

SMART CONTRACT

SEGURO DE GRUPOS MÚTUOS COM BLOCKCHAIN

COOVER

INSTITUTO DE TECNOLOGIA E LIDERANÇA – INTELI

SMART CONTRACT

SEGURO DE GRUPOS MÚTUOS COM BLOCKCHAIN COOVER

Autores: Eric Tachdjian

Giovanna Furlan Torres

Henri Harari

Maria Luísa Vilaronga Maia

Matheus Fidelis

Ueliton Moreira Rocha

Data de criação: 20 de Fevereiro de 2023

SÃO PAULO – SP

2023

Sumário

Controle de Documento.....	5
1. Introdução.....	6
1.1 Smart Contracts.....	6
1.2 Registro Smart Contract.....	6
1.3 Partes Interessadas.....	7
2. Requisitos de Negócio.....	8
3. Arquitetura – Diagrama de Blocos.....	11
4. Diagrama de sequência UML.....	13
4.1 UML – User Story 1 e 2.....	13
4.2 UML – User Story 3 e 4.....	14
4.3 UML – User Story 5.....	15
5. Resultados.....	17
5.1 Relação User Story e Função Smart Contract.....	17

Índice de figuras

Figure 1: Diagrama de blocos.....	12
Figura 2: Diagrama de sequência UML - User Story 1 e 2.....	13
Figura 3: Diagrama de sequência UML - User Story 3 e 4.....	14
Figura 4: Diagrama de sequência UML - User Story 5.....	15
Figura 5: Adição de usuário no seguro.....	17
Figura 6: Viabilidade do seguro.....	18
Figura 7: Cobrar valor de transferência.....	18
Figura 8: Remover Usuário.....	19
Figura 9: Renovar Contrato.....	19
Figura 10: Pedido de Indenização.....	20
Figura 11: Recebimento de pagamento.....	20
Figura 12: Reposição de reserva de risco.....	21

Índice de tabelas

Table 1: Controle de documento.....	6
-------------------------------------	---

Controle de Documento

Histórico de Revisões

Table 1: Controle de documento

Data	Autor	Versão	Resumo da Atividade
20/02/2023	Giovanna Furlan	1	Criação do documento Tópicos 1, 2, 3, 4 e 5

1. Introdução

A Coover é uma seguradora, que oferece seguros mútuos de celulares, inserida em um mercado com ótima oportunidade de crescimento, sendo que apenas 30% dos brasileiros possuem tal serviço que atua na segurança pessoal e de bens materiais. A busca por maior facilidade e adesão dos usuários para contratação de seguros celulares é através da tecnologia de *Smart Contracts* via rede blockchain, possibilitando uma solução confiável e descentralizada.

1.1 Smart Contracts

Um smart contract funciona como um acordo entre as partes interessadas da contratação, sendo um programa de computador, que ao inserir as cláusulas executa automaticamente, para validar se os termos exigidos, estão sendo cumpridos. Sua execução, ocorre via rede blockchain, a qual pode-se definir como uma rede descentralizada, que visa proporcionar maior segurança e transparência aos seus usuários, armazenando informações e realizando transações.

O contrato é programado em solidity, que recebe comandos referentes ao administrador do contrato e funções que remetem as condições que precisam ser atendidas. Após estabelecidos os termos e obrigações contratuais, os mesmos são apresentados de forma transparente e imutável para os contratantes.

1.2 Registro Smart Contract

O acesso ao contrato após sua publicação, é somente para visualização, uma vez que as informações constadas são imutáveis. Sendo assim, se o documento for publicado com algum erro ou problema contratual, a única forma de corrigi-lo é criar um contrato novo. Por essa razão, apresenta-se neste documento a documentação prevista para a construção do smart contract da Coover. Inclui-se neste documento informações sobre as partes envolvidas, as condições e/os requisitos do contrato, diagramações e as regras de execução do contrato. Espera-se que as partes interessadas entendam plenamente o contrato e as suas implicações.

1.3 Partes Interessadas

As principais partes interessadas são a Coover (seguradora), os titulares dos celulares segurados e a rede de blockchain. Cada parte executa um papel fundamental para a execução do processo de seguro celular, lista-se abaixo a descrição de participação, presando pela transparência e eficiência do processo.

1. Seguradora: É o administrador do contrato (smart contract), responsável por sua criação e estabelecimento das condições do seguro mútuo, tais como o gerenciamento dos pedidos de indenizações, as coberturas e as regras de execução do contrato.
2. Titulares dos celulares segurados: Usuários que adquirem o seguro mútuo da Coover e são as partes que prezam pela proteção fornecida pelo smart contract. Ao pagar o valor acordado do seguro, possuem direito à indenizações caso ocorra um sinistro coberto pelo contrato.
3. Rede de blockchain: Possui a responsabilidade do armazenamento e execução do smart contract, de forma automática e segura. Além de fornecer a garantia de que as cláusulas contratuais sejam seguidas de forma imutável e transparente.

2. Requisitos de Negócio

Nesta sessão apresenta-se os requisitos de negócios, que são descrições claras e precisas das necessidades e expectativas da empresa em relação ao projeto. Tais condições colaboram para o processo de desenvolvimento, e o smart contract as seguirá quando for acionado.

- **Requisito 1:** Armazenar a quantidade de usuários atualmente no contrato;

O objetivo é armazenar a quantidade atual de usuários participantes do grupo de seguros mútuos ao qual pertence tal smart contract. O intuito é que o contrato gerencie a quantidade de usuários que estão usando o serviço de seguro, de acordo com a quantidade estipulada nos termos.

- **Requisito 2:** Armazenar a data de validade do contrato;

Armazenamento da data de validade do contrato no smart contract, é utilizado para que o contrato possa verificar se ele ainda é válido. Uma vez que ativo, e a data de validade em vigor, o contrato poderá deixar de permitir novos usuários e/ou realizar outras ações que envolvam o contrato, como iniciar os pedidos de aceite para liberação de indenizações.

- **Requisito 3:** Armazenar a quantidade mínima e máxima de usuários permitidas;

O objetivo é armazenar a quantidade mínima e máxima de usuários permitida no grupo de seguro mútuo de celulares no smart contract. Tais valores auxiliam a execução do primeiro requisito, atuando no controle da quantidade de usuários que podem participar do seguro, de acordo com o range cadastrado, garantindo o gerenciamento da quantidade de riscos assumidos pela seguradora em cada grupo.

- **Requisito 4:** Armazenar a lista de usuários que aceitaram o termo e estão ativos;

O armazenamento de usuários ativos no contrato permite a relação com a carteira de Criptomoedas associada, garantindo que o pagamento de sinistros de seguros sejam pagos corretamente.

- **Requisito 5:** Armazenar a lista de carteiras de usuários;

Relacionado ao requisito 4, o armazenamento das carteiras permite que o contrato verifique a origem do pagamento de um usuário em particular, para o gerenciamento do pagamento e recebimento da mensalidade atribuída aquele grupo, visando contabilizar quais contas já fizeram o depósito e estão liberadas para participarem do seguro.

- **Requisito 6:** Adicionar um novo usuário ao seguro;

Apresentando ligação com o requisito 3, se o contrato estiver com a quantidade de pessoas ainda dentro do range estipulado para seu funcionamento, ocorre a adição de novos usuários ao seguro por meio do smart contract.

- **Requisito 7:** Verificar a viabilidade do contrato;

Correlacionado ao requisito 2, entende-se que o contrato tem um tempo de vigência estipulado, portanto tal funcionalidade verifica a viabilidade do contrato, oferecendo a possibilidade de renovação quando a data de expiração estiver próxima, além de remover os usuários que não desejam continuar, abrindo vaga para novos contratantes, neste mesmo grupo.

- **Requisito 8:** Cobrar um valor do usuário para entrar no seguro;

Quando o usuário escolhe um grupo de seguro para participar, está ciente que um valor deve ser paga para que seu dispositivo seja coberto pelo seguro. Tal funcionalidade garante que esse valor será cobrado das carteiras cadastradas, permitindo um gerenciamento adequado do risco e do valor que é necessário arrecadar para pagar eventuais indenizações.

- **Requisito 9:** Remover um usuário do seguro;

A função de remover um usuário do seguro, está conectada com o requisito 7, uma vez que o usuário até então possui a possibilidade de sair do grupo, quando a vigência acaba e não ocorre a validação, sendo feita automaticamente por meio do smart contract.

- **Requisito 10:** Pedido de indenização;

O objetivo é permitir que um usuário possa solicitar indenização do seguro por meio do smart contract em caso de algum acontecimento envolvendo seu aparelho cadastrado. O contrato pode verificar se o usuário tem direito à indenização e dinheiro depositado, e assim, efetuar o pagamento correspondente.

- **Requisito 11:** Reposição da reserva de risco;

Quando ocorre uma retirada de sinistro, o valor deve ser recolocado na conta principal do grupo, garantindo que o contrato possa arcar com eventuais indenizações dos usuários sem precisar fazer aporte adicional de recursos.

- **Requisito 12:** Criação do smart contract e aprovações do seguro;

O requisito principal que envolve o administrador do contrato, no caso a Coover, a qual lhe dá a responsabilidade de criação do smart contract e fornecimento de aprovações necessárias para o andamento do contrato.

3. Arquitetura – Diagrama de Blocos

Nesta sessão, apresenta-se a diagramação em blocos, descrevendo cada etapa para o cumprimento dos requisitos listados acima. Considera-se um diagrama de blocos, uma representação visual da estrutura e funcionamento de um sistema, aos quais contém os componentes do sistema e linhas para representar as conexões ou interações entre eles. Com eles, procura-se entender a estrutura geral do sistema e as relações entre os diferentes componentes.

Para a construção do diagrama previsto para a solução, entende-se como partes principais cinco atores, sendo eles:

- **Clientes:**

Os clientes são os proprietários dos celulares que adquirem o seguro mútuo da Coover. Fornecendo informações sobre o dispositivo, realizando as transações entre as carteiras e aceitando os termos do contrato inteligente.

- **Plataforma Cliente:**

A plataforma do cliente é a interface utilizada para adquirir e gerenciar os seguros mútuos contratados. Fornece uma comunicação com a plataforma da seguradora que prossegue com as solicitações necessárias para dar andamento ao contrato, sinistros e transações.

- **Plataforma Seguradora (API da Seguradora):**

A plataforma da seguradora, é o componente que faz a ponte entre o aplicativo e os Smart Contracts em blockchain. Recebendo as solicitações dos clientes, consultando as informações dos contratos inteligentes na blockchain e realizando as transações necessárias, ou seja, conta com todo o gerenciamento do seguro.

- **Smart Contracts em Blockchain:**

São onde os termos do contrato são estipulados, gerenciando todas as regras de negócio do seguro mútuo. Executados na rede blockchain e contém todas as cláusulas e condições acordadas entre as partes. Além disso, gerenciam o

pagamento de indenizações e garantem transparência e segurança na execução do contrato.

- **Redes Blockchain:**

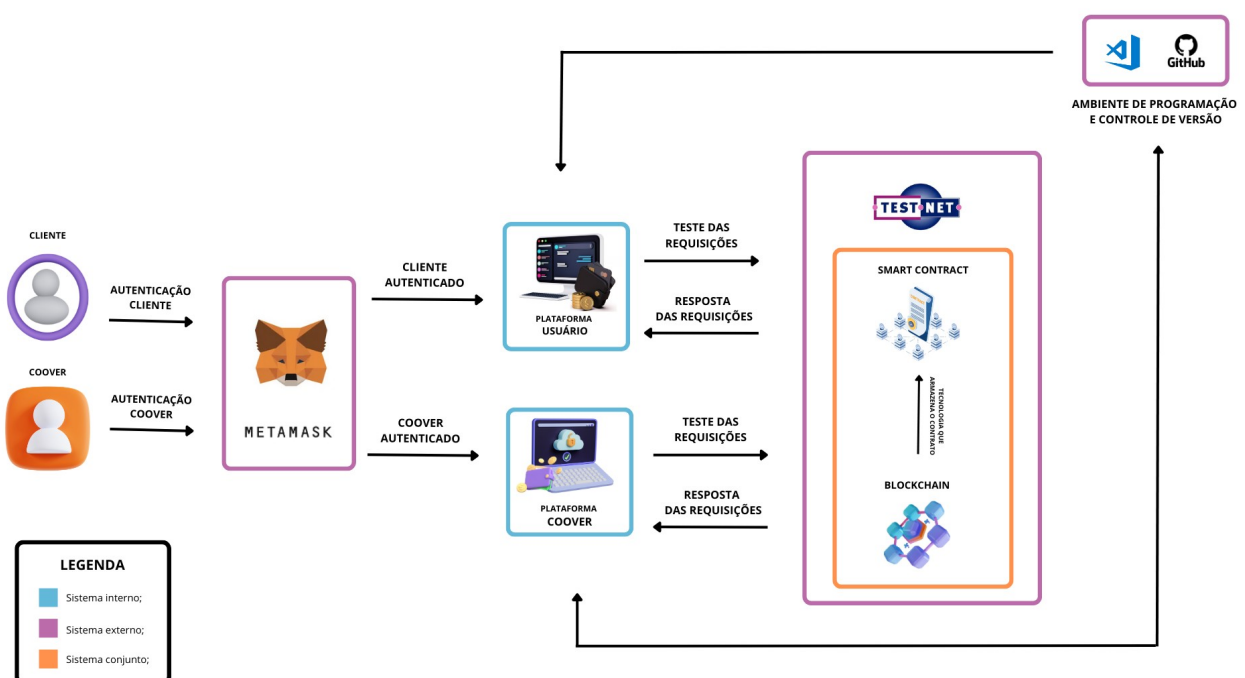
Rede responsável por armazenar os Smart Contracts, responsável por garantir segurança e a imutabilidade dos registros, além de que as transações sejam realizadas sem a necessidade de um intermediário.

- **TestNet:**

Tal etapa entra na diagramação como prevenção de erros e sem riscos de gastos com recursos financeiros, sendo uma rede usada para testar e validar as funcionalidades do contrato, antes de ser enviado para a rede principal (mainnet).

A construção do diagrama se baseia na utilização lógica das tecnologias envolvidas, que seguem o mesmo fluxo para a implementação de quaisquer requisito, dentre os citados anteriormente. Para a solução do problema, desenhou-se o seguinte diagrama, apresentado na figura 1.

Figure 1: Diagrama de blocos



4. Diagrama de sequência UML

Nesta sessão, apresenta-se os diagrama de sequência UML, representação gráfica de interação entre objetos e sistema. Com este, visualiza-se a sequência de eventos que ocorrem durante um processo.

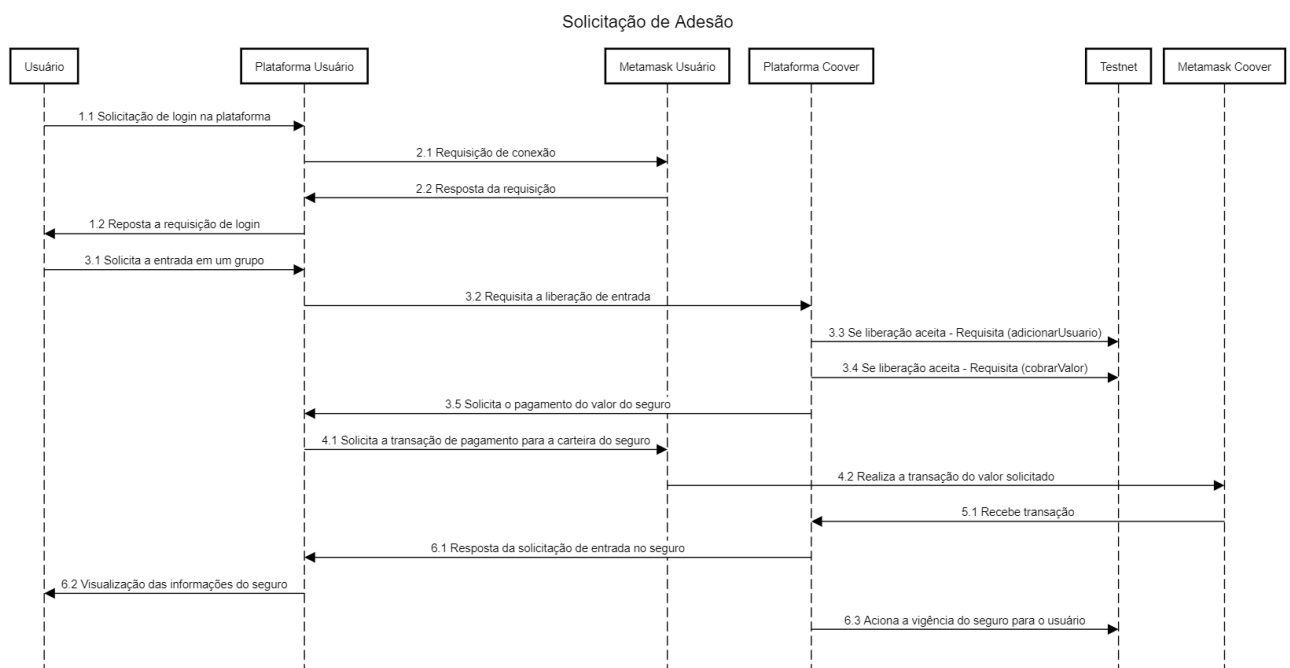
4.1 UML – User Story 1 e 2

Descrição das user stories utilizadas para a criação do diagrama de sequência UML.

- **Primeira User Story:** Eu, como usuário insatisfeito de uma seguradora, quero adquirir um novo seguro Coover, para que eu consiga proteger meu celular de forma mais eficiente.
- **Segunda User Story:** Eu, como colaborador da Coover, posso aprovar ou não usuários para a plataforma, para garantir a adesão de clientes aptos.

O diagrama tem por objetivo representar as sequências de eventos que ocorrem durante o processo de aquisição e aceite de um novo seguro Coover e colabora para o entendimento das responsabilidades de cada parte envolvida no processo. Exibido na figura 2 abaixo.

Figura 2: Diagrama de sequência UML - User Story 1 e 2



Fonte: Autoria própria

O diagrama de sequência UML, interage com o sistema da Coover por uma interface de usuário. Neste, através do sistema utilizado, ocorre uma solicitação de adesão para ingresso no seguro celular. O colaborador pode aprovar ou rejeitar o usuário para a plataforma, e o sistema envia uma notificação para o usuário com a resposta da requisição. Se aprovado realiza o pagamento do valor solicitado para a carteira do seguro em ETH, e o smart contract adiciona o usuário na lista de participantes daquele grupo.

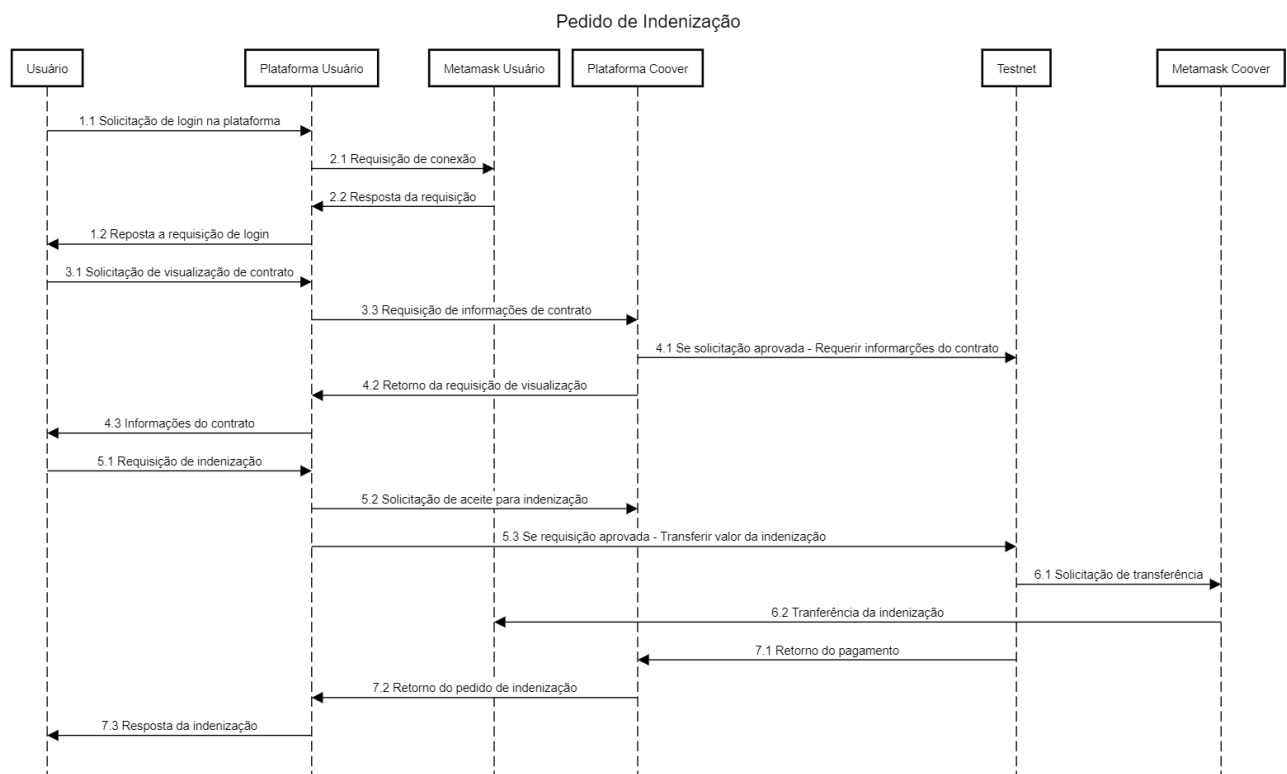
4.2 UML – User Story 3 e 4

Descrição das user stories utilizadas para a criação do diagrama de sequência UML.

- **Terceira User Story:** Eu, como usuário da Coover, desejo pedir indenização, para que eu possa utilizar o dinheiro do sinistro para reparar o dano.
- **Quarta User Story:** Eu, como colaborador da Coover, desejo aprovar ou não indenizações, para que o sistema seja funcional.

O diagrama apresentado na figura 3, representa a sequência de eventos que ocorrem durante o processo de pedido de indenização e visualização de informações do seguro.

Figura 3: Diagrama de sequência UML - User Story 3 e 4



Fonte: Autoria própria

Nesse diagrama, o usuário interage com o sistema da Coover selecionando a opção de pedir indenização com os detalhes do sinistro. A Coover verifica as informações do sinistro. Se o pedido for aprovado, o sistema envia a indenização para a wallet do usuário e notifica o usuário sobre a aprovação e o pagamento da indenização, além de ser possível visualizar as informações do grupo, a qual participa, antes de pedir a indenização.

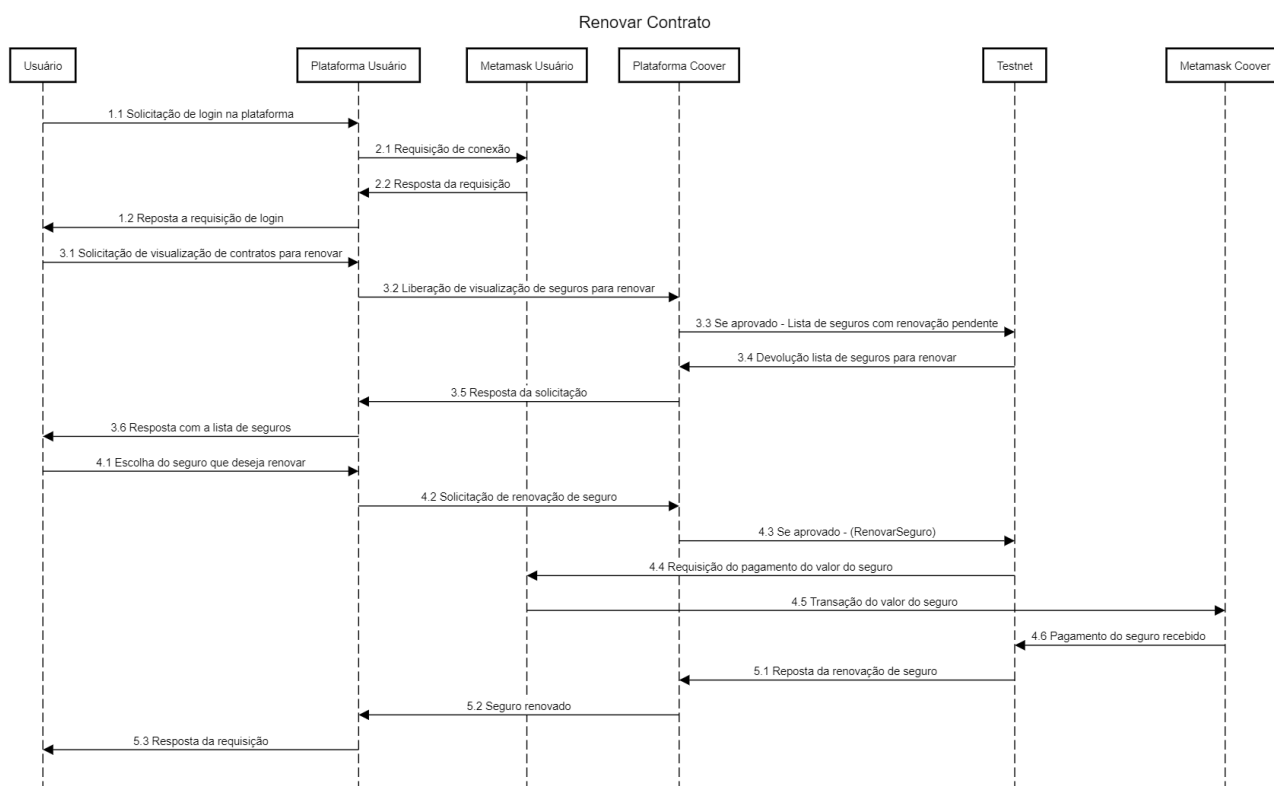
4.3 UML – User Story 5

Descrição das user stories utilizadas para a criação do diagrama de sequência UML.

- **Quinta User Story:** Eu, como usuário da Coover, desejo renovar meus contratos, para que eu siga tendo proteção no meu celular.

O diagrama apresentado na figura 4, representa a sequência de eventos que ocorrem durante o processo de renovação de seguro celular.

Figura 4: Diagrama de sequência UML - User Story 5



Fonte: Autoria própria

Nesse diagrama, o usuário acessa a plataforma da Coover e seleciona o contrato que deseja renovar. A aprovação é dada pela Coover, que requisita ao smart contract, e se faz necessário o pagamento do seguro para renovação. A Coover confirma a renovação do contrato e o sistema gera um novo contrato com as informações atualizadas.

5. Resultados

Nessa sessão apresenta-se até o momento a relação construída entre o smart contract e as user stories, além das funções que já estão implementadas, com os seus respectivos resultados, e as que estão em processo de desenvolvimento.

5.1 Relação User Story e Função Smart Contract

As user stories são utilizadas para descrever os requisitos do usuário dentro do projeto. Colaboram para as definições das funcionalidades que serão implementadas, tornando mais fácil para a equipe de desenvolvimento entender as necessidades do usuário e criar soluções que atendam essas necessidades.

Em conjunto com as funções do smart contract as user stories definem as funcionalidades que o usuário deseja que o smart contract execute, e as funções do smart contract são a implementação dessas funcionalidades em código. Abaixo, apresenta-se a relação entre as user stories e as funções do smart contract prevista para a construção deste projeto.

- **Primeira:** Adicionar um usuário;
 - A função apresentada abaixo é responsável por inserir um usuário no seguro e sua ligação se dá precisamente com a primeira user story descrita.

Figura 5: Adição de usuário no seguro

```
// adiciona um novo usuário ao contrato

// address - tem que ser o hash da carteira
function adicionarUsuario(address usuario) public onlyOwner {
    // Verifica se o contrato está ativo
    require(viabilidadeContrato() == 1, "O contrato está ativo.");

    // Verifica se o usuário já está na lista de carteiras
    for (uint i = 0; i < carteira.length; i++) {
        require(carteira[i].carteiraUsuario != usuario, "O usuário já está na lista de carteiras.");
    }

    // Adiciona o usuário à lista de carteiras
    carteira.push(Carteira(usuario));

    if(quantUsuario > minPessoas && quantUsuario < maxPessoas){
        // Atualiza o número de usuários
        quantUsuario++;
    }
}
```

Fonte: Autoria própria

- **Segunda:** Viabilidade do contrato;
 - A função apresentada abaixo é responsável por verificar sua viabilidade de execução e sua ligação se da precisamente como critério de aceitação em todas as user stories descritas.

Figura 6: Viabilidade do seguro

```
function viabilidadeContrato() public view returns (uint) {
    if (quantUsuario >= minPessoas && block.timestamp <= dataValidade && quantUsuario <= maxPessoas) {
        return 1; // Contrato Ativo
    } else if (quantUsuario < minPessoas && block.timestamp <= dataValidade) {
        return 2; // Contrato em Progresso
    } else if (block.timestamp > dataValidade && quantUsuario < minPessoas) {
        return 3; // Contrato Inativo
    } else {
        revert("Erro ao verificar o contrato");
    }
}
```

Fonte: Autoria própria

- **Terceira:** Valor pago do seguro;
 - A função apresentada abaixo é responsável por cobrar o valor do seguro e sua ligação se da precisamente como critério de aceitação nas user stories que precisam de quaisquer transferências, sendo as 1, 3 e 5.

Figura 7: Cobrar valor de transferência

```
function cobrarValor(uint valor) public {
    valor = 10;

    // Verifica se o contrato está ativo
    require(viabilidadeContrato() == 1, "O contrato esta ativo.");

    // Calcula o valor total a ser cobrado
    uint valorTotal = valor * quantUsuario;
    // Calcula a comissão do dono do contrato
    uint comissao = (valorTotal * 2) / 100;

    // Verifica se o valor total é compatível com o saldo da carteira principal
    require(carteiraPrincipal >= comissao, "Saldo insuficiente na carteira principal.");

    // Divide o valor total entre os usuários do contrato
    uint valorPorUsuario = (valorTotal - comissao) / quantUsuario;
    // Cobre o valor de cada usuário
    for (uint i = 0; i < carteira.length; i++) {
        carteira[i].saldo -= valorPorUsuario;
    }
    // Transfere a comissão para o dono do contrato
    for (uint i = 0; i < carteira.length; i++) {
        if (carteira[i].carteiraUsuario == owner) {
            carteira[i].saldo += comissao;
            break;
        }
    }
    carteiraPrincipal -= comissao;
}
```

Fonte: Autoria própria

- **Quarta: Remover Usuário;**
 - A função apresentada abaixo é responsável remover um usuário do seguro, caso o mesmo não realize o pagamento ou não renove o contrato e sua ligação se dá indiretamente com a user story 4.

Figura 8: Remover Usuário

```
function removerUsuario(address usuario) public onlyOwner {
    // Verifica cada carteira do projeto
    for (uint i = 0; i < carteira.length; i++) {
        // Verifica se a carteira do usuário corresponde ao endereço fornecido
        if (carteira[i].carteiraUsuario == usuario) {
            // Remove o usuário da lista de carteiras
            delete carteira[i];
            // Atualiza o número de usuários
            quantUsuario--;
            // Sai do loop
            break;
        }
    }
}
```

Fonte: Autoria própria

- **Quinta: Renovar contrato;**
 - A função apresentada abaixo é responsável por renovar a validade do seguro contratado sua ligação se dá diretamente com a user story 5.

Figura 9: Renovar Contrato

```
// Função para renovar o contrato (USER STORY 5)
function renovarContrato(uint _novaDataValidade) public onlyOwner {
    // Verifica se a nova data de validade é no futuro
    require(_novaDataValidade > block.timestamp, "A nova data de validade deve ser no futuro.");

    // Cria um array para armazenar os índices dos usuários que não aceitaram o novo termo
    uint[] memory indicesRemover = new uint[](quantUsuario);
    uint quantRemover = 0;

    // Verifica se cada usuário aceitou o novo termo, adicionando o índice à lista de remoção, caso contrário
    for (uint i = 0; i < quantUsuario; i++) {
        if (!termoAceito[carteira[i].carteiraUsuario]) {
            indicesRemover[quantRemover] = i;
            quantRemover++;
        }
    }

    // Remove os usuários que não aceitaram o novo termo
    for (uint i = 0; i < quantRemover; i++) {
        removerUsuario(carteira[indicesRemover[i]].carteiraUsuario);
    }

    // Verifica se a quantidade de usuários é compatível com o mínimo e o máximo definidos no contrato
    require(quantUsuario >= minPessoas && quantUsuario <= maxPessoas, "A quantidade de usuarios nao e compativel com o minimo e o maximo definidos no contrato.");

    // Define a nova data de validade
    dataValidade = _novaDataValidade;
}
```

Fonte: Autoria própria

- **Sexta:** Pedido de Indenização;
 - A função apresentada abaixo é responsável por viabilizar o pedido de indenização e sua ligação se da diretamente com as user stories 3 e 4.

Figura 10: Pedido de Indenização

```
function pedirIndenizacao(uint valor) public {  
  
    // Verifica se o contrato está ativo  
    require(viabilidadeContrato() == 1, "O contrato esta ativo.");  
  
    // Verifica se o usuário está na lista de carteiras  
    for (uint i = 0; i < carteira.length; i++) {  
  
        if (carteira[i].carteiraUsuario == msg.sender) {  
  
            // Verifica se o usuário tem saldo suficiente  
            require(carteira[i].saldo >= valor, "O usuario nao tem saldo suficiente.");  
  
            // Verifica se o valor solicitado é menor ou igual ao valor do celular do usuário  
            require(valor <= carteira[i].valorCelular, "O valor solicitado e maior que o valor do celular do usuario.");  
  
            // Adiciona o valor ao saldo do usuário  
            carteira[i].saldo -= valor;  
            break;  
        }  
    }  
}
```

Fonte: Autoria própria

- **Sétima:** Recebimento de pagamento;
 - A função apresentada abaixo é responsável receber as transações dos usuários e sua ligação se da indiretamente com qualquer user story que seja necessário o recebimento de transação pela metamask.

Figura 11: Recebimento de pagamento

```
function receberPagamento(uint valor) public onlyOwner {  
    // Verifica se o contrato está ativo  
    require(viabilidadeContrato() == 1, "O contrato esta ativo.");  
  
    // Verifica se o usuário está na lista de carteiras  
    for (uint i = 0; i < carteira.length; i++) {  
        // Adiciona o valor à carteira principal  
        carteiraPrincipal += valor;  
    }  
}
```

Fonte: Autoria própria

- **Oitava:** Reposição de reserva de risco;
 - A função apresentada abaixo é responsável pela reposição de reserva de risco quando o valor é retirado do para um rateio e/ou indenização pessoal e sua ligação se da indiretamente com o critério de pagamentos da user story 1 e 2, para aceite no contrato.

Figura 12: Reposição de reserva de risco

```
function reposicaoReservaRisco(uint valor) public onlyOwner {
    // Verifica se o contrato está ativo
    require(viabilidadeContrato() == 1, "O contrato esta ativo.");

    // Verifica se o valor é menor ou igual ao saldo da carteira principal
    require(valor <= carteiraPrincipal, "O valor e maior que o saldo da carteira principal.");

    // Adiciona o valor à carteira principal
    carteiraPrincipal -= valor;
}
```

Fonte: Autoria própria

Para a entrega da Sprint 2, somente algumas funções foram completamente implementadas, e serão apresentadas separadamente em um arquivo, diferenciando-se das que ainda estão em andamento. O arquivo que contém as quatro funções já em funcionamento, também possui o deploy realizado com sucesso no remix, sendo elas: 1) função join; 2) função balance; 3) função remove user; e 4) Viabilidade do contrato.

Lista-se abaixo onde encontrar os respectivos arquivos no GitHub, estando presente dentro da pasta contrato.

- Nome do arquivo com as funções implementadas: *codigoImplementado.sol*
- Nome do arquivo com as funções em desenvolvimento: *codigoDesenvolvimento.sol*