Documentation - Security Management System - Wayne Enterprises.

Documentação do Sistema de Gerenciamento de Segurança - Indústrias Wayne.

Introdução

O Sistema de Gerenciamento de Segurança das Indústrias Wayne é uma aplicação web projetada para facilitar o controle de acesso e o gerenciamento de recursos dentro da empresa. Com funcionalidades como registro de usuários, login seguro e visualização de recursos em gráficos, a aplicação visa melhorar a eficiência e a segurança da gestão de recursos.

Objetivos

- *Cadastro de Usuários:* > Permitir a criação de novos usuários com diferentes níveis de acesso.

- *Gerenciamento de Recursos:* > Facilitar a adição, edição e remoção de recursos, assegurando que apenas usuários autorizados possam realizar essas ações.

- *Visualização de Dados:* > Fornecer uma representação gráfica da quantidade de recursos, permitindo uma análise rápida.

Estrutura do Projeto

/wayne-security-system

|-- index.html # Página principal da aplicação

|-- style.css # Estilos e layout da aplicação

|-- script.js # Lógica e comportamento da aplicação

1. index.html

O arquivo index.html é a base da interface do usuário. É onde todos os elementos visuais são definidos e organizados.

Estrutura do HTML:

- **Cabeçalho (<head>):**
  - *Meta Tags:*
    - Define o charset para UTF-8 e configura a viewport para responsividade.
    - Inclui uma descrição do sistema para SEO.
  - *Links de Estilo:*
    - Importa a fonte Roboto do Google Fonts.
    - Inclui o Normalize.css para garantir uma aparência consistente entre navegadores.
    - Inclui o style.css, que contém os estilos personalizados da aplicação.
  - *Título da Página:*
    - Define o título que aparecerá na aba do navegador.

- **Corpo da Página (<body>):**
  - **Cabeçalho (<header>):**
    - Contém o logotipo da Indústrias Wayne, título e descrição do sistema.
  - **Corpo Principal (<main>):**
    - **Controle de Acesso (#controleAcesso):**
      - Formulário para login com campos para usuário e senha, além de botões para ações (login, esqueci a senha, novo usuário).
    - **Identificação do Usuário (#userIdentification):**
      - Exibe nome e nível de credencial do usuário logado, além de um botão para logout.
    - **Cadastro de Usuário (#cadastro):**

- Formulário para registrar novos usuários, incluindo campos para nome, e-mail, senha e tipo de usuário.

- **Gerenciamento de Recursos (#gerenciamentoRecursos):**

- Formulário para adicionar e gerenciar recursos, incluindo categorias, quantidade, descrição e prioridade.

- **Lista de Recursos (#listaRecursosCard):**

- Área dinâmica para exibir recursos cadastrados em uma lista.

- **Painel de Controle (#painelControle):**

- Área para exibir gráficos que representam a quantidade de recursos cadastrados.

- **Rodapé (<footer>):**

- Contém informações de copyright do sistema.

- **Botão de Voltar ao Topo (#voltarAoTopo):**

- Botão que aparece ao rolar a página, permitindo que o usuário retorne ao topo rapidamente.

- *Scripts:*

- Inclui a biblioteca Chart.js para gráficos e o script.js para a lógica da aplicação.

2. style.css

O arquivo style.css contém todos os estilos da aplicação, organizando a apresentação visual e garantindo uma interface amigável.

Estrutura de Estilos:

- *Cores e Temas:*

- Define variáveis CSS para gerenciar cores de fundo, texto e botões, facilitando a manutenção e as modificações futuras.

```css
:root {
    --cor-fundo: rgba(255, 255, 255, 0.9);
    --cor-primaria: #4CAF50;
    --cor-secundaria: #3498DB;
    --cor-terciaria: #e74c3c;
    --cor-texto: #333;
    --cor-texto-branco: #fff;
}
```

- *Estilos Globais:*
  - Define a fonte padrão da aplicação, margens e padding para todos os elementos, garantindo uma aparência consistente.

```css
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
    font-family: 'Roboto', sans-serif;
}
```

- *Estilos do Cabeçalho:*
  - Estiliza o cabeçalho, incluindo layout flexível para centralizar o logo e o texto.

```css
.header-container {
    display: flex;
    justify-content: center;
```

```
    align-items: center;
    width: 100%;
    max-width: 1300px;
    margin: 20px auto;
    padding: 0 25px;
    background: rgba(0, 0, 0, 0.1);
    border-radius: 10px;
}
```

- *Formulários e Botões:*
  - Estilos para campos de entrada e botões, incluindo efeitos de hover, melhorando a usabilidade.

css

```
.input-group input,
.input-group select {
    width: 100%;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
    transition: border 0.3s;
}

.input-group input:focus,
.input-group select:focus {
    border-color: #e74c3c;
}
```

- *Cards:*

- Estilização dos cards para gerenciamento de recursos e exibição de dados.

```
css
.painel-card {
    background-color: rgba(255, 255, 255, 0.3);
    border-radius: 10px;
    padding: 20px;
    box-shadow: 0 4px 20px var(--cor-sombra);
}
```

3. script.js

O arquivo script.js contém a lógica e o comportamento da aplicação, gerenciando a interação do usuário e a manipulação de dados.

Principais Funcionalidades:

- *Gerenciamento de Usuários:*
  - **carregarUsuarios()**:
    - Carrega a lista de usuários salvos do localStorage ao iniciar a aplicação.

```
javascript
const carregarUsuarios = () => {
    const usuariosSalvos = localStorage.getItem('usuarios');
    if (usuariosSalvos) {
        usuarios = JSON.parse(usuariosSalvos);
    }
};
```

- **cadastrarUsuario(usuario, senha, confirmacaoSenha, tipo, email, codigoAdmin)**:

  - Registra novos usuários com validação de campos.

  - Verifica se o usuário ou e-mail já existe para evitar duplicações.

  - Exibe mensagens de erro apropriadas se a validação falhar.

javascript

```
const cadastrarUsuario = (usuario, senha, confirmacaoSenha, tipo, email, codigoAdmin) => {
  // Validação de campos obrigatórios
  if (!usuario || !senha || !confirmacaoSenha || !tipo || !email) {
    alert("Todos os campos devem ser preenchidos!");
    return;
  }
  // Validação do email
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  if (!emailRegex.test(email)) {
    alert("Por favor, insira um email válido!");
    return;
  }
  // Verificação de senhas iguais
  if (senha !== confirmacaoSenha) {
    alert("As senhas não coincidem!");
    return;
  }
  // Verificação de usuário existente
  const usuarioExistente = usuarios.find(u => u.nome === usuario);
  if (usuarioExistente) {
    alert("Já existe um usuário cadastrado com este nome!");
    return;
  }
```

```
    // Cadastro do usuário
    usuarios.push({ nome: usuario, senha, tipo, email, codigoAdmin });
    localStorage.setItem('usuarios', JSON.stringify(usuarios));
    alert("Cadastro realizado com sucesso!");
};
```

- **fazerLogin()**:
  - Autentica o usuário verificando as credenciais inseridas no formulário.
  - Exibe uma mensagem de boas-vindas e a seção de identificação do usuário ao logar com sucesso.

```
javascript
const fazerLogin = () => {
    const usuario = document.querySelector("#usuario").value;
    const senha = document.querySelector("#senha").value;

    // Verificação de credenciais
    const usuarioAutenticado = usuarios.find(u => u.nome === usuario && u.senha === senha);
    if (usuarioAutenticado) {
        document.querySelector("#boasVindas").innerHTML = `
            Bem-vindo, <span id="usuarioNome" class="usuario-nome">${usuarioAutenticado.nome}</span>!
            Você está logado como: <span id="nivelCredencial" class="nivel-credencial">${usuarioAutenticado.tipo}</span>
        `;
        document.querySelector("#boasVindas").style.display = "block";
        // Exibir painel correspondente
    } else {
        alert("Usuário ou senha incorretos.");
    }
```

};

- **esqueciSenhaButton**:
  - Permite que o usuário recupere a senha, solicitando o e-mail cadastrado.

javascript
```
const esqueciSenhaButton = document.querySelector('#esqueciSenhaButton');
esqueciSenhaButton.addEventListener('click', () => {
    let emailDigitado;
    do {
        emailDigitado = prompt('Digite seu e-mail cadastrado:');
    } while (emailDigitado === null || emailDigitado.trim() === '' || !tratarRecuperacaoSenha(emailDigitado));
});
```

- *Gerenciamento de Recursos:*
  - **carregarRecursos()**:
    - Carrega recursos salvos do localStorage ao iniciar a aplicação.

javascript
```
const carregarRecursos = () => {
    const recursosSalvos = localStorage.getItem('recursos');
    if (recursosSalvos) {
        recursos = JSON.parse(recursosSalvos);
    }
};
```

- **adicionarRecurso()**:

- Adiciona um novo recurso, validando todos os campos antes de salvar.
- Exibe um alerta de sucesso ou falha após a operação.

javascript

```
const adicionarRecurso = () => {
    const nome = document.querySelector("#nomeRecurso").value;
    const categoria = document.querySelector("#categoriaRecurso").value;
    const quantidade = document.querySelector("#quantidadeRecurso").value;
    const descricao = document.querySelector("#descricaoRecurso").value;
    const prioridade = document.querySelector("#prioridadeRecurso").value;

    if (!nome || !categoria || !quantidade || !descricao || !prioridade) {
        alert("Por favor, preencha todos os campos do Gerenciamento de Recursos.");
        return;
    }

    const novoRecurso = {
        nome,
        categoria,
        quantidade: parseInt(quantidade),
        descricao,
        prioridade
    };

    recursos.push(novoRecurso);
    localStorage.setItem('recursos', JSON.stringify(recursos));
    alert("Recurso adicionado com sucesso!");
};
```

- **editarRecurso(nome)**:

  - Permite a edição de um recurso existente, verificando as permissões do usuário logado.

javascript

```
const editarRecurso = (nome) => {
    const recurso = recursos.find(r => r.nome === nome);
    if (recurso) {
        const novoNome = prompt("Digite o novo nome do recurso:", recurso.nome);
        if (novoNome) {
            recurso.nome = novoNome; // Atualiza o nome do recurso
            localStorage.setItem('recursos', JSON.stringify(recursos));
            alert("Recurso editado com sucesso!");
        }
    } else {
        alert("Recurso não encontrado.");
    }
};
```

- **removerRecurso(nome)**:

  - Remove um recurso específico após confirmação do usuário.

javascript

```
const removerRecurso = (nome) => {
    if (confirm(`Tem certeza que deseja remover o recurso "${nome}"?`)) {
        recursos = recursos.filter(recurso => recurso.nome !== nome);
        localStorage.setItem('recursos', JSON.stringify(recursos));
        alert("Recurso removido com sucesso!");
    }
```

};

- **filtrarRecursos()**:

  - Permite ao usuário filtrar recursos com base em critérios fornecidos pelo usuário.

javascript

```
const filtrarRecursos = () => {
    const criterio = prompt("Digite o critério para filtrar os recursos (nome, categoria, etc.):");
    const recursosFiltrados = recursos.filter(recurso => recurso.nome.includes(criterio));
    if (recursosFiltrados.length > 0) {
        alert(`Recursos encontrados: ${recursosFiltrados.map(r => r.nome).join(', ')}`);
    } else {
        alert("Nenhum recurso encontrado.");
    }
};
```

- **atualizarListaRecursos(lista)**:

  - Atualiza a exibição da lista de recursos na interface, criando elementos dinâmicos.

javascript

```
const atualizarListaRecursos = () => {
    const listaRecursos = document.querySelector("#listaRecursos");
    listaRecursos.innerHTML = ''; // Limpa a lista existente

    recursos.forEach(recurso => {
```

```
        const item = document.createElement("li");
        item.textContent = recurso.nome; // Adiciona o nome do recurso à lista
        listaRecursos.appendChild(item);
    });
};
```

- *Interação com a UI:*
  - Manipula a exibição de diferentes seções (como login, cadastro, gerenciamento) com base nas ações do usuário.
  - Vincula eventos a botões e formulários para acionar funções específicas (login, logout, adicionar recurso).
  - *Eventos de Clique e Mudança:*
    - Ações como clique em botões de cadastro e login, além de mudanças no tipo de usuário, que afetam a exibição de elementos na interface.
  - *Scroll:*
    - Exibe o botão de "Voltar ao Topo" conforme o usuário rola a página.

javascript

```
document.querySelector("#logoutButton").addEventListener("click", fazerLogout);
document.querySelector("#voltarAoTopo").addEventListener("click", () => {
    window.scrollTo({ top: 0, behavior: 'smooth' });
});
```

Considerações de Segurança

- *Validação de Entrada:*
  - Todos os campos em formulários devem ser validados para evitar entradas inválidas e garantir a integridade dos dados.

- *Armazenamento Local:*

  - O uso do localStorage deve ser feito com cautela, pois os dados não são criptografados e podem ser acessados por scripts maliciosos. Para produção, considere usar um backend seguro com autenticação e autorização.

- *Senhas:*

  - As senhas devem ser tratadas com segurança. Em uma aplicação real, implemente armazenamento seguro e criptografia para senhas de usuários. Use métodos como hashing (por exemplo, bcrypt) para armazenar senhas.

- *Controle de Acesso:*

  - Limite o acesso a funcionalidades sensíveis (como edição e remoção de recursos) com base no nível de credencial do usuário.

Considerações de Usabilidade

- *Feedback do Usuário:*

  - Forneça mensagens claras após ações do usuário, como sucesso ou falha nas operações, para melhorar a experiência do usuário.

- *Acessibilidade:*

  - Certifique-se de que a aplicação é utilizável por todos os usuários, incluindo aqueles com deficiências, utilizando práticas de acessibilidade, como descrições alternativas para imagens e navegação por teclado.

Exemplo de Uso

1. *Cadastro de Usuário:*

   - O usuário preenche o formulário de cadastro e clica em "Cadastrar". O sistema valida os campos e registra o usuário no localStorage.

2. *Login:*

- O usuário insere seu nome de usuário e senha, clica em "Login" e, se as credenciais estiverem corretas, é redirecionado para a tela principal com uma mensagem de boas-vindas.

3. *Gerenciamento de Recursos:*

- Após o login, o usuário pode adicionar, editar ou remover recursos. As alterações são refletidas imediatamente na lista exibida na interface.

4. *Visualização de Dados:*

- O usuário pode visualizar dados em gráficos, permitindo uma análise rápida da quantidade de recursos cadastrados.

Diagrama de Arquitetura

```

1. Navegador do Usuário

- Interface do Usuário (UI)

- Formulários:

- Login:

- Campos: Usuário, Senha.

- Botões: Login, Esqueci a Senha, Novo Usuário.

- Cadastro:

- Campos: Nome, E-mail, Senha, Confirmar Senha, Tipo de Usuário.

- Botão: Cadastrar.

- Gerenciamento de Recursos:

- Campos: Nome do Recurso, Categoria, Quantidade, Descrição, Prioridade.

- Botões: Adicionar Recurso, Editar Recurso, Remover Recurso.

- Área de Visualização:

- Listagem de Recursos.

- Gráficos de Quantidade de Recursos.
```

2. Frontend (HTML, CSS, JavaScript)

- HTML (index.html):
  - Estrutura da página com seções bem definidas para cada funcionalidade.
- CSS (style.css):
  - Estilos para layout responsivo e design estético.
  - Variáveis CSS para cores e fontes.
- JavaScript (script.js):
  - Funções principais:
    - carregarUsuarios(): Carrega usuários do LocalStorage.
    - cadastrarUsuario(usuario, senha, confirmacaoSenha, tipo, email): Registra novos usuários com validação.
    - fazerLogin(): Autentica o usuário e exibe mensagem de boas-vindas.
    - adicionarRecurso(): Adiciona novos recursos com validação de entrada.
    - editarRecurso(nome): Edita recursos existentes após confirmação.
    - removerRecurso(nome): Remove recursos com confirmação do usuário.
    - atualizarListaRecursos(): Atualiza a exibição da lista de recursos na UI.
    - filtrarRecursos(): Permite filtrar recursos com base em critérios específicos.

3. LocalStorage

- Armazenamento de dados temporários:
  - Dados de Usuários:
    - Estrutura: { nome: "", senha: "", tipo: "", email: "" }.
  - Dados de Recursos:
    - Estrutura: { nome: "", categoria: "", quantidade: 0, descricao: "", prioridade: "" }.
- Permite persistência entre sessões do usuário no navegador.

4. Backend (API REST)

- Autenticação de Usuários:
  - Endpoints sugeridos (não implementados no código atual):
    - POST /api/login: Verifica credenciais do usuário.

- POST /api/register: Registra novos usuários.
- Operações CRUD (Create, Read, Update, Delete) para Recursos:
- Endpoints sugeridos (não implementados no código atual):
- POST /api/recursos: Adiciona um novo recurso.
- GET /api/recursos: Retorna a lista de recursos.
- PUT /api/recursos/{id}: Atualiza um recurso existente.
- DELETE /api/recursos/{id}: Remove um recurso.
- Comunicação com o Banco de Dados:
- Processa requisições e manipula dados de forma segura.

5. Banco de Dados (Ex: MongoDB, MySQL)
- Estrutura de dados:
- Coleção de Usuários:
- Campos: { id, nome, senha (hash), tipo, email }.
- Coleção de Recursos:
- Campos: { id, nome, categoria, quantidade, descricao, prioridade }.
- Armazenamento de dados de forma persistente e estruturada.

```

### **Fluxo de Dados:**

- O usuário interage com o *Navegador do Usuário* e preenche formulários.
- As requisições são enviadas para o *Frontend*, que processa os dados e atualiza a UI.
- O *LocalStorage* é utilizado para armazenar informações temporárias, permitindo recuperação rápida de dados.
- O *Frontend* comunica-se com o *Backend* para operações de autenticação e manipulação de recursos.
- O *Backend* realiza operações no *Banco de Dados* e retorna os resultados ao *Frontend*.
```

- O *Banco de Dados* armazena dados de forma persistente, garantindo a integridade e a segurança das informações.

Considerações Adicionais:

- *Segurança*: Implementar hashing de senhas e validação de entrada para prevenir SQL Injection e XSS.

- *Escalabilidade*: Planejar a estrutura da API para suportar um aumento no número de usuários e recursos.

- *Acessibilidade*: Garantir que a interface do usuário seja acessível a todos os usuários, incluindo aqueles com deficiências.

Se precisar de mais informações ou ajustes específicos, estou à disposição! Aqui está a representação visual do diagrama de arquitetura para o seu Sistema de Gerenciamento de Segurança:

Este diagrama ilustra a interação entre os componentes do sistema, incluindo o navegador do usuário, o frontend, o armazenamento local, o backend e o banco de dados. Se precisar de mais ajustes ou de uma explicação sobre qualquer parte específica do diagrama, estou à disposição!

Referências e Recursos

- *Chart.js*: [Documentação do Chart.js](https://www.chartjs.org/docs/latest/)

- *Normalize.css*: [Documentação do Normalize.css](https://necolas.github.io/normalize.css/)

- *Google Fonts*: [Google Fonts](https://fonts.google.com/)

Tópicos de Manutenção e Contribuição

- *Manutenção do Código:*

  - Revisar e atualizar o código periodicamente para garantir que ele esteja alinhado com as melhores práticas e padrões de segurança.

- *Contribuição:*

  - Se o projeto for colaborativo, forneça diretrizes para outros desenvolvedores contribuírem, como criar pull requests e seguir um padrão de codificação.

Conclusão

Esta documentação fornece uma visão abrangente e detalhada da estrutura e funcionalidade do sistema de gerenciamento de segurança desenvolvido para as Indústrias Wayne. A aplicação oferece um sistema interativo para login, cadastro de usuários, gerenciamento de recursos e visualização de dados em gráficos.