Guia para o Projeto Global Solution: ResilientRoute

Este guia oferece um direcionamento mais conciso para cada disciplina, ajudando você a focar nos pontos-chave para o sucesso do seu projeto "ResilientRoute".

📄 ADVANCED BUSINESS DEVELOPMENT WITH .NET

- Objetivo Principal: Construir a API REST (o backend) principal do ResilientRoute.
- Requisitos Chave do PDF:
    - API REST com boas práticas.
    - Persistência em Banco de Dados relacional.
    - Pelo menos um relacionamento 1:N.
    - Documentação com Swagger.
    - Aplicação de Razor e TagHelpers.
    - Uso correto da Migration.
- Foco Simplificado:

1. API: Defina endpoints claros para os recursos principais (ex: /incidentes, /abrigos, /alertas). Use verbos HTTP (GET, POST, PUT, DELETE) corretamente.

2. Banco de Dados (PostgreSQL): Use Entity Framework Core. Crie suas classes (Modelos) como Incidente, Usuario, Abrigo. Garanta um relacionamento 1:N (ex: um Usuario pode ter vários Incidentes reportados).

3. Swagger & Migrations: Configure o Swagger desde o início para documentar sua API. Use Migrations do EF Core para criar e atualizar seu banco de dados conforme seus modelos mudam.

4. Razor/TagHelpers: Se precisar de uma interface web simples (ex: um painel administrativo básico para visualizar dados), use Razor Pages ou MVC Views com TagHelpers.

☑️ COMPLIANCE, QUALITY ASSURANCE & TESTS

- Objetivo Principal: Planejar e documentar a arquitetura da solução ResilientRoute.
- Requisitos Chave do PDF:
    - Arquitetura da solução com TOGAF, usando a ferramenta ARCHI.
        - Visão de arquitetura (stakeholders, objetivos, requisitos).
        - Arquitetura de negócio (processos, funções, atores, papéis).

        - Arquitetura de sistemas (componentes de aplicação e dados, camadas).
        - Arquitetura de tecnologia (rede, devices, servidores, softwares).
    - Documento de apresentação do projeto (nome, equipe, desafio, público alvo, impacto com dados/estatísticas).
- Foco Simplificado:

1. TOGAF com ARCHI (Simplificado):
    - Visão: *Para quem?* (Cidadãos, Defesa Civil). *Qual problema?* (Ajudar em enchentes/deslizamentos). *O que precisa ter?* (Alertas, rotas, informações).
    - Negócio: *Principais ações do sistema?* (Alertar usuários, mostrar abrigos, permitir reporte de incidentes).
    - Sistemas: *Partes do software?* (Aplicativo móvel, API, Banco de Dados).
    - Tecnologia: *Tecnologias usadas?* (React Native, .NET, PostgreSQL, nuvem se houver).
    - Desenhe diagramas simples no Archi para cada um desses pontos.

2. Apresentação do Projeto: Descreva o ResilientRoute, sua equipe e o problema. Para o impacto, pesquise notícias ou dados da Defesa Civil local sobre enchentes/deslizamentos para justificar a importância da sua solução.

☁ DEVOPS TOOLS & CLOUD COMPUTING

- Objetivo Principal: "Empacotar" (conteinerizar) sua API .NET e o banco de dados usando Docker.
- Requisitos Chave do PDF:
    - Container para Aplicação (.NET API): via Dockerfile, usuário não-root, diretório de trabalho, variável de ambiente, porta exposta, CRUD completo.
    - Container do Banco de Dados (ex: PostgreSQL): imagem pública, volume para persistência, variável de ambiente, porta exposta.
    - Repositório no GitHub com todo o necessário para rodar.
    - Vídeo demonstrativo.
- Foco Simplificado:

1. API Dockerfile: Crie um arquivo Dockerfile para sua API .NET. Ele deve copiar o código, instalar dependências, definir como rodar a API e expor a porta correta. Use uma variável de ambiente para a string de conexão do banco de dados.

2. Banco de Dados Docker: Rode o PostgreSQL como outro contêiner. Configure um "volume" para que os dados não se percam se o contêiner parar.

3. Conexão e CRUD: Garanta que sua API no contêiner consiga se conectar ao banco de dados no outro contêiner e realize as operações de Criar, Ler, Atualizar e Deletar (CRUD).

4. GitHub e Vídeo: Suba tudo (código da API, Dockerfile, scripts de BD se houver) para o GitHub. Grave um vídeo mostrando o build dos containers e o CRUD funcionando.

👹 DISRUPTIVE ARCHITECTURES: IOT, IOB & GENERATIVE IA (Foco em Visão Computacional)

- Objetivo Principal: Usar Visão Computacional para analisar imagens e ajudar na avaliação de danos pós-desastre. (Adaptado do IoT para Visão Computacional, conforme sua necessidade).
- Requisitos Chave do PDF (Adaptados para Visão Computacional):
    - Sistema IoT (aqui, Sistema de Análise de Imagens) com dispositivos (aqui, algoritmos/modelos de CV que processam dados de imagens).
    - Hardware/Simulador (aqui, seu PC/ambiente de desenvolvimento Python; simulador Wokwi não se aplica diretamente, mas pode-se usar datasets como simulação).
    - Dashboard para monitorar/controlar (aqui, para visualizar imagens processadas e resultados).
    - Gateway (aqui, um script ou serviço que gerencia o fluxo de processamento de imagens).
    - Protocolos (aqui, HTTP para upload/download de imagens, JSON para resultados).
    - Entregáveis: Protótipo funcional, documentação no GitHub, vídeo.
- Foco Simplificado:

1. Dataset e Ferramenta:
    - Dataset: Procure por imagens de desastres (enchentes, deslizamentos). O site do "Disasters Charter" é uma boa fonte de inspiração para ver tipos de imagens de satélite. Para datasets práticos, pesquise por "xBD dataset" (danos em edificações) ou "Maxar Open Data Program".
    - Ferramenta: Use Python com a biblioteca OpenCV.

2. Análise Simples: Crie um script Python que:

    - Carregue uma imagem de uma área afetada.
    - Tente uma análise simples (ex: identificar grandes áreas com água usando limiarização de cores, ou se tiver imagens de antes/depois, tentar uma detecção de mudança básica).

3. Visualização e "Gateway":

    - Visualização: Salve a imagem processada com marcações ou imprima um resumo da análise. Um passo adiante: use Flask (Python) para criar uma página web simples onde você faz upload da imagem e vê o resultado.
    - "Gateway": Seu script Python ou a aplicação Flask já cumprem esse papel de forma simplificada.

4. GitHub e Vídeo: Coloque seu código e algumas imagens de exemplo no GitHub. Faça um vídeo mostrando o sistema analisando uma imagem e o resultado.

☕ JAVA ADVANCED

- Objetivo Principal: Desenvolver um microsserviço ou uma API complementar em Java com Spring Boot (ex: para análises mais específicas ou um módulo focado do ResilientRoute).
- Requisitos Chave do PDF:
    - API Rest com Spring Boot.
    - Persistência em Banco de Dados Relacional (Oracle, se for usar com a disciplina de Banco de Dados) usando Spring Data JPA.
    - Mapeamento de relacionamento entre entidades.
    - Validação com Bean Validation.
    - Paginação, ordenação e filtros.
    - Documentação da API com Swagger.
    - Autenticação com JWT.
    - Deploy em nuvem.
- Foco Simplificado:

1. API Focada: Crie uma API pequena com um propósito claro (ex: uma API que recebe dados de vários incidentes e calcula um "índice de risco" para diferentes bairros, ou um serviço para gerenciar voluntários).

2. Spring Boot & JPA: Use Spring Boot para criar a API e Spring Data JPA para interagir com o banco de dados (Oracle, se a disciplina de BD exigir para integração com Java, ou outro de sua escolha). Defina entidades e seus relacionamentos.

3. Funcionalidades Essenciais: Implemente endpoints para CRUD, adicione validação simples aos dados de entrada (@NotNull, @Size etc.) e documente com Swagger. Se tiver tempo, explore paginação para listas longas.

4. Segurança (Básico JWT): Se conseguir, implemente um endpoint protegido por JWT. Caso contrário, foque em ter a API funcional.

5. Deploy (Opcional Avançado): Se o deploy em nuvem for complexo, foque em ter a API rodando localmente de forma robusta. Se quiser tentar, Heroku ou AWS Elastic Beanstalk são opções.

💾 MASTERING RELATIONAL AND NON-RELATIONAL DATABASE

- Objetivo Principal: Projetar e implementar o banco de dados relacional que será usado por uma das suas APIs (a API Java, se estiver usando Oracle e integrando, ou a API .NET com PostgreSQL).
- Requisitos Chave do PDF:
    - Modelagem Relacional (3FN).
    - Criação das Tabelas com Restrições (PRIMARY KEY, FOREIGN KEY, etc.).
    - Procedures DML (INSERT, UPDATE, DELETE) por Tabela.
    - Pelo menos 2 Funções para retorno de dados processados.
    - 2 Blocos Anônimos com consultas complexas (JOIN, GROUP BY, etc.).
    - Cursores Explícitos.
    - Pelo menos 5 Consultas SQL Complexas (relatórios).
    - Integração com Projeto Java (se for o caso da API Java com Oracle).
- Foco Simplificado:

1. Modelagem e Tabelas: Desenhe o modelo do seu banco (quais tabelas, quais colunas, como se relacionam) para a API escolhida. Crie as tabelas usando SQL, definindo chaves primárias e estrangeiras.

2. Procedimentos e Consultas:
    - Crie procedures de INSERT e UPDATE para pelo menos uma tabela principal.
    - Escreva 2 ou 3 consultas SELECT mais elaboradas que busquem informações úteis (ex: "listar todos os incidentes de enchente reportados na última semana", "contar quantos abrigos têm vagas por bairro").

3. Funções (se der tempo): Crie uma função simples (ex: uma que calcula a ocupação percentual de um abrigo).

4. Blocos Anônimos e Cursores: Se o tempo for curto, foque em ter as tabelas e as consultas/procedures principais bem feitas. Estes são conceitos mais avançados de PL/SQL (Oracle) ou pgPL/SQL (PostgreSQL).

5. Integração: Garanta que sua API (.NET ou Java) consiga se conectar e usar esse banco de dados.

📱 MOBILE APPLICATION DEVELOPMENT

- Objetivo Principal: Desenvolver o aplicativo móvel ResilientRoute para os cidadãos, usando React Native.
- Requisitos Chave do PDF:
    - Mínimo de 5 telas e navegação entre elas (React Navigation ou Expo Router).
    - Implementar um CRUD utilizando a API criada na disciplina JAVA ou .NET (Axios ou Fetch).
    - Estilização do aplicativo (cores, fontes, imagens).
    - Arquitetura (organização de arquivos, nomes, componentes).
    - Vídeo de todas as funcionalidades do App.
- Foco Simplificado:

1. Navegação Básica: Crie um projeto React Native. Configure a navegação (React Navigation é popular) para ter umas 5 telas simples (ex: Tela Inicial, Mapa de Alertas, Reportar Incidente, Lista de Abrigos, Contatos Úteis).

2. Conexão com API (Leitura e Escrita):
    - Escolha uma tela para *mostrar* dados da sua API .NET (ex: listar alertas na Tela de Alertas). Use Axios ou Fetch para buscar os dados.
    - Escolha outra tela para *enviar* dados para sua API (ex: um formulário simples na tela "Reportar Incidente" que envia os dados para a API).

3. Estilo Básico: Aplique cores e fontes consistentes para dar uma identidade visual mínima ao app. Não precisa ser super complexo, mas que pareça um aplicativo do ResilientRoute.

4. Organização e Vídeo: Mantenha seus arquivos de código organizados. Grave um vídeo mostrando a navegação e as telas que interagem com a API.