

# Tutorial de criação do sistema de Doação de Sangue

Este tutorial irá instruir de forma clara e objetiva como criar uma aplicação simples de doação de sangue.

Nós já criamos uma instância de banco de dados MySQL na AWS então não se preocupe em criar um banco de dados no Docker localmente, mas o tutorial de criar a instância no Docker estará disponível caso queiram testar. (Caso forem criar um banco de dados local usar o MySQL pois a ORM utilizada no backend é uma ORM de MySQL)

Observações: Devido a bloqueios da rede do IFTM pode ser que o banco de dados criado na AWS não funcione, nesse caso é necessário usar uma rede externa (Roteada de um celular por exemplo). Se não for possível, deve ser criado um banco de dados local.

Os códigos de backend e frontend estarão disponíveis publicamente no Github caso queiram conferir algo : Backend -> <https://github.com/FelipeGuglielmeli/Doacao-sangue>  
Frontend -> <https://github.com/FelipeGuglielmeli/DoacaoFront>

## Etapa 1 (Opcional) : Criação do banco de dados.

### 1. Instalando Docker

Abrir o console do Linux e digitar os seguintes comandos.

- Atualizar o sistema:

```
sudo apt update  
sudo apt upgrade -y
```

- Instalar dependências:

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common -y
```

- Adicionar a chave GPG do Docker:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

- Adicionar o repositório Docker:

```
sudo add-apt-repository "deb [arch=amd64]  
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

- Instalar Docker:

```
sudo apt update  
sudo apt install docker-ce -y
```

- Verificar a instalação do Docker:

```
sudo systemctl status docker
```

- Criando um Contêiner MySQL  
Puxar a imagem oficial do MySQL:

```
docker pull mysql:latest
```

- Executar o contêiner MySQL:

```
docker run --name mysql-container -e MYSQL_ROOT_PASSWORD=your_password  
-e MYSQL_DATABASE=mydb -e MYSQL_USER=myuser -e  
MYSQL_PASSWORD=mypassword -p 3306:3306 -d mysql:latest
```

**Nota:**

Substitua *your\_password* pela senha que você deseja para o usuário root.

*mydb* é o nome do banco de dados que será criado.

*myuser* é o nome do usuário que será criado.

*mypassword* é a senha para *myuser*.

- Verificar se o contêiner está rodando:

```
docker ps
```

- Conectar ao banco de dados MySQL:  
Você pode usar o cliente MySQL que vem com a instalação do MySQL ou o mysql-client. Para instalar o mysql-client no Linux, use:

```
sudo apt install mysql-client -y
```

- Conectar ao MySQL no contêiner:

```
mysql -h 127.0.0.1 -P 3306 -u myuser -p
```

Digite a senha mypassword quando solicitado.

- Verificar a criação do banco de dados e do usuário:

```
SHOW DATABASES;  
USE mydb;  
SHOW TABLES;
```

## **Etapa 2 : Criação do backend.**

O backend será criado em Python, usando o Framework Flask. E a ORM será o sqlalchemy.

Instalar o python: abra o terminal para executar os seguintes comandos.

A maioria das distribuições Linux já vem com Python instalado. Verifique a instalação:

```
python3 --version
```

Se Python não estiver instalado, instale-o usando o gerenciador de pacotes da sua distribuição. Para distribuições baseadas em Debian (como Ubuntu), use:

```
sudo apt update  
sudo apt install python3 python3-venv python3-pip
```

Após instalar o python, crie o diretório onde ficará seu backend e abra-o no Vscode. Abra o terminal do Vscode e execute:

```
python3 -m venv venv
```

Esse comando criará o ambiente virtual que será executada sua aplicação. Ative o ambiente virtual com o seguinte comando:

```
source venv/bin/activate
```

Com o ambiente virtual ativado, instale Flask usando pip:

```
pip install flask
```

Crie um arquivo 'requirements.txt' para listar todas as dependências:

```
echo "Flask" > requirements.txt
```

## **Criação dos primeiros arquivos de configuração.**

Para criar os primeiros arquivos de configuração devemos antes instalar as dependências necessárias, execute o seguinte comando no terminal:

```
pip install alembic==1.13.1 blinker==1.8.2 click==8.1.7 colorama==0.4.6  
Flask==3.0.3 Flask-Migrate==4.0.7 Flask-SQLAlchemy==3.1.1 greenlet==3.0.3  
itsdangerous==2.2.0 Jinja2==3.1.4 Mako==1.3.5 MarkupSafe==2.1.5 PyMySQL==1.1.1
```

*python-dotenv==1.0.1 SQLAlchemy==2.0.30 typing\_extensions==4.12.2 Werkzeug==3.0.3*

Obs.: Alguma dessas dependências podem ter erro na instalação mas não tem problema. Caso o sistema não rode e indique que está faltando uma das dependências basta dar um novo comando pip install na dependencia necessária.

Após instalar as depêndencias, crie o arquivo `__init__.py` na pasta app do projeto (Caso o comando do flask ainda não tenha criado). O arquivo `__init__.py` deve conter esse código:

```
1 from flask import Flask
2 from flask_sqlalchemy import SQLAlchemy
3 from flask_migrate import Migrate
4 from flask_cors import CORS # Importar CORS
5 from .config import Config # Certifique-se de usar o caminho relativo
6
7 db = SQLAlchemy()
8 migrate = Migrate()
9
10 def create_app():
11     app = Flask(__name__)
12     app.config.from_object(Config)
13
14     db.init_app(app)
15     migrate.init_app(app, db)
16
17     CORS(app) # Configurar CORS no aplicativo Flask
18
19     from .routes import main as main_blueprint
20     app.register_blueprint(main_blueprint)
21
22     return app
23
```

Depois crie o arquivo `config.py` nas pasta app também, que deve conter esse código:

```
app > config.py > Config
You, last week | 1 author (You)
1 import os
2 from dotenv import load_dotenv
3
4 load_dotenv()
5
6 # Debugging: Verificar se as variáveis de ambiente estão carregadas corretamente
7 print(f"DB_NAME: {os.environ.get('DB_NAME')}")
8 print(f"DB_USER: {os.environ.get('DB_USER')}")
9 print(f"DB_PASSWORD: {os.environ.get('DB_PASSWORD')}")
10 print(f"DB_HOST: {os.environ.get('DB_HOST')}")
11 print(f"DB_PORT: {os.environ.get('DB_PORT')}")
12
13 You, last week | 1 author (You)
14 class Config:
15     SECRET_KEY = os.environ.get('SECRET_KEY') or 'you-will-never-guess'
16     SQLALCHEMY_DATABASE_URI = f'mysql+pymysql://{os.environ.get("DB_USER")}:
17                                     {os.environ.get("DB_PASSWORD")}@{os.environ.get("DB_HOST")}:
18                                     {os.environ.get("DB_PORT")}/{os.environ.get("DB_NAME")}'
19     SQLALCHEMY_TRACK_MODIFICATIONS = False
20
```

Para que a conexão do banco de dados funcione precisamos criar um arquivo .env que contenha as credenciais do banco de dados, **na raiz do projeto** crie um arquivo chamado .env que deve conter o seguinte código:

```
1 DB_NAME=database-1
2 DB_USER=admin
3 DB_PASSWORD=senha123
4 DB_HOST=database-1.clqemya4qt6m.sa-east-1.rds.amazonaws.com
5 DB_PORT=3306
6
```

Obs.: Caso você tenha criado seu próprio banco de dados localmente você deve inserir as credenciais do banco que você criou, caso contrário o backend vai continuar conectado no banco de dados que está na AWS.

Após ter criado os arquivos de configuração da aplicação, devemos criar o arquivo de models que será usado no backend:

```
app > models.py > ...
You, 7 days ago | 1 author (You)
1 from . import db
2
You, 7 days ago | 1 author (You)
3 class Doador(db.Model):
4     __tablename__ = 'Doadores'
5
6     id = db.Column(db.Integer, primary_key=True)
7     nome = db.Column(db.String(150), nullable=False)
8     cpf = db.Column(db.String(14), nullable=False, unique=True)
9     contato = db.Column(db.String(15), nullable=False)
10    tipo_sanguineo = db.Column(db.Enum('A', 'B', 'AB', 'O'), nullable=False)
11    rh = db.Column(db.Enum('positivo', 'negativo'), nullable=False)
12    tipo_e_rh_corretos = db.Column(db.Boolean, nullable=False)
13    situacao = db.Column(db.Enum('ativo', 'inativo'), nullable=False)
14
15    doacoes = db.relationship('Doacao', backref='doador', lazy=True)
16
You, 7 days ago | 1 author (You)
17 class Doacao(db.Model):
18     __tablename__ = 'Doacoes'
19
20     id = db.Column(db.Integer, primary_key=True)
21     data = db.Column(db.Date, nullable=False)
22     hora = db.Column(db.Time, nullable=False)
23     volume = db.Column(db.Float, nullable=False)
24     id_doador = db.Column(db.Integer, db.ForeignKey('Doadores.id'), nullable=False)
25     situacao = db.Column(db.Enum('ativo', 'inativo'), nullable=False)
```

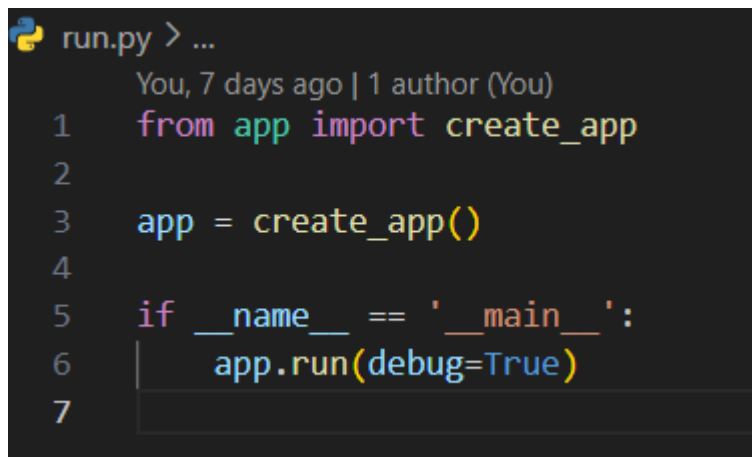
Os.: Se estiverem usando o banco de dados que foi criado localmente é necessário rodar os comandos de migração, para criar as tabelas no banco de dados:

*flask db init* (inicia o banco de dados)

*flask db migrate -m "Descrição da migração"* (Cria o arquivo de migração)

*flask db upgrade* (Faz o upgrade do banco de dados com as tabelas)

Agora, antes de criar o arquivo com as rotas (endpoints) do backend devemos criar o arquivo que roda o servidor. Crie o arquivo `run.py` na **RAIZ** do projeto:

A screenshot of a code editor with a dark background. At the top left, there is a Python logo and the text 'run.py > ...'. Below this, the code is as follows:

```
You, 7 days ago | 1 author (You)
1  from app import create_app
2
3  app = create_app()
4
5  if __name__ == '__main__':
6      app.run(debug=True)
7
```

Para iniciar o servidor, antes devemos incluir o script de run na variável de ambiente localmente, digite no terminal:

*\$env:FLASK\_APP="run.py"*

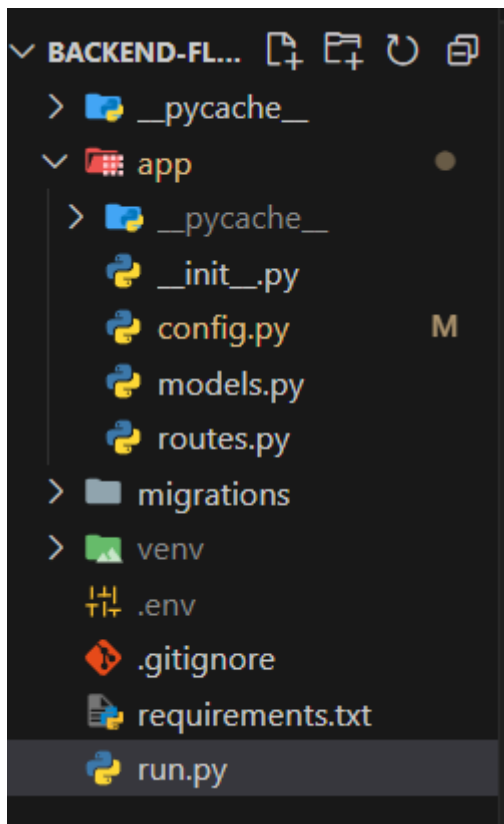
Após inserir o script de run na variável de ambiente basta rodar o script e testar se o servidor está rodando agora:

*flask run*

ou

*python run.py*

Se o servidor rodou com sucesso podemos seguir para a criação dos endpoints da API, até o momento a estrutura do projeto deve estar parecida com isso:



Agora podemos criar o arquivo de rotas e ir testando no PostMan ou Insomnia (softwares que testam requisições HTTP) - Opcional.

Na pasta app do projeto crie um arquivo chamado routes.py, nesse arquivo ficarão todas as rotas necessárias:

```
pp > routes.py > get_doadores
You, 7 days ago | 1 author (You)
1 from flask import Blueprint, jsonify, request
2 from .models import Doador, Doacao
3 from . import db
4
5 main = Blueprint('main', __name__)
6
7 @main.route('/doadores', methods=['GET'])
8 def get_doadores():
9     doadores = Doador.query.all()
10    doadores_list = [
11        {'id': doador.id, 'nome': doador.nome}
12        for doador in doadores
13    ]
14    return jsonify(doadores_list)
15
```

Essa primeira rota será um GET que irá retornar todos os Doadores da base de dados, o nosso banco de dados que está na AWS já está populado com dados fictícios, então essa rota já irá retornar todos os doadores do banco de dados. Ela pode ser testada no postman com a url <http://127.0.0.1:5000/doadores> (por padrão o Flask inicia o servidor na porta 5000, mas sempre verifique em qual porta o servidor está rodando). E o resultado desse GET será algo parecido com isso:

```
{
  "id": 1,
  "nome": "João Silveira"
},
{
  "id": 2,
  "nome": "Maria Oliveira"
},
{
  "id": 3,
  "nome": "Carlos Souza"
},
{
  "id": 4,
  "nome": "Ana Júlia"
},
{
  "id": 5,
  "nome": "Pedro Almeida"
},
.....
```

Agora implemente as demais rotas necessárias para o CRUD do sistema:



Rota de lista doações:

```
@main.route('/doacoes', methods=['GET'])
def get_doacoes():
    nome_doador = request.args.get('nome')
    data_inicio = request.args.get('data_inicio')
    data_fim = request.args.get('data_fim')

    query = Doacao.query.join(Doador).with_entities(
        Doacao.volume, Doador.nome, Doacao.data, Doacao.hora
    )

    if nome_doador:
        query = query.filter(Doador.nome.ilike(f'%{nome_doador}%'))

    if data_inicio:
        query = query.filter(Doacao.data >= data_inicio)

    if data_fim:
        query = query.filter(Doacao.data <= data_fim)

    doacoes = query.all()
    doacoes_list = [
        {
            'volume': doacao.volume,
            'doador': doacao.nome,
            'data': doacao.data.strftime('%Y-%m-%d'),
            'hora': doacao.hora.strftime('%H:%M:%S')
        }
        for doacao in doacoes
    ]
    return jsonify(doacoes_list)
```

```
@main.route('/doadores', methods=['POST'])
def create_doador():
    data = request.get_json()
    novo_doador = Doador(
        nome=data['nome'],
        cpf=data['cpf'],
        contato=data['contato'],
        tipo_sanguineo=data['tipo_sanguineo'],
        rh=data['rh'],
        tipo_e_rh_corretos=data['tipo_e_rh_corretos'],
        situacao=data['situacao']
    )
    db.session.add(novo_doador)
    db.session.commit()
    return jsonify({'message': 'Doador created successfully'}), 201

@main.route('/doacoes', methods=['POST'])
def create_doacao():
    data = request.get_json()
    nova_doacao = Doacao(
        data=data['data'],
        hora=data['hora'],
        volume=data['volume'],
        id_doador=data['id_doador'],
        situacao=data['situacao']
    )
    db.session.add(nova_doacao)
    db.session.commit()
    return jsonify({'message': 'Doacao created successfully'}), 201
```

Rota de criar Doadores e  
Doações:

Outra rota de criar doação mais detalhada:

```
# Nova rota para doar sangue
@main.route('/doar-sangue', methods=['POST'])
def doar_sangue():
    data = request.get_json()
    doador_id = data['doador_id']
    volume = data['volume']
    data_doacao = data['data']
    hora_doacao = data['hora']
    tipo_sanguineo = data['tipo_sanguineo']
    rh = data['rh']

    doador = Doador.query.get(doador_id)
    if not doador:
        return jsonify({'message': 'Doador não encontrado'}), 404

    doador.tipo_sanguineo = tipo_sanguineo
    doador.rh = rh
    doador.tipo_e_rh_corretos = True

    nova_doacao = Doacao(
        data=data_doacao,
        hora=hora_doacao,
        volume=volume,
        id_doador=doador_id,
        situacao='ativo'
    )

    db.session.add(nova_doacao)
    db.session.commit()

    return jsonify({'message': 'Doação registrada com sucesso'}), 201
```

A rota para ver as doações de um doador específico:

```
# Nova rota para ver doações de um doador específico
@main.route('/doacoes/doador/<int:doador_id>', methods=['GET'])
def ver_doacoes_doador(doador_id):
    doador = Doador.query.get(doador_id)
    if not doador:
        return jsonify({'message': 'Doador não encontrado'}), 404

    doacoes = Doacao.query.filter_by(id_doador=doador_id).all()
    doacoes_list = [
        {
            'data': doacao.data.strftime('%Y-%m-%d'),
            'hora': doacao.hora.strftime('%H:%M:%S'),
            'volume': doacao.volume,
            'situacao': doacao.situacao
        }
        for doacao in doacoes
    ]

    return jsonify(doacoes_list), 200
```

E uma rota para verificar as doações feitas dentro de um período específico:

```
# Nova rota para ver doações dentro de um período específico
@main.route('/doacoes/periodo', methods=['GET'])
def ver_doacoes_periodo():
    data_inicio = request.args.get('data_inicio')
    data_fim = request.args.get('data_fim')

    query = Doacao.query
    if data_inicio:
        query = query.filter(Doacao.data >= data_inicio)
    if data_fim:
        query = query.filter(Doacao.data <= data_fim)

    doacoes = query.all()
    doacoes_list = [
        {
            'data': doacao.data.strftime('%Y-%m-%d'),
            'hora': doacao.hora.strftime('%H:%M:%S'),
            'volume': doacao.volume,
            'situacao': doacao.situacao,
            'doador': doacao.doador.nome
        }
        for doacao in doacoes
    ]

    return jsonify(doacoes_list), 200
```

Testadas todas as rotas do backend está finalizado a parte do backend, agora podemos criar o frontend.

### Etapa 3 : Criação do frontend.

Para criar a aplicação frontend primeiro instale o node no Linux

Abra o terminal e execute os seguintes comandos para instalar Node.js e npm:

```
sudo apt update
sudo apt install nodejs npm -y
```

Verifique a instalação do Node.js e npm:

```
node --version
npm --version
```

Instalar o npx (se não estiver instalado)

O npx é usado para executar comandos npm sem precisar instalar o pacote globalmente.

Verifique se o npx está instalado:

```
npx --version
```

Se o npx não estiver instalado, atualize o npm, pois ele vem incluído nas versões mais recentes do npm:

```
sudo npm install -g npm
```

Agora crie o diretório em que ficará seu projeto frontend e abra-o no Vscode. Após abrir o Vscode crie o projeto no terminal com o comando:

```
npx create-react-app my-react-app
```

Navegue até o diretório do projeto criado:

```
cd my-react-app
```

Agora instale todas as dependências necessárias para o projeto com o comando:

```
npm install @babel/plugin-proposal-private-property-in-object@^7.21.11  
@testing-library/jest-dom@^5.17.0 @testing-library/react@^13.4.0  
@testing-library/user-event@^13.5.0 axios@^1.7.2 react@^18.3.1 react-dom@^18.3.1  
react-router-dom@^6.23.1 react-scripts@5.0.1 web-vitals@^2.1.4 autoprefixer@^10.4.19  
postcss@^8.4.38 tailwindcss@^3.4.4
```

Muitas dessas dependências já vem com o React então elas serão ignoradas, isso não tem problema.

A estrutura inicial do React já vem criada. Então vamos nos preocupar em criar os arquivos de configuração que precisamos para iniciar o projeto:

Lembre-se que em caso de dúvidas você pode acessar o código através do link do GitHub:  
<https://github.com/FelipeGuglielmeli/DoacaoFront/tree/master>

Agora crie os arquivos iniciais (Caso o React não tenha criado).

**Arquivo App.js, esse arquivo é o responsável por chamar as rotas dos componentes:**

```
import React from 'react';  
import { BrowserRouter as Router, Route, Routes, Link } from 'react-router-dom';  
import CreateDoador from './components/CreateDoador';  
import CreateDoacao from './components/CreateDoacao';  
import DoadorList from './components/DoadorList';  
import DoacaoList from './components/DoacaoList';  
  
function App() {  
  return (  
    <Router>  
      <div className="container mx-auto p-4">
```

```

    <nav className="mb-4 bg-gray-800 p-4 rounded-lg shadow-md">
      <ul className="flex space-x-4 justify-center">
        <li>
          <Link to="/" className="text-white hover:text-gray-300">Lista de Doadores</Link>
        </li>
        <li>
          <Link to="/doadores" className="text-white hover:text-gray-300">Criar
Doador</Link>
        </li>
        <li>
          <Link to="/doacoes" className="text-white hover:text-gray-300">Criar
Doação</Link>
        </li>
        <li>
          <Link to="/doacoes/list" className="text-white hover:text-gray-300">Lista de
Doações</Link>
        </li>
      </ul>
    </nav>
    <Routes>
      <Route path="/" element={<DoadorList />} />
      <Route path="/doadores" element={<CreateDoador />} />
      <Route path="/doacoes" element={<CreateDoacao />} />
      <Route path="/doacoes/list" element={<DoacaoList />} />
    </Routes>
  </div>
</Router>
);
}

export default App;

```

### Arquivo index.js:

```

import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

```

```
// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

**Arquivo config.js (que contém apenas a url da api):**

```
const API_URL = 'http://127.0.0.1:5000';

export default API_URL;
```

**O arquivo index.css deve ficar dessa forma:**

```
@tailwind base;
@tailwind components;
@tailwind utilities;

@keyframes fade-in {
  0% {
    opacity: 0;
    transform: scale(0.95);
  }
  100% {
    opacity: 1;
    transform: scale(1);
  }
}

.animate-fade-in {
  animation: fade-in 0.3s ease-in-out;
}
```

**O arquivo App.css precisa de algumas alterações também:**

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}
```

```
@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}
```

```
.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}
```

```
.App-link {
  color: #61dafb;
}
```

```
@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

=====

Esses são os arquivos iniciais do projeto, com eles o servidor já pode ser iniciado com o comando:

*npm start*

Com os arquivos de configuração inicial criados vamos aos componentes do nosso frontend. Crie uma pasta chamada components dentro da pasta src para criar os componentes

Para que fique mais fácil e a formatação do documento não quebre, irei fornecer o link dos códigos do componente que está no github.

A o primeiro componente a ser criado será a página de criar nova doação. Crie o arquivo CreateDoacao.js e insira o código que está no link:

<https://github.com/FelipeGuglielmeli/DoacaoFront/blob/master/src/components/CreateDoacao.js>

O segundo componente será a página de criar um novo doador. O arquivo se chamará CreateDoador, insira o código que está no link:

<https://github.com/FelipeGuglielmeli/DoacaoFront/blob/master/src/components/CreateDoador.js>

A terceira página será a que lista todas as doações. O arquivo se chamará DoacaoList.js, insira o código que está no link:

<https://github.com/FelipeGuglielmeli/DoacaoFront/blob/master/src/components/DoacaoList.js>

A quarta e última página é a que lista todos os doadores. O arquivo se chamará DoadorList.js, insira o código do link:

<https://github.com/FelipeGuglielmeli/DoacaoFront/blob/master/src/components/DoadorList.js>

E por último precisamos de um Modal que mostra todas as doações de um determinado doador, crie um componente chamado Modal.js e insira o seguinte código:

<https://github.com/FelipeGuglielmeli/DoacaoFront/blob/master/src/components/Modal.js>

No fim a estrutura do seu projeto deve parecer com isso :

