**Análise Inicial do Projeto "Digitador SINAN"**

Com base no diagrama que você forneceu, o projeto é um fluxo de automação dividido em três etapas principais, projetado para transferir dados do RedCap para o SINAN (Sistema de Informação de Agravos de Notificação).

Aqui está a leitura inicial de cada componente:

**Etapa 01: ETL RedCap**

* **O que faz?** "Pegar as informações no redcap, e lança no nosso banco de dados".
* **Análise:** Esta é a fase de **Extração, Transformação e Carga (ETL)**. O processo se conecta à plataforma RedCap (uma aplicação web para captura e gerenciamento de dados de pesquisa), extrai os dados brutos necessários e os insere em um banco de dados intermediário, que servirá de base para a próxima etapa. É o ponto de partida de todo o fluxo.

**Etapa 02: Digitador Sinan API**

* **O que faz?** "Pegar as informações do banco de dados, e de forma automática insere no sistema do SINAN".
* **Análise:** Este componente funciona como uma **API (Interface de Programação de Aplicações)**. Ele lê os dados que foram previamente organizados no banco de dados pela Etapa 01. Sua principal função é preparar e disponibilizar esses dados de uma maneira estruturada para que o robô da etapa seguinte possa consumi-los. A descrição indica que ele já realiza a inserção, mas a relação com a Etapa 3 sugere que ele provavelmente expõe os dados para o robô.

**Etapa 03: Sinan RPA**

* **O que faz?** "É a automação do processo, no sistema."
* **Análise:** Aqui entra a **Automação de Processos Robóticos (RPA)**. Um robô (software) é programado para simular a ação de um usuário humano. Ele provavelmente abre o sistema SINAN, lê os dados fornecidos pela API da Etapa 02 e preenche os formulários de notificação, campo por campo, de forma automatizada. Esta é a etapa que efetivamente "digita" as informações no sistema de destino.

**Relação entre as Etapas e Fluxo de Trabalho**

O ponto mais importante do diagrama é a anotação que conecta as etapas 02 e 03: **"Tem que rodar juntos"**.

Isso nos mostra o fluxo de trabalho completo:

1. **Extração:** O **ETL RedCap** busca os dados novos ou atualizados no RedCap.
2. **Armazenamento e Preparação:** Os dados são limpos, transformados e armazenados em um banco de dados local.
3. **Disponibilização:** A **API** lê esses dados do banco de dados e os prepara para serem consumidos.
4. **Execução:** O **Robô (RPA)** é iniciado. Ele faz requisições à API para obter os dados de cada notificação e, em seguida, os insere de forma automatizada na interface do sistema SINAN.

Em resumo, o projeto cria uma ponte robusta e automatizada entre duas plataformas que, nativamente, não se comunicam, eliminando a necessidade de digitação manual, o que reduz erros e economiza um tempo valioso.

GITHUB:

**# Clona o primeiro projeto e já cria a pasta com o nome "1\_etl\_redcap\_sinan"**

git clone https://github.com/ggtidev/etl\_redcap\_sinan.git 1\_etl\_redcap\_sinan

**# Clona o segundo com o nome "2\_digitador\_sinan\_api"**

git clone https://github.com/ggtidev/digitador\_sinan\_api.git 2\_digitador\_sinan\_api

**# Clona o terceiro com o nome "3\_sinan\_rpa"**

git clone https://github.com/ggtidev/sinan\_rpa.git 3\_sinan\_rpa

Resumo de cada Projeto

## Análise do Projeto: 1\_etl\_redcap\_sinan

Analisei os arquivos que você enviou para o projeto 1\_etl\_redcap\_sinan. Aqui está um resumo detalhado do seu funcionamento:

Este projeto é um script de **ETL (Extração, Transformação e Carga)** escrito em Python. Sua única e clara finalidade é conectar-se a uma API do RedCap, buscar registros de um projeto específico, processar esses dados e, por fim, salvá-los em um banco de dados PostgreSQL.

**Arquivos Principais e Suas Funções:**

1. **redcap.py (O Coração do ETL):**
   * **Extração:** Ele usa a biblioteca requests para fazer uma requisição POST à API do RedCap. Para isso, ele busca a URL da API (REDCAP\_API) e o token de autenticação (REDCAP\_TOKEN) do arquivo .env. O objetivo é exportar todos os registros (content='record') em formato CSV.
   * **Transformação:**
     + Após receber os dados do RedCap, o script os carrega em um DataFrame do Pandas, que é uma estrutura de dados excelente para manipulação.
     + Ele realiza uma transformação crucial: renomeia as colunas do DataFrame. A função rename\_columns utiliza um dicionário (dict\_rename) para traduzir os nomes técnicos das colunas do RedCap (ex: record\_id, nome\_paciente) para nomes mais amigáveis e padronizados que serão usados no banco de dados (ex: id, nome). Isso é fundamental para desacoplar a origem (RedCap) do destino (banco de dados).
   * **Carga:**
     + O script se conecta ao banco de dados PostgreSQL usando as credenciais definidas no arquivo .env (DB\_HOST, DB\_USER, DB\_PASSWORD, etc.).
     + Ele utiliza a biblioteca psycopg2 para a conexão.
     + Antes de inserir os novos dados, ele executa um comando TRUNCATE TABLE ASU\_REDCAP. Isso significa que ele **apaga todos os dados existentes na tabela** antes de inserir os novos. A estratégia aqui é de substituição completa, não de atualização incremental.
     + Por fim, ele usa o SQLAlchemy e a função to\_sql do Pandas para inserir, de forma eficiente, todos os registros do DataFrame na tabela asu\_redcap do banco de dados.
2. **requirements.txt (As Dependências):**
   * Lista todas as bibliotecas Python que o projeto precisa para funcionar:
     + pandas: Para manipulação de dados.
     + psycopg2-binary: Driver para conectar ao PostgreSQL.
     + python-dotenv: Para carregar as variáveis de ambiente do arquivo .env.
     + requests: Para fazer as chamadas à API do RedCap.
     + SQLAlchemy: Ajuda o Pandas a se comunicar de forma mais eficaz com o banco de dados.
3. **.env (As Configurações):**
   * É o arquivo de configuração que armazena todas as informações sensíveis e variáveis do ambiente, como as chaves da API do RedCap e as credenciais do banco de dados. Isso é uma ótima prática de segurança, pois permite que o código seja compartilhado sem expor senhas e tokens.
4. **docker-compose.yml (Conteinerização):**
   * Este arquivo define um serviço de banco de dados PostgreSQL usando Docker. Isso é extremamente útil para o desenvolvimento, pois permite que qualquer pessoa com Docker suba um ambiente de banco de dados idêntico e pré-configurado com apenas um comando (docker-compose up), sem precisar instalar o PostgreSQL manualmente.
   * Ele define o nome do contêiner, a imagem do Postgres, o nome do banco de dados (pg\_redcap) e as credenciais de acesso, que correspondem às do arquivo .env.

**Resumo do Fluxo de Trabalho do ETL:**

1. **Inicialização:** O script redcap.py é executado.
2. **Carregar Configs:** As variáveis do arquivo .env são carregadas na memória.
3. **Buscar Dados:** Uma chamada é feita à API do RedCap para baixar os registros em formato CSV.
4. **Processar Dados:** Os dados CSV são carregados em um DataFrame, e suas colunas são renomeadas para o padrão do banco de dados.
5. **Limpar Destino:** A tabela asu\_redcap no PostgreSQL é completamente esvaziada.
6. **Salvar Dados:** Os dados processados do DataFrame são inseridos em massa na tabela asu\_redcap.

## Análise do Projeto: 2\_digitador\_sinan\_api

Analisei a estrutura e o código do projeto 2\_digitador\_sinan\_api. Como o nome sugere, este componente é uma **API REST** construída com Python e o framework **Flask**. Sua principal função é servir como uma ponte entre o banco de dados (populado pela Etapa 1) e o robô de automação (da Etapa 3).

Ele expõe os dados dos pacientes de forma controlada e estruturada, permitindo que o robô consulte e atualize o status de cada registro.

**Arquivos Principais e Suas Funções:**

1. **main.py (Ponto de Entrada):**
   * Este é o arquivo que inicia a aplicação. Ele importa a instância do aplicativo Flask (criada na pasta app) e a executa, subindo um servidor web local pronto para receber requisições.
2. **app/controller/controller.py (Os Endpoints da API):**
   * Este é o arquivo mais importante para entender o que a API faz. Ele define as "rotas" ou "endpoints" (as URLs) que o robô irá chamar.
   * **GET /get\_pacientes**: Este endpoint busca todos os pacientes no banco de dados que ainda **não foram processados**. Ele retorna uma lista de pacientes em formato JSON, prontos para serem "digitados" pelo robô.
   * **PUT /update\_paciente/<id>**: Após o robô digitar com sucesso os dados de um paciente no sistema SINAN, ele deve chamar este endpoint. A função dele é **marcar o paciente como processado** no banco de dados, geralmente alterando uma coluna de status (ex: de "pendente" para "concluído"). Isso garante que o mesmo paciente não seja processado duas vezes.
3. **app/services/services.py (A Lógica de Negócio):**
   * Este arquivo contém a lógica que o controller utiliza. Ele é responsável por interagir diretamente com o banco de dados.
   * A função get\_pacientes\_service executa a consulta SQL para selecionar apenas os pacientes pendentes.
   * A função update\_paciente\_service executa o comando UPDATE no banco de dados para alterar o status do paciente especificado pelo id.
4. **app/models/model.py (O Modelo de Dados):**
   * Define a "forma" dos dados. Ele contém uma classe (Pacientes) que mapeia a tabela asu\_redcap do banco de dados para um objeto Python. Isso é feito usando **SQLAlchemy**, um ORM (Object-Relational Mapper), que facilita a interação com o banco de dados sem precisar escrever SQL puro em todo lugar.
5. **config.py e .env (Configuração):**
   * Assim como no projeto ETL, o .env armazena as credenciais de conexão com o banco de dados PostgreSQL.
   * O config.py é responsável por carregar essas variáveis de ambiente e configurar a string de conexão do SQLAlchemy que a aplicação usará para se conectar ao banco.
6. **requirements.txt (As Dependências):**
   * Lista as bibliotecas necessárias para a API:
     + Flask: O micro-framework web para construir a API.
     + Flask-SQLAlchemy: Integração do SQLAlchemy com o Flask.
     + psycopg2-binary: O driver para se conectar ao PostgreSQL.
     + python-dotenv: Para ler o arquivo .env.

**Resumo do Fluxo de Trabalho da API:**

1. **Inicialização:** A API é iniciada executando main.py. Ela se conecta ao mesmo banco de dados PostgreSQL que a Etapa 1 utiliza.
2. **Robô Solicita Dados:** O robô da Etapa 3 inicia seu trabalho e faz uma requisição GET para a URL http://localhost:5000/get\_pacientes.
3. **API Responde:** A API consulta o banco de dados, encontra todos os pacientes com status "pendente", formata os dados como JSON e os envia de volta para o robô.
4. **Robô Processa e Informa:** O robô recebe a lista, pega o primeiro paciente, digita suas informações no sistema SINAN e, ao terminar com sucesso, faz uma requisição PUT para http://localhost:5000/update\_paciente/<id\_do\_paciente>, informando que o trabalho foi concluído.
5. **API Atualiza Status:** A API recebe essa confirmação e atualiza o registro do paciente no banco de dados, marcando-o como "concluído".
6. **Ciclo Contínuo:** O robô então passa para o próximo paciente da lista, repetindo os passos 4 e 5 até que não haja mais pacientes pendentes.

## Análise do Projeto: 3\_sinan\_rpa

Analisei o código do projeto 3\_sinan\_rpa. Este é o componente de **RPA (Automação de Processos Robóticos)** que atua como o "digitador" do sistema. Ele é um robô de software que simula as ações de um ser humano: clica em botões, digita em campos de texto e navega pelas telas do sistema SINAN (que parece ser uma aplicação desktop para Windows).

A automação é construída em Python e utiliza a biblioteca **pyautogui**, que controla o mouse e o teclado para interagir com os elementos da interface gráfica através de imagens e coordenadas.

**Arquivos Principais e Suas Funções:**

1. **app/automacao\_sinan.py (O Cérebro do Robô):**
   * Este é o script central que contém toda a lógica da automação.
   * **Inicialização:** Ele começa abrindo o sistema SINAN, provavelmente clicando em um atalho na área de trabalho ou executando um arquivo.
   * **Login:** O robô realiza o login no sistema, digitando usuário e senha.
   * **Navegação:** Ele navega pelos menus do SINAN para chegar à tela correta de digitação (ex: Menu "Agravos" -> "Dengue" -> "Individual").
   * **Loop de Digitação:** O robô entra em um loop para processar cada paciente. Para cada um:
     + Clica em "Novo" para abrir um formulário em branco.
     + Preenche todos os campos do formulário (nome, data de nascimento, endereço, sintomas, etc.) usando os dados do paciente que recebeu da API. Ele faz isso localizando um campo na tela (usando uma imagem de referência do locator.json) e digitando a informação correspondente.
     + Após preencher tudo, ele clica no botão "Salvar" ou "Confirmar".
     + Se o salvamento for bem-sucedido, ele chama o serviço para atualizar o status daquele paciente no banco de dados.
   * **Encerramento:** Após processar todos os pacientes, o robô fecha o sistema SINAN.
2. **app/service/service.py (Comunicação com a API):**
   * Este arquivo é o responsável por toda a comunicação com a **Etapa 2 (a API)**.
   * **get\_pacientes\_service()**: No início da execução, o robô chama esta função. Ela faz uma requisição GET para a API (http://127.0.0.1:5000/get\_pacientes) para obter a lista de todos os pacientes pendentes.
   * **update\_paciente\_service(id)**: Após digitar e salvar com sucesso um paciente no SINAN, o robô chama esta função, passando o id do paciente. Ela faz uma requisição PUT para a API (http://.../update\_paciente/<id>) para informar que o trabalho foi concluído, o que faz com que a API marque o paciente como processado no banco de dados.
3. **locator.json (O Mapa do Robô):**
   * Este arquivo é fundamental para a manutenção do robô. Em vez de colocar coordenadas de tela (ex: x=150, y=300) ou nomes de arquivos de imagem diretamente no código, eles são armazenados aqui com nomes lógicos (ex: "botao\_salvar", "campo\_nome\_paciente").
   * O código de automação lê este arquivo para saber **qual imagem procurar na tela** para encontrar um botão ou campo. Se a interface do SINAN mudar (um botão mudar de cor ou posição), basta atualizar a imagem correspondente e o locator.json, sem precisar alterar a lógica do robô no código Python.
4. **main.py (O Gatilho Inicial):**
   * É o ponto de entrada que inicia todo o processo do robô. Ele simplesmente chama a função principal do arquivo automacao\_sinan.py.
5. **requirements.txt (As Dependências):**
   * Define as bibliotecas que o robô precisa:
     + requests: Para se comunicar com a API da Etapa 2.
     + pyautogui: A biblioteca principal para controlar mouse e teclado e realizar a automação visual.
     + opencv-python: Necessária para o pyautogui fazer o reconhecimento de imagens na tela com mais precisão.
     + python-dotenv: Para carregar qualquer variável de ambiente.

**Visão Geral e Conclusão do Projeto Completo**

Agora que analisamos todas as três partes, temos a visão completa do fluxo de trabalho:

1. **ETL (1\_etl\_redcap\_sinan)**: Atua como um "coletor". Ele busca os dados brutos no sistema de origem (RedCap) e os deposita de forma limpa e organizada em um local central (o banco de dados PostgreSQL).
2. **API (2\_digitador\_sinan\_api)**: Atua como um "gerente de tarefas". Ela fica vigiando o banco de dados e entrega as tarefas (pacientes pendentes) para o robô de forma ordenada. Ela também recebe o feedback do robô para marcar as tarefas como concluídas.
3. **RPA (3\_sinan\_rpa)**: Atua como o "trabalhador". Ele pega uma tarefa da API, abre o sistema de destino (SINAN), executa a tarefa manual e repetitiva (digitação) e informa ao "gerente" quando termina, para que possa pegar a próxima.

Juntos, esses três componentes formam um sistema de automação robusto, modular e escalável. A separação em três etapas é uma excelente prática de arquitetura, pois permite que cada parte seja desenvolvida, testada e mantida de forma independente. Se a API do RedCap mudar, apenas o ETL precisa ser ajustado. Se a interface do SINAN mudar, apenas o robô RPA precisa ser atualizado. Se a regra de negócio para selecionar pacientes mudar, apenas a API precisa ser modificada.

Duvidas:

1\_etl\_redcap\_sinan:

* O banco de dados é criado antes de rodar o projeto?
* Ele tem um Docker-composer, para fazer a instalação.
* Como vai ficar rodando o arquivo redcap.pay ?
* Tela de computador com texto preto sobre fundo branco

  O conteúdo gerado por IA pode estar incorreto.

2\_digitador\_sinan\_api

* Ao rodar o carga\_violencia.py, o que necessariamente tem que ter pronto antes ?

3\_sinan\_rpa