UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Pablo Rinco Montezano

SISTEMA DIGITAL PARA NOTIFICAÇÕES BASEADO EM BLOCKCHAIN

Florianópolis

2018

Pablo Rinco Montezano

SISTEMA DIGITAL PARA NOTIFICAÇÕES BASEADO EM BLOCKCHAIN

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador: Prof. Dr. Ricardo Custódio

Florianópolis

2018

Pablo Rinco Montezano

SISTEMA DIGITAL PARA NOTIFICAÇÕES BASEADO EM BLOCKCHAIN

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovado em sua forma final pelo Programa de Graduação em Ciências da Computação.

Florianópolis, 30 de outubro 2018.

Prof. Dr. Rafael Luiz Cancian
Coordenador do Curso

Banca Examinadora:

Martin Augusto Gagliotti Vigil

Prof. Dr. Ricardo Custódio
Orientador

Ricardo Pereira e Silva

Dedico esse trabalho à meu pai, In Memorian, que não pôde acompanhar nesses últimos anos meu progresso na vida acadêmica, mas sempre acreditou e incentivou a busca pelo conhecimento.

AGRADECIMENTOS

Agradeço à toda a minha família, que dentro de seus limites, forneceram todo o apoio necessário para que pudesse seguir a vida acadêmica. Em especial, agradeço Valéria Gontarczyk que me tornou uma pessoa completa. Às pessoas que caminharam comigo juntos nessa jornada. Em especial ao Lucas Perin, sempre presente nos momentos de dificuldade, por me incentivar e acreditar no meu potencial. Ao meu orientador Ricardo Custódio por partilhar seu conhecimento e fomentar minha busca pelo aprimoramento pessoal. Ao LabSEC por me acolher e contribuir fortemente para a minha formação acadêmica e profissional. Agradeço também à todos membros do LabSEC por todo companheirismo e conversa jogada fora que tivemos nos últimos anos.

Escuta Genos, você tem que continuar! Não importa o quão difícil fique. Faça 100 abdominais, faça 100 flexões e faça 100 agachamentos. E depois corra 10 quilômetros. Faça isso todo o santo dia! [...] Mas nunca use ar condicionado no verão ou aquecedor no inverno. Isso vai fortalecer a sua mente.

Saitama

RESUMO

A adoção de ferramentas tecnológicas digitais cresceu em várias áreas inclusive no meio jurídico. Atualmente, os meios pelos quais são realizadas as notificações, muitas vezes carecem de agilidade e garantias, além do alto custo aos cofres públicos, cartórios e para o cidadão. Visando atingir principalmente estes três aspectos, o objetivo do presente trabalho é permitir a digitalização dos meios de notificação previstos na legislação brasileira. Para alcançar os objetivos propostos, foram aplicadas algumas tecnologias atuais com o intuito de mitigar a ineficiência dos métodos convencionais de notificação e permitir que a ferramenta implementada seja extensível a tecnologias futuras.

Palavras-chave: Notificação, cartórios, blockchain, segurança da informação

LISTA DE FIGURAS

Figura 1 Estrutura básica de função de hash criptográfico. Fonte: Cryptography and Network Security: Principles and Practice. 26
Figura 2 Funcionamento básico de uma assinatura digital. Fonte: How do Digital Signatures keep you protected?(KUNDU,) 27
Figura 3 Estrutura básica de uma blockchain. Fonte: adaptação de Bitcoin: A Peer-to-Peer Electronic Cash System 30
Figura 4 Diagrama de relacionamento entre os componentes. Fonte: Elaboração própria 42
Figura 5 Diagrama de sequência da interface web. Fonte: Elaboração própria 43
Figura 6 Diagrama de sequência das interações por email. Fonte: Elaboração própria 44
Figura 7 Diagrama de sequência do intermediador. Fonte: Elaboração própria 44
Figura 8 Diagrama de sequência das chamadas dos notificadores. Fonte: Elaboração própria 45
Figura 9 Diagrama UML do contrato inteligente *Descentralized-Notification*. Fonte: Elaboração própria 48
Figura 10 Diagrama da descrição formal das interfaces REST. Fonte: Elaboração própria 58
Figura 11 Página inicial. Fonte: Elaboração própria 59
Figura 12 Página de autenticação. Fonte: Elaboração própria 60
Figura 13 Página para requisição de notificação. Fonte: Elaboração própria 60
Figura 14 Página principal. Fonte: Elaboração própria 61

LISTA DE TABELAS

Tabela 1 Custos com servidores e banco de dados 64
Tabela 2 Custos da blockchain ethereum 64
Tabela 3 Informações das formas de notificação 66

LISTA DE ABREVIATURAS E SIGLAS

RSA	Rivest Shamir Adleman
FIPS	Federal Information Processing Standard
DSA	Digital Signature Algorithm
NIST	National Institute of Standards and Technology
ECSA	Elptic Curve Digital Signature Algorithm
BGP	Byzantine General Problem
PBFT	Practical Byzantine Fault Tolerance
PoW	Proof of Work
DDoS	Distributed Denial of Service
CPC	Código de Processo Civil
Art.	Artigo
AR	Aviso de Recebimento
CPF	Cadastro de Pessoa Física
ICP	Infraestrutura de Chaves Públicas
PKCS	Public Key Cryptography Standards
MVC	Model Vision Control
SMS	Small Message Service
S/MIME	Secure/Multipurpose Internet Mail Extensions

LISTA DE SÍMBOLOS

H	Função *hash*
mod	Operação módulo
Φ	Phi de N
PU	Chave pública
PK	Chave privada

SUMÁRIO

1 INTRODUÇÃO 23
1.1 OBJETIVOS 23
1.1.1 OBJETIVOS ESPECÍFICOS 24
1.2 METODOLOGIA 24
1.3 ORGANIZAÇÃO DO TRABALHO 24
2 FUNDAMENTAÇÃO TEÓRICA 25
2.1 HASH 25
2.2 ASSINATURA DIGITAL 26
2.2.1 Algoritmos 27
2.2.1.1 Exemplo 28
2.3 BLOCKCHAIN 30
2.3.1 Algoritmos de Consenso 30
2.3.1.1 Practical Byzantine Fault Tolerance 31
2.3.2 Prova de Trabalho 31
2.3.3 Incentivos para se manter a rede descentralizada em execução 32
2.4 CONTRATOS INTELIGENTES 32
2.5 ETHEREUM 33
2.6 NOTIFICAÇÕES 34
2.6.1 Citação 35
2.6.2 Notificação Extrajudicial 38
2.6.3 Notificações Judiciais 39
2.6.4 Intimação 40
3 NOTIFICAÇÕES DIGITAIS 41
3.1 DESCRIÇÃO 41
3.2 ARQUITETURA DO PROJETO 41
3.2.1 Interface com Usuário 41
3.2.2 Sistemas Auxiliares 43
3.2.3 Sistema de Notificações 45
4 DESENVOLVIMENTO 47
4.1 CONTRATOS INTELIGENTES 47
4.1.1 Descrição do Contrato 47
4.1.2 Otimizações do Contrato Inteligente 50
4.2 REDE PRIVADA ETHEREUM 50
4.3 SERVIDOR DE EMAIL 53
4.4 SERVIDORES DA APLICAÇÃO 53
4.4.1 Servidor Intermediador 54

4.4.1.1 Implementação 54
4.4.1.2 Especificação dos Contratos de Entrada e Saída 55
4.5 SERVIDOR WEB 57
4.6 PÁGINA WEB 58
5 DISCUSSÃO 63
5.1 VANTAGENS DO SISTEMA 63
5.1.1 Redução do Custo 63
5.1.2 Facilidade de Entrega 65
5.1.3 Agilidade na Entrega 65
5.1.4 Comprovação de entrega 66
5.2 PONTOS FRACOS 66
5.2.1 Publicação em Blockchain Pública 67
5.2.2 Garantia de Entrega de Email 67
6 CONCLUSÃO 69
6.1 TRABALHOS FUTUROS 69
REFERÊNCIAS 71
APÊNDICE A – Artigo 75
ANEXO A – Código 91

1 INTRODUÇÃO

Os cartórios de títulos e documentos estão presentes em inúmeras fases da vida dos brasileiros (Moreira, 2017). Vários serviços são prestados e todos têm como finalidade garantir alguma forma de segurança. A burocracia inerente aos cartórios é muitas vezes é um dos responsáveis por trazer as garantias e veracidade dos documentos processados. Em artigo escrito ao site jusbrasil[1] (Isonal, 2016) define o servidor notarial ou registral como sendo *"um conselheiro comunitário, um agente de prevenção de litígios, visando a paz social"*.

O interesse pela digitalização dos serviços notariais tem crescido, seja para agilizar os serviços ou para garantir a segurança dos dados(Falcão, 2013). Ao longo dos anos várias propostas de digitalização de alguns serviços foram elaboradas como a emissão de registro de nascimento pela internet proposta por Dantas(DANTAS, 2001), mas não foram aplicados em sua plenitude.

Embora alguns serviços simples possam ser realizados pela internet(GUIMARãES, 2014), a maioria ainda depende do deslocamento do cidadão até um cartório de documentos e registro. Dentre estes está as notificações extrajudiciais.

As notificações judiciais e extrajudiciais, intimações e citações são muito comuns pois tratam de assuntos do cotidiano como por exemplo, notificar um cliente sobre um débito. Devido à seriedade deste serviço, sua execução toma tempo e pode ser dificultada pelo notificado ao se esquivar do oficial de justiça ou recusando a assinar o comprovante de entrega do documento.

Visando aprimorar os serviços de notificação, este trabalho tem como objetivo auxiliar estes serviços agregando tecnologias modernas como blockchain, redes sociais e comunicação por dispositivos móveis, ampliando o alcance à pessoa notificada.

1.1 OBJETIVOS

O presente trabalho tem como principal trabalho, elaborar um sistema que permita a inclusão de várias tecnologias de comunicação que permitam um maior alcance e visibilidade das notificações judiciais, extrajudiciais, intimações e citações.

[1] https://www.jusbrasil.com.br/

1.1.1 OBJETIVOS ESPECÍFICOS

- Analisar as leis que regem as notificações
- Compreender o funcionamento das notificações
- Elaborar um projeto de um sistema que atenda às formas atuais de notificação
- Implementar o sistema

1.2 METODOLOGIA

Devido à natureza do problema, a primeira etapa realizada foi o estudo e análise da legislação sobre notificações judiciais, extrajudiciais, intimações e citações. Para que o presente trabalho pudesse ter continuidade, foi feita a revisão da Lei n° 11.419[2] que trata da informatização do processo judicial, comunicação eletrônica dos atos processuais e o processo eletrônico.

Tendo adquirido o conhecimento necessário da legislação que rege os assuntos a serem abordados, iniciou-se o estudo das possíveis tecnologias a serem adotadas no processo de desenvolvimento do projeto. As tecnologias escolhidas necessitariam estar em conformidade com as leis brasileiras, direcionando a pesquisa apenas para tecnologias regulamentadas no país.

Enfim foi desenvolvido o protótipo do sistema de notificações de forma a atender às especificidades do problema como: privacidade, tempo de entrega e garantia de entrega da notificação. O protótipo também conta com a capacidade de ampliação do sistema para se manter atualizado com novos meios de notificação que surjam no futuro.

1.3 ORGANIZAÇÃO DO TRABALHO

Os capítulos que seguem estão organizados da seguinte forma: no capítulo 2 é levantado os principais conceitos utilizados para compreender e elaborar o sistema. O capítulo 3 discorre como é desenvolvido o projeto e o sistema da solução elaborada pelo trabalho. O capítulo 5 trás a discussão da solução, seus pontos positivos e negativos. Por fim, o capítulo 6 apresenta as considerações finais.

[2] http://www.planalto.gov.br/ccivil_03/_Ato2004-2006/2006/Lei/L11419.htm

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo estão descritos os conceitos básicos que formam a base do Blockchain e a criptografia moderna.

2.1 HASH

Como aponta Cornel (CORNELL,), uma função hash é uma função que recebe dados de tamanho arbitrário e produz dados h = H(x) de tamanho fixo, onde geralmente, o tamanho da entrada é maior que o tamanho da saída. Quando apenas um bit do dado de entrada é modificado, aproximadamente metade dos bits do valor da saída é modificado. Este efeito é chamado de efeito avalanche. As características básicas de uma função hash são:

1. Tamanho variável da entrada: H pode ser aplicada a um bloco de dados de qualquer tamanho
2. Tamanho fixo da saída: o código hash resultante da função H tem o mesmo tamanho para qualquer entrada de dados
3. Eficiência: a verificação de H(x) precisa ser de fácil verificação, viabilizando sua utilização

Dentre as classes de funções hash estão as chamadas funções hash criptográficas que além das características apresentadas acima dispõe também:

1. Resistente à pre-imagem (caminho único): dado um hash h, é impraticável encontrar um y tal que H(y) = h
2. Resistente a segunda pre-imagem (baixa resistência a colisão): Para qualquer bloco x, é impraticável encontrar um y $\neq$ x em que H(x) = H(y)
3. Resistente a colisão (forte resistência a colisão): Impraticável encontrar um par (x, y) em que H(x) = H(y)
4. Pseudo-aleatoriedade: Saída de H atende a padrões de pseudo aleatoriedade

A figura 1 apresenta o esquemático da estrutura básica de uma função de hash criptográfico.

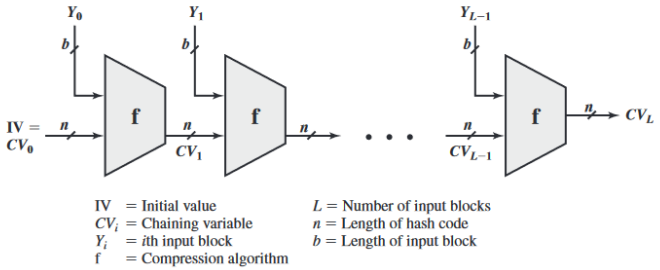

Figura 1 – Estrutura básica de função de hash criptográfico. Fonte: Cryptography and Network Security: Principles and Practice.

2.2 ASSINATURA DIGITAL

A noção de assinatura digital foi introduzida em 1975 por Diffie e Hellman, propondo a utilização de uma função "arapuca", isto é, uma função que é fácil de calcular em uma direção, mas difícil de calcular a sua inversa sem o conhecimento de outras informações. Apenas um ano depois, **R**ivest, **S**hamir e **A**dleman apresentaram um algoritmo que implementa a teoria proposta por Diffie-Hellman, chamado RSA. Em suma, este algoritmo introduz o conceito de chave assimétrica, no qual existe duas chaves criptográficas: uma pública e uma privada. O processo para a geração de uma assinatura digital básica é realizado da seguinte forma:

1. Calcular o hash do documeto que será assinado, utilizando qualquer função hash.
2. Cifrar este hash com a chave privada do usuário que deseja assinar.
3. Anexar ao documento o hash cifrado e informações para a verificação da assinatura.

Para realizar a verificação da assinatura basta fazer:

1. Calcular o hash do documento original.
2. Decifrar o hash do documento assinado.
3. Comparar os dois hashs, caso sejam iguais a assinatura é válida, senão inválida.

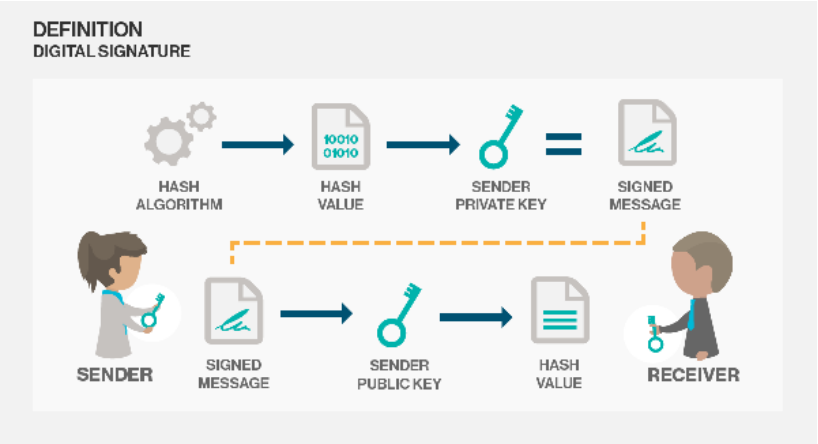

Figura 2 – Funcionamento básico de uma assinatura digital. Fonte: How do Digital Signatures keep you protected?(KUNDU,)

Sobretudo, uma assinatura digital se difere de uma assinatura manuscrita devido a três propriedades que ela proporciona:

- Autenticidade: O destinatário pode confirmar se quem assinou o documento é realmente quem diz ser. Caso a verificação ocorra com sucesso, o destinatário pode concluir que quem assinou aquele documento foi o proprietário daquela chave pública, dado que é praticamente impossível gerar uma colisão de chaves.
- Integridade: O destinatário pode confirmar se o documento foi alterado após a assinatura. Uma vez que uma pequena modificação no documento gera um hash totalmente diferente, no momento da verificação da assinatura os dois hashs não serão equivalentes.
- Não-repúdio: O signatário não pode negar que assinou um devido documento. Dado que a chave pública do signatário está associada a um certificado digital e a verificação pode ser feita sem o consentimento do signatário, é possível identificar quem realizou a assinatura e este não pode negá-lá.

2.2.1 Algoritmos

Além do uso de algoritmos de hashs, assinaturas digitais utilizam algoritmos de criptografia assimétrica:

- Geração de um par de chaves (pública e privada)
- Função de cifragem dos dados (algoritmo de assinatura)
- Função de decifragem dos dados (algoritmo de verificação)

Atualmente, três algoritmos são descritos na FIPS 186-4 como algoritmos aprovados para uso em assinaturas digitais, são eles:

- RSA: O primeiro e até hoje o mais utilizado.
- DSA: Proposto pelo NIST em 1991 para uso nos padrões de assinatura digital (DSS).
- ECDSA: Uma variante do DSA que utiliza curvas elípticas.

Todos implementam os três principais algoritmos de criptografia assimétrica.

2.2.1.1 Exemplo

Para este exemplo, optou-se pelo algoritmo RSA.

Geração do par de chaves
Elementos necessários:

- dois números primos p e q (privados, escolhidos)
- n = pq (público, calculado)
- e, relativamente primo à Φ(n) (público, escolhido)
- d, inverso multiplicativo de e mod(Φ(n)) (privado, calculado)

Assim sendo, d e e são números inversos multiplicativos mod($\Phi(n)$). Ou seja, ed mod (Φ(n)) = 1. Exemplo numérico:

1. Selecionar dois números primos, p = 17 e q = 11
2. Calcular n = pq = $17x11$ = 187
3. Calcular $\Phi(n)$ = (p - 1)(q - 1) = $16x10$ = 160
4. Selecionar e tal que seja relativamente primo à $\Phi(n)$ e $< \Phi(n)$, e = 7

5. Determinar d tal que $de \bmod \Phi(n) = 1$. Neste caso, $d = 23$ pois $23x7 = 161 \bmod \Phi(n) = 1$

Logo, os pares são: $PU = \{7, 187\}$ e $PK = \{23, 187\}$. Porém, não existe uma regra para a definição da chave pública e privada. Como ambas são inversas multiplicativas de mod $(\Phi(n))$, pode-se dizer que elas são chaves equivalentes. Sendo assim, uma chave só se difere da outra no momento em que o proprietário da chave decide publicar uma delas, tornando-a a chave pública.

Cifragem de dados

Para cifrar utilizando a chave pública:

$$C = M^e mod(n)$$

Para cifrar utilizando a chave privada:

$$C = M^d mod(n)$$

Exemplo numérico utilizando as chaves geradas anteriormente:
Dado $M = 88$ e utilizando a chave privada para cifragem, tem-se que:

1. $C = 88^{23} mod(187)$
2. $C = 11$

Decifragem de dados

Para a decifragem é só realizar o caminho contrário. Para decifrar utilizando a chave privada:

$$M = C^e modn$$

Para decifrar utilizando a chave pública:

$$M = C^d modn$$

Exemplo numérico utilizando a assinatura gerada anteriormente:
Dado que $C = 11$ e utilizando a chave pública para decifragem:

1. $M = 11^7 mod(187)$
2. $M = 88$

2.3 BLOCKCHAIN

Em poucas palavras, Blockchain é uma sequência de blocos encadeados um nos outros. A primeira utilização da blockchain como conhecemos atualmente foi proposta por (NAKAMOTO, 2008) embora o conceito exista desde 1980 como proposto por (FIPS, 1980).

Este encadeamento garante a imutabilidade da sequência de blocos, pois um bloco está diretamente ligado ao seu antecessor e assim sucessivamente. Um dos métodos mais simples de encadear blocos digitais é através da utilização de funções de resumo criptográfico (Hash criptográfico). Neste modelo de encadeamento, um bloco seria composto pelos dados do bloco (geralmente denominados transações) e metadados compostos pelo resumo criptográfico do bloco anterior e outras informações importantes para a aplicação. O esquemático apresentado na figura 3 descreve a estrutura básica de uma blockchain.

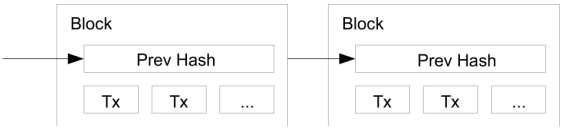

Figura 3 – Estrutura básica de uma blockchain. Fonte: adaptação de Bitcoin: A Peer-to-Peer Electronic Cash System

Embora o encadeamento de blocos seja poderoso o suficiente a ponto de permitir que uma Blockchain funcione como um banco de dados distribuído e descentralizado, por si só, não é suficiente para resolver alguns dos problemas que surgem em sistemas com estas características, como por exemplo, o consenso.

2.3.1 Algoritmos de Consenso

Alcançar consenso em uma rede de nodos onde não há confiança (e certamente não se conhecem) é um dos maiores problemas de sistemas distribuídos. Este problema é reduzido ao Problema dos Generais Bizantinos.

O problema dos Generais Bizantinos foi inicialmente apresentado através da seguinte analogia: "O general do exército bizantino deseja atacar seu inimigo, mas para que o ataque seja bem sucedido, todos os comandantes do seu exército precisam entrar em consenso entre atacar, ou

bater em retirada. O problema surge, quando existe a necessidade de enviar a mensagem do general aos comandantes, mas pode haver um comandante traidor, que para sabotar o ataque, repassa a mensagem errada para os demais comandantes."Este problema foi apresentado com este nome por (TLAMPORT, 1982) com foco em solucionar casos onde "Sistemas de computadores confiáveis precisam lidar com componentes que dão informações conflitantes para diferentes partes do sistema". Embora a solução encontrada seja eficaz, ela está longe de ser eficiente como o próprio autor apresentou.

2.3.1.1 Practical Byzantine Fault Tolerance

Em 1999, Castro e Liskov (LISKOV, 1999) apresentaram um algoritmo chamado de "Practical Byzantine Fault Tolerance"que serviu como a base para a maioria dos algoritmos de consenso que conhecemos hoje.

Para alcançar consenso, o PBFT utiliza o conceito de votação: através de algumas regras definidas: um nodo *primário* é selecionado e será responsável por indicar o próximo bloco. O algoritmo funciona contanto que pelo menos 2/3 dos nodos da rede sejam honestos. É necessário também, que exista algum tipo de identificação dos nodos de forma a permitir que haja a votação do nodo *primário*.

O trabalho de Castro e Liskov, aprimorou consideravelmente a solução do problema dos Generais Bizantinos, mas ainda sim inviável para a maioria das aplicações. Um dos desafios de se utilizar este algoritmo em blockchains se dá pois a identidade dos nodos precisa ser conhecida.

2.3.2 Prova de Trabalho

Em 2009 Nakamoto, teve como uma de suas principais contribuições o primeiro algoritmo de consenso realmente praticável e baseado no PBFT, o "Proof of Work"(Prova de trabalho). Em sua forma mais simples, a prova de trabalho visa provar para os demais, que algum determinado tipo de trabalho foi desenvolvido, dando direito ao ganhador da prova, opinar na decisão a ser tomada.

Mais especificamente, no Bitcoin (e outras criptomoedas), a prova de trabalho funciona da seguinte forma:

- Encontrar o *hash* de um bloco de dados pré-determinado onde H(x) = y tal que $y \in \{0,1\}^*$
- Procurar aleatoriamente uma sequência de bits $x \in \{0,1\}^*$ tal que $H(yx) = z$ onde os n bits mais significativos de $z \in \{0,1\}^+$ são zero.
- Repetir o processo acima até encontrar um valor de n que satisfaça a condição da segunda etapa

O consenso da prova de trabalho se encontra no fato de que a primeira pessoa que encontrar o número n, tem direito de ditar quais serão os dados do próximo bloco. Em outras palavras, o "poder de voto"de cada pessoa é diretamente equivalente ao seu poder computacional, já que várias iterações dos passos acima serão necessários até que se encontre a solução desejada.

2.3.3 Incentivos para se manter a rede descentralizada em execução

Sendo a blockchain um sistema distribuído, descentralizado e ponto a ponto, é necessário que os nodos desta rede mantenham-se em execução. Mas para que seja interessante aos nodos participarem desta rede, é necessário algum tipo de incentivo.

O modelo mais utilizado atualmente, é o conceito de mineração, onde os nodos são recompensados por se dedicar a realizar a prova de trabalho, de forma a continuar a adição de dados na rede, consequentemente, mantendo o progresso da blockchain.

2.4 CONTRATOS INTELIGENTES

Contratos Inteligentes, do inglês "Smart Contract", foram introduzidos por (SZABO, 1997) e recebem este nome por realizarem algumas funções inerentes a um contrato. Segundo o dicionário online Aulete[1], um contrato é um:

> *Acordo entre duas ou mais partes (pessoas, empresas, instituições, governos etc.) com o propósito de atribuir, contrair, modificar, transferir, preservar ou revogar direitos e/ou obrigações*

[1] http://www.aulete.com.br/contrato

Unindo a definição clássica da palavra, com o conceito da palavra *"smart"*, temos um contrato que realiza o intermédio entre duas partes mesmo em um ambiente onde não há confiança entre os envolvidos. Uma vez em execução, sua programação será executada sempre que as condições estabelecidas forem acionadas.

Uma das formas de se formular um contrato inteligente, é através de protocolos de computadores, desenvolvidos para cumprir tarefas bem específicas, tendo expressividade igual a de programas "Turing Completo", mas não deve ser confundido com simples programas de computador. Adicionalmente, é comum que este protocolo esteja diretamente atrelado a valores monetários.

2.5 ETHEREUM

Em 2013, Vitalik Buterin propôs a ideia de uma maquina virtual descentralizada para a execução de contratos inteligentes, chamada de Ethereum (BUTERIN, 2013). Em sua publicação, Buterin descreve sua intenção com com o desenvolvimento desta blockchain como

> *juntar e aprimorar os conceitos de scripting, altcoins e meta-protocolos em cadeia, e permitir que desenvolvedores criem aplicações arbitrárias baseadas em consenso que tenham escalabilidade, padronização, completude de características, facilidade de desenvolvimento e interoperabilidade, ao oferecer estes diferentes paradigmas, todos ao mesmo tempo.*

A definição mais simples dada pelo próprio site da ethereum[2] é a de um "Computador Mundial". Este computador está disponível a todo o tempo, sem interrupções e com quanta memória for necessária. Isso se faz possível, devido à natureza descentralizada e distribuída entre os nodos interessados em participar. Esta plataforma de computador mundial, permite que qualquer pessoa possa realizar tarefas computacionais simples, com um baixo custo e sem custo de infraestrutura enquanto fornece algumas características:

- Autenticação de usuário
- Lógica de pagamento completamente customizável
- 100% resistente a DDOS

[2] https://ethereum.gitbooks.io/frontier-guide/content/ethereum.html

- Armazenamento de dados seguro (mas não privado)
- Interoperabilidade entre os componentes do ecosistema Ethereum
- Não necessita de servidores

2.6 NOTIFICAÇÕES

Em sua definição mais simples retirada do dicionário Aulete, uma notificação é "o ato de comunicar formalmente por escrito"[3]. A legislação brasileira prevê vários tipos de notificação. Dentre os tipos previstos, a Lei n° 13.105de 16 de março de 2015, Art. 726 a 729[4] descreve as notificações judiciais e extrajudiciais que diz:

> *Art. 726. Quem tiver interesse em manifestar formalmente sua vontade a outrem sobre assunto juridicamente relevante poderá notificar pessoas participantes da mesma relação jurídica para dar-lhes ciência de seu propósito.*
> *§ 1o Se a pretensão for a de dar conhecimento geral ao público, mediante edital, o juiz só a deferirá se a tiver por fundada e necessária ao resguardo de direito.*
> *§ 2o Aplica-se o disposto nesta Seção, no que couber, ao protesto judicial.*
> *Art. 727. Também poderá o interessado interpelar o requerido, no caso do art. 726, para que faça ou deixe de fazer o que o requerente entenda ser de seu direito.*
> *Art. 728. O requerido será previamente ouvido antes do deferimento da notificação ou do respectivo edital:*
> *I - se houver suspeita de que o requerente, por meio da notificação ou do edital, pretende alcançar fim ilícito;*
> *II - se tiver sido requerida a averbação da notificação em registro público.*
> *Art. 729. Deferida e realizada a notificação ou interpelação, os autos serão entregues ao requerente.*

No Brasil, notificações são documentos utilizadas para dar ciência de algo e garantir que a pessoa o tenha recebido. Existem alguns

[3] http://www.aulete.com.br/notificação
[4] http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2015/Lei/L13105.htm#art726

tipos de notificações com propósitos diferentes extraídos dos artigos 726 a 729. São eles: notificação judicial/extrajudicial, citação e intimação.

As formas previstas na legislação para realização de uma citação, são em sua grande maioria compartilhados pelas outras formas de notificação. Sendo assim, as definições das formas de notificação herdam as características da citação a menos que apresentado de outra forma.

2.6.1 Citação

Uma citação se assemelha muito à uma notificação. Prevista no artigo 238 do Código de Processo Civíl, Lei n° 13.105, *"Citação é o ato pelo qual são convocados o réu, o executado ou o interessado para integrar a relação processual"*. Dessa forma, ela tem duas principais funções que são: certificar o réu da existência de uma demanda ajuizada em seu desfavor e convocar o réu a comparecer em juízo.(GORERI, 2018)

Quando uma citação é executada com sucesso, a relação processual é completada e o réu está devidamente convocado. Pelo art. 5^{a} da constituição federal de 1988, isto garante também ao réu, seu direito de defesa e à contradição.

Uma sentença não pode ser válida, caso a citação não tenha sido realizada. O Art. 214 diz que *"para validade do processo, é indispensável a citação inicial do réu"*.

Essa exigência legal diz respeito a todos os processos (de conhecimento, de execução e cautelar), sejam quais forem os procedimentos (comum ou especiais). Até mesmo os procedimentos de jurisdição voluntária, quando envolverem interesses de terceiros, tornam obrigatória a citação (Art. 1.105). (JúNIOR, 2014).

Entretanto, a existência de uma citação não é suficiente para dizer que existe um processo, pois para que seja emitida uma citação, já precisa existir uma relação jurídico-processual entre o autor e o Estado-Juiz. Isto pode ser confirmado pelo fato de que o indeferimento de uma petição inicial garante automaticamente um resultado favorável ao réu.

O CPC prevê várias formas de citação: pelo correios, por oficial de justiça, por hora certa, por escrivão, por edital e por meios eletrônicos.

Pelos correios

Esta forma de citação é a regra geral, devendo ser utilizada caso não haja lei que obrigue o uso de outras formas. Conforme previsto no

Art. 248 do CPC, para que seja válida a correspondência deve conter os seguintes itens:

- Acompanhar cópia da petição inicial do juiz estar acompanhada da cópia da petição inicial e do despacho do juiz;
- Informar o prazo e endereço para resposta
- Ser enviado na modalidade correspondência AR

Vale frisar, que em caso de recusa de recebimento ou de assinatura do recibo a citação não é considerada válida, visto que o carteiro não tem fé pública. O autor pode então requerer por mandado cobrando ao réu as custas da diligência fracassada (ARAGãO, 2004).

Por oficial de justiça

Esta é a forma de citação adotada, em caso de frustração da entrega por correio. O Art. 250 diz que:

> *Art. 250. O mandado que o oficial de justiça tiver de cumprir conterá:*
> *I - os nomes do autor e do citando e seus respectivos domicílios ou residências;*
> *II - a finalidade da citação, com todas as especificações constantes da petição inicial, bem como a menção do prazo para contestar, sob pena de revelia, ou para embargar a execução;*
> *III - a aplicação de sanção para o caso de descumprimento da ordem, se houver;*
> *IV - se for o caso, a intimação do citando para comparecer, acompanhado de advogado ou de defensor público, à audiência de conciliação ou de mediação, com a menção do dia, da hora e do lugar do comparecimento;*
> *V - a cópia da petição inicial, do despacho ou da decisão que deferir tutela provisória;*
> *VI - a assinatura do escrivão ou do chefe de secretaria e a declaração de que o subscreve por ordem do juiz".*

Ao procurar o citando e onde o encontrar, cabe ao oficial de justiça:

- Ler o mandado
- Entregar-lhe o documento

- Certificar o recebimento ou recusa

Por hora certa

Na citação por mandado, prevê o artigo 252, do CPC que:

> *Quando, por 2 (duas) vezes, o oficial de justiça houver procurado o citando em seu domicílio ou residência sem o encontrar, deverá, havendo suspeita de ocultação, intimar qualquer pessoa da família ou, em sua falta, qualquer vizinho de que, no dia útil imediato, voltará a fim de efetuar a citação, na hora que designar.*

Por escrivão ou chefe de secretaria

Prevista no art. 246, inciso III, do Código de Processo Civil

> *Caso o citando compareça no Juízo em que está sendo demandado, o Escrivão ou Chefe de Secretaria deverá realizar a sua citação (Art. 152, inciso II, CPC), simplificando o procedimento citatório.*

Por edital

A citação por edital pode ser feita quando:

- Citando for desconhecido ou incerto
- Local do citando for incerto ou inacessível
- Casos expressos em lei

É considerado local incerto, quando foram realizadas várias tentativas frustradas de localizar o citando, mesmo dispondo de busca em cadastro de orgãos públicos (Art. 256, § 3°, CPC).

Os requisitos pra citação por edital estão previstas no Art. 257 do CPC:

- Afirmação do autor ou a certidão do oficial informando a presença das circunstâncias autorizadoras
- Publicação do edital na web, no site do tribunal e na plataforma de editais do Conselho Nacional de Justiça
- Determinação do juiz, prazo (entre 20 e 60 dias), a partir da data da publicação

- Advertência de que será nomeado curador especial em caso de revelia

A citação por edital poderá também, em determinação feita pelo juiz, ser realizada em jornal local de ampla circulação.

Por meio eletrônico

As citações por meios eletrônicos estão previstas na legislação brasileira no 9° da Lei $n°$ 11.419/2006 que criou e regulamentou o processo eletrônico que diz: "No processo eletrônico, todas as citações, intimações e notificações, inclusive da Fazenda Pública, serão feitas por meio eletrônico, na forma desta Lei".

Esta forma de citação deve ser o principal meio de citação das empresas públicas e privadas que deverão manter o cadastro nos sistemas de processos e autos eletrônicos. Microempresas, empresas de pequeno porte, a Uinão, os Estados, Municípios e entidades de administração deverão ser citadas por mandado ou carta precatória.

Citações por meio eletrônico passam a contabilizar o prazo para contestação a partir do dia útil seguinte à consulta ao teor da citação.

2.6.2 Notificação Extrajudicial

O tipo de notificação mais comum é a notificação extrajudicial. Com este modelo é possível por exemplo:

- Realizar cobranças
- Solicitar liberação de imóveis
- Prevenir responsabilidades
- Observância de prazos

Para que tenha validade, este tipo de notificação requer que a obrigatoriedade da comprovação de recebimento em mãos do notificado ou representante legal. Esta comprovação pode ser feita através de assinatura de algum comprovante ou algum outro meio legal. Os meios comumente utilizados para tal, são os cartórios de títulos e documentos e notificações enviadas pelo correio (cartas com rastreio (AR)). Outros meios de notificação aceitos são e-mail e whatsapp desde que se comprove o recebimento da mensagem.

A maior dificuldade de validar uma notificação é a comprovação de entrega do documento, pois o mesmo pode ser recusado ou difícil encontrar o notificado. Desta forma, o intermédio do cartório de títulos e documentos se fez viável, devido à fé pública[5] atribuída ao notário ou tabelião.

Segundo website jusbrasil[6] o processo de notificação através do cartório de títulos e documentos é feito da seguinte forma:

- A elaboração do documento com o conteúdo não tem um padrão e pode ser feito livremente desde que siga as seguintes observações:
  - Conter o título: "Notificação Extrajudicial"
  - Constar o nome completo e o endereço da pessoa notificada
  - Informar o objetivo da notificação com exigências e providências a serem tomadas, prazos e medidas a serem tomadas em caso de não cumprimento.
- O cartório fará três tentativas de entrega do documento em horários e dias distintos. Caso o notificado se recuse a receber o documento, prevalece a fé pública do oficial indicando que a diligência foi realizada mas o destinatário recusou.
- O cartório emite uma certidão referente à notificação, que comprova o resultado da diligência (recebimento, recusa, mudança de endereço, etc).

Os serviços notariais e de registro tem um custo[7]. O custo médio encontrado em alguns cartórios distribuídos pelo País foi de R$140,00 para diligências e R$70,00 para notificações por correios (AR).

2.6.3 Notificações Judiciais

Com os mesmos objetivos que a notificação extrajudicial, este tipo de notificação é executada pelo poder judiciário. O artigo 726 §1 da Lei número 13.105 do CPC informa da possibilidade de levar a notificação a conhecimento público, desde que a necessidade seja bem fundamentada e necessária para resguardar o direito.

[5] http://www.planalto.gov.br/ccivil_03/leis/L8935.htm

[6] https://lucenatorres.jusbrasil.com.br/artigos/518887517/o-que-e-e-para-que-serve-uma-notificacao-extrajudicial

[7] http://servicos.cdtsp.com.br/down/tabela_completaz_2018.pdf

Notificações judiciais são entregues em mãos por oficiais de justiça. O artigo 721 do CPC garante o direito da pessoa notificada ser ouvida antes do deferimento da notificação pelo juiz em caso de detecção dos casos descritos nos incisos I e II:

- Se houver suspeita de que o requerente, por meio da notificação ou do edital, pretende alcançar fim ilícito
- Se tiver sido requerida a averbação da notificação em registro público

2.6.4 Intimação

As intimações muito se assemelham à citação. Conforme o Art. 234 do CPC, *"intimação é o ato pelo qual se dá ciência a alguém dos atos e termos do processo, para que faça ou deixe de fazer alguma coisa"*. Em suma, trata-se de comunicar não só às partes e patronos, mas todo aquele que participa do processo. Um dos casos em que se faz necessário uma intimação, é ao fazer necessário o depoimento de uma testemunha(MOTA, 2013).

A atual CPC permite que as notificações sejam utilizadas para dar ciência de algum ato ocorrido no processo ou para determinar a prática de um ato, abolindo assim, a distinção entre intimação e notificação.

Como regra geral, a intimação deve ser feita por publicação em ato no orgão oficial. Quando não for possível, a intimação deve ser feita por meio dos correios. Caso ambos os meios de intimação falhem, o Art. 239 do CPC indica que a intimação seja feita por oficial de justiça.

Uma outra forma de intimação, é através dos escrivãos e chefes de secretaria como explicado por Marinoni e Arenhart (MARINONI, 2010).

> *Outra forma de intimação diretamente feita, mas que dispensa a participação do oficial de justiça, é aquela que se procede diretamente às partes, a seus representantes legais ou advogados, em cartório ou na própria audiência (como acontece na previsão do art. 242, § 1º, do CPC), mediante termo nos autos. Comparecendo esses sujeitos em cartório, para qualquer ato que seja, pode-se proceder diretamente à intimação deles em cartório ou na audiência em que o ato esteja sendo realizado, sem que se tenha de expedir comunicação formal para a sua ciência (art. 238 do CPC).*

3 NOTIFICAÇÕES DIGITAIS

Este capítulo tem como objetivo descrever a solução proposta neste trabalho. A seção 3.1 descreve uma visão geral do projeto, a seção 3.2 detalha a arquitetura do projeto em módulos e componentes.

3.1 DESCRIÇÃO

Uma notificação tem como principal objetivo, informar uma pessoa algo que venha a ser de seu interesse. A maior dificuldade de se realizar uma notificação com sucesso é prover provas de que o notificado recebeu a notificação em mãos. Este é o principal motivo pelo qual o cidadão opta por utilizar os serviços do cartório de títulos e as cartas com aviso de recebimento. Este trabalho foi desenvolvido com o intuito de ampliar a viabilidade e acrescentar novos meios válidos de se efetuar uma notificação.

Para alcançar este objetivo, foi elaborado um sistema digital intermediador, capaz de enviar uma notificação por vários meios eletrônicos simultaneamente, ampliando as chances de uma notificação bem sucedida. Este sistema intermediador recebe, processa e encaminha as notificações.

3.2 ARQUITETURA DO PROJETO

Conceitualmente o sistema é dividido em três módulos, compostos por componentes menores. A imagem 4 ilustra uma visão geral da arquitetura do projeto. A subseção 3.2.1 descreve as interações entre o usuário e o sistema, a subseção 3.2.2 apresenta os sistemas auxiliares e a subseção 3.2.3 demonstra um pouco dos sistemas notificadores implementados e como estender os tipos de notificações.

3.2.1 Interface com Usuário

Este módulo tem como principal objetivo permitir que o interessado em efetuar uma notificação extrajudicial, possa emitir uma requisição de notificação que será analisada, processada e encaminhada ao notificado.

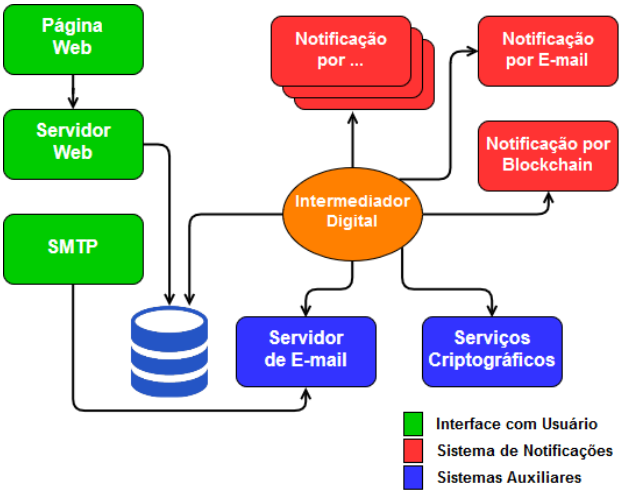

Figura 4 – Diagrama de relacionamento entre os componentes.
Fonte: Elaboração própria

Todo sistema computacional com foco na utilização do usuário, deve ter uma boa usabilidade e abranger o maior público possível. Para alcançar este objetivo, o sistema permite que as requisições de notificação sejam enviados por dois meios: por email - respeitando o formato pré estabelecido dos dados esperados - e por um banco de dados onde são armazenadas as notificações ainda não processadas.
O usuário pode então, acessar a página web pela internet e emitir sua requisição de notificação pelos campos do formulário. O backend da página web está preparado para trabalhar com outros formatos front-end, como por exemplo, aplicativos para dispositivos móveis.

A imagem 5 apresenta o diagrama de sequência da interface web onde está descrito todas as operações em que há interação com o usuário.

Outra forma implementada de interação entre o usuário e o intermediador digital é através do envio emails para o domínio do sistema de notificação. Além dos dados já fornecidos no cabeça-lho do email (destinatário e remetente) deve ser enviado o CPF do notificado e em anexo, o documento referente ao texto da notificação.

O diagrama apresentado na figura 6 descreve as interações que

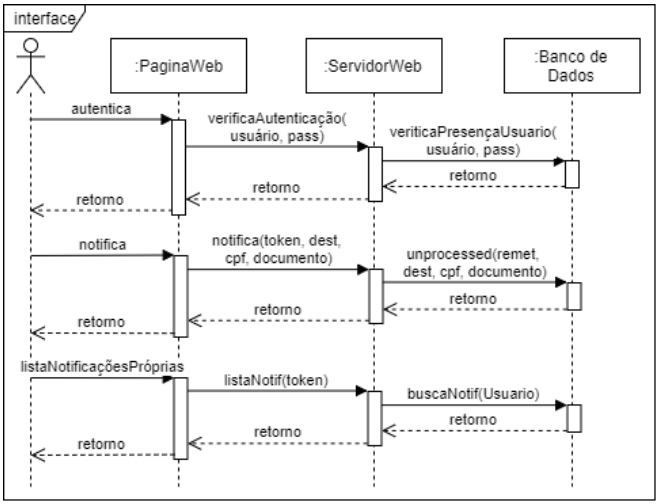

Figura 5 – Diagrama de sequência da interface web.
Fonte: Elaboração própria

são possíveis entre o usuário e o servidor de emails do sistema de notificações digitais.

3.2.2 Sistemas Auxiliares

Para alcançar todas as funcionalidades e características do sistema, alguns serviços foram necessários para garantir a persistência, acessibilidade e segurança dos dados.

A persistência e acessibilidade fazem uso de um banco de dados tanto para as notificações processadas quanto para as não processadas. Um servidor de emails exerce o papel de gerenciar as notificações enviadas por email, servindo também como base de dados para as notificações requisitadas através deste meio de comunicação.

Para garantir a autenticidade e evitar que notificações falsas sejam emitidas em nome do sistema, é utilizado o componente de serviços criptográficos. Este componente necessita de um certificado assinado por uma ICP (ICP-Brasil) de confiança afim de assinar as notificações enviadas permitindo que o notificado verifique a autenticidade da mensagem que acabou de receber. Dentre as operações realizadas por este componente estão a gerencia de certificados e chaves, assinatura e

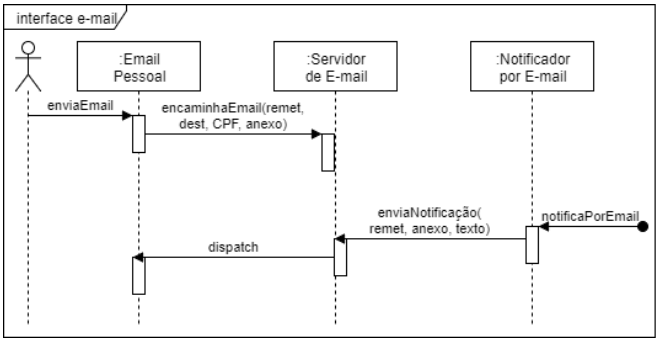

Figura 6 – Diagrama de sequência das interações por email.
Fonte: Elaboração própria

verificação de assinatura dos documentos.

A figura 7 descreve a sequência em que os módulos auxiliares são chamados.

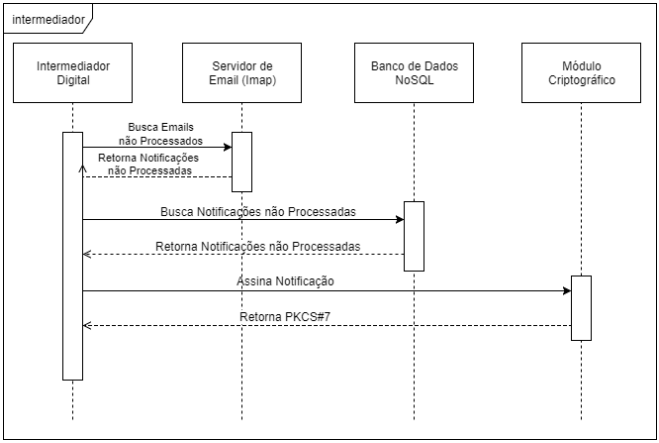

Figura 7 – Diagrama de sequência do intermediador.
Fonte: Elaboração própria

Incialmente, o sistema intermediador busca as notificações não processadas no servidor de emails e no banco de dados. Em seguida, todas as notificações não processadas são assinadas e empacotadas em uma estrutura PKCS#7 que contém o certificado do signatário, o documento referente à notificação e a assinatura do documento.

3.2.3 Sistema de Notificações

Para ampliar as chances de uma notificação bem sucedida, várias formas de envio de mensagens digitais podem ser adicionados ao sistema. Isso é possível padrão adotado na implementação do intermediador, que se assemelha ao Controle do modelo MVC, não realizando nenhuma operação, mas administrando o fluxo das informações. Desta forma, adicionar novos meios de envio de notificação torna-se uma tarefa simples. A imagem 8 descreve a sequência com que os principais módulos são executados.

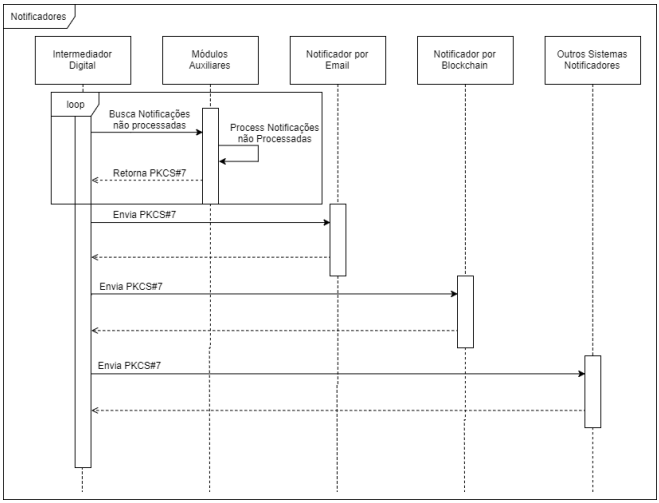

Figura 8 – Diagrama de sequência das chamadas dos notificadores.
Fonte: Elaboração própria

As informações manipuladas pelo intermediador são: email do notificante, email e cpf do notificante e o documento anexado a ser encaminhado na notificação.

Este sistema foi desenvolvido para permitir que a quantidade e tipo de notificações enviadas simultaneamente ao notificado possa ser configurado e ampliado. Entretanto, a fim de demonstração, foram implementados dois tipos de envio de notificação: por email e publicação em blockchain pública.

A notificação por email é composta pela mensagem na integra, com todos os dados da notificação enviada pelo notificante e do documento assinado pelo intermediador. A mensagem deve conter informa-

ções relevantes do notificante como: nome, endereço, email, etc, o texto que trata do assunto da notificação e informações do notificado.

A notificação publicada em blockchain pública tem o objetivo de servir como um jornal ou periódico de circulação livre. Diferentemente da notificação por email, as informações apresentadas são simplificadas a fim de evitar a divulgação de dados sensíveis. A presença destes dados em uma blockchain pública permite por exemplo, que uma entidade interessada tenha uma página web de notícias ou busca que informe as notificações emitidas.

Dentre os dados presentes nesta blockchain estão o email do notificante, cpf do notificado, o hash do documento de notificação e o carimbo de tempo.

Para tipos de notificação que podem ser efetuados por meios digitais, pode-se implementar um notificador que contenha todos os dados necessários para uma notificação desde que respeitada a Lei Geral de Proteção de Dados Pessoais[1]

[1] http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2018/Lei/L13709.htm

4 DESENVOLVIMENTO

Neste capítulo será apresentado todo o desenvolvimento do projeto proposto no capítulo 3.

4.1 CONTRATOS INTELIGENTES

Inicialmente o projeto teria como foco, notificações compostas quase completamente por artefatos presentes ou armazenados em blockchain. Sendo assim, o primeiro passo tomado foi o desenvolvimento do contrato inteligente que executaria as funcionalidades propostas no capítulo 3. As seções a seguir descrevem o processo de desenvolvimento do contrato inteligente inclusive a versão final e as motivações que levaram a tal.

4.1.1 Descrição do Contrato

Durante a concepção dos dados relevantes para uma notificação, obteve-se o seguinte resultado:

- Endereço que permita alcançar o notificado
- Endereço do notificante que permita resposta do notificado
- Estampa do tempo que permita averiguar a data de publicação da notificação
- Dados que remetam aos dados contidos na notificação
- Algum dado único que permita ao notificado verificar suas notificações

Com base nos requisitos levantados, foi elaborado o seguinte código para armazenamento dos dados na blockchain Ethereum:

```
1 struct Notification {
2     string notif_hash;
3     string sender_mail;
4     string dest_mail;
5     uint32 timestamp;
6 }
7
8 mapping (int40 => Notification[]) notifications;
```

No momento da elaboração do contrato, ficou claro que armazenar o documento completo na blockchain teria um custo de transação muito além do que se esperava para esta aplicação, optando-se por armazenar uma string representando o hash do documento entregue ao notificado no membro do contrato *notif_hash*. A estrutura dos dados pode ser vista na imagem 9 e sua descrição é realizada na listagem a seguir.

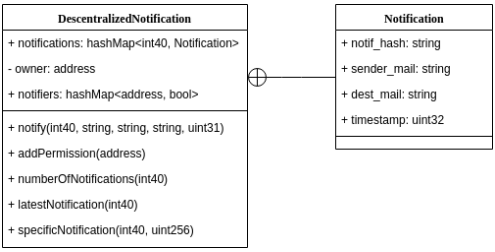

Figura 9 – Diagrama UML do contrato inteligente *DescentralizedNotification*.
Fonte: Elaboração própria

- *sender_mail*: endereço de email do notificante
- *dest_mail*: endereço de email do notificado
- *timestamp*: estampa de tempo em que a notificação foi publicada
- *notif_hash*: código hash do documento da notificação
- *notifications*: hash table com identificador único ao notificado

Embora pude-se se utilizar a própria data da Ethereum como forma de estampa do tempo, este poderia não refletir o tempo em que a notificação foi primeiramente publicada, devido ao tempo até que a transação seja minerada pela rede. Desta forma, optou-se por enviar também a estampa do tempo em que a notificação foi processada e publicada pela autoridade e armazenando-a na variável *timestamp*. Por último, o identificador único do notificado é enviado, mas serve apenas como índice da "hash table"*notifications*.

Além do armazenamento de dados, o contrato dispõe de algumas funcionalidades para receber e listar notificações. Adicionalmente, foram implementadas algumas funcionalidades administrativas, como por exemplo, adicionar novas autoridades notificadoras.

Abaixo, o código simplificado apresentando principalmente os "contratos" estabelecidos pelas funções:

```
function notify(int40 cpf, string notif_hash, string
    sender_mail, string dest_mail, uint32 timestamp)
public
onlyNotifier()
{
    Notification memory not = Notification(notif_hash,
        sender_mail, dest_mail, timestamp);
    notifications[cpf].push(not);
    emit NewNotification(sender_mail, dest_mail,
        notif_hash, timestamp);
}

function addPermission(address notifier)
public
onlyOwner()
{
    notifiers[notifier] = true;
}

function numberOfNotifications(int40 cpf) public view
    returns(uint256) {
    Notification[] storage not = notifications[cpf];
    return not.length;
}

function latestNotification(int40 cpf) public view
    returns(string, string, string){
    Notification[] storage not = notifications[cpf];
    uint256 index = not.length-1;
    return (not[index].sender_mail, not[index].
        dest_mail, not[index].notif_hash);
}

function specificNotification(int40 cpf, uint256 index
    ) public view returns(string, string, string){
    Notification[] storage not = notifications[cpf];
    return (not[index].notif_hash, not[index].
        sender_mail, not[index].dest_mail);
}
```

O código desenvolvido para contratos inteligentes tem uma característica de ser códigos curtos, que realizam poucas operações e que tendem a ter uma boa eficiência. Abaixo segue uma breve listagem dos métodos e suas funcionalidades:

- *notify*: recebe uma notificação com os dados descritos acima
- *addPermission*: adiciona permissão para uma nova autoridade notificadora

- *numberOfNotifications*: lista a quantidade de notificações um CPF contém

- *latestNotification*: retorna a última notificação emitida para um CPF

- *specificNotification*: retorna uma notificação a partir de um CPF e um índice

4.1.2 Otimizações do Contrato Inteligente

Algumas otimizações de código próprias da Ethereum foram aplicadas a fim de reduzir o custo da implantação do contrato e execução das transações.

As funções *numberOfNotifications*, *latestNotification* e *specificNotification* foram configuradas com o modificador *view* a fim de eliminar o custo de transação ao chamar estas funções de fora da rede Ethereum.

Para reduzir o custo de algumas execuções, os modificadores memory e storage foram utilizados para indicar ao compilador, que estas variáveis deveriam ser armazenadas na Ethereum, ou se deveriam ser colocadas na pilha para serem descartadas ao final da execução.

Uma última otimização realizada foi na construção do código ao reduzir o tamanho do código fonte. Estes detalhes podem ser vistos no modificador *onlyOwner* e *onlyNotifier*, ao reduzir em tempo de compilação o tamanho do código, consequentemente, os custos de implantar o código na rede Ethereum.

4.2 REDE PRIVADA ETHEREUM

Existem diversas formas de desenvolver e testar contratos inteligentes para a rede Ethereum, cada um com suas características positivas e negativas. Durante o desenvolvimento do projeto, várias dessas formas foram postas em prática e levantado tais características a fim de encontrar a que se melhor adéqua à solução proposta. Para este projeto, decidiu-se instanciar uma rede privada, com restrições de acesso permitindo um maior controle da quantidade de nodos, tempo de mineração e acesso à rede.

Para tal, foi utilizado o *Geth*, uma interface de linha de comando para a execução de um nodo Ethereum completo e implementado na

linguagem de programação Go[1].

Este nodo completo foi instalado em um servidor ubuntu interno disponibilizado pelo LabSEC. Como toda inicialização de uma rede Ethereum, foi utilizado algumas configurações para o nodo *Genesis*, que é o primeiro bloco da rede. As configurações realizadas estão no exemplo de código **??**.

Listing 4.1 – Configurações do bloco Genesis.

```
{
    "nonce": "0x0000000000000042",
    "timestamp": "0x0",
    "parentHash": "0x000000...00000000000",
    "extraData": "0x00",
    "gasLimit": "0x8000000",
    "difficulty": "0x4000",
    "mixhash": "0x000000...00000000000",
    "coinbase": "0
        x3333333333333333333333333333333333333333"
}
```

Abaixo uma breve descrição dos campos do arquivo de configuração[2]:

- *nonce*: valor que resulta na prova de trabalho realizada
- *timestamp*: tempo da criação do bloco
- *parentHash*: o hash Keccak 256-bits do cabeçalho do bloco pai
- *extraData*: parametro opcional. Pode conter dados relevantes a este bloco
- *gasLimit*: limite atual de gas despendido por bloco
- *difficulty*: dificuldade da prova de trabalho
- *mixHash*: hash 256-bit que em conjunto com o nonce, formam o resultado da prova de trabalho deste bloco
- *coinbase*: endereço da conta que receberá a recompensa da mineração. Neste bloco não há recompensa, logo, este campo pode ser qualquer valor

[1] https://github.com/ethereum/go-ethereum/wiki/geth
[2] https://www.asynclabs.co/blog/params-in-ethereum-genesis-block-explained/

Esta rede foi montada e configurada para experimentos internos utilizando alguns dos parametros fornecidos pela interface de linha de comando Geth. A listagem a seguir, apresenta o código para inicialização da rede a partir do bloco genesis.

Listing 4.2 – Comando para criar a rede privada Ethereum

```
1 geth --datadir "~/ethereum/nodo" init "./genesis.json"
```

Com a rede já configurada e criada, ainda não existe nenhum nodo em execução. A listagem de código a seguir apresenta os comandos para iniciar um nodo completo na rede recém criada configurado para as necessidades do projeto.

Listing 4.3 – Inicialização de um nodo completo

```
1 geth -port 30303 --ws --wsport=8445 --wsorigins "*" --
    wsaddr "0.0.0.0" --wsapi personal,eth,web3 --
    datadir "~/ethereum/nodo1" --networkid 5555 --
    rpcapi personal,eth,net,web3 --rpc --rpcport 8545
    --rpccorsdomain "*" --rpcaddr "0.0.0.0"
```

Dentre as configurações feitas durante a inicialização do nodo, vale ressaltar os seguintes parametros:

- *port*: porta local de comunicação deste nodo na rede ethereum
- *ws*: habilita utilização de *WebSockets*, úteis para trabalhar com os eventos da ethereum
- *wsapi*: quais apis estarão disponíveis através do *WebSocket*
- *networkid*: id da rede ethereum. Para a rede principal da Ethereum, utilizar 1
- *rpc*: habilita a utilização da interface RPC
- *rpcapi*: quais apis estarão disponíveis através da interface RPC

Foi configurado um nodo completo da rede Ethereum encarregado de minerar e atender às demandas dos nodos light da rede. Testes foram executados com outros nodos completos, mas o resultado apresentado não agregava ao projeto, logo foram desabilitados.

Todas as consultas e transações enviadas à rede foram encaminhadas ao nodo completo configurado acima, concentrando o trabalho nele. Embora para este projeto tenha sido utilizado esta topologia, não

recomenda-se concentrar tantas funções em um único nodo, principalmente por se tratar de um nodo completo. O ideal é que haja alguns nodos leves, que realizem funções bem definidas, e apenas estes nodos leves comuniquem com o nodo completo.

4.3 SERVIDOR DE EMAIL

Para uma completa prototipação do sistema aqui apresentado, foi necessário a configuração de um servidor de emails. Como o foco do trabalho não era entender como funciona um servidor de emails, sendo este, apenas uma ferramenta para alcançar os objetivos definidos, optou-se por utilizar um sistema já consolidado e de fácil configuração, o iRedMail.

A primeira configuração realizada, foi a criação do domínio *mailchain.org*, que sustenta a base das contas de mail configuradas neste servidor. Em seguida, foi configurado o usuário administrador *postmaster* por ser por padrão, a primeira conta de um servidor de email. Em seguida, foi criado uma conta de emails com nome de usuário *notifications*. Esta conta foi configurada para funcionar como um "*sinkhole*", ou seja, uma conta que recebe todos os emails enviados para o domínio *mailchain.org*. Desta forma, o servidor intermediador precisa buscar apenas os emails presentes na caixa de entrada e não lidos desta conta de email.

4.4 SERVIDORES DA APLICAÇÃO

Analisando o projeto definido no capítulo 3, identificou-se rapidamente que a melhor forma de desenvolver a aplicação, era elaborar um intermediador entre entre a entrada de dados (inicialmente por email) e a saída de dados (por email e blockchain, simultaneamente). Baseando-se nestas análises e nos componentes já implantados: Blockchain e servidor de email, decidiu-se implementar um servidor intermediador, expansível em todas as suas funcionalidades: entrada, processamento e saída.

4.4.1 Servidor Intermediador

4.4.1.1 Implementação

Duas opções de linguagem de programação foram levantadas como possibilidade para a implementação do servidor principal intermediador: Java e JavaScript. Embora Java seja uma linguagem consolidada no meio corporativo pela sua robustez e portabilidade, a api fornecida para comunicação com nodos Ethereum não são amigáveis, tornando o desenvolvimento lento. Ao contrário do Java, a api fornecida pelo projeto web3 para Node.js assemelha-se muito aos comandos executados diretamente na interface de linha de comandos Geth, utilizada para inicializar o nodo Ethereum da nossa rede privada. Adicionalmente, outras tecnologias agregadas ao decorrer do desenvolvimento do projeto teriam um alto grau de compatibilidade com o Node.js. Desta forma, ao longo do desenvolvimento, decidiu-se seguir utilizando JavaScript como linguagem de programação base do servidor intermediador.

Desconsiderando a configuração de rede do servidor por ser trivial e irrelevante à escrita do documento, o servidor é composto de uma função principal que procura novas notificações e executa uma função que é passada por parâmetro para ser executada por cada uma das notificações. O código desta função é apresentado abaixo:

Listing 4.4 – Função de busca e processamento de notificações

```
let imapPrivate = new Imap(config.private_mail);
mail.fetchAttachments(imapPrivate, sendNotification,
    () => console.log('error fetching notifications')
);
```

A função *sendNotification* realiza todo o trabalho que necessita ser realizado para preparar a notificação e então emiti-la.

```
function sendNotification(
  sender_mail, cpf, dest, attachment
) {
  let transporter = nodemailer.createTransport(config.
      private_mail_sender);
  // Configura envio de emails
  crypto.sign(attachment).then((cipherAttachment) =>
  {
    User.findOne({email: sender_mail}, function (err,
        user) {
      let userExist = user !== null;
      if(!userExist) {
```

```
11          // Cria usuario caso ainda nao exista
12        }
13        crypto.hash(attachment).then(
14          (hash) =>{
15            // Envia notificacao para a rede ethereum
16            blockchain.notify(hash, sender_mail, cpf,
                  dest)
17            let data = {
18              // Dados da notificacao
19            };
20          let notif = new Notification(data);
21          notif.save()
22          // Salta as notificacoes em base de dados
23          mail_sender.sendMail(transporter, dest, [
                attachments]);
24          mail_sender.sendMail(transporter, sender_mail,
                  confirmation);
25          // Envia email de confirmacao ao notificante e
26          // notificacao com anexos ao notificado.
27          // Adicionar aqui outras formas de notificacao
28        }
29      )
30    }
31  }
```

Listing 4.5 – Função de processamento das notificações

4.4.1.2 Especificação dos Contratos de Entrada e Saída

A implementação final do servidor intermediador busca as notificações em um servidor de emails e em uma base de dados NoSQL MongoDB. Inicialmente, o intermediador foi configurado para buscar os emails não lidos na caixa de entrada da conta de email notifications-mailchain.org para processa-los. Os campos do email para requisitar uma notificação seguem o padrão:

- Destinatário: “destinatariodominio.com”mailchain.org
- Assunto: CPF do notificado
- Campo de texto: campo ignorado, pode conter qualquer informação
- Anexo: arquivo .pdf redigido seguindo os padrões válidos de notificação

Vale ressaltar o formato do campo destinatário pois este contém uma sintaxe diferente. Por padrão, os endereços de email podem conter strings complexas e com caracteres especiais, desde que estes dados estejam entre aspas duplas. Utilizou-se então deste mecanismo para enviar o email para o notificado indiretamente onde: "destinatário@dominio.com" é o email do destinatário entre aspas duplas e @mailchain.org, o domínio de email do sistema notificador.

Para que a notificação obtenha sucesso, é essencial que o documento seja redigido e salvo em um arquivo no formato .pdf.

Com a evolução do projeto e do desenvolvimento dos servidores, percebeu-se que a adição da opção de leitura de notificações armazenadas em uma base de dados agregaria muito para o projeto como um todo, ao permitir que vários tipos de módulos de entrada pudessem agregar a solução.

Tal melhoria foi realizada implementado uma base de dados NoSQL para as notificações não processadas. A solução foi modelada com duas relações: Notifications e Unprocessed. A descrição simplificada das duas relações pode ser encontrada nas duas listagens que seguem.

```
{
    "author": { type: ObjectId, ref: 'User' },
    "destMail": String,
    "destCpf": Number,
    "attachment": String,
    "hash": String,
    "transaction": String
}
```

Listing 4.6 – Esquema da relação Notification.

```

{
    "unprocessed": { type: ObjectId, ref: '
        Notification' }
}
```

Listing 4.7 – Esquema da relaçao Unprocessed.

O servidor intermediador busca as entradas na relação Unprocessed que tem uma ligação para uma entrada na relação Notifications que contem os dados completos da notificação. De posse da notificação, o servidor intermediador processa os dados obtidos da relação, e caso seja executado com sucesso, remove a entrada presente na relação Unprocessed.

4.5 SERVIDOR WEB

Embora a entrada de notificações por email funcione bem e seja muito acessível, a popularidade deste meio de comunicação tem decrescido principalmente entre a população mais jovem. A entrada por banco de dados também funciona bem, mas sua utilização não foi feita para o usuário comum. Adicionalmente, garantir ao usuário permissão de escrita no banco de dados é uma prática extremamente desaconselhável.

Com foco na atualização tecnológica e garantias de segurança do sistema, foi desenvolvido um servidor web com a linguagem de programação Javascript e framework Node.js, configurado interfaces REST, autenticação através do JWT e escrita no banco de dados MongoDB disponibilizado pelo site mlab[3].

O servidor conta com várias interfaces REST, onde algumas necessitam de autenticação, e outras não. Em seguida as interfaces que não necessitam de autenticação ou que são opcionais:

- GET 'notifications/': retorna todas as notificações armazenadas no banco de dados
- GET 'notifications/:notification': retorna uma notificação específica representada pelos dados presentes em :notification

Abaixo as interfaces REST onde a autenticação é obrigatória:

- GET 'notifications/feed': retorna todas as notificações do usuário logado
- POST 'notifications/': envia a requisição de notificação

A figura 10 apresenta o diagrama com descrição formal das interfaces REST do sistema.

De forma resumida, o servidor web funciona como um caminho intermediário, facilitador e uma camada de segurança entre o usuário e o banco de dados.

Esta camada adicionada traz grandes benefícios ao permitir que aplicativos móveis, e qualquer outro tipo de aplicativo que se comunique através deste modelo, possam ser facilmente desenvolvidos para enviarem notificações ao servidor web través das interfaces REST implementadas aqui.

[3] https://mlab.com/

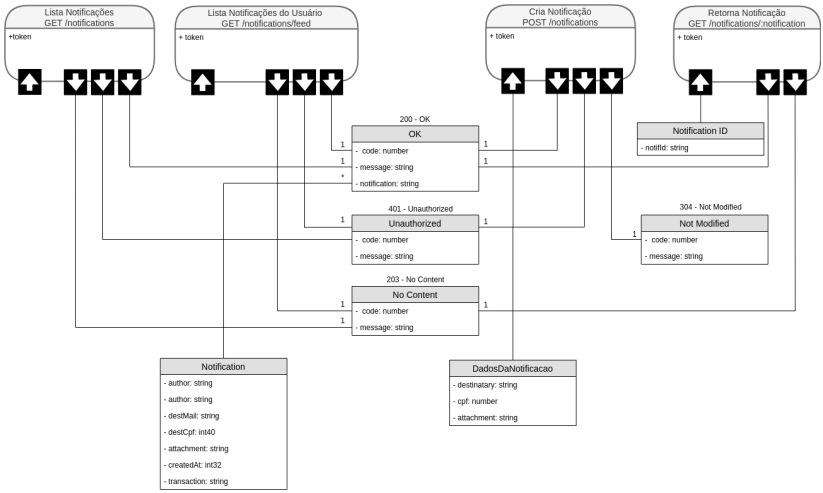

Figura 10 – Diagrama da descrição formal das interfaces REST.
Fonte: Elaboração própria

4.6 PÁGINA WEB

A implementação do servidor web disponibilizou para as aplicações e programadores as funcionalidades apresentadas na seção 4.5 mas é inviável a utilização dos recursos por usuários comuns. Para melhorar a apresentação do projeto desenvolvido e demonstrar as funcionalidades do servidor web, foi implementado uma página web que permite a realização dos serviços básicos de notificação. Adicionalmente, um quadro de listagem das transações blockchain relacionadas ao projeto foi adicionado à página web a fim de demonstrar seu funcionamento em conjunto com as notificações.

A implementação da página foi feita utilizando: a linguagem de programação Javascript, framework Node.js e React. Além das tecnologias acima mencionadas, foram utilizados algumas abordagens de desenvolvimento de software front-end. A principal delas foi o desenvolvimento orientado a componentes como é sugerido ao utilizar React. Também foi utilizado os padrões de página *stateless* e padrão de *routes*, mas devido ao pouco conhecimento nestes dois últimos padrões, a implementação ficou limitada.

As funcionalidades básicas implementadas na página web foram:

- Autenticação de usuário

- Emissão de notificação
- Visualização das notificações enviadas pelo usuário autenticado
- Visualização das notificações enviadas por todos os usuários
- Visualização das transações ethereum referentes às notificações

Ao abrir a página web do sistema de notificações, o sistema apresentará ao usuário uma página semelhante à apresentada na figura 11, onde é possível acompanhar as notificações emitidas e as transações ethereum realizadas referentes a este contrato. Tais funcionalidades estão disponíveis para qualquer usuário que acessar a página web.

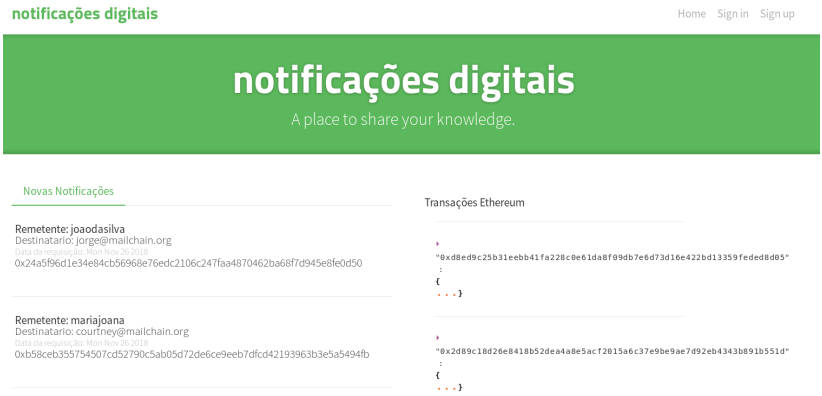

Figura 11 – Página inicial.
Fonte: Elaboração própria

A figura 12 apresenta a tela de autenticação onde o usuário tem a possibilidade de se autenticar ou se registrar no sistema, caso ainda não possua um usuário cadastrado. A autenticação do usuário no sistema é realizada utilizando o padrão JWT[4] e sua biblioteca correspondente para Node.js.

Autenticado, o usuário pode então requisitar uma notificação através do formulário como o apresentado na figura 13. Ao ser aceita, a notificação será processada, e ao final da etapa de processamento, o status da notificação é alterado de pendente e informações da transação ethereum referente à esta notificação atualizará este campo.

Por fim, a figura 14 apresenta a tela que pode ser vista por um usuário logado no sistema.

[4]https://jwt.io/

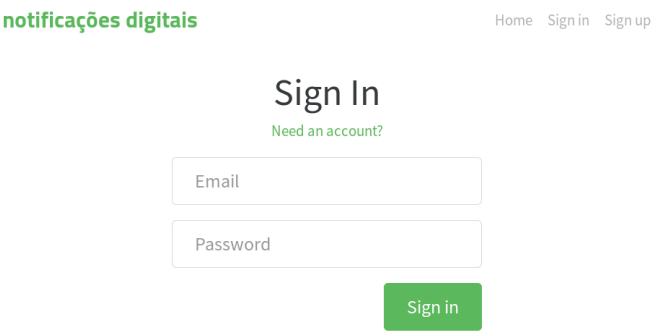

Figura 12 – Página de autenticação.
Fonte: Elaboração própria

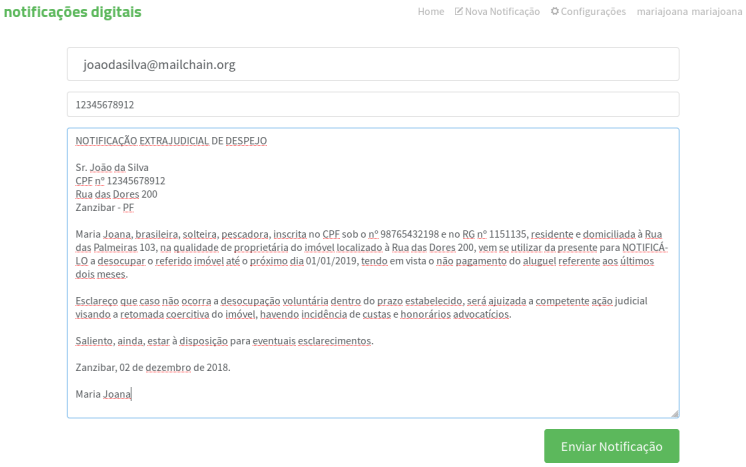

Figura 13 – Página para requisição de notificação.
Fonte: Elaboração própria

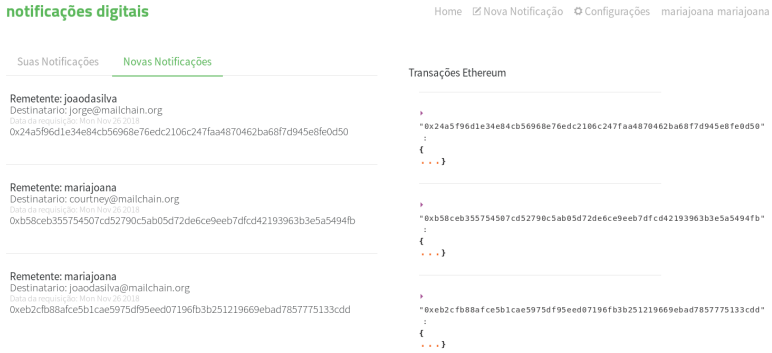

Figura 14 – Página principal.
Fonte: Elaboração própria

5 DISCUSSÃO

Neste capítulo será apresentada a discussão do projeto

5.1 VANTAGENS DO SISTEMA

A principal motivação para a a elaboração deste trabalho foi a de melhorar a eficiência das notificações disponíveis na legislação brasileira. Dentre as melhorias provenientes do sistema proposto em relação ao modelo atual, vale ressaltar:

- Redução do custo
- Facilidade de entrega
- Agilidade na entrega
- Comprovação de entrega

A implementação do sistema proposto teve como objetivo, auxiliar na solução dos problemas mencionados acima através das seguintes abordagens:

5.1.1 Redução do Custo

Todos os tipos de notificação tem seu custo, seja para o notificante ou para o Estado. Quando a notificação pode ser emitida por imprensa oficial ou por correios, existe o custo da publicação. Caso seja realizada por oficial de justiça, existe o custo de deslocar o oficial até o local. Se for utilizado o cartório de títulos e documentos, existe o custo cobrado pelo serviço.

A base do sistema proposto faz uso de 2 servidores, uma base de dados NoSQL e um servidor de email. Os módulos podem ser distribuídos nos servidores da seguinte forma: Servidor web em um servidor para processar as requisições, o intermediador e o servidor de emails juntos em um servidor de alto desempenho e a base de dados NoSQL em um servidor dedicado. Considerando a solução completa como apresentado na figura 4, a página web pode acompanhar o servidor web e seria necessário um outro servidor de alto desempenho para executar o nodo ethereum.

Foi levantado uma estimativa do custo dos servidores necessários para atender às demandas do projeto com base nos dados coletados no site Amazon AWS[1]. Devido à dificuldade em mensurar a quantidade de notificações são emitidas por segundo no Brasil, optou-se por superdimensionar a capacidade dos servidores utilizados.
O levantamento utiliza valores por hora utilizada do servidor, sendo que os serviços cotados escalam de acordo com o consumo de recursos, podendo reduzir o custo dos mesmos.
Para a aproximação dos custos com base de dados[2], foi utilizado o tamanho dos dados da base de dados utilizado nos testes do projeto. O cálculo também leva em conta o acesso aos dados, onde foi estipulado 200 leituras e 20 gravações por segundo, considerando que o sistema completo pode dispor de sistemas de balanço e distribuição de carga e otimização de acesso aos dados.

Serviço	**Qtd**	**Custo Max./Hora**	**Total Aprox.**
Amazon EC2 c5.18xlarge	2	\$4,716	$\approx$ \$6791,04
Amazon EC2 c4.8xlarge	2	\$2,47	$\approx$ \$3556,80
Amazon DynamoDB	1	-	$\approx$ \$1142,08
Total Mensal	-	-	$\approx$ \$11495,92

Tabela 1 – Custos com servidores e banco de dados

O notificador baseado em blockchain ethereum é um dos módulos que tem custo por notificação baseado no custo de uma transação ethereum. A tabela 2 apresenta os custos referentes a este notificador.

Serviço	**Custo em Ether**	**Custo em Dólar**
Implantação do Contrato	1910554 Gas	$\approx$ \$1.12
Envio de Notificação	$\approx$ 327552 Gas	$\approx$ \$0.19

Tabela 2 – Custos da blockchain ethereum

Como as notificações feitas utilizando a blockchain tem um custo direto por cada notificação, este custo pode ser repassado ao notificante visto que o gasto para se enviar uma notificação é baixo se efetuado poucas vezes, mas pode se tornar alto se considerarmos a quantidade de notificações que são emitidas diariamente no país.

A fim de comparação, no último edital[3] para concurso para oficial de justiça de Santa Catarina a remuneração era de R\$6.156,63 sem os

[1] https://aws.amazon.com/pt/ec2/pricing/on-demand/
[2] http://calculator.s3.amazonaws.com/index.html#s=DYNAMODB
[3] http://netstorage.fgv.br/tjsc2018/retificado1_Edital_TJSC_20_04_2018.pdf

benefícios. O Brasil tem aproximadamente 310 cidades com mais de 100.000 habitantes[4]. Se cada uma destas cidades tem pelo menos um oficial de justiça e considerarmos que nem todos os estados pagam o mesmo valor para oficiais de justiça, calculando uma média (reduzirmos a remuneração pela metade do valor pago por Santa Catarina), tem-se que é pago aproximadamente R$954.277,36 somente em salários, sem adicionar os benefícios e os custos de deslocamento para a realização de uma notificação.

Apresentado os dados, pode-se concluir que existe uma possibilidade de redução de custos dado que pelo menos 10% das notificações realizadas digitalmente obtiveram sucesso sem a necessidade do deslocamento do oficial de justiça. Adicionalmente, a redução de custos para o notificante pode ser ainda maior visto que uma notificação bem sucedida através deste sistema evitaria a necessidade de recorrer a um cartório.

5.1.2 Facilidade de Entrega

Como alternativa de supressão da dificuldade de entrega da notificação, optou-se por ampliar os meios de entrega da notificação. Isto se dá, pelo fato de que para cada meio legal utilizado para entrega da notificação, mais chances existirão de se encontrar com sucesso o notificado desde que ele não esteja se esquivando do notificador.

Adicionalmente, ao permitir sistemas notificantes digitais e conectados à internet, as chances de alcançar uma pessoa aumentam pois as notificações podem ser entregues mesmo que o notificado esteja ausente de sua residência, tomando ciência da notificação em tempo hábil de agir de acordo com o requisitado no documento.

5.1.3 Agilidade na Entrega

O tempo gasto entre o momento em que a notificação é disparada (seja por ordem judicial, sistema notarial ou particular) e a recepção do notificado pode ser muito alto. O prazo para deslocamento de um oficial de justiça ou cartório para realizar a primeira tentativa de entrega de uma notificação pode variar de acordo com a demanda na respectiva comarca. A entrega de uma notificação via correios tem um prazo médio estimado de 2 dias a partir da data de postagem, podendo ser

[4]https://www.ibge.gov.br/estatisticas-novoportal/sociais/populacao.html

mais alto dependendo da eficiência do serviço dos correios.

Serviço	Primeira Tentativa	N° Tentativas	Divulgação
Oficial de Justiça	Variável	2	Privada
Edital	1 a 24 horas	1	Pública
Carta Registrada	$\approx$ 2 dias	1	Privada
Cartório	Variável	3	Púb./Priv.
Email	Instantânea	Configurável	Privada
Blockchain	$\approx$ 3.5 min.[5]	1	Púb./Priv.

Tabela 3 – Informações das formas de notificação

Além das formas de notificação identificadas na tabela 3, o sistema proposto permite a inclusão de novos meios de notificação, como por exemplo, Whatsapp e SMS, que fornecem uma entrega rápida da mensagem ao notificado.

5.1.4 Comprovação de entrega

Neste quesito os modelos atuais de notificação são eficazes embora não sejam eficientes. Os meios digitais de comunicação são muito eficientes em entregar mensagens e até mostrar que a mensagem foi lida, mas confirmar que a pessoa notificada recebeu a mensagem em mãos pode ser um desafio e cabível de interpretações.

A legislação que rege as notificações requerem que seja garantido que a pessoa notificada ou representante legal receba a mensagem em mãos. Existem protocolos que confirmam a abertura ou leitura de email que podem ser uma solução. Outras possibilidade: utilização de links consumíveis com confirmação de leitura, mensagens de Whatsapp (por conter indicador de leitura).

5.2 PONTOS FRACOS

O sistema proposto resolve problemas e aprimora características inerentes de uma notificação. Entretanto, para que estas melhorias sejam possíveis, algumas medidas precisam ser tomadas a

[5] Estimativa realizada em https://ethgasstation.info/calculatorTxV.php com valores 327552 *gas* utilizados e preço de *gas* médio (8.8 *Gwei*)

5.2.1 Publicação em Blockchain Pública

Publicar a notificação em blockchain pública tem a capacidade de substituir o jornal oficial de circulação pública. A dificuldade de se implementar este meio encontra-se em tornar a blockchain um meio oficial de circulação de notícias. A legislação sobre blockchain no Brasil não está consolidada, o que pode dificultar ou atrasar sua adoção pelos orgãos públicos.

Outra dificuldade encontrada é a necessidade que o notificado tenha acesso aos meios de comunicação digital (embora este fato torna-se realidade a cada dia).

5.2.2 Garantia de Entrega de Email

A agilidade de entrega de um email é um fato inegável. Entretanto, uma das dificuldades de realizar a notificação por email, é a garantia que o notificado tomou ciência da notificação. Existem serviços que fornecem a garantia de entrega da mensagem, como é o caso do PostaCertificata[6]. Este serviço baseia-se no protocolo S/MIME (baseado em PKCS#7) e fornece as seguintes características:

- A autenticação do remetente
- Garantia de envio da mensagem pelo remetente (não repúdio)
- Envio da mensagem através do protocolo S/MIME
- Destinatário verifica autenticidade do documento
- Resposta de recebimento ao armazenar mensagem na caixa de email

Serviços de email e mensagens digitais em geral se beneficiariam muito de uma abordagem desta natureza ao adicionar garantias adicionais e mais segurança às mensagens digitais. Embora o PostaCertificata não seja uma aplicação de alta complexidade, sua implantação necessitaria de implementação e regulamentação por parte dos interessados para que então, pudesse ser agregado à legislação brasileira.

[6] https://postacertificata.com/

6 CONCLUSÃO

Este trabalho descreveu as formas de notificações previstas na legislação brasileira e apresentou algumas melhorias e aprimoramentos nas mesmas através de tecnologias digitais atuais, tais como a blockchain.

Dentre os objetivos alcançados, vale destacar, a possibilidade de incrementar o sistema com novas tecnologias permitindo alcançar o notificado e melhorar as chances de sucesso na notificação e a redução de custos da notificação para o Estado, cartórios e para o cidadão.

Embora os principais objetivos tenham sido alcançados, algumas dificuldades podem ser encontradas para implantar o sistema. A maioria das formas de notificação digital têm dificuldade de comprovar o recebimento da diligência por parte do notificado. Outra dificuldade encontrada é a falta de regulamentação da tecnologia blockchain, que tem uma ampla aplicabilidade neste sistema.

6.1 TRABALHOS FUTUROS

A maioria dos objetivos propostos no trabalho foram alcançados na implementação do sistema de notificação. Seguindo o conceito de um "sistema expansível", um dos trabalhos futuros possíveis, é a implementação de novos módulos de saída e entrada com o objetivo de ampliar o alcance aos notificados e a elaboração de módulos que no futuro solucionem de forma adequada o problema da dificuldade de comprovação de recebimento da notificação.

O formato em que o sistema de notificações foi implementado, permite que se monte uma hierarquia de notificadores digitais seguindo o seguinte conceito: Um intermediador central recebe as notificações e a encaminha para outro intermediador pertencente a uma comarca que trata e entrega as requisições conforme determinar mais conveniente.

Outra sugestão de trabalhos futuros, é a monetização do sistema de notificações, visto que os cartórios poderiam se beneficiar das melhorias aqui apresentadas. Grande parte das funcionalidades necessárias para a monetização já foram implementadas e estão disponíveis como apresentado neste trabalho.

Por último, para que o sistema possa ser devidamente implantado, seria necessário ter um cuidado extra quanto à transparência dos dados a fim de não publicar dados sensíveis e protegidos pelas leis.

REFERÊNCIAS

ARAGãO, E. D. M. de. *Comentários ao Código de Processo Civil*. tenth. [S.l.]: Forense, 2004.

BUTERIN, V. A next-generation smart contract and decentralized application platform. p. 1–36, 2013.

CORNELL. *CS 3110 Lecture 21 - Hash functions*. <http://www.cs.cornell.edu/courses/cs3110/2008fa/lectures/lec21.html>.

DANTAS, L. A. ECN: Protocolo Criptográfico para Emissão de Certidões de Nascimento na Internet. 2001. <https://repositorio.ufsc.br/bitstream/handle/123456789/80433/199825.pdf?se

Falcão. *Cartórios vão emitir certidões eletrônicas em todo o país*. 2013. <http://www.cnj.jus.br/images/stories/docs$_c$*orregedoria/recomendacoes/reco*

FIPS, F. I. P. S. DES MODES OF OPERATION. 1980. <https://csrc.nist.gov/CSRC/media/Publications/fips/81/archive/1980-12-02/documents/fips81.pdf>.

GORERI, J. Da citação e sua natureza jurídica e suas modalidades previstas no novo código de processo civil. *Revista Jus Navigandi*, p. ano 23, n. 5317, jan 2018.

GUIMARãES. *Cartórios vão emitir certidões eletrônicas em todo o país*. 2014. <https://exame.abril.com.br/tecnologia/cartorios-vao-emitir-certidoes-eletronicas-em-todo-o-pais/>. Acessado em 27/10/2018.

Isonal. *A imporância dos cartórios*. 2016. <https://3tabelionatodenotas.jusbrasil.com.br/artigos/376794437/a-importancia-dos-cartorios>. Acessado em 26/10/2018.

JúNIOR, H. T. *Curso de Direito Processual Civil*. [S.l.]: Forense, 2014.

KUNDU, S. *How do Digital Signatures keep you protected?* <https://keylines.net/how-do-digital-signatures-keep-you-protected/>.

LISKOV, C. e. Practical byzantine fault tolerance. *OSDI 99: Proceedings of the third symposium on Operating systems design and implementation*, p. 173–186, 1999.

MARINONI, L. *Processo de conhecimento*. Sao Paulo: Ed. Revista dos Tribunais, 2010. ISBN 9788520336335.

Moreira. *Tipos de Cartórios*. 2017. <https://direitoeloisa.jusbrasil.com.br/artigos/481510539/tipos-de-cartorios>. Acessado em 27/10/2018.

MOTA, L. A. M. L. A. A comunicação dos atos processuais no direito brasileiro. *Revista Jus Navigandi*, n. 3614, may 2013.

NAKAMOTO, S. Bitcoin: A Peer-to-Peer Electronic Cash System. *Www.Bitcoin.Org*, p. 9, 2008. ISSN 09254560. <https://bitcoin.org/bitcoin.pdf>.

SZABO, N. Formalizing and Securing Relationships on Public Networks. *First Monday*, 1997. ISSN 1396-0466. <https://doi.org/10.5210/fm.v2i9.548>.

TLAMPORT, S. e. P. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, v. 4, n. 3, p. 382–401, 1982.

APÊNDICE A – Artigo

Sistema Digital para Notificações Baseado em Blockchain

Pablo Rinco Montezano

[1]Departamento de Informática e Estatistica– Universidade Federal de Santa Catarina (UFSC)

Abstract. *The adoption of digital technology tools has grown in several areas including the legal environment. At present, the means by which the notifications are made, often lack agility and guarantees, besides the high cost to the public purse, registry offices and to the citizen. Aiming at reaching mainly these three aspects, the objective of the present work is to allow the digitization of the means of sending notifications foreseen in the Brazilian legislation. To achieve the proposed objectives, some current technologies were applied in order to mitigate the inefficiency of conventional notification methods and allow the implemented tool to be extended to future technologies.*

Resumo. *A adoção de ferramentas tecnológicas digitais cresceu em várias áreas inclusive no meio jurídico. Atualmente, os meios pelos quais são realizadas as notificações, muitas vezes carecem de agilidade e garantias, além do alto custo aos cofres públicos, cartórios e para o cidadão. Visando atingir principalmente estes três aspectos, o objetivo do presente trabalho é permitir a digitalização dos meios de notificação previstos na legislação brasileira. Para alcançar os objetivos propostos, foram aplicadas algumas tecnologias atuais com o intuito de mitigar a ineficiência dos métodos convencionais de notificação e permitir que a ferramenta implementada seja extensível a tecnologias futuras.*

1. Introdução

Os cartórios de títulos e documentos estão presentes em inúmeras fases da vida dos brasileiros [Moreira 2017]. Vários serviços são prestados e todos têm como finalidade garantir alguma forma de segurança. A burocracia inerente aos cartórios é muitas vezes é um dos responsáveis por trazer as garantias e veracidade dos documentos processados. Em artigo escrito ao site jusbrasil[1] [Isonal 2016] define o servidor notarial ou registral como sendo *"um conselheiro comunitário, um agente de prevenção de litígios, visando a paz social"*.

O interesse pela digitalização dos serviços notariais tem crescido, seja para agilizar os serviços ou para garantir a segurança dos dados[Falcão 2013]. Ao longo dos anos várias propostas de digitalização de alguns serviços foram elaboradas como a emissão de registro de nascimento pela internet proposta por Dantas[Dantas 2001], mas não foram aplicados em sua plenitude.

Embora alguns serviços simples possam ser realizados pela internet[Guimarães 2014], a maioria ainda depende do deslocamento do cidadão até um cartório de documentos e registro. Dentre estes está as notificações extrajudiciais.

As notificações judiciais e extrajudiciais, intimações e citações são muito comuns pois tratam de assuntos do cotidiano como por exemplo, notificar um cliente sobre um

[1]https://www.jusbrasil.com.br/

débito. Devido à seriedade deste serviço, sua execução toma tempo e pode ser dificultada pelo notificado ao se esquivar do oficial de justiça ou recusando a assinar o comprovante de entrega do documento.

Visando aprimorar os serviços de notificação, este trabalho tem como objetivo auxiliar estes serviços agregando tecnologias modernas como blockchain, redes sociais e comunicação por dispositivos móveis, ampliando o alcance à pessoa notificada.

2. Conceitos

2.1. Blockchain

Em poucas palavras, Blockchain é uma sequência de blocos encadeados um nos outros. A primeira utilização da blockchain como conhecemos atualmente foi proposta por [Nakamoto 2008] embora o conceito exista desde 1980 como proposto por [FIPS 1980].

Este encadeamento garante a imutabilidade da sequência de blocos, pois um bloco está diretamente ligado ao seu antecessor e assim sucessivamente. Um dos métodos mais simples de encadear blocos digitais é através da utilização de funções de resumo criptográfico (Hash criptográfico). Neste modelo de encadeamento, um bloco seria composto pelos dados do bloco (geralmente denominados transações) e metadados compostos pelo resumo criptográfico do bloco anterior e outras informações importantes para a aplicação. O esquemático apresentado na figura 1 descreve a estrutura básica de uma blockchain.

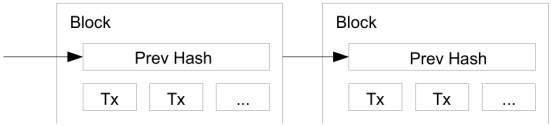

Figura 1. Estrutura básica de uma blockchain. Fonte: adaptação de Bitcoin: A Peer-to-Peer Electronic Cash System

2.2. Contratos Inteligentes

Contratos Inteligentes, do inglês "Smart Contract", foram introduzidos por [Szabo 1997] e recebem este nome por realizarem algumas funções inerentes a um contrato.

Unindo a definição clássica da palavra, com o conceito da palavra *"smart"*, temos um contrato que realiza o intermédio entre duas partes mesmo em um ambiente onde não há confiança entre os envolvidos. Uma vez em execução, sua programação será executada sempre que as condições estabelecidas forem acionadas.

2.3. Ethereum

Em 2013, Vitalik Buterin propôs a ideia de uma maquina virtual descentralizada para a execução de contratos inteligentes, chamada de Ethereum [Buterin 2013].

A definição mais simples dada pelo próprio site da ethereum[2] é a de um "Computador Mundial". Este computador está disponível a todo o tempo, sem interrupções e com quanta memória for necessária. Isso se faz possível, devido à natureza descentralizada

[2]https://ethereum.gitbooks.io/frontier-guide/content/ethereum.html

e distribuída entre os nodos interessados em participar. Esta plataforma de computador mundial, permite que qualquer pessoa possa realizar tarefas computacionais simples, com um baixo custo e sem custo de infraestrutura enquanto fornece algumas características:

- Autenticação de usuário
- Lógica de pagamento completamente customizável
- 100% resistente a DDOS
- Armazenamento de dados seguro (mas não privado)
- Interoperabilidade entre os componentes do sistema Ethereum
- Não necessita de servidores

2.4. Notificações

No Brasil, notificações são documentos utilizadas para dar ciência de algo e garantir que a pessoa o tenha recebido. Existem alguns tipos de notificações com propósitos diferentes extraídos dos artigos 726 a 729. Abaixo uma descrição sucinta dos tipos de notificação com respaldo legal:

- Citação: certifica o réu da existência de uma demanda ajuizada em seu desfavor a integrar a relação processual.
- Intimação: dar ciência a alguém dos atos e dos termos de um processo em andamento. Muito utilizado para intimar testemunhas.
- Notificação judicial: manifesta formalmente através de uma autoridade do poder judiciário a vontade de alguém a outrem sobre assunto juridicamente relevante dando-lhe ciência de seu propósito.
- Notificação extra-judicial: manifesta formalmente a vontade de alguém a outrem sobre assunto juridicamente relevante dando-lhe ciência de seu propósito.

O Código de Processos Civil prevê vários meios de enviar as notificações, cada um com suas especificidades dependendo do tipo da notificação. Os meios de envio de notificação mais comumente utilizados e geralmente preferenciais são por Correios (carta com aviso de recebimento), por oficial de justiça, por hora certa, por edital e por meios eletrônicos.

3. Proposta

Este capítulo tem como objetivo descrever a solução proposta neste trabalho. A seção 3.1 descreve uma visão geral do projeto, a seção 3.2 detalha a arquitetura do projeto em módulos e componentes.

3.1. Descrição

Uma notificação tem como principal objetivo, informar uma pessoa algo que venha a ser de seu interesse. A maior dificuldade de se realizar uma notificação com sucesso é prover provas de que o notificado recebeu a notificação em mãos. Este é o principal motivo pelo qual o cidadão opta por utilizar os serviços do cartório de títulos e as cartas com aviso de recebimento.

Para alcançar este objetivo, foi elaborado um sistema digital intermediador, capaz de enviar uma notificação por vários meios eletrônicos simultaneamente, ampliando as chances de uma notificação bem sucedida. Este sistema intermediador recebe, processa e encaminha as notificações.

3.2. Arquitetura do Projeto

Conceitualmente o sistema é dividido em três módulos, compostos por componentes menores. A imagem 2 ilustra uma visão geral da arquitetura do projeto. A subseção 3.2.1 descreve as interações entre o usuário e o sistema, a subseção 3.2.2 apresenta os sistemas auxiliares e a subseção 3.2.3 demonstra um pouco dos sistemas notificadores implementados e como estender os tipos de notificações.

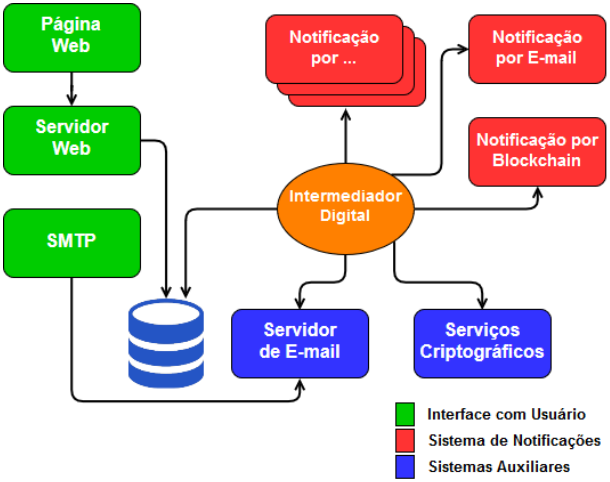

Figura 2. Diagrama de relacionamento entre os componentes

3.2.1. Interface com Usuário

Este módulo tem como principal objetivo permitir que o interessado em efetuar uma notificação extrajudicial, possa emitir uma requisição de notificação que será analisada, processada e encaminhada ao notificado.

Todo sistema computacional com foco na utilização do usuário, deve ter uma boa usabilidade e abranger o maior público possível. Para alcançar este objetivo, o sistema permite que as requisições de notificação sejam enviados por dois meios: por email - respeitando o formato pré estabelecido dos dados esperados - e por um banco de dados onde são armazenadas as notificações ainda não processadas. O usuário pode então, acessar a página web pela internet e emitir sua requisição de notificação pelos campos do formulário. O back-end da página web está preparado para trabalhar com outros formatos front-end, como por exemplo, aplicativos para dispositivos móveis.

3.2.2. Sistemas Auxiliares

Para alcançar todas as funcionalidades e características do sistema, alguns serviços foram necessários para garantir a persistência, acessibilidade e segurança dos dados.

A persistência e acessibilidade fazem uso de um banco de dados tanto para as notificações processadas quanto para as não processadas. Um servidor de emails exerce o papel de gerenciar as notificações enviadas por email, servindo também como base de dados para as notificações requisitadas através deste meio de comunicação.

Para garantir a autenticidade e evitar que notificações falsas sejam emitidas em nome do sistema, é utilizado o componente de serviços criptográficos. Este componente necessita de um certificado assinado por uma ICP (ICP-Brasil) de confiança afim de assinar as notificações enviadas permitindo que o notificado verifique a autenticidade da mensagem que acabou de receber. Dentre as operações realizadas por este componente estão a gerencia de certificados e chaves, assinatura e verificação de assinatura dos documentos.

3.2.3. Sistema de Notificações

Este sistema foi desenvolvido para permitir que a quantidade e tipo de notificações enviadas simultaneamente ao notificado possa ser configurado e ampliado. Entretanto, a fim de demonstração, foram implementados dois tipos de envio de notificação: por email e publicação em blockchain pública.

A notificação por email é composta pela mensagem na integra, com todos os dados da notificação enviada pelo notificante e do documento assinado pelo intermediador. A mensagem deve conter informações relevantes do notificante como: nome, endereço, email, etc, o texto que trata do assunto da notificação e informações do notificado.

A notificação publicada em blockchain pública tem o objetivo de servir como um jornal ou periódico de circulação livre. Diferentemente da notificação por email, as informações apresentadas são simplificadas a fim de evitar a divulgação de dados sensíveis. A presença destes dados em uma blockchain pública permite por exemplo, que uma entidade interessada tenha uma página web de notícias ou busca que informe as notificações emitidas.

Dentre os dados presentes nesta blockchain estão o email do notificante, cpf do notificado, o hash do documento de notificação e o carimbo de tempo.

Para tipos de notificação que podem ser efetuados por meios digitais, pode-se implementar um notificador que contenha todos os dados necessários para uma notificação desde que respeitada a Lei Geral de Proteção de Dados Pessoais[3]

4. Desenvolvimento

Nesta seção será apresentado todo o desenvolvimento do projeto proposto na seção **??**.

4.1. Contratos Inteligentes

Inicialmente o projeto teve como foco, notificações compostas quase completamente por artefatos presentes ou armazenados em blockchain. Após análises e protótipos elaborados, percebeu-se que esta abordagem aumentaria muito o custo das transações, o que mudou o rumo da aplicação da blockchain no projeto.

[3]http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2018/Lei/L13709.htm

Com base nos requisitos levantados, foi elaborado o código do contrato inteligente como o contido na listagem **??** de forma a armazenar informações que permitem a inferência de uma notificação, mas que não é completa por si só.

```
struct Notification {
    string notif_hash;
    string sender_mail;
    string dest_mail;
    uint32 timestamp;
}
mapping (int40 => Notification[]) notifications;
```

Embora pude-se se utilizar a própria data da Ethereum como forma de estampa do tempo, este poderia não refletir o tempo em que a notificação foi primeiramente publicada, devido ao tempo até que a transação seja minerada pela rede. Desta forma, optou-se por enviar também a estampa do tempo em que a notificação foi processada e publicada pela autoridade e armazenando-a na variável *timestamp*. Por último, o identificador único do notificado é enviado, mas serve apenas como índice da "hash table" *notifications*.

Além do armazenamento de dados, o contrato dispõe de algumas funcionalidades para receber e listar notificações. Adicionalmente, foram implementadas algumas funcionalidades administrativas, como por exemplo, adicionar novas autoridades notificadoras. Abaixo segue uma breve listagem dos métodos e suas funcionalidades:

- *notify*: recebe uma notificação com os dados descritos acima
- *addPermission*: adiciona permissão para uma nova autoridade notificadora
- *numberOfNotifications*: lista a quantidade de notificações um CPF contém
- *latestNotification*: retorna a última notificação emitida para um CPF
- *specificNotification*: retorna uma notificação a partir de um CPF e um índice

Otimizações do Contrato Inteligente

Algumas otimizações de código próprias da Ethereum foram aplicadas a fim de reduzir o custo da implantação do contrato e execução das transações.

Dentre as otimizações realizadas, vale citar: o uso de modificadores, reduzindo o custo da execução e implantação dos contratos; uso de memry e storage, reduzindo o custo de execução da transação;

4.2. Servidores da Aplicação

Analisando o projeto definido no capítulo **??**, identificou-se rapidamente que a melhor forma de desenvolver a aplicação, era elaborar um intermediador entre entre a entrada de dados (inicialmente por email) e a saída de dados (por email e blockchain, simultaneamente). Baseando-se nestas análises e nos componentes já implantados: Blockchain e servidor de email, decidiu-se implementar um servidor intermediador, expansível em todas as suas funcionalidades: entrada, processamento e saída.

4.2.1. Servidor Intermediador

Implementação

Duas opções de linguagem de programação foram levantadas como possibilidade para a implementação do servidor principal intermediador: Java e JavaScript. Embora Java seja uma linguagem consolidada no meio corporativo pela sua robustez e portabilidade, a api fornecida para comunicação com nodos Ethereum não são amigáveis, tornando o desenvolvimento lento. Ao contrário do Java, a api fornecida pelo projeto web3 para Node.js assemelha-se muito aos comandos executados diretamente na interface de linha de comandos Geth, utilizada para inicializar o nodo Ethereum da nossa rede privada. Adicionalmente, outras tecnologias agregadas ao decorrer do desenvolvimento do projeto teriam um alto grau de compatibilidade com o Node.js. Desta forma, ao longo do desenvolvimento, decidiu-se seguir utilizando JavaScript como linguagem de programação base do servidor intermediador.

O servidor é composto de uma função principal que procura novas notificações e executa uma função que é passada por parâmetro para ser executada por cada uma das notificações. O código desta função é apresentado abaixo:

Listing 1. Função de busca e processamento de notificações

```
let imapPrivate = new Imap(config.private_mail);
mail.fetchAttachments(imapPrivate, sendNotification,
    () => console.log('error fetching notifications')
);
```

A função *sendNotification* realiza todo o trabalho que necessita ser realizado para preparar a notificação e então emiti-la.

```
function sendNotification(
  sender_mail, cpf, dest, attachment
) {
  let transporter = nodemailer.createTransport(config.
     private_mail_sender);
  // Configura envio de emails
  crypto.sign(attachment).then((cipherAttachment) =>
  {
    User.findOne({email: sender_mail}, function (err, user) {
      let userExist = user !== null;
      if(!userExist) {
        // Cria usuario caso ainda nao exista
      }
      crypto.hash(attachment).then(
        (hash) =>{
          // Envia notificacao para a rede ethereum
          blockchain.notify(hash, sender_mail, cpf, dest)
          let data = {
            // Dados da notificacao
          };
        let notif = new Notification(data);
        notif.save()
        // Salta as notificacoes em base de dados
        mail_sender.sendMail(transporter, dest, [attachments]);
        mail_sender.sendMail(transporter, sender_mail,
           confirmation);
        // Envia email de confirmacao ao notificante e
        // notificacao com anexos ao notificado.
```

```
            // Adicionar aqui outras formas de notificacao
        }
    )
  }
}
```

Listing 2. Função de processamento das notificações

Especificação dos Contratos de Entrada e Saída

A implementação final do servidor intermediador busca as notificações em um servidor de emails e em uma base de dados NoSQL MongoDB. Inicialmente, o intermediador foi configurado para buscar os emails não lidos na caixa de entrada da conta de email notificationsmailchain.org para processa-los. Os campos do email para requisitar uma notificação seguem o padrão:

- Destinatário: “destinatariodominio.com”mailchain.org
- Assunto: CPF do notificado
- Campo de texto: campo ignorado, pode conter qualquer informação
- Anexo: arquivo .pdf redigido seguindo os padrões válidos de notificação

Por último, foi adicionado ao sistema uma base de dados NoSQL para as notificações não processadas. A solução foi modelada com duas relações: Notifications e Unprocessed. A descrição simplificada das duas relações pode ser encontrada nas duas listagens que seguem.

```
{
    "author": { type: ObjectId, ref: 'User' },
    "destMail": String,
    "destCpf": Number,
    "attachment": String,
    "hash": String,
    "transaction": String
}
```

Listing 3. Esquema da relação Notification.

```
{
    "unprocessed": { type: ObjectId, ref: 'Notification' }
}
```

Listing 4. Esquema da relaçao Unprocessed.

O servidor intermediador busca as entradas na relação Unprocessed que tem uma ligação para uma entrada na relação Notifications que contem os dados completos da notificação. De posse da notificação, o servidor intermediador processa os dados obtidos da relação, e caso seja executado com sucesso, remove a entrada presente na relação Unprocessed.

4.3. Servidor Web

Com foco na atualização tecnológica e garantias de segurança do sistema, foi desenvolvido um servidor web com a linguagem de programação Javascript e framework Node.js, configurado interfaces REST, autenticação através do JWT e escrita no banco de dados MongoDB disponibilizado pelo site mlab[4].

O servidor conta com várias interfaces REST, onde algumas necessitam de autenticação, e outras não. Em seguida as interfaces que não necessitam de autenticação ou que são opcionais:

- GET 'notifications/': retorna todas as notificações armazenadas no banco de dados
- GET 'notifications/:notification': retorna uma notificação específica representada pelos dados presentes em :notification

Abaixo as interfaces REST onde a autenticação é obrigatória:

- GET 'notifications/feed': retorna todas as notificações do usuário logado
- POST 'notifications/': envia a requisição de notificação

A figura 3 apresenta o diagrama com descrição formal das interfaces REST do sistema.

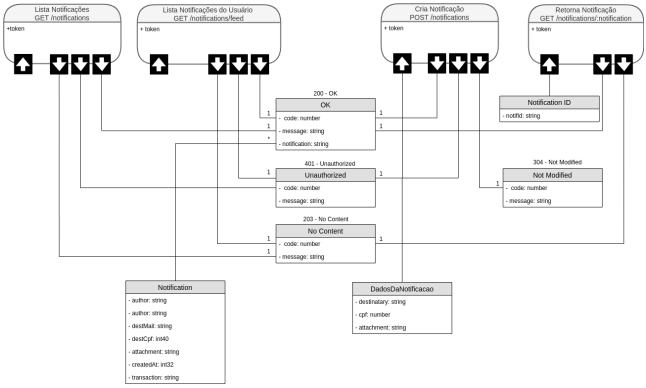

Figura 3. Diagrama da descrição formal das interfaces REST.
Fonte: Elaboração própria

Esta camada adicionada traz grandes benefícios ao permitir que aplicativos móveis, e qualquer outro tipo de aplicativo que se comunique através deste modelo, possam ser facilmente desenvolvidos para enviarem notificações ao servidor web través das interfaces REST implementadas aqui.

4.4. Página Web

A implementação do servidor web disponibilizou para as aplicações e programadores as funcionalidades apresentadas na seção 4.3 mas é inviável a utilização dos recursos por

[4]https://mlab.com/

usuários comuns. Adicionalmente, um quadro de listagem das transações blockchain relacionadas ao projeto foi adicionado à página web a fim de demonstrar seu funcionamento em conjunto com as notificações. As funcionalidades básicas implementadas na página web foram:

- Autenticação de usuário
- Emissão de notificação
- Visualização das notificações enviadas pelo usuário autenticado
- Visualização das notificações enviadas por todos os usuários
- Visualização das transações ethereum referentes às notificações

Dentre as várias páginas criadas, a principal delas é a apresentada ao usuário logado no sistema. Nela é possível enviar uma nova notificação, verificar informações das notificações enviadas e dados das notificações no contrato Ethereum. A figura 4 apresenta a tela que pode ser vista por um usuário logado no sistema.

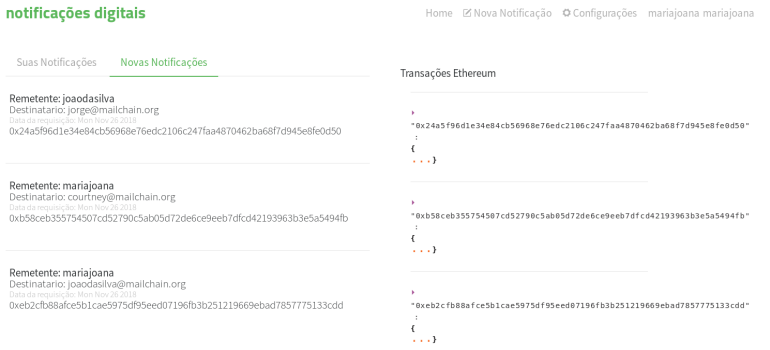

Figura 4. Página principal

5. Resultados

A principal motivação para a a elaboração deste trabalho foi a de melhorar a eficiência das notificações disponíveis na legislação brasileira. Dentre as melhorias provenientes do sistema proposto em relação ao modelo atual, obtivemos redução do custo, agilidade na entrega e comprovação de entrega para alguns modelos de notificação.

5.1. Redução de custo

Dentre as melhorias citadas, a redução de custo beneficia-se do baixo custo de se manter servidores elásticos executando as notificações. Ao se utilizar este tipo de servidor, o tempo ocioso do servidor é reduzido e por consequência, os custos. Sempre que uma nova notificação é requisitada e obtém sucesso, outros métodos convencionais e mais dispendiosos são evitados. As tabelas 1 e 2 apresentam os custos de manter o sistema em execução e o custo por notificação, respectivamente.

Serviço	Qtd	Custo Max./Hora	Total Aprox.
Amazon EC2 c5.18xlarge	2	$4,716	$\approx \$6791,04$
Amazon EC2 c4.8xlarge	2	$2,47	$\approx \$3556,80$
Amazon DynamoDB	1	-	$\approx \$1142,08$
Total Mensal	-	-	$\approx \$11495,92$

Tabela 1. Custos com servidores e banco de dados

O notificador baseado em blockchain ethereum é um dos módulos que tem custo por notificação baseado no custo de uma transação ethereum. A tabela 2 apresenta os custos referentes a este notificador.

Serviço	Custo em Ether	Custo em Dólar
Implantação do Contrato	1910554 Gas	$\approx \$1.12$
Envio de Notificação	$\approx$ 327552 Gas	$\approx \$0.19$

Tabela 2. Custos da blockchain ethereum

5.2. Agilidade na entrega

O tempo gasto entre o momento em que a notificação é disparada (seja por ordem judicial, sistema notarial ou particular) e a recepção do notificado pode ser muito alto. O prazo para deslocamento de um oficial de justiça ou cartório para realizar a primeira tentativa de entrega de uma notificação pode variar de acordo com a demanda na respectiva comarca. A entrega de uma notificação via correios tem um prazo médio estimado de 2 dias a partir da data de postagem, podendo ser mais alto dependendo da eficiência do serviço dos correios.

Serviço	Primeira Tentativa	N° Tentativas	Divulgação
Oficial de Justiça	Variável	2	Privada
Edital	1 a 24 horas	1	Pública
Carta Registrada	$\approx$ 2 dias	1	Privada
Cartório	Variável	3	Púb./Priv.
Email	Instantânea	Configurável	Privada
Blockchain	$\approx$ 3.5 min.	1	Púb./Priv.

Tabela 3. Informações das formas de notificação

Além das formas de notificação identificadas na tabela 3, o sistema proposto permite a inclusão de novos meios de notificação, como por exemplo, Whatsapp e SMS, que fornecem uma entrega rápida da mensagem ao notificado.

5.3. Comprovação de entrega

Neste quesito os modelos atuais de notificação são eficazes embora não sejam eficientes. Os meios digitais de comunicação são muito eficientes em entregar mensagens e até mostrar que a mensagem foi lida, mas confirmar que a pessoa notificada recebeu a mensagem em mãos pode ser um desafio e cabível de interpretações.

A legislação que rege as notificações requerem que seja garantido que a pessoa notificada ou representante legal receba a mensagem em mãos. Existem protocolos que confirmam a abertura ou leitura de email que podem ser uma solução. Outras possibilidade:

utilização de links consumíveis com confirmação de leitura, mensagens de Whatsapp (por conter indicador de leitura).

6. Conclusão

Este trabalho descreveu as formas de notificações previstas na legislação brasileira e apresentou algumas melhorias e aprimoramentos nas mesmas através de tecnologias digitais atuais, tais como a blockchain.

Dentre os objetivos alcançados, vale destacar, a possibilidade de incrementar o sistema com novas tecnologias permitindo alcançar o notificado e melhorar as chances de sucesso na notificação e a redução de custos da notificação para o Estado, cartórios e para o cidadão.

Embora os principais objetivos tenham sido alcançados, algumas dificuldades podem ser encontradas para implantar o sistema. A maioria das formas de notificação digital têm dificuldade de comprovar o recebimento da diligência por parte do notificado. Outra dificuldade encontrada é a falta de regulamentação da tecnologia blockchain, que tem uma ampla aplicabilidade neste sistema.

6.1. Trabalhos Futuros

A maioria dos objetivos propostos no trabalho foram alcançados na implementação do sistema de notificação. Seguindo o conceito de um "sistema expansível", um dos trabalhos futuros possíveis, é a implementação de novos módulos de saída e entrada com o objetivo de ampliar o alcance aos notificados e a elaboração de módulos que no futuro solucionem de forma adequada o problema da dificuldade de comprovação de recebimento da notificação.

O formato em que o sistema de notificações foi implementado, permite que se monte uma hierarquia de notificadores digitais seguindo o seguinte conceito: Um intermediador central recebe as notificações e a encaminha para outro intermediador pertencente a uma comarca que trata e entrega as requisições conforme determinar mais conveniente.

Outra sugestão de trabalhos futuros, é a monetização do sistema de notificações, visto que os cartórios poderiam se beneficiar das melhorias aqui apresentadas. Grande parte das funcionalidades necessárias para a monetização já foram implementadas e estão disponíveis como apresentado neste trabalho.

Por último, para que o sistema possa ser devidamente implantado, seria necessário ter um cuidado extra quanto à transparência dos dados a fim de não publicar dados sensíveis e protegidos pelas leis.

Referências

Buterin, V. (2013). A next-generation smart contract and decentralized application platform. pages 1–36.

Dantas, L. A. (2001). ECN: Protocolo Criptográfico para Emissão de Certidões de Nascimento na Internet.

Falcão (2013). Cartórios vão emitir certidões eletrônicas em todo o país.

FIPS, F. I. P. S. (1980). DES MODES OF OPERATION.

Guimarães (2014). Cartórios vão emitir certidões eletrônicas em todo o país.

Isonal (2016). A imporância dos cartórios.

Moreira (2017). Tipos de Cartórios.

Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. *Www.Bitcoin.Org*, page 9.

Szabo, N. (1997). Formalizing and Securing Relationships on Public Networks. *First Monday*.

ANEXO A – Código

```
var jwt = require('express-jwt');
var secret = require('../config').secret;

function getTokenFromHeader(req){
    if (req.headers.authorization && req.headers.
        authorization.split(' ')[0] === 'Token' ||
        req.headers.authorization && req.headers.
            authorization.split(' ')[0] === 'Bearer')
            {
        return req.headers.authorization.split(' ')
            [1];
    }

    return null;
}

var auth = {
    required: jwt({
        secret: secret,
        userProperty: 'payload',
        getToken: getTokenFromHeader
    }),
    optional: jwt({
        secret: secret,
        userProperty: 'payload',
        credentialsRequired: false,
        getToken: getTokenFromHeader
    })
};

module.exports = auth;
```

Listing A.1 – broker/auth.js

<!—->

```
const Web3 = require('web3');

const blockchainConfig = {
    nodeAccountAddress: "0
        xd7904971ab3b5c4cc6f6781ee60324c423d4e94d",
    nodeRPCAddress: 'http://',
    nodeWSAddress: 'ws://',
    gasPrice: '20000000000'
};

const web3 = new Web3(new Web3.providers.
    WebsocketProvider(blockchainConfig.nodeWSAddress,
    {
    headers: {
        Origin: "narga"
    }
```

```
}));
web3.eth.defaultAccount = blockchainConfig.
    nodeAccountAddress;

export {blockchainConfig, web3};
```

Listing A.2 – broker/BlockchainConfig.js

<!—->

```
const notificationContract = {
    descentralizedNotificationContract: [{"constant":
        true,"inputs":[{"name":"cpf","type":"int40
        "}],"name":"latestNotification","outputs":[{"
        name":"","type":"string"},{"name":"","type":"
        string"},{"name":"","type":"string"}],"payable
        ":false,"stateMutability":"view","type":"
        function"},{"constant":false,"inputs":[{"name
        ":"cpf","type":"int40"},{"name":"notif_hash","
        type":"string"},{"name":"senderMail","type":"
        string"},{"name":"destMail","type":"string
        "},{"name":"timestamp","type":"uint32"}],"name
        ":"notify","outputs":[],"payable":false,"
        stateMutability":"nonpayable","type":"function
        "},{"constant":false,"inputs":[{"name":"
        notifier","type":"address"}],"name":"
        addPermission","outputs":[],"payable":false,"
        stateMutability":"nonpayable","type":"function
        "},{"constant":true,"inputs":[{"name":"cpf","
        type":"int40"}],"name":"numberOfNotifications
        ","outputs":[{"name":"","type":"uint256"}],"
        payable":false,"stateMutability":"view","type
        ":"function"},{"constant":true,"inputs":[{"
        name":"cpf","type":"int40"},{"name":"index","
        type":"uint256"}],"name":"specificNotification
        ","outputs":[{"name":"","type":"string"},{"
        name":"","type":"string"},{"name":"","type":"
        string"}],"payable":false,"stateMutability":"
        view","type":"function"},{"inputs":[],"payable
        ":false,"stateMutability":"nonpayable","type
        ":"constructor"},{"anonymous":false,"inputs
        ":[{"indexed":false,"name":"senderMail","type
        ":"string"},{"indexed":false,"name":"destMail
        ","type":"string"},{"indexed":false,"name":"
        notif_hash","type":"string"},{"indexed":false
        ,"name":"timestamp","type":"uint32"}],"name":"
        NewNotification","type":"event"}],
    descentralizedNotificationContractAddress: '0
        x71c662a2347856c3e82132997cfcd15b83be2aca'
};

```

```
6 export {notificationContract};
```

Listing A.3 – broker/BlockchainContracts.js

<!—->

```
1  import nodemailer from 'nodemailer';
2  import mongoose from 'mongoose';
3
4  import * as crypto from './service/CryptoModule';
5  import * as config from '../config';
6  import * as mail from './service/MailFetcher';
7  import * as mail_sender from './service/MailSender'
8  import * as blockchain from './service/blockchain/
       EthereumConnectionHandler';
9  require('./models/User');
10 require('./models/Notification');
11 require('./models/Unprocessed');
12 const Notification = mongoose.model('Notification');
13 const Unprocessed = mongoose.model('Unprocessed');
14 const User = mongoose.model('User');
15
16
17 function prepareNotifications() {
18
19     var Imap = require('imap');
20
21     let imapPrivate = new Imap(config.private_mail);
22     mail.fetchAttachments(imapPrivate,
           sendNotification,
23         () => console.log('error signing attachment')
24     );
25
26     Unprocessed.find({}, 'unprocessed', function (err,
           docs) {
27
28         docs.forEach((element, index, array) => {
29             Notification.findById(element.unprocessed,
                   function (err, notif) {
30                 User.findById(notif.author, function(
                       err, user) {
31                     sendNotification(user.email, notif
                           .destCpf, notif.destMail,
                           notif.attachment)
32                 });
33                 Unprocessed.deleteOne({unprocessed:
                       notif._id}, function (err) {})
34             })
35         })
36     });
37 }
38
```

```
39 function sendNotification (sender_mail, cpf, dest,
       attachment) {
40     let transporter = nodemailer.createTransport(
           config.private_mail_sender);
41
42     crypto.sign(attachment).then((cipherAttachment) =>
43         {
44             User.findOne({email: sender_mail},
                   function (err, user) {
45                 let userExist = user !== null;
46                 if(!userExist) {
47                     user = new User();
48                     user.username = sender_mail.
                           substring(0, sender_mail.
                           indexOf("@"));
49                     user.email = sender_mail;
50                     user.setPassword("123456");
51
52                     user.save();
53                 }
54
55                 crypto.hash(attachment)
56                     .then(
57                         (hash) =>{
58
59                             blockchain.notify(hash,
                                   sender_mail, cpf, dest
                                   ).then((
                                   newContractInstance) =
                                   > {
60
61                                 let data = {
62                                     author: user._id,
63                                     destMail: dest,
64                                     destCpf: cpf,
65                                     attachment:
                                           attachment,
66                                     hash: hash,
67                                     transaction:
                                           newContractInstance
                                           .
                                           transactionHash
68                                 };
69                                 let notif = new
                                       Notification(data)
                                       ;
70
71                                 notif.save(function(
                                       err) {
72                                     if (err) {
73                                         console.log(
```

```
                                        err);
74                                  }
75                                  else {
76                                      mail_sender.
                                            sendMail(
                                            transporter
                                            , dest, [
77                                          {
78                                              filename
                                                    :
                                                    '
                                                    Notificatio
                                                    .
                                                    pdf
                                                    ',
79                                              content
                                                    :
                                                    attachment
                                                    ,
80                                              encoding
                                                    :
                                                    '
                                                    base64
                                                    '
81                                          },
82                                          {
83                                              filename
                                                    :
                                                    '
                                                    signature
                                                    .
                                                    p7
                                                    ',
84                                              content
                                                    :
                                                    cipherAttac
                                                    ,
85                                              contentType
                                                    :
                                                    '
                                                    text
                                                    /
                                                    plain
                                                    '
86                                          }]
87                                      );
88                                  }
89
90                                  let confirmation =
                                        'Your
                                        notification
                                        was
```

```
                                        successfully
                                        sent.\n' +
91                                      'To track the
                                            progress
                                            of your
                                            notification
                                            , access
                                            http://
                                            your.
                                            notification
                                            .com and
                                            login with
                                            :\n' +
92                                      'Username: ' +
                                             user.
                                            username +
                                             '\n' +
93                                      (userExist? ''
                                             : '
                                            Password:
                                            123456\
                                            nPlease,
                                            change
                                            your
                                            password
                                            as soon as
                                             possible
                                            .');
94                                  mail_sender.
                                        sendMail(
                                        transporter,
                                        sender_mail,
                                        confirmation);
95                                  });
96                              })
97                          }, () => console.log("Error
                                hashing the data")
98                      )
99                  }
100             );
101         }
102     )
103 }
104
105 export {prepareNotifications, sendNotification}
```

Listing A.4 – broker/Broker.js

<!—->

```
1 // export default ()();
2
```

```
3
4  const server = {
5              hostname: "127.0.0.1",
6              port: 3001
7          };
8  const private_mail = {
9              user: 'notifications@mailchain.org',
10             password: '',
11             host: 'mail.blockchain.org',
12             port: 993,
13             tls: true,
14             tlsOptions: {
15                 rejectUnauthorized: false
16             }
17         };
18 const public_mail = {
19             user: 'notifications@mailchain.org',
20             password: '',
21             host: 'mail.blockchain.org',
22             port: 993,
23             tls: true,
24             tlsOptions: {
25                 rejectUnauthorized: false
26             }
27         };
28
29 const private_mail_sender = {
30             host: 'mail.blockchain.org',
31             port: 587,
32             secure: false, // true for 465, false for
                   other ports
33             auth: {
34                 user: 'notifications@mailchain.org',
                       // generated ethereal user
35                 pass: '' // generated ethereal
                       password
36             },
37             tls: {
38                 // do not fail on invalid certs
39                 rejectUnauthorized: false
40             }
41         };
42
43 export {server, private_mail, private_mail_sender,
       public_mail }
```

Listing A.5 – broker/config.js

<!—->

```
1  var forge = require('node-forge');
2  var pkcs7 = forge.pkcs7;
```

```
3  var fs = require('fs');
4  const crypto = require('crypto');
5
6  var folder = './resources/';
7  var cert = fs.readFileSync(folder + 'server.crt').
       toString();
8  var key = fs.readFileSync(folder + 'server.key').
       toString();
9
10 function verify(pkg_sig, pkg) {
11     var crypto = require('crypto');
12
13     var msg = pkcs7.messageFromPem(pkg_sig);
14     var sig = msg.rawCapture.signature;
15
16     var buf = new Buffer(pkg, 'binary');
17
18     var verifier = crypto.createVerify("RSA-SHA256");
19     verifier.update(buf);
20     var verified = verifier.verify(cert, sig, 'binary
           ');
21
22     console.log('verification is: %s', verified);
23 }
24
25 function sign(pkg) {
26
27
28     return new Promise((resolve, reject) => {
29         var p7 = pkcs7.createSignedData();
30         p7.content = forge.util.createBuffer(pkg);
31         p7.addCertificate(cert);
32         p7.addSigner({
33             key: forge.pki.privateKeyFromPem(key),
34             certificate: cert,
35             digestAlgorithm: forge.pki.oids.sha256
36         });
37         p7.sign({detached: true});
38         resolve(pkcs7.messageToPem(p7));
39     })
40 }
41
42 function hash(pkg) {
43     return new Promise((resolve, reject) => {
44         let hash = crypto.createHash('sha256');
45         hash.update(pkg);
46         resolve(hash.digest('base64'));
47     })
48 }
49
```

```
50 export {sign, verify, hash};
```

Listing A.6 – broker/CryptoModule.js

<!—->

```
1  pragma solidity ^0.4.7;
2  contract DescentralizedNotification {
3
4      struct Notification {
5          string notif_hash;
6          string sender_mail;
7          string dest_mail;
8          uint32 timestamp;
9      }
10
11     event NewNotification(
12         string sender_mail,
13         string dest_mail,
14         string notif_hash,
15         uint32 timestamp
16     );
17
18     mapping (int40 => Notification[]) notifications;
19     address private owner = msg.sender;
20     mapping (address => bool) notifiers;
21
22     constructor() public {
23         owner = msg.sender;
24         notifiers[owner] = true;
25     }
26
27     modifier onlyOwner() {
28         require(msg.sender == owner);
29         _;
30     }
31
32     modifier onlyNotifier() {
33         require(notifiers[msg.sender]);
34         _;
35     }
36
37     function notify(int40 cpf, string notif_hash,
           string sender_mail, string dest_mail, uint32
           timestamp)
38     public
39     onlyNotifier()
40     {
41
42         Notification memory not = Notification(
               notif_hash, sender_mail, dest_mail,
               timestamp);
```

```
43          notifications[cpf].push(not);
44          emit NewNotification(sender_mail, dest_mail,
                notif_hash, timestamp);
45      }
46
47      function addPermission(address notifier)
48      public
49      onlyOwner()
50      {
51          notifiers[notifier] = true;
52      }
53
54      function numberOfNotifications(int40 cpf) public
            view returns(uint256) {
55          Notification[] storage not = notifications[cpf
                ];
56          return not.length;
57      }
58
59      function latestNotification(int40 cpf) public view
             returns(string, string, string){
60          Notification[] storage not = notifications[cpf
                ];
61          uint256 index = not.length-1;
62          return (not[index].sender_mail, not[index].
                dest_mail, not[index].notif_hash);
63      }
64
65      function specificNotification(int40 cpf, uint256
            index) public view returns(string, string,
            string){
66          Notification[] storage not = notifications[cpf
                ];
67          return (not[index].notif_hash, not[index].
                sender_mail, not[index].dest_mail);
68      }
69  }
```

Listing A.7 – broker/DescentralizedNotification.sol

<!—->

```
1   import * as blockchain from './BlockchainConfig';
2   import * as contracts from './BlockchainContracts';
3
4   var countContract = new blockchain.web3.eth.Contract(
        contracts.notificationContract.
        descentralizedNotificationContract,
5       contracts.notificationContract.
            descentralizedNotificationContractAddress, {
6           from: blockchain.blockchainConfig.
                nodeAccountAddress,
```

```
7           gasPrice: blockchain.blockchainConfig.gasPrice
8       });
9
10  /*
11  Receive the CPF to consult as parameter and return a
        Promise with a
12  JSON parameter with the following struture
13  Result {
14      '0': 'newname',
15      '1': 'newsender@mail.com',
16      '2': 'receipient@hothot.com',
17      '3': 'HASHNOTIFICATION2' }
18   */
19  function latestNotification(cpf) {
20      return countContract.methods.latestNotification(
            cpf)
21          .call({from: blockchain.blockchainConfig.
                nodeAccountAddress})
22  }
23
24  /*
25  Receive the CPF to consult  and index of the desired
        notification
26   as parameter and return a Promise with a JSON
         parameter with the
27   following structure:
28  Result {
29      '0': 'newname',
30      '1': 'newsender@mail.com',
31      '2': 'receipient@hothot.com',
32      '3': 'HASHNOTIFICATION2' }
33   */
34  function specificNotification(cpf, index) {
35      return countContract.methods.latestNotification(
            cpf, index)
36          .call({from: blockchain.blockchainConfig.
                nodeAccountAddress})
37  }
38
39  /*
40  Create a notification to the CPF listed. The
        notification must include
41  the hash of the notification document, sender email
        address, sender name
42  or any alias and destinatary mail.
43  The timestamp will be generated on this requisition,
        not the blockchain time.
44   */
45  function notify(hash, sender_mail, cpf, dest_mail) {
46      return countContract.methods.notify(cpf, hash,
            sender_mail, dest_mail, new Date().getTime())
47          .send({
```

```
48              from: blockchain.blockchainConfig.
                    nodeAccountAddress,
49              gas: 200000
50          });
51  }
52  export {latestNotification, specificNotification,
        notify}
```

Listing A.8 – broker/EthereumConnectionHandler.js

<!—->

```
1   var router = require('express').Router();
2
3   router.use('/', require('./users'));
4   router.use('/notifications', require('./notifications
        '));
5
6   router.use(function(err, req, res, next){
7       if(err.name === 'ValidationError'){
8           return res.status(422).json({
9               errors: Object.keys(err.errors).reduce(
                    function(errors, key){
10                  errors[key] = err.errors[key].message;
11
12                  return errors;
13              }, {})
14          });
15      }
16
17      return next(err);
18  });
19
20  module.exports = router;
```

Listing A.9 – broker/index.js

<!—->

```
1   var inspect = require('util').inspect;
2   var fs      = require('fs');
3   var base64  = require('base64-stream');
4   var Imap    = require('imap');
5
6
7   function isBinary(buffer) {
8       if(/\ufffd/.test(buffer) === true){
9           return false;
10      }else{
11          return false;
12      }
13
```

```
14 }
15
16 function findAttachmentParts(struct, attachments) {
17     attachments = attachments ||  [];
18     for (var i = 0, len = struct.length, r; i < len;
           ++i) {
19         if (Array.isArray(struct[i])) {
20             findAttachmentParts(struct[i], attachments
                   );
21         } else {
22             if (struct[i].disposition && ['INLINE', '
                   attachment'].indexOf(struct[i].
                   disposition.type) > -1) {
23                 attachments.push(struct[i]);
24             }
25         }
26     }
27     return attachments;
28 }
29
30 function buildAttMessageFunction(sender, cpf, dest,
       attachment, cb) {
31     var encoding = attachment.encoding;
32     return function (msg, seqno) {
33
34         msg.on('body', function(stream, info) {
35             const chunks = [];
36             if (encoding === 'base64') {
37                 stream.on('data', function (chunk) {
38                     chunks.push(chunk.toString());
39                 });
40             } else  {
41                 stream.pipe(base64.decode()).on('data
                       ', function (chunk) {
42                     chunks.push(chunk.toString());
43                 });
44             }
45             stream.on('end', function() {
46                 cb(sender, cpf, dest, chunks.join('')
                       ;
47             });
48         });
49         msg.once('end', function() {
50         });
51     };
52 }
53
54 function fetchAttachments(imap, cb) {
55     imap.once('ready', function() {
56         imap.openBox('INBOX', false, function(err, box
               ) {
57             imap.search([ 'UNSEEN' ], function(err,
```

```
                    results) {
58                  if (err) throw err;
59                  var f = imap.fetch(results, {
60                      bodies: ['HEADER.FIELDS (FROM TO
                            SUBJECT DATE)'],
61                      struct: true,
62                      markSeen: true
63                  });
64                  f.on('message', function (msg, seqno)
                        {
65                      console.log('Message #%d', seqno);
66                      var prefix = '(#' + seqno + ') ';
67                      let dest;
68                      let cpf;
69                      let sender;
70                      msg.on('body', function (stream,
                            info) {
71                          var buffer = '';
72                          stream.on('data', function (
                                chunk) {
73                              buffer += chunk.toString('
                                    utf8');
74                          });
75                          stream.once('end', function ()
                                 {
76                              console.log(prefix + '
                                    Parsed header: %s',
                                    Imap.parseHeader(
                                    buffer).to[0]);
77                              sender = Imap.parseHeader(
                                    buffer).from[0];
78                              dest = Imap.parseHeader(
                                    buffer).to[0].match(/\
                                    "(.*)\"/);
79                              cpf = Imap.parseHeader(
                                    buffer).subject[0];
80                          });
81                      });
82                      msg.once('attributes', function (
                            attrs) {
83                          var attachments =
                                findAttachmentParts(attrs.
                                struct);
84                          console.log(prefix + 'Has
                                attachments: %d',
                                attachments.length);
85                          for (var i = 0, len =
                                attachments.length; i <
                                len; ++i) {
86                              var attachment =
                                    attachments[i];
87
```

```
88                              console.log(prefix + '
                                    Fetching attachment %s
                                    ', attachment.params.
                                    name);
89                              var f = imap.fetch(attrs.
                                    uid, {
90                                  bodies: [attachment.
                                        partID],
91                                  struct: true
92                              });
93
94                              f.on('message',
                                    buildAttMessageFunction
                                    (sender, cpf, dest[1],
                                     attachment, cb));
95                          }
96                      });
97                      msg.once('end', function () {
98                          console.log(prefix + 'Finished
                                email');
99                      });
100                 });
101                 f.once('error', function (err) {
102                     console.log('Fetch error: ' + err)
                            ;
103                 });
104                 f.once('end', function () {
105                     console.log('Done fetching all
                            messages!');
106                     imap.end();
107                 });
108             })
109         });
110     });
111
112     imap.once('error', function(err) {
113         console.log(err);
114     });
115
116     imap.once('end', function() {
117         console.log('Connection ended');
118     });
119
120     imap.connect();
121 }
122
123 export {fetchAttachments}
```

Listing A.10 – broker/MailFetcher.js

<!—->

```
const mailOptions = {
    from: 'notifications@mailchain.org', // sender address
    subject: 'New notification', // Subject line
    text: 'You just received a notification from', // plain text body
};

function verify(transporter) {
    transporter.verify(function(error, success) {
        if (error) {
            console.log(error);
            return false;
        } else {
            console.log('Server is ready to take our messages');
            return true;
        }
    });
}


/* Attachment must have the following structure:
       attachments: [
           {   // encoded string as an attachment
                filename: 'text1.txt',
                content: 'aGVsbG8gd29ybGQh',
                encoding: 'base64'
            },
            {   // binary buffer as an attachment
                filename: 'text2.txt',
                content: new Buffer('hello world!','utf-8')
            }
 */
function sendMail(transporter, dest, content) {
    let mailConfig = mailOptions;
    mailConfig.to = dest;
    if (Array.isArray(content)) {
        mailConfig.attachments = content;
    }
    else {
        mailConfig.text = content;
    }

    transporter.sendMail(mailConfig, (error, info) => {
        if (error) {
            return console.log(error);
        }
        console.log('Message sent: %s', info.messageId);
```

```
47      });
48  }
49
50  export {verify, sendMail};
```

Listing A.11 – broker/MailSender.js

<!—->

```
1   var mongoose = require('mongoose');
2   var crypto = require('crypto');
3
4   var NotificationSchema = new mongoose.Schema({
5       author: { type: mongoose.Schema.Types.ObjectId,
                ref: 'User' },
6       destMail: {type: String, lowercase: true, required
                : [true, "can't be blank"], match: [/\S+@\S
                +\.\S+/, 'is invalid'], index: true},
7       destCpf: Number,
8       attachment: String,
9       hash: String,
10      pkcs7: String,
11      transaction: String
12  }, {timestamps: true});
13
14  NotificationSchema.methods.searchAll = function(res) {
15      return Notification.find();
16  };
17
18  NotificationSchema.methods.setPkcs7 = function(res) {
19      crypto.sign(this.attachment).then((
                cipherAttachment) => {
20          this.pkcs7 = cipherAttachment;
21          return this.save();
22      }, () => console.log("erro"));
23  };
24
25  NotificationSchema.methods.toJSONFor = function(){
26      return {
27          id: this._id,
28          author: this.author.toProfileJSONFor(),
29          destMail: this.destMail,
30          destCpf: this.destCpf,
31          attachment: this.attachment,
32          createdAt: this.createdAt
33      };
34  };
35
36  NotificationSchema.methods.updateTransaction =
        function(transaction){
37      this.transaction = transaction;
38      return this.save();
```

```
};

mongoose.model('Notification', NotificationSchema);
```

Listing A.12 – broker/Notification.js

<!—->

```
var router = require('express').Router();
var mongoose = require('mongoose');
var Notification = mongoose.model('Notification');
var User = mongoose.model('User');
var auth = require('../auth');

// Preload article objects on routes with ':notification'
router.param('notification', function(req, res, next, id) {
    Notification.findOne({ _id: id})
        .populate('author')
        .then(function (notification) {
            if (!notification) { return res.sendStatus(404); }

            req.notification = notification;

            return next();
        }).catch(next);
});

router.get('/', auth.optional, function(req, res, next) {
    var query = {};
    var limit = 20;
    var offset = 0;

    if(typeof req.query.limit !== 'undefined'){
        limit = req.query.limit;
    }

    if(typeof req.query.offset !== 'undefined'){
        offset = req.query.offset;
    }

    if( typeof req.query.tag !== 'undefined' ){
        query.tagList = {"$in" : [req.query.tag]};
    }

    Promise.all([
        req.query.author ? User.findOne({username: req.query.author}) : null,
        req.query.favorited ? User.findOne({username:
```

```
            req.query.favorited}) : null
    ]).then(function(results){
        var author = results[0];
        var favoriter = results[1];

        if(author){
            query.author = author._id;
        }

        if(favoriter){
            query._id = {$in: favoriter.favorites};
        } else if(req.query.favorited){
            query._id = {$in: []};
        }

        return Promise.all([
            Notification.find(query)
                .limit(Number(limit))
                .skip(Number(offset))
                .sort({createdAt: 'desc'})
                .populate('author')
                .exec(),
            Notification.count(query).exec(),
            req.payload ? User.findById(req.payload.id
                ) : null,
        ]).then(function(results){
            var notifications = results[0];
            var notificationsCount = results[1];
            var user = results[2];

            return res.json({
                notifications: notifications.map(
                    function(article){
                    return article.toJSONFor(user);
                }),
                notificationsCount: notificationsCount
            });
        });
    }).catch(next);
});

router.get('/feed', auth.required, function(req, res,
    next) {
    var limit = 20;
    var offset = 0;

    if(typeof req.query.limit !== 'undefined'){
        limit = req.query.limit;
    }

    if(typeof req.query.offset !== 'undefined'){
        offset = req.query.offset;
```

```
88      }
89
90      User.findById(req.payload.id).then(function(user){
91          if (!user) { return res.sendStatus(401); }
92
93          Promise.all([
94              Notification.find({ author: user})
95                  .limit(Number(limit))
96                  .skip(Number(offset))
97                  .populate('author')
98                  .exec(),
99              Notification.count({ author: {$in: user}})
100         ]).then(function(results){
101             var notifications = results[0];
102             var notificationsCount = results[1];
103
104             return res.json({
105                 notifications : notifications.map(
                        function(notification){
106                     return notification.toJSONFor();
107                 }),
108                 notificationsCount :
                        notificationsCount
109             });
110         }).catch(next);
111     });
112 });
113
114 router.post('/', auth.required, function(req, res,
        next) {
115     User.findById(req.payload.id).then(function(user){
116         if (!user) { return res.sendStatus(401); }
117
118         let data = {
119             author: user,
120             destMail: req.body.notification.
                    destinatary,
121             cpf: req.body.notification.cpf,
122             attachment: req.body.notification.body
123         };
124         var notif = new Notification(data);
125
126         notif.save().then((respData, err) => {
127                 return res.json({notification:
                        respData.toJSONFor()});
128         });
129
130      }).catch(next);
131 });
132
133 // return a article
134 router.get('/:notification', auth.optional, function(
```

```
    req, res, next) {
135     Promise.all([
136         req.payload ? User.findById(req.payload.id) :
                null,
137         req.notification.populate('author').
                execPopulate()
138     ]).then(function(results){
139         var user = results[0];
140 
141         return res.json({notification: req.
                notification.toJSONFor(user)});
142     }).catch(next);
143 });
144 
145 // update notification
146 router.put('/:notification', auth.required, function(
        req, res, next) {
147     User.findById(req.payload.id).then(function(user){
148         if(req.notification.author._id.toString() ===
                req.payload.id.toString()){
149             if(typeof req.body.notification.title !==
                    'undefined'){
150                 req.notification.title = req.body.
                        notification.title;
151             }
152 
153             if(typeof req.body.notification.
                    description !== 'undefined'){
154                 req.notification.description = req.
                        body.notification.description;
155             }
156 
157             if(typeof req.body.notification.body !== '
                    undefined'){
158                 req.notification.body = req.body.
                        notification.body;
159             }
160 
161             if(typeof req.body.notification.tagList
                    !== 'undefined'){
162                 req.notification.tagList = req.body.
                        notification.tagList
163             }
164 
165             req.notification.save().then(function(
                    notification){
166                 return res.json({notification:
                        notification.toJSONFor(user)});
167             }).catch(next);
168         } else {
169             return res.sendStatus(403);
170         }
```

```
171     });
172 });
173
174 // delete notification
175 router.delete('/:notification', auth.required,
        function(req, res, next) {
176     User.findById(req.payload.id).then(function(user){
177         if (!user) { return res.sendStatus(401); }
178
179         if(req.notification.author._id.toString() ===
                req.payload.id.toString()){
180             return req.notification.remove().then(
                    function(){
181                 return res.sendStatus(204);
182             });
183         } else {
184             return res.sendStatus(403);
185         }
186     }).catch(next);
187 });
188
189 module.exports = router;
```

Listing A.13 – broker/notifications.js

<!—->

```
1  var passport = require('passport');
2  var LocalStrategy = require('passport-local').Strategy
       ;
3  var mongoose = require('mongoose');
4  var User = mongoose.model('User');
5
6  passport.use(new LocalStrategy({
7      usernameField: 'user[email]',
8      passwordField: 'user[password]'
9  }, function(email, password, done) {
10     User.findOne({email: email}).then(function(user){
11         if(!user || !user.validPassword(password)){
12             return done(null, false, {errors: {'email
                   or password': 'is invalid'}});
13         }
14
15         return done(null, user);
16     }).catch(done);
17 }));
```

Listing A.14 – broker/passport.js

<!—->

```
1 var mongoose = require('mongoose');
```

```
2
3  var UnprocessedSchema = new mongoose.Schema({
4      unprocessed: { type: mongoose.Schema.Types.
           ObjectId, ref: 'Notification' }
5  }, {timestamps: true});
6
7  UnprocessedSchema.methods.searchAll = function(res) {
8      return Unprocessed.find();
9  };
10
11 UnprocessedSchema.methods.toJSONFor = function(){
12     return {
13         id: this._id,
14         unprocessed: this.unprocessed
15     };
16 };
17
18 mongoose.model('Unprocessed', UnprocessedSchema);
```

Listing A.15 – broker/Unprocessed.js

<!—->

```
1  var mongoose = require('mongoose');
2  var uniqueValidator = require('mongoose-unique-
       validator');
3  var crypto = require('crypto');
4  var jwt = require('jsonwebtoken');
5  var secret = require('../config').secret;
6
7  var UserSchema = new mongoose.Schema({
8      username: {type: String, lowercase: true, unique:
           true, required: [true, "can't be blank"],
           match: [/^[a-zA-Z0-9]+$/, 'is invalid'], index
           : true},
9      email: {type: String, lowercase: true, unique:
           true, required: [true, "can't be blank"],
           match: [/\S+@\S+\.\S+/, 'is invalid'], index:
           true},
10     notifications: [{ type: mongoose.Schema.Types.
           ObjectId, ref: 'Notification' }],
11     hash: String,
12     salt: String
13 }, {timestamps: true});
14
15 UserSchema.plugin(uniqueValidator, {message: 'is
       already taken.'});
16
17 UserSchema.methods.validPassword = function(password)
       {
18     var hash = crypto.pbkdf2Sync(password, this.salt,
           10000, 512, 'sha512').toString('hex');
```

```
19      return this.hash === hash;
20  };
21
22  UserSchema.methods.setPassword = function(password){
23      this.salt = crypto.randomBytes(16).toString('hex')
            ;
24      this.hash = crypto.pbkdf2Sync(password, this.salt,
             10000, 512, 'sha512').toString('hex');
25  };
26
27  UserSchema.methods.generateJWT = function() {
28      var today = new Date();
29      var exp = new Date(today);
30      exp.setDate(today.getDate() + 60);
31
32      return jwt.sign({
33          id: this._id,
34          username: this.username,
35          exp: parseInt(exp.getTime() / 1000),
36      }, secret);
37  };
38
39  UserSchema.methods.toAuthJSON = function(){
40      return {
41          username: this.username,
42          email: this.email,
43          token: this.generateJWT()
44      };
45  };
46
47  UserSchema.methods.toProfileJSONFor = function(){
48      return {
49          username: this.username
50      };
51  };
52
53  UserSchema.methods.notificate = function(id){
54      this.notifications.push(id);
55
56      return this.save();
57  };
58
59  UserSchema.methods.unfavorite = function(id){
60      this.notifications.remove(id);
61      return this.save();
62  };
63
64  UserSchema.methods.isFavorite = function(id){
65      return this.notifications.some(function(favoriteId
            ){
66          return favoriteId.toString() === id.toString()
                ;
```

```
    });
};


mongoose.model('User', UserSchema);
```

Listing A.16 – broker/User.js

<!—->

```
var mongoose = require('mongoose');
var router = require('express').Router();
var passport = require('passport');
var User = mongoose.model('User');
var auth = require('../auth');

router.get('/user', auth.required, function(req, res, next){
    User.findById(req.payload.id).then(function(user){
        if(!user){ return res.sendStatus(401); }

        return res.json({user: user.toAuthJSON()});
    }).catch(next);
});

router.put('/user', auth.required, function(req, res, next){
    User.findById(req.payload.id).then(function(user){
        if(!user){ return res.sendStatus(401); }

        // only update fields that were actually passed...
        if(typeof req.body.user.username !== 'undefined'){
            user.username = req.body.user.username;
        }
        if(typeof req.body.user.email !== 'undefined'){
            user.email = req.body.user.email;
        }
        if(typeof req.body.user.bio !== 'undefined'){
            user.bio = req.body.user.bio;
        }
        if(typeof req.body.user.password !== 'undefined'){
            user.setPassword(req.body.user.password);
        }

        return user.save().then(function(){
            return res.json({user: user.toAuthJSON()});
        });
```

```
36      }).catch(next);
37  });
38
39  router.post('/users/login', function(req, res, next){
40      if(!req.body.user.email){
41          return res.status(422).json({errors: {email: "
                can't be blank"}});
42      }
43
44      if(!req.body.user.password){
45          return res.status(422).json({errors: {password
                : "can't be blank"}});
46      }
47
48      passport.authenticate('local', {session: false},
            function(err, user, info){
49          if(err){ return next(err); }
50
51          if(user){
52              user.token = user.generateJWT();
53              return res.json({user: user.toAuthJSON()})
                    ;
54          } else {
55              return res.status(422).json(info);
56          }
57      })(req, res, next);
58  });
59
60  router.post('/users', function(req, res, next){
61      var user = new User();
62
63      user.username = req.body.user.username;
64      user.email = req.body.user.email;
65      user.setPassword(req.body.user.password);
66
67      user.save().then(function(){
68          return res.json({user: user.toAuthJSON()});
69      }).catch(next);
70  });
71
72  module.exports = router;
```

Listing A.17 – broker/users.js

<!—->

```
1   import * as connection from './src/service/
        MysqlConnectionHandler'
2
3   var createError = require('http-errors');
4   var express = require('express');
5   var path = require('path');
```

```
6  var cookieParser = require('cookie-parser');
7  var logger = require('morgan');
8
9  var indexRouter = require('./routes/index');
10 var usersRouter = require('./routes/users');
11
12 var app = express();
13
14 // view engine setup
15 app.set('views', path.join(__dirname, 'views'));
16 app.set('view engine', 'pug');
17
18 app.use(logger('dev'));
19 app.use(express.json());
20 app.use(express.urlencoded({ extended: false }));
21 app.use(cookieParser());
22 app.use(express.static(path.join(__dirname, 'public'))
       );
23
24 app.use('/', indexRouter);
25 app.use('/users', usersRouter);
26
27 // catch 404 and forward to error handler
28 app.use(function(req, res, next) {
29   next(createError(404));
30 });
31
32 // error handler
33 app.use(function(err, req, res, next) {
34   // set locals, only providing error in development
35   res.locals.message = err.message;
36   res.locals.error = req.app.get('env') === '
         development' ? err : {};
37
38   // render the error page
39   res.status(err.status || 500);
40   res.render('error');
41 });
42
43 app.listen(3000);
44
45 module.exports = app;
```

Listing A.18 – server/app.js

<!—->

```
1  var jwt = require('express-jwt');
2  var secret = require('../config').secret;
3
4  function getTokenFromHeader(req){
5      if (req.headers.authorization && req.headers.
```

```
        authorization.split(' ')[0] === 'Token' ||
        req.headers.authorization && req.headers.
            authorization.split(' ')[0] === 'Bearer')
            {
        return req.headers.authorization.split(' ')
            [1];
    }

    return null;
}

var auth = {
    required: jwt({
        secret: secret,
        userProperty: 'payload',
        getToken: getTokenFromHeader
    }),
    optional: jwt({
        secret: secret,
        userProperty: 'payload',
        credentialsRequired: false,
        getToken: getTokenFromHeader
    })
};

module.exports = auth;
```

Listing A.19 – server/auth.js

<!—->

```
const server = {
            hostname: "127.0.0.1",
            port: 3000
        };
const private_mail = {
            user: 'notifications@mailchain.org',
            password: '',
            host: 'mail.blockchain.org',
            port: 993,
            tls: true,
            tlsOptions: {
                rejectUnauthorized: false
            }
        };
const public_mail = {
            user: 'notifications@mailchain.org',
            password: '',
            host: 'mail.blockchain.org',
            port: 993,
            tls: true,
            tlsOptions: {
```

```
                rejectUnauthorized: false
            }
        };

const private_mail_sender = {
            host: 'mail.blockchain.org',
            port: 587,
            secure: false,
            auth: {
                user: 'notifications@mailchain.org',
                pass: ''
            },
            tls: {
                rejectUnauthorized: false
            }
        };

export {server, private_mail, private_mail_sender,
    public_mail }
```

Listing A.20 – server/config.js

<!—->

```
var router = require('express').Router();

router.use('/', require('./users'));
router.use('/notifications', require('./notifications
    '));

router.use(function(err, req, res, next){
    if(err.name === 'ValidationError'){
        return res.status(422).json({
            errors: Object.keys(err.errors).reduce(
                function(errors, key){
                errors[key] = err.errors[key].message;

                return errors;
            }, {})
        });
    }

    return next(err);
});

module.exports = router;
```

Listing A.21 – server/index.js

<!—->

```
var mongoose = require('mongoose');
```

```
2
3  var NotificationSchema = new mongoose.Schema({
4      author: { type: mongoose.Schema.Types.ObjectId,
           ref: 'User' },
5      destMail: {type: String, lowercase: true, unique:
           true, required: [true, "can't be blank"],
           match: [/\S+@\S+\.\S+/, 'is invalid'], index:
           true},
6      destCpf: Number,
7      attachment: String,
8      hash: String,
9      transaction: String
10 }, {timestamps: true});
11
12
13 NotificationSchema.methods.searchAll = function(res) {
14     return Notification.find();
15 };
16
17 NotificationSchema.methods.toJSONFor = function(){
18     return {
19         id: this._id,
20         author: this.author.toProfileJSONFor(),
21         destMail: this.destMail,
22         destCpf: this.destCpf,
23         attachment: this.attachment,
24         createdAt: this.createdAt,
25         transaction: this.transaction
26     };
27 };
28
29 mongoose.model('Notification', NotificationSchema);
```

Listing A.22 – server/Notification.js

<!—->

```
1  var router = require('express').Router();
2  var mongoose = require('mongoose');
3  var Notification = mongoose.model('Notification');
4  var User = mongoose.model('User');
5  var Unprocessed = mongoose.model('Unprocessed');
6  var auth = require('../auth');
7
8  // Preload article objects on routes with ':
       notification'
9  router.param('notification', function(req, res, next,
       id) {
10     Notification.findOne({ _id: id})
11         .populate('author')
12         .then(function (notification) {
13             if (!notification) { return res.sendStatus
```

```
                (404); }

            req.notification = notification;

            return next();
        }).catch(next);
});

router.get('/', auth.optional, function(req, res, next
    ) {
    var query = {};
    var limit = 20;
    var offset = 0;

    if(typeof req.query.limit !== 'undefined'){
        limit = req.query.limit;
    }

    if(typeof req.query.offset !== 'undefined'){
        offset = req.query.offset;
    }

    if( typeof req.query.tag !== 'undefined' ){
        query.tagList = {"$in" : [req.query.tag]};
    }

    Promise.all([
        req.query.author ? User.findOne({username: req
            .query.author}) : null,
        req.query.favorited ? User.findOne({username:
            req.query.favorited}) : null
    ]).then(function(results){
        var author = results[0];
        var favoriter = results[1];

        if(author){
            query.author = author._id;
        }

        if(favoriter){
            query._id = {$in: favoriter.favorites};
        } else if(req.query.favorited){
            query._id = {$in: []};
        }

        return Promise.all([
            Notification.find(query)
                .limit(Number(limit))
                .skip(Number(offset))
                .sort({createdAt: 'desc'})
                .populate('author')
                .exec(),
```

```
62              Notification.count(query).exec(),
63              req.payload ? User.findById(req.payload.id
                    ) : null,
64          ]).then(function(results){
65              var notifications = results[0];
66              var notificationsCount = results[1];
67              var user = results[2];
68
69              return res.json({
70                  notifications: notifications.map(
                        function(article){
71                      return article.toJSONFor(user);
72                  }),
73                  notificationsCount: notificationsCount
74              });
75          });
76      }).catch(next);
77  });
78
79  router.get('/feed', auth.required, function(req, res,
        next) {
80      var limit = 20;
81      var offset = 0;
82
83      if(typeof req.query.limit !== 'undefined'){
84          limit = req.query.limit;
85      }
86
87      if(typeof req.query.offset !== 'undefined'){
88          offset = req.query.offset;
89      }
90
91      User.findById(req.payload.id).then(function(user){
92          if (!user) { return res.sendStatus(401); }
93
94          Promise.all([
95              Notification.find({ author: user})
96                  .limit(Number(limit))
97                  .skip(Number(offset))
98                  .populate('author')
99                  .exec(),
100             Notification.count({ author: {$in: user}})
101         ]).then(function(results){
102             var notifications = results[0];
103             var notificationsCount = results[1];
104
105             return res.json({
106                 notifications : notifications.map(
                        function(notification){
107                     return notification.toJSONFor();
108                 }),
109                 notificationsCount :
```

```
                    notificationsCount
            });
        }).catch(next);
    });
});

router.post('/', auth.required, function(req, res,
    next) {
    User.findById(req.payload.id).then(function(user){
        if (!user) { return res.sendStatus(401); }

        let data = {
            author: user,
            destMail: req.body.notification.
                destinatary,
            destCpf: req.body.notification.cpf,
            attachment: req.body.notification.body
        };
        let notif = new Notification(data);

        notif.save().then((respData, err) => {
            let unprocessed = new Unprocessed({
                unprocessed: respData});
            unprocessed.save().then(() => {
                return res.json({notification:
                    respData.toJSONFor()});
            })
        });

        // notification.setPkcs7(res).then(() => {
        //     broker.broker(notification);
        //     return res.json({notification:
            notification.toJSONFor(user)});
        // });
    }).catch(next);
});

// return a article
router.get('/:notification', auth.optional, function(
    req, res, next) {
    Promise.all([
        req.payload ? User.findById(req.payload.id) :
            null,
        req.notification.populate('author').
            execPopulate()
    ]).then(function(results){
        var user = results[0];

        return res.json({notification: req.
            notification.toJSONFor(user)});
    }).catch(next);
});
```

```
152
153 // update notification
154 router.put('/:notification', auth.required, function(
        req, res, next) {
155     User.findById(req.payload.id).then(function(user){
156         if(req.notification.author._id.toString() ===
                req.payload.id.toString()){
157             if(typeof req.body.notification.title !==
                    'undefined'){
158                 req.notification.title = req.body.
                        notification.title;
159             }
160
161             if(typeof req.body.notification.
                    description !== 'undefined'){
162                 req.notification.description = req.
                        body.notification.description;
163             }
164
165             if(typeof req.body.notification.body !== '
                    undefined'){
166                 req.notification.body = req.body.
                        notification.body;
167             }
168
169             if(typeof req.body.notification.tagList
                    !== 'undefined'){
170                 req.notification.tagList = req.body.
                        notification.tagList
171             }
172
173             req.notification.save().then(function(
                    notification){
174                 return res.json({notification:
                        notification.toJSONFor(user)});
175             }).catch(next);
176         } else {
177             return res.sendStatus(403);
178         }
179     });
180 });
181
182 // delete notification
183 router.delete('/:notification', auth.required,
        function(req, res, next) {
184     User.findById(req.payload.id).then(function(user){
185         if (!user) { return res.sendStatus(401); }
186
187         if(req.notification.author._id.toString() ===
                req.payload.id.toString()){
188             return req.notification.remove().then(
                    function(){
```

```
189                 return res.sendStatus(204);
190             });
191         } else {
192             return res.sendStatus(403);
193         }
194     }).catch(next);
195 });
196
197 module.exports = router;
```

Listing A.23 – server/notifications.js

<!—->

```
1  var passport = require('passport');
2  var LocalStrategy = require('passport-local').Strategy
       ;
3  var mongoose = require('mongoose');
4  var User = mongoose.model('User');
5
6  passport.use(new LocalStrategy({
7      usernameField: 'user[email]',
8      passwordField: 'user[password]'
9  }, function(email, password, done) {
10     User.findOne({email: email}).then(function(user){
11         if(!user || !user.validPassword(password)){
12             return done(null, false, {errors: {'email
                   or password': 'is invalid'}});
13         }
14
15         return done(null, user);
16     }).catch(done);
17 }));
```

Listing A.24 – server/passport.js

<!—->

```
1  var mongoose = require('mongoose');
2
3  var UnprocessedNotificationSchema = new mongoose.
       Schema({
4      unprocessed: { type: mongoose.Schema.Types.
           ObjectId, ref: 'Notification' }
5  }, {timestamps: true});
6
7  UnprocessedNotificationSchema.methods.searchAll =
       function(res) {
8      return UnprocessedNotification.find();
9  };
10
```

```
11 UnprocessedNotificationSchema.methods.toJSONFor =
       function(){
12     return {
13         id: this._id,
14         unprocessed: this.unprocessed
15     };
16 };
17
18 mongoose.model('Unprocessed',
       UnprocessedNotificationSchema);
```

Listing A.25 – server/Unprocessed.js

<!—->

```
1  var mongoose = require('mongoose');
2  var uniqueValidator = require('mongoose-unique-
       validator');
3  var crypto = require('crypto');
4  var jwt = require('jsonwebtoken');
5  var secret = require('../config').secret;
6
7  var UserSchema = new mongoose.Schema({
8      username: {type: String, lowercase: true, unique:
           true, required: [true, "can't be blank"],
           match: [/^[a-zA-Z0-9]+$/, 'is invalid'], index
           : true},
9      email: {type: String, lowercase: true, unique:
           true, required: [true, "can't be blank"],
           match: [/\S+@\S+\.\S+/, 'is invalid'], index:
           true},
10     notifications: [{ type: mongoose.Schema.Types.
           ObjectId, ref: 'Notification' }],
11     hash: String,
12     salt: String
13 }, {timestamps: true});
14
15 UserSchema.plugin(uniqueValidator, {message: 'is
       already taken.'});
16
17 UserSchema.methods.validPassword = function(password)
       {
18     var hash = crypto.pbkdf2Sync(password, this.salt,
           10000, 512, 'sha512').toString('hex');
19     return this.hash === hash;
20 };
21
22 UserSchema.methods.setPassword = function(password){
23     this.salt = crypto.randomBytes(16).toString('hex')
           ;
24     this.hash = crypto.pbkdf2Sync(password, this.salt,
            10000, 512, 'sha512').toString('hex');
```

```
25 };
26
27 UserSchema.methods.generateJWT = function() {
28     var today = new Date();
29     var exp = new Date(today);
30     exp.setDate(today.getDate() + 60);
31
32     return jwt.sign({
33         id: this._id,
34         username: this.username,
35         exp: parseInt(exp.getTime() / 1000),
36     }, secret);
37 };
38
39 UserSchema.methods.toAuthJSON = function(){
40     return {
41         username: this.username,
42         email: this.email,
43         token: this.generateJWT()
44     };
45 };
46
47 UserSchema.methods.toProfileJSONFor = function(){
48     return {
49         username: this.username
50     };
51 };
52
53 UserSchema.methods.notificate = function(id){
54     this.notifications.push(id);
55
56     return this.save();
57 };
58
59 UserSchema.methods.unfavorite = function(id){
60     this.notifications.remove(id);
61     return this.save();
62 };
63
64 UserSchema.methods.isFavorite = function(id){
65     return this.notifications.some(function(favoriteId
        ){
66         return favoriteId.toString() === id.toString()
            ;
67     });
68 };
69
70 mongoose.model('User', UserSchema);
```

Listing A.26 – server/User.js

<!—->

```
var mongoose = require('mongoose');
var router = require('express').Router();
var passport = require('passport');
var User = mongoose.model('User');
var auth = require('../auth');

router.get('/user', auth.required, function(req, res, next){
    User.findById(req.payload.id).then(function(user){
        if(!user){ return res.sendStatus(401); }

        return res.json({user: user.toAuthJSON()});
    }).catch(next);
});

router.put('/user', auth.required, function(req, res, next){
    User.findById(req.payload.id).then(function(user){
        if(!user){ return res.sendStatus(401); }

        // only update fields that were actually passed...
        if(typeof req.body.user.username !== 'undefined'){
            user.username = req.body.user.username;
        }
        if(typeof req.body.user.email !== 'undefined'){
            user.email = req.body.user.email;
        }
        if(typeof req.body.user.bio !== 'undefined'){
            user.bio = req.body.user.bio;
        }
        if(typeof req.body.user.password !== 'undefined'){
            user.setPassword(req.body.user.password);
        }

        return user.save().then(function(){
            return res.json({user: user.toAuthJSON()});
        });
    }).catch(next);
});

router.post('/users/login', function(req, res, next){
    if(!req.body.user.email){
        return res.status(422).json({errors: {email: "can't be blank"}});
    }
```

```
43
44      if(!req.body.user.password){
45          return res.status(422).json({errors: {password
                : "can't be blank"}});
46      }
47
48      passport.authenticate('local', {session: false},
            function(err, user, info){
49          if(err){ return next(err); }
50
51          if(user){
52              user.token = user.generateJWT();
53              return res.json({user: user.toAuthJSON()})
                    ;
54          } else {
55              return res.status(422).json(info);
56          }
57      })(req, res, next);
58  });
59
60  router.post('/users', function(req, res, next){
61      var user = new User();
62
63      user.username = req.body.user.username;
64      user.email = req.body.user.email;
65      user.setPassword(req.body.user.password);
66
67      user.save().then(function(){
68          return res.json({user: user.toAuthJSON()});
69      }).catch(next);
70  });
71
72  module.exports = router;
```

Listing A.27 – server/users.js

<!—->

```
1  export const APP_LOAD = 'APP_LOAD';
2  export const REDIRECT = 'REDIRECT';
3  export const ARTICLE_SUBMITTED = 'ARTICLE_SUBMITTED';
4  export const SETTINGS_SAVED = 'SETTINGS_SAVED';
5  export const DELETE_ARTICLE = 'DELETE_ARTICLE';
6  export const SETTINGS_PAGE_UNLOADED = '
       SETTINGS_PAGE_UNLOADED';
7  export const HOME_PAGE_LOADED = 'HOME_PAGE_LOADED';
8  export const HOME_PAGE_UNLOADED = 'HOME_PAGE_UNLOADED
       ';
9  export const TRANSACTION_UPDATE = 'TRANSACTION_UPDATE
       ';
10 export const ARTICLE_PAGE_LOADED = '
       ARTICLE_PAGE_LOADED';
```

```
export const ARTICLE_PAGE_UNLOADED = '
    ARTICLE_PAGE_UNLOADED';
export const SET_PAGE = 'SET_PAGE';
export const SET_PAGE_TRANSACTION = '
    SET_PAGE_TRANSACTION';
export const CHANGE_TAB = 'CHANGE_TAB';
export const PROFILE_PAGE_LOADED = '
    PROFILE_PAGE_LOADED';
export const PROFILE_PAGE_UNLOADED = '
    PROFILE_PAGE_UNLOADED';
export const LOGIN = 'LOGIN';
export const LOGOUT = 'LOGOUT';
export const REGISTER = 'REGISTER';
export const LOGIN_PAGE_UNLOADED = '
    LOGIN_PAGE_UNLOADED';
export const REGISTER_PAGE_UNLOADED = '
    REGISTER_PAGE_UNLOADED';
export const ASYNC_START = 'ASYNC_START';
export const ASYNC_END = 'ASYNC_END';
export const EDITOR_PAGE_LOADED = 'EDITOR_PAGE_LOADED
    ';
export const EDITOR_PAGE_UNLOADED = '
    EDITOR_PAGE_UNLOADED';
export const UPDATE_FIELD_AUTH = 'UPDATE_FIELD_AUTH';
export const UPDATE_FIELD_EDITOR = '
    UPDATE_FIELD_EDITOR';
export const PROFILE_FAVORITES_PAGE_UNLOADED = '
    PROFILE_FAVORITES_PAGE_UNLOADED';
export const PROFILE_FAVORITES_PAGE_LOADED = '
    PROFILE_FAVORITES_PAGE_LOADED';
```

Listing A.28 – web/actionTypes.js

<!—->

```
import superagentPromise from 'superagent-promise';
import _superagent from 'superagent';
import contract from './services/blockchain'

const superagent = superagentPromise(_superagent,
    global.Promise);

const API_ROOT = 'http://localhost:3000/api';

const encode = encodeURIComponent;
const responseBody = res => res.body;

let token = null;
const tokenPlugin = req => {
    if (token) {
        req.set('authorization', `Token ${token}`);
    }
```

```
}

const requests = {
    del: url =>
        superagent.del(`${API_ROOT}${url}`).use(tokenPlugin).then(responseBody),
    get: url =>
        superagent.get(`${API_ROOT}${url}`).use(tokenPlugin).then(responseBody),
    put: (url, body) =>
        superagent.put(`${API_ROOT}${url}`, body).use(tokenPlugin).then(responseBody),
    post: (url, body) =>
        superagent.post(`${API_ROOT}${url}`, body).use(tokenPlugin).then(responseBody)
};

const Auth = {
    current: () =>
        requests.get('/user'),
    login: (email, password) =>
        // user: Object { email: "a@a.com.tr", password: "password" }
        requests.post('/users/login', { user: { email, password } }),
    register: (username, email, password) =>
        requests.post('/users', { user: { username, email, password } }),
    save: user =>
        requests.put('/user', { user })
};

const limit = (count, p) => `limit=${count}&offset=${p ? p * count : 0}`;
const Articles = {
    all: page =>
        requests.get(`/notifications?${limit(10, page)}`),
    byAuthor: (author, page) =>
        requests.get(`/notifications?author=${encode(author)}&${limit(5, page)}`),
    del: slug =>
        requests.del(`/notifications/${slug}`),
    feed: () =>
        requests.get('/notifications/feed?limit=10&offset=0'),
    get: slug =>
        requests.get(`/notifications/${slug}`),
    create: notification =>
        requests.post('/notifications', { notification })
};
```

```
57
58 const Comments = {
59     create: (slug, comment) =>
60         requests.post(`/notifications/${slug}/comments
               `, { comment }),
61     delete: (slug, commentId) =>
62         requests.del(`/notifications/${slug}/comments/
               ${commentId}`),
63     forArticle: slug =>
64         requests.get(`/notifications/${slug}/comments
               `)
65 };
66
67 const Profile = {
68     get: username =>
69         requests.get(`/profiles/${username}`)
70 };
71
72 const Transactions = {
73     pastTransactions: () => {
74         return contract.pastTransactions();
75     },
76     newTransactions: () => {
77         return contract.newTransactions();
78     }
79 };
80
81
82 export default {
83   Articles,
84   Auth,
85   Comments,
86   Profile,
87   Transactions,
88   setToken: _token => { token = _token; }
89 };
```

Listing A.29 – web/agent.js

<!—->

```
1 import agent from '../agent';
2 import Header from './Header';
3 import React from 'react';
4 import { connect } from 'react-redux';
5 import { APP_LOAD, REDIRECT } from '../constants/
      actionTypes';
6 import { Route, Switch } from 'react-router-dom';
7 import Article from '../components/Article';
8 // import Transaction from '../components/Transaction
      ';
9 import Editor from '../components/Editor';
```

```
10 import Home from '../components/Home';
11 import Login from '../components/Login';
12 import Profile from '../components/Profile';
13 import Register from '../components/Register';
14 import Settings from '../components/Settings';
15 import { store } from '../store';
16 import { push } from 'react-router-redux';
17
18 const mapStateToProps = state => {
19   return {
20     appLoaded: state.common.appLoaded,
21     appName: state.common.appName,
22     currentUser: state.common.currentUser,
23     redirectTo: state.common.redirectTo
24   }};
25
26 const mapDispatchToProps = dispatch => ({
27   onLoad: (payload, token) =>
28     dispatch({ type: APP_LOAD, payload, token,
         skipTracking: true }),
29   onRedirect: () =>
30     dispatch({ type: REDIRECT })
31 });
32
33 class App extends React.Component {
34   componentWillReceiveProps(nextProps) {
35     if (nextProps.redirectTo) {
36       // this.context.router.replace(nextProps.
           redirectTo);
37       store.dispatch(push(nextProps.redirectTo));
38       this.props.onRedirect();
39     }
40   }
41
42   componentWillMount() {
43     const token = window.localStorage.getItem('jwt');
44     if (token) {
45       agent.setToken(token);
46     }
47
48     this.props.onLoad(token ? agent.Auth.current() :
         null, token);
49   }
50
51   render() {
52     if (this.props.appLoaded) {
53       return (
54         <div>
55           <Header
56             appName={this.props.appName}
57             currentUser={this.props.currentUser} />
58             <Switch>
```

```
            <Route exact path="/" component={Home}/>
            <Route path="/login" component={Login} />
            <Route path="/register" component={
                Register} />
            <Route path="/editor/:slug" component={
                Editor} />
            <Route path="/editor" component={Editor} /
                >
            <Route path="/notification/:id" component
                ={Article} />
            <Route path="/settings" component={
                Settings} />
            <Route path="/@:username" component={
                Profile} />
            {/*<Route path="/transaction/:transaction"
                 component={Transaction} />*/}
            </Switch>
        </div>
      );
    }
    return (
      <div>
        <Header
          appName={this.props.appName}
          currentUser={this.props.currentUser} />
      </div>
    );
  }
}

// App.contextTypes = {
//   router: PropTypes.object.isRequired
// };

export default connect(mapStateToProps,
    mapDispatchToProps)(App);
```

Listing A.30 – web/App.js

<!—->

```
import { Link } from 'react-router-dom';
import React from 'react';
import agent from '../../agent';
import { connect } from 'react-redux';
import { DELETE_ARTICLE } from '../../constants/
    actionTypes';

const mapDispatchToProps = dispatch => ({
  onClickDelete: payload =>
    dispatch({ type: DELETE_ARTICLE, payload })
});
```

```
11
12 const ArticleActions = props => {
13   const notification = props.notification;
14   const del = () => {
15     props.onClickDelete(agent.Articles.del(
           notification.slug))
16   };
17   if (props.canModify) {
18     return (
19       <span>
20
21         <Link
22           to={`/editor/${notification.slug}`}
23           className="btn btn-outline-secondary btn-sm"
               >
24           <i className="ion-edit"></i> Edit Article
25         </Link>
26
27         <button className="btn btn-outline-danger btn-
               sm" onClick={del}>
28           <i className="ion-trash-a"></i> Delete
                 Article
29         </button>
30
31       </span>
32     );
33   }
34
35   return (
36     <span>
37     </span>
38   );
39 };
40
41 export default connect(() => ({}), mapDispatchToProps)
     (ArticleActions);
```

Listing A.31 – web/ArticleActions.js

<!—->

```
1  import {
2    SET_PAGE,
3    HOME_PAGE_LOADED,
4    HOME_PAGE_UNLOADED,
5    CHANGE_TAB,
6    PROFILE_PAGE_LOADED,
7    PROFILE_PAGE_UNLOADED,
8    PROFILE_FAVORITES_PAGE_LOADED,
9    PROFILE_FAVORITES_PAGE_UNLOADED
10 } from '../constants/actionTypes';
11
```

```
12 export default (state = {}, action) => {
13   switch (action.type) {
14     case SET_PAGE:
15       return {
16         ...state,
17         notifications: action.payload.notifications,
18         notificationsCount: action.payload.
               notificationsCount,
19         currentPage: action.page
20       };
21     case HOME_PAGE_LOADED:
22       return {
23         ...state,
24         pager: action.pager[0],
25         notifications: action.payload[0].notifications
               ,
26         notificationsCount: action.payload[0].
               notificationsCount,
27         currentPage: 0,
28         tab: action.tab
29       };
30     case HOME_PAGE_UNLOADED:
31       return {};
32     case CHANGE_TAB:
33       return {
34         ...state,
35         pager: action.pager,
36         notifications: action.payload.notifications,
37         notificationsCount: action.payload.
               notificationsCount,
38         tab: action.tab,
39         currentPage: 0
40       };
41     case PROFILE_PAGE_LOADED:
42     case PROFILE_FAVORITES_PAGE_LOADED:
43       return {
44         ...state,
45         pager: action.pager[0],
46         notifications: action.payload[0].notifications
               ,
47         notificationsCount: action.payload[0].
               notificationsCount,
48         currentPage: 0
49       };
50     case PROFILE_PAGE_UNLOADED:
51     case PROFILE_FAVORITES_PAGE_UNLOADED:
52       return {};
53     default:
54       return state;
55   }
```

```
56 };
```

Listing A.32 – web/articleList.js

<!—->

```
1  import ArticlePreview from './ArticlePreview';
2  import ListPagination from './ListPagination';
3  import React from 'react';
4
5  const ArticleList = props => {
6    if (!props.notifications) {
7      return (
8        <div className="article-preview">Loading...</div
             >
9      );
10   }
11
12   if (props.notifications.length === 0) {
13     return (
14       <div className="article-preview">
15         Nenhuma notifica  o
16       </div>
17     );
18   }
19
20   return (
21     <div>
22       {
23         props.notifications.map(notification => {
24           return (
25             <ArticlePreview
26                 notification={notification}
27                 key={notification.id}
28                 onNotificationClick={props.
                       onNotificationClick}
29             />
30           );
31         })
32       }
33
34       <ListPagination
35         pager={props.pager}
36         notificationsCount={props.notificationsCount}
37         currentPage={props.currentPage} />
38     </div>
39   );
40 };
41
42 export default ArticleList;
```

Listing A.33 – web/ArticleList.js

<!—->

```
import ArticleActions from './ArticleActions';
import { Link } from 'react-router-dom';
import React from 'react';

const ArticleMeta = props => {
  const notification = props.notification;
  return (
    <div className="article-meta">
      <Link to={`/@${notification.author.username}`}>
        <img src={notification.author.image} alt={
          notification.author.username} />
      </Link>

      <div className="info">
        <Link to={`/@${notification.author.username}`}
            className="author">
          {notification.author.username}
        </Link>
        <span className="date">
          {new Date(notification.createdAt).
              toDateString()}
        </span>
      </div>

      <ArticleActions canModify={props.canModify}
          notification={notification} />
    </div>
  );
};

export default ArticleMeta;
```

Listing A.34 – web/ArticleMeta.js

<!—->

```
import React from 'react';
import { Link } from 'react-router-dom';
import agent from '../agent';
import { connect } from 'react-redux';

const mapDispatchToProps = () => ({
});

function verifiedTransaction(transaction) {
    if(transaction !== undefined) {
        return <span className="preview-link">
                {transaction}
              </span>
    }
```

```
    else {
        return <span className="preview-link">
                {'Transa  o  n  o  confirmada.'}
              </span>
    }
}

const ArticlePreview = props => {
  const notification = props.notification;

  return (
    <div className="article-preview">
      <div className="article-meta">
        <div className="info">
          <span className="author">
            {"Remetente: " + notification.author.
                username}
          </span>
          <span className="destinatary">
            {"Destinatario: " + notification.destMail}
          </span>
          <span className="date">
            {"Data da requisi  o: " + new Date(
                notification.createdAt).toDateString()
                }
          </span>
          {verifiedTransaction(notification.
              transaction)}
        </div>
      </div>
    </div>
  );
}

export default connect(() => ({}), mapDispatchToProps)
    (ArticlePreview);
```

Listing A.35 – web/ArticlePreview.js

<!—->

```
import {
  LOGIN,
  REGISTER,
  LOGIN_PAGE_UNLOADED,
  REGISTER_PAGE_UNLOADED,
  ASYNC_START,
  UPDATE_FIELD_AUTH
} from '../constants/actionTypes';

export default (state = {}, action) => {
  switch (action.type) {
```

```
    case LOGIN:
    case REGISTER:
      return {
        ...state,
        inProgress: false,
        errors: action.error ? action.payload.errors :
            null
      };
    case LOGIN_PAGE_UNLOADED:
    case REGISTER_PAGE_UNLOADED:
      return {};
    case ASYNC_START:
      if (action.subtype === LOGIN || action.subtype
          === REGISTER) {
        return { ...state, inProgress: true };
      }
      break;
    case UPDATE_FIELD_AUTH:
      return { ...state, [action.key]: action.value };
    default:
      return state;
  }

  return state;
};
```

Listing A.36 – web/auth.js

<!—->

```
import React from 'react';

const Banner = ({ appName, token }) => {
  if (token) {
    return null;
  }
  return (
    <div className="banner">
      <div className="container">
        <h1 className="logo-font">
          {appName.toLowerCase()}
        </h1>
        <p>A place to share your knowledge.</p>
      </div>
    </div>
  );
};

export default Banner;
```

Listing A.37 – web/Banner.js

<!—->

```
const Web3 = require('web3');

const blockchainConfig = {
    nodeAccountAddress: "0
        xd7904971ab3b5c4cc6f6781ee60324c423d4e94d",
    nodeRPCAddress: 'http://',
    nodeWSAddress: 'ws://',
    gasPrice: '20000000000'
};

const web3 = new Web3(new Web3.providers.
    WebsocketProvider(blockchainConfig.nodeWSAddress,
    {
    headers: {
        Origin: "narga"
    }
}));
web3.eth.defaultAccount = blockchainConfig.
    nodeAccountAddress;

// module.exports = Object.freeze(web3);
export {blockchainConfig, web3};
```

Listing A.38 – web/BlockchainConfig.js

<!—->

```
const notificationContract = {
    descentralizedNotificationContract: [{"constant":
        true,"inputs":[{"name":"cpf","type":"int40
        "}],"name":"latestNotification","outputs":[{"
        name":"","type":"string"},{"name":"","type":"
        string"},{"name":"","type":"string"}],"payable
        ":false,"stateMutability":"view","type":"
        function"},{"constant":false,"inputs":[{"name
        ":"cpf","type":"int40"},{"name":"notif_hash","
        type":"string"},{"name":"senderMail","type":"
        string"},{"name":"destMail","type":"string
        "},{"name":"timestamp","type":"uint32"}],"name
        ":"notify","outputs":[],"payable":false,"
        stateMutability":"nonpayable","type":"function
        "},{"constant":false,"inputs":[{"name":"
        notifier","type":"address"}],"name":"
        addPermission","outputs":[],"payable":false,"
        stateMutability":"nonpayable","type":"function
        "},{"constant":true,"inputs":[{"name":"cpf","
        type":"int40"}],"name":"numberOfNotifications
        ","outputs":[{"name":"","type":"uint256"}],"
        payable":false,"stateMutability":"view","type
        ":"function"},{"constant":true,"inputs":[{"
```

```
        name":"cpf","type":"int40"},{"name":"index","
        type":"uint256"}],"name":"specificNotification
        ","outputs":[{"name":"","type":"string"},{"
        name":"","type":"string"},{"name":"","type":"
        string"}],"payable":false,"stateMutability":"
        view","type":"function"},{"inputs":[],"payable
        ":false,"stateMutability":"nonpayable","type
        ":"constructor"},{"anonymous":false,"inputs
        ":[{"indexed":false,"name":"senderMail","type
        ":"string"},{"indexed":false,"name":"destMail
        ","type":"string"},{"indexed":false,"name":"
        notif_hash","type":"string"},{"indexed":false
        ,"name":"timestamp","type":"uint32"}],"name":"
        NewNotification","type":"event"}],
    descentralizedNotificationContractAddress: '0
        x71c662a2347856c3e82132997cfcd15b83be2aca'
};

export {notificationContract};
```

Listing A.39 – web/BlockchainContracts.js

<!—->

```
import * as blockchain from './BlockchainConfig';
import * as contracts from './BlockchainContracts';

var contract = new blockchain.web3.eth.Contract(
    contracts.notificationContract.
    descentralizedNotificationContract,
    contracts.notificationContract.
        descentralizedNotificationContractAddress, {
        from: blockchain.blockchainConfig.
            nodeAccountAddress,
        gasPrice: blockchain.blockchainConfig.gasPrice
    });

const pastTransactions = () => {
        return contract.getPastEvents('allEvents',
            {
                fromBlock: 0,
                toBlock: 'latest'
            });
};
const newTransactions = () => {
        return contract.events.allEvents({
            fromBlock: 0,
            toBlock: 'latest'
        });
};

```

```
export default { pastTransactions, newTransactions }
```

Listing A.40 – web/blockchain.js

<!—->

```
import {
  APP_LOAD,
  REDIRECT,
  LOGOUT,
  ARTICLE_SUBMITTED,
  SETTINGS_SAVED,
  LOGIN,
  REGISTER,
  DELETE_ARTICLE,
  ARTICLE_PAGE_UNLOADED,
  EDITOR_PAGE_UNLOADED,
  HOME_PAGE_UNLOADED,
  TRANSACTION_UPDATE,
  PROFILE_PAGE_UNLOADED,
  PROFILE_FAVORITES_PAGE_UNLOADED,
  SETTINGS_PAGE_UNLOADED,
  LOGIN_PAGE_UNLOADED,
  REGISTER_PAGE_UNLOADED
} from '../constants/actionTypes';

const defaultState = {
  appName: 'Notifica  es Digitais',
  token: null,
  viewChangeCounter: 0
};

export default (state = defaultState, action) => {
  switch (action.type) {
    case APP_LOAD:
      return {
        ...state,
        token: action.token || null,
        appLoaded: true,
        currentUser: action.payload ? action.payload.
            user : null
      };
    case REDIRECT:
      return { ...state, redirectTo: null };
    case LOGOUT:
      return { ...state, redirectTo: '/', token: null,
          currentUser: null };
    case ARTICLE_SUBMITTED:
      const redirectUrl = `/notifications/${action.
          payload.notification.slug}`;
      return { ...state, redirectTo: redirectUrl };
    case SETTINGS_SAVED:
```

```
      return {
        ...state,
        redirectTo: action.error ? null : '/',
        currentUser: action.error ? null : action.
            payload.user
      };
    case LOGIN:
    case REGISTER:
      return {
        ...state,
        redirectTo: action.error ? null : '/',
        token: action.error ? null : action.payload.
            user.token,
        currentUser: action.error ? null : action.
            payload.user
      };
    case DELETE_ARTICLE:
      return { ...state, redirectTo: '/' };
    case ARTICLE_PAGE_UNLOADED:
    case EDITOR_PAGE_UNLOADED:
    case HOME_PAGE_UNLOADED:
    case TRANSACTION_UPDATE:
    case PROFILE_PAGE_UNLOADED:
    case PROFILE_FAVORITES_PAGE_UNLOADED:
    case SETTINGS_PAGE_UNLOADED:
    case LOGIN_PAGE_UNLOADED:
    case REGISTER_PAGE_UNLOADED:
      return { ...state, viewChangeCounter: state.
          viewChangeCounter + 1 };
    default:
      return state;
  }
};
```

Listing A.41 – web/common.js

<!—->

```
import React from 'react';
import agent from '../../agent';
import { connect } from 'react-redux';
import { DELETE_COMMENT } from '../../constants/
    actionTypes';

const mapDispatchToProps = dispatch => ({
  onClick: (payload, commentId) =>
    dispatch({ type: DELETE_COMMENT, payload,
        commentId })
});

const DeleteButton = props => {
  const del = () => {
```

```
13      const payload = agent.Comments.delete(props.slug,
            props.commentId);
14      props.onClick(payload, props.commentId);
15    };
16
17    if (props.show) {
18      return (
19        <span className="mod-options">
20          <i className="ion-trash-a" onClick={del}></i>
21        </span>
22      );
23    }
24    return null;
25  };
26
27  export default connect(() => ({}), mapDispatchToProps)
      (DeleteButton);
```

Listing A.42 – web/DeleteButton.js

<!—->

```
1   import {
2     EDITOR_PAGE_LOADED,
3     EDITOR_PAGE_UNLOADED,
4     ARTICLE_SUBMITTED,
5     ASYNC_START,
6     UPDATE_FIELD_EDITOR
7   } from '../constants/actionTypes';
8
9   export default (state = {}, action) => {
10    switch (action.type) {
11      case EDITOR_PAGE_LOADED:
12        return {
13          ...state,
14          destinatary: action.payload ? action.payload.
                notification.destinatary : '',
15          cpf: action.payload ? action.payload.
                notification.cpf : '',
16          body: action.payload ? action.payload.
                notification.body : ''
17        };
18      case EDITOR_PAGE_UNLOADED:
19        return {};
20      case ARTICLE_SUBMITTED:
21        return {
22          ...state,
23          inProgress: null,
24          errors: action.error ? action.payload.errors :
                null
25        };
26      case ASYNC_START:
```

```
      if (action.subtype === ARTICLE_SUBMITTED) {
        return { ...state, inProgress: true };
      }
      break;
    case UPDATE_FIELD_EDITOR:
      return { ...state, [action.key]: action.value };
    default:
      return state;
  }

  return state;
};
```

Listing A.43 – web/editor.js

<!—->

```
import ListErrors from './ListErrors';
import React from 'react';
import agent from '../agent';
import { connect } from 'react-redux';
import {
  EDITOR_PAGE_LOADED,
  ARTICLE_SUBMITTED,
  EDITOR_PAGE_UNLOADED,
  UPDATE_FIELD_EDITOR
} from '../constants/actionTypes';

const mapStateToProps = state => ({
  ...state.editor
});

const mapDispatchToProps = dispatch => ({
  onLoad: payload =>
    dispatch({ type: EDITOR_PAGE_LOADED, payload }),
  onSubmit: payload =>
    dispatch({ type: ARTICLE_SUBMITTED, payload }),
  onUnload: payload =>
    dispatch({ type: EDITOR_PAGE_UNLOADED }),
  onUpdateField: (key, value) =>
    dispatch({ type: UPDATE_FIELD_EDITOR, key, value
        })
});

class Editor extends React.Component {
  constructor() {
    super();

    const updateFieldEvent =
      key => ev => this.props.onUpdateField(key, ev.
          target.value);
    this.changeDestinatary = updateFieldEvent('
```

```
        destinatary');
    this.changeCpf = updateFieldEvent('cpf');
    this.changeBody = updateFieldEvent('body');

    this.watchForEnter = ev => {
      if (ev.keyCode === 13) {
        ev.preventDefault();
        this.props.onAddTag();
      }
    };

    this.submitForm = ev => {
      ev.preventDefault();
      const notification = {
        destinatary: this.props.destinatary,
        cpf: this.props.cpf,
        body: this.props.body
      };

      var rest = agent.Articles.create(notification);
      this.props.onSubmit(rest);
    };
  }

  componentWillReceiveProps(nextProps) {
    if (this.props.match.params.slug !== nextProps.
        match.params.slug) {
      if (nextProps.match.params.slug) {
        this.props.onUnload();
        return this.props.onLoad(agent.Articles.get(
            this.props.match.params.slug));
      }
      this.props.onLoad(null);
    }
  }

  componentWillMount() {
    if (this.props.match.params.slug) {
      return this.props.onLoad(agent.Articles.get(this
          .props.match.params.slug));
    }
    this.props.onLoad(null);
  }

  componentWillUnmount() {
    this.props.onUnload();
  }

  render() {
    return (
      <div className="editor-page">
        <div className="container page">
```

```
82              <div className="row">
83                <div className="col-md-10 offset-md-1 col-
                      xs-12">
84
85                  <ListErrors errors={this.props.errors}><
                        /ListErrors>
86
87                  <form>
88                    <fieldset>
89
90                      <fieldset className="form-group">
91                        <input
92                          className="form-control form-
                                control-lg"
93                          type="text"
94                          placeholder="Email do
                                Destinat rio"
95                          value={this.props.destinatary}
96                          onChange={this.changeDestinatary
                                } />
97                      </fieldset>
98
99                      <fieldset className="form-group">
100                       <input
101                         className="form-control"
102                         type="text"
103                         placeholder="CPF do
                                destinat rio"
104                         value={this.props.cpf}
105                         onChange={this.changeCpf} />
106                     </fieldset>
107
108                     <fieldset className="form-group">
109                       <textarea
110                         className="form-control"
111                         rows="8"
112                         placeholder="Dados a serem
                                enviados"
113                         value={this.props.body}
114                         onChange={this.changeBody}>
115                       </textarea>
116                     </fieldset>
117
118                     {/*<fieldset className="form-group">
                            */}
119                       {/*<input*/}
120                         {/*className="form-control"*/}
121                         {/*type="text"*/}
122                         {/*placeholder="Enter tags"*/}
123                         {/*value={this.props.tagInput
                                }*/}
124                         {/*onChange={this.changeTagInput
```

```
                            }*/}
125                       {/*onKeyUp={this.watchForEnter}
                              />*/}
126
127                     {/*<div className="tag-list">*/}
128                       {/*{*/}
129                         {/*(this.props.tagList || []).
                                map(tag => {*/}
130                           {/*return (*/}
131                             {/*<span className="tag-
                                    default tag-pill" key
                                    ={tag}>*/}
132                               {/*<i  className="ion-
                                    close-round"*/}
133                                 {/*onClick={this.
                                      removeTagHandler
                                      (tag)}>*/}
134                               {/*</i>*/}
135                               {/*{tag}*/}
136                             {/*</span>*/}
137                           {/*);*/}
138                         {/*})*/}
139                       {/*}*/}
140                     {/*</div>*/}
141                   {/*</fieldset>*/}
142
143                   <button
144                     className="btn btn-lg pull-xs-
                          right btn-primary"
145                     type="button"
146                     disabled={this.props.inProgress}
147                     onClick={this.submitForm}>
148                     Enviar Notifica  o
149                   </button>
150
151                 </fieldset>
152               </form>
153
154             </div>
155           </div>
156         </div>
157       </div>
158     );
159   }
160 }
161
162 export default connect(mapStateToProps,
      mapDispatchToProps)(Editor);
```

Listing A.44 – web/Editor.js

<!—->

```
const Web3 = require('web3');

const blockchainConfig = {
    nodeAccountAddress: "0
        xd7904971ab3b5c4cc6f6781ee60324c423d4e94d",
    nodeRPCAddress: 'http://',
    nodeWSAddress: 'ws://',
    gasPrice: '20000000000'
};

const blockchainContracts = {
    descentralizedNotificationContract: [{"constant":
        true,"inputs":[{"name":"cpf","type":"int40
        "}],"name":"latestNotification","outputs":[{"
        name":"","type":"string"},{"name":"","type":"
        string"},{"name":"","type":"string"}],"payable
        ":false,"stateMutability":"view","type":"
        function"},{"constant":false,"inputs":[{"name
        ":"cpf","type":"int40"},{"name":"notif_hash","
        type":"string"},{"name":"senderMail","type":"
        string"},{"name":"destMail","type":"string
        "},{"name":"timestamp","type":"uint32"}],"name
        ":"notify","outputs":[],"payable":false,"
        stateMutability":"nonpayable","type":"function
        "},{"constant":false,"inputs":[{"name":"
        notifier","type":"address"}],"name":"
        addPermission","outputs":[],"payable":false,"
        stateMutability":"nonpayable","type":"function
        "},{"constant":true,"inputs":[{"name":"cpf","
        type":"int40"}],"name":"numberOfNotifications
        ","outputs":[{"name":"","type":"uint256"}],"
        payable":false,"stateMutability":"view","type
        ":"function"},{"constant":true,"inputs":[{"
        name":"cpf","type":"int40"},{"name":"index","
        type":"uint256"}],"name":"specificNotification
        ","outputs":[{"name":"","type":"string"},{"
        name":"","type":"string"},{"name":"","type":"
        string"}],"payable":false,"stateMutability":"
        view","type":"function"},{"inputs":[],"payable
        ":false,"stateMutability":"nonpayable","type
        ":"constructor"},{"anonymous":false,"inputs
        ":[{"indexed":false,"name":"senderMail","type
        ":"string"},{"indexed":false,"name":"destMail
        ","type":"string"},{"indexed":false,"name":"
        notif_hash","type":"string"},{"indexed":false
        ,"name":"timestamp","type":"uint32"}],"name":"
        NewNotification","type":"event"}],
    descentralizedNotificationContractAddress: '0
        x93f968be17a0b1d29f7d6d6a789af43e6b54e1d0'
};
```

```
14 const web3 = new Web3(new Web3.providers.
       WebsocketProvider(blockchainConfig.nodeWSAddress,
       {
15     headers: {
16         Origin: ""
17     }
18 }));
19 web3.eth.defaultAccount = blockchainConfig.
       nodeAccountAddress;
20
21 web3.eth.getBalance(web3.eth.defaultAccount).then(r =>
       {
22     console.log(r);
23 });
24
25 var contract = new web3.eth.Contract(
       blockchainContracts.
       descentralizedNotificationContract,
26     blockchainContracts.
           descentralizedNotificationContractAddress, {
27         from: blockchainConfig.nodeAccountAddress,
28         gasPrice: blockchainConfig.gasPrice
29     });
30 contract.events.allEvents({
31     fromBlock: 0,
32     toBlock: 'latest'
33 })
34     .on('data', function(event) {
35         console.log(event.returnValues);
36     })
37     .on('changed', console.log)
38     .on('error', console.log);
39
40 /*
41 Receive the CPF to consult as parameter and return a
       Promise with a
42 JSON parameter with the following struture
43 Result {
44     '0': 'newname',
45     '1': 'newsender@mail.com',
46     '2': 'receipient@hothot.com',
47     '3': 'HASHNOTIFICATION2' }
48  */
49 function latestNotification(cpf) {
50     return contract.methods.latestNotification(cpf)
51         .call({from: blockchainConfig.
               nodeAccountAddress})
52 }
53
54 /*
55 Receive the CPF to consult  and index of the desired
       notification
```

```
 as parameter and return a Promise with a JSON
     parameter with the
 following structure:
Result {
    '0': 'newname',
    '1': 'newsender@mail.com',
    '2': 'receipient@hothot.com',
    '3': 'HASHNOTIFICATION2' }
 */
function specificNotification(cpf, index) {
    return contract.methods.latestNotification(cpf,
        index)
        .call({from: blockchainConfig.
            nodeAccountAddress})
}

/*
Create a notification to the CPF listed. The
    notification must include
the hash of the notification document, sender email
    address, sender name
or any alias and destinatary mail.
The timestamp will be generated on this requisition,
    not the blockchain time.
 */
function notify(hash, sender_mail, cpf, dest_mail) {
    return contract.methods.notify(cpf, hash,
        sender_mail, dest_mail, new Date().getTime())
        .send({
            from: blockchainConfig.nodeAccountAddress,
            gas: 200000
        });
}

/*
Create a notification to the CPF listed. The
    notification must include
the hash of the notification document, sender email
    address, sender name
or any alias and destinatary mail.
The timestamp will be generated on this requisition,
    not the blockchain time.
 */
function transactionByHash(transaction) {
    return web3.eth.getTransaction(transaction);
}
function transactionInput(transaction) {
    contract.getPastEvents('allEvents', {
        fromBlock: 0,
        toBlock: 'latest'
    }).then(console.log);
```

```
97 }
```

Listing A.45 – web/EthereumConnectionHandler.js

<!—->

```
1  import React from 'react';
2  import { Link } from 'react-router-dom';
3
4  const LoggedOutView = props => {
5    if (!props.currentUser) {
6      return (
7        <ul className="nav navbar-nav pull-xs-right">
8
9          <li className="nav-item">
10           <Link to="/" className="nav-link">
11             Home
12           </Link>
13         </li>
14
15         <li className="nav-item">
16           <Link to="/login" className="nav-link">
17             Sign in
18           </Link>
19         </li>
20
21         <li className="nav-item">
22           <Link to="/register" className="nav-link">
23             Sign up
24           </Link>
25         </li>
26
27       </ul>
28     );
29   }
30   return null;
31 };
32
33 const LoggedInView = props => {
34   if (props.currentUser) {
35     return (
36       <ul className="nav navbar-nav pull-xs-right">
37
38         <li className="nav-item">
39           <Link to="/" className="nav-link">
40             Home
41           </Link>
42         </li>
43
44         <li className="nav-item">
45           <Link to="/editor" className="nav-link">
46             <i className="ion-compose"></i> Nova
```

```
                Notifica  o
47            </Link>
48          </li>
49
50          <li className="nav-item">
51            <Link to="/settings" className="nav-link">
52              <i className="ion-gear-a"></i> 
                    Configura  es
53            </Link>
54          </li>
55
56          <li className="nav-item">
57            <Link
58              to={`/@${props.currentUser.username}`}
59              className="nav-link">
60              <img src={props.currentUser.image}
                    className="user-pic" alt={props.
                    currentUser.username} />
61              {props.currentUser.username}
62            </Link>
63          </li>
64
65        </ul>
66      );
67    }
68
69    return null;
70  };
71
72  class Header extends React.Component {
73    render() {
74      return (
75        <nav className="navbar navbar-light">
76          <div className="container">
77
78            <Link to="/" className="navbar-brand">
79              {this.props.appName.toLowerCase()}
80            </Link>
81
82            <LoggedOutView currentUser={this.props.
                  currentUser} />
83
84            <LoggedInView currentUser={this.props.
                  currentUser} />
85          </div>
86        </nav>
87      );
88    }
89  }
90
```

```
91 export default Header;
```

Listing A.46 – web/Header.js

<!—->

```
1  import { HOME_PAGE_LOADED, HOME_PAGE_UNLOADED } from
       '../constants/actionTypes';
2
3  export default (state = {}, action) => {
4    switch (action.type) {
5      case HOME_PAGE_LOADED:
6        return {
7          ...state
8        };
9      case HOME_PAGE_UNLOADED:
10       return {};
11     default:
12       return state;
13   }
14 };
```

Listing A.47 – web/home.js

<!—->

```
1  import ReactDOM from 'react-dom';
2  import { Provider } from 'react-redux';
3  import React from 'react';
4  import { store, history} from './store';
5
6  import { Route, Switch } from 'react-router-dom';
7  import { ConnectedRouter } from 'react-router-redux';
8
9  import App from './components/App';
10
11 ReactDOM.render((
12   <Provider store={store}>
13     <ConnectedRouter history={history}>
14       <Switch>
15         <Route path="/" component={App} />
16       </Switch>
17     </ConnectedRouter>
18   </Provider>
19
20 ), document.getElementById('root'));
```

Listing A.48 – web/index.js

<!—->

```
import React from 'react';

class ListErrors extends React.Component {
  render() {
    const errors = this.props.errors;
    if (errors) {
      return (
        <ul className="error-messages">
          {
            Object.keys(errors).map(key => {
              return (
                <li key={key}>
                  {key} {errors[key]}
                </li>
              );
            })
          }
        </ul>
      );
    } else {
      return null;
    }
  }
}

export default ListErrors;
```

Listing A.49 – web/ListErrors.js

<!—->

```
import React from 'react';
import agent from '../agent';
import { connect } from 'react-redux';
import { SET_PAGE } from '../constants/actionTypes';

const mapDispatchToProps = dispatch => ({
  onSetPage: (page, payload) =>
    dispatch({ type: SET_PAGE, page, payload })
});

const ListPagination = props => {
  if (props.notificationsCount <= 10) {
    return null;
  }

  const range = [];
  for (let i = 0; i < Math.ceil(props.
      notificationsCount / 10); ++i) {
    range.push(i);
  }

```

```
21    const setPage = page => {
22      if(props.pager) {
23        props.onSetPage(page, props.pager(page));
24      }else {
25        props.onSetPage(page, agent.Articles.all(page))
26      }
27    };
28
29    return (
30      <nav>
31        <ul className="pagination">
32
33          {
34            range.map(v => {
35              const isCurrent = v === props.currentPage;
36              const onClick = ev => {
37                ev.preventDefault();
38                setPage(v);
39              };
40              return (
41                <li
42                  className={ isCurrent ? 'page-item
                      active' : 'page-item' }
43                  onClick={onClick}
44                  key={v.toString()}>
45
46                  <a className="page-link" href="">{v +
                      1}</a>
47
48                </li>
49              );
50            })
51          }
52
53        </ul>
54      </nav>
55    );
56  };
57
58  export default connect(() => ({}), mapDispatchToProps)
      (ListPagination);
```

Listing A.50 – web/ListPagination.js

<!—->

```
1  import { Link } from 'react-router-dom';
2  import ListErrors from './ListErrors';
3  import React from 'react';
4  import agent from '../agent';
5  import { connect } from 'react-redux';
6  import {
```

```
  UPDATE_FIELD_AUTH,
  LOGIN,
  LOGIN_PAGE_UNLOADED
} from '../constants/actionTypes';

const mapStateToProps = state => ({ ...state.auth });

const mapDispatchToProps = dispatch => ({
  onChangeEmail: value =>
    dispatch({ type: UPDATE_FIELD_AUTH, key: 'email',
        value }),
  onChangePassword: value =>
    dispatch({ type: UPDATE_FIELD_AUTH, key: 'password
        ', value }),
  onSubmit: (email, password) =>
    dispatch({ type: LOGIN, payload: agent.Auth.login(
        email, password) }),
  onUnload: () =>
    dispatch({ type: LOGIN_PAGE_UNLOADED })
});

class Login extends React.Component {
  constructor() {
    super();
    this.changeEmail = ev => this.props.onChangeEmail(
        ev.target.value);
    this.changePassword = ev => this.props.
        onChangePassword(ev.target.value);
    this.submitForm = (email, password) => ev => {
      ev.preventDefault();
      this.props.onSubmit(email, password);
    };
  }

  componentWillUnmount() {
    this.props.onUnload();
  }

  render() {
    const email = this.props.email;
    const password = this.props.password;
    return (
      <div className="auth-page">
        <div className="container page">
          <div className="row">

            <div className="col-md-6 offset-md-3 col-
                xs-12">
              <h1 className="text-xs-center">Sign In</
                  h1>
              <p className="text-xs-center">
                <Link to="/register">
```

```
52                     Need an account?
53                   </Link>
54                 </p>
55
56                 <ListErrors errors={this.props.errors} />
57
58                 <form onSubmit={this.submitForm(email,
                       password)}>
59                   <fieldset>
60
61                     <fieldset className="form-group">
62                       <input
63                         className="form-control form-
                             control-lg"
64                         type="email"
65                         placeholder="Email"
66                         value={email}
67                         onChange={this.changeEmail} />
68                     </fieldset>
69
70                     <fieldset className="form-group">
71                       <input
72                         className="form-control form-
                             control-lg"
73                         type="password"
74                         placeholder="Password"
75                         value={password}
76                         onChange={this.changePassword} />
77                     </fieldset>
78
79                     <button
80                       className="btn btn-lg btn-primary
                           pull-xs-right"
81                       type="submit"
82                       disabled={this.props.inProgress}>
83                       Sign in
84                     </button>
85
86                   </fieldset>
87                 </form>
88               </div>
89
90             </div>
91           </div>
92         </div>
93       );
94     }
95   }
96
97   export default connect(mapStateToProps,
```

```
    mapDispatchToProps)(Login);
```

Listing A.51 – web/Login.js

<!—->

```
import ArticleList from '../ArticleList';
import React from 'react';
import agent from '../../agent';
import { connect } from 'react-redux';
import { CHANGE_TAB } from '../../constants/
    actionTypes';

const YourFeedTab = props => {
  if (props.token) {
    const clickHandler = ev => {
      ev.preventDefault();
      props.onTabClick('feed', agent.Articles.feed,
          agent.Articles.feed());
    }

    return (
      <li className="nav-item">
        <a  href=""
            className={ props.tab === 'feed' ? 'nav-
                link active' : 'nav-link' }
            onClick={clickHandler}>
          Suas Notifica  es
        </a>
      </li>
    );
  }
  return null;
};

const GlobalFeedTab = props => {
  const clickHandler = ev => {
    ev.preventDefault();
    props.onTabClick('all', agent.Articles.all, agent.
        Articles.all());
  };

  return (
    <li className="nav-item">
      <a
        href=""
        className={ props.tab === 'all' ? 'nav-link
            active' : 'nav-link' }
        onClick={clickHandler}>
        Novas Notifica  es
      </a>
    </li>
```

```
  );
};

const mapStateToProps = state => ({
  ...state.notificationList,
  token: state.common.token
});

const mapDispatchToProps = dispatch => ({
  onTabClick: (tab, pager, payload) => dispatch({ type
      : CHANGE_TAB, tab, pager, payload })
});

const MainView = props => {

  return (
    <div className="col-md-6">
      <div className="feed-toggle">
        <ul className="nav nav-pills outline-active">
          <YourFeedTab
            token={props.token}
            tab={props.tab}
            onTabClick={props.onTabClick} />
          <GlobalFeedTab tab={props.tab} onTabClick={
              props.onTabClick} />
        </ul>
      </div>
      <ArticleList
        pager={props.pager}
        notifications={props.notifications}
        loading={props.loading}
        notificationsCount={props.notificationsCount}
        currentPage={props.currentPage} />
    </div>
  );
};

export default connect(mapStateToProps,
    mapDispatchToProps)(MainView);
```

Listing A.52 – web/MainView.js

<!—->

```
import agent from './agent';
import {
  ASYNC_START,
  ASYNC_END,
  LOGIN,
  LOGOUT,
  REGISTER
} from './constants/actionTypes';
```

```
9
10 const promiseMiddleware = store => next => action => {
11   if (isPromise(action.payload)) {
12     store.dispatch({ type: ASYNC_START, subtype:
           action.type });
13
14     const currentView = store.getState().
           viewChangeCounter;
15     const skipTracking = action.skipTracking;
16
17     action.payload.then(
18       res => {
19         const currentState = store.getState()
20         if (!skipTracking && currentState.
               viewChangeCounter !== currentView) {
21           return
22         }
23         console.log('RESULT', res);
24         action.payload = res;
25         store.dispatch({ type: ASYNC_END, promise:
               action.payload });
26         store.dispatch(action);
27       },
28       error => {
29         const currentState = store.getState()
30         if (!skipTracking && currentState.
               viewChangeCounter !== currentView) {
31           return
32         }
33         console.log('ERROR', error);
34         action.error = true;
35         action.payload = error.response.body;
36         if (!action.skipTracking) {
37           store.dispatch({ type: ASYNC_END, promise:
                 action.payload });
38         }
39         store.dispatch(action);
40       }
41     );
42
43     return;
44   }
45
46   next(action);
47 };
48
49 const localStorageMiddleware = store => next => action
       => {
50   if (action.type === REGISTER || action.type ===
         LOGIN) {
51     if (!action.error) {
52       window.localStorage.setItem('jwt', action.
```

```
            payload.user.token);
      agent.setToken(action.payload.user.token);
    }
  } else if (action.type === LOGOUT) {
    window.localStorage.setItem('jwt', '');
    agent.setToken(null);
  }

  next(action);
};

function isPromise(v) {
  return v && typeof v.then === 'function';
}


export { promiseMiddleware, localStorageMiddleware }
```

Listing A.53 – web/middleware.js

<!—->

```
import {
  ARTICLE_PAGE_LOADED,
  ARTICLE_PAGE_UNLOADED
} from '../constants/actionTypes';

export default (state = {}, action) => {
  switch (action.type) {
    case ARTICLE_PAGE_LOADED:
      return {
        ...state,
        notification: action.payload[0].notification
      };
    case ARTICLE_PAGE_UNLOADED:
      return {};
    default:
      return state;
  }
};
```

Listing A.54 – web/notification.js

<!—->

```
import {
  PROFILE_PAGE_LOADED,
  PROFILE_PAGE_UNLOADED
} from '../constants/actionTypes';

export default (state = {}, action) => {
  switch (action.type) {
```

```
    case PROFILE_PAGE_LOADED:
      return {
        ...action.payload[0].profile
      };
    case PROFILE_PAGE_UNLOADED:
      return {};
    default:
      return state;
  }
};
```

Listing A.55 – web/profile.js

<!—->

```
import ArticleList from './ArticleList';
import React from 'react';
import { Link } from 'react-router-dom';
import agent from '../agent';
import { connect } from 'react-redux';
import {
  PROFILE_PAGE_LOADED,
  PROFILE_PAGE_UNLOADED
} from '../constants/actionTypes';

const EditProfileSettings = props => {
  if (props.isUser) {
    return (
      <Link
        to="/settings"
        className="btn btn-sm btn-outline-secondary
            action-btn">
        <i className="ion-gear-a"></i> Edit Profile
            Settings
      </Link>
    );
  }
  return null;
};

const FollowUserButton = props => {
  if (props.isUser) {
    return null;
  }

  let classes = 'btn btn-sm action-btn';
  if (props.user.following) {
    classes += ' btn-secondary';
  } else {
    classes += ' btn-outline-secondary';
  }

```

```
36    const handleClick = ev => {
37      ev.preventDefault();
38      if (props.user.following) {
39        props.unfollow(props.user.username)
40      } else {
41        props.follow(props.user.username)
42      }
43    };
44
45    return (
46      <button
47        className={classes}
48        onClick={handleClick}>
49        <i className="ion-plus-round"></i>
50         
51        {props.user.following ? 'Unfollow' : 'Follow'} {
              props.user.username}
52      </button>
53    );
54  };
55
56  const mapStateToProps = state => ({
57    ...state.notificationList,
58    currentUser: state.common.currentUser,
59    profile: state.profile
60  });
61
62  const mapDispatchToProps = dispatch => ({
63    onLoad: payload => dispatch({ type:
          PROFILE_PAGE_LOADED, payload }),
64    onUnload: () => dispatch({ type:
          PROFILE_PAGE_UNLOADED })
65  });
66
67  class Profile extends React.Component {
68    componentWillMount() {
69      this.props.onLoad(Promise.all([
70        agent.Profile.get(this.props.match.params.
              username),
71        agent.Articles.byAuthor(this.props.match.params.
              username)
72      ]));
73    }
74
75    componentWillUnmount() {
76      this.props.onUnload();
77    }
78
79    renderTabs() {
80      return (
81        <ul className="nav nav-pills outline-active">
82          <li className="nav-item">
```

```
83            <Link
84              className="nav-link active"
85              to={`/@${this.props.profile.username}`}>
86              Minhas Notifica  es
87            </Link>
88          </li>
89        </ul>
90      );
91    }
92
93    render() {
94      const profile = this.props.profile;
95      if (!profile) {
96        return null;
97      }
98
99      const isUser = this.props.currentUser &&
100       this.props.profile.username === this.props.
            currentUser.username;
101
102     return (
103       <div className="profile-page">
104
105         <div className="user-info">
106           <div className="container">
107             <div className="row">
108               <div className="col-xs-12 col-md-10
                      offset-md-1">
109
110                 <img src={profile.image} className="
                        user-img" alt={profile.username} /
                        >
111                 <h4>{profile.username}</h4>
112                 <p>{profile.bio}</p>
113
114                 <EditProfileSettings isUser={isUser} /
                        >
115                 <FollowUserButton
116                   isUser={isUser}
117                   user={profile}
118                   />
119
120               </div>
121             </div>
122           </div>
123         </div>
124
125         <div className="container">
126           <div className="row">
127
128             <div className="col-xs-12 col-md-10 offset
                    -md-1">
```

```
129 |
130 |               <div className="notifications-toggle">
131 |                 {this.renderTabs()}
132 |               </div>
133 |
134 |               <ArticleList
135 |                 pager={this.props.pager}
136 |                 notifications={this.props.
    |                     notifications}
137 |                 notificationsCount={this.props.
    |                     notificationsCount}
138 |                 state={this.props.currentPage} />
139 |             </div>
140 |
141 |           </div>
142 |         </div>
143 |
144 |       </div>
145 |     );
146 |   }
147 | }
148 |
149 | export default connect(mapStateToProps,
    |     mapDispatchToProps)(Profile);
150 | export { Profile, mapStateToProps };
```

Listing A.56 – web/Profile.js

<!—->

```
1  | import article from './reducers/notification';
2  | import notificationList from './reducers/articleList';
3  | import transactionList from './reducers/
   |     transactionList';
4  | import auth from './reducers/auth';
5  | import { combineReducers } from 'redux';
6  | import common from './reducers/common';
7  | import editor from './reducers/editor';
8  | import home from './reducers/home';
9  | import profile from './reducers/profile';
10 | import settings from './reducers/settings';
11 | import { routerReducer } from 'react-router-redux';
12 |
13 | export default combineReducers({
14 |   article,
15 |   notificationList,
16 |   transactionList,
17 |   auth,
18 |   common,
19 |   editor,
20 |   home,
21 |   profile,
```

```
  settings,
  router: routerReducer
});
```

Listing A.57 – web/reducer.js

<!—->

```
import { Link } from 'react-router-dom';
import ListErrors from './ListErrors';
import React from 'react';
import agent from '../agent';
import { connect } from 'react-redux';
import {
  UPDATE_FIELD_AUTH,
  REGISTER,
  REGISTER_PAGE_UNLOADED
} from '../constants/actionTypes';

const mapStateToProps = state => ({ ...state.auth });

const mapDispatchToProps = dispatch => ({
  onChangeEmail: value =>
    dispatch({ type: UPDATE_FIELD_AUTH, key: 'email',
        value }),
  onChangePassword: value =>
    dispatch({ type: UPDATE_FIELD_AUTH, key: 'password
        ', value }),
  onChangeUsername: value =>
    dispatch({ type: UPDATE_FIELD_AUTH, key: 'username
        ', value }),
  onSubmit: (username, email, password) => {
    const payload = agent.Auth.register(username,
        email, password);
    dispatch({ type: REGISTER, payload })
  },
  onUnload: () =>
    dispatch({ type: REGISTER_PAGE_UNLOADED })
});

class Register extends React.Component {
  constructor() {
    super();
    this.changeEmail = ev => this.props.onChangeEmail(
        ev.target.value);
    this.changePassword = ev => this.props.
        onChangePassword(ev.target.value);
    this.changeUsername = ev => this.props.
        onChangeUsername(ev.target.value);
    this.submitForm = (username, email, password) =>
        ev => {
      ev.preventDefault();
```

```
37        this.props.onSubmit(username, email, password);
38      }
39    }
40
41    componentWillUnmount() {
42      this.props.onUnload();
43    }
44
45    render() {
46      const email = this.props.email;
47      const password = this.props.password;
48      const username = this.props.username;
49
50      return (
51        <div className="auth-page">
52          <div className="container page">
53            <div className="row">
54
55              <div className="col-md-6 offset-md-3 col-
                    xs-12">
56                <h1 className="text-xs-center">Sign Up</
                      h1>
57                <p className="text-xs-center">
58                  <Link to="/login">
59                    Have an account?
60                  </Link>
61                </p>
62
63                <ListErrors errors={this.props.errors} /
                      >
64
65                <form onSubmit={this.submitForm(username
                      , email, password)}>
66                  <fieldset>
67
68                    <fieldset className="form-group">
69                      <input
70                        className="form-control form-
                              control-lg"
71                        type="text"
72                        placeholder="Username"
73                        value={this.props.username}
74                        onChange={this.changeUsername} /
                              >
75                    </fieldset>
76
77                    <fieldset className="form-group">
78                      <input
79                        className="form-control form-
                              control-lg"
80                        type="email"
81                        placeholder="Email"
```

```
                        value={this.props.email}
                        onChange={this.changeEmail} />
                    </fieldset>

                    <fieldset className="form-group">
                      <input
                        className="form-control form-
                            control-lg"
                        type="password"
                        placeholder="Password"
                        value={this.props.password}
                        onChange={this.changePassword} /
                            >
                    </fieldset>

                    <button
                      className="btn btn-lg btn-primary
                          pull-xs-right"
                      type="submit"
                      disabled={this.props.inProgress}>
                      Sign up
                    </button>

                  </fieldset>
                </form>
              </div>

            </div>
          </div>
        );
    }
}

export default connect(mapStateToProps,
    mapDispatchToProps)(Register);
```

Listing A.58 – web/Register.js

<!—->

```
import {
  SETTINGS_SAVED,
  SETTINGS_PAGE_UNLOADED,
  ASYNC_START
} from '../constants/actionTypes';

export default (state = {}, action) => {
  switch (action.type) {
    case SETTINGS_SAVED:
      return {
        ...state,
```

```
12          inProgress: false,
13          errors: action.error ? action.payload.errors :
                null
14        };
15      case SETTINGS_PAGE_UNLOADED:
16        return {};
17      case ASYNC_START:
18        return {
19          ...state,
20          inProgress: true
21        };
22      default:
23        return state;
24    }
25  };
```

Listing A.59 – web/settings.js

<!—->

```
1  import ListErrors from './ListErrors';
2  import React from 'react';
3  import agent from '../agent';
4  import { connect } from 'react-redux';
5  import {
6    SETTINGS_SAVED,
7    SETTINGS_PAGE_UNLOADED,
8    LOGOUT
9  } from '../constants/actionTypes';
10
11 class SettingsForm extends React.Component {
12   constructor() {
13     super();
14
15     this.state = {
16       image: '',
17       username: '',
18       bio: '',
19       email: '',
20       password: ''
21     };
22
23     this.updateState = field => ev => {
24       const state = this.state;
25       const newState = Object.assign({}, state, { [
             field]: ev.target.value });
26       this.setState(newState);
27     };
28
29     this.submitForm = ev => {
30       ev.preventDefault();
31
```

```
32        const user = Object.assign({}, this.state);
33        if (!user.password) {
34          delete user.password;
35        }
36
37        this.props.onSubmitForm(user);
38      };
39    }
40
41    componentWillMount() {
42      if (this.props.currentUser) {
43        Object.assign(this.state, {
44          image: this.props.currentUser.image || '',
45          username: this.props.currentUser.username,
46          bio: this.props.currentUser.bio,
47          email: this.props.currentUser.email
48        });
49      }
50    }
51
52    componentWillReceiveProps(nextProps) {
53      if (nextProps.currentUser) {
54        this.setState(Object.assign({}, this.state, {
55          image: nextProps.currentUser.image || '',
56          username: nextProps.currentUser.username,
57          bio: nextProps.currentUser.bio,
58          email: nextProps.currentUser.email
59        }));
60      }
61    }
62
63    render() {
64      return (
65        <form onSubmit={this.submitForm}>
66          <fieldset>
67
68            <fieldset className="form-group">
69              <input
70                className="form-control"
71                type="text"
72                placeholder="URL of profile picture"
73                value={this.state.image}
74                onChange={this.updateState('image')} />
75            </fieldset>
76
77            <fieldset className="form-group">
78              <input
79                className="form-control form-control-lg"
80                type="text"
81                placeholder="Username"
82                value={this.state.username}
83                onChange={this.updateState('username')}
```

```
                    />
84              </fieldset>
85
86              <fieldset className="form-group">
87                <textarea
88                  className="form-control form-control-lg"
89                  rows="8"
90                  placeholder="Short bio about you"
91                  value={this.state.bio}
92                  onChange={this.updateState('bio')}>
93                </textarea>
94              </fieldset>
95
96              <fieldset className="form-group">
97                <input
98                  className="form-control form-control-lg"
99                  type="email"
100                 placeholder="Email"
101                 value={this.state.email}
102                 onChange={this.updateState('email')} />
103             </fieldset>
104
105             <fieldset className="form-group">
106               <input
107                 className="form-control form-control-lg"
108                 type="password"
109                 placeholder="New Password"
110                 value={this.state.password}
111                 onChange={this.updateState('password')}
                      />
112             </fieldset>
113
114             <button
115               className="btn btn-lg btn-primary pull-xs-
                      right"
116               type="submit"
117               disabled={this.state.inProgress}>
118               Update Settings
119             </button>
120
121           </fieldset>
122         </form>
123       );
124     }
125   }
126
127   const mapStateToProps = state => ({
128     ...state.settings,
129     currentUser: state.common.currentUser
130   });
131
132   const mapDispatchToProps = dispatch => ({
```

```
  onClickLogout: () => dispatch({ type: LOGOUT }),
  onSubmitForm: user =>
    dispatch({ type: SETTINGS_SAVED, payload: agent.
        Auth.save(user) }),
  onUnload: () => dispatch({ type:
      SETTINGS_PAGE_UNLOADED })
});

class Settings extends React.Component {
  render() {
    return (
      <div className="settings-page">
        <div className="container page">
          <div className="row">
            <div className="col-md-6 offset-md-3 col-
                xs-12">

              <h1 className="text-xs-center">Your
                  Settings</h1>

              <ListErrors errors={this.props.errors}><
                  /ListErrors>

              <SettingsForm
                currentUser={this.props.currentUser}
                onSubmitForm={this.props.onSubmitForm}
                    />

              <hr />

              <button
                className="btn btn-outline-danger"
                onClick={this.props.onClickLogout}>
                Or click here to logout.
              </button>

            </div>
          </div>
        </div>
      </div>
    );
  }
}

export default connect(mapStateToProps,
    mapDispatchToProps)(Settings);
```

Listing A.60 – web/Settings.js

<!—->

```
import { applyMiddleware, createStore } from 'redux';
```

```
import { createLogger } from 'redux-logger'
import { composeWithDevTools } from 'redux-devtools-
    extension/developmentOnly';
import { promiseMiddleware, localStorageMiddleware }
    from './middleware';
import reducer from './reducer';

import { routerMiddleware } from 'react-router-redux'
import createHistory from 'history/
    createBrowserHistory';

export const history = createHistory();

// Build the middleware for intercepting and
    dispatching navigation actions
const myRouterMiddleware = routerMiddleware(history);

const getMiddleware = () => {
  if (process.env.NODE_ENV === 'production') {
    return applyMiddleware(myRouterMiddleware,
        promiseMiddleware, localStorageMiddleware);
  } else {
    // Enable additional logging in non-production
        environments.
    return applyMiddleware(myRouterMiddleware,
        promiseMiddleware, localStorageMiddleware,
        createLogger())
  }
};

export const store = createStore(
  reducer, composeWithDevTools(getMiddleware()));
```

Listing A.61 – web/store.js

<!—->

```
import {
    SET_PAGE_TRANSACTION,
  HOME_PAGE_LOADED,
  HOME_PAGE_UNLOADED,
  TRANSACTION_UPDATE
} from '../constants/actionTypes';

export default (state = {}, action) => {
  switch (action.type) {
    case SET_PAGE_TRANSACTION:
      return {
        ...state,
        transactions: action.payload.slice(action.page
            * 10, (action.page +1) * 10),
        transactionsCount: action.payload.length,
```

```
        currentPage: action.page
      };
    case HOME_PAGE_LOADED:
      return {
        ...state,
        pager: action.pager[1],
        transactions: action.payload[1],
        transactionsCount: action.payload[1].length,
        currentPage: 0
      };
    case HOME_PAGE_UNLOADED:
      return {};
    case TRANSACTION_UPDATE:
      return {
          ...state,
          pager: action.pager[1],
          transactions: action.payload[0],
          transactionsCount: action.payload[0].length,
          currentPage: 0
      };
    default:
      return state;
  }
};
```

Listing A.62 – web/transactionList.js

<!—->

```
import TransactionPreview from './TransactionPreview';
import TransactionPagination from './
    TransactionPagination';
import React from 'react';

const TransactionList = props => {
  if (!props.transactions) {
    return (
      <div className="article-preview">Loading...</div
          >
    );
  }

  if (props.transactions.length === 0) {
    return (
      <div className="article-preview">
        Nenhuma notifica  o
      </div>
    );
  }

  return (
    <div>
```

```
22          {
23              props.transactions.reverse().map(transaction
                    => {
24              return (
25                <TransactionPreview
26                    transaction={transaction}
27                    key={transaction.transactionHash}
28                />
29              );
30            })
31          }
32
33          {/*<TransactionPagination*/}
34            {/*pager={props.pager}*/}
35            {/*transactionsCount={props.transactionsCount
                  }*/}
36            {/*currentPage={props.currentPage} />*/}
37        </div>
38    );
39  };
40
41  export default TransactionList;
```

Listing A.63 – web/TransactionList.js

<!—->

```
1   import React from 'react';
2   import agent from '../agent';
3   import { connect } from 'react-redux';
4   import { SET_PAGE_TRANSACTION } from '../constants/
        actionTypes';
5
6   const mapDispatchToProps = dispatch => ({
7     onSetPage: (page, payload) =>
8       dispatch({ type: SET_PAGE_TRANSACTION, page,
            payload })
9   });
10
11  const TransactionPagination = props => {
12    if (props.transactionsCount <= 10) {
13      return null;
14    }
15
16    console.log('tcount');
17    console.log(props.transactionsCount);
18    const range = [];
19    for (let i = 0; i < Math.ceil(props.
          transactionsCount / 10); ++i) {
20      range.push(i);
21    }
22
```

```
23    const setPage = page => {
24      if(props.pager) {
25        props.onSetPage(page, props.pager(page));
26      }else {
27        props.onSetPage(page, agent.Transactions.
              pastTransactions())
28      }
29    };
30
31    return (
32      <nav>
33        <ul className="pagination">
34
35          {
36            range.map(v => {
37              const isCurrent = v === props.currentPage;
38              const onClick = ev => {
39                ev.preventDefault();
40                setPage(v);
41              };
42              return (
43                <li
44                  className={ isCurrent ? 'page-item
                        active' : 'page-item' }
45                  onClick={onClick}
46                  key={v.toString()}>
47
48                  <a className="page-link" href="">{v +
                        1}</a>
49
50                </li>
51              );
52            })
53          }
54
55        </ul>
56      </nav>
57    );
58  };
59
60  export default connect(() => ({}), mapDispatchToProps)
        (TransactionPagination);
```

Listing A.64 – web/TransactionPagination.js

<!—->

```
1  import React from 'react';
2  import { connect } from 'react-redux';
3  import ReactJson from 'react-json-view';
4
5  const mapDispatchToProps = ()=> ({
```

```
});

const TransactionPreview = props => {
    const transaction = props.transaction;

    return (
        <div className="article-preview">
            <ReactJson name={transaction.
                transactionHash}
                       src={transaction}
                       theme="bright:inverted"
                       displayDataTypes={false}
                       displayObjectSize={false}
                       indentWidth={2}
                       collapsed={0}
                       collapseStringsAfterLength={20}
                       enableClipboard={false}
                       style={ {
                           fontWeight: "300",
                           fontSize: "0.7rem",
                           paddingTop: "0px",
                           paddingBottom: "0px"}}
            />
        </div>
    );
}

export default connect(() => ({}), mapDispatchToProps)
    (TransactionPreview);
```

Listing A.65 – web/TransactionPreview.js

<!—->

```
import React from 'react';
import { connect } from 'react-redux';
import TransactionList from "../TransactionList";

const mapStateToProps = state => ({
  ...state.transactionList
});

const mapDispatchToProps = dispatch => ({
});

const TransactionView = props => {

  return (
    <div className="col-md-9">
      <TransactionList
        pager={props.pager}
        transactions={props.transactions}
```

```
        loading={props.loading}
        transactionsCount={props.transactionsCount}
        currentPage={props.currentPage} />
    </div>
  );
};

export default connect(mapStateToProps,
    mapDispatchToProps)(TransactionView);
```

Listing A.66 – web/TransactionView.js

<!—->