Documentação do Projeto: Meu Controle de Pressão

Este documento fornece um guia completo para configurar, executar e entender o projeto "Meu Controle de Pressão".

1. Visão Geral do Projeto

"Meu Controle de Pressão" é uma aplicação web desenvolvida com Flask que permite aos usuários monitorar e registrar suas medições de pressão arterial. O sistema oferece funcionalidades de autenticação (cadastro, login, logout), um dashboard personalizado para visualização de dados, registro de medições com campos adicionais (horário e remédios tomados), classificação automática das medições, gráficos interativos, filtros por período, exportação de dados (PDF e CSV) e uma área de perfil do usuário com foto e dados pessoais.

2. Estrutura de Diretórios

A estrutura de diretórios do projeto é a seguinte:

```
meu-controle-pressao-v2/
├── app/
│   ├── static/
│   │   ├── css/
│   │   │   └── style.css
│   │   ├── js/
│   │   │   └── main.js
│   │   └── profile_pics/  (Diretório para fotos de perfil)
│   ├── templates/
│   │   ├── base.html
│   │   ├── index.html
│   │   ├── login.html
│   │   ├── register.html
│   │   ├── dashboard.html
│   │   └── profile.html
│   ├── instance/
│   │   └── database.db  (Banco de dados SQLite)
│   ├── __init__.py
│   ├── models.py
│   ├── forms.py
│   ├── auth_routes.py
│   └── main_routes.py
├── requirements.txt
├── run.py
├── .env
```

Documentação do Projeto: Meu Controle de Pressão

```
├── instance/
│   └── database.db
└── venv/ (Ambiente virtual Python - será criado)
```

3. Instalação e Configuração

Siga os passos abaixo para configurar e executar o projeto em seu ambiente local:

3.1. Pré-requisitos

Certifique-se de ter o Python 3 e o `pip` (gerenciador de pacotes do Python) instalados em seu sistema.

3.2. Clonar o Repositório (ou criar os arquivos)

Se você recebeu o projeto como um arquivo compactado, descompacte-o. Caso contrário, você pode criar a estrutura de diretórios e os arquivos manualmente.

3.3. Configurar o Ambiente Virtual

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

```bash
cd meu-controle-pressao-v2
python3 -m venv venv
source venv/bin/activate  # No Linux/macOS
# venv\Scripts\activate  # No Windows
```

3.4. Instalar Dependências

Com o ambiente virtual ativado, instale as dependências listadas no `requirements.txt`:

```bash
pip install -r requirements.txt
```

3.5. Configurar Variáveis de Ambiente

Crie um arquivo `.env` na raiz do projeto (`meu-controle-pressao-v2/`) com o seguinte conteúdo:

```
SECRET_KEY=sua_chave_secreta_aqui
```

Substitua `sua_chave_secreta_aqui` por uma string aleatória e segura. Você pode gerar uma usando Python:

```python
import secrets
print(secrets.token_hex(16))
```

3.6. Inicializar o Banco de Dados

O banco de dados SQLite (`database.db`) será criado automaticamente na primeira vez que a aplicação for executada. Certifique-se de que o diretório `instance` exista dentro de `app/`:

```bash
mkdir -p app/instance
```

3.7. Executar a Aplicação

Com o ambiente virtual ativado, execute o arquivo `run.py`:

```bash
python3 run.py
```

A aplicação estará acessível em `http://localhost:5000` no seu navegador.

4. Código Fonte

A seguir, o conteúdo de cada arquivo importante do projeto:

`requirements.txt`
```
Flask
Flask-SQLAlchemy
Flask-Login
Flask-WTF
python-dotenv
```

Documentação do Projeto: Meu Controle de Pressão

```
Flask-Bcrypt
email_validator
```

`run.py`
```python
from app import create_app

app = create_app()

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

`app/__init__.py`
```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager
import os

db = SQLAlchemy()
login_manager = LoginManager()
login_manager.login_view = 'auth.login'
login_manager.login_message_category = 'info'

def create_app():
    app = Flask(__name__)
    app.config['SECRET_KEY'] = os.environ.get('SECRET_KEY',
'a_default_secret_key_for_dev')
    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'

    db.init_app(app)
    login_manager.init_app(app)

    from app.auth_routes import auth
    from app.main_routes import main

    app.register_blueprint(auth)
    app.register_blueprint(main)

    with app.app_context():
        db.create_all()
```

```
    return app
```

`app/models.py`
```python
from datetime import datetime
from app import db, login_manager
from flask_login import UserMixin

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(60), nullable=False)
    profile_image = db.Column(db.String(20), nullable=False, default='default.jpg') #
Novo campo
    full_name = db.Column(db.String(100), nullable=True) # Novo campo
    date_of_birth = db.Column(db.DateTime, nullable=True) # Novo campo
    weight = db.Column(db.Float, nullable=True) # Novo campo
    height = db.Column(db.Float, nullable=True) # Novo campo
    blood_type = db.Column(db.String(5), nullable=True) # Novo campo
    medicoes = db.relationship("Medicao", backref="autor", lazy=True)

    def __repr__(self):
        return f"User('{self.username}', '{self.email}', '{self.profile_image}',
'{self.full_name}', '{self.date_of_birth}', '{self.weight}', '{self.height}',
'{self.blood_type}')"

class Medicao(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    sistolica = db.Column(db.Integer, nullable=False)
    diastolica = db.Column(db.Integer, nullable=False)
    data_medicao = db.Column(db.DateTime, nullable=False,
default=datetime.utcnow)
    notas = db.Column(db.Text, nullable=True)
    remedios_tomados = db.Column(db.Text, nullable=True)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"), nullable=False)

    def __repr__(self):
```

```
        return f"Medicao('{self.sistolica}/{self.diastolica}', '{self.data_medicao}',
'{self.remedios_tomados}')"
```

`app/forms.py`
```python
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField, IntegerField,
TextAreaField, DateField, FloatField
from wtforms.validators import DataRequired, Length, Email, EqualTo,
ValidationError, NumberRange, Optional
from flask_wtf.file import FileField, FileAllowed
from app.models import User

class RegistrationForm(FlaskForm):
    username = StringField("Nome de Usuário",
                           validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField("Email",
                        validators=[DataRequired(), Email(), Length(min=6, max=120)])
    password = PasswordField("Senha", validators=[DataRequired(), Length(min=6,
max=60)])
    confirm_password = PasswordField("Confirmar Senha",
                                     validators=[DataRequired(), EqualTo("password")])
    submit = SubmitField("Cadastrar")

    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user:
            raise ValidationError("Esse nome de usuário já está em uso. Por favor,
escolha outro.")

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError("Esse email já está em uso. Por favor, escolha outro.")

class LoginForm(FlaskForm):
    email = StringField("Email",
                        validators=[DataRequired(), Email()])
    password = PasswordField("Senha", validators=[DataRequired()])
    submit = SubmitField("Login")

class MedicaoForm(FlaskForm):
```

```
    sistolica = IntegerField("Pressão Sistólica", validators=[DataRequired(),
NumberRange(min=50, max=300)])
    diastolica = IntegerField("Pressão Diastólica", validators=[DataRequired(),
NumberRange(min=30, max=200)])
    notas = TextAreaField("Notas (opcional)", validators=[Length(max=200)])
    remedios_tomados = TextAreaField("Remédios Tomados (opcional)",
validators=[Length(max=200)])
    submit = SubmitField("Adicionar Medição")

class ProfileForm(FlaskForm):
    full_name = StringField("Nome Completo", validators=[Optional(),
Length(max=100)])
    date_of_birth = DateField("Data de Nascimento (AAAA-MM-DD)",
format="%Y-%m-%d", validators=[Optional()])
    weight = FloatField("Peso (kg)", validators=[Optional(), NumberRange(min=0)])
    height = FloatField("Altura (cm)", validators=[Optional(), NumberRange(min=0)])
    blood_type = StringField("Tipo Sanguíneo", validators=[Optional(),
Length(max=5)])
    profile_image = FileField("Foto de Perfil", validators=[FileAllowed(["jpg", "png"])])
    submit = SubmitField("Atualizar Perfil")
```

`app/auth_routes.py`

```
from flask import Blueprint, render_template, url_for, flash, redirect, request
from app import db
from app.forms import RegistrationForm, LoginForm
from app.models import User
from flask_login import login_user, current_user, logout_user, login_required
from flask_bcrypt import Bcrypt
from werkzeug.security import generate_password_hash, check_password_hash

auth = Blueprint("auth", __name__)

bcrypt = Bcrypt() # Inicialize Bcrypt aqui

@auth.route("/", methods=["GET"])
def index():
    if current_user.is_authenticated:
        return redirect(url_for("main.dashboard"))
    return render_template("index.html", title="Início")

@auth.route("/register", methods=["GET", "POST"])
def register():
```

```
    if current_user.is_authenticated:
        return redirect(url_for("main.dashboard"))
    form = RegistrationForm()
    if form.validate_on_submit():
        hashed_password =
bcrypt.generate_password_hash(form.password.data).decode("utf-8")
        user = User(username=form.username.data, email=form.email.data,
password=hashed_password)
        db.session.add(user)
        db.session.commit()
        flash("Sua conta foi criada com sucesso! Agora você pode fazer login.",
"success")
        return redirect(url_for("auth.login"))
    return render_template("register.html", title="Cadastro", form=form)

@auth.route("/login", methods=["GET", "POST"])
def login():
    if current_user.is_authenticated:
        return redirect(url_for("main.dashboard"))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            login_user(user, remember=False)
            next_page = request.args.get("next")
            return redirect(next_page) if next_page else
redirect(url_for("main.dashboard"))
        else:
            flash("Login sem sucesso. Por favor, verifique seu email e senha", "danger")
    return render_template("login.html", title="Login", form=form)

@auth.route("/logout")
def logout():
    logout_user()
    return redirect(url_for("auth.login"))
```

`app/main_routes.py`

```python
from flask import Blueprint, render_template, url_for, flash, redirect, request, jsonify
from app import db
from app.forms import MedicaoForm, ProfileForm
from app.models import Medicao, User
from flask_login import current_user, login_required
```

Documentação do Projeto: Meu Controle de Pressão

```
from datetime import datetime, timedelta
import os
import secrets
from PIL import Image

main = Blueprint("main", __name__)

def save_picture(form_picture):
    random_hex = secrets.token_hex(8)
    _, f_ext = os.path.splitext(form_picture.filename)
    picture_fn = random_hex + f_ext
    picture_path = os.path.join(main.root_path, 'static/profile_pics', picture_fn)

    output_size = (125, 125)
    i = Image.open(form_picture)
    i.thumbnail(output_size)
    i.save(picture_path)

    return picture_fn

@main.route("/dashboard", methods=["GET", "POST"])
@login_required
def dashboard():
    form = MedicaoForm()
    if form.validate_on_submit():
        medicao = Medicao(sistolica=form.sistolica.data,
diastolica=form.diastolica.data,
                          notas=form.notas.data,
remedios_tomados=form.remedios_tomados.data, autor=current_user)
        db.session.add(medicao)
        db.session.commit()
        flash("Medição adicionada com sucesso!", "success")
        return redirect(url_for("main.dashboard"))

    # Fetch measurements for the current user
    medicoes =
Medicao.query.filter_by(user_id=current_user.id).order_by(Medicao.data_medicao.d
esc()).all()

    # Prepare data for chart
    labels = [m.data_medicao.strftime("%d/%m/%Y %H:%M") for m in medicoes]
    sistolicas = [m.sistolica for m in medicoes]
    diastolicas = [m.diastolica for m in medicoes]
```

```
    return render_template("dashboard.html", title="Dashboard", form=form,
medicoes=medicoes,
                           labels=labels, sistolicas=sistolicas, diastolicas=diastolicas)

@main.route("/get_medicoes", methods=["GET"])
@login_required
def get_medicoes():
    start_date_str = request.args.get("start_date")
    end_date_str = request.args.get("end_date")

    query = Medicao.query.filter_by(user_id=current_user.id)

    if start_date_str:
        start_date = datetime.strptime(start_date_str, "%Y-%m-%d")
        query = query.filter(Medicao.data_medicao >= start_date)
    if end_date_str:
        end_date = datetime.strptime(end_date_str, "%Y-%m-%d") + timedelta(days=1)
- timedelta(microseconds=1) # Include full end day
        query = query.filter(Medicao.data_medicao <= end_date)

    medicoes = query.order_by(Medicao.data_medicao.desc()).all()

    medicoes_data = []
    for m in medicoes:
        status = "Normal"
        if m.sistolica >= 140 or m.diastolica >= 90:
            status = "Hipertensão Estágio 2"
        elif m.sistolica >= 130 or m.diastolica >= 80:
            status = "Hipertensão Estágio 1"
        elif m.sistolica >= 120 and m.diastolica < 80:
            status = "Elevada"

        medicoes_data.append({
            "id": m.id,
            "sistolica": m.sistolica,
            "diastolica": m.diastolica,
            "data_medicao": m.data_medicao.strftime("%d/%m/%Y %H:%M"),
            "notas": m.notas,
            "remedios_tomados": m.remedios_tomados,
            "status": status
        })
    return jsonify(medicoes_data)

@main.route("/profile", methods=["GET", "POST"])
```

Documentação do Projeto: Meu Controle de Pressão

```
@login_required
def profile():
    form = ProfileForm()
    if form.validate_on_submit():
        if form.profile_image.data:
            picture_file = save_picture(form.profile_image.data)
            current_user.profile_image = picture_file
        current_user.full_name = form.full_name.data
        current_user.date_of_birth = form.date_of_birth.data
        current_user.weight = form.weight.data
        current_user.height = form.height.data
        current_user.blood_type = form.blood_type.data
        db.session.commit()
        flash("Seu perfil foi atualizado com sucesso!", "success")
        return redirect(url_for("main.profile"))
    elif request.method == "GET":
        form.full_name.data = current_user.full_name
        form.date_of_birth.data = current_user.date_of_birth
        form.weight.data = current_user.weight
        form.height.data = current_user.height
        form.blood_type.data = current_user.blood_type
    image_file = url_for("static", filename="profile_pics/" + current_user.profile_image)
    return render_template("profile.html", title="Perfil", image_file=image_file,
form=form)
```

`app/static/css/style.css`

```css
:root {
    --background-color-light: #f8f9fa;
    --text-color-light: #212529;
    --card-bg-light: #ffffff;
    --card-border-light: #dee2e6;
    --input-bg-light: #ffffff;
    --input-border-light: #ced4da;
    --jumbotron-bg-light: #e9ecef;
    --jumbotron-text-light: #495057;

    --background-color-dark: #343a40;
    --text-color-dark: #f8f9fa;
    --card-bg-dark: #495057;
    --card-border-dark: #6c757d;
    --input-bg-dark: #6c757d;
    --input-border-dark: #adb5bd;
```

Documentação do Projeto: Meu Controle de Pressão

```
    --jumbotron-bg-dark: #212529;
    --jumbotron-text-dark: #f8f9fa;

    --background-color: var(--background-color-light);
    --text
(Content truncated due to size limit. Use line ranges to read in chunks)
```