

Desafio 04:

Automação da compra de VR/VA

Sumário

1.Introdução

2. Requisitos

- 2.1 Requisitos por sistema
- 2.2 Recursos mínimos recomendados
- 2.3 Portas / Firewall
- 2.4 Proxy/Corporate
- 2.5 Sistema de arquivos & permissões

3. Arquitetura

- 3.1 Fatores que Influenciam o Desempenho
- 3.2 Escalabilidade

4. Visão geral

- 4.1 main.py
- 4.2 agentes.py
- 4.3 calendars.py
- 4.4 tools.py
- 4.5 config.py
- 4.6 data_schemas.py
- 4.7 export_layout.py
- 4.8 validations.py
- 4.9 vectorstore_setup.py

5. Ambiente, Dependências e Deploy

- 5.1 DockerFile
- 5.2 Requeriments
- 5.3 docker-compose.yml
- 5.4 docker-compose.windows.yml

6. Passo a passo

- 6.1 Linux
- 6.2 Windows

7. Funcionamento na prática

8. Conclusão

9. Referências Bibliográficas

1- Introdução

Automação de compra de VR/VA (Agentes + Containers)

Este documento mostra o passo a passo da solução para automação do processo mensal de compra de Vale Refeição (VR/VA),

Todo ambiente está containerizado, e foi orquestrado com agentes de IA: LangGraph e Ollama, os mesmos consultam as regras e validações.

Com funções criadas realizamos a consolidação, saneamento/validação, exclusões de elegibilidade e cálculo.

O pipeline integra as bases de Ativos, Férias, Desligados, Cadastral (admitidos), Sindicato x Valor e Dias úteis por colaborador, aplica as regras de férias (integrais/parciais), admissões/desligamentos (incluindo a regra do comunicado até dia 15) e considera feriados por sindicato/UF/município.

Para a entrega é gerada a planilha VR Mensal contendo o valor do VR por colaborador, além do rateio de custo (empresa 80% / profissional 20%), em conformidade com o solicitado na descrição do Desafio.

2 - Requisitos

2.1 Requisitos por Sistema

Linux

- Docker Engine 24+ e Docker Compose V2 (docker compose version).
- Virtualização habilitada no BIOS/UEFI.
- SELinux (Fedora/RHEL):
- Preferencial: usar volumes com :Z (já previsto no compose Linux).
- Para o diretório do Ollama (./ollama), se ocorrer “permission denied”, aplique um dos ajustes:
`sudo chcon -R -t container_file_t ./ollama` ou

Windows

- Docker Desktop 4.27+ com WSL 2 backend habilitado.
- Durante a instalação, marque “Use WSL 2 instead of Hyper-V”.
- WSL 2 ativo no Windows e um kernel atualizado (o Docker Desktop guia esse passo).
- Virtualização habilitada no BIOS/UEFI (Intel VT-x/AMD-V).
- Permissões de administrador para instalar o Docker Desktop.
- Espaço em disco: ver “Recursos recomendados” abaixo.

2.2 Recursos mínimos recomendados

- CPU: 2 vCPUs (4 vCPUs recomendado).
- Memória RAM: 4 GB mínimo (8 GB recomendado).
- Disco: 10 GB livres para imagens/volumes +
- Modelos do Ollama (~5-8 GB para o llama3.1:8b-instruct-q4_K_M) + espaço para seus dados/planilhas.

2.3 Portas / Firewall

- 8000/tcp (ChromaDB) — acesso local.
- 11434/tcp (Ollama) — acesso local.
- Se já estiverem em uso, ajuste as portas publicadas no docker-compose.

2 - Requisitos

2.4 Proxy/Corporate

Se estiver atrás de proxy, exporte `HTTP_PROXY`, `HTTPS_PROXY` e `NO_PROXY` para o Docker/serviços (ou configure pelo Docker Desktop). O Ollama e o pip precisam alcançar a Internet para o primeiro download.

2.5 Sistema de arquivos & permissões

- Windows: evite caminhos com espaços/caracteres especiais e pastas “protegidas”.
- Linux: garanta que as pastas `./ollama`, `./dados` e `./out` existam e tenham leitura/escrita para o usuário que roda o Docker.

3 - Arquitetura

A arquitetura é composta por três serviços em contêiner que se falam por HTTP e compartilham volumes do host: app, chroma e ollama.

App: O App consome o arquivo .zip em ./dados/, executa a extração e a padronização das planilhas, consolida as bases (ATIVOS, FÉRIAS, DESLIGADOS, CADASTRAL e SINDICATO×VALOR), aplica as regras de elegibilidade definidas no desafio e calcula dias úteis e valores de VR/VA. Ao final, gera automaticamente o arquivo de compra no formato exigido, disponível em ./out/.

Chroma: O chroma é um banco vetorial (porta 8000) que armazena textos de políticas/validações. Enquanto o app consulta esse repositório para recuperar trechos de referência e registrar mensagens explicativas, sem interferir nos cálculos matemáticos.

Ollama: o Ollama disponibiliza um modelo de linguagem local para produzir justificativas e explicações em linguagem natural, preservando a autonomia do pipeline determinístico: caso o componente de IA não esteja disponível, o cálculo segue operacional.

4 - Visão Geral

4.1 main.py

Quando executamos a main, ela primeiro carrega as variáveis do .env e monta as configurações do sistema (endereços, portas e nome do modelo).

Em seguida, ele lê as regras e políticas que estão na pasta /app/policies e as coloca num índice de busca para que, mais adiante, seja fácil localizar trechos de regras de sindicato.

Depois, ele garante que o modelo do Ollama esteja disponível localmente; se não estiver, manda baixar e espera um instante. Com isso pronto, ele cria o “secretário” (o LLM), que serve apenas para consultar contexto das regras, sem interferir nos cálculos, e monta a linha de montagem do processo **(a sequência: *buscar regras* → *carregar bases* → *validar* → *aplicar exclusões* → *calcular* → *exportar*)**.

Antes de rodar, prepara uma o estado, onde cada etapa vai guardando o que produz: avisos, erros, bases consolidadas, resultado final e, principalmente, o caminho do arquivo a ser entregue ao fornecedor.

Por fim, ele aciona toda a sequência de etapas e imprime no terminal onde o arquivo final foi salvo.

4 - Visão Geral

4.2 agentes.py

Aqui temos um passo-a-passo automático para calcular VR/VA de cada pessoa e gerar o arquivo final de compra.

Usando:

- Funções que seguem regras fixas para carregar planilhas, validar, calcular e exportar.
- Um LLM (Ollama) para busca trechos de regras e layouts no banco de conhecimento (RAG) para consulta.

retrieval (buscar contexto)

Procura no repositório de documentos (Chroma) por regras de sindicato, validações. Junta os trechos encontrados e salva em docs_ctx (um campo do “estado”).

ingestion (carregar bases)

Usa load_bases(...) para ler as planilhas/fontes de dados. Guarda tudo em dfs (os dataframes carregados).

validation (pré-validações)

Roda run_prevalidations(...) para identificar erros e avisos. Também salva listas de erros e avisos.

exclusion (aplicar exclusões/regras)

Executa apply_exclusions(...) para tirar quem não deve entrar no cálculo (férias, afastamentos, etc.). Gera uma base consolidada e outra com excluídos dentro de dfs.

compute (calcular dias e valores)

Chama compute_days_and_values(...) para, com base nas regras e no calendário, contar dias e calcular o VR/VA de cada pessoa. E guarda o resultado final em df_final.

export (gerar o arquivo de compra)

Com export_to_layout(...), cria o layout que será enviado ao fornecedor

4 - Visão Geral

4.3 `calendars.py`

Aqui temos um filtro de calendário” que devolve quais são os dias úteis dentro de um período, considerando as regras do sindicato, a UF (estado) e o município, quando esses dados forem informados.

Ele parte de uma planilha de calendário (com uma coluna de data e, se existir, uma coluna que indica se o dia é útil ou não) e faz o seguinte: olha apenas para as datas entre início e fim;

- se você informou um sindicato, mantém só as linhas daquele sindicato; se informou uma UF, mantém só aquela UF;
- se informou um município, compara o nome sem acentos (para “São Paulo” e “Sao Paulo” serem tratados como iguais).

Se a planilha tiver a marcação “é dia útil”, ele fica só com esses dias; se, depois de todos os filtros, não sobrar nenhum dia, ele tenta novamente usando apenas o período (e, se houver, ainda respeitando a marcação de dia útil).

No final, ele entrega a lista única de datas (sem repetidas) que representam os dias úteis válidos para aquele contexto — é isso que o time usará nos cálculos de VR/VA e afins.

4 - Visão Geral

4.4 tools.py

Aqui temos o que seria um kit de ferramentas da automação. Ele começa procurando e abrindo todas as planilhas.

Aceita CSV e Excel, padroniza cabeçalhos e datas e, se não achar um calendário, cria um calendário simples de segunda a sexta só para não parar o processo.

Com as bases roda pré-validações e depois aplica as regras de exclusão: tira cargos não elegíveis (diretoria, estagiários, aprendizes), quem está em afastamento ou exterior, e cuida de desligamentos (se houve comunicado, aquele mês zera).

Também padroniza as matrículas (para não dar erro por causa de “.0” ou espaços) e completa dados quando possível (ex.: puxa a data de admissão da base cadastral).

Na parte de valores, ele tenta inferir a UF de cada pessoa (se faltar explícito) a partir de textos como sindicato ou empresa e cruza isso com a planilha de “sindicato x valor”. Nessa hora, ele entende número no formato brasileiro (com vírgula) e transforma em valor numérico certinho.

A etapa de cálculo faz assim: para cada pessoa, determina a “janela efetiva” do mês (considerando a data de admissão e, se tiver, a de desligamento) e conta os dias úteis dessa janela usando o calendário respeitando sindicato, UF e município e ignorando acentos nos nomes. Se a pessoa teve férias, abate esses dias do total.

No fim, multiplica **dias úteis × valor unitário do VR e gera o VR total**, já mostrando também a divisão **80% empresa / 20% profissional**.

4 - Visão Geral

4.5 config.py

Aqui é a parte onde estão as planilhas, qual mês você quer calcular, onde salvar a saída e como falar com os serviços de apoio (Chroma e Ollama). Em vez de espalhar essas infos pelo código, tudo fica num lugar só.

Aqui temos:

Settings: é como uma caixa organizada com todos os campos de configuração.

- `competencia`: o mês/ano de referência (ex.: 2025-05-01) para contar dias e calcular VR/VA.
- `base_ativos` / `base_ferias` / `base_desligados` / `base_cadastral` / `base_sindicato_valor`: caminhos dos arquivos de entrada (planilhas).
- `calendario_path`: caminho do calendário/feriados.
- `out_dir`: pasta onde a planilha final será salva.
- `chroma_host` / `chroma_port`: endereço do “banco de busca” (Chroma) que guarda as regras.
- `ollama_base_url` / `ollama_model`: como acessar o “secretário de contexto” (Ollama) e qual modelo usar.
- `load_settings()`: monta essa “caixinha” lendo variáveis de ambiente (as chaves do seu `.env` ou do Docker).

Se alguma variável não existir, usa um valor padrão:

- `COMPETENCIA` → padrão: 2025-05-01 (formato ISO: AAAA-MM-DD).
- `ATIVOS`, `FERIAS`, `DESLIGADOS`, `CADASTRAL`, `SINDICATO_VALOR` → padrão: arquivos dentro de `/dados/`.
- `CALENDARIO` → padrão: `/dados/calendario_feriados.parquet`.
- `OUT_DIR` → padrão: `/out`.
- `CHROMA_HOST/CHROMA_PORT` → padrão: `chroma:8000`.
- `OLLAMA_BASE_URL` → padrão: `http://ollama:11434`.
- `OLLAMA_MODEL` → padrão: `llama3.1:8b-instruct-q4_K_M`.

4 - Visão Geral

4.6 data_schemas.py

Aqui temos um ajustador das planilhas antes de o resto do sistema trabalhar com elas. A primeira função, `normalize_headers`, padroniza os nomes das colunas: tira espaços no começo/fim, deixa tudo minúsculo, troca espaços por underlines e remove acentos.

Na prática, cabeçalhos diferentes viram algo uniforme. Já a `parse_dates` dá uma passada nas colunas e, sempre que achar no nome a palavra “data” ou terminar com “_inicio” ou “_fim”, tenta converter o conteúdo para formato de data de verdade.

Se tiver algo que não é data ali, ele vira vazio e assim o resultado é uma planilha limpa e previsível: cabeçalhos padronizados e campos de data prontos para contar dias, filtrar períodos e cruzar informações sem dor de cabeça.

4 - Visão Geral

4.7 export_layout.py

Aqui geramos a planilha final com colunas formatadas se tiver o `xlsxwriter` instalado, caso não, salva sem formatação (mas funciona do mesmo jeito)

4 - Visão Geral

4.8 validations.py

Aqui fazemos checagens rápidas antes do cálculo para pegar problemas comuns logo de cara.

- Confere a tabela de "sindicato x valor": se existir a coluna valor_vr e ela tiver valores faltando, gera erros do tipo
- VIG001 — "Sindicato X sem valor vigente." (ou seja, não dá pra calcular VR de quem está nesse sindicato sem o valor unitário).
- Confere datas de férias: se houver ferias_inicio e ferias_fim, marca avisos quando a data de fim vier antes da data de início (inconsistência de datas). Ex.: DAT001 — "Matr 123: ferias_fim < ferias_inicio".
- Saída: devolve dois DataFrames, um de erros e outro de avisos, para irem como abas no Excel final e/ou bloquear o processo se algo crítico aparecer.

4 - Visão Geral

4.9 vectorstore_setup.py

Aqui fazemos a comunicação com o Chroma e colocamos lá dentro os documentos de políticas/regras. Mais tarde, quando a pipeline “perguntar” por regras (RAG), o sistema busca nessa coleção e encontra os trechos certos rapidamente.

5 - Ambiente, Dependências e Deploy

5.1 Dockerfile

- **Base leve:** FROM python:3.11-slim-bookworm → começa de uma imagem Python mínima (Debian 12 “bookworm”), deixando o container pequeno.
- **Ajustes do Python/pip:** PYTHONDONTWRITEBYTECODE=1 (não cria .pyc), PYTHONUNBUFFERED=1 (logs aparecem na hora), PIP_NO_CACHE_DIR=1 (pip não guarda cache → imagem menor).
- **Diretório de trabalho:** WORKDIR /app → tudo passa a acontecer dentro de /app.
- **Dependências primeiro:** copia requirements.txt e roda pip install → melhora o cache de build (só refaz quando mudar as deps).
- **Código da app:** COPY . . → coloca seu projeto dentro do container.
- **Usuário sem root (permite rodar com menos privilégio):** Define ARG HOST_UID/GID (padrão 1000) e cria appuser/appgrp com esses IDs. Isso evita dor de cabeça com permissões ao montar volumes.

5 - Ambiente, Dependências e Deploy

5.2 requirements

Orquestração de agentes / RAG

- langgraph, langchain, langchain-community → framework dos “agentes” e do fluxo (pipeline) que coordena as etapas e o uso do LLM/RAG.

Banco de busca (vetorial)

- chromadb → onde você indexa e busca os textos de regras/políticas para o RAG.

Manipulação de dados e arquivos

- pandas → coração do tratamento de planilhas e tabelas.
- pyarrow → suporte a Parquet/Arrow e acelera IO com o pandas.
- openpyxl → ler/escrever .xlsx (Excel moderno).
- XlsxWriter → escrever .xlsx com formatação bonitinha (cabeçalho, moeda etc.).
- xlrd → ler .xls (Excel antigo).
- xlwt → escrever .xls (Excel antigo).

Configuração / validação

- python-dotenv → carrega variáveis do .env (caminhos, portas, modelo do LLM).
- pydantic → valida/organiza estruturas de dados (ex.: Settings).

Utilidades

- rapidfuzz → comparações “fuzzy” de textos (ajuda a casar nomes, UF, etc.).
- requests → chamadas HTTP simples (ex.: puxar modelo no Ollama).

5 - Ambiente, Dependências e Deploy

5.3 Dockercompose - Linux

Aqui subimos:

- **ollama** → o “cérebro” (servidor do modelo). Abre a porta 11434, guarda os modelos em ./ollama (persistente) e tem healthcheck pra confirmar que está no ar.
- **chroma** → o “banco de busca de textos” (RAG). Abre a porta 8000, salva os dados no volume chroma_data, e também tem healthcheck.
- **app** → a aplicação carrega bases, valida, calcula e exporta. Só inicia depois do chroma estar saudável e do ollama ter iniciado.

Portas:

- **11434:11434 (ollama) e 8000:8000 (chroma)** ficam acessíveis na sua máquina.

Ambiente e permissões:

- **app** roda como o seu usuário (UID/GID) para não dar problema de permissão nos arquivos.
- Variáveis como HOME, caches do HF/Transformers e telemetry off já estão configuradas.
- **ollama**, OLLAMA_NOPRUNE=1 evita apagar modelos automaticamente.
- **Flags** :Z e security_opt: label=disable ajudam no SELinux (ex.: Fedora) a não travar volumes.

5 - Ambiente, Dependências e Deploy

5.4 docker-compose.windows - Windows

Aqui subimos:

- **ollama** → o “cérebro” (servidor do modelo). Abre a porta 11434, guarda os modelos em ./ollama (persistente) e tem healthcheck pra confirmar que está no ar.
- **chroma** → o “banco de busca de textos” (RAG). Abre a porta 8000, salva os dados no volume chroma_data, e também tem healthcheck.
- **app** → a aplicação carrega bases, valida, calcula e exporta. Só inicia depois do chroma estar saudável e do ollama ter iniciado.

Portas:

- **11434:11434 (ollama) e 8000:8000 (chroma)** ficam acessíveis na sua máquina.

Ambiente e permissões:

- **Sem user: na app:** no Windows a app roda como root dentro do container (isso evita rolo de permissões que no Linux você resolveu com UID/GID).
- **Sem :Z e sem security_opt:** no Windows não tem SELinux, então não precisa desses ajustes.

6 - Passo a Passo

6.1 Linux

Dentro da pasta VR_CH, no seu terminal coloque os seguintes comandos:

```
export UID=$(id -u) GID=$(id -g)
docker compose up -d --build
```

O que fazem?

Exportar UID/GID atuais

Cria (se não existirem) e sobe os serviços definidos no docker-compose.yml (rede, volumes, containers, etc.) e roda em segundo plano

Se ainda assim der “permission denied”, faça:

```
sudo chcon -Rt svirt_sandbox_file_t app dados out ollama
```

Se deseja conferir os status, faça:

```
docker ps
docker logs -f vrva-agent      # logs do app
docker logs ollama            # servidor LLM
docker logs chroma            # banco vetorial
```

6 - Passo a Passo

6.1 Linux

Dentro da pasta VR_CH, no seu terminal coloque o comando:

```
docker compose -f docker-compose.windows.yml up -d --build
```

O que fazem?

Exportar UID/GID atuais

Cria (se não existirem) e sobe os serviços definidos no docker-compose.yml (rede, volumes, containers, etc.) e roda em segundo plano

Se deseja conferir os status, faça:

```
docker ps  
docker logs -f vrva-agent      # logs do seu app
```

7 - Funcionamento na prática

Na pasta dados teremos o zip com os excels disponibilizados para o desafio, e uma pasta que os guardará após serem descompactados automaticamente:

```
galvao@fedora:~/VR_CH/dados/UNPACKED

galvao@fedora:~/VR_CH/dados$ ls
'Desafio 4 - Dados.zip'  UNPACKED
galvao@fedora:~/VR_CH/dados$ cd UNPACKED
galvao@fedora:~/VR_CH/dados/UNPACKED$ ls
galvao@fedora:~/VR_CH/dados/UNPACKED$
```

Na pasta out ficará o excel com a tabela final:

```
galvao@fedora:~/VR_CH/out$ ls
galvao@fedora:~/VR_CH/out$
```

Após aplicar os comandos referentes ao seu sistema operacional e todas dependências forem instaladas devidamente e tudo for iniciado como esperado, poderemos vizualizar nossa planilha final:

```
galvao@fedora:~/VR_CH$ export UID=$(id -u) GID=$(id -g)

docker compose up -d --build
bash: UID: a variável permite somente leitura
[+] Building 1.6s (13/13) FINISHED
--> [internal] load local bake definitions 0.0s
--> reading from stdin SBTB 0.0s
--> [internal] load build definition from Dockerfile 0.0s
--> == transferring dockerfile: 40B 0.0s
--> [internal] load metadata for docker.io/library/python:3.11-slim-bookworm 1.4s
--> [internal] load .dockerignore 0.0s
--> == transferring context: 2B 0.0s
--> [1/6] FROM docker.io/library/python:3.11-slim-bookworm@sha256:8261f4d 0.0s
--> [internal] load build context 0.0s
--> == transferring context: 3.10kB 0.0s
--> CACHED [2/6] WORKDIR /app 0.0s
--> CACHED [3/6] COPY requirements.txt ./ 0.0s
--> CACHED [4/6] RUN pip install --no-cache-dir -r requirements.txt 0.0s
--> CACHED [5/6] COPY . 0.0s
--> CACHED [6/6] RUN groupadd -g 1000 appgrp || true && useradd -u 10 0.0s
--> exporting to image 0.0s
--> == exporting layers 0.0s
--> == writing image sha256:c4e1a3f77308cf4d0fa0262ac0d03c14d0ba37a2210 0.0s
--> == pushing to docker.io/library/vr_ch-app 0.0s
--> repulishing provenance for metadata file 0.0s
[+] Running 4/4
✔ vr_ch-app Built 0.0s
✔ Container chroma Healthy 10.7s
✔ Container ollama Started 0.2s
✔ Container vrva-agent Started 0.1s
galvao@fedora:~/VR_CH$
```

Na pasta UNPACKED vemos todos arquivos descompactados:

```
galvao@fedora:~/VR_CH/out$ ls
'VR Mensal 05.2025.xlsx'
galvao@fedora:~/VR_CH/out$ cd ..
galvao@fedora:~/VR_CH$ ls
app      docker-compose.windows.yml  ollama      out
dados    docker-compose.yml          ollama.bak  README.md
galvao@fedora:~/VR_CH$ cd dados
galvao@fedora:~/VR_CH/dados$ ls
'Desafio 4 - Dados.zip'  UNPACKED
galvao@fedora:~/VR_CH/dados$ cd UNPACKED
galvao@fedora:~/VR_CH/dados/UNPACKED$ ls
'ADMISSÃO ABRIL.xlsx'  'Base dias uteis.xlsx'  EXTERIOR.xlsx
AFASTAMENTOS.xlsx     'Base sindicato x valor.xlsx'  FÉRIAS.xlsx
APRENDIZ.xlsx         DESLIGADOS.xlsx        'VR MENSAL 05.2025.xlsx'
ATIVOS.xlsx           ESTAGIO.xlsx
galvao@fedora:~/VR_CH/dados/UNPACKED$
```

7 - Funcionamento na prática

Na pasta out teremos nosso Excel final:

```
galvao@fedora:~/VR_CH/out
galvao@fedora:~/VR_CH/out$ ls
'VR Mensal 05.2025.xlsx'
galvao@fedora:~/VR_CH/out$
```

VR Mensal 05.2025.xlsx — LibreOffice Calc

Arquivo Editar Exibir Inserir Formatar Estilos Planilha Dados Ferramentas Janela Ajuda

Calibri 11 pt N I S A B % 0.0 0.00 0.0

C1 fx Σ = cpf

	A	B	C	D	E	F	G	H	I	J	K
1	matricula	nome	cpf	sindicato	lotacao_uf	competencia	dias uteis_mes	vr_unitario	vr_total	custo_empresa	custo_profissior
2	34941			SINDPD SP - SIND.TRAB.EM PROC.DADOS E EMPR.EMPRESAS	SP	2025-05	22	R\$ 37,50	R\$ 825,00	R\$ 660,00	R\$ 165,00
3	24401			SINDPPD RS - SINDICATO DOS TRAB. EM PROC. DE DADOS RI	RS	2025-05	22	R\$ 35,00	R\$ 770,00	R\$ 616,00	R\$ 154,00
4	32104			SINDPPD RS - SINDICATO DOS TRAB. EM PROC. DE DADOS RI	RS	2025-05	12	R\$ 35,00	R\$ 420,00	R\$ 336,00	R\$ 84,00
5	35254			SINDPPD RS - SINDICATO DOS TRAB. EM PROC. DE DADOS RI	RS	2025-05	22	R\$ 35,00	R\$ 770,00	R\$ 616,00	R\$ 154,00
6	31731			SINDPPD RS - SINDICATO DOS TRAB. EM PROC. DE DADOS RI	RS	2025-05	22	R\$ 35,00	R\$ 770,00	R\$ 616,00	R\$ 154,00
7	31009			SINDPPD RS - SINDICATO DOS TRAB. EM PROC. DE DADOS RI	RS	2025-05	22	R\$ 35,00	R\$ 770,00	R\$ 616,00	R\$ 154,00
8	28979			SINDPPD RS - SINDICATO DOS TRAB. EM PROC. DE DADOS RI	RS	2025-05	22	R\$ 35,00	R\$ 770,00	R\$ 616,00	R\$ 154,00

8 - Conclusão

A solução que conseguimos propôr para automatizar o cálculo mensal de VR/VA se baseou em regras determinísticas e dados consolidados, o que vem a reduzir o esforço manual e a chance de erro.

Nosso fluxo integra múltiplas bases (ativos, férias, desligados, cadastral e valores por sindicato/UF), aplica exclusões previstas (férias, afastamentos, aprendiz/estágio/diretoria).

O resultado é um arquivo Excel padronizado. Fizemos o uso de RAG (Chroma + Ollama) que fica restrito à consulta de contexto (regras/layout), preservando a confiabilidade dos cálculos, que são 100% determinísticos.

A containerização (Docker/Compose) garante reprodutibilidade e portabilidade entre Linux, ambiente onde foi desenvolvido e Windows.

9. Referências Bibliográficas

CHROMA. Chroma (vector database). Disponível em: [CHROMA. Chroma \(vector database\). Disponível em: https://www.trychroma.com/](https://www.trychroma.com/).

OLLAMA. Ollama (run LLMs locally). Disponível em: [CHROMA. Chroma \(vector database\). Disponível em: https://www.trychroma.com/](https://www.trychroma.com/).

Docker Compose — Documentação Docker. Disponível em: [CHROMA. Chroma \(vector database\). Disponível em: https://www.trychroma.com/](https://www.trychroma.com/).

STACK OVERFLOW. What is :z flag in docker containers volumes-from option? Disponível em: [CHROMA. Chroma \(vector database\). Disponível em: https://www.trychroma.com/](https://www.trychroma.com/).

PYPI (Python Package Index). XlsxWriter. Disponível em: [CHROMA. Chroma \(vector database\). Disponível em: https://www.trychroma.com/](https://www.trychroma.com/).

