Regras de Cadastro e Gerenciamento de Usuários

Este documento detalha as regras e procedimentos para o cadastro e gerenciamento de usuários no sistema Bolsa Família, destinado tanto a usuários finais quanto a desenvolvedores front-end.

Para Usuários Finais

1. Cadastro de Novo Usuário

Para se cadastrar no sistema, você precisará fornecer as seguintes informações:

- **Nome Completo:** Seu nome como consta em seus documentos.
- **CPF:** Seu Cadastro de Pessoa Física. O CPF deve ser válido e único no sistema.
- **Email:** Um endereço de email válido e que você acesse regularmente. Este email também deve ser único no sistema.
- **Senha:** Uma senha segura para proteger seu acesso.

Importante:

- Certifique-se de que seu CPF e Email ainda não foram utilizados em outro cadastro neste sistema, nem mesmo como membro familiar (parente) de outro usuário.
- O sistema validará o formato do seu CPF e Email. Insira-os corretamente.
- Ao se cadastrar, você automaticamente será registrado como o "Responsável" pela sua unidade familiar.

2. Atualização de Dados Cadastrais

Você pode atualizar seus dados cadastrais (Nome, CPF, Email, Senha) a qualquer momento após fazer login no sistema.

- **Restrição:** Você só pode alterar os dados do seu próprio cadastro.
- **Validação:** As mesmas validações de formato e unicidade de CPF e Email aplicadas no cadastro inicial também se aplicam na atualização. Se você tentar usar um CPF que já pertence a um membro familiar de outro usuário, a atualização não será permitida.

3. Alteração de Senha

Caso esqueça sua senha, você pode solicitar a alteração fornecendo:

- **CPF:** Seu CPF cadastrado.
- **Email:** O email associado ao seu CPF no cadastro.
- **Nova Senha:** A nova senha que deseja utilizar.

Validação: O sistema verificará se o CPF existe e se o Email fornecido corresponde exatamente ao email cadastrado para aquele CPF antes de permitir a alteração da senha.

Para Desenvolvedores Front-End

Esta seção detalha os endpoints da API, payloads esperados, validações e respostas relacionadas ao gerenciamento de usuários.

1. Endpoints da API

- `POST /api/Usuarios` (Público/Anônimo)
    - **Descrição:** Cria um novo usuário.
    - **Request Body:** `UsuarioInputDto` `json { "nome": "string", "cpf": "string", "email": "string", "senha": "string" }`
    - **Validações Back-end:**
        - Formato de CPF (`ValidadorUtils.CpfValido`).
        - Formato de Email (`ValidadorUtils.EmailValido`).
        - Unicidade de CPF (verifica se já existe na tabela `Usuario` OU `Parente`).
        - Unicidade de Email (verifica se já existe na tabela `Usuario`).
    - **Processamento:**
        - Senha é hasheada com BCrypt.
        - Cria um registro na tabela `Usuario` .
        - Cria um registro na tabela `Parente` associado ao novo usuário, com `GrauParentesco = "Responsável"` .
    - **Respostas:**
        - **Sucesso (200 OK):** `Response<bool>` `json { "success": true, "message": "Usuário cadastrado com sucesso.", "data": true, "errors": null }`
        - **Falha (400 Bad Request):** `Response<bool>` com mensagens de erro específicas. `json // Exemplo CPF inválido { "success": false, "message": "CPF informado é inválido.", "data": false, "errors": ["CPF informado é inválido."] } // Exemplo CPF já existente { "success": false, "message": "Já`

existe um usuário cadastrado com este CPF.", "data": false, "errors": ["Já existe um usuário cadastrado com este CPF."] } // Exemplo Email já existente { "success": false, "message": "Já existe um usuário cadastrado com este e-mail.", "data": false, "errors": ["Já existe um usuário cadastrado com este e-mail."] } // Exemplo CPF pertence a Parente de outro usuário { "success": false, "message": "Já existe um usuário cadastrado com este CPF como membro familiar de outro usuário.", "data": false, "errors": ["Já existe um usuário cadastrado com este CPF como membro familiar de outro usuário."] }

- **PUT /api/Usuarios/{id}** (Requer Autenticação)
  - **Descrição:** Atualiza os dados de um usuário existente.
  - **Parâmetro de Rota:** `id` (ID do usuário a ser atualizado).
  - **Request Body:** `UsuarioInputDto` (mesmo formato do cadastro).
  - **Validações Back-end:**
    - Verifica se o ID do usuário autenticado (`ClaimTypes.NameIdentifier`) é igual ao `id` da rota.
    - Formato de CPF.
    - Formato de Email.
    - Verifica se o CPF informado já existe como `Parente` de outro usuário (permite manter o próprio CPF ou alterá-lo para um CPF não vinculado a outros).
    - Verifica se o usuário com o `id` fornecido existe.
  - **Processamento:** Atualiza `Nome` , `Cpf` , `Email` e `SenhaHash` (senha é re-hasheada) do usuário.
  - **Respostas:**
    - **Sucesso (200 OK):** `Response<bool>` json { "success": true, "message": "Usuário atualizado com sucesso.", "data": true, "errors": null }
    - **Falha (400 Bad Request):** `Response<bool>` com mensagens de erro. json // Exemplo: Tentando alterar outro usuário { "success": false, "message": "Só é possível alterar o próprio cadastro.", "data": false, "errors": ["Só é possível alterar o próprio cadastro."] } // Exemplo: CPF pertence a Parente de outro usuário { "success": false, "message": "Já existe um usuário cadastrado com este CPF como membro familiar de outro usuário.", "data": false, "errors": ["Já existe um usuário cadastrado com este CPF como membro familiar de outro usuário."] } // Exemplo: Usuário não encontrado { "success": false, "message": "Usuário não encontrado.", "data": false, "errors": ["Usuário não encontrado."] }

- **PUT /api/Usuarios/AlterarSenha** (Público/Anônimo)
    - **Descrição:** Altera a senha de um usuário validando por CPF e Email.
    - **Request Body:** PasswordInputDto `json { "cpf": "string", "email": "string", "novaSenha": "string" }`
    - **Validações Back-end:**
        - Verifica se existe um usuário com o `cpf` fornecido.
        - Verifica se o `email` fornecido corresponde (ignorando case) ao email cadastrado para o usuário encontrado.
    - **Processamento:** Atualiza o `SenhaHash` do usuário com a `novaSenha` hasheada.
    - **Respostas:**
        - **Sucesso (200 OK):** `Response<bool> json { "success": true, "message": "Senha atualizada com sucesso.", "data": true, "errors": null }`
        - **Falha (400 Bad Request):** `Response<bool>` com mensagens de erro. `json // Exemplo: CPF não encontrado { "success": false, "message": "CPF informado não foi encontrado.", "data": false, "errors": ["CPF informado não foi encontrado."] } // Exemplo: Email não confere { "success": false, "message": "CPF e e-mail não conferem.", "data": false, "errors": ["CPF e e-mail não conferem."] }`
- **Endpoints Restritos (Admin):**
    - `GET /api/Usuarios`
    - `GET /api/Usuarios/{id}`
    - `GET /api/Usuarios/cpf/{cpf}`
    - `DELETE /api/Usuarios/{id}`
    - **Observação:** Estes endpoints requerem autenticação e a role "Admin".

2. Estrutura de Resposta Padrão

Todas as respostas da API seguem o padrão `Response<T>`:

```
 public class Response<T>
{
    public bool Success { get; set; }
    public string Message { get; set; }
    public T Data { get; set; }
    public List<string> Errors { get; set; }
}
```

- `Success`: Indica se a operação foi bem-sucedida.

- `Message`: Mensagem descritiva sobre o resultado da operação.
- `Data`: Os dados retornados pela operação (pode ser `bool`, um objeto DTO, ou uma lista de DTOs).
- `Errors`: Uma lista de mensagens de erro detalhadas em caso de falha (`Success = false`). O front-end deve priorizar exibir a `Message` principal, mas pode usar `Errors` para logs ou detalhes adicionais se necessário.

3. Validações Importantes

- **CPF/Email:** Utilizar máscaras e validações no front-end para melhorar a experiência do usuário, mas a validação final de formato e unicidade é feita no back-end.
- **Unicidade:** O back-end garante que um CPF não pode ser usado para cadastrar um novo usuário se já existir na tabela `Usuario` ou `Parente`. O mesmo CPF também não pode ser usado na atualização se pertencer a um `Parente` de outro usuário. Emails devem ser únicos na tabela `Usuario`.
- **Autorização:** Garantir que os tokens JWT sejam enviados corretamente nos headers das requisições para endpoints protegidos.

Para Usuários Finais (Continuação)

4. Cadastro e Gerenciamento de Parentes

Após o cadastro inicial, você pode gerenciar os membros da sua unidade familiar (parentes) no sistema.

- **Adicionar Parente:** Você pode cadastrar novos parentes associados ao seu usuário, fornecendo:
  - Nome Completo do Parente
  - CPF do Parente (deve ser válido e único entre todos os parentes de todos os usuários, e também não pode ser um CPF já cadastrado como usuário principal)
  - Grau de Parentesco (ex: Filho, Cônjuge - as opções válidas são definidas pelo administrador)
  - Sexo
  - Estado Civil
  - Ocupação
  - Telefone
  - Renda Mensal do Parente
- **Atualizar Parente:** Você pode alterar os dados de um parente que você cadastrou.
  - **Restrição:** Só é possível alterar parentes vinculados ao seu próprio cadastro.

    - **Validação:** O CPF informado na atualização não pode pertencer a um parente de outro usuário.
- **Listar Parentes:** Você pode visualizar a lista de todos os parentes cadastrados em sua unidade familiar.
- **Remover Parente:** Você pode remover um parente do seu cadastro.
    - **Restrição:** Só é possível remover parentes vinculados ao seu próprio cadastro.
- **Calcular Renda Familiar:** O sistema permite calcular a renda per capita da sua unidade familiar com base nos parentes cadastrados e verificar a elegibilidade para o Bolsa Família, comparando com um valor base definido pelo administrador.

5. Informações Gerais (Administração)

Certas configurações do sistema, como o valor base da renda per capita para elegibilidade e os tipos de parentesco permitidos, são gerenciadas por administradores e não podem ser alteradas por usuários comuns.

Para Desenvolvedores Front-End (Continuação)

4. Endpoints da API - Parentes

Estes endpoints requerem autenticação do usuário.

- **`POST /api/Parentes`**
    - **Descrição:** Cadastra um novo parente vinculado ao usuário logado.
    - **Request Body:** `ParenteInputDto` `json { "nome": "string", "cpf": "string", "grauParentesco": "string", // Deve ser um dos tipos permitidos "sexo": 0, // Enum: 0=NaoInformado, 1=Masculino, 2=Feminino "estadoCivil": 0, // Enum: 0=NaoInformado, 1=Solteiro, 2=Casado, ... "ocupacao": "string", "telefone": "string", "renda": 0 }`
    - **Validações Back-end:**
        - Formato de CPF.
        - Verifica se o usuário está logado.
        - Unicidade de CPF do parente (não pode existir na tabela `Parente` para o mesmo usuário logado).
        - Unicidade de CPF do parente (não pode existir na tabela `Parente` para nenhum usuário).
    - **Processamento:** Cria um registro na tabela `Parente` associado ao `UsuarioId` do usuário logado.

    - **Respostas:**
        - **Sucesso (200 OK):** `Response<bool>` { "success": true, "message": "Parente cadastrado com sucesso!", ... }
        - **Falha (400 Bad Request):** `Response<bool>` com mensagens de erro (CPF inválido, Usuário não logado, CPF já cadastrado para este usuário, CPF já cadastrado para outro usuário).

- **`PUT /api/Parentes/{id}`**
    - **Descrição:** Atualiza um parente existente vinculado ao usuário logado.
    - **Parâmetro de Rota:** `id` (ID do parente a ser atualizado).
    - **Request Body:** `ParenteInputDto` (mesmo formato do cadastro).
    - **Validações Back-end:**
        - Formato de CPF.
        - Verifica se o usuário está logado.
        - Verifica se o parente com o `id` fornecido pertence ao usuário logado.
        - Verifica se o CPF informado já existe como `Parente` de outro usuário.
    - **Processamento:** Atualiza os dados do parente.
    - **Respostas:**
        - **Sucesso (200 OK):** `Response<bool>` { "success": true, "message": "Parente atualizado com sucesso!", ... }
        - **Falha (400 Bad Request):** `Response<bool>` com mensagens de erro (CPF inválido, Usuário não logado, Parente não encontrado, CPF pertence a outro usuário).

- **`GET /api/Parentes`**
    - **Descrição:** Lista todos os parentes cadastrados pelo usuário logado.
    - **Validações Back-end:** Verifica se o usuário está logado.
    - **Respostas:**
        - **Sucesso (200 OK):** `Response<IEnumerable<ParenteDto>>`
        - **Falha (400 Bad Request):** `Response<IEnumerable<ParenteDto>>` { "success": false, "message": "Usuário não logado ou ID de usuário não encontrado.", ... }

- **`GET /api/Parentes/cpf/{cpf}`**
    - **Descrição:** Busca um parente específico do usuário logado pelo CPF.
    - **Validações Back-end:** Formato de CPF, Verifica se o usuário está logado.
    - **Respostas:**
        - **Sucesso (200 OK):** `Response<ParenteDto>`

        - **Falha (400 Bad Request):** `Response<ParenteDto>` com mensagens de erro (CPF inválido, Usuário não logado, Parente não encontrado).

- **`DELETE /api/Parentes/{id}`**
    - **Descrição:** Remove um parente vinculado ao usuário logado.
    - **Parâmetro de Rota:** `id` (ID do parente a ser removido).
    - **Validações Back-end:** Verifica se o usuário está logado, Verifica se o parente pertence ao usuário logado.
    - **Respostas:**
        - **Sucesso (200 OK):** `Response<bool>` { "success": true, "message": "Parente removido com sucesso!", ... }
        - **Falha (400 Bad Request):** `Response<bool>` com mensagens de erro (Usuário não logado, Parente não encontrado).

- **`GET /api/Parentes/renda`**
    - **Descrição:** Calcula a renda familiar per capita e verifica a elegibilidade para o Bolsa Família com base nos parentes do usuário logado.
    - **Validações Back-end:** Verifica se o usuário está logado.
    - **Respostas:**
        - **Sucesso (200 OK):** `Response<string>` com a mensagem indicando se é elegível ou não.
        - **Falha (400 Bad Request):** `Response<string>` { "success": false, "message": "Usuário não encontrado ou não está logado.", ... }

5. Endpoints da API - Informações Gerais (Admin)

Estes endpoints requerem autenticação e a role "Admin".

- **`GET /api/Admin`**
    - **Descrição:** Lista as informações gerais (configurações do sistema).
    - **Respostas:**
        - **Sucesso (200 OK):** `Response<InfoGeraisDto>`
        - **Falha (400 Bad Request):** `Response<InfoGeraisDto>` { "success": false, "message": "Configurações gerais não encontradas...", ... }

- **`PUT /api/Admin/{id}`**
    - **Descrição:** Atualiza as informações gerais.
    - **Parâmetro de Rota:** `id` (ID do registro de InfoGerais, geralmente será 1).

- **Request Body:** `InfoGeraisInputDto` `json { "valorBaseRendaPerCapita": 0, "tiposParentescoPermitidos": "string" // Ex: "Filho, Cônjuge, Pai, Mãe" }`
- **Validações Back-end:** Verifica se o registro de InfoGerais existe.
- **Respostas:**
  - **Sucesso (200 OK):** `Response<bool>` { "success": true, "message": "Informações gerais atualizadas com sucesso!", ... }
  - **Falha (400 Bad Request):** `Response<bool>` { "success": false, "message": "Configurações gerais não encontradas para atualização.", ... }

6. Endpoints da API - DropDowns (Público/Anônimo)

Estes endpoints fornecem listas de opções para preenchimento de formulários.

- **`GET /api/DropDowns/estados-civis`**
  - **Descrição:** Retorna a lista de estados civis disponíveis (Enum `EstadoCivil`).
  - **Resposta (Sucesso 200 OK):** `Response<List<object>>` onde cada objeto tem `value` (int) e `name` (string). `json { "success": true, "message": "Lista de estados civis obtida com sucesso.", "data": [ { "value": 0, "name": "NaoInformado" }, { "value": 1, "name": "Solteiro" }, // ... outros estados civis ], "errors": null }`
- **`GET /api/DropDowns/generos`**
  - **Descrição:** Retorna a lista de gêneros disponíveis (Enum `Sexo`).
  - **Resposta (Sucesso 200 OK):** `Response<List<object>>` onde cada objeto tem `value` (int) e `name` (string). `json { "success": true, "message": "Lista de gêneros obtida com sucesso.", "data": [ { "value": 0, "name": "NaoInformado" }, { "value": 1, "name": "Masculino" }, { "value": 2, "name": "Feminino" } ], "errors": null }`
- **`GET /api/DropDowns/tipos-parentesco`**
  - **Descrição:** Retorna a lista de tipos de parentesco permitidos, conforme configurado nas Informações Gerais.
  - **Resposta (Sucesso 200 OK):** `Response<List<string>>`. `json { "success": true, "message": "Lista de tipos de parentesco obtida com sucesso.", "data": ["Filho", "Cônjuge", "Pai", "Mãe"], // Exemplo "errors": null }`