**Documentação do Projeto GridHack**

**1. Introdução**

O projeto GridHack tem como objetivo fornecer uma simulação interativa e pedagógica de identificação e isolamento de falhas em uma rede elétrica. Através de:

* **Aplicação Console (C#):** Permite o treino de operadores na identificação de falhas, isolamento de nós e restauração da rede, com validação de conectividade e registro de histórico.
* **Interface Mobile (React Native):** Oferece uma experiência mais visual e responsiva para simulação de cenários de falha em dispositivos móveis.

Essa proposta educa engenheiros, técnicos e estudantes sobre como garantir a confiabilidade e resiliência de sistemas de distribuição de energia.

**2. Escopo e Motivação**

**Motivação:**

* Redes elétricas são vulneráveis a falhas físicas como desastres naturais e ataques cibernéticos (SCADA) que podem gerar apagões em larga escala.
* É fundamental que operadores entendam rapidamente quais nós apresentam falhas e como conter o efeito cascata. Assim podendo reparar a rede de maneira rápida e efetiva.

**Escopo:**

* Implementar um simulador de rede elétrica básica (5 nós) com bordas de conectividade.
* Permitir autenticação simples de usuário, visualização do estado de cada nó, isolamento/restauração e cálculo de pontuação.
* Manter histórico persistido em JSON e oferecer relatório de estatística

**3. Requisitos Funcionais**

1. **Autenticação de Usuário 1.1.** Deve haver usuário e senha fixos para acesso inicial. 1.2. Apenas usuários autenticados podem interagir com a rede.
2. **Visualização Dinâmica da Rede 2.1.** Exibir, em modo texto com cores, cada nó classificado como:
   * **Falha** (vermelho, pulsante)
   * **OK** (verde)
   * **Isolado** (amarelo) 2.2. Exibir conectividade: linha de texto indicando "Rede Conectada: Sim/Não".
3. **Isolamento e Restauração de Nós 3.1.** O usuário seleciona um nó com falha e altera seu estado para "Isolado".  
   3.2. Se a ação de isolar for inválida (nó sem falha ou inexistente), exibir alerta. 3.3. Manter o usuário em um loop contínuo de visualização e ação até solicitar retorno ao menu principal.
4. **Validação de Conectividade 4.1.** Verificar grafo dos nós não isolados para garantir que todos estejam alcançáveis entre si. 4.2. Se a rede não estiver conectada, exibir mensagem de falha antes de finalizar a atividade.
5. **Finalização de Atividade 5.1.** Só pode ocorrer quando todos os nós marcados como "Falha" estiverem isolados. 5.2. Verificar conectividade Pós-Isolamento: rede deve permanecer conectada. 5.3. Calcular pontuação:
   * **+20 pontos** por nó com falha isolado corretamente.
   * **-10 pontos** por nó sem falha isolado (ação válida mas penalizada). 5.4. Determinar Rank:
   * **Ouro:** ≥ 60 pontos.
   * **Prata:** ≥ 30 e < 60.
   * **Bronze:** < 30. 5.5. Registrar no histórico:
   * ***Data e hora*** (formato: dd/MM/yyyy HH:mm:ss).
   * *Quant. falhas isoladas*.
   * *Tempo de execução* (segundos desde início).
   * *Pontuação e Rank*.
6. **Histórico de Atividades 6.1.** Persistir lista de registros (GameRecord) em arquivo history.json.  
   6.2. Ao iniciar o simulador, carregar histórico existente (se houver) e mantê-lo em memória.
7. 6.3. Permitir ao usuário limpar todo o histórico após confirmação.

**4. Requisitos Não Funcionais**

1. **Desempenho**
   * Ações de toggle e checagem de conectividade devem ser realizadas em < 100 ms.
2. **Usabilidade**
   * Interface de console deve usar cores e espaçamento legível.
   * Mensagens devem ser claras e feedback de sucesso/erro destacado**.**
3. **Manutenibilidade**
   * Código organizado em pastas (Models, Services, Utils, Program.cs).
   * Classes coesas com responsabilidade única (SRP).
   * Nomes de métodos e variáveis autoexplicativos.
4. **Portabilidade**
   * Compilar em Windows, macOS, Linux via .NET 5/6.
   * Não depender de bibliotecas proprietárias.
5. **Persistência e Integridade**
   * Usar System.Text.Json para serializar/deserializar histórico.
   * Validar esquemas JSON ao carregar (tratamento de exceção em caso de arquivo corrompido).
6. **Segurança Básica**
   * Autenticação simples (usuário fixo).
   * Histórico sensível não deve vazar dados extra além dos registros.

**5. Modelagem e Fluxogramas**

Texto

O conteúdo gerado por IA pode estar incorreto.

**5.2. Fluxograma de Operação Principal**

Texto

O conteúdo gerado por IA pode estar incorreto.

**6. Tecnologias e Ferramentas**

* **.NET 5/6 (C#)**: Linguagem principal para aplicação console.
* **System.Text.Json**: Serialização de objetos em JSON (histórico e incidentes).
* **Console Colorido** (System.Console): Para melhorar usabilidade e feedback.
* **Stopwatch** (System.Diagnostics): Cronometrar tempo de execução das atividades.
* **IDE:** Visual Studio 2022 / Visual Studio Code.
* **Controlador de Versão:** Git (repositório GitHub).

**7. Regras de Negócio Detalhadas**

1. **Login Restrito:** Usuário fixo ("user"/"1234"). Sem login válido, sem acesso.
2. \*\*Somente Nós com \*\***Failed == true** podem ser isolados.
3. **Isolar Nó sem Falha:** A ação é validada e bloqueada; gera mensagem de erro.
4. **Loop Interativo:** User permanece no fluxo de isolamento/restauração até digitar '0'.
5. **Finalização Condicional:** Todos nós falhos devem estar isolados **e** a rede conectada.
6. **Pontuação e Rank:** Calculados somente após finalização bem-sucedida.
7. **Histórico Imutável:** Cada finalização gera um registro; histórico apenas apagado pelo usuário.
8. **Persistência:** Histórico é gravado imediatamente após cada finalização.
9. **Validação Pós-Isolamento:** A rede deve permanecer conectada; se não, bloqueia finalização.