**Modelagem de Casos de Uso**

**Documentação do modelo de casos de uso (documentação de visão)**

Nesse sistema de atendimento automatizado, identificamos os seguintes atores:

***-Desenvolvedor Java:*** Indivíduo que calculará a decisão do modal, porém, no todo, irá fazer um diagrama de classes, possuindo todas as classes necessárias para o sistema, adicionando atributos, construtores e métodos, e usando a Programação Orientada a Objetos para devolver o tipo de guincho adequado

***-Desenvolvedor de outras áreas:*** Indivíduo que herdará o Desenvolvedor Java caso o mesmo se ausente por motivos de doença, eventos muito importantes, etc.

.

**Diagrama de Caso de uso 1: Decidir Modal**

**Diagrama

Descrição gerada automaticamente**

**Descrição da situação:**

O estudo de caso é sobre um sistema que visa efetuar um método diferente para ser mais assertivo na escolha do guincho baseada na situação do usuário, usando tecnologia.

**Após o levantamento de requisitos iniciais do sistema, os analistas chegaram a seguinte lista de requisitos funcionais:**

**R1:** O desenvolvedor Java deve fazer o programa para verificar se o tipo de carroceria do usuário se adequa ao guincho pesado

**R2:** O desenvolvedor Java deve fazer o programa para verificar se existem modificações que o usuário fez que possam auxiliar na decisão

**R3:** O desenvolvedor Java deve fazer o programa para verificar se a distância necessária é adequada ao guincho pesado

**R4:** O desenvolvedorJava deve fazer o programa para verificar se o peso do veículo com carga se adequa ao guincho pesado

**R5:** O desenvolvedor deve criar um programa para pegar todos os resultados anteriores e com isso, verificar se o guincho é pesado

**R6:** O desenvolvedor Java deve retornar essa informação para o sistema

**Regras de negócio:**

RN01 – O programa só deverá funcionar caso o usuário já estiver colocado as informações anteriormente

RN02 – Se não existir modificações, devolver como nulo

RN03 – A distância será puxada usando outro sistema, já previsto em Requisitos Não Funcionais

RN04 – O tipo de guincho poderá ter dois tipos: pesado ou não-pesado

RN05 – Definir tipo de guincho como pesado se ao menos 2 valores necessitarem de guincho pesado

RN06 – O retorno do guincho deve ser feito ao final do programa

**Verificar se o tipo de carroceria se adequa ao guincho pesado – CSUO1**

**Sumário:** Um programa que definirá um dos parâmetros essenciais para decidir o tipo de guincho

**Ator Primário:** Desenvolvedor Java.

**Atores Secundários:** Desenvolvedor das outras áreas.

**Precondições:** O usuário está reconhecido com suas informações pelo sistema, deve ter acionado o sistema de guincho e colocado a informação do tipo da carroceria

**Fluxo Principal:**

1. O programa irá puxar a informação que o usuário colocou

2. Essa informação cairá em um laço de repetição onde terá uma lista de reconhecimento

3. O programa irá verificar se o tipo de carroceria é equivalente ao da nossa lista

4. Ele irá retornar a adequação ao guincho pesado

**Fluxo Alternativo:** *Não compatível com a lista*

a) Ele irá retornar uma mensagem dizendo que não é adequado ao guincho pesado e sairá do código

**Fluxo de exceção:** *Violação da Regra de Negócio 1: não ter enviado as informações*

1. Se o usuário não tiver enviado as informações, o sistema vai devolver a informação como nulo, previsto na Regra de Negócio 2, e vai emitir uma mensagem dizendo que não poderá ser concluído a decisão do modal

**Pós-condições:** O programa irá automaticamente ir para a próxima verificação

**Regra de Negócio:** RN01, RN02

**Verificar se existem modificações compatíveis com o modal pesado – CSUO2**

**Sumário:** Um programa que pegará todas as modificações, e automaticamente, devolver um parâmetro que será necessário para decidir se o guincho é pesado

**Ator Primário:** Desenvolvedor Java.

**Atores Secundários:** Desenvolvedor das outras áreas.

**Precondições:** O programa já deve ter verificado o tipo de carroceria

**Fluxo Principal:**

1. O programa irá puxar todas as informações que o usuário colocou

2. O programa colocará em uma lista

3. O programa vai verificar, elemento por elemento, quais deles que possam alterar as modificações

4. Se existir ao menos duas modificações, devolver que nesse quesito, está aprovado o uso de guincho pesado

**Fluxo Alternativo:**

1. *Não compatível com a lista (Passo 3)*: Ele irá retornar uma mensagem dizendo que não é adequado ao guincho pesado e sairá do código.
2. Não ter modificações (Passo 1): O programa automaticamente irá devolver que não será preciso o guincho pesado nesse quesito
3. Tiver uma modificação só (Passo 4): O programa irá devolver que não será preciso o guincho pesado nessa situação

**Fluxo de exceção:** *Violação da Regra de Negócio 1: não ter enviado as informações*

1. Se o usuário não tiver enviado as informações, o sistema vai devolver a informação como nulo, previsto na Regra de Negócio 2, e vai emitir uma mensagem dizendo que não poderá ser concluído a decisão do modal

**Pós-condições:** O programa irá automaticamente ir para a próxima verificação

**Regra de Negócio:** RN01, RN02

**Verificar se a distância é compatível com o modal pesado – CSUO3**

**Sumário:** Um programa que verifica a distância e devolve sua compatibilidade com a decisão

**Ator Primário:** Desenvolvedor Java.

**Atores Secundários:** Desenvolvedor das outras áreas.

**Precondições:** A distância deve ter sido calculada por outro sistema

**Fluxo Principal:**

1. O programa irá pegar a distância do outro sistema

2. O programa irá comparar com a distância média necessária de um guincho pesado

3. Se for na média ou acima, irá devolver que precisa de um guincho pesado

**Fluxo Alternativo:**

1. *Abaixo da média (Passo 3)*: Ele irá retornar uma mensagem dizendo que não é adequado ao guincho pesado e sairá do código.

**Fluxo de exceção:** *Erro no cálculo da distância*

1. Se a distância não tiver sido calculada, o sistema vai devolver a informação como nulo, previsto na Regra de Negócio 2, e vai emitir uma mensagem dizendo que não poderá ser concluído a decisão do modal

**Pós-condições:** O programa irá automaticamente ir para a próxima verificação

**Regra de Negócio:** RN03

**Verificar se o peso do veículo com carga é adequado – CSUO4**

**Sumário:** Um programa que verifica o peso do veículo com carga e devolve sua compatibilidade com a decisão

**Ator Primário:** Desenvolvedor Java.

**Atores Secundários:** Desenvolvedor das outras áreas.

**Precondições:** O usuário deve ter colocado essa informação

**Fluxo Principal:**

1. O programa irá pegar o peso do veículo com a carga

2. O programa irá comparar com o capacidade do guincho pesado

3. O programa irá efetuar um cálculo de subtração

4. Se der abaixo ou igual a um número aproximado que estará no nosso sistema, ele devolverá que precisa de um guincho pesado

**Fluxo Alternativo:**

1. *Acima do número (Passo 4)*: Ele irá retornar uma mensagem dizendo que não é adequado ao guincho pesado e sairá do código.

**Fluxo de exceção:** *Violação da Regra de Negócio 1: não ter enviado as informações*

1. Se o usuário não tiver enviado as informações, o sistema vai devolver a informação como nulo, previsto na Regra de Negócio 2, e vai emitir uma mensagem dizendo que não poderá ser concluído a decisão do modal

**Pós-condições:** O programa irá automaticamente ir para o passo de decisão final

**Regra de Negócio:** RN01

**Calcular decisão do guincho – CSUO5**

**Sumário:** É decidido o tipo de modal que irá fazer o atendimento ao veículo necessitado.

**Ator Primário:** Desenvolvedor Java, Desenvolvedor de outras áreas

**Precondições:** A solicitação do pedido do guincho está no sistema e todas as verificações feitas

**Fluxo Principal:**

1. Pegar todas as outras funções de verificação e colocá-las em parâmetro

2. Efetuar um cálculo para ver quantas delas se adequam no total

3. Comparar com várias situações reais

4. Se cumprir com 2 parâmetros ao menos, devolver como pesado

5. Caso não cumpra, devolver que não precisa de um guincho pesado

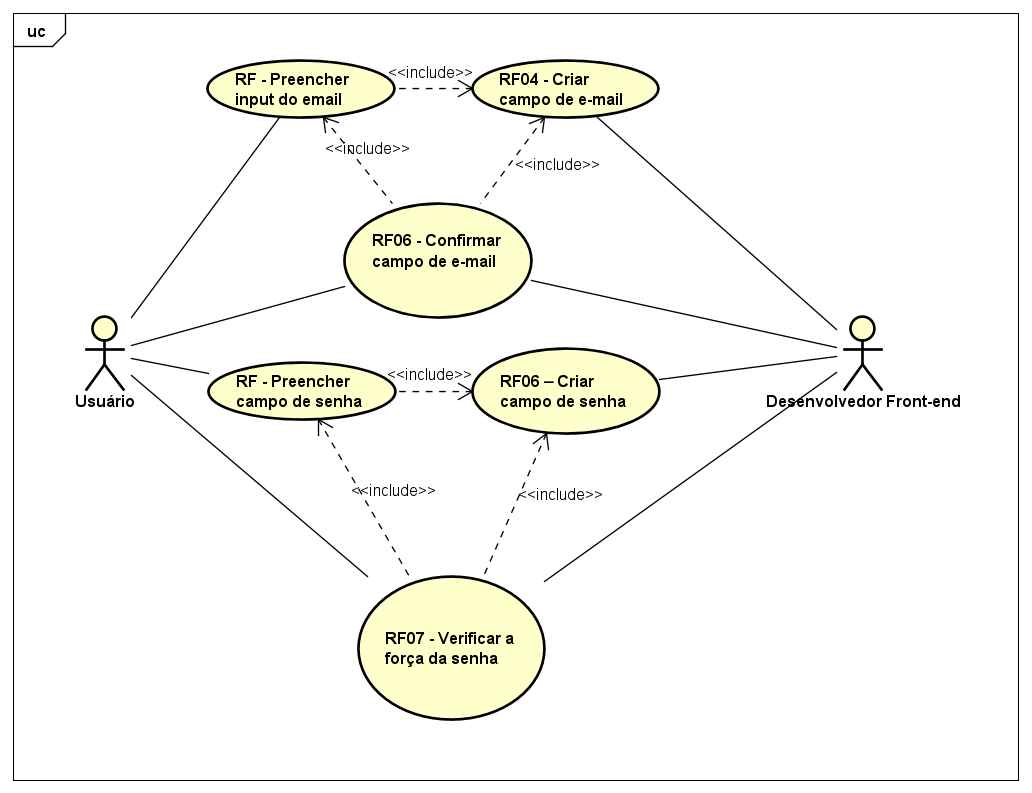
**Fluxo Alternativo:**

**Fluxo de Exceção:**

**Pós-condições:** Após a escolha do modal, será acionado o guincho.

**Regra de Negócio:** RN05, RN06

**Diagrama de Caso de uso 2: Decidir Modal**

****

**Descrição da situação:**

O estudo de caso é sobre um sistema que visa efetuar um método diferente para ser mais assertivo na escolha do guincho baseada na situação do usuário, usando tecnologia. Esse diagrama mostra a parte do cadastro do cliente no nosso sistema que vai direto para o banco de dados.

**Após o levantamento de requisitos iniciais do sistema, os analistas chegaram a seguinte lista de requisitos funcionais:**

**R1:** Criar campo de e-mail

**R2:** Preencher campo do e-mail

**R3:** Criar campo de confirmação de e-mail

**R4:** Preencher campo de confirmação do e-mail

**R5:** Criar campo de senha

**R5:** Preencher campo de senha

**R6:** Criar campo de verificação de senha

**R7:** Preencher campo de verificação de senha

**R8:** Preencher identificação do cliente

**Regras de negócio:**

RN01 – No input do email, o formato deve ser uma combinação de palavras com um @ e o sistema válido, como por exemplo: hotmail.com

RN02 – O campo deve ser obrigatório para o usuário preencher

RN03 – Na confirmação do e-mail, os dois e-mails colocados devem ser coincidentes

RN04 – O campo da senha deve permitir números, letras e caracteres especiais

RN05 – O campo da senha é obrigatório

RN06 – No input da senha, o usuário deve colocar caracteres especiais por quesitos de segurança

RN07 – Se a senha do usuário for fraca (ausência de caracteres especiais), invalidar a senha

**Criar campo do e-mail – CSUO1**

**Sumário:** Um campo interativo para o usuário inserir o e-mail dele

**Ator Primário:** Desenvolvedor Front-end.

**Atores Secundários:** Desenvolvedor das outras áreas (Java).

**Precondições:** O usuário deve ter entrado no sistema e clicado no botão de cadastro

**Fluxo Principal:**

1. O desenvolvedor irá fazer um código para o campo

2. O desenvolvedor irá conectar com o banco de dados

3. O código irá para a fase de teste

4. O campo aparecerá para o cliente

**Fluxo Alternativo:** *O código não está funcionando corretamente*

a) Se o desenvolvedor principal não conseguir, ele deverá entrar em contato com os outros programadores para resolver. Além disso, ele deve colocar um status: “Em construção”

**Fluxo de exceção:** *RN01 – No input do e-mail, o formato deve ser uma combinação de palavras com um @ e o sistema válido, como por exemplo: hotmail.com*

1. Se essa regra for violada, uma mensagem deve aparecer dizendo: “Insira um e-mail válido, seguindo a regra @x.com”

**Pós-condições:** O campo de e-mail estará apto para o cliente preencher corretamente

**Regra de Negócio:** RN01, RN02

**Preencher campo do e-mail – CSUO2**

**Sumário:** Um campo interativo para o usuário inserir o e-mail dele

**Ator Primário:** Usuário.

**Precondições:** O campo de e-mail deve estar criado e o usuário já deve estar na aba de cadastro

**Fluxo Principal:**

1. O usuário vai clicar no campo do e-mail

2. Ele vai inserir o e-mail no formato @servico.com

**Fluxo Alternativo:**

**Fluxo de exceção 1:** *RN01 – No input do e-mail, o formato deve ser uma combinação de palavras com um @ e o sistema válido, como por exemplo: hotmail.com*

1. Se essa regra for violada, uma mensagem deve aparecer dizendo: “Insira um e-mail válido, seguindo a regra @x.com”

**Fluxo de exceção 2:** *Violação da RN02 – O preenchimento é obrigatório*

1. Caso o cliente queira avançar sem inserir um e-mail, uma mensagem deve aparecer na tela dizendo que é obrigatório

**Pós-condições:** O programa irá automaticamente para a próxima verificação

**Regra de Negócio:** RN01, RN02

**Confirmar campo do e-mail – CSUO3**

**Sumário:** Um campo interativo para o usuário confirmar o e-mail inserido anteriormente

**Ator Primário:** Desenvolvedor Front-end.

**Atores Secundários:** Desenvolvedor das outras áreas (Java).

**Precondições:** O usuário deve ter entrado no sistema, clicado no botão de cadastro e inserido o primeiro campo de e-mail

**Fluxo Principal:**

1. O desenvolvedor irá fazer um código para a confirmação do e-mail

2. O desenvolvedor irá colocá-lo abaixo do outro campo

3. O código irá para a fase de teste

4. O campo aparecerá para o cliente

**Fluxo Alternativo:** *O código não está funcionando corretamente*

a) Se o desenvolvedor principal não conseguir, ele deverá entrar em contato com os outros programadores para resolver. Além disso, ele deve colocar um status: “Em construção”

**Fluxo de exceção:** *RN03 – Na confirmação do e-mail, os dois e-mails colocados devem ser coincidentes*

1. Se o usuário colocar dois e-mails diferentes, uma mensagem deve aparecer na tela dizendo: “Os dois e-mails devem ser iguais”

**Pós-condições:** O programa irá automaticamente para os campos de senha

**Regra de Negócio:** RN03

**Criar campo de senha – CSUO4**

**Sumário:** Um campo interativo para o usuário inserir a senha dele

**Ator Primário:** Desenvolvedor Front-end.

**Atores Secundários:** Desenvolvedor das outras áreas (Java).

**Precondições:** O botão do e-mail já deve estar pronto

**Fluxo Principal:**

1. O desenvolvedor irá fazer um código para o campo de senha

2. Ele deverá colocar o campo como obrigatório

3. Tratar as exceções caso o cliente escreva errado

4. O campo aparecerá para o cliente

**Fluxo Alternativo:** *O código não está funcionando corretamente*

a) Se o desenvolvedor principal não conseguir, ele deverá entrar em contato com os outros programadores para resolver. Além disso, ele deve colocar um status: “Em construção”

**Fluxo de exceção:**

**Pós-condições:** O campo de senha estará apto para o cliente preencher corretamente

**Regra de Negócio:** RN04

**Preencher campo de senha – CSUO5**

**Sumário:** Um campo interativo para o usuário inserir a senha dele

**Ator Primário:** Usuário.

**Precondições:** O campo de senha deve estar criado e o usuário já deve ter inserido o campo de e-mail

**Fluxo Principal:**

1. O usuário vai clicar no campo da senha

2. Ele vai colocar a senha dele

**Fluxo Alternativo:**

**Fluxo de exceção 1:** *Violação da RN05 - O campo da senha é obrigatório*

1. Caso o usuário não coloque nada no campo da senha, uma mensagem deve aparecer dizendo que o campo é obrigatório

**Pós-condições:** A verificação da força está apta para funcionar

**Regra de Negócio:** RN05

**Verificar força da senha – CSUO6**

**Sumário:** Um campo interativo para o usuário inserir a senha dele

**Ator Primário:** Desenvolvedor Front-end.

**Precondições:** O campo de senha deve ter sido preenchido

**Fluxo Principal:**

1. Na tela, vai ser mostrado uma barra indicando a força dele

2. Se for forte, a senha pode ser prosseguida

**Fluxo Alternativo:** *Senha fraca*

1. Caso a senha do usuário for fraca, uma mensagem de dica deve aparecer na tela

**Fluxo de exceção 1:** *Violação da RN07 – Ausência de caracteres especiais*

1. Colocar uma mensagem na tela obrigando o usuário a colocar caracteres especiais por motivos de segurança

**Pós-condições:** A seção de cadastrar cliente está pronta

**Regra de Negócio:** RN06, RN07