ETAPA 3 · TAREFA:

Enviar até o dia 11/11/2024: Um arquivo de texto em formato PDF contendo as respostas de cada um dos passos (descrições e imagens). (Incluir no arquivo o nome do grupo, os nomes dos integrantes e suas respectivas matrículas).
Localização do link de envio no AVA: "ENVIAR | Registros de Evidências do Levantamento de Necessidades". (Basta que um único integrante do grupo faça o envio).

Observações:

1. **Não será permitida a adoção de aplicativos/ferramentas diferentes das descritas aqui**, salvo em situações extremamente necessárias consentidas pelo professor.

2. **Esse guia é apenas uma referência**. Cabe ao aluno/grupo fazer as pesquisas complementares para solucionar eventuais problemas naturais, ou adequações ao funcionamento esperado.

Guia Didático para a Construção da Single Page Application (SPA)

Visão Geral

Este guia focará na construção do *backend* da SPA usando Flask, MongoDB e Docker. O *frontend* React foi configurado para interagir com o *backend* e enviar/receber dados.

Cada grupo de alunos irá adaptar o conteúdo do *frontend* para os temas propostos (Micro e Pequenos Empreendimentos, Comunidades carentes, Escolas Públicas).

Obs.: Convém fazer o backup dos arquivos construídos até o momento, para restaurá-los caso seja necessário.

Passo 1: Funcionalidades Fundamentais

As funcionalidades que serão implementadas no *backend* são:

1. Listar Projetos (GET /projects)
2. Cadastrar Novo Projeto (POST /projects)
3. Visualizar Detalhes de um Projeto (GET /projects/:id)

1.1. Preparar o *Backend* (Flask)

No *backend*, vamos adicionar essas rotas usando o framework Flask e conectá-las ao MongoDB para que os dados sejam persistidos.

1. Criar o arquivo `app.py` no diretório `backend/`:

```
from flask import Flask, jsonify, request
from pymongo import MongoClient
from bson.objectid import ObjectId
from flask_cors import CORS

app = Flask(__name__)
CORS(app)   # Habilita o CORS para permitir requisições do frontend

client = MongoClient('mongodb://mongo:27017/')
db = client.ongdb   # Conecta ao banco de dados chamado "ongdb"

# Rota para listar todos os projetos
@app.route('/projects', methods=['GET'])
def get_projects():
    projects = list(db.projects.find({}))   # Recupera todos os
projetos do MongoDB
    for project in projects:
        project['_id'] = str(project['_id'])   # Converte ObjectId
para string
    return jsonify(projects), 200

# Rota para adicionar um novo projeto
@app.route('/projects', methods=['POST'])
def add_project():
    new_project = request.get_json()
    result = db.projects.insert_one(new_project)   # Insere o novo
projeto no MongoDB
    return jsonify(str(result.inserted_id)), 201

# Rota para visualizar detalhes de um projeto específico
@app.route('/projects/<project_id>', methods=['GET'])
def get_project_details(project_id):
    project = db.projects.find_one({"_id": ObjectId(project_id)})
    if project:
        project['_id'] = str(project['_id'])
        return jsonify(project), 200
    return jsonify({"error": "Projeto não encontrado"}), 404

if __name__ == '__main__':
```

```
    app.run(host='0.0.0.0', port=5000, debug=True)
```

2. Configurar Dependências no arquivo `backend/requirements.txt`:

```
Flask==2.3.3
Flask-RESTful==0.3.9
pymongo==4.1.1
Flask-CORS==3.0.10
```

3. Executar o *backend* com Docker Compose:

Certifique-se de que o `docker-compose.yml` está configurado para rodar o MongoDB e o Flask.

Passo 2: Funcionalidades Complementares do *Backend*

2.1. Deletar Projeto:

Implementar a rota DELETE no *backend* para remover um projeto do MongoDB:

```
# Rota para remover um projeto do MongoDB
@app.route('/projects/<project_id>', methods=['DELETE'])
def delete_project(project_id):
    result = db.projects.delete_one({"_id": ObjectId(project_id)})
    if result.deleted_count > 0:
        return jsonify({"message": "Projeto deletado com sucesso"}),
200
    return jsonify({"error": "Projeto não encontrado"}), 404
```

2.2. Atualizar Projeto:

Implementar a rota PUT para atualizar um projeto:

```
# Rota para atualizar um projeto
@app.route('/projects/<project_id>', methods=['PUT'])
def update_project(project_id):
    updated_data = request.get_json()
    result = db.projects.update_one({"_id": ObjectId(project_id)},
{"$set": updated_data})
    if result.modified_count > 0:
        return jsonify({"message": "Projeto atualizado com
sucesso"}), 200
    return jsonify({"error": "Projeto não encontrado"}), 404
```

Passo 3: Preparar o *Frontend* (React)

O *frontend* servirá como interface simples para enviar e receber dados do *backend*. Vamos usar o React para criar um formulário de cadastro e exibir a lista de projetos.

3.1. Criar Componentes no *Frontend*

1. Editar o `src/App.js` para integrar o *frontend* com o *backend*:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

function App() {
  const [projects, setProjects] = useState([]);
  const [newProject, setNewProject] = useState({ name: '',
description: '' });
  const [editingProject, setEditingProject] = useState(null);

  // Função para buscar projetos do backend
  useEffect(() => {
    axios.get('http://localhost:5000/projects')
      .then(response => setProjects(response.data))
      .catch(error => console.error('Erro ao buscar projetos:',
error));
  }, []);

  // Função para lidar com mudanças nos campos de formulário
  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setNewProject(prevState => ({ ...prevState, [name]: value }));
  };

  // Função para adicionar um novo projeto
  const handleAddProject = () => {
    axios.post('http://localhost:5000/projects', newProject)
      .then(response => {
        setProjects([...projects, { ...newProject, _id:
response.data }]);
        setNewProject({ name: '', description: '' });
        scrollToSection('projectList');  // Rola para a lista de
projetos
      })
      .catch(error => console.error('Erro ao adicionar projeto:',
error));
  };

  // Função para deletar um projeto
  const handleDeleteProject = (projectId) => {
    axios.delete(`http://localhost:5000/projects/${projectId}`)
      .then(response => {
```

```
        setProjects(projects.filter(project => project._id !==
projectId));
      })
      .catch(error => console.error('Erro ao deletar projeto:',
error));
  };

  // Função para iniciar a edição de um projeto
  const handleEditProject = (project) => {
    setEditingProject(project._id);
    setNewProject({ name: project.name, description:
project.description });
    scrollToSection(`updateProject_${project._id}`);  // Rola para o
formulário de atualização
  };

  // Função para atualizar um projeto existente
  const handleUpdateProject = () => {
    axios.put(`http://localhost:5000/projects/${editingProject}`,
newProject)
      .then(response => {
        setProjects(projects.map(project => (
          project._id === editingProject ? { ...project,
...newProject } : project
        )));
        setEditingProject(null);
        setNewProject({ name: '', description: '' });
      })
      .catch(error => console.error('Erro ao atualizar projeto:',
error));
  };

  // Função para rolar a tela para uma seção específica
  const scrollToSection = (sectionId) => {
    const element = document.getElementById(sectionId);
    if (element) {
      element.scrollIntoView({ behavior: 'smooth' });
    }
  };

  return (
    <div>
      {/* Menu superior */}
      <nav style={{ backgroundColor: '#f5f5f5', padding: '10px',
marginBottom: '20px' }}>
        <button onClick={() => scrollToSection('projectList')}>Lista
de Projetos</button>
        <button onClick={() =>
scrollToSection('addProject')}>Adicionar Projeto</button>
      </nav>

      {/* Seção de Lista de Projetos */}
      <section id="projectList">
        <h2>Lista de Projetos da ONG</h2>
        {projects.length === 0 ? (
          <p>Nenhum projeto foi adicionado ainda.</p>
        ) : (
          <ul>
            {projects.map((project, index) => (
              <li key={index}>
```

```
                <strong>{index + 1}. Nome:</strong> {project.name}
<br />
                <strong>Descrição:</strong> {project.description}
                <br />
                <button onClick={() =>
handleDeleteProject(project._id)}>Deletar</button>
                <button onClick={() =>
handleEditProject(project)}>Editar</button>
                {editingProject === project._id && (
                  <section id={`updateProject_${project._id}`}
style={{ marginTop: '20px' }}>
                    <h3>Atualizar Projeto</h3>
                    <input
                      type="text"
                      name="name"
                      value={newProject.name}
                      onChange={handleInputChange}
                      placeholder="Nome do Projeto"
                    />
                    <br />
                    <input
                      type="text"
                      name="description"
                      value={newProject.description}
                      onChange={handleInputChange}
                      placeholder="Descrição do Projeto"
                    />
                    <br />
                    <button onClick={handleUpdateProject}>Salvar
Alterações</button>
                  </section>
                )}
              </li>
            ))}
          </ul>
        )}
      </section>

      {/* Seção para Adicionar Novo Projeto */}
      <section id="addProject" style={{ marginTop: '50px' }}>
        <h2>Adicionar Novo Projeto</h2>
        <input
          type="text"
          name="name"
          value={newProject.name}
          onChange={handleInputChange}
          placeholder="Nome do Projeto"
        />
        <br />
        <input
          type="text"
          name="description"
          value={newProject.description}
          onChange={handleInputChange}
          placeholder="Descrição do Projeto"
        />
        <br />
        <button onClick={handleAddProject}>Adicionar
Projeto</button>
      </section>
    </div>
```

```
  );
}

export default App;
```

3.2. Instalar Axios

No diretório do *frontend*, certifique-se de que o Axios está instalado com:

```
npm install axios
```

Passo 4: Customização do *backend* conforme as Regras de Negócio e Interface definidas

NESSE PASSO, CADA GRUPO OU ALUNO IRÁ REALIZAR OS ESTUDOS E AS PESQUISAS NECESSÁRIOS PARA DESENVOLVER **O *BACKEND***.

NO *FRONTEND*, SERÃO AVALIADOS A ADERÊNCIA À ORGANIZAÇÃO DA INTERFACE E REGRAS DE NEGÓCIO DEFINIDOS NA ETAPA 2. DEMAIS ASPECTOS VISUAIS OU ESTÉTICOS NÃO SERÃO AVALIADOS, EMBORA POSSAM SER ÚTEIS PARA ENTENDER O CONTEXTO PROPOSTO.

Obs.: Convém fazer o backup dos arquivos construídos até o momento, para restaurá-los caso seja necessário.

Adicione uma funcionalidade exclusiva ou comportamento dinâmico à sua SPA, que deve ser ajustado de acordo com o tema escolhido. A escolha da funcionalidade deve estar alinhada ao tema e às regras de negócio específicas definidas na Etapa 2.

Adaptar o estilo visual da sua SPA para refletir o tema escolhido (MPEs, Comunidades carentes, ou Escolas Públicas), usando HTML + CSS. Utilize o mockup como guia para organizar os componentes do *frontend*.

Seguem algumas sugestões de funcionalidades e comportamentos, mas o grupo poderá escolher outra implementação diferente das relacionadas aqui.

Sugestão 1. Filtros Dinâmicos e Ordenação na Lista de Projetos

- Implemente um sistema de filtros dinâmicos para a Lista de Projetos. O usuário poderá filtrar os projetos por categorias específicas ou palavras-chave, de acordo com o tema do projeto.

- Adicione a opção de ordenar os projetos por diferentes critérios, como nome, data de criação, ou relevância (conforme definido pelas regras de negócio).

- Filtros e ordenação podem ser adicionados na interface com seletores (<select> ou <input>), e a lista de projetos deve ser atualizada dinamicamente com base nos critérios selecionados.

Sugestão 2. Integração com APIs Externas

- Integrar sua aplicação com uma API externa para obter dados adicionais que complementem as funcionalidades do projeto. Esses dados devem ser exibidos na interface e agregados às informações da Lista de Projetos.

- Exemplo de API Externa: Google Maps API, OpenWeather API, Wikipedia API, entre outras APIs gratuitas que possam fornecer dados complementares.

Sugestão 3. Funcionalidade de Notificações Dinâmicas

- Adicione uma funcionalidade de notificações no *frontend* que informe o usuário sobre eventos ou atualizações na aplicação.

- As notificações podem ser visuais (ex.: banners ou pop-ups) ou enviadas via e-mail (com integração de *back-end*).

- As notificações podem ser implementadas com bibliotecas de *frontend* para alertas (ex.: react-toastify).

- Para notificações por e-mail, você pode integrar com serviços de envio de e-mail, como SendGrid ou Mailgun.

Sugestão 4. Campos Personalizados no Formulário de Cadastro de Projetos

- Adapte os campos do Formulário de Cadastro de Projetos para refletir informações específicas e exclusivas do tema. Por exemplo, incluir 3 novos campos no formulário, de acordo com o tema escolhido.

- O aluno deve adaptar o formulário *frontend* e ajustar o *backend* para lidar com esses novos campos.

Sugestão 5. Implementação de Gráficos Dinâmicos

- Adicione uma seção à SPA que exiba gráficos dinâmicos com base nos dados dos projetos cadastrados. Isso permitirá que o usuário visualize métricas e estatísticas de maneira visual.

- Utilize bibliotecas como Chart.js ou Recharts para integrar gráficos no *frontend*.

- As métricas exibidas nos gráficos devem ser calculadas com base nas regras de negócio definidas.

Dica 1: Quando estiver desenvolvendo, use apenas docker-compose up (sem --build), pois isso já observará as mudanças no código. Somente use --build se houver mudanças no Dockerfile ou nas dependências instaladas (ex.: requirements.txt ou package.json)

Dica 2: É possível verificar as mudanças em tempo real sempre que o código do frontend ou backend é alterado. Para isso, você pode configurar seu ambiente Docker para utilizar o Hot Reloading tanto no backend quanto no frontend.