📘 Documentação Técnica — WebApp Livro

Introdução

Objetivo do Sistema

O **WebApp Livro** é uma aplicação web desenvolvida para modernizar e informatizar o fluxo de registro, acompanhamento e gestão de denúncias e fiscalizações realizadas pela Vigilância Sanitária. Antes, esse processo era feito manualmente em livros físicos, o que tornava o controle difícil e pouco eficiente.

O sistema permite o cadastro e a consulta de denúncias, registros de relatórios técnicos, e a gestão de usuários fiscais responsáveis pela fiscalização. Com ele, toda a tramitação e histórico das denúncias ficam registrados digitalmente, facilitando a consulta, análise e tomada de decisão.

Tecnologias Utilizadas

O projeto foi desenvolvido utilizando as seguintes tecnologias principais:

- **Node.js** — ambiente de execução para JavaScript no backend.
- **Express** — framework para criação das rotas e API REST.
- **Sequelize** — ORM para interação com o banco de dados MySQL.
- **MySQL** — banco de dados relacional utilizado para armazenamento das informações.
- **EJS (Embedded JavaScript Templates)** — engine de template para renderização das views.
- **Body-Parser** — middleware para tratar requisições HTTP com payload JSON e urlencoded.
- **Middleware personalizado de Paginação** — para facilitar o controle de limite e offset nas consultas.

- **Outras ferramentas** — uso de módulos separados por responsabilidade (models, controllers, middlewares), organização modular.

Visão Geral da Arquitetura

O sistema segue o padrão **MVC (Model-View-Controller)**, que promove a separação de responsabilidades e facilita a manutenção:

- **Models:** definem a estrutura das tabelas no banco de dados e os relacionamentos entre elas. Exemplos: usuários, denúncias, relatórios.

- **Controllers:** implementam as regras de negócio e manipulação dos dados, respondendo às requisições HTTP.

- **Views:** templates em EJS que geram o HTML para visualização das informações pelo usuário final.

- **Middleware:** funções intermediárias que tratam requisições, como a paginação, validação, autenticação, etc.

A comunicação entre o frontend e backend ocorre principalmente via rotas HTTP, que manipulam dados no banco de dados e apresentam informações dinâmicas ao usuário.

Documentação das Constantes

`/constants/denunciationSender.js`

Define os **tipos de origem** de uma denúncia recebida. Estas opções são utilizadas no momento do registro de uma nova denúncia e ajudam a classificar a forma como ela foi encaminhada à Vigilância Sanitária.

Constante	Descrição
`PESSOA_PRESENCIAL`	Denúncia feita de forma presencial, anonimamente
`PESSOA_TELEFONE`	Denúncia feita por telefone, também anônima
`OFICIO`	Encaminhamento por documento oficial (ofício)

`/constants/denunciationStatus.js`

Define os **estágios de andamento** de uma denúncia no sistema. Cada status possui uma cor associada (para visualização na interface) e um identificador (`slug`) usado no banco e na lógica interna.

Slug	Rótulo na interface	Cor associada	Situação
`REGISTRADA`	Registrada	Azul (#3498db)	Denúncia foi registrada, mas ainda não atribuída
`EM_ANDAMENTO`	Em Andamento	Laranja (#f39c12)	Fiscal já está atuando no caso
`NOTIFICADA`	Notificada	Verde (#2ecc71)	Denunciado foi notificado formalmente
`EM_PUBLICACAO`	Em Publicação	Roxo (#8e44ad)	Ato ou notificação em trâmite de publicação oficial
`FEITO_AIF`	Em Processo Administrativo	Verde escuro (#16a085)	Ação interna formal instaurada (ex: AIF)
`FINALIZADA`	Finalizada	Cinza (#95a5a6)	Caso encerrado, sem pendências

---

/constants/reportType.js

Define os **tipos de relatórios** que podem ser adicionados a uma denúncia. Usado internamente para distinguir alterações de status de observações descritivas.

```
const REPORT_TYPE = {
   STATUS_CHANGE: 0,
   TEXT: 1
};
```

Constante	Valor	Descrição
`STATUS_CHANGE`	`0`	Relatório criado automaticamente ao mudar o status da denúncia
`TEXT`	`1`	Relatório textual adicionado manualmente pelo fiscal

Banco de Dados — Estrutura e Relacionamentos

A aplicação utiliza **MySQL** como SGBD e o ORM **Sequelize** para definição e manipulação dos modelos. A conexão está definida no arquivo `/database/database.js`.

Conexão com o Banco (`/database/database.js`)

```
const Sequelize = require('sequelize');
const connection = new Sequelize('livro_visa', 'root', 'masterkey', {
    host: '127.0.0.1',
    dialect: 'mysql',
    dialectOptions: { connectTimeout: 60000 },
    pool: {
        max: 5,
        min: 0,
        acquire: 30000,
        idle: 10000
    },
});
```

- Banco: **livro_visa**
- Usuário: **root**
- Senha: **masterkey**
- Host: `127.0.0.1`
- Dialeto: **MySQL**
- Pool de conexões configurado

Tabelas

`denunciations` — Denúncias

Modelo: `/denunciations/denunciationsModel.js`

Armazena os registros de denúncias sanitárias.

Campo	Tipo	Obrigatório	Descrição
`id`	INTEGER (PK)	Sim	Chave primária
`year`	INTEGER	Sim	Ano da denúncia
`number`	INTEGER	Sim	Número da denúncia (por ano)
`registration_type`	STRING	Não	Tipo de registro (ex: presencial, telefone, ofício)
`status`	STRING	Não	Status atual da denúncia
`title`	STRING	Sim	Título/resumo da denúncia
`description`	TEXT	Sim	Descrição detalhada
`endereco`	STRING	Sim	Logradouro do local
`numero`	STRING	Sim	Número do local
`bairro`	STRING	Sim	Bairro do local
`complemento`	STRING	Não	Complemento do endereço
`created_at`	DATE	Não	Data de criação da denúncia
`user_id`	INTEGER (FK)	Não	Usuário responsável pelo registro (chave estrangeira para `users`)

Índice Único: `year + number` para evitar duplicações no mesmo ano.

`reports` — Relatórios Técnicos

Modelo: `/reports/reportsModel.js`

Cada denúncia pode possuir vários relatórios relacionados.

Campo	Tipo	Obrigatório	Descrição
`id`	INTEGER (PK)	Sim	Chave primária
`description`	TEXT	Não	Descrição do relatório técnico
`created_at`	DATE	Não	Data de criação do relatório
`denunciation_id`	INTEGER (FK)	Sim	Referência à denúncia (chave estrangeira para `denunciations`)
`user_id`	INTEGER (FK)	Sim	Usuário responsável pelo relatório (chave estrangeira para `users`)
`type`	INTEGER	Sim	Tipo de relatório (0 = mudança de status, 1 = texto livre)

`users` — Usuários

Modelo: `/users/usersModel.js`

Representa os fiscais que registram ou atualizam denúncias e relatórios.

Campo	Tipo	Obrigatório	Descrição
`id`	INTEGER	Sim	Chave primária
`name`	STRING	Sim	Nome do usuário
`ativo`	BOOLEAN	Sim	Define se o usuário está ativo no sistema (padrão: true)

Relacionamentos

Configurados no `index.js` principal da aplicação:

```
user.hasMany(denunciation, { foreignKey: 'user_id' });
denunciation.belongsTo(user, { foreignKey: 'user_id' });

denunciation.hasMany(report, { foreignKey: 'denunciation_id' });
report.belongsTo(denunciation, { foreignKey: 'denunciation_id' });

user.hasMany(report, { foreignKey: 'user_id' });
report.belongsTo(user, { foreignKey: 'user_id' });
```

- Um **usuário** pode registrar muitas **denúncias** e **relatórios**.
- Cada **denúncia** pode conter vários **relatórios**.
- Cada **relatório** pertence a um único **usuário** e a uma única **denúncia**.

Documentação do Controller: `/denunciations/denunciationsController.js`

Este controller é responsável por gerenciar todo o fluxo relacionado às **denúncias sanitárias**, incluindo cadastro, edição, atribuição de fiscais, visualização, e busca.

🔹 `GET /cadastro/denuncia`

- **Objetivo**: Exibe o formulário de cadastro de nova denúncia.
- **Lógica**:
  - Verifica se há alguma denúncia já registrada no banco.
  - Caso não exista, inicia a contagem a partir de `111`.
  - Se existirem denúncias no **ano atual**, gera o próximo número sequencial.
- **View renderizada**: `denunciation/new`

🔹 `POST /denuncia/registrar`

- **Objetivo**: Registra uma nova denúncia no banco de dados.
- **Validações**:
  - Verifica se já existe denúncia com o mesmo **endereço, número e bairro** com status diferente de "Finalizada".
  - Se houver, exibe mensagem informando e evita duplicidade.
- **Campos esperados (body)**:
  - `year`, `number`, `registration_type`, `title`, `endereco`, `numero`, `bairro`, `complemento`, `description`
- **Redirecionamento**: Página inicial (`/`), ou formulário novamente com mensagem.

🔹 `GET /denuncia/:id`

- **Objetivo**: Visualizar uma denúncia específica com seus relatórios e fiscal associado.
- **Modelos incluídos**:
    - `reportsModel` (relatórios)
    - `userModel` (usuário/fiscal)
- **View renderizada**: `denunciation/show`

🔹 `GET /denuncia/:id/edit`

- **Objetivo**: Exibe o formulário de edição de uma denúncia existente.
- **Validação**: Verifica se a denúncia existe antes de renderizar.
- **View renderizada**: `denunciation/edit`

🔹 `POST /denuncia/edit/:id`

- **Objetivo**: Salva alterações feitas na denúncia.
- **Campos esperados (body)**:
    - `registration_type`, `title`, `endereco`, `numero`, `bairro`, `complemento`, `description`, `status`
- **Redirecionamento**: Página da denúncia editada (`/denuncia/:id`)

🔹 `GET /atribuir`

- **Objetivo**: Lista todas as denúncias **sem fiscal atribuído** e todos os fiscais ativos disponíveis para atribuição.
- **Includes**:
    - `reportsModel` com `userModel` (detalhes do relatório e fiscal)
- **View renderizada**: `denunciation/associate`

🔹 `GET /atribuir/:id`

- **Objetivo**: Página para selecionar um fiscal específico para uma denúncia.
- **Busca**:
    - Denúncia pelo `id`, incluindo o fiscal atual.
    - Lista de usuários ativos.
- **View renderizada**: `denunciation/assign`

🔹 `POST /atribuir/:id`

- **Objetivo**: Realiza a atribuição de um fiscal à denúncia.
- **Campos esperados (body)**:
    - `userId`
- **Ações realizadas**:
    - Atualiza a denúncia com `user_id`.
    - Registra relatório informando a atribuição automática (`"Denúncia atribuída ao fiscal..."`).
- **Redirecionamento**: `/atribuir` com query de sucesso ou erro.

🔹 `GET /buscar`

- **Objetivo**: Exibe a interface de busca avançada de denúncias.
- **Carrega**:
    - Lista de usuários (fiscais)
- **View renderizada**: `denunciation/search`

🔹 `GET /buscar/resultados`

- **Objetivo**: Processa os filtros da busca e retorna denúncias que correspondem aos critérios.
- **Filtros possíveis (query params)**:
    - `year`, `number`, `endereco`, `numero`, `bairro`, `status`, `registration_type`, `userId`
- **Tratamento especial**:
    - Limpa prefixos comuns do campo `endereco` (Rua, Av., Travessa, etc.)
- **Ordenação**:
    - Por `year DESC`, `number DESC`
- **View renderizada**: `denunciation/search` com resultados

Observações Gerais

- Os **status de denúncia** e os **tipos de origem** são definidos nas constantes:
    - `DENUNCIATION_STATUS`
    - `DENUNCIATION_SENDER`
- O model `denunciationsModel` representa a tabela principal `denuncias`.
- O relacionamento com `reportsModel` e `userModel` garante rastreabilidade completa.
- O sistema utiliza Sequelize para manipulação do banco de dados MariaDB.

Documentação do Controller: `/reports/reportsController.js`

Este controller é responsável pela **criação, visualização e edição de relatórios** associados a uma denúncia. Relatórios são registros das ações realizadas pelos fiscais e incluem tanto **observações textuais** quanto **alterações de status**.

🔹 `POST /denuncia/:id/adicionar-relatorio`

- **Objetivo**: Adiciona um novo relatório à denúncia especificada. Pode conter:
  - Um texto descritivo simples.
  - Uma mudança de status da denúncia (opcional).
- **Regras de negócio**:
  - Se nenhum campo for preenchido (`description` e `status`), retorna erro.
  - Se houver alteração de status, um relatório automático é criado com a descrição da transição.
- **Parâmetros**:
  - `:id` → ID da denúncia
- **Body esperado**:
  - `description` (opcional) – texto digitado pelo fiscal
  - `status` (opcional) – novo status da denúncia
  - `user_id` (obrigatório) – ID do fiscal que está realizando a ação
- **Criações**:
  - Um relatório com `type: REPORT_TYPE.TEXT` se `description` existir
  - Um relatório com `type: REPORT_TYPE.STATUS_CHANGE` se `status` for alterado
- **Redirecionamento**: `/denuncia/:id`

🔹 `GET /relatorio/:id`

- **Objetivo**: Exibe o formulário de **edição de um relatório** específico.
- **Parâmetros**:
    - `:id` → ID do relatório
- **Busca**:
    - O relatório (`reportsModel`)
    - A denúncia relacionada (`denunciationsModel`)
- **View renderizada**: `reports/show`
- **Campos renderizados**:
    - `description`, `type`, `denunciation_id`, status atual

🔹 `POST /relatorio/:id/editar`

- **Objetivo**: Atualiza o conteúdo de um relatório já existente.
- **Parâmetros**:
    - `:id` → ID do relatório
- **Body esperado**:
    - `description` – nova descrição (se tipo for texto)
    - `status` – novo status da denúncia (apenas se tipo for `STATUS_CHANGE`)
- **Lógica**:
    - Se o relatório for do tipo `STATUS_CHANGE`, a descrição é automaticamente sobrescrita com a nova transição de status.
    - A denúncia é atualizada com o novo status, se aplicável.
- **Redirecionamento**: `/denuncia/:id`

Observações Técnicas

- Todos os relatórios possuem:
    - `description` → descrição textual da ação realizada
    - `type` → definido pelas constantes `REPORT_TYPE`
    - `user_id` → autor do relatório
    - `denunciation_id` → denúncia associada
- Existem dois **tipos de relatórios**:
    - `TEXT (1)` – Texto informativo ou descritivo livre
    - `STATUS_CHANGE (0)` – Indica mudança de status da denúncia, usado para rastreabilidade
- Os relatórios são sempre associados a uma **denúncia específica** e a um **usuário (fiscal)**.

Documentação do Controller: `/users/usersController.js`

Este controller gerencia as **operações relacionadas a usuários do sistema**, incluindo:

- Cadastro de novos usuários (fiscais)
- Edição de usuários ativos
- Visualização da área individual de trabalho dos fiscais
- Gerenciamento da lista de denúncias atribuídas

🔹 `GET /cadastro/usuario`

- **Objetivo**: Exibe a interface para cadastrar e visualizar usuários (fiscais) já registrados no sistema.
- **Busca**:
    - Todos os usuários, ordenados alfabeticamente pelo nome.
- **View renderizada**: `users/new`

🔹 `POST /usuario/registrar`

- **Objetivo**: Registra um novo usuário no banco de dados.
- **Body esperado**:
    - `username` (campo obrigatório)
- **Redirecionamento**: Para `/cadastro/usuario` após a criação.
- **Validação**:
    - Se `username` estiver vazio, redireciona para `/` sem criar.

🔹 `GET /area-fiscal`

- **Objetivo**: Exibe a tela de **seleção de fiscal**.
- **Busca**:
    - Todos os usuários cadastrados, listando apenas `id` e `name`.
- **View renderizada**: `users/select`

🔹 `GET /area-fiscal/:id`

- **Objetivo**: Mostra o **painel individual do fiscal**, com todas as denúncias atribuídas a ele.
- **Parâmetros**:
    - `:id` – ID do fiscal selecionado
    - `status` – filtro de status (query param opcional, padrão `REGISTRADA`)
    - `page` – controle de paginação (query param, padrão 1)
- **Busca**:
    - Todas as denúncias do fiscal, com paginação (50 por página)
    - Inclui relatórios da denúncia
- **View renderizada**: `users/home`
- **Campos renderizados**:
    - `fiscal` (objeto completo do usuário)
    - `denuncias` (lista paginada)
    - `selectedStatus`, `hasMore`, `nextPage`

🔹 `GET /editar-usuario/:id`

- **Objetivo**: Exibe o formulário de **edição de um usuário existente**.
- **Parâmetros**:
  - `:id` – ID do usuário a ser editado
- **Busca**:
  - Dados completos do usuário pelo `id`
- **View renderizada**: `users/edit`

🔹 `POST /usuario/atualizar/:id`

- **Objetivo**: Atualiza os dados do usuário.
- **Parâmetros**:
  - `:id` – ID do usuário
- **Body esperado**:
  - `name` – novo nome do usuário
  - `active` – se presente, inverte o status atual do campo `ativo`
- **Validações**:
  - Verifica se o usuário existe antes de atualizar
- **Redirecionamento**: `/cadastro/usuario` após sucesso

Observações Técnicas

- O campo `ativo` (booleano) controla se o usuário está habilitado a ser atribuído a denúncias.
- A view `users/home` faz paginação manual com limite fixo de 50 itens por página.
- Usuários cadastrados são utilizados em:
    - Associação a denúncias
    - Relatórios
    - Área fiscal individual

Documentação Técnica: Paginação de Denúncias

Arquivos envolvidos

- `/index.js`
- `/middlewares/pagination.js`
- `/loadings/loadingsController.js`
- `models/denunciationsModel.js` (modelo de denúncias)
- `models/usersModel.js` (modelo de usuários)

Middleware: Paginação

Arquivo: `/middlewares/pagination.js`

```
const DEFAULT_LIMIT = 50;

module.exports = (req, res, next) => {
    const offset = parseInt(req.query.offset) || 0;
    const limit = parseInt(req.query.limit) || DEFAULT_LIMIT;

    req.pagination = { limit, offset };
    next();
};
```

Descrição:
Este middleware extrai os parâmetros `offset` e `limit` da query string e os injeta no objeto `req.pagination`, tornando a paginação reutilizável em diferentes rotas da aplicação.

- `offset`: deslocamento inicial da consulta (default: `0`)
- `limit`: quantidade de registros por página (default: `50`)

Rota de API: Buscar Denúncias Paginadas

Arquivo: `/loadings/loadingsController.js`

```
router.get('/api/denuncias', async (req, res) => {
    const { limit = 50, offset = 0 } = req.query;

    try {
        const denuncias = await denunciationsModel.findAll({
            limit: parseInt(limit),
            offset: parseInt(offset),
            include: [{
                model: usersModel,
                as: 'user',
                attributes: ['name']
            }],
            order: [
                ['year', 'DESC'],
                ['number', 'DESC']
            ]
        });

        res.json(denuncias);
    } catch (error) {
        console.error('Erro ao buscar denúncias:', error);
        res.status(500).send('Erro ao carregar denúncias.');
    }
});
```

Descrição:
Retorna uma lista de denúncias paginadas via `limit` e `offset`. Também inclui os nomes dos fiscais responsáveis.

🏠 Página Inicial: Paginação de Listagem

Arquivo: `/index.js`

```
app.get('/', async (req, res) => {
    try {
        const limit = parseInt(req.query.limit) || 50;
        const offset = parseInt(req.query.offset) || 0;

        const denuncias = await denunciation.findAll({
            include: [{
                model: user,
                as: 'user',
                attributes: ['name']
            }],
            order: [
                ['year', 'DESC'],
                ['number', 'DESC']
            ],
            limit: limit,
            offset: offset
        });

        res.render('index', {
            denuncias: denuncias,
            DENUNCIATION_SENDER: DENUNCIATION_SENDER,
            offset: offset + limit,
            limit: limit
        });

    } catch (error) {
        console.error('Erro ao buscar denúncias:', error);
        res.status(500).send('Erro ao carregar denúncias.');
    }
});
```

Descrição:
Carrega a página inicial do sistema com as denúncias mais recentes, em ordem decrescente de ano e número. A paginação é feita via query string, usando os parâmetros `offset` e `limit`. Cada clique em "ver mais" avança a lista.

✅ Exemplo de Requisições

API:

```
GET /api/denuncias?limit=50&offset=100
```

Página inicial:

```
GET /?limit=50&offset=100
```

Considerações Finais sobre a Área do Fiscal e Perspectivas de Melhoria

A área do fiscal foi projetada para facilitar o acesso às denúncias atribuídas a cada usuário responsável pela fiscalização, com filtros básicos por status da denúncia e ordenação por ano e número, permitindo uma navegação inicial entre os registros. Atualmente, a visualização exibe as denúncias completas sem divisão em páginas.

No entanto, considerando o volume potencialmente grande de denúncias, a implementação de paginação na área do fiscal é uma melhoria importante a ser incorporada futuramente. A paginação tornaria a experiência do fiscal mais eficiente, ao permitir a consulta gradual das denúncias em blocos controlados, reduzindo o tempo de carregamento e melhorando a performance geral do sistema.

Esse recurso também facilitaria o controle de navegação entre as páginas de denúncias, com parâmetros que definem o número de itens por página e o deslocamento dos registros, possibilitando filtros dinâmicos sem prejuízo para a usabilidade.

Em resumo, a paginação na área do fiscal é uma evolução necessária para garantir escalabilidade e melhor desempenho, especialmente à medida que o sistema cresce em número de denúncias e usuários. Essa melhoria está prevista para versões futuras, com foco em otimizar a rotina dos fiscais e tornar o sistema mais ágil e responsivo.

Esta documentação registra esse ponto como uma recomendação de aprimoramento, visando a entrega de uma solução cada vez mais robusta e alinhada às necessidades reais dos usuários.