das transações não era efectuado.

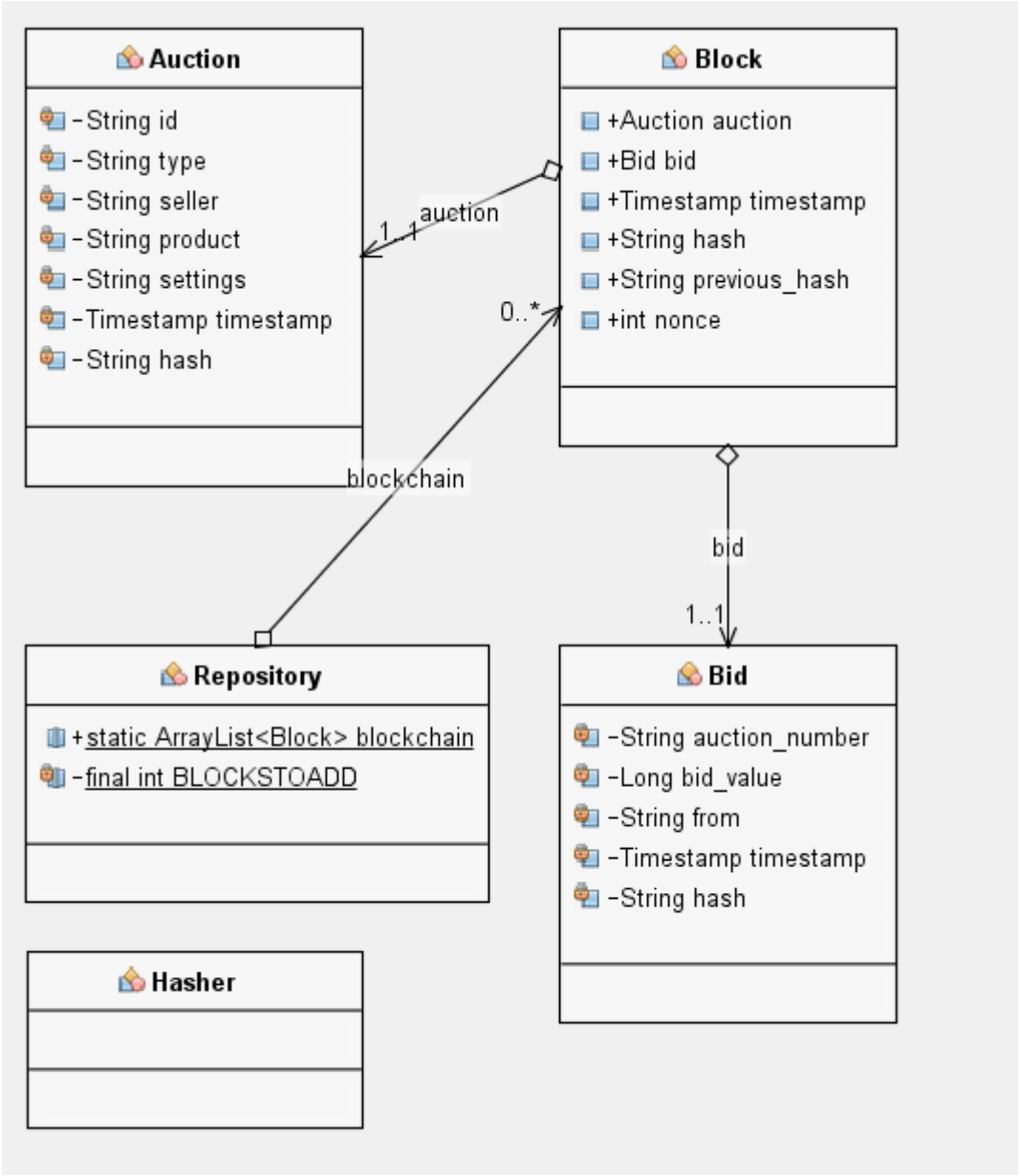
Seguem-se, em anexo, os diagramas de classe dos módulos.




Settings
-double initialValue
-double minimumIncrement
-double maximumIncrement
-boolean encryptedBidder
-boolean encryptedBidVale
-int time
-boolean encryptedAuthor
Hasher
Message
-String type
-String action
-String data


Auction
-String id
-String seller
-String product
-String description
-String settings
-Timestamp timestamp
-String hash
Bid
-String bid_id
-String auction_number
-String bid_value
-String from
-Timestamp timestamp
-String hash
Message
-String type
-String action
-String data
Hasher
AuctionManager
-Selector selector
-Map<SocketChannel, List> dataMapper
-InetSocketAddress listenAddress
-static HashMap<String, Settings> auctionSettings
-static HashMap<String, String> auctionPasswordsE
-static HashMap<String, String> auctionPasswordsD
-static Integer RepositoryPort
-static Integer ManagerPort
auctionSettings
0..*
Settings
-double initialValue
-double minimumIncrement
-double maximumIncrement
-boolean encryptedBidder
-boolean encryptedBidValue
-int time
-boolean encryptedAuthor


Auction
-String id
-String seller
-String product
-String description
-String settings
-Timestamp timestamp
-String hash
Block
+Auction auction
+Bid bid
+Timestamp timestamp
+String hash
+String previous_hash
+int nonce
Message
-String type
-String action
-String data
Hasher
auction
1..1
0..*
1..1
terminatedBlockchain
0..*
active_blockchains
1..1
bid
1..1
AuctionRepository
-static HashMap<String, ArrayList<Block>> active_blockchains
-static HashMap<String, ArrayList<Block>> terminatedBlockchain
-static Integer RepositoryPort
-static Integer ManagerPort
-final int BLOCKSTOADD
Bid
-String bid_id
-String auction_number
-String bid_value
-String from
-Timestamp timestamp
-String hash
Settings
-double initialValue
-double minimumIncrement
-double maximumIncrement
-boolean encryptedBidder
-boolean encryptedBidValue
-int time
-boolean encryptedAuthor


Auction
-String id
-String type
-String seller
-String product
-String settings
-Timestamp timestamp
-String hash
Block
+Auction auction
+Bid bid
+Timestamp timestamp
+String hash
+String previous_hash
+int nonce
auction
1..1
0..*
blockchain
bid
1..1
Repository
+static ArrayList<Block> blockchain
-final int BLOCKSTOADD
Bid
-String auction_number
-Long bid_value
-String from
-Timestamp timestamp
-String hash
Hasher