Guia de Contribuição para o Phishing Manager

Bem-vindo ao projeto Phishing Manager! Agradecemos seu interesse em contribuir para tornar esta ferramenta ainda melhor. Este guia tem como objetivo fornecer as informações necessárias para que você possa começar a contribuir de forma eficaz e seguir as melhores práticas do projeto.

1. Como Contribuir

Existem diversas formas de contribuir para o Phishing Manager:

- **Relatar Bugs**: Se você encontrar um bug, por favor, abra uma issue no GitHub com uma descrição clara e passos para reproduzir o problema.
- **Sugerir Novas Funcionalidades**: Tem uma ideia para melhorar o Phishing Manager? Abra uma issue para discutir a funcionalidade proposta.
- **Escrever Código**: Contribua com novas funcionalidades, correções de bugs ou melhorias de desempenho. Siga as diretrizes de codificação e abra um Pull Request (PR).
- **Melhorar a Documentação**: A documentação é crucial. Ajude a aprimorar este guia, a referência da API, o manual do usuário ou qualquer outra parte da documentação.
- **Revisar Código**: Ajude a revisar Pull Requests de outros contribuidores, fornecendo feedback construtivo.
- **Testar**: Ajude a testar novas funcionalidades ou a reproduzir bugs relatados.

2. Configurando seu Ambiente de Desenvolvimento

Para começar a desenvolver no Phishing Manager, siga os passos abaixo para configurar seu ambiente.

2.1. Pré-requisitos

Certifique-se de ter as seguintes ferramentas instaladas em seu sistema:

- **Git**: Para controle de versão.
- **Python 3.8+**: Para o backend Flask.
- **Node.js 14+ e npm/yarn**: Para o frontend React.
- **Docker e Docker Compose** (Opcional, mas recomendado para desenvolvimento).

2.2. Clonando o Repositório

Primeiro, clone o repositório do Phishing Manager:

```
git clone https://github.com/Dedeg0/phishing-manager.git
cd phishing-manager
```

2.3. Configurando o Backend (Flask)

É altamente recomendável usar um ambiente virtual para isolar as dependências do projeto.

```
# Navegue para o diretório do backend
cd phishing-manager/phishing-manager

# Crie e ative um ambiente virtual
python3 -m venv venv
source venv/bin/activate  # No Windows: venv\Scripts\activate

# Instale as dependências do Python
pip install -r requirements.txt

# Inicialize o banco de dados (SQLite por padrão)
flask db upgrade

# Crie um usuário administrador (opcional, para testes)
flask create-admin --username admin --email admin@example.com --password
SenhaSegura123!

# Execute o backend
flask run
```

O backend estará disponível em `http://127.0.0.1:5000`.

2.4. Configurando o Frontend (React)

```
# Navegue para o diretório do frontend
cd phishing-manager/phishing-manager-frontend # Assumindo que o frontend está em um diretório separado

# Instale as dependências do Node.js
npm install # ou yarn install

# Inicie o servidor de desenvolvimento do React
npm start # ou yarn start
```

O frontend estará disponível em `http://localhost:3000`.

2.5. Usando Docker (Recomendado)

Para uma configuração mais fácil e consistente, você pode usar Docker e Docker Compose.

```
# No diretório raiz do projeto (onde está o docker-compose.yml)
docker-compose up --build
```

Isso irá construir e iniciar os serviços de backend e frontend em contêineres Docker.

3. Fluxo de Trabalho de Contribuição

Seguimos o fluxo de trabalho de

Pull Request (PR) para todas as contribuições de código.

1. **Fork o Repositório**: Faça um fork do repositório `Dedeg0/phishing-manager` para sua conta GitHub.
2. **Crie uma Branch**: Crie uma nova branch a partir da branch `main` para sua funcionalidade ou correção de bug. Use um nome descritivo, como `feature/nova-funcionalidade` ou `bugfix/correcao-autenticacao`. `bash git checkout -b feature/sua-nova-feature`
3. **Faça suas Alterações**: Implemente suas alterações no código. Certifique-se de seguir as diretrizes de codificação e escrever testes para suas alterações.
4. **Commit suas Alterações**: Faça commits atômicos e com mensagens claras e descritivas. Cada commit deve representar uma única alteração lógica. `bash git`

`commit -m "feat: Adiciona nova funcionalidade X"`

5. **Envie para seu Fork**: Envie suas alterações para seu repositório forked no GitHub. `bash git push origin feature/sua-nova-feature`
6. **Abra um Pull Request (PR)**: Vá para o repositório original no GitHub e abra um Pull Request da sua branch para a branch `main` do projeto principal. Forneça uma descrição detalhada das suas alterações, incluindo:
7. O problema que o PR resolve (se for um bug).
8. A funcionalidade que o PR adiciona.
9. Como as alterações foram testadas.
10. Quaisquer considerações adicionais.
11. **Revisão de Código**: Seu PR será revisado pela equipe do projeto. Esteja preparado para receber feedback e fazer ajustes. Uma vez aprovado, seu PR será mesclado na branch `main`.

4. Diretrizes de Codificação

Para manter a consistência e a qualidade do código, siga as seguintes diretrizes:

4.1. Python (Backend)

- **PEP 8**: Siga as diretrizes de estilo do PEP 8. Utilize ferramentas como `flake8` ou `black` para formatar o código automaticamente.
- **Docstrings**: Escreva docstrings claras e concisas para todos os módulos, classes, funções e métodos, explicando seu propósito, argumentos e valores de retorno.
- **Comentários**: Use comentários para explicar partes complexas do código ou decisões de design não óbvias.
- **Nomenclatura**: Use `snake_case` para nomes de variáveis, funções e módulos. Use `CamelCase` para nomes de classes.
- **Tratamento de Erros**: Utilize blocos `try-except` para lidar com exceções de forma graciosa. Evite `pass` em blocos `except` vazios.
- **Segurança**: Esteja sempre atento a possíveis vulnerabilidades de segurança (SQL Injection, XSS, etc.) e utilize as ferramentas e práticas de segurança implementadas no projeto.

4.2. JavaScript/React (Frontend)

- **ESLint/Prettier**: Utilize ESLint para linting e Prettier para formatação de código. As configurações do projeto devem ser seguidas.
- **Componentes Reutilizáveis**: Crie componentes React pequenos, focados e reutilizáveis. Siga o princípio de

componentes puros sempre que possível. - **Gerenciamento de Estado**: Utilize a Context API do React ou bibliotecas de gerenciamento de estado (se houver) para gerenciar o estado da aplicação de forma eficiente. - **Chamadas de API**: Centralize as chamadas de API em módulos de serviço (`src/services/`) para manter a lógica de comunicação separada dos componentes da UI. - **Acessibilidade**: Desenvolva com acessibilidade em mente, utilizando atributos ARIA e garantindo que a interface seja utilizável por todos. - **Responsividade**: Certifique-se de que a interface seja responsiva e funcione bem em diferentes tamanhos de tela (desktop, tablet, mobile).

5. Testes

Testes são uma parte crucial do processo de desenvolvimento. Todas as novas funcionalidades e correções de bugs devem vir acompanhadas de testes apropriados.

- **Testes Unitários**: Para funções e classes individuais (backend) e componentes/funções puras (frontend).
- **Testes de Integração**: Para verificar a comunicação entre módulos (backend) e a interação entre componentes (frontend).
- **Testes End-to-End (E2E)**: Para simular o fluxo completo do usuário na aplicação.

5.1. Executando Testes

Backend

```
# No diretório phishing-manager/phishing-manager
python3 -m pytest tests/
```

Para verificar a cobertura de código:

```
python3 -m pytest tests/ --cov=src --cov-report=term-missing
```

Frontend

(Assumindo que você está no diretório `phishing-manager/phishing-manager-frontend`)

```
npm test # ou yarn test
```

6. CI/CD (Integração Contínua / Entrega Contínua)

O projeto utiliza GitHub Actions para automação do pipeline de CI/CD. Ao abrir um Pull Request, os testes automatizados serão executados automaticamente para garantir que suas alterações não introduzam regressões.

- **`.github/workflows/ci.yml`**: Este arquivo define o pipeline de CI/CD, incluindo a execução de testes, linting e outras verificações de qualidade de código.

7. Documentação

Manter a documentação atualizada é tão importante quanto manter o código. Se você adicionar uma nova funcionalidade ou alterar uma existente, certifique-se de atualizar a documentação relevante:

- **`ARCHITECTURE_GUIDE.md`**: Para mudanças na arquitetura geral do sistema.
- **`API_REFERENCE.md`**: Para novas rotas de API, mudanças em endpoints existentes, ou novos modelos de requisição/resposta.
- **`DATABASE_SCHEMA.md`**: Para mudanças no esquema do banco de dados (novas tabelas, campos, relacionamentos).
- **`USER_MANUAL.md`**: Para novas funcionalidades que afetam a experiência do usuário final.
- **`INSTALLER_GUIDE.md`**: Para mudanças no processo de instalação ou configuração.

8. Boas Práticas Adicionais

- **Mantenha-se Atualizado**: Antes de começar a trabalhar em uma nova funcionalidade ou correção, puxe as últimas alterações da branch `main` para evitar conflitos.
- **Comunique-se**: Se você tiver dúvidas, precisar de ajuda ou quiser discutir uma ideia, use as issues do GitHub ou o canal de comunicação da equipe (se houver).
- **Seja Paciente**: O processo de revisão de código leva tempo. Seja paciente e esteja aberto a feedback construtivo.

Autor: Manus AI **Data:** 28 de Junho de 2025

Referências:

[1] GitHub Flow: https://docs.github.com/en/get-started/quickstart/github-flow [2] PEP 8 -- Style Guide for Python Code: https://peps.python.org/pep-0008/ [3] React Documentation: https://react.dev/ [4] Pytest Documentation: https://docs.pytest.org/en/stable/ [5] GitHub Actions Documentation: https://docs.github.com/en/actions