# CyberAccessMonitor – Projeto C# (WPF)

## 1. Cenário e Problema

Durante falhas de energia, especialmente em condomínios residenciais, diversos sistemas digitais como painéis de acesso, câmeras, interfones e controles de segurança ficam vulneráveis. Esse ambiente propício a falhas operacionais pode abrir espaço para acessos indevidos ou perda de informações críticas, sem que haja qualquer tipo de registro.  
  
O CyberAccessMonitor foi desenvolvido para suprir essa lacuna: trata-se de uma aplicação WPF funcional que permite o registro e monitoramento local de eventos durante falhas energéticas, mantendo a rastreabilidade e integridade das informações, mesmo em situações offline.

## 2. Objetivo da Solução

Criar um sistema desktop simples e funcional, com foco em segurança, para registrar manualmente eventos relevantes ocorridos durante falhas de energia em ambientes com pouca supervisão técnica, como portarias ou centrais de controle predial.

## 3. Requisitos Funcionais

- Permitir login do usuário  
- Registrar eventos manuais (com descrição, data e origem)  
- Exibir lista de eventos registrados  
- Exportar eventos em formato JSON  
- Registrar logs internos com timestamp

## 4. Requisitos Não Funcionais

- Aplicação deve funcionar localmente, sem depender de internet  
- Interface clara e objetiva para uso em ambientes de baixa luminosidade  
- Estrutura modular com separação entre serviços, UI e modelos  
- Manutenção facilitada com boas práticas de desenvolvimento

## 5. Tecnologias Utilizadas

- Linguagem: C#  
- Plataforma: WPF (.NET Framework)  
- Serialização: Newtonsoft.Json  
- Armazenamento: arquivos locais JSON e TXT  
- IDE: Visual Studio 2022

## 6. Regras de Negócio

- Apenas usuários autenticados podem registrar eventos  
- Todo evento deve conter: descrição, origem e data  
- Logs devem ser gerados para todas ações relevantes ou exceções  
- O sistema deve exportar os dados sob demanda em JSON

## 7. Simulação de Erros

Todos os formulários utilizam `try-catch` para tratar erros comuns:  
- Campos obrigatórios vazios  
- Exportações com falha de disco  
- Login com credenciais inválidas

## 8. Boas Práticas Aplicadas

- Separação em camadas (UI, Service, Model)  
- Classes com responsabilidade única  
- Métodos bem nomeados, com retorno e parâmetros claros  
- Código indentado, limpo e comentado  
- Lógica reutilizada via serviços

## 9. Funcionalidades Atendidas

1. Login obrigatório  
2. Registro de eventos  
3. Exportação de eventos (.json)  
4. Geração de logs (.txt)  
5. Exibição de histórico de registros

## 10. Estrutura no GitHub

Repositório publicado contendo:  
- Código-fonte  
- README.md com instruções de execução  
- Estrutura organizada por pastas (Models, Services, Views)  
- Licença acadêmica livre

## 11. Vídeo Pitch

O vídeo de apresentação (link no README) contém:  
- Introdução ao problema  
- Apresentação da solução desenvolvida  
- Demonstração do uso real da aplicação  
- Destaque dos diferenciais e da aplicabilidade no mundo real