**Projeto: Comunica & Ajuda!**

**Comunica & Ajuda!** é um sistema web desenvolvido para permitir que os cidadãos reportem facilmente problemas e compartilhem informações com órgãos públicos ou grupos comunitários relevantes, contribuindo para uma cidade melhor. Diferente do projeto anterior, este sistema será **aberto a todos**, permitindo um registro rápido e anônimo, sem a necessidade de contas de usuário ou logins.

Este aplicativo permitirá aos usuários:

* **Reportar um problema público:** Cidadãos poderão enviar rapidamente relatórios sobre diversos problemas em suas comunidades.
* **Visualizar relatórios existentes:** A transparência é fundamental! Usuários poderão ver o que outras pessoas reportaram.

O sistema também contará com uma interface administrativa simplificada para gerenciar os problemas reportados.

**Cenário: Manutenção de Espaços Públicos**

Imagine uma cidade onde os cidadãos podem reportar problemas relacionados a espaços públicos, como parques, ruas e praças. Isso poderia incluir coisas como:

* **Bancos danificados em um parque**
* **Grafites em muros públicos**
* **Lixeiras transbordando**
* **Postes de luz quebrados**
* **Buracos em calçadas**
* **Descarte irregular de lixo**

**Funcionalidades do Aplicativo**

O aplicativo web terá duas interfaces principais:

1. **Interface Pública de Relato:**
   * Permite que qualquer usuário envie um relatório sem fazer login.
   * **Campos para um relatório:**
     + **Título:** Um resumo conciso do problema (ex: "Banco Quebrado no Parque Central").
     + **Descrição:** Informações detalhadas sobre o problema.
     + **Urgência:** Uma escala de 1 a 5 (sendo 1 baixa urgência e 5 muito alta urgência).
     + **Categoria:** Selecione entre categorias predefinidas como "Manutenção de Parques", "Iluminação Pública", "Gestão de Resíduos", etc. (carregadas de um endpoint Spring REST).
     + **Localização:** Um campo de texto para o endereço ou, idealmente, uma simples integração de mapa (mas, para este exercício, um campo de texto para o endereço será suficiente para simplificar).
     + **Imagem:** Usuários podem fazer upload de uma única imagem para acompanhar o relatório.
   * **Visualizar Relatórios:** Usuários podem ver uma lista de todos os relatórios enviados, incluindo seu status (ex: "Recebido", "Em Progresso", "Resolvido").
2. **Interface do Administrador (Acesso Simplificado):**
   * Esta interface será acessível via uma URL específica e predefinida ou um link direto (sem uma página de login).
   * **Funcionalidades:**
     + **Gerenciar Categorias:** Operações CRUD (Criar, Ler, Atualizar, Deletar) para as categorias de problemas (ex: "Manutenção de Parques", "Iluminação Pública").
     + **Visualizar Todos os Relatórios:** Listar todos os relatórios enviados com seus detalhes.
     + **Atualizar Status do Relatório:** Mudar o status de um relatório (ex: de "Recebido" para "Em Progresso" ou "Resolvido").
     + **Excluir Relatório:** Remover relatórios inadequados ou resolvidos.
     + **Adicionar Notas Internas:** Um campo de texto para que os administradores adicionem notas privadas sobre um relatório (não visíveis para usuários públicos).

**Pilha Tecnológica**

* **Backend:** Java com Spring Boot (Spring REST)
* **Frontend:** HTML, CSS, JavaScript (Interatividade e comunicação com o backend via FetchAPI)
* **Banco de Dados:** PostgreSQL

**Desafio: Adicionar Dados de Localização (Texto ou Mapa Simples)**

Em vez de apenas uma imagem, vamos nos desafiar a incluir uma **localização** para o relatório. Para simplificar, inicialmente, este pode ser um campo de texto para um endereço. Se você se sentir mais ambicioso, pode explorar a integração de um display de mapa muito básico e somente de leitura (ex: usando OpenStreetMap) para a localização.

**Fases**

1. **Definição e Criação da Base de Dados, Entidades e Repositórios JPA:**
   * Entidade Report (id, titulo, descricao, urgencia, categoria\_id, localizacao, url\_imagem, status, notas\_admin, criado\_em)
   * Entidade Category (id, nome)
   * Repositórios JPA para Report e Category.
2. **Definição dos Serviços (APIs Rest do Spring):**
   * **Endpoints Públicos (sem necessidade de autenticação):**
     + GET /api/categorias: Listar todas as categorias de problemas disponíveis.
     + POST /api/relatos: Enviar um novo relatório (inclui upload de imagem).
     + GET /api/relatos: Listar todos os relatórios públicos (talvez com filtragem/paginação básica).
     + GET /api/relatos/{id}: Visualizar detalhes de um relatório específico.
   * **Endpoints do Administrador (acessíveis diretamente, sem login):**
     + GET /admin/api/categorias: Listar todas as categorias.
     + POST /admin/api/categorias: Criar uma nova categoria.
     + PUT /admin/api/categorias/{id}: Atualizar uma categoria existente.
     + DELETE /admin/api/categorias/{id}: Excluir uma categoria.
     + GET /admin/api/relatos: Listar todos os relatórios (incluindo notas do administrador).
     + PUT /admin/api/relatos/{id}/status: Atualizar o status de um relatório.
     + PUT /admin/api/relatos/{id}/notas: Adicionar/atualizar notas internas do administrador para um relatório.
     + DELETE /admin/api/relatos/{id}: Excluir um relatório.
3. **Frontend - Interface Pública de Relato:**
   * **Página Inicial (**index.html**):**
     + Um formulário simples para enviar um novo relatório.
     + Exibir uma lista de relatórios recentes.
     + Um botão "Ver Todos os Relatórios".
   * **Formulário de Envio de Relatório:**
     + Campos de entrada para título, descrição, urgência (dropdown), categoria (dropdown, populado pela API), localização e entrada de arquivo para imagem.
     + Validação no lado do cliente.
     + Chamada AJAX para POST /api/relatos.
   * **Página de Lista de Relatórios:**
     + Tabela ou cards exibindo os relatórios públicos.
     + Clicar em um relatório mostra seus detalhes.
4. **Frontend - Interface do Administrador:**
   * **Dashboard do Administrador (**admin.html**):**
     + Seções para "Gerenciar Categorias" e "Gerenciar Relatórios".
   * **Gerenciar Categorias:**
     + Formulário para adicionar uma nova categoria.
     + Tabela listando categorias existentes com botões "Editar" e "Excluir".
     + Modais ou formulários inline para edição de categorias.
   * **Gerenciar Relatórios:**
     + Tabela listando todos os relatórios com detalhes (incluindo status e notas do administrador).
     + Opções para atualizar o status (dropdown), adicionar/editar notas do administrador (textarea) e excluir.

**Formato de Troca de Dados**

* **JSON** será usado para toda a troca de informações entre o frontend e as APIs do backend.

**Próximos Passos**

Para começar, você pode iniciar com a Fase 1: configuração do banco de dados e definição de suas entidades Spring Boot e repositórios JPA. Assim que estiver sólido, você pode passar para a implementação de suas APIs REST do Spring para as funcionalidades públicas e administrativas.

O que você achou dessa ideia de projeto revisada? Há alguma parte específica que você gostaria de explorar primeiro, como o esquema do banco de dados, ou talvez a configuração do seu projeto Spring Boot?