# VIPP Pet Center — Sistema de Gestão de Planos de Saúde (Projeto completo)

Documento entregue: código de exemplo (backend + templates), instruções passo-a-passo para leigos, e scripts/arquivos necessários para deploy no **Render** com **Postgres**. Copie/cole os arquivos no seu repositório e siga o passo-a-passo da sessão “Passo a Passo (leigo)”.

## O que este pacote contém (visão geral)

1. Arquitetura simples: **Flask** (API + HTML) + **Postgres** (produção) / SQLite (desenvolvimento) + **Gunicorn** para produção.
2. Modelos (SQLAlchemy): Tutor, Pet, Plan, PlanBenefit, Enrollment, Installment, Payment.
3. Upload de planilhas Excel (Planos De Saúde VIPP programação.xlsx) para importar automaticamente os planos e benefícios (cada sheet representa um plano). O import adapta colunas comuns como direitos, Limites de uso, Co Participação, Carência, etc.
4. CRUD básico via forms HTML (leigo-friendly) para: cadastrar tutor, cadastrar pet, criar adesão (enrollment), criar/visualizar parcelas, marcar parcelas como pagas.
5. Página de **busca por CPF** do tutor: retorna dados do tutor, pets associados, plano(s) contratados, e situação das parcelas (pagas / vencidas / pendentes).
6. Arquivos para deploy: requirements.txt, Procfile, .gitignore e instruções para conexão com o Postgres do Render.

## Estrutura de arquivos (como você deve organizar no repositório)

vipp-plan-system/  
├── app.py # Aplicação Flask principal (contém rotas e modelos)  
├── templates/  
│ ├── base.html  
│ ├── index.html # busca por CPF  
│ ├── tutor\_detail.html # exibe tutor, pets, planos e parcelas  
│ ├── admin\_upload.html # formulário para enviar o Excel  
│ └── forms.html # formulário para criação rápida (tutor/pet/adesao)  
├── static/ # css/js se quiser personalizar  
├── requirements.txt  
├── Procfile  
├── README.md # este texto resumido também  
└── .gitignore

## Código principal (coloque em app.py)

**IMPORTANTE:** este app.py é intencionalmente simples e auto-contido para facilitar o deploy para um leigo. Em produção você pode separá-lo em módulos, adicionar autenticação, validação, testes e backups.

# app.py  
import os  
from datetime import datetime, timedelta  
from decimal import Decimal  
from flask import Flask, render\_template, request, redirect, url\_for, flash, jsonify  
from flask\_sqlalchemy import SQLAlchemy  
from sqlalchemy.orm import joinedload  
import pandas as pd  
  
app = Flask(\_\_name\_\_)  
app.secret\_key = os.environ.get('SECRET\_KEY', 'troque-esta-chave')  
  
# DATABASE: usa DATABASE\_URL se definido (Render Postgres), caso contrário usa sqlite local  
DATABASE\_URL = os.environ.get('DATABASE\_URL')  
if DATABASE\_URL:  
 app.config['SQLALCHEMY\_DATABASE\_URI'] = DATABASE\_URL  
else:  
 app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///vipp\_plans.db'  
  
app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False  
db = SQLAlchemy(app)  
  
# ----------------------------- MODELS ---------------------------------  
class Tutor(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 nome = db.Column(db.String(200), nullable=False)  
 cpf = db.Column(db.String(20), unique=True, nullable=False)  
 email = db.Column(db.String(200))  
 telefone = db.Column(db.String(50))  
 endereco = db.Column(db.String(400))  
 created\_at = db.Column(db.DateTime, default=datetime.utcnow)  
 pets = db.relationship('Pet', backref='tutor', lazy=True)  
 enrollments = db.relationship('Enrollment', backref='tutor', lazy=True)  
  
class Pet(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 tutor\_id = db.Column(db.Integer, db.ForeignKey('tutor.id'), nullable=False)  
 nome = db.Column(db.String(200), nullable=False)  
 especie = db.Column(db.String(50))  
 raca = db.Column(db.String(100))  
 nascimento = db.Column(db.Date)  
 microchip = db.Column(db.String(100))  
  
class Plan(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 nome = db.Column(db.String(200), unique=True, nullable=False)  
 mensalidade = db.Column(db.Numeric(10,2), default=0)  
 taxa\_adm = db.Column(db.Numeric(10,2), default=0)  
 descricao = db.Column(db.Text)  
 benefits = db.relationship('PlanBenefit', backref='plan', lazy=True)  
  
class PlanBenefit(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 plan\_id = db.Column(db.Integer, db.ForeignKey('plan.id'), nullable=False)  
 direito = db.Column(db.String(300))  
 limite = db.Column(db.String(100))  
 condicao = db.Column(db.String(100))  
 coparticipacao = db.Column(db.String(100))  
 reembolso = db.Column(db.String(100))  
 carencia = db.Column(db.String(100))  
  
class Enrollment(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 tutor\_id = db.Column(db.Integer, db.ForeignKey('tutor.id'), nullable=False)  
 plan\_id = db.Column(db.Integer, db.ForeignKey('plan.id'), nullable=False)  
 data\_inicio = db.Column(db.Date, default=datetime.utcnow)  
 status = db.Column(db.String(50), default='ativo')  
 forma\_pagamento = db.Column(db.String(50))  
 valor\_plano = db.Column(db.Numeric(10,2), default=0)  
 parcelas = db.Column(db.Integer, default=1)  
 taxa\_administracao = db.Column(db.Numeric(10,2), default=0)  
 created\_at = db.Column(db.DateTime, default=datetime.utcnow)  
  
 installments = db.relationship('Installment', backref='enrollment', lazy=True, cascade='all, delete-orphan')  
  
class Installment(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 enrollment\_id = db.Column(db.Integer, db.ForeignKey('enrollment.id'), nullable=False)  
 numero = db.Column(db.Integer)  
 valor = db.Column(db.Numeric(10,2))  
 vencimento = db.Column(db.Date)  
 pago = db.Column(db.Boolean, default=False)  
 pago\_em = db.Column(db.DateTime, nullable=True)  
  
class Payment(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 installment\_id = db.Column(db.Integer, db.ForeignKey('installment.id'), nullable=False)  
 valor = db.Column(db.Numeric(10,2))  
 data\_pagamento = db.Column(db.DateTime, default=datetime.utcnow)  
 metodo = db.Column(db.String(100))  
  
# --------------------------- UTILIDADES -------------------------------  
  
def create\_installments(enrollment, primeira\_vencimento=None):  
 """  
 Gera as parcelas para uma enrollment.  
 primeira\_vencimento: datetime.date (se None, usa hoje + 30 dias)  
 """  
 if primeira\_vencimento is None:  
 primeira\_vencimento = datetime.utcnow().date() + timedelta(days=30)  
 total = float(enrollment.valor\_plano) \* enrollment.parcelas  
 # Se a taxa de admin for cobrada separadamente, podemos somar  
 # aqui, mas o modelo já guarda taxa\_administracao.  
 valor\_base = float(enrollment.valor\_plano)  
 parcelas = int(enrollment.parcelas)  
 # Distribui o valor base entre parcelas (simples)  
 for n in range(1, parcelas + 1):  
 venc = primeira\_vencimento + timedelta(days=30\*(n-1))  
 inst = Installment(enrollment\_id=enrollment.id, numero=n, valor=round(valor\_base,2), vencimento=venc)  
 db.session.add(inst)  
 db.session.commit()  
  
# ---------------------------- ROTAS ----------------------------------  
  
@app.route('/')  
def index():  
 return render\_template('index.html')  
  
# Buscar por CPF e exibir detalhes  
@app.route('/search')  
def search():  
 cpf = request.args.get('cpf')  
 if not cpf:  
 flash('Informe o CPF para busca', 'warning')  
 return redirect(url\_for('index'))  
 tutor = Tutor.query.options(joinedload(Tutor.pets), joinedload(Tutor.enrollments)).filter\_by(cpf=cpf).first()  
 if not tutor:  
 flash('Tutor não encontrado', 'danger')  
 return redirect(url\_for('index'))  
 # Carrega detalhes das enrollments e installments  
 for e in tutor.enrollments:  
 e.plan = Plan.query.get(e.plan\_id)  
 e.installments = Installment.query.filter\_by(enrollment\_id=e.id).all()  
 return render\_template('tutor\_detail.html', tutor=tutor)  
  
# Form simples para criar tutor  
@app.route('/tutor/new', methods=['GET','POST'])  
def new\_tutor():  
 if request.method == 'POST':  
 nome = request.form['nome']  
 cpf = request.form['cpf']  
 email = request.form.get('email')  
 telefone = request.form.get('telefone')  
 endereco = request.form.get('endereco')  
 if Tutor.query.filter\_by(cpf=cpf).first():  
 flash('CPF já cadastrado', 'danger')  
 return redirect(url\_for('new\_tutor'))  
 t = Tutor(nome=nome, cpf=cpf, email=email, telefone=telefone, endereco=endereco)  
 db.session.add(t)  
 db.session.commit()  
 flash('Tutor criado com sucesso', 'success')  
 return redirect(url\_for('index'))  
 return render\_template('forms.html')  
  
# Adicionar pet  
@app.route('/pet/new', methods=['POST'])  
def new\_pet():  
 tutor\_cpf = request.form['tutor\_cpf']  
 tutor = Tutor.query.filter\_by(cpf=tutor\_cpf).first()  
 if not tutor:  
 flash('Tutor não encontrado para cadastrar pet', 'danger')  
 return redirect(url\_for('index'))  
 pet = Pet(tutor\_id=tutor.id, nome=request.form['nome'], especie=request.form.get('especie'), raca=request.form.get('raca'))  
 db.session.add(pet)  
 db.session.commit()  
 flash('Pet cadastrado', 'success')  
 return redirect(url\_for('search', cpf=tutor\_cpf))  
  
# Criar enrollment (adesão)  
@app.route('/enrollment/new', methods=['POST'])  
def new\_enrollment():  
 cpf = request.form['cpf']  
 plan\_id = int(request.form['plan\_id'])  
 forma\_pag = request.form.get('forma\_pagamento', 'boleto')  
 parcelas = int(request.form.get('parcelas', 1))  
 valor = float(request.form.get('valor', 0))  
 taxa = float(request.form.get('taxa', 0))  
 tutor = Tutor.query.filter\_by(cpf=cpf).first()  
 if not tutor:  
 flash('Tutor não encontrado', 'danger')  
 return redirect(url\_for('index'))  
 enroll = Enrollment(tutor\_id=tutor.id, plan\_id=plan\_id, forma\_pagamento=forma\_pag, valor\_plano=valor, parcelas=parcelas, taxa\_administracao=taxa)  
 db.session.add(enroll)  
 db.session.commit()  
 create\_installments(enroll)  
 flash('Adesão criada', 'success')  
 return redirect(url\_for('search', cpf=cpf))  
  
# Marcar parcela como paga  
@app.route('/installment/pay/<int:inst\_id>', methods=['POST'])  
def pay\_installment(inst\_id):  
 inst = Installment.query.get\_or\_404(inst\_id)  
 inst.pago = True  
 inst.pago\_em = datetime.utcnow()  
 db.session.commit()  
 flash('Parcela marcada como paga', 'success')  
 return redirect(url\_for('search', cpf=inst.enrollment.tutor.cpf))  
  
# Admin: upload Excel para importar planos  
@app.route('/admin/import', methods=['GET','POST'])  
def import\_plans():  
 if request.method == 'POST':  
 file = request.files.get('file')  
 if not file:  
 flash('Envie um arquivo .xlsx', 'danger')  
 return redirect(url\_for('import\_plans'))  
 # Lê planilha com pandas  
 xls = pd.ExcelFile(file)  
 for sheet in xls.sheet\_names:  
 df = pd.read\_excel(xls, sheet\_name=sheet)  
 # Cria ou atualiza Plan  
 plan\_name = sheet.strip()  
 mensalidade = request.form.get('mensalidade', 0) or 0  
 plan = Plan.query.filter\_by(nome=plan\_name).first()  
 if not plan:  
 plan = Plan(nome=plan\_name, mensalidade=float(mensalidade))  
 db.session.add(plan)  
 db.session.commit()  
 # Tenta mapear colunas comuns  
 cols = [c.lower() for c in df.columns]  
 # Vamos tentar ler cada linha como benefit  
 for \_, row in df.iterrows():  
 direito = None  
 limite = None  
 condicao = None  
 copart = None  
 reembolso = None  
 carencia = None  
 # aproximação por nomes  
 for k in df.columns:  
 lk = k.lower()  
 v = row[k]  
 if 'direit' in lk or 'direitos' in lk or 'direto' in lk:  
 direito = str(v)  
 if 'limite' in lk:  
 limite = str(v)  
 if 'condi' in lk or 'condição' in lk or 'condIção' in lk:  
 condicao = str(v)  
 if 'co' in lk or 'copart' in lk:  
 copart = str(v)  
 if 'reembol' in lk:  
 reembolso = str(v)  
 if 'carên' in lk or 'carencia' in lk or 'carência' in lk:  
 carencia = str(v)  
 if direito and (not pd.isna(direito)):  
 benefit = PlanBenefit(plan\_id=plan.id, direito=direito, limite=str(limite), condicao=str(condicao), coparticipacao=str(copart), reembolso=str(reembolso), carencia=str(carencia))  
 db.session.add(benefit)  
 db.session.commit()  
 flash('Importação concluída', 'success')  
 return redirect(url\_for('index'))  
 return render\_template('admin\_upload.html')  
  
# Rota para checar/vigiar parcelas vencidas (pode ser chamada manualmente ou por job)  
@app.route('/cron/check\_overdue')  
def check\_overdue():  
 today = datetime.utcnow().date()  
 overdue = Installment.query.filter(Installment.vencimento < today, Installment.pago == False).all()  
 return jsonify({'overdue\_count': len(overdue)})  
  
# Inicialização do DB  
if \_\_name\_\_ == '\_\_main\_\_':  
 db.create\_all()  
 app.run(debug=True, host='0.0.0.0')

## Templates (exemplos simples)

### templates/base.html

<!doctype html>  
<html lang="pt-br">  
 <head>  
 <meta charset="utf-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <title>VIPP Plano - Sistema</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">  
 </head>  
 <body class="p-4">  
 <div class="container">  
 <h1>VIPP Pet Center — Sistema de Planos</h1>  
 <hr/>  
 {% with messages = get\_flashed\_messages(with\_categories=true) %}  
 {% if messages %}  
 {% for cat,msg in messages %}  
 <div class="alert alert-{{cat}}">{{msg}}</div>  
 {% endfor %}  
 {% endif %}  
 {% endwith %}  
 {% block body %}{% endblock %}  
 </div>  
 </body>  
</html>

### templates/index.html

{% extends 'base.html' %}  
{% block body %}  
 <form method="get" action="/search" class="row g-2">  
 <div class="col-md-6">  
 <input class="form-control" name="cpf" placeholder="Digite o CPF do tutor (somente números)" required>  
 </div>  
 <div class="col-auto">  
 <button class="btn btn-primary" type="submit">Buscar</button>  
 </div>  
 </form>  
 <hr/>  
 <a class="btn btn-secondary" href="/admin/import">Importar planilha (Excel)</a>  
 <a class="btn btn-outline-primary" href="/tutor/new">Novo Tutor / Formulários</a>  
{% endblock %}

### templates/tutor\_detail.html

{% extends 'base.html' %}  
{% block body %}  
 <h2>Tutor: {{ tutor.nome }} (CPF: {{ tutor.cpf }})</h2>  
 <p>Email: {{ tutor.email }} — Tel: {{ tutor.telefone }}</p>  
  
 <h3>Pets</h3>  
 <ul>  
 {% for pet in tutor.pets %}  
 <li>{{pet.nome}} — {{pet.especie or ''}} {{pet.raca or ''}} (microchip: {{pet.microchip or '-' }})</li>  
 {% else %}  
 <p>Sem pets cadastrados</p>  
 {% endfor %}  
 </ul>  
  
 <h3>Planos / Adesões</h3>  
 {% for e in tutor.enrollments %}  
 <div class="card mb-3 p-2">  
 <strong>Plano:</strong> {{ e.plan.nome if e.plan else e.plan\_id }} <br/>  
 <strong>Valor:</strong> R$ {{ e.valor\_plano }} — <strong>Parcelas:</strong> {{ e.parcelas }} — <strong>Forma:</strong> {{ e.forma\_pagamento }}  
 <h5>Parcelas</h5>  
 <table class="table">  
 <thead><tr><th>#</th><th>Vencimento</th><th>Valor</th><th>Status</th><th>Ação</th></tr></thead>  
 <tbody>  
 {% for p in e.installments %}  
 <tr>  
 <td>{{p.numero}}</td>  
 <td>{{p.vencimento}}</td>  
 <td>R$ {{p.valor}}</td>  
 <td>{% if p.pago %}Pago em {{p.pago\_em}}{% elif p.vencimento < (now().date()) %}<span class="text-danger">Vencida</span>{% else %}Pendente{% endif %}</td>  
 <td>  
 {% if not p.pago %}  
 <form method="post" action="/installment/pay/{{p.id}}">  
 <button class="btn btn-sm btn-success">Marcar como paga</button>  
 </form>  
 {% endif %}  
 </td>  
 </tr>  
 {% endfor %}  
 </tbody>  
 </table>  
 </div>  
 {% endfor %}  
{% endblock %}

### templates/admin\_upload.html

{% extends 'base.html' %}  
{% block body %}  
 <h3>Importar planilha de Planos</h3>  
 <p>Faça upload do arquivo Excel com as abas correspondentes a cada plano. O import tentará mapear as colunas automaticamente.</p>  
 <form action="" method="post" enctype="multipart/form-data">  
 <div class="mb-3">  
 <input type="file" name="file" accept=".xlsx" required>  
 </div>  
 <div class="mb-3">  
 <label>Valor mensal padrão (opcional).</label>  
 <input type="text" name="mensalidade" class="form-control" placeholder="0.00">  
 </div>  
 <button class="btn btn-primary">Enviar</button>  
 </form>  
{% endblock %}

### templates/forms.html

{% extends 'base.html' %}  
{% block body %}  
 <h3>Novo Tutor</h3>  
 <form method="post" action="/tutor/new">  
 <input type="text" name="nome" placeholder="Nome" class="form-control mb-2" required>  
 <input type="text" name="cpf" placeholder="CPF" class="form-control mb-2" required>  
 <input type="email" name="email" placeholder="Email" class="form-control mb-2">  
 <input type="text" name="telefone" placeholder="Telefone" class="form-control mb-2">  
 <input type="text" name="endereco" placeholder="Endereço" class="form-control mb-2">  
 <button class="btn btn-success">Criar Tutor</button>  
 </form>  
 <hr/>  
 <h3>Cadastrar Pet</h3>  
 <form method="post" action="/pet/new">  
 <input type="text" name="tutor\_cpf" placeholder="CPF do tutor" class="form-control mb-2" required>  
 <input type="text" name="nome" placeholder="Nome do Pet" class="form-control mb-2" required>  
 <input type="text" name="especie" placeholder="Espécie" class="form-control mb-2">  
 <input type="text" name="raca" placeholder="Raça" class="form-control mb-2">  
 <button class="btn btn-primary">Cadastrar Pet</button>  
 </form>  
 <hr/>  
 <h3>Criar Adesão (Enrollment)</h3>  
 <form method="post" action="/enrollment/new">  
 <input type="text" name="cpf" placeholder="CPF do tutor" class="form-control mb-2" required>  
 <input type="number" name="plan\_id" placeholder="ID do plano (veja no banco)" class="form-control mb-2" required>  
 <input type="text" name="valor" placeholder="Valor do plano (ex: 120.00)" class="form-control mb-2" required>  
 <input type="number" name="parcelas" placeholder="Parcelas" class="form-control mb-2" value="1" required>  
 <input type="text" name="forma\_pagamento" placeholder="Forma (boleto/cartao)" class="form-control mb-2">  
 <input type="text" name="taxa" placeholder="Taxa administrativa (ex: 10.00)" class="form-control mb-2">  
 <button class="btn btn-warning">Criar Adesão</button>  
 </form>  
{% endblock %}

## Arquivos auxiliares

requirements.txt

Flask>=2.2  
Flask-SQLAlchemy  
pandas  
openpyxl  
gunicorn  
python-dotenv  
psycopg2-binary

Procfile

web: gunicorn app:app

.gitignore

\_\_pycache\_\_/  
\*.pyc  
instance/  
.vscode/  
.env  
vipp\_plans.db

## Passo a passo (leigo) — do início ao fim

### Pré-requisitos (o mínimo que você precisa)

* Conta no GitHub (https://github.com) — para armazenar o código e conectar ao Render.
* Conta no Render (https://dashboard.render.com) — para hospedar a aplicação e o banco Postgres.
* Familiaridade mínima com o Windows: abrir Prompt/PowerShell, instalar Python se necessário.
* Python 3.10+ instalado no PC (https://www.python.org/downloads/).

### 1) Preparar o repositório no GitHub

1. Crie um repositório novo no GitHub chamado vipp-plan-system (público ou privado).
2. No seu PC, crie uma pasta vipp-plan-system e dentro cole todos os arquivos descritos neste documento (copie/cole os códigos nas respetivas ficheiros).
3. Abra PowerShell/Terminal nessa pasta e rode:

* git init  
  git add .  
  git commit -m "Inicial: VIPP plano de saude"  
  git branch -M main  
  git remote add origin https://github.com/SEUUSUARIO/vipp-plan-system.git  
  git push -u origin main

Se preferir, use o GitHub Desktop — interface gráfica.

### 2) Testar localmente (no seu computador)

1. Crie um ambiente virtual:

* python -m venv venv  
  venv\Scripts\activate # Windows  
  pip install -r requirements.txt

1. Rode a aplicação localmente:

* python app.py

1. Abra o navegador e acesse http://localhost:5000 — deverá ver a tela de busca.
2. Importe a planilha (menu Importar planilha) para criar planos automaticamente: selecione Planos De Saúde VIPP programação.xlsx e envie.

### 3) Preparar o Render e banco Postgres

1. Entre no Render (crie conta se necessário).
2. No Dashboard clique em **New -> PostgreSQL** e crie um banco (nome: vipp-db por exemplo). Guarde o **Connection String** que o Render mostra — você precisará dele.
3. Ainda no Render, clique em **New -> Web Service** e conecte seu repositório GitHub vipp-plan-system.
   * Branch: main
   * Build Command: (normalmente vazio — Render detecta requirements.txt e fará o pip install). Se precisar, deixe: pip install -r requirements.txt.
   * Start Command: gunicorn app:app
4. Quando criar o Web Service, vá na página do serviço → **Environment** → adicione uma variável chamada DATABASE\_URL com a **Connection String** do banco Postgres criado (copie/cole). Também adicione SECRET\_KEY com valor forte.
5. Salve e aguarde o deploy — Render vai construir e iniciar sua aplicação.

Nota: o link **Deploy Flask** do Render explica este fluxo e tem exemplos. (O serviço criará a URL pública do seu app.)

### 4) Ajustes/primeiro uso após deploy

1. Acesse a URL pública fornecida pelo Render.
2. Use a tela de import (admin) para enviar a sua planilha Excel com as abas dos planos — o app criará automaticamente plans + benefits.
3. Cadastre um tutor, adicione pets e crie uma adesão (defina parcelas). O sistema gera as parcelas automaticamente.
4. Para marcar parcelas pagas, use o botão “Marcar como paga” na listagem do tutor.

### 5) Observações importantes de produção

* **Segurança**: este exemplo não possui autenticação administrativa. Em produção, adicione login/roles para proteger o upload da planilha e os endpoints de marcação de pagamento.
* **Backups**: implemente backup do Postgres (Render oferece snapshots e export). Faça dumps periódicos.
* **Job para verificar parcelas vencidas**: o app contém /cron/check\_overdue. No Render você pode criar um **Scheduled Job** (cron) que chama essa URL diariamente para marcar ou notificar.
* **Ambiente**: use DATABASE\_URL do Render (automaticamente com credenciais). Não comite segredos no Git.

## Se quiser eu faço para você (opcional)

1. Posso gerar o repositório pronto (com todos os arquivos) e fornecer instruções para você apenas *clonar* e *push* para o GitHub.
2. Posso também preparar um docker-compose.yml para rodar localmente com Postgres.

## Próximos passos recomendados (curto prazo)

* Adicionar autenticação admin (Flask-Login) para proteger /admin/import.
* Melhorar a lógica de criação de parcelas (juros, arredondamento das últimas parcelas).
* Criar relatórios: clientes com parcelas vencidas, mapas de uso de benefícios por plano (para avaliar custos).
* Integrar pagamento (ex.: gerador de boletos ou integração com gateway) para marcar parcelas automaticamente como pagas.

FIM DO DOCUMENTO.