**EveryFin - Cadastro e Gestão de Usuários e Clientes (Documento Técnico e Negocial) Versão Detalhada e Aprimorada**

---

# Sumário



---

# 1. Visão Geral do Cadastro de Usuários e Clientes

Esta seção detalha a funcionalidade crucial de gerenciamento de usuários e clientes dentro do EveryFin, visando otimizar os fluxos de trabalho e garantir a integridade dos dados.

## 1.1. Objetivo

O objetivo principal é permitir o cadastro e a gestão de usuários e clientes de forma estruturada e segura. Além disso, busca facilitar a vinculação de clientes nos lançamentos financeiros (entradas e saídas) através de uma funcionalidade de autocomplete, agilizando o processo e reduzindo erros de digitação.

## 1.2. Escopo e Valor de Negócio

- **Cadastro Inteligente**: A implementação de um sistema de cadastro inteligente para clientes e usuários melhora significativamente o fluxo de trabalho geral do sistema.

- **Eficiência e Precisão**: Reduz o tempo de digitação manual e aumenta a precisão dos lançamentos financeiros, um aspecto crítico para a confiabilidade dos dados financeiros.
- **Controle e Permissões**: Permite um controle refinado por empresa, usuário e permissões, preparando o sistema para cenários de uso corporativo e multiempresa.
- **Preparação para SaaS**: Esta funcionalidade é fundamental para preparar o sistema para um modelo SaaS (Software as a Service) multiempresa, permitindo que times distintos operem de forma isolada e segura dentro da mesma plataforma.

# 2. Cadastro de Usuários

O módulo de cadastro de usuários é a base para o controle de acesso e segurança do sistema.

## 2.1. Campos e Requisitos

Os campos essenciais para o cadastro de um novo usuário incluem:

- `nome`: Nome completo do usuário.
- `CPF`: Cadastro de Pessoa Física, utilizado como identificador único e para futuras integrações ou verificações de identidade.
- `e-mail`: Endereço de e-mail, que servirá como nome de usuário para o login e para comunicações.
- `senha`: Senha segura, que será hashada e armazenada de forma criptografada.
- `empresa` (para perfil `admin`): Identificador da empresa à qual o usuário administrador está vinculado, essencial para o isolamento de dados em um modelo multiempresa.

## 2.2. Endpoint da API

- **`POST /usuarios`**: Será a rota responsável por receber os dados do novo usuário e processar seu registro no backend.

## 2.3. Validação e Segurança

- **Validação com Zod**: Todos os campos de entrada serão rigorosamente validados no frontend e no backend usando `Zod`. As validações incluirão:
  - Formato correto de `CPF` (ex: 11 dígitos numéricos, com ou sem máscara).
  - Formato válido de `e-mail`.
  - Força da `senha` (requisitos mínimos de comprimento, caracteres especiais, números, letras maiúsculas/minúsculas).
- **Hash de Senha**: No backend, as senhas serão hashadas com `bcrypt` (com 12 salt rounds) antes de serem armazenadas no banco de dados, garantindo que as senhas originais nunca sejam expostas.

## 2.4. Separação de Perfil (Admin vs. Comum)

- O sistema diferenciará usuários com perfil `admin` de usuários `comuns`. O perfil `admin` terá acesso a funcionalidades de gerenciamento de usuários e clientes em nível de empresa, enquanto usuários `comuns` terão acesso restrito às suas próprias transações financeiras e aos clientes que eles cadastraram ou que são visíveis para sua empresa.
- Essa separação será gerenciada por um campo `role` ou `isAdmin` no modelo `User` do banco de dados.

# 3. Cadastro e Gestão de Clientes

O módulo de clientes é essencial para categorizar e rastrear as origens de receitas e destinos de despesas.

## 3.1. Campos e Atributos

Os campos para o cadastro de clientes incluirão:

- `nome`: Nome completo do cliente ou nome fantasia da empresa.
- `CPF/CNPJ`: Documento de identificação (Cadastro de Pessoa Física ou Cadastro Nacional de Pessoa Jurídica). Este campo pode ser opcional ou condicional, dependendo do tipo de cliente.
- `razão social` (opcional): Para clientes do tipo pessoa jurídica.
- Outros campos opcionais: telefone, email, endereço.

## 3.2. Relacionamento com Usuário e Empresa

- Cada cliente será relacionado a um `User` (o usuário que o cadastrou) e/ou a uma `Company` (a empresa que o usuário pertence), garantindo o isolamento de dados e a visibilidade correta em cenários multiempresa.
- Os modelos do Prisma (`Cliente`, `User`, `Company`) refletirão essas relações através de chaves estrangeiras (`userId`, `companyId`).

## 3.3. Funcionalidades CRUD e Listagem

- Serão implementadas todas as operações CRUD (Create, Read, Update, Delete) para os registros de clientes, permitindo aos usuários:
  - Cadastrar novos clientes.
  - Visualizar uma lista paginada e filtrável de clientes.
  - Editar informações de clientes existentes.
  - Excluir registros de clientes (com confirmação para evitar exclusões acidentais).
- A tela de listagem de clientes (`ListarClientes.tsx`) incluirá busca, paginação e opções de exclusão.

# 4. Funcionalidade de Autocomplete para Clientes

A funcionalidade de autocomplete é um aprimoramento de UX crucial para agilizar o registro de transações.

### 4.1. Uso em Lançamentos Financeiros

- Os formulários de entrada e saída financeira usarão um campo inteligente de autocomplete para a seleção do cliente/fornecedor, em vez de um campo de texto simples. Isso reduz a digitação manual e garante que os lançamentos sejam vinculados a clientes existentes, aumentando a precisão dos dados.

### 4.2. Busca em Tempo Real e `Debounce`

- O campo de autocomplete fará buscas em tempo real na base de clientes associada ao usuário ou empresa logada.
- A funcionalidade de `debounce` será implementada para otimizar as requisições ao backend. Isso significa que a busca só será disparada após um pequeno atraso (ex: 300ms) desde a última digitação do usuário, evitando requisições excessivas enquanto o usuário ainda está digitando.

### 4.3. Experiência de Usuário do Autocomplete

- À medida que o usuário digita, uma lista suspensa de sugestões de clientes será exibida.
- Ao selecionar uma sugestão, o campo será automaticamente preenchido e o ID do cliente correspondente será vinculado ao lançamento financeiro (`Entry` ou `Exit`).
- Opção para criar um novo cliente diretamente do autocomplete, caso a busca não retorne resultados.

# 5. Integração Backend (Modelos e Rotas)

O backend será a espinha dorsal para o gerenciamento de usuários e clientes.

### 5.1. Modelos Prisma Detalhados

O esquema do Prisma incluirá os seguintes modelos:

- **`User`**: Conforme definido anteriormente (id, name, email, cpf, password, isAdmin, createdAt, updatedAt). Relacionamentos com `Client` e `Company`.
- **`Client`**:
    - `id` (Int, @id, @default(autoincrement()))
    - `name` (String)
    - `cpfCnpj` (String?, @unique)
    - `razaoSocial` (String?, opcional para PJ)
    - `userId` (Int?, chave estrangeira para o User que o cadastrou, opcional se vinculado apenas à empresa)
    - `companyId` (Int?, chave estrangeira para a Company, essencial para multiempresa)
    - `createdAt` (DateTime)
    - `updatedAt` (DateTime)
    - **Relacionamentos**: `user` (User, @relation), `company` (Company, @relation), `entries` (Entry[]), `exits` (Exit[]).

- **`Company` (Opcional, para multiempresa)**: Um modelo `Company` pode ser introduzido para representar as empresas no cenário multiempresa, com um relacionamento para `User` e `Client`.

### 5.2. Rotas Seguras e Permissões

- **Rotas Protegidas com JWT**: Todas as rotas de cadastro e gestão de usuários e clientes serão protegidas por autenticação JWT, garantindo que apenas usuários autenticados possam acessá-las.
- **Permissões Diferenciadas por Role**: A autorização será implementada para aplicar permissões diferenciadas baseadas no `role` (papel) do usuário (ex: `admin` pode gerenciar todos os usuários/clientes da empresa, `comum` apenas os seus próprios ou os visíveis para ele).
- **Endpoint de Autocomplete**: **`GET /clientes?search=`** será a rota do backend para a funcionalidade de autocomplete, retornando clientes filtrados por nome, CPF/CNPJ, etc.

# 6. Componentes Frontend Dedicados

O frontend contará com componentes React específicos para cada funcionalidade.

### 6.1. `FormUsuario.tsx` (Formulário de Cadastro/Edição de Usuário)

- Componente responsável pela UI e lógica dos formulários de cadastro e edição de usuários.
- Utilizará `React Hook Form` para gerenciamento de estado do formulário e `Zod` para validação em tempo real.

### 6.2. `FormCliente.tsx` (Formulário de Cadastro/Edição de Cliente)

- Componente similar ao `FormUsuario.tsx`, mas focado nos campos e lógica para clientes.
- Incluirá validações para `CPF/CNPJ` e outros campos específicos de clientes.

### 6.3. `ListarClientes.tsx` (Listagem de Clientes)

- Componente que exibirá a tabela de clientes com funcionalidades de busca, paginação e opções de exclusão.
- Permitirá a navegação para os formulários de edição de clientes.

### 6.4. `AutocompleteInput.tsx` (Componente de Autocomplete)

- Componente reutilizável que encapsula a lógica do campo de autocomplete, incluindo o `debounce` na busca e o tratamento do evento `onSelect`.
- Será integrado aos formulários de entradas e saídas financeiras.

# 7. Testes Automatizados

Uma estratégia de testes robusta garantirá a qualidade e a confiabilidade das funcionalidades de cadastro e autocomplete.

### 7.1. Estratégia de Testes

- **Unitários**: Testes para as lógicas de validação (Zod), serviços do backend (ex: criação de usuário/cliente), e componentes de formulário isolados.
- **De Integração**: Testes que simulam o fluxo completo de cadastro (frontend enviando para backend, backend salvando no DB) e a resposta da API de autocomplete.
- **End-to-End (E2E) com Cypress**:
    - **Fluxo de Cadastro**: Teste o processo completo de cadastro de um novo usuário e de um novo cliente, verificando se os dados são persistidos corretamente e se o usuário é redirecionado para a tela esperada.
    - **Autocomplete em Lançamento**: Simule a digitação em um campo de lançamento de transação, verifique se as sugestões de autocomplete aparecem e se a seleção de um cliente funciona corretamente.

### 7.2. Ferramentas de Teste

- **Jest + React Testing Library**: Para testes unitários e de integração no frontend.
- **Supertest**: Para testes de integração da API do backend.
- **Cypress**: Para testes End-to-End.

### 7.3. Cobertura de Testes

- Será estabelecida uma meta de alta cobertura de testes (ex: 90%) para as lógicas de cadastro, autenticação e a funcionalidade de autocomplete, dado sua criticidade para a integridade dos dados e a experiência do usuário.

## 8. CI/CD

As funcionalidades de cadastro e autocomplete serão integradas ao pipeline de CI/CD para garantir entregas rápidas e de alta qualidade.

### 8.1. Deploy Automático e Testes Contínuos

- O pipeline de CI/CD (GitHub Actions) será configurado para rodar automaticamente os testes de lint, unidade, integração e E2E a cada push ou Pull Request.
- Somente builds que passarem em todos os testes serão elegíveis para deploy automático nos ambientes de staging e produção (via Vercel para frontend e Render.com para backend).

## 9. Proposta Negocial Aprimorada

A implementação do cadastro de usuários e clientes com autocomplete adiciona valor estratégico e operacional significativo ao EveryFin.

### 9.1. Valor Agregado

- **Eficiência Operacional**: A funcionalidade de cadastro inteligente de clientes e usuários melhora substancialmente o fluxo de trabalho dos usuários, tornando as operações mais rápidas e menos propensas a erros.
- **Precisão dos Dados**: A redução do tempo de digitação e o aumento da precisão dos lançamentos financeiros resultam em dados mais confiáveis para relatórios e análises.
- **Controle Aprimorado**: A capacidade de controlar usuários por empresa e gerenciar permissões prepara o sistema para um crescimento futuro e para atender a múltiplos clientes corporativos.

### 9.2. Benefícios Chave

- **Experiência do Usuário (UX) Superior**: O autocomplete é um recurso de UX que melhora a usabilidade e a satisfação do usuário ao interagir com o sistema.
- **Preparação para SaaS Multiempresa**: A arquitetura de dados e de permissões estabelece uma base sólida para a evolução do EveryFin para um modelo de Software as a Service (SaaS) que pode atender a várias empresas e seus times de forma isolada e segura.
- **Integridade dos Dados**: A validação robusta e a seleção de clientes a partir de uma base de dados centralizada minimizam a criação de dados duplicados ou inconsistentes.

# 10. Prompt Reutilizável

Este prompt pode ser utilizado para solicitar a implementação de funcionalidades semelhantes em outros sistemas, seguindo o mesmo padrão de excelência e detalhamento.

"Olá, [Nome do Modelo]!

Implemente o cadastro e a gestão de usuários e clientes em um sistema financeiro, com os seguintes requisitos:

1. **Cadastro de Usuários**: Inclua campos essenciais (nome, CPF, e-mail, senha) com validações rigorosas (Zod) e separação de perfil (admin vs. comum).
2. **Cadastro e Gestão de Clientes**: Permita cadastro de clientes (nome, CPF/CNPJ, razão social) com relacionamento ao usuário e/ou empresa.
3. **Autocomplete em Lançamentos**: Implemente um campo de autocomplete para clientes/fornecedores em formulários de lançamento de entradas e saídas, com busca em tempo real (`GET /clientes?search=`) e `debounce`.
4. **Integração Completa**: Garanta a integração frontend-backend com modelos Prisma detalhados para `Cliente` e `User`, e rotas seguras com JWT e permissões diferenciadas.
5. **Componentes Dedicados**: Desenvolva componentes React específicos como `FormCliente.tsx`, `ListarClientes.tsx` e `AutocompleteInput.tsx`.

6. **Testes Automatizados**: Inclusão de testes unitários (validação, resposta da API), e testes E2E (fluxo de cadastro completo, uso do autocomplete em lançamento).
7. **CI/CD Habilitado**: Garanta que o deploy automático e os testes contínuos sejam habilitados no pipeline de CI/CD.

Finalize com um documento técnico-negocial detalhado em PDF, incluindo visão geral, funcionalidades, tecnologias, componentes, testes, CI/CD e a proposta de valor.

*Documento expandido para excelência máxima e alinhamento com padrões UX e de negócios.*"

Fontes

Criar Resumo em Áudio

Deep Research
Canvas
Vídeo

O Gemini pode cometer erros. Por isso