**1) Visão geral (entregável final)**

A) Para cada docente que respondeu (campo sis\_user) o sistema deve:

1. Extrair respostas do questionário do Canvas (CSV ou JSON) via API. (https://famonline.instructure.com/courses/15812/modules)
2. Tratar / agregar os dados (por pergunta, percentuais, média, etc).
3. Renderizar um PDF estilizado (igual/semelhante ao da CPA) — um PDF por docente ou um consolidado conforme regra.
4. Salvar local e **disponibilizar** (upload via Canvas API para diretório do Canvas definido **ou** enviar por e-mail).
5. Executar automaticamente (por lote), com logs, retries e monitoramento.

B) Para cada mês:

1. Renderizar um PDF macro estilizado com o consolidado mensal
2. Salvar local e em cloud (a definir)
3. Executar após execução granulada.

**2) Roadmap detalhado (etapas / entregas)**

## ****Fase 0 — Preparação****

* Instalar ambiente Python 3.10+ no PC.
* Criar pasta do projeto (reports\_project/).
* Criar ambiente virtual (python -m venv venv).
* Instalar dependências mínimas:
* pip install pandas weasyprint jinja2 matplotlib
* Guardar o token do Canvas em um .env (ou variável de ambiente).

## ****Fase 1 — Coletar dados (Canvas → CSV local)****

* Usar API do Canvas para baixar as respostas do questionário.
  + Pode usar requests ou a lib canvasapi.
  + Salvar como CSV em data/raw/2025-02\_respostas.csv.
* **(Atalho inicial)**: se já tiver o CSV exportado manualmente, pode começar direto pelo arquivo local e só depois automatizar a coleta.

## ****Fase 2 — Tratamento com Pandas****

* Abrir o CSV em um DataFrame.
* Identificar colunas importantes (sis\_user, pergunta, resposta, data, etc).
* Criar funções para:
  + Filtrar respostas por docente (sis\_user).
  + Calcular estatísticas básicas (percentuais de concordo, discordo, etc).
  + Gerar um dict ou DataFrame já pronto para enviar ao relatório.

## ****Fase 3 — Geração de PDF (mínimo viável)****

* Criar um **template HTML simples** com:
  + Título do relatório,
  + Nome do docente,
  + Uma tabela com estatísticas.
* Preencher esse template com Jinja2.
* Converter HTML → PDF com WeasyPrint.
* Salvar o PDF em data/reports/{sis\_user}.pdf.

No início: pode ser **texto puro**. Depois, com ajuda do pessoal de HTML/CSS, você deixa parecido com o print que me mostrou.

## ****Fase 4 — Loop por docente****

* Pegar a lista de sis\_user únicos no CSV.
* Para cada docente:
  + Rodar o tratamento,
  + Gerar 1 PDF,
  + Salvar em /reports/.
* Resultado: 60 relatórios (um por docente).

## ****Fase 5 — Distribuição (manual no início)****

* Como é local/manual:
  + Entregue PDFs em pasta compartilhada (OneDrive, Google Drive, etc).
  + **OU** envie manualmente por e-mail.
* Automatizar envio via Canvas API ou SMTP só quando for necessário.

## ****Fase 6 — Melhorias futuras (se expandir depois)****

* Automatizar a coleta (API Canvas rodando agendada).
* Automatizar envio (upload para Canvas ou e-mails automáticos).
* Refinar layout (HTML + CSS bonito, gráficos em matplotlib).
* Adicionar logging simples (salvar log.txt).
* Rodar automaticamente (cron job, GitHub Actions ou script agendado no Windows Task Scheduler).

**3) Tecnologias / bibliotecas recomendadas**

* Linguagem: **Python 3.10+**
* Dependências principais:
  + requests (http generic)
  + canvasapi (facilita Canvas; opcional)
  + pandas (tratamento de dados)
  + numpy
  + Jinja2 (templating HTML)
  + WeasyPrint ou pdfkit/wkhtmltopdf ou ReportLab (gerar PDF)
  + matplotlib (gráficos)
  + openpyxl (se precisar xlsx)
  + pytest (testes)
  + loguru ou structlog / logging (logs)
  + python-dotenv / dynaconf (config)
  + boto3 (se usar S3) / sendgrid (se enviar por API)
  + docker (containerizar)
* Infra/serviços:
  + Servidor Linux com Docker OR AWS/GCP/Azure (se preferir cloud).
  + Banco (opcional) Postgres para histórico de execuções e metadados.
  + Redis + Celery (se precisar processamento assíncrono em grande escala).
  + GitHub/GitLab + CI (GitHub Actions).

**4) Arquitetura e fluxo (texto)**

1. **Scheduler** (cron / GitHub Actions / Airflow) dispara job.
2. **Extractor**: Canvas API → raw CSV/JSON → /data/raw.
3. **Transformer**: pandas → normalized DataFrame.
4. **Renderer**: preenche template Jinja2, gera gráficos, weasyprint → PDF.
5. **Uploader**: POST para Canvas Files API (ou enviar e-mail).
6. **Recorder**: grava status em DB/log.
7. **Alert**: notifica no Slack/e-mail se erro crítico.

**5) Estrutura de repositório recomendada**

**6) Boas práticas operacionais**

* Não armazene tokens no código — use variáveis de ambiente/secret manager.
* Teste primeiro em ambiente de homologação do Canvas.
* Implemente idempotência (se job reexecutado não duplicar uploads).
* Trate rate limits: respeitar Retry-After e exponential backoff.
* Garantir logs legíveis e arquivo de auditoria por relatório (quem gerou, quando, etc).
* Privacidade: anonimizar dados sensíveis se necessário; cumprir LGPD.

**7) MVP mínimo (para entregar algo útil rápido)**

1. Script que baixa o CSV do Canvas (manual ou token).
2. Script transformer.py que produz um DataFrame com métricas por sis\_user.
3. Um template HTML simples + geração de **um** PDF local para um sis\_user.
4. Execução local: python orchestrator.py --sis\_user <id> produce /data/reports/<id>.pdf.

Esse MVP permite validar template/visual e ajustes com usuários (secretaria).

**8) Riscos comuns e mitigações**

* **Sem acesso à API** → solicitar acesso/homologação na TI da instituição (prioridade).
* **CSV/Schema inconsistentes** → criar validações e contrato de dados; negociar padrão.
* **Design do PDF demorado** → priorizar HTML simples e iterar com stakeholders.
* **Rate limits do Canvas** → implementar throttling e retries.
* **Privacidade / LGPD** → documentar base legal e proteger storage.