

**MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO**

**1º Ten RAPHAEL ALBERTO MOORE MACEDO
Asp R2 JOÃO FELIPE NASCIMENTO MATTOS
Asp R2 VICTOR CESAR MILARÉ CORRÊA DE ANDRADE**

SIMULADOR DE INTERAÇÃO EM SISTEMAS DE BLOCKCHAIN

**Rio de Janeiro
2019**

INSTITUTO MILITAR DE ENGENHARIA

**1º Ten RAPHAEL ALBERTO MOORE MACEDO
Asp R2 JOÃO FELIPE NASCIMENTO MATTOS
Asp R2 VICTOR CESAR MILARÉ CORRÊA DE ANDRADE**

**SIMULADOR DE INTERAÇÃO EM SISTEMAS DE
BLOCKCHAIN**

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Prof. Leandro de Mattos Ferreira - M.Sc.

Rio de Janeiro
2019

c2019

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 - Praia Vermelha
Rio de Janeiro - RJ CEP 22290-270

Este exemplar é de propriedade do Instituto Militar de Engenharia, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es) e do(s) orientador(es).

Macedo, Raphael Alberto Moore
Simulador de interação em sistemas de blockchain
/ Raphael Alberto Moore Macedo, João Felipe Nascimento Mattos, Victor Cesar Milaré Corrêa de Andrade, orientado por Leandro de Mattos Ferreira - Rio de Janeiro: Instituto Militar de Engenharia, 2019.

71p.: il.

Projeto de Fim de Curso (graduação) - Instituto Militar de Engenharia, Rio de Janeiro, 2019.

1. Curso de Graduação em Engenharia de Computação - projeto de fim de curso. 1. Simulador. 2. Blockchain. 3. Django. I. Ferreira, Leandro de Mattos . II. Título. III. Instituto Militar de Engenharia.

INSTITUTO MILITAR DE ENGENHARIA

1º Ten RAPHAEL ALBERTO MOORE MACEDO
Asp R2 JOÃO FELIPE NASCIMENTO MATTOS
Asp R2 VICTOR CESAR MILARÉ CORRÊA DE ANDRADE

SIMULADOR DE INTERAÇÃO EM SISTEMAS DE
BLOCKCHAIN


Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Prof. Leandro de Mattos Ferreira - M.Sc.

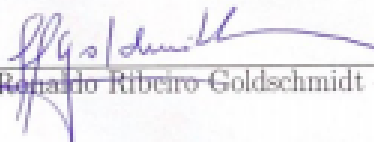
Aprovado em 08 de Outubro de 2019 pela seguinte Banca Examinadora:



Prof. Leandro de Mattos Ferreira - M.Sc. do IME - Presidente



Prof. José Antonio Moreira Xexéo - D.Sc. do IME



Prof. Ronaldo Ribeiro Goldschmidt - D.Sc. do IME

Rio de Janeiro
2019

SUMÁRIO

LISTA DE ILUSTRAÇÕES	6
1 INTRODUÇÃO	8
1.1 Contextualização	8
1.2 Motivação	9
1.3 Objetivos do Projeto	10
1.4 Plano de Trabalho	11
1.5 Justificativa	11
1.6 Metodologia	12
1.7 Estrutura do texto	13
2 FUNDAMENTAÇÃO TEÓRICA	14
2.1 Blockchain	14
2.1.1 Mineração	16
2.2 Modelagem do processo de mineração	17
2.2.1 Ruína do jogador e gasto duplo	18
3 MODELAGEM	20
3.1 Requisitos do sistema	20
3.2 Detalhamento dos requisitos	21
3.2.1 Inicialização do sistema	21
3.2.2 Análise de recompensa da rede	22
3.2.3 Análise do consenso da rede	22
3.2.4 Consulta ao sistema	22
3.2.5 Exportação das configurações	22
3.2.6 Carregamento de configurações	23
3.3 Diagrama de classes	23
4 ARQUITETURA DO PROJETO	25
4.1 Versões do programa	25
4.2 Linguagem de programação	25
4.2.1 Django	26
4.2.2 Numpy	26
4.2.3 Javascript	27

5	IMPLEMENTAÇÃO DO PROJETO	28
5.1	Model	28
5.2	View	29
5.3	Controller	31
6	GUIA DE USUÁRIO E RESULTADOS	34
6.1	Tela inicial de configuração	34
6.2	Tela de simulação	35
6.2.1	Passagem de tempo e eventos	35
6.2.2	Salvamento	36
6.2.2.1	Painel de simulação	36
6.2.3	Gráfico	37
7	CONCLUSÃO	39
7.1	Projeto	39
7.2	Trabalhos Futuros	39
8	REFERÊNCIAS BIBLIOGRÁFICAS	41
9	APÊNDICES	43
9.1	APÊNDICE 1: Código Model	44
9.2	APÊNDICE 2: Código Controller	47
9.3	APÊNDICE 3: Código View	57
9.3.1	index.html	57
9.3.2	start.html	62
9.3.3	getContent.js	67

LISTA DE ILUSTRAÇÕES

FIG.1.1	Arquitetura MVC	12
FIG.2.1	Esquema de Assinatura Digital no Bitcoin	14
FIG.2.2	Cadeia de blocos válidos	16
FIG.3.1	Diagrama de classes	24
FIG.5.1	Interface do programa	30
FIG.5.2	Resultado de uma simulação	31
FIG.6.1	Botões de passagem de tempo e evento	36
FIG.6.2	Botões de salvamento	36
FIG.6.3	Painel de informações	37
FIG.6.4	Gráfico	37
FIG.6.5	Gráfico com lucro negativo	38

RESUMO

O presente trabalho de conclusão de curso de graduação objetivou empregar conceitos e técnicas aprendidos no decorrer do curso para desenvolvimento de um Simulador de Interação em Sistemas de Blockchain. São explorados e explicados os principais conceitos que norteiam o uso e aplicações dessa inovadora tecnologia, além da modelagem de classes e discussão das principais decisões de projeto tomadas ao longo do desenvolvimento.

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Contratos, registros e transações constituem a base de organização da sociedade civil, pois representam a formalização acordos entre partes envolvidas nos mais diversos processos, de transações financeiras a receituários médicos. O crescimento significativo da população a partir do século XIX aliado a um ritmo intenso de desenvolvimento tecnológico aumentou consideravelmente o número de interações entre atuantes do sistema, tornando necessária a criação de métodos para lidar com o volume de dados.

O desenvolvimento de redes de comunicação e da Internet também afetou o compartilhamento de dados, facilitando o contato e diminuindo consideravelmente o tempo gasto para troca de informações. Além disso, possibilitou o compartilhamento de bancos de dados auxiliando a aproximação e o relacionamento entre as diversas partes.

O estabelecimento de contratos e transações entre quaisquer duas partes em uma rede, a despeito da capacidade de armazenar e trocar informações por meio digital de maneira eficiente, ficou por muito tempo dependente da existência de uma terceira parte garantidora de consenso e geradora de confiança entre as partes interessadas. Esse cenário só se alterou a partir do momento que (NAKAMOTO, 2008) propôs um meio digital de pagamentos intitulado Bitcoin que dispensa a necessidade de uma terceira parte reguladora. Isso foi possível através da integração dos desenvolvimentos em estruturas de dados, redes e sistemas distribuídos com técnicas criptográficas para conseguir garantir consenso, fazendo dos usuários os próprios garantidores do sistema.

Devido a essa mudança de paradigma no modelo de relacionamento e confiança potencialmente proporcionado por essa tecnologia observou-se um grande pico de interesse nas possíveis aplicações. Como efeito, a tecnologia de blockchain ganhou grande destaque e atenção por parte da mídia, governos, empresas e no meio acadêmico. Por ser um tema recente e amplamente veiculado em canais informais optou-se por fornecer referências que corroboram a relevância social do tema, fornecendo notícias e relatórios não acadêmicos para tal exclusivamente neste capítulo.

A confiança é um pilar fundamental e imprescindível das relações sociais, precedendo e viabilizando a consolidação de instituições como o governo ou o sistema monetário e financeiro. O conceito de moeda é amplamente amparado na confiança de sua capacidade

como meio de troca irretratável e escasso, ou seja, uma vez transacionada por bens e serviços a moeda não pode ser usada novamente, o que é conhecido como restrição ao gasto duplo e que se apresenta como grande entrave para uma moeda digital. Outros tipos de registro como contratos também dependem da irretratabilidade no sentido de que uma das partes envolvidas não pode anular seu compromisso, além de que em uma rede de comunicação possivelmente insegura deve haver a preocupação com a autenticidade das partes e integridade das informações trocadas.

O método difundido para resolver o problema da confiabilidade, especialmente em ambientes digitais, é transferir a responsabilidade de segurança e consenso dos dados para que instituições mediadoras resolvam determinado tipo de tarefa. Como exemplo, quando um pagamento é realizado por meio de um cartão de crédito, o operador do cartão é notificado e esse requisita informações para o banco de origem sobre o saldo atual do comprador, a fim de poder garantir e transmitir confiança para o vendedor.

Um problema que surge com a difusão tecnológica é o aumento na complexidade estrutural desse tipo de transação, já que a escalabilidade crescente demanda a criação de diversas e sucessivas camadas verificadoras. Isso aumenta consideravelmente o custo de transação uma vez que muitas das entidades são remuneradas à frente de suas responsabilidades, além de diminuir a velocidade de transação e a disponibilidade, relacionada ao tempo necessário para se obter resposta desses serviços centralizados. Conforme ilustrado em InfoMoney (2019), em procedimentos de custódia e liquidação investidores só recebem a custódia de certos investimentos alguns dias úteis após realizar a comprar, devido à complexidade e ao número de usuários dos sistemas.

1.2 MOTIVAÇÃO

A tecnologia de blockchain, que forma a base do Bitcoin proposto por Nakamoto (2008), possibilita mitigar o problema de falta de consenso e confiança informacional em redes descentralizadas, provendo transparência, imutabilidade e segurança aos registros digitais contidos nela, o que possibilita troca de informações e valores sem necessidade de uma autoridade verificadora central.

No campo monetário essa tecnologia é o que sustenta o surgimento e proliferação como meio de pagamento, e ativo especulativo, das chamadas criptomoedas, denominação dada a um conjunto de moedas digitais que ganharam tração desde 2012 e tem valor de mercado acumulado estimado em USD 190 bilhões e volume financeiro movimentado diário de USD 40 bilhões como aponta Cap (2019).

Outras aplicações na área financeira procuram suprir deficiências operacionais em diversos campos: comércio, liquidação e custódia de ativos financeiros em bolsas de valores ao redor do mundo, como mostrado em Forbes (2019). A capacidade desintermediadora torna-se portanto um forte destaque dessa tecnologia nos mercados de capitais, uma vez que os gastos de intermediação são estimados em USD 100 bilhões anualmente por Wyman (2016), o que justifica o forte movimento pela adoção da tecnologia também visto em Forbes (2019) em diversas bolsas de valores como a Nasdaq e ASX. No setor bancário, o Valor (2017) nota uma organização dos grandes bancos brasileiros no sentido de integrar a tecnologia a seus serviços.

Devido à característica de transparência, imutabilidade dos registros e segurança providas pela tecnologia, vislumbram-se diversas oportunidades de sua aplicação em serviços públicos oferecidos pelo Estado como sugere a OCDE (2016), com algumas nações já experimentando com a tecnologia, visando melhorar a oferta de serviços públicos. A Estônia é inovadora nessa esfera, provendo serviços de e-cidadania que englobam identificação civil, pagamento de impostos e possibilitam até a abertura de empresas, diminuindo a burocracia e democratizando o acesso a serviços estatais. Segundo Deloitte (2017), a Índia emprega a tecnologia no registro de posse patrimonial e de terras e até o Brasil tem experimentado com a tecnologia com o Congresso Nacional já armazenando registros das sessões plenárias em blockchains e o Bacen usando a tecnologia para viabilizar o sistema de Open Banking conforme Estadao (2019).

Um outro campo com potenciais aplicações para o uso de blockchains é na elaboração de contratos, com a rede Ethereum provendo estrutura para desenvolvimento dos chamados *smart contracts*. Essa estrutura de contratos pode ser usada para evitar a formação de conluio em leilões virtuais como mostra Wu et al. (2019).

1.3 OBJETIVOS DO PROJETO

O presente projeto consistiu em modelar o funcionamento, planejar a execução e desenvolver um simulador de blockchain com mecanismo de consenso baseado em Proof-Of-Work (PoW), com a devida etapa de testes e validação da ferramenta procurando benchmarks com exemplos reais. O simulador busca permitir ao usuário analisar a interação de componentes da rede e acompanhar a evolução de uma blockchain em um ambiente operacional customizável. Procurou-se representar os processos mais relevantes como: a mineração de blocos, que consiste no procedimento de criação de novos blocos de transação no sistema; e a orfanização de blocos, que consiste na geração de blocos de transação que não farão

parte da estrutura oficial de registros. Desta forma, objetivou-se criar um sistema para fazer a avaliação das seguintes funcionalidades:

- Análise de *payoff* do processo de mineração.
- Análise de taxa de blocos orfãos e tempo para confirmação de transações e consenso sobre estado da cadeia.
- Análise do comportamento da cadeia e da rede em diferentes tipos de ataques.

O projeto conta com uma interface visual para auxiliar o usuário durante a execução, favorecendo a visualização de processos e possibilitando o ajuste paramétrico de características de interesse.

1.4 PLANO DE TRABALHO

Para atingir os objetivos propostos na seção anterior, as seguintes etapas foram realizadas:

- Estudo e pesquisa bibliográfica sobre o tema.
- Levantamento e detalhamento de requisitos.
- Realização da modelagem e criação do diagrama das classes.
- Implementação das versões do programa.
- Teste de cada versão.
- Customização do programa com parâmetros padrão de algumas blockchains reais.

1.5 JUSTIFICATIVA

A partir da ampla gama de aplicações e usos potenciais da tecnologia, nota-se seu imenso potencial disruptivo, que é comprovado por estimativas feitas em Davos (2015) até 2027 10% do PIB mundial estará armazenado em blockchains.

Dois grandes empecilhos surgem para se reproduzir fidedignamente o comportamento operacional de uma blockchain baseada em PoW: o espalhamento geográfico e condições de rede dos usuários do registro e o elevado consumo de poder computacional e consequentemente de energia elétrica. Digiconomist (2018) estima que o consumo de energia do Bitcoin já é comparável ao de países como Colômbia e Suíça, estando atualmente na ordem de 60TWh.

A dificuldade prática, portanto, de se representar em um modelo todas as interações do sistema com reprodução das condições exatas de operação da rede e do protocolo de consenso faz com que seja necessário o uso de um simulador para estudar propriedades de blockchains. Desta maneira, torna-se possível simular parâmetros de rede e consenso da blockchain para deduzir propriedades importantes como dinâmica de evolução e vulnerabilidade a ataques, além de poder prover ferramentas para estimar se o *payoff* do processo de mineração é positivo dado um consumo de energia.

1.6 METODOLOGIA

A metodologia adotada para o desenvolvimento do projeto foi a seguinte:

- **Fundamentação Teórica:** a primeira parte do projeto consistiu em estudar sobre os conceitos fundamentais da blockchain conforme proposto por (NAKAMOTO, 2008).
- **Modelagem:** a segunda parte fundamentou-se em levantar os principais requisitos desejados, além de planejar a estrutura das classes pertencentes ao programa.
- **Implementação:** a terceira fase foi destinada a implementar e desenvolver as versões propostas do projeto, posteriormente detalhadas neste documento. Cabe ressaltar que a implementação foi baseada no padrão de arquitetura MVC (Model-View-Controller) conforme a figura 1.1, e com isso cada integrante do grupo ficou diretamente responsável por um segmento.
- **Testes:** a última fase foi destinada a testar os programas sob as óticas dos requisitos levantados. Essa fase ocorreu mais de uma vez, sempre após a conclusão da implementação de uma versão.

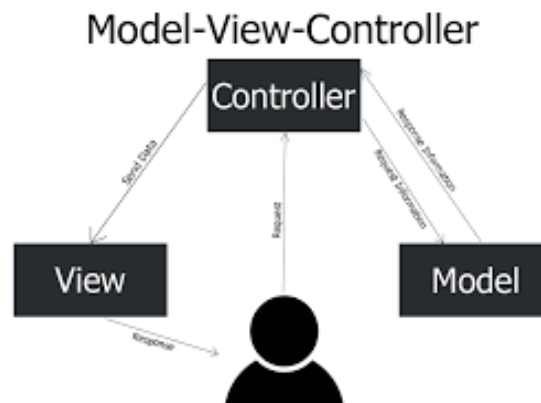


FIG. 1.1: Arquitetura MVC

1.7 ESTRUTURA DO TEXTO

O documento está segmentado na seguinte estrutura: o capítulo 2 aborda os conceitos necessários para o entendimento do projeto, especialmente a *blockchain*, seu funcionamento e o processo de mineração. O capítulo 3 apresenta a modelagem proposta pelos integrantes do trabalho. Já o capítulo 4 discute as principais decisões tomadas pelo grupo a respeito da implementação do projeto, enquanto o capítulo 5 detalha a execução e as decisões de projeto tomadas pelo grupo. Por fim, os capítulos 6 e 7 concluem apresentando um manual da ferramenta, avaliando o resultado do trabalho e propondo possibilidades para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 BLOCKCHAIN

A tecnologia de blockchain é um registro de dados distribuído compartilhado por usuários através de uma rede pública ou privada. O nome blockchain remete à estrutura de dados usada para armazenar os dados de interesse, com a unidade básica de armazenamento sendo um conjunto de informações transacionadas entre os usuários da rede chamado de **bloco**. A autenticidade das transações pode ser garantida por meio de uma arquitetura de chave pública - privada.

No modelo do Bitcoin proposto por Nakamoto (2008), a moeda digital consiste em uma sequência de assinaturas digitais de tal forma que uma transação de moeda do usuário A para o usuário B conteria a assinatura digital de A, assinada com a chave privada de A e um hash da chave pública de B com a transação anterior conforme a figura 2.1, possibilitando a qualquer membro da rede verificar a cadeia de propriedade da moeda através das assinaturas.

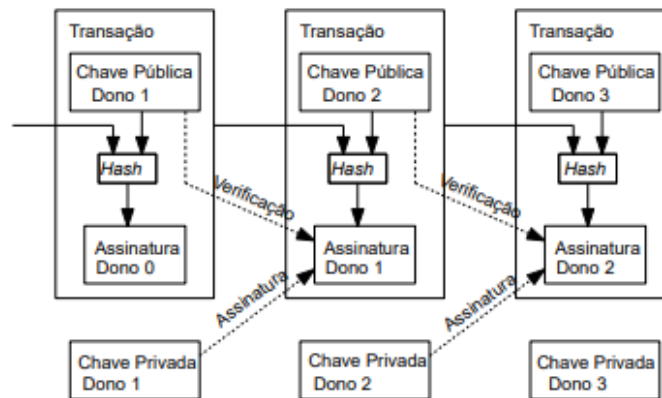


FIG. 2.1: Esquema de Assinatura Digital no Bitcoin

Os blocos são sequencialmente encadeados de forma que cada bloco possui um campo que é o hash do bloco anterior e todos os nós da rede distribuída possuem uma cópia da cadeia de blocos, de maneira que todos os nós tem acesso às transações previamente realizadas na rede. O fato dos blocos serem encadeados com o hash do bloco prévio na sequência garante consistência e imutabilidade dos registros, uma vez que se um usuário mal intencionado alterar algum elo da cadeia o hash correspondente mudará, tornando a

referência do bloco subsequente inválida e todas as posteriores, evidenciando a fraude. À medida que mais blocos são adicionados à cadeia só se torna possível alterar um bloco qualquer refazendo-se toda a cadeia de blocos a partir do bloco alterado.

Segundo Nakamoto (2008) esse mecanismo desincentiva o chamado gasto duplo, processo pelo qual um usuário mal intencionado transfere pagamento via moeda digital para um terceiro recebendo bens e serviços em troca e imediatamente depois reutilizando a moeda transferida em outra transação. O racional apresentado é que se o ofertante de bens e serviços esperar uma determinada quantidade de blocos minerados contendo referência ao bloco com a transação acaba tornando-se demasiadamente custoso alterar o bloco com a transação, uma vez que isso demanda reconstruir toda a cadeia de blocos subsequente ao bloco com a transação alvo de gasto duplo. Quanto maior for a quantidade de blocos de confirmação (*lag*) que o comerciante esperar, mais seguro pode estar de que o consumidor não terá incentivos para alterar a cadeia e reverter o pagamento dado a baixa probabilidade de o consumidor conseguir reconstruir toda a cadeia posterior, como evidenciado em Nakamoto (2008) .

A adição de um bloco à cadeia principal se dá por meio de um processo conhecido como mineração, onde cada nó dito minerador implementa um algoritmo de validação para encontrar um bloco considerado correto e anuncia para a rede o bloco encontrado segundo uma política de propagação de informação. Ao receber um bloco de outro nó o usuário verifica se o bloco recebido é válido e adiciona-o à sua versão da blockchain, continuando em seguida a trabalhar para minerar o próximo bloco. Cada blockchain possui sua política de consenso sobre a cadeia de blocos válida na rede, com o modelo tradicional proposto em Nakamoto (2008) considerando a maior cadeia de blocos como a correta. O algoritmo de validação mais utilizado é o *Proof of Work* (PoW), onde cada minerador vota sobre a validade de um bloco de transações, sendo o poder de voto proporcional à capacidade computacional relativa disponível no nó minerador da rede.

É válida uma importante distinção entre os tipos de usuários da rede, com mineradores tendo o referido poder de voto para validar os blocos da cadeia enquanto os demais usuários tomam uma posição passiva, tendo suas transações confirmadas pelos mineradores. Os mineradores por sua vez são incentivados a validar blocos por meio de taxas de confirmação sobre as transações confirmadas e/ou emissão de unidades monetárias da criptomoeda correspondente. No Bitcoin, por exemplo, cada bloco minerado gera 12,5 BTC novas para o minerador que validou o bloco além de taxas de validação da transação correspondente, com o ritmo de emissão de moeda sendo decrescente ao longo do tempo até a quantidade total de moedas em circulação atingir 21 milhões.

Depreende-se daí que um nó minerador que possuir mais da metade de todo o poder computacional da rede poderá controlar a validação de blocos e conseqüentemente a rede, já que a maior cadeia forjada é aceita como válida. Ataques em que se procura obter a maioria do poder computacional total são conhecidos como ataques de 51% e foram responsáveis pela derrocada de algumas criptomoedas como a Vertcoin e a Bitcoin Gold como mostra PortalBitcoin (2018), evidenciando uma fragilidade de blockchains com algoritmo de consenso PoW.

2.1.1 MINERAÇÃO

O processo de mineração se dá pelo uso de processamento computacional por parte dos mineradores. No Bitcoin os mineradores incrementam continuamente um contador chamado *nonce* que faz parte da estrutura de dados do bloco de transações juntamente com o hash do bloco anterior e uma estampa temporal (*timestamp*) usada para garantir a cronologia da cadeia. A cada iteração e incremento de *nonce* é computado o *hash* do bloco de transações com o respectivo *nonce* da iteração correspondente, verificando-se então se o valor do *hash* é menor do que uma cota superior determinada, condição em que considera-se o bloco como válido. Isso é equivalente a exigir que a representação binária do hash comece com uma quantidade específica de bits 0. Esse limite para o hash deve ser ajustável visando manter a dificuldade de resolução do PoW compatível com o poder computacional total da rede, a fim de garantir um tempo médio de criação de bloco constante. Na figura 2.2 extraída de (NAKAMOTO, 2008) é possível ver o encadeamento de blocos válidos gerados pela mineração:

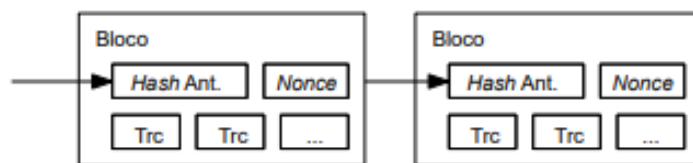


FIG. 2.2: Cadeia de blocos válidos

O dispêndio de processamento computacional demanda gasto com energia elétrica para manutenção do funcionamento do hardware, o que representa um custo adicional a ser incorrido pelo minerador além do investimento inicial com o hardware dedicado para a atividade. Em determinadas condições de taxa de validação da rede de blockchain, poder computacional relativo à rede e, no caso das criptomoedas, taxa de conversão para moedas

tradicionais, pode ser financeiramente lucrativo para um minerador validar os blocos, o que constitui um dos objetos de análise deste trabalho.

De maneira geral as blockchains possuem um parâmetro razoavelmente fixo de tempo de criação de blocos para manter a velocidade de processamento de transações na rede constante já que pode-se supor a quantidade de transações por bloco constante e limitada ao tamanho do bloco. A tabela 2.1 mostra um comparativo do tempo médio de criação dos blocos e tamanho máximo dos blocos para diferentes criptomoedas segundo Gervais et al. (2016)

TAB. 2.1: Comparativo de intervalo de criação e tamanho de bloco

Criptomoeda	Intervalo de blocos	Tamanho de bloco
Bitcoin	10 min	534KB
Litecoin	2.5 min	6.11 KB
Dogecoin	1 min	8KB
Ethereum	10 - 20s	1.5KB

O tempo de criação de blocos limita a velocidade de processamento de transações, sendo um fator restritivo de escalabilidade para blockchains baseadas em PoW. Outro ponto a ser considerado é que, a depender da dificuldade de solução do PoW relativa à atividade computacional da rede como um todo, é possível que durante a propagação de um bloco minerado pelos nós da rede outro bloco seja minerado de maneira independente, o que levaria a versões conflitantes sobre o último bloco da cadeia. O bloco que tiver maior aceitação primeiro pelos outros integrantes da rede será aceito e o outro será considerado orfanizado, diz-se que ocorreu um *fork*. Desta forma, parâmetros de rede como largura de banda e latência de comunicação tornam-se fatores importantes para se determinar a aceitação de uma versão da cadeia.

2.2 MODELAGEM DO PROCESSO DE MINERAÇÃO

A maneira convencionalmente mais usada para modelar o processo de mineração é considerá-lo um processo estocástico de Poisson com média igual à quantidade esperada de blocos por unidade de tempo segundo o esquema do protocolo usado, conforme observado em Rosenfeld (2014) . Em condições de operação real de um sistema de blockchain podem haver flutuações no poder computacional total da rede com uma adaptação possivelmente retardada da dificuldade do problema de PoW, o que geraria uma mudança no tempo médio de criação de blocos, acarretando prejuízos à modelagem do processo como de Poisson. Em Bowden et al. (2018) é feita uma análise dos intervalos de chegada de blocos

na rede do Bitcoin, chegando-se à conclusão de que a distribuição dos tempos de chegada conforma-se com razoável aderência a uma distribuição exponencial, o que corrobora a hipótese de modelagem por processos de Poisson.

2.2.1 RUÍNA DO JOGADOR E GASTO DUPLO

Em Nakamoto (2008) o problema do gasto duplo e dos *lags* de confirmação necessários para se ter confiança em uma transação é tratado sob a ótica de um passeio aleatório binomial, reduzindo-se ultimamente a um problema de ruína do jogador.

No jogo proposto o atacante parte de uma quantidade de blocos e busca forjar uma cadeia alternativa à cadeia aceita pela rede, buscando recuperar-se de um déficit de z blocos, em que z é quantidade de blocos fazendo referência à transação de interesse que um prestador de bens ou serviços espera antes de fornecê-los a um consumidor em troca de pagamento em criptomoedas. Uma vez que a probabilidade do atacante ter sucesso na mineração é proporcional ao seu poder computacional relativo à rede q , temos que a probabilidade da diferença entre o tamanho da cadeia alternativa do atacante e a da rede reduza em uma unidade é igual a q . Da mesma forma, a probabilidade complementar da rede ter êxito na mineração e conseqüentemente a distância entre as cadeias aumentar em uma unidade é igual a $1 - q = p$. O problema de ruína do jogador configura-se uma vez que para haver manipulação da cadeia o atacante deve conseguir superar a diferença de z blocos para confirmação de transação, forjando uma cadeia com $z + 1$ blocos de maneira a anular o efeito da transação já que sua cadeia alternativa será aceita pela rede como maior cadeia, e portanto válida.

Segundo Nakamoto (2008) a probabilidade de um atacante com fração q do poder computacional total conseguir recuperar-se de uma diferença de z blocos é de:

$$q_z = \begin{cases} 1, & \text{if } q > p \\ \left(\frac{q}{p}\right)^z, & q \leq p \end{cases} \quad (2.1)$$

A equação apresentada não está inteiramente correta uma vez que supõe que a disponibilidade de recursos para minerar seja irrestrita, ou seja, uma modelagem de ruína do jogador em que o jogador tenha recursos infinitos para jogar, o que se configura uma hipótese infundada para efeitos práticos. Ademais, conforme argumenta (OZISIK; LEVINE, 2017) o expoente da fração deve ser $z + 1$, uma vez que não basta alcançar a maior corrente de blocos, sendo necessário ultrapassá-la em pelo menos uma unidade para que a cadeia alternativa tenha aceitação unânime pela rede. É válido notar que o primeiro caso

em que $q > p$ refere-se ao caso de um ataque de 51% , enquanto o segundo, em que $q \leq p$, evidencia uma vulnerabilidade da blockchain para casos em que o atacante detenha menos da maioria do poder computacional total e destaca a importância da quantidade de *lags* para dar segurança ao processamento das transações.

Essa questão dos *lags* é um grande restritivo de escalabilidade das blockchains baseadas em PoW pois fica estabelecido um *trade-off* entre velocidade de processamento de transações e segurança das mesmas, o que na prática inibe que transações correntes diárias sejam executadas.

3 MODELAGEM

O principal objetivo deste capítulo é apresentar a modelagem proposta para o projeto. A tarefa se dividiu nas seguintes partes: levantamento e detalhamentos dos requisitos.

3.1 REQUISITOS DO SISTEMA

De acordo com Pressman e Maxim (2016) os requisitos constituem uma estrutura básica do projeto, já que representam os principais recursos de um produto e seus fluxos de informação.

Segundo Bezerra (2015), o levantamento de requisitos é uma etapa crucial durante o planejamento de um projeto, pois possui como objetivo geral a aceitação entre as partes das finalidades do programa. O mesmo autor também classifica os requisitos de acordo com a sua natureza:

- Requisitos funcionais: são os requisitos que representam as necessidades que um sistema deve prover.
- Requisitos não-funcionais: são os requisitos que representam as qualidades que o sistema deve ter e não estão relacionados com as finalidades desse.

A principal técnica de levantamento de requisitos utilizada foi *brainstorm*, e os requisitos funcionais são apresentados abaixo:

- RF01 - Inicialização do sistema: o programa deve prover campos de entrada para o usuário escolher os principais parâmetros que irão caracterizar a simulação.
- RF02 - Análise de recompensa da rede: o usuário deve ser capaz de analisar o retorno esperado da atividade de mineração para determinada *blockchain* de forma dinâmica, ou seja, a taxa de inserção de mineradores varia ao longo do tempo.
- RF03 - Análise do consenso da rede: o sistema deve ser capaz de retornar informações estatísticas sobre o consenso de determinada *blockchain*.
- RF04 - Consulta ao sistema: o sistema deve ser capaz de retornar os eventos ocorridos até o determinado momento.

- RF05 - Exportação das configurações: o sistema deve prover ferramentas para o usuário exportar as informações construídas através de simulações realizadas.
- RF06 - Carregamento de configurações: o sistema deve ser capaz de reproduzir uma mesma simulação realizada por um usuário posteriormente.

3.2 DETALHAMENTO DOS REQUISITOS

As subseções a seguir detalham cada um dos requisitos levantados.

3.2.1 INICIALIZAÇÃO DO SISTEMA

Identificador e nome: RF-01 Inicializar sistema.

Ator principal: Usuário.

Pré-condição: O caso de uso começa quando o usuário inicia a aplicação.

Fluxo Principal:

1 - O simulação é iniciada.

2- O sistema apresenta os seguintes campos para o usuário preencher:

- Nome da simulação
- Custo energético
- Consumo energético
- Poder computacional médio dos participantes.
- Poder computacional do usuário,
- Distribuição probabilística
- Média da distribuição probabilística, se houver.
- Recompensa por minerar um bloco.
- Tempo médio para geração de blocos.

2 - O sistema notifica o *upload* dos dados e o caso de uso termina.

Pós condição: O sistema irá ter armazenado os dados disponibilizados pelo usuário.

3.2.2 ANÁLISE DE RECOMPENSA DA REDE

Identificador e nome: RF-02 Análise de recompensa da rede.

Ator principal: Usuário.

Pré-condição: O sistema deve ter sido inicializado.

Fluxo principal:

1 - O usuário seleciona o botão correspondente.

2 - O programa retorna o gráfico lucro x tempo.

Pós-condição: Análise de recompensa da rede apresentada.

3.2.3 ANÁLISE DO CONSENSO DA REDE

Identificador e nome: RF-03 Análise do consenso da rede.

Ator principal: Usuário.

Pré-condição: O sistema deve ter sido inicializado.

Fluxo Principal:

1- O usuário seleciona o botão referente no menu de opções.

2- O sistema recupera os eventos de *fork* do backlog gerado durante a simulação.

3- O número de ocorrências de *fork* é exibido em um campo determinado.

Pós-condição: Análise de consenso da rede apresentada.

3.2.4 CONSULTA AO SISTEMA

Identificador e nome: RF-04 Consulta à blockchain.

Ator principal: Usuário.

Pré-condição: O sistema deve ter sido inicializado.

Fluxo Principal:

1- O usuário seleciona o botão referente no menu de opções.

2- O programa disponibiliza um arquivo .csv com o *log* dos eventos ocorridos, o tempo de ocorrência e se o usuário foi o ator de terminado evento.

Pós-condição: Consulta ao sistema realizada.

3.2.5 EXPORTAÇÃO DAS CONFIGURAÇÕES

Identificador e nome: RF-05 Exportação das configurações.

Ator principal: Usuário.

Pré-condição: O sistema deve ter sido inicializado.

Fluxo Principal:

- 1- O usuário seleciona o botão referente.
- 2- O programa disponibiliza um arquivo .bds com as configurações criptografadas da simulação realizada.

Pós-condição: Dados exportados com sucesso.

3.2.6 CARREGAMENTO DE CONFIGURAÇÕES

Identificador e nome: RF-06 Carregamento de configurações.

Ator principal: Usuário.

Pré-condição: O sistema deve ter sido inicializado.

Fluxo Principal:

- 1- O usuário seleciona o botão referente.
- 2- O programa disponibiliza uma área para fazer o *upload* de um arquivo .bds com as configurações.
- 3- O sistema retorna a simulação correspondente.

Pós-condição: Dados carregados com sucesso.

3.3 DIAGRAMA DE CLASSES

Durante a modelagem foram implementadas as classes de usuário, simulação, blockchain, bloco, minerador e evento, no contexto de simulação da dinâmica de funcionamento da blockchain. Um evento pode ser a adição de um bloco à blockchain, entrada de um minerador à simulação ou ocorrência de um fork.

Com o objetivo de guardar informações relativas aos eventos transcorridos criou-se a classe de eventos. Visando diminuir o overhead, isto é, o alto número de objetos no sistema e procurando minimizar o tempo de carregamento dos logs, optou-se por associar um objeto evento por blockchain.

A figura 3.1 representa o diagrama de classes final.

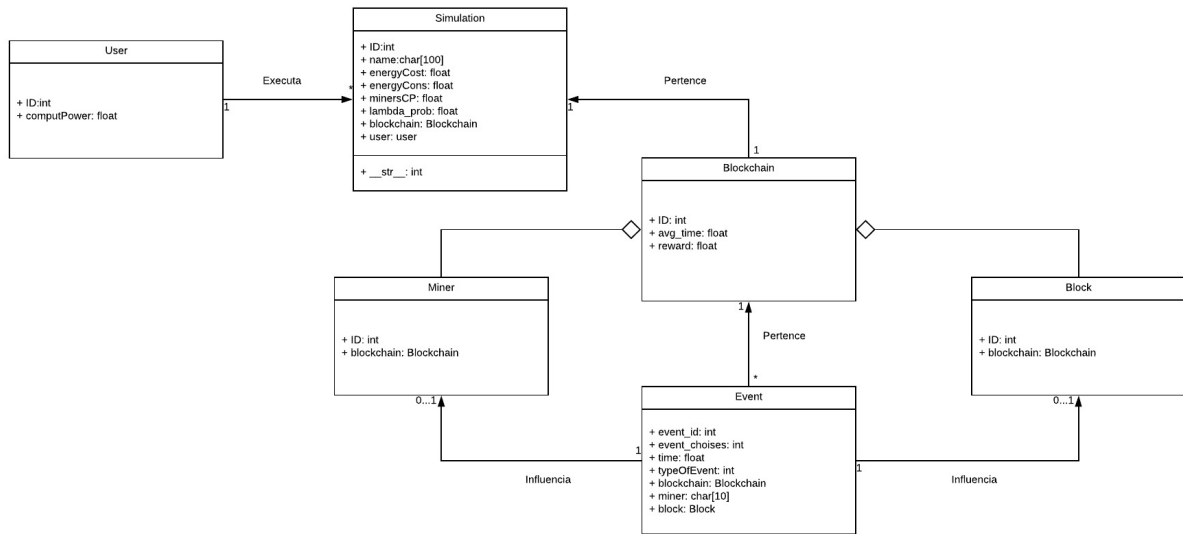


FIG. 3.1: Diagrama de classes

4 ARQUITETURA DO PROJETO

Este capítulo é destinado para detalhar como os participantes do grupo se planejaram para desenvolver o projeto e quais técnicas foram utilizadas.

4.1 VERSÕES DO PROGRAMA

O desenvolvimento do projeto foi feito de forma iterativa, ou seja, os objetivos do projeto foram divididos em versões para implementação e testes. A versão 1 consistiu em prover ferramentas para analisar o *payoff* da rede e a versão 2 para a análise de *fork* e consenso de rede. Inicialmente planejou-se como objetivo final a implementação da versão 3, a qual seria capaz de simular ataques ao sistema. No entanto tal objetivo foi relaxado, devido à complexidade e à restrição de tempo, conforme esperado por todos os membros do projeto.

4.2 LINGUAGEM DE PROGRAMAÇÃO

A escolha da linguagem de programação a ser utilizada neste trabalho, levou em consideração diversos fatores, entre eles: facilidade de implementação, velocidade de execução, complexidade e popularidade. A linguagem de programação Python atendeu a todos os requisitos e, portanto, foi a linguagem escolhida para a implementação do simulador. Algumas das vantagens que motivaram o uso dessa linguagem foram o amplo suporte à bibliotecas desenvolvidas por terceiros e sua capacidade de integração ao desenvolver aplicações web.

Por ser uma linguagem interpretada, apresenta desempenho inferior à outras linguagens como Java, C++ e C. No entanto, para aplicações web, essa característica deixa de ser uma desvantagem e passa a ser uma vantagem, pois o programa não precisa ser compilado previamente, isto é, a página é interpretada ao longo de sua execução. Atualmente, diversas empresas eminentes utilizam Python, como por exemplo, Google, Facebook, Instagram, Spotify, Netflix.

Todo o projeto foi desenvolvido utilizando Django, um framework de alto nível e código aberto construído para possibilitar o desenvolvimento web rápido. Os cálculos e gráficos apresentados na aplicação foram realizados por meio das bibliotecas Numpy e construídos em Javascript, respectivamente.

4.2.1 DJANGO

Django é um *framework web*, isto é, um conjunto de componentes que ajudam a desenvolver de forma rápida aplicações web. Além de ser gratuito, escrito em Python e possuir código aberto, Django oferece um servidor padrão para testes.

Toda a estrutura do *framework* foi projetada para ser fracamente acoplada e fortemente coesa, isto é, diferentes partes da estrutura, apesar de conectadas umas as outras, não são interdependentes. Um dos principais fatores que possibilita essa interdependência é sua arquitetura. Por ser baseada no padrão Model-View-Controller (MVC), o framework separa a lógica do aplicativo, a interface com o usuário (UI) e as camadas de acesso a dados, visando permitir que cada camada seja modificada de forma independente, sem afetar as outras camadas.

De acordo com sua documentação, sua arquitetura segue o padrão Model-Template-View (MTV). A camada Model representa a camada de acesso a dados, onde o aplicativo interage com os bancos de dados e seus provedores. A camada Template define como os dados devem ser apresentados ao usuário. Nesse padrão de arquitetura, a camada View descreve quais dados devem ser apresentados ao usuário. Não define a forma exata de apresentação, delega essa tarefa à camada Template.

4.2.2 NUMPY

O NumPy é uma biblioteca em Python que permite trabalhar com elementos matemáticos N-dimensionais, possibilitando operações complexas e trabalhosas. É semelhante ao software Matlab, porém apresenta maior eficiência e permite usufruir de todo o poder da linguagem de programação Python.

Além disso, oferece a possibilidade de herança, permitindo a customização do comportamento (por exemplo, dos operadores típicos de adição, subtração e multiplicação). Por ser implementado em C, todas as operações matemáticas são realizadas em um tempo extremamente curto.

4.2.3 JAVASCRIPT

O javascript é uma linguagem interpretada amplamente usada para desenvolvimento web. Sua flexibilidade de uso torna-o presente na maioria das aplicações e por meio dele provê-se interatividade para páginas HTML. Seu uso facilita a criação e incorporação de gráficos em aplicações.

5 IMPLEMENTAÇÃO DO PROJETO

Como foi descrito seção 1.6, cada integrante do grupo foi o principal responsável por uma parte do projeto, já que se utilizou a arquitetura MVC. Dito isso, este capítulo destina-se à apresentar as principais decisões tomadas em cada frente de desenvolvimento.

5.1 MODEL

Além da modelagem proposta na seção 3.3, outra decisão de projeto foi transferir funções que envolviam cálculo para o Controller, que inicialmente eram realizadas no modelo, para realizar o cálculo somente sob demanda, diminuindo o custo computacional. Essas funções eram responsáveis por retornar o poder computacional total da rede e retornar informações no tempo como número de mineradores, blocos minerados e eventos ocorridos.

Vale ressaltar que a figura 3.1 representa o diagrama após a transferência das funções citadas acima. O código 5.1 abaixo apresenta o código das funções que foram transferidas para o controller.

```
def get_num_info(blockchain, time):
    num_miners = Event.objects.filter(blockchain=blockchain).filter(typeOfEvent=3).
        filter(time__lte=time).count()
    blocks_add = Event.objects.filter(blockchain=blockchain).filter(typeOfEvent=1).
        filter(time__lte=time).count()
    blocks_remove = Event.objects.filter(blockchain=blockchain).filter(typeOfEvent
        =2).filter(time__lte=time).count()
    num_blocks = blocks_add - blocks_remove
    num_forks = Event.objects.filter(blockchain=blockchain).filter(typeOfEvent=5).
        filter(time__lte=time).count()
    num_events = Event.objects.filter(blockchain=blockchain).filter(time__lte=time)
        .count()
    dados = {"num_miners": num_miners, "num_blocks": num_blocks, "num_events"
        : num_events, "num_forks": num_forks}
return dados
```

```
def get_total_cp(blockchain, time):
    totalCP = 0
    simulation = Simulation.objects.filter(blockchain=blockchain)[0]
    event_set = Event.objects.filter(blockchain=blockchain).filter(typeOfEvent=3).
        filter(time__lte=time).count()
    totalCP = event_set*simulation.minersCP
    return totalCP + simulation.user.computPower
```

Listing 5.1: Funções implementadas - Model

5.2 VIEW

Na frente de View foram desenvolvidas duas páginas principais, sendo um formulário para *input* das informações de inicialização da simulação e outra para visualização do estado da blockchain e do gráfico de *payoff* ao longo do tempo. Procurando-se garantir uma melhor experiência para o usuário a seção de formulário conta com uma opção de preenchimento dos parâmetros replicando-se o observado no Bitcoin e outra inteiramente personalizável.

A figura 5.3 representa a interface inicial e qual o padrão utilizado para receber os dados de entrada. Já a figura 5.4 mostra o resultado de uma simulação, tanto o gráfico de recompensa quando o número de ocorrência de *fork*.

Inicie sua simulação

Preencha os dados da simulação abaixo ou escolha uma configuração padrão (bitcoin). Você também pode carregar um arquivo .bds de configuração.



Configurações da Simulação

Nome da simulação:

Custo energético (em R\$/kWh):

Consumo energético (em kW):

Poder computacional do usuário (em TH/s):

Poder computacional dos outros mineradores (em TH/s):

Distribuição probabilística de entrada de novos mineradores:

Média da distribuição probabilística:

Configurações da Blockchain

Recompensa por minerar (em R\$):

Tempo médio para geração de blocos (em min.):

Configs iniciais (opcional .bds): Nenhum arquivo selecionado

FIG. 5.1: Interface do programa



FIG. 5.2: Resultado de uma simulação

Nota-se também a apresentação de informações de estado da simulação como número de mineradores, blocos minerados bem como o tempo, quantidade de eventos ocorridos.

Entre as decisões de projeto tomadas nessa frente pode-se elencar a utilização de flexbox para modularização dos componentes da página por facilitar o desenvolvimento multiplataforma (web, mobile) e o uso de requisições ajax para realizar chamadas aos controllers, evitando a reinicialização da página ao abrir-se o log por exemplo.

5.3 CONTROLLER

Na seção de Controller foram modularizadas as principais funções do programa: geração de dados para alimentar o gráfico de lucro na view, criação de um objeto de simulação e um objeto de blockchain, preenchimento dos dados iniciais e cálculo da função de análise de consenso da rede.

A função geradora de eventos foi feita admitindo-se que o usuário possa especificar o total de tempo transcorrido na simulação ou a quantidade de eventos que se deseja gerar, com cada evento sendo armazenado em uma instância da classe evento associada à blockchain em questão.

O tempo para geração dos blocos foi calculado a partir de uma distribuição exponencial com parâmetro configurável para o tempo médio de geração de um bloco. Uma vez calculado o tempo para o próximo bloco, procurou-se estabelecer qual minerador foi o responsável por originar o novo bloco. Isso foi feito considerando-se haver duas classes de mineradores: o usuário e o restante, e sorteando-se uma variável aleatória uniforme no intervalo $[0,1]$. Caso o valor sorteado fosse menor do que a proporção de poder computacional do usuário sobre o total, que também é sua probabilidade de minerar um bloco, considerou-se que o bloco foi minerado pelo usuário.

Por fim, para calcular os pontos do gráfico, computou-se o número de mineradores por dia, considerando um espaço amostral fixo de 1800 dias, corresponde a cinco anos. Para isso, o cálculo é feito de forma dinâmica, ou seja, é uma distribuição de Poisson com a taxa de entrada variável. Esse parâmetro de entrada foi modelado por uma distribuição exponencial, de forma que se torna menor de acordo com a quantidade de mineradores total do sistema, tendendo para zero no limite do número de mineradores tendendo ao infinito.

A função que retorna a quantidade de conflitos (ou *forks*) durante a geração de um bloco foi implementada como uma distribuição de Poisson com um parâmetro de média modelado como o intervalo para geração do próximo bloco dividido pelo tempo médio de geração de um bloco. Tomando como exemplo a *bitcoin*, a qual possui uma *blockchain* com tempo médio de geração de 10 minutos, o parâmetro utilizado é 0.1 vezes o intervalo.

A ideia por trás desse procedimento foi inicialmente verificar para o intervalo de geração do próximo bloco qual o número médio de blocos que o processo de contagem de Poisson subjacente deveria gerar nesse intervalo, que deve ser dado pela razão entre o intervalo e o tempo médio de geração de bloco. Em seguida, sorteou-se uma variável aleatória com distribuição de Poisson e média igual ao parâmetro previamente calculado e verificou-se se a quantidade de contagens pelo processo foi maior ou menor do que um. Caso a quantidade de contagens tenha sido maior que 1 considera-se que houve um *fork*, caso contrário considera-se que só o bloco originalmente minerado realmente o foi no intervalo necessário para sua geração.

Para implementar a exportação e o carregamento de dados, foi desenvolvida uma função que recebe como entrada a identificação da simulação e retorna um arquivo criptografado, que será utilizado posteriormente para carregar a mesma simulação. Para isso foi desenvolvida uma função de criptografia e decifração que utiliza o método AES - 32 bits.

O log de eventos foi implementado para exportação em um arquivo .csv, devido à facilidade de manutenção dos dados. O arquivo contém: o tipo de evento, o tempo em que aconteceu o evento e a indicação se o usuário foi o ator do evento ou não - caso seja um evento de mineração de blocos. A decisão tomada de exportar os dados e não visualizar agilizou bastante o processamento, de forma que não foi necessário o uso de um banco de dados não relacional.

6 GUIA DE USUÁRIO E RESULTADOS

Este capítulo busca fornecer uma explicação mais detalhada acerca das funcionalidades da ferramenta desenvolvida exemplificando com casos de uso e mostrando alguns resultados que podem ser obtidos.

6.1 TELA INICIAL DE CONFIGURAÇÃO

Ao iniciar a ferramenta o usuário será direcionado para a tela inicial onde é possível configurar os parâmetros de uma nova simulação ou carregar uma simulação previamente realizada. Esta tela está representada na figura 5.3 do capítulo anterior na sessão de View.

Após o carregamento da página, caso usuário queira escolher iniciar uma nova simulação de *blockchain* é possível fazê-lo com parâmetros customizados e definidos pelo próprio usuário ou usar as configurações pré-definidas que buscam replicar as condições de operação do Bitcoin.

Caso opte pela opção personalizada o usuário deve certificar-se que a caixa indicada como "Personalizada" está selecionada e com a cor azul, e os campos de preenchimento estão somente com os *placeholders*. A seguir é necessário escolher os valores para as variáveis de simulação, sendo elas:

- Nome da simulação: nome arbitrário usado para identificar uma simulação e o arquivo de simulação do tipo .bds que pode ser posteriormente salvo.
- Custo energético: custo em reais para consumir 1kWh, depende da distribuidora de energia do usuário e pode ser encontrado na conta de luz.
- Consumo energético: indica qual é a potência do hardware usado para fins de mineração em kW, podendo ser encontrado nas especificações do fabricantes dos chips ASICS usualmente usados para esse fim.
- Poder computacional do usuário: medido em TH/s, indica quantas tentativas de resolução do problema de PoW o usuário faz por segundo.
- Poder computacional dos outros mineradores: análogo ao do usuário, sendo relativo ao poder de cada minerador.

- Média da distribuição probabilística: número médio de mineradores que entram na rede por minuto.

Após escolher os parâmetros de configuração da simulação o usuário deve preencher a seção de Configurações da Blockchain ao final da página, escolhendo as variáveis de :

- Recompensa por minerar: valor dado como recompensa por conseguir resolver o PoW e minerar um bloco. Para criptomoedas a unidade de valor geralmente corresponde à quantidade de moedas virtuais criadas, usamos a unidade em R\$ por conveniência.
- Tempo médio de geração de blocos: tempo em minutos correspondente ao inverso do parâmetro de Poisson do processo de geração de blocos.

Caso o usuário tivesse inicialmente optado por carregar uma simulação previamente realizada, poderia fazê-lo clicando no botão Escolher Arquivo ao lado da opção "Configs iniciais (opcional.bds)" e selecionando um arquivo .bds com a simulação prévia. Finalizado o preenchimento dessa seção o usuário deve apertar no botão "Iniciar Simulação" para passar à tela principal da simulação.

6.2 TELA DE SIMULAÇÃO

A tela principal é a mesma apresentada no capítulo 5 na figura 5.2 . Vamos analisar individualmente as componentes da página destacando as funcionalidades.

6.2.1 PASSAGEM DE TEMPO E EVENTOS

Na parte superior da tela encontram-se os painéis de passagem de tempo e de evento conforme a figura 6.1. O usuário pode escolher transcorrer a simulação orientando-se pela quantidade de tempo ou de eventos desejado. Caso deseje a abordagem orientada a tempo o usuário deve escolher a quantidade de minutos que deseja simular digitando o valor no painel da esquerda, com o símbolo do relógio, ou incrementar pela seta até o valor desejado, apertando o botão "Ir" para iniciar.

Do contrário, caso opte por simular uma determinada quantidade de eventos o usuário pode digitar o valor desejado no painel da direita, com o símbolo triangular com sinal de exclamação, apertando o botão "Ir" para iniciar.

É válido ressaltar que as informações de tempo e evento iniciam-se zeradas, e que também é possível retroceder a simulação de tempos e eventos, retornando para um tempo ou quantidade e eventos menor que o atual caso seja de interesse.

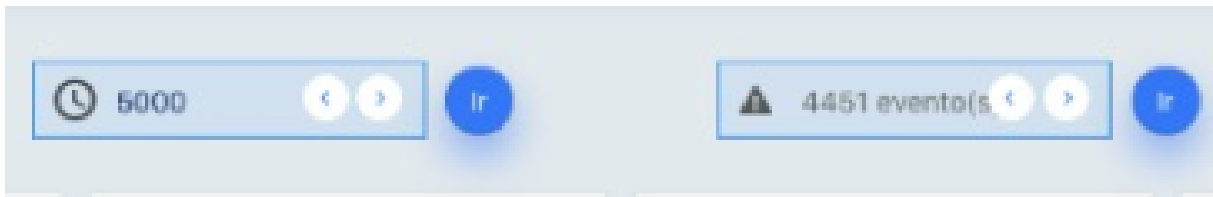


FIG. 6.1: Botões de passagem de tempo e evento

6.2.2 SALVAMENTO

Também na parte superior mas à direita encontram-se os botões para salvamento do log e salvamento da simulação conforme apresentado na figura 6.2.

Caso o usuário selecione a opção de baixar log apertando o botão "Salvar log" será baixado um arquivo .xls com um log com as informações detalhadas sobre os eventos transcorridos, com indicação do timestamp em que ocorreu o evento, qual foi o tipo do evento e se foi um bloco minerado pelo usuário.

Alternativamente, ao selecionar o botão "Salvar Simulação" será baixado um arquivo criptografado do tipo .bds que poderá ser posteriormente usado para carregar a mesma simulação.

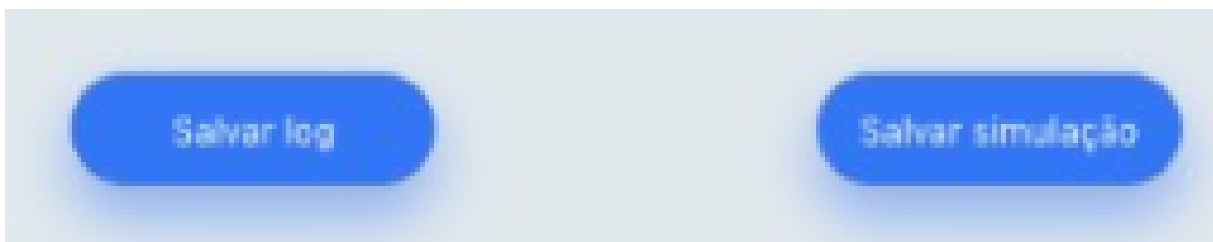


FIG. 6.2: Botões de salvamento

6.2.2.1 PAINEL DE SIMULAÇÃO

Logo abaixo dos botões para passagem de tempo e evento e dos botões de salvamento encontra-se o painel de simulação, que apresenta todas as informações acerca do estado da blockchain e é mostrado na figura 6.3.

Da esquerda para a direita temos informações sobre:

- Quantidade total de mineradores no sistema
- Quantidade total de blocos minerados pela rede
- Contagem do número de ocorrências de *fork*
- Tempo total transcorrido

- Quantidade de eventos ocorridos até então



FIG. 6.3: Painel de informações

6.2.3 GRÁFICO

Abaixo do painel de informações encontra-se o gráfico de análise de lucratividade da atividade de mineração. Este gráfico apresenta um horizonte de tempo fixo de 5 anos e mostra qual é o valor esperado de lucro para esse horizonte de tempo, conforme a figura 6.4. O cálculo de lucratividade é apresentado considerando os parâmetros de simulação e o poder computacional relativos constantes a partir do instante de tempo considerado.

O caráter não linear é devido à taxa variável de entrada de mineradores no tempo, o que é considerado na elaboração dos gráficos.

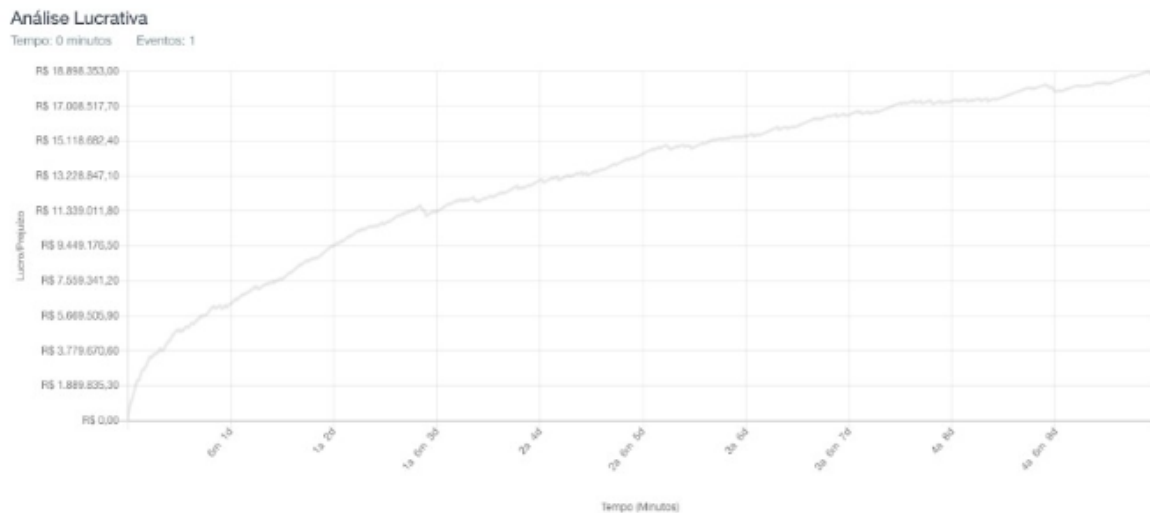


FIG. 6.4: Gráfico

Como para construção do gráfico considera-se o custo variável de energia como única componente de custo temos que o gráfico inicia-se na origem. A receita do usuário advém do processo de mineração e é o valor informado como recompensa por minerar um bloco.

Podem haver situações em que o gráfico tome valores negativos, ou seja, a receita esperada de mineração não compensa o gasto energético dispendido. Essa situação é apresentada na figura 6.5.

Análise Lucrativa

Tempo: 5000 minutos Eventos: 4451

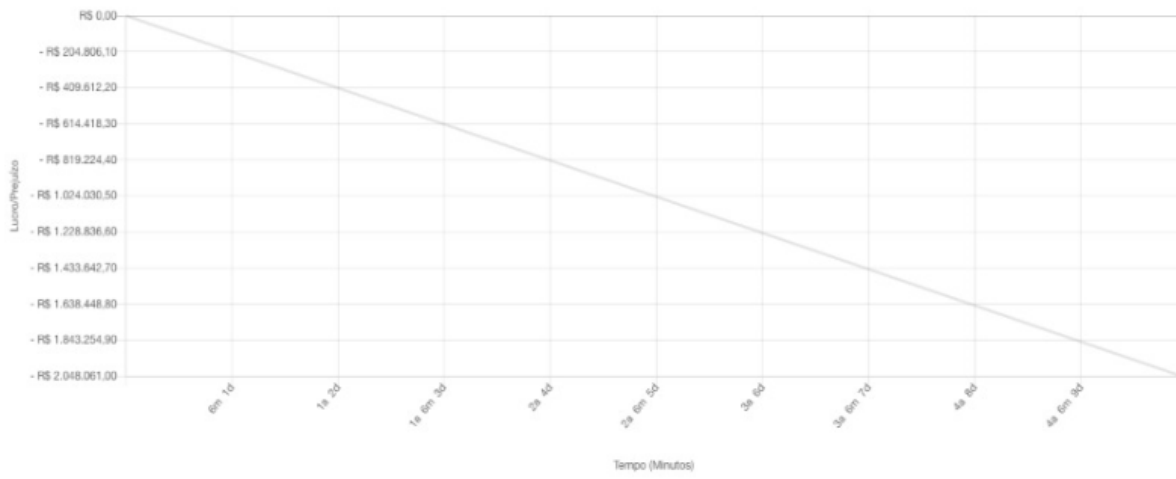


FIG. 6.5: Gráfico com lucro negativo

7 CONCLUSÃO

Este capítulo possui como objetivo apresentar as conclusões do projeto e apontar possíveis trabalhos futuros.

7.1 PROJETO

O trabalho teve como objetivo o projeto e o desenvolvimento de um simulador de interação em sistemas de *blockchain* capaz de reproduzir as condições de um sistema real. Além disso, a ferramenta também possibilita as análises de recompensa da rede, de forma dinâmica, e de consenso da rede. De forma a melhorar a usabilidade, o sistema também é capaz de exportar e carregar informações acerca de um sistema simulado.

A primeira parte do projeto consistiu em pesquisar sobre os sistemas de *blockchain* e suas principais vantagens, e também decidir quais funcionalidades o programa deveria ter, tomando como base as principais formas de emprego da tecnologia. Durante o desenvolvimento, cada integrante do grupo se tornou diretamente responsável por uma área ("Model-View-Control") e todas as escolhas relacionadas à implementação foram tomadas juntas, buscando uma visão holística do projeto.

O projeto proveu a oportunidade de estudar sobre um dos temas mais recentes da área de segurança da informação - a tecnologia de *blockchain* - e de viver a experiência de gerenciar e executar um projeto, visto o papel de cada integrante.

7.2 TRABALHOS FUTUROS

Dos objetivos iniciais propostos somente a ferramenta de simular ataques ao sistema não foi implementada, em muito devido à elevada complexidade de desenvolvimento e imponderáveis no decorrer do projeto, se tornando uma possibilidade de trabalho futuro.

A partir da estrutura construída, os próximos passos para implementação e estudo de possíveis de ataques à rede da *blockchain* deveriam partir do chamado ataque de gasto duplo. Esse tipo de ataque, conforme discutido na seção 2.2, é caracterizado quando um atacante, após realizar o pagamento e receber bens e serviços de um mercador tendo esperado o *lag* de confirmação requerido, divulga uma cadeia de blocos que seja maior do que a cadeia atualmente aceita pela rede. Ao fazê-lo, o atacante altera o histórico de

transações modificando a propriedade do pagamento que realizou pelos bens e serviços, ficando ao final tanto com a moeda quanto com o bem/serviço em questão.

Além deste ataque, outros tipos de ataque podem focar nos efeitos da disposição espacial da rede e dos parâmetros de transmissão para ataques do tipo *eclipse*. Nesse caso, o atacante procura isolar a vítima para conseguir controlar sua troca de mensagens com a rede e convencê-la da sua versão adulterada da cadeia.

8 REFERÊNCIAS BIBLIOGRÁFICAS

- BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. 3. ed. [S.l.]: Campus, 2015. 416 p.
- R. BOWDEN AND H. P. KEELER AND A. E. KRZESINSKI AND P. G. TAYLOR. Block arrivals in the Bitcoin blockchain. Disponível em: <<https://arxiv.org/abs/1801.07447v1>>. Acesso em: 20 set. 2019.
- COIN MARKET CAP. Total Cryptocurrencies Market Capitalization. Disponível em: <<https://coinmarketcap.com/charts/>>. Acesso em: 08 mai. 2019.
- DAVOS. Technology tipping points and social impact. Disponível em: <http://www3.weforum.org/docs/WEF_GAC15_Technological_Tipping_Points_report_2015.pdf> .*Acesso em : 08mai.2019.*
- DELOITTE. Blockchain in Public Sector. Disponível em: <<https://www2.deloitte.com/content/dam/Deloitte/in/Documents/public-sector/in-ps-blockchain-noexp.pdf>>. Acesso em: 08 mai. 2019.
- DIGICONOMIST. Bitcoin's energy consumption. Disponível em: <<https://digiconomist.net/bitcoin-energy-consumption>>. Acesso em: 08 mai. 2019.
- ESTADAO. Bacen blockchain. Disponível em: <<https://politica.estadao.com.br/blogs/fausto-macedo/banco-central-incentiva-fintechs-para-impulsionar-o-mercado-brasileiro/>>. Acesso em: 10 jul. 2019.
- FORBES. Blockchain global stock trading. Disponível em: <<https://www.forbes.com/sites/ericervin/2018/08/16/blockchain-technology-set-to-revolutionize-global-stock-trading/147c30724e56>>. Acesso em: 08 mai. 2019.
- GERVAIS, A.; KARAME, G.; WÜST, K.; GLYKANTZIS, V.; RITZDORF, H. ; CAPKUN, S. On the security and performance of proof of work blockchains. In: PROCEEDINGS OF THE 23ND ACM SIGSAC CONFERENCE ON COMPUTER AND COMMUNICATION SECURITY (CCS), 1., 2016. **Anais...** [S.l.: s.n.], 2016. Disponível em: <<https://eprint.iacr.org/2016/555.pdf>>. Acesso em: 23 set. 2019.
- INFOMONEY. Prazo e liquidação: entenda o que acontece após comprar ou vender uma ação. Disponível em: <<https://www.infomoney.com.br/educacao/guias/noticia/568491/prazo->

liquidacao-entenda-que-acontece-apos-comprar-vender-uma-acao>. Acesso em: 08 mai. 2019.

SATOSHI NAKAMOTO. Bitcoin: A Peer-to-Peer Electronic Cash System. Disponível em: <<https://bitcoin.org/bitcoin.pdf>>. Acesso em: 07 mai. 2019.

OCDE. Blockchains Unchained: Blockchain Technology and its Use in the Public Sector. Disponível em: <<https://www.oecd-ilibrary.org/docserver/3c32c429-en.pdf?expires=1557408120id=idaccname=guestchecksum=0EA96681C52F010AC756415C080C5D7D>>. Acesso em: 08 mai. 2019.

A. PINAR OZISIK AND BRIAN NEIL LEVINE. An Explanation of Nakamoto's Analysis of Double-spend Attacks. Disponível em: <<https://arxiv.org/pdf/1701.03977.pdf>>. Acesso em: 20 set. 2019.

PORTALBITCOIN. Ataque 51. Disponível em: <<https://portaldobitcoin.com/criptomoeda-e-alvo-de-ataque-de-51-e-prejuizo-pode-chegar-a-r-380-mil/>>. Acesso em: 08 mai. 2019.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software: Uma Abordagem Profissional**. 8. ed. [S.l.]: AMGH, 2016. 968 p.

MENI ROSENFELD. Analysis of Hashrate-Based Double Spending. Disponível em: <<https://arxiv.org/pdf/1402.2009.pdf>>. Acesso em: 20 set. 2019.

VALOR. Itaú Bradesco B3 blockchain. Disponível em: <<https://www.valor.com.br/empresas/4951308/itau-bradesco-e-b3-testam-tecnologia-blockchain>>. Acesso em: 07 mai. 2019.

WU, S.; CHEN, Y.; WANG, Q.; LI, M.; WANG, C. ; LUO, X. Cream: A smart contract enabled collusion-resistant e-auction. **IEEE Transactions on Information Forensics and Security**, v. 14, n. 7, p. 1687–1701, 2019.

OLIVER WYMAN. Blockchain in Capital Markets. Disponível em: <<https://www.oliverwyman.com/content/dam/oliverwyman/global/en/2016/feb/BlockChain-In-Capital-Markets.pdf>>. Acesso em: 08 mai. 2019.

9 APÊNDICES

APÊNDICE 1: CÓDIGO MODEL

```
from django.db import models
from django.urls import reverse
import uuid

class Simulation(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4,
                          help_text="ID único da simulação.")
    name = models.CharField(max_length=100, null=True)

    blockchain = models.ForeignKey(
        'Blockchain', on_delete=models.CASCADE, null=True)
    user = models.ForeignKey(
        'User', on_delete=models.CASCADE, null=True)
    energyCost = models.FloatField(null=False, help_text='Custo em R$/kWh.')
    energyCons = models.FloatField(
        null=False, help_text='Gasto de energia em kW.')
    minersCP = models.FloatField(
        null=False, default=14, help_text='Poder computacional dos mineradores.')
    lambda_prob = models.FloatField(
        null=False, default=0.8, help_text='Média da distribuição probabilística.')

    def __str__(self):
        return f'{self.id}'

class Blockchain(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4,
                          help_text='ID único da blockchain.')
    avg_time = models.FloatField(
```

```

        null=False, help_text='Tempo médio para geração de novos blocos.')
reward = models.FloatField(
    null=False, default=562505.875, help_text='Recompensa por minerar um bloco.')
```

```

class Block(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4,
                          help_text='ID único do bloco.')
    blockchain = models.ForeignKey(
        'Blockchain', on_delete=models.CASCADE, null=False)
```

```

class Event(models.Model):
    event_id = models.IntegerField(
        null=True, default=1, help_text='ID único do evento.')
    event_choices = ((1, 'Adição de bloco.'), (2, 'Exclusão de bloco.'),
                    (3, 'Inserção de minerador.'), (4, 'Exclusão de minerador.'), (5
time = models.FloatField(
    null=False, help_text='Tempo em que ocorreu o evento.')
typeOfEvent = models.IntegerField(
    null=False, choices=event_choices, help_text='Tipo de evento.')
blockchain = models.ForeignKey(
    'Blockchain', on_delete=models.CASCADE, null=False)
miner = models.CharField(max_length=10, null=True)
block = models.ForeignKey('Block', on_delete=models.CASCADE, null=True)
```

```

class User(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, help_text='ID único do
computPower = models.FloatField(null=True, help_text='Poder computacional do usuá
```

```

class Miner(models.Model):
    computPower = models.FloatField(
        null=False, help_text='Poder computacional do minerador.')
```

```
blockchain = models.ForeignKey(  
    'Blockchain', on_delete=models.CASCADE, null=True)
```


APÊNDICE 2: CÓDIGO CONTROLLER

```
from django.shortcuts import render
from simulator.models import Simulation, Blockchain, Block, Event, User, Miner
from django.shortcuts import render, get_object_or_404
from django.http import HttpResponseRedirect
from django.urls import reverse
from simulator.forms import createSimulForm
from django.http import JsonResponse
import datetime
import math
import numpy as np
from numpy.random import choice
import xlwt
from django.http import HttpResponse
from Crypto.Cipher import AES
import base64
import math

MASTER_KEY = "blockchain-simulator-encryption-key"

def encrypt_val(clear_text):
    enc_secret = AES.new(MASTER_KEY[:32])
    tag_string = (str(clear_text) +
                  (AES.block_size -
                   len(str(clear_text)) % AES.block_size) * "\0")
    cipher_text = base64.b64encode(enc_secret.encrypt(tag_string))

    return cipher_text
```

```

def decrypt_val(cipher_text):
    dec_secret = AES.new(MASTER_KEY[:32])
    raw_decrypted = dec_secret.decrypt(base64.b64decode(cipher_text))
    clear_val = raw_decrypted.decode().rstrip("\0")
    return clear_val

def create_simul(request):
    if request.method == "POST":
        dados = request.POST
        return JsonResponse(dados)
    return render(request, 'simulator/start.html')

def plotGraph(time, sid):
    # Set initial variables
    data = []
    label = []
    # Get objects
    simulation = Simulation.objects.get(id=sid)
    blockchain = simulation.blockchain
    user = simulation.user
    # Get params
    energyCost = simulation.energyCost
    energyCons = simulation.energyCons
    reward = blockchain.reward
    avg_time = blockchain.avg_time
    # Get variables
    totalCP = blockchain.get_total_cp(time)
    # Do the math
    num_miners = Miner.objects.filter(blockchain=blockchain).count()
    for time_interval in range(1801):
        time_interval_temp = time_interval
        # Ajustamos a média da poisson para o período de 1 dia
        miner_entered = np.random.poisson(

```

```

        simulation.lambda_prob*24*(1 - math.exp((-1)/num_miners)))
num_miners += miner_entered
totalCP += miner_entered*simulation.minersCP
# Estamos calculando o numero de blocos encontrados em média pelo usuário por
blocos_por_tempo_med = 60*user.computPower*time_interval / \
    (totalCP*avg_time)
custo = energyCost*energyCons*time_interval*3600
ganho_esperado = (reward)*blocos_por_tempo_med - custo
data.append(int(ganho_esperado))
log_string = ''
if time_interval >= 360:
    log_string += str(int(time_interval//360)) + "a "
    time_interval = time_interval % 360
if time_interval >= 30:
    log_string += str(int(time_interval/30)) + "m "
    time_interval = time_interval % 30
if time_interval > 0:
    log_string += str(int(time_interval)) + "d"
label.append(log_string)
time_interval = time_interval_temp
min_value = min(data)
max_value = max(data)
steps = (max_value - min_value)/10
dados = {'label': label, 'data': data,
        'min_value': min_value, 'max_value': max_value, 'steps': steps}
return dados

```

```

def save_blockchain(request):
    simulation_id = request.GET['sid']
    # simulation_id.strip("-")
    # print(simulation_id)
    content = encrypt_val(simulation_id)
    response = HttpResponse(
        content, content_type='application/force-download; charset=utf8')

```

```

response['Content-Disposition'] = 'attachment; filename="blockchain_details.bds"'
return response

# def load_blockchain(file):

def get_log(request):
    if request.method == "GET":
        response = HttpResponse(content_type='application/ms-excel')
        response['Content-Disposition'] = 'attachment; filename="blockchain_log.xls"'
        wb = xlwt.Workbook(encoding='utf-8')
        ws = wb.add_sheet("Log")
        row_num = 0
        font_style = xlwt.XFStyle()

        font_style.font.bold = True
        columns = ['Event type', 'Time', 'Miner (if applies)']
        for col_num in range(len(columns)):
            ws.write(row_num, col_num, columns[col_num], font_style)
        font_style = xlwt.XFStyle()
        simulation_id = request.GET['sid']
        simulation = Simulation.objects.get(id=simulation_id)
        blockchain = simulation.blockchain
        event = int(request.GET['e'])
        events = Event.objects.filter(
            blockchain=blockchain).filter(event_id__lte=event)
        for event_row in events:
            row_num = row_num + 1
            ws.write(row_num, 0, event_row.get_typeOfEvent_display(), font_style)
            ws.write(row_num, 1, event_row.time, font_style)
            if event_row.typeOfEvent == 1 and event_row.miner == 'user':
                ws.write(row_num, 2, 'User', font_style)
            else:
                ws.write(row_num, 2, '-', font_style)

```

```

dados = dict()
wb.save(response)
return response

def start_simul(request):
    if request.is_ajax():
        simulation_id = request.GET['sid']
        event = request.GET['e']
        time = request.GET['t']
        button = request.GET['operation']
        simulation = Simulation.objects.get(id=simulation_id)
        blockchain = simulation.blockchain
        latest_time = Event.objects.filter(
            blockchain=blockchain).latest('time').time
        latest_event = Event.objects.filter(
            blockchain=blockchain).count()
        if button == 'next_time':
            if int(time) < 0:
                time = 0
            while int(time) > latest_time:
                generate_events(180, simulation, latest_time)
                latest_time = Event.objects.filter(
                    blockchain=blockchain).latest('time').time
            num_events = Event.objects.filter(
                blockchain=simulation.blockchain).filter(time_lte=time).count()
        if button == 'next_event':
            if int(event) < 1:
                event = 1
            num_events = event
            if int(event) > latest_event:
                events_to_generate = int(event)-latest_event
                generate_events(events_to_generate, simulation, latest_time)
                time = Event.objects.filter(

```

```

        blockchain=blockchain).filter(event_id=int(event)).first().time
    else:
        time = Event.objects.filter(
            blockchain=blockchain).filter(event_id=int(event)).first().time
    dados = simulation.blockchain.get_num_info(time=time)
    print(dados)
    dados['time'] = int(time)
    dados['num_forks'] = Event.objects.filter(
        blockchain=blockchain).filter(typeOfEvent=5).count()
    dados['num_blocks'] = Block.objects.filter(
        blockchain=blockchain).count()
    dados['num_miners'] = Miner.objects.filter(
        blockchain=blockchain).count()
    dados['num_events'] = num_events
    dados_graph = plotGraph(time=time, sid=simulation_id)
    dados['dados_graph'] = dados_graph
    return JsonResponse(dados)
if request.method == "POST":
    if 'uploaded1' in request.FILES:
        upload1 = request.FILES['uploaded1'].read()
        data1 = ""
        for x in upload1:
            data1 = data1 + chr(x)
        # print(decrypt_val(data1))
        simulation = Simulation.objects.filter(
            id=decrypt_val(data1)).first()
        if simulation:
            blockchain = simulation.blockchain
            num_events = Event.objects.filter(
                blockchain=blockchain).count()
            num_miners = Miner.objects.filter(
                blockchain=blockchain).count()
            num_blocks = Block.objects.filter(
                blockchain=blockchain).count()
            num_forks = Event.objects.filter(

```

```

        blockchain=blockchain).filter(typeOfEvent=5).count()
last_time = Event.objects.filter(
    blockchain=blockchain).last().time
dados = {"simulation_id": f'{simulation.id}',
        "blockchain_id": f'{blockchain.id}',
        "num_miners": num_miners,
        "num_blocks": num_blocks,
        "num_events": num_events,
        "num_forks": num_forks,
        "time": int(last_time)}
print(dados)
return render(request, 'simulator/index.html', context=dados)
else:
    # Os próximos dados entrarão via post (formulário)
    avg_time = float(request.POST["avgTime"])
    # Custo dado em R$/kWh
    energCost = float(request.POST["energyCos"])
    # Consumo de energia dado em kW
    energCons = float(request.POST["energyCons"])
    # Poder computacional dado em TH/s
    ownCP = float(request.POST["ownCP"])
    minersCP = float(request.POST["minersCP"])
    reward = float(request.POST["reward"])
    lambda_prob = float(request.POST["medProb"])
    name = request.POST['simulName']

    # A partir daqui crio a blockchain, simulação, usuário e seto o primeiro p
    blockchain = Blockchain(avg_time=avg_time, reward=reward)
    blockchain.save()
    user = User(computPower=ownCP)
    user.save()
    simulation = Simulation(blockchain=blockchain, name=name,
                            energyCons=energCons, minersCP=minersCP, lambda_pr
    simulation.save()
    start_miner = Miner(blockchain=blockchain,

```

```

        computPower=minersCP)
start_miner.save()
event = Event(time=0, typeOfEvent=3, miner=start_miner,
              blockchain=blockchain)
event.save()
message = str(int(0)) + ' hora(s): Inserção de minerador.'

num_dados = generate_events(0, simulation)

dados = {"simulation_id": f'{simulation.id}',
         "blockchain_id": f'{blockchain.id}',
         "num_miners": num_dados['num_miners'],
         "num_blocks": num_dados['num_blocks'],
         "num_events": num_dados['num_events'],
         "num_forks": num_dados['num_forks'],
         "time": "0", }

dados_graph = plotGraph(time=0, sid=simulation.id)
dados['dados_graph'] = dados_graph
dados['simul_name'] = simulation.name
return render(request, 'simulator/index.html', context=dados)
else:
    return render(request, 'simulator/start.html')

def generate_events(num_events, simulation, time=0):
    events = 0
    while(events < num_events):
        # 0 valor do minerCP em MHash/s
        minersCP = simulation.minersCP
        # computPower em MHash/s
        userCP = simulation.user.computPower
        # TotalCP em MHash/s
        totalCP = simulation.blockchain.get_total_cp(time)
        # Average time em minutos
        avg_time = simulation.blockchain.avg_time

```



```

# Cálculo da dificuldade com total MHash
dificuldade = totalCP*avg_time*60
values = []
for i in range(5):
    values.append(np.random.exponential(scale=avg_time))
interval = round(np.average(values))
time += interval
have_fork = np.random.poisson(interval/avg_time)
if(have_fork > 1):
    last_id = Event.objects.filter(
        blockchain=simulation.blockchain).latest('event_id').event_id
    event = Event(time=time, event_id=last_id+1, typeOfEvent=5,
        blockchain=simulation.blockchain)
    event.save()

num_miners = simulation.blockchain.get_num_info(time)['num_miners']
user_prob = userCP/totalCP
random_num = np.random.random_sample()
last_id = Event.objects.filter(
    blockchain=simulation.blockchain).latest('event_id').event_id

block = Block(blockchain=simulation.blockchain)
block.save()

events += 1
if(random_num <= user_prob):

    event = Event(time=time, event_id=last_id+1, typeOfEvent=1,
        blockchain=simulation.blockchain, block=block, miner='user')
    event.save()

else:
    event = Event(time=time, event_id=last_id+1, typeOfEvent=1,
        blockchain=simulation.blockchain, block=block)
    event.save()

```

```

for i in range(int(interval)):
    miner_entered = np.random.poisson(
        simulation.lambda_prob)
    if(miner_entered > 0):
        for miner in range(miner_entered):
            miner = Miner(blockchain=simulation.blockchain,
                           computPower=minersCP)
            miner.save()
            last_id = Event.objects.filter(
                blockchain=simulation.blockchain).latest('event_id').event_id
            event = Event(time=time + i, event_id=last_id + 1, typeOfEvent=3,
                           blockchain=simulation.blockchain)
            event.save()

    events += 1

return simulation.blockchain.get_num_info(time)

```

APÊNDICE 3: CÓDIGO VIEW

9.3.1 INDEX.HTML

```
{% load static %}
<script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0/dist/Chart.min.js"></script>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js" type="text/javascript"></script>

<script>
    var simulation_id = "{{ simulation_id }}";

    var url_to_call = '{% url "start-simul" %}';
    var url_modal = '{% url "get-log" %}';
</script>

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Blockchain Simulator</title>
    <link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}" />
    <link rel="stylesheet" type="text/css" href="{% static 'css/modal.css' %}" />
</head>

<body>
    <div id="loading" class="loading">
    <header id="main-header">
        <div class="content">
            <p>Blockchain Simulator</p>
        </div>
    </header>
</body>
</html>
```

```

<div class="profile">
  Victor Milaré
</div>
</div>
</header>
<div class="main-content">
  <div class="principal">
    <div class="actions">
      <div class="simulation_info">
         {{simul_name}}
        <div class="search">
          <!-- Tempo (em dias) -->
          <input id="time" type="text" placeholder="" />
          <nav class="arrows">
            
            
          </nav>
          <button id="ir_time">Ir</button>
        </div>
        <div class="search">
          <!-- Evento -->
          <input type="text" id="event" class="event" placeholder="35" />
          <nav class="arrows">
            
            
          </nav>
          <button id="ir_event">Ir</button>
        </div>
      </div>
    </div>
  </div>
  <form action="api/log/" method="GET">
    <input id="simulation_id" type="hidden" name="sid" value="{{simulation_id}}>
    <input id="event_hidden" type="hidden" name="e" value="1">

    <button id="log-show">Salvar log</button>

```

```

</form>
<form action="api/save/" method="GET">
  <input id="simulation_id" type="hidden" name="sid" value="{{simulation_id}}>

  <button id="chain-save">Salvar simulação</button>

</form>
</div>
<div class="info">
  <ul>
    <li>
      <strong>
        <miners id="miners">12</miners>
      </strong><span>Mineradores</span>
    </li>
    <li>
      <strong>
        <blocks id="blocks">20</blocks>
      </strong><span>Blocos minerados</span>
    </li>
    <li>
      <strong>
         <fork id="num_forks">
      </strong>
    </li>
    <li>
      <strong><time
        id="time_trans">22</time></strong><span>Minutos transcorridos</span>
    </li>
    <li>
      <strong id="events_occ">
        <events id="events">35</events>
      </strong><span>Eventos ocorridos</span>
    </li>
  </ul>

```



```

var event_input = document.getElementById("event");
time_input.placeholder = result.time + " minutos";
event_input.placeholder = result.num_events + " evento(s)";
document.getElementById("miners").innerHTML = result.num_miners;
document.getElementById("blocks").innerHTML = result.num_blocks;
document.getElementById("time_trans").innerHTML = result.time;
document.getElementById("time_trans_graph").innerHTML = result.time;
document.getElementById("events").innerHTML = result.num_events;
document.getElementById("events_graph").innerHTML = result.num_events;
document.getElementById("num_forks").innerHTML = result.num_forks;
}
</script>
<!-- # End function to set values -->
<!-- # Loads modal scripts -->
<script type="text/javascript" src="{% static 'js/modal.js' %}"></script>
<!-- # End modal scripts -->
<!-- # Function to set ajax requests -->
<script type="text/javascript" src="{% static 'js/get_content.js' %}"></script>
<!-- # End function to set ajax requests -->
<!-- # Function to load values when doc is ready -->
<script>
$(document).ready(function () {
    // console.log("{{dados_graph.data}}");
    var time_input = document.getElementById("time");
    var event_input = document.getElementById("event");
    time_input.placeholder = "{{time}} minutos";
    event_input.placeholder = "{{num_events}} evento(s)";
    results = {
        num_miners: "{{num_miners}}",
        num_blocks: "{{num_blocks}}",
        time: "{{time}}",
        num_events: "{{num_events}}",
        num_forks: "{{num_forks}}"
    };
    // console.log(results);

```

```

    setValues(results);
    set_content("{{time}}", "{{num_events}}", "next_time");
  });
$(document).on("click", "#log-show", function(){
  var event = document.getElementById("events").innerHTML.toString();
  var event_hidden = document.getElementById("event_hidden").value = event;
});
// $(document).on("click", "#chain-save", function(){
//   var event = document.getElementById("events").innerHTML.toString();
//   var event_hidden = document.getElementById("event_hidden").value = event;
// });
</script>
<!-- # End function to load values when doc is ready -->

```

9.3.2 START.HTML

```

{% load static %}

<!DOCTYPE html>
<html lang="en">
  <script
    src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"
    type="text/javascript"
  ></script>

  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Blockchain Simulator</title>
    <link rel="stylesheet" href="{% static 'css/start_style.css' %}" />
  </head>

  <body>
    <header id="main-header">

```



```

<div class="content">
  <p>Blockchain Simulator</p>
  <div class="profile">
    <a href="" class="login">Login</a><a href="">Sobre nós</a>
    <a href="">Ajuda</a>
  </div>
</div>
</header>
<div class="main-content">
  <div class="central">
    <div class="title">
      Inicie sua simulação<span class="subtitle"
      >Preencha os dados da simulação abaixo ou escolha uma configuração
      padrão (bitcoin). Você também pode <a href="">se logar</a> e
      escolher entre suas blockchains salvas.</span
      >
    </div>
    <div class="container-exemplo">
      <div id="personalizado_box" class="box-selected">
        <div class="checkbox">
          <label class="container">
            <input type="checkbox" id="personalizado" checked="checked" />
            <span class="checkmark"></span>
          </label>
        </div>
        <div id="personalizado_title" class="title-selected">
          Personalizada
        </div>
        <div id="personalizado_description" class="description-selected">
          Insira cada parâmetro de configuração de sua blockchain e de sua
          simulação.
        </div>
      </div>
      <div id="bitcoin_box" class="box-not-selected">
        <div class="checkbox">

```

```

<label class="container">
  <input id="bitcoin" type="checkbox" unchecked />
  <span class="checkmark"></span>
</label>
</div>
<div id="bitcoin_title" class="title-not-selected">Bitcoin</div>
<div id="bitcoin_description" class="description-not-selected">
  Configuração pré-carregada com os parâmetros do Bitcoin.
</div>
</div>
</div>
<form
  action="{% url 'start-simul' %}"
  method="post"
  enctype="multipart/form-data"
>
  {% csrf_token %}
  <div class="title">Configurações da Simulação</div>
  <div class="config">
    <fieldset>
      <label for="simulName">Nome da simulação: </label>
      <input
        type="text"
        name="simulName"
        id="simulName"
        placeholder="Simulação #1 (ex.)"
      />
    </fieldset>
    <fieldset>
      <label for="energyCos">Custo energético (em R$/kWh): </label>
      <input
        type="text"
        name="energyCos"
        id="energyCos"
        placeholder="0.43 (ex.)"
      />
    </fieldset>
  </div>
</form>

```

```

/>
</fieldset>
<fieldset>
  <label for="energyCons">Consumo energético (em kW): </label>
  <input
    type="text"
    name="energyCons"
    id="energyCons"
    placeholder="1.3 (ex.)"
  />
</fieldset>
<fieldset>
  <label for="ownCP"
    >Poder computacional do usuário (em TH/s):
  </label>
  <input
    type="text"
    name="ownCP"
    id="ownCP"
    placeholder="14 (ex.)"
  />
</fieldset>
<fieldset>
  <label for="minersCP"
    >Poder computacional dos outros mineradores (em TH/s):</label
  >
  <input
    type="text"
    name="minersCP"
    id="minersCP"
    placeholder="14 (ex.)"
  />
</fieldset>
<fieldset>
  <label for="distProb"

```

```

    >Distribuição probabilística de entrada de novos
    mineradores:</label
  >
  <select name="distProb" id="distProb">
    <option value="poisson">Poisson</option>
  </select>
</fieldset>
<fieldset>
  <label for="medProb">Média da distribuição probabilística:</label>
  <input
    type="text"
    name="medProb"
    id="medProb"
    placeholder="0.8 (ex.)"
  />
</fieldset>
</div>
<div class="title">Configurações da Blockchain</div>
<div class="config">
  <fieldset>
    <label for="reward">Recompensa por minerar (em R$): </label>
    <input
      type="text"
      name="reward"
      id="reward"
      placeholder="562505 (ex.)"
    />
  </fieldset>
  <fieldset>
    <label for="avgTime"
      >Tempo médio para geração de blocos (em min.):
    </label>
    <input
      type="text"
      name="avgTime"

```

```

        id="avgTime"
        placeholder="10 (ex.)"
    />
</fieldset>
<fieldset>
    <label for="avgTime">Configs iniciais (opcional .bds): </label>
    <input type="file" name="uploaded1" accept=".bds" />
</fieldset>
</div>
<div></div>
<div class="submit">
    <button class="initSimul" type="submit">Iniciar simulação</button>
</div>
</form>
</div>
</div>
</body>
<script type="text/javascript" src="{% static 'js/select.js' %}"></script>
</html>

```

9.3.3 GETCONTENT.JS

```

$("#prev_event").click(function (e) {
    var prev_event = parseInt(document.getElementById('events').innerHTML.toString()) -
    var time = "";
    set_content(time, prev_event, "next_event");
});

```

```

$("#next_event").click(function (e) {
    var next_event = parseInt(document.getElementById('events').innerHTML.toString()) +
    var time = "";
    set_content(time, next_event, "next_event");
});

```

```

$("#prev_time").click(function (e) {

```

```

    var prev_time = parseInt(document.getElementById('time_trans').innerHTML.toString());
    var event = "";
    set_content(prev_time, event, "next_time");
});

$("#next_time").click(function (e) {
    var next_time = parseInt(document.getElementById('time_trans').innerHTML.toString());
    var event = "";
    set_content(next_time, event, "next_time");
});

$("#ir_time").click(function (e) {
    var time = $("#time").val();
    var event = "";
    set_content(time, event, "next_time");
});

$("#ir_event").click(function (e) {
    var event = $("#event").val();
    var time = "";
    set_content(time, event, "next_event");
});

function set_content(time, event, operation) {
    $.ajax({
        type: "GET",
        url: url_to_call,
        data: {
            sid: simulation_id,
            e: event,
            t: time,
            operation: operation
        },
        beforeSend: function () {
            $("#loading").css("visibility", "visible");
        }
    });
}

```

```

},
complete: function () {
    $("#loading").css("visibility", "hidden");
},
success: function (result) {
    resetCanvas();
    setValues(result);
    var ctx = document.getElementById("graph").getContext("2d");
    var config = {
        type: "line",
        data: {
            labels: result.dados_graph.label,
            datasets: [
                {
                    // label: false,
                    // backgroundColor: window.chartColors.red,
                    // borderColor: window.chartColors.red,
                    data: result.dados_graph.data,
                    fill: false,
                    pointRadius: 0
                }
            ]
        },
        options: {
            legend: {
                display: false
            },
            responsive: true,
            title: {
                display: false
            },
            scales: {
                xAxes: [
                    {
                        display: true,

```

```

        scaleLabel: {
            display: true,
            labelString: "Tempo (Minutos)"
        },
        ticks: {
            autoSkip: true,
            maxTicksLimit: 10
        }
    }
],
yAxes: [
    {
        display: true,
        scaleLabel: {
            display: true,
            labelString: "Lucro/Prejuízo"
        },
        ticks: {
            min: result.dados_graph.min_value,
            max: result.dados_graph.max_value,
            // forces step size
            stepSize: result.dados_graph.steps,
            callback: function (value) {
                return addCommas(value);
            }
        }
    }
]
}
};

// console.log(typeof graph);
// if (typeof graph !== "undefined") {
//     console.log('destroy');
//     graph.destroy();

```



```

    // }
    var graph = new Chart(ctx, config);
  },
  error: function (result) {
    alert("error");
  }
});
}
function addCommas(nStr) {
  // console.log("comma");
  nStr += "";
  nStr = nStr.replace(".", ",");
  x = nStr.split(",");
  x1 = x[0];
  x2 = x.length > 1 ? "," + x[1] + "0" : ",00";
  var rgx = /(\d+)(\d{3})/;
  while (rgx.test(x1)) {
    x1 = x1.replace(rgx, "$1" + "." + "$2");
  }
  positive = x1.split("-");
  formatted =
    positive.length > 1 ? "- R$ " + positive[1] + x2 : "R$ " + x1 + x2;
  return formatted;
}
function resetCanvas() {
  $('#graph').remove(); // this is my <canvas> element
  $('#graph-container').append('<canvas id="graph" height="500px">a</canvas>');
}

```