Orientações sobre o Projeto Personal Client

Funcionalidades Principais:

Cadastro de Alunos:

- Formulário para inserção de informações dos alunos (nome, idade, contato, etc.).
- Upload de foto opcional para identificação visual dos alunos.

Gestão de Pagamentos:

- Registro e controle de pagamentos (mensais, trimestrais, anuais, etc.).
- Notificações automáticas ou lembretes sobre vencimentos de pagamentos.

Histórico de Pagamentos:

- Exibição do histórico de pagamentos com detalhes como data, valor, método de pagamento (cartão, transferência, etc.).
- Possibilidade de exportar o histórico em PDF ou Excel.

Agenda de Aulas:

- Sistema de calendário para gerenciar os horários das aulas.
- Controle de presença dos alunos.

Comunicação:

- Envio de notificações e mensagens via e-mail ou SMS para os alunos sobre datas importantes ou mudanças de horários.

Relatórios e Estatísticas:

- Relatórios financeiros mensais ou anuais.
- Estatísticas sobre a frequência dos alunos nas aulas.

Tecnologias Sugeridas com Python:

Back-end:

- **Django ou Flask:** O Django é um framework completo para o desenvolvimento de aplicações web, enquanto o Flask é mais leve e flexível. Ambos oferecem integração com sistemas de autenticação e segurança de dados.
- **Django Rest Framework (DRF):** Para criar uma API RESTful, o DRF é excelente no desenvolvimento de APIs que poderão ser consumidas por aplicações móveis ou web.

Banco de Dados:

- **PostgreSQL** ou **MySQL**: Ambas são ótimas opções para armazenar os dados dos alunos e seus respectivos pagamentos.
- **SQLAlchemy (com Flask)** ou **Django ORM**: Para interagir com o banco de dados de forma simplificada, aproveitando as funcionalidades de ORM (Object-Relational Mapping).

Autenticação e Segurança:

- **JWT (JSON Web Token)** ou **Django Auth** para garantir a autenticação segura dos usuários.
- **Proteção contra CSRF e XSS** (nativamente disponíveis no Django).

Frontend (Opcional):

- Se o aplicativo for web, **Django Templates** pode ser usado para renderizar páginas HTML dinâmicas. Caso queira uma interface mais moderna e responsiva, você pode usar frameworks como **React.js** (que pode se comunicar com a API criada com Django ou Flask).
- **Bootstrap** ou **Tailwind CSS** para estilização responsiva e moderna das páginas.

Agenda e Controle de Aulas:

- Para gerenciar a agenda, você pode integrar bibliotecas como o **FullCalendar** ou criar um sistema simples de agenda com exibição das datas diretamente no frontend.
- Para controle de presença, é possível marcar as presenças diretamente no sistema e gerar relatórios sobre as aulas assistidas.

Relatórios:

- **ReportLab** (para geração de PDFs com relatórios de pagamentos e presença).
- **Pandas** (para manipulação e geração de relatórios em formatos como CSV e Excel).

Considerações sobre o Desenvolvimento:

Desenvolvimento da API:

- Utilize o Django Rest Framework (ou Flask com Flask-RESTful) para criar uma API que será consumida pela interface (web ou mobile).
- Essa API deve gerenciar o CRUD (Create, Read, Update, Delete) dos alunos e o controle de pagamentos e aulas.

Segurança:

- Garanta que todos os dados sensíveis (como senhas) sejam devidamente criptografados.
- Use SSL/TLS para proteger a comunicação entre o cliente e o servidor.

Hospedagem:

- O aplicativo pode ser hospedado em serviços como **Heroku**, **PythonAnywhere** ou **AWS**.
- Se for uma aplicação para dispositivos móveis, pode-se usar frameworks como **Kivy** (para desenvolvimento mobile nativo com Python), ou utilizar React Native para criar a interface móvel e conectar-se à API Python.