**Relatório Detalhado do Progresso do Projeto Agendador Whatsapp – 14/12**

**Objetivo do Projeto**

Criar uma aplicação Flask para gerenciamento de agendamentos via WhatsApp, incluindo funcionalidades administrativas para cadastro de usuários, agendamentos e envio de notificações.

**Etapas Concluídas**

**1. Estrutura do Projeto**

Criamos e configuramos a estrutura básica do projeto com as seguintes pastas e arquivos:

Projeto\_Agendamento\_Whatsapp/

├── app/

│ ├── \_\_init\_\_.py

│ ├── config/

│ │ ├── \_\_init\_\_.py

│ │ ├── development.py

│ ├── extensions.py

│ ├── models/

│ │ ├── \_\_init\_\_.py

│ │ ├── user.py

│ │ ├── appointment.py

│ ├── routes/

│ │ ├── \_\_init\_\_.py

│ │ ├── main.py

│ │ ├── appointments.py

├── migrations/

├── requirements.txt

├── run.py

* **app/\_\_init\_\_.py**: Configuramos a função create\_app() para inicializar o Flask e registrar extensões e rotas.
* **app/config/development.py**: Definimos as configurações de desenvolvimento, incluindo a URL do banco de dados MySQL.
* **app/extensions.py**: Configuramos o SQLAlchemy e o Flask-Migrate.

**2. Banco de Dados**

1. **Modelos Criados:**
   * **User** (app/models/user.py):
   * from app.extensions import db
   * class User(db.Model):
   * \_\_tablename\_\_ = "users"
   * id = db.Column(db.Integer, primary\_key=True)
   * name = db.Column(db.String(100), nullable=False)
   * phone = db.Column(db.String(15), unique=True, nullable=False)
   * created\_at = db.Column(db.DateTime, default=db.func.now())
   * def \_\_repr\_\_(self):

return f"<User {self.name}>"

* + **Appointment** (app/models/appointment.py):
  + from app.extensions import db
  + from sqlalchemy.orm import relationship
  + class Appointment(db.Model):
  + \_\_tablename\_\_ = "appointments"
  + id = db.Column(db.Integer, primary\_key=True)
  + user\_id = db.Column(db.Integer, db.ForeignKey("users.id"))
  + date = db.Column(db.DateTime, nullable=False)
  + type = db.Column(db.String(50), nullable=False)
  + duration = db.Column(db.Integer, nullable=False)
  + created\_at = db.Column(db.DateTime, default=db.func.now())
  + user = relationship("User", backref="appointments")
  + def \_\_repr\_\_(self):

return f"<Appointment {self.type} on {self.date}>"

1. **Migrações Aplicadas:**
   * Inicializamos as migrações com o Flask-Migrate:
   * flask db init
   * flask db migrate -m "Initial migration"

flask db upgrade

* + Verificamos que as tabelas users, appointments e alembic\_version foram criadas corretamente no MySQL.

1. **Timeout do MySQL:**
   * Ajustamos a configuração para evitar problemas de timeout:
     + Configuramos o pool do SQLAlchemy no arquivo development.py:
     + SQLALCHEMY\_ENGINE\_OPTIONS = {
     + "pool\_pre\_ping": True,
     + "pool\_recycle": 280,
     + "pool\_size": 10,
     + "max\_overflow": 5,

}

**3. Rotas Implementadas**

1. **Rotas de Usuários** (app/routes/main.py):
   * **GET /users**: Lista todos os usuários cadastrados.
   * @main.route("/users", methods=["GET"])
   * def get\_users():
   * try:
   * users = User.query.all()
   * return jsonify([
   * {"id": user.id, "name": user.name, "phone": user.phone} for user in users
   * ])
   * except OperationalError as e:

return jsonify({"error": "Database connection issue", "details": str(e)}), 500

* + **POST /users**: Cria um novo usuário no banco de dados.
  + @main.route("/users", methods=["POST"])
  + def create\_user():
  + try:
  + data = request.json
  + if not data.get("name") or not data.get("phone"):
  + return jsonify({"error": "Name and phone are required"}), 400
  + new\_user = User(name=data["name"], phone=data["phone"])
  + db.session.add(new\_user)
  + db.session.commit()
  + return jsonify({"message": "User created", "id": new\_user.id}), 201
  + except OperationalError as e:
  + db.session.rollback()
  + return jsonify({"error": "Database connection issue", "details": str(e)}), 500
  + except Exception as e:
  + db.session.rollback()

return jsonify({"error": "An unexpected error occurred", "details": str(e)}), 500

1. **Testes no Postman:**
   * **GET /users**:
     + Retornou a lista de usuários corretamente.
     + Respostas esperadas:
       - Com usuários: [{"id": 1, "name": "João", "phone": "11999999999"}]
       - Sem usuários: []
   * **POST /users**:
     + Criou novos usuários com sucesso.
     + Retorno esperado:
     + {
     + "message": "User created",
     + "id": 1

}

**Onde Paramos**

* Concluímos os endpoints para **usuários** e testamos no Postman com sucesso.
* Verificamos que o banco de dados está funcionando corretamente e que os ajustes de timeout no MySQL estão aplicados.

**Próximos Passos**

1. **Implementar Rotas de Agendamentos:**
   * Criar os endpoints para GET /appointments e POST /appointments.
   * Testar no Postman:
     + Listagem de agendamentos.
     + Criação de agendamentos vinculados a usuários existentes.
2. **Iniciar Integração com WhatsApp:**
   * Decidir qual API de terceiros usar (ex.: Twilio, WhatsApp Cloud API).
   * Configurar as credenciais e iniciar a implementação do envio de notificações.
3. **Melhorar o Tratamento de Erros:**
   * Garantir que todos os endpoints lidam corretamente com erros inesperados.

Se precisar de qualquer ajuste ou detalhamento adicional, é só avisar! 🚀