**Documentação Completa do Projeto Secure Web Platform**

**1. Visão Geral do Projeto**

O projeto é uma **plataforma web segura** construída em **Node.js** utilizando **Express.js**, **EJS** para templates, **PostgreSQL** como banco de dados e diversos middlewares de segurança.

O objetivo é permitir que usuários se registrem, façam login, enviem e baixem arquivos, e tenham um **dashboard de auditoria** com todas as ações registradas.

O sistema prioriza **segurança**, incluindo:

* CSRF (Cross-Site Request Forgery)
* Sessões com cookies HTTP-only e SameSite
* Proteção contra ataques de força bruta em login (rate limiting)
* Hash de senhas usando bcrypt
* Auditoria detalhada de eventos

**2. Estrutura de Pastas (Recomendada)**

secure-web-project/

├─ app.js # Arquivo principal do servidor

├─ db.js # Configuração do pool PostgreSQL

├─ auth.js # Funções de registro e login de usuários

├─ utils.js # Funções utilitárias (sanitize, safeJoin)

├─ uploads/ # Diretório para armazenar arquivos enviados

├─ views/ # Templates EJS

│ ├─ layout.ejs

│ ├─ login.ejs

│ ├─ register.ejs

│ └─ dashboard.ejs

├─ public/ # Arquivos estáticos (CSS, JS, imagens)

├─ package.json

├─ .env # Variáveis de ambiente

**3. Banco de Dados**

O projeto utiliza **PostgreSQL** com duas tabelas principais e sequências automáticas:

**3.1 Tabela users**

| **Coluna** | **Tipo** | **Observações** |
| --- | --- | --- |
| id | SERIAL PRIMARY KEY | Gerado automaticamente pelo PostgreSQL |
| username | VARCHAR(50) | Único, obrigatório |
| email | VARCHAR(100) | Opcional |
| password\_hash | VARCHAR(255) | Hash da senha usando bcrypt |
| created\_at | TIMESTAMP | Default CURRENT\_TIMESTAMP |

**Uso:** Armazena os usuários registrados. A senha nunca é salva em texto.

**3.2 Tabela resources**

| **Coluna** | **Tipo** | **Observações** |
| --- | --- | --- |
| id | SERIAL PRIMARY KEY | Associado à sequence resources\_id\_seq |
| user\_id | INTEGER | FK → users.id |
| stored\_filename | VARCHAR(255) | Nome do arquivo no servidor |
| original\_name | VARCHAR(255) | Nome original do arquivo |
| created\_at | TIMESTAMP | Default CURRENT\_TIMESTAMP |

**Uso:** Armazena todos os arquivos enviados pelos usuários, relacionando cada arquivo ao dono.

**3.3 Tabela audit\_logs**

| **Coluna** | **Tipo** | **Observações** |
| --- | --- | --- |
| id | SERIAL PRIMARY KEY | Auditoria de eventos |
| user\_id | INTEGER | Opcional (NULL se não logado) |
| event\_type | VARCHAR(50) | Tipo de evento (login\_success, file\_upload, request, etc.) |
| event\_metadata | JSON | Dados extras (ex.: caminho, filename) |
| ip\_addr | VARCHAR(50) | IP de origem da requisição |
| created\_at | TIMESTAMP | Default CURRENT\_TIMESTAMP |

**Uso:** Registra **todas as ações importantes**, incluindo requisições, login/logout, uploads e downloads.

**4. Middlewares de Segurança e Utilitários**

**4.1 Helmet**

Protege contra **várias vulnerabilidades HTTP** (XSS, clickjacking, etc.)

**4.2 express-session + connect-pg-simple**

* Sessões armazenadas no PostgreSQL.
* Cookies seguros (httpOnly, SameSite=lax, secure em produção)
* Permite autenticação persistente.

**4.3 CSRF**

* Protegido via middleware csurf.
* Cada formulário renderiza <input type="hidden" name="\_csrf" value="<%= csrfToken %>">
* Em uploads (multipart/form-data) o middleware CSRF é aplicado **depois do multer** para evitar conflito.

**4.4 Rate Limiter**

* Limita **tentativas de login** a 10 por 15 minutos.

**4.5 Multer**

* Gerencia uploads de arquivos.
* Armazena no diretório uploads/ com nome único gerado via uuidv4.

**4.6 Funções utilitárias**

* safeJoin(base, target) → garante que downloads não escapem da pasta uploads/.
* sanitizeForLog() → limpa strings antes de salvar no audit log.

**5. Fluxos de Usuário**

**5.1 Registro**

1. GET /register → Renderiza formulário com CSRF token.
2. POST /register → Recebe username, email, password
   * Valida campos obrigatórios
   * Criptografa senha com bcrypt
   * Insere no users
   * Registra evento no audit\_logs (user\_registered)
   * Redireciona para login

**5.2 Login**

1. GET /login → Renderiza formulário com CSRF token
2. POST /login → Recebe username e password
   * Busca usuário no banco
   * Compara senha com bcrypt
   * Se correto:
     + Cria sessão (req.session.userId)
     + Registra login\_success
     + Redireciona para /dashboard
   * Se incorreto:
     + Registra login\_failed
     + Retorna erro 401

**5.3 Dashboard**

* GET /dashboard → protegido (requireAuth)
* Mostra últimos eventos do usuário e upload de arquivos
* CSRF token presente para formulários de upload e logout

**5.4 Upload de Arquivos**

* POST /upload → protegido
* Multer salva arquivo em uploads/ com nome único
* Registra no resources e audit\_logs (file\_upload)
* Redireciona para dashboard

**5.5 Download de Arquivos**

* GET /files/:id → protegido
* Verifica se arquivo pertence ao usuário
* Usa safeJoin para prevenir path traversal
* Registra file\_download no audit log
* res.download(filepath, original\_name)

**5.6 Logout**

* POST /logout → protegido
* Destroi sessão
* Limpa cookie
* Registra logout no audit log

**6. Segurança Adicional Implementada**

1. CSRF aplicado individualmente para cada rota (inclusive uploads)
2. Rate limiting no login
3. Sessão segura (httpOnly, SameSite=lax, secure)
4. Hash de senhas com bcrypt
5. Auditoria completa de todas ações importantes
6. Path traversal mitigado via safeJoin
7. Uploads isolados no diretório uploads/ com nomes aleatórios
8. Proteção via Helmet

**7. Problemas já resolvidos**

1. **Erro CSRF em uploads** → resolvido aplicando csurf após multer.
2. **Permissão negada em sequence PostgreSQL (resources\_id\_seq)** → resolvido concedendo USAGE, SELECT, UPDATE no sequence e INSERT na tabela para o usuário Node.
3. **Metadados exibindo [object Object]** → resolver na renderização do dashboard via JSON.stringify.

**8. Possíveis próximos passos e melhorias**

1. Limitar uploads por **tipo de arquivo** e **tamanho máximo**
2. Melhorar dashboard: exibir **links de download** para arquivos
3. Paginação ou filtros nos audit logs
4. Emails de notificação de ações importantes (opcional)
5. Implementar **roles** (admin, usuário normal)
6. Melhorar UI/UX do dashboard
7. Logging assíncrono com arquivos separados ou serviço externo
8. Backup automático dos uploads
9. Testes unitários e de integração

**9. Variáveis de Ambiente (.env)**

PORT=3000

DB\_USER=meu\_usuario

DB\_PASS=minha\_senha

DB\_HOST=localhost

DB\_PORT=5432

DB\_NAME=secure\_web

SESSION\_SECRET=segredo\_super\_secreto

UPLOAD\_DIR=./uploads

NODE\_ENV=development

**10. Comandos Úteis**

* **Instalar dependências**

npm install express express-ejs-layouts pg connect-pg-simple express-session helmet csurf express-rate-limit multer bcrypt dotenv

* **Rodar servidor**

node app.js

* **Criar tabelas no PostgreSQL**

CREATE TABLE users (

id SERIAL PRIMARY KEY,

username VARCHAR(50) UNIQUE NOT NULL,

email VARCHAR(100),

password\_hash VARCHAR(255) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE resources (

id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(id),

stored\_filename VARCHAR(255),

original\_name VARCHAR(255),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE SEQUENCE resources\_id\_seq;

CREATE TABLE audit\_logs (

id SERIAL PRIMARY KEY,

user\_id INTEGER,

event\_type VARCHAR(50),

event\_metadata JSON,

ip\_addr VARCHAR(50),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**✅ Resumo**

Este documento fornece **uma visão completa de toda a plataforma**, incluindo:

* Arquivos e estrutura do projeto
* Banco de dados, tabelas e relações
* Fluxos de usuário e middlewares
* Auditoria e segurança
* Problemas resolvidos e próximos passos

Com isso, qualquer IA ou desenvolvedor com zero conhecimento do projeto poderia **dar continuidade, adicionar funcionalidades ou corrigir bugs** sem precisar acessar sua base original.