RAv – REGISTRO DE AVALIAÇÃO (DESCRIÇÃO DO PROCESSO DE APRENDIZAGEM DO ESTUDANTE)

Considerando a importância que a avaliação para as aprendizagens tem na vida dos estudantes, o Formulário Descrição do Processo de Aprendizagem do Estudante tem caráter dinâmico e a sua prática não se limita à atribuição de conceitos ou métricas, que expressem a proficiência obtida pelos estudantes. A avaliação formativa exige do professor(a), uma observação atenta e reflexiva. O registro processual dessas observações deve estar articulado à reflexão e à intervenção pedagógica, com elementos que sejam capazes de dar visibilidade ao percurso estudantil da criança em relação aos objetivos propostos, bem como dos aspectos que necessitam de atenção para que sejam alcançados.

No formulário, **Descrição do Processo de Aprendizagem do Estudante,** deve estar apresentado a construção da aprendizagem e do desenvolvimento do estudante durante o bimestre, considerando os principais avanços e o que o estudante demonstrou ter apreendido durante este processo.

No relato deverá conter questões que evidenciem:

* O diagnóstico de cada estudante, baseado nos objetivos de aprendizagem de cada ano estabelecidos pela 2a. edição do Currículo em Movimento do Distrito Federal - Ensino Fundamental;
* Os objetivos de aprendizado alcançados pelo estudante no bimestre, tomando como referência o diagnóstico feito pelo professor e o Currículo em Movimento do Distrito Federal - Ensino Fundamental;
* As intervenções para a progressão das aprendizagens do estudante realizadas pelo professor(a) e pelos demais sujeitos envolvidos no processo de ensino e aprendizagem;
* As dificuldades do estudante percebidas durante o bimestre;
* As ações que a serem feitas no próximo bimestre para que o aluno tenha avanços.

Para referenciar as informações dadas no registro o professor poderá utilizar diferentes instrumentos, tais como: observação, provas, exercícios, pesquisas, ficha individual, portfólio, produções de trabalhos do estudante individual e/ou em grupo, reagrupamentos, projetos interventivos e outros instrumentos de que dispuser, e que dizem respeito à trajetória do estudante na unidade escolar (DIRETRIZES PEDAGÓGICAS PARA ORGANIZAÇÃO ESCOLAR DO 2o CICLO, 2014).

ESTRUTURA:

1. Banco de Dados:

- Tabelas:

- Alunos: informações básicas (nome, matrícula, etc.).

- Disciplinas: lista de disciplinas do quinto ano.

- Habilidades: habilidades específicas de cada disciplina, conforme o Currículo em Movimento.

- Avaliações: registros das avaliações dos alunos (aluno, disciplina, habilidade, nível de desenvolvimento, comentários).

2. Interface do Usuário:

- Login: para professores acessarem o sistema.

- Dashboard: painel inicial com acesso rápido às turmas e alunos.

- Formulário de Avaliação:

- Selecionar aluno.

- Selecionar disciplina.

- Selecionar habilidade.

- Inserir nível de desenvolvimento e comentários adicionais.

3. Funcionalidades:

- Cadastro e gestão de alunos.

- Cadastro e gestão de disciplinas e habilidades.

- Formulário de avaliação.

- Geração de relatórios individuais e por turma.

- Exportação de relatórios em PDF ou outro formato desejado.

4. Controles de Acesso:

- Garantir que apenas professores autorizados possam acessar e editar as avaliações.

Telas

1. Login:

- Campos: Nome de usuário e senha.

- Funcionalidades: Validação e autenticação do usuário.

2. Dashboard:

- Funcionalidades:

- Exibição das turmas e alunos.

- Navegação para avaliação e relatórios.

- Componentes:

- Lista de turmas/alunos.

- Botões para adicionar novas avaliações.

3. Formulário de Avaliação:

- Campos:

- Seleção de aluno (dropdown ou busca).

- Seleção de disciplina (dropdown).

- Seleção de habilidade (baseada na disciplina).

- Seleção de nível de desenvolvimento (dropdown).

- Comentários adicionais (textbox).

- Funcionalidades: Salvar avaliação, validação de campos obrigatórios.

4. Relatórios:

- Funcionalidades:

- Geração de relatórios individuais e por turma.

- Opção de exportação (PDF, Excel).

5. Cadastro e Gestão:

- Campos:

- Alunos (nome, matrícula, turma).

- Disciplinas (nome).

- Habilidades (descrição, disciplina associada).

- Objetivos (descrição, disciplina associada).

- Conteúdos (descrição, objetivo associado).

- Níveis de proficiência (descrição).

- Funcionalidades: CRUD (Create, Read, Update, Delete) para cada entidade.

### Plano de Desenvolvimento

1. Configuração do Ambiente:

- Configuração do repositório no GitHub.

- Configuração do ambiente de desenvolvimento (IDE, banco de dados, etc.).

2. Desenvolvimento do Backend:

- Criação das tabelas no banco de dados.

- Desenvolvimento da API para gerenciar dados de alunos, disciplinas, habilidades, etc.

3. Desenvolvimento do Frontend:

- Criação das telas de login e dashboard.

- Desenvolvimento do formulário de avaliação.

- Implementação das funcionalidades de cadastro e gestão.

4. Testes:

- Testes unitários para o backend.

- Testes de interface para o frontend.

- Testes de integração.

5. Documentação:

- Documentação do código.

- Manual de usuário.

6. Deploy:

- Configuração do servidor.

- Deploy da aplicação.

**Próximos Passos**

1. **Testes**: Implementar testes unitários e de integração para garantir que todas as funcionalidades estão funcionando corretamente.
2. **Documentação**: Adicionar documentação para as APIs, como endpoints, métodos suportados, parâmetros esperados, e exemplos de respostas.
3. **Segurança**: Avaliar e implementar medidas adicionais de segurança, como hashing de senhas e proteção contra ataques CSRF.
4. **Desempenho**: Otimizar consultas ao banco de dados e avaliar a performance geral da aplicação.
5. **Deploy**: Configurar o ambiente de produção e implementar a aplicação em um servidor ou serviço de cloud, como Heroku, AWS, ou Azure.

### Próximos Passos

1. **Testes Mais Abrangentes**:
   * Adicionar mais testes para cobrir todos os cenários, incluindo casos de erro.
   * Usar pytest-cov para medir a cobertura de testes e garantir que todo o código relevante seja testado.
2. **Implantação**:
   * Configurar um ambiente de produção para o Flask.
   * Usar um servidor WSGI como gunicorn para servir o aplicativo Flask.
   * Configurar um banco de dados PostgreSQL em um servidor de produção.
3. **Documentação**:
   * Documentar todas as rotas da API e como usá-las.
   * Incluir instruções para desenvolvedores futuros sobre como configurar e contribuir para o projeto.
4. **CI/CD**:
   * Configurar integração contínua (CI) para executar testes automaticamente em cada push.
   * Configurar entrega contínua (CD) para automatizar a implantação do aplicativo.

Estrutura do Projeto

projetoRAv/

├── migrations/

│ └── \_\_pycache\_\_/

├── public/

│ ├── scripts.js

│ ├── styles.css

│ └── index.html

├── src/

│ ├── \_\_init\_\_.py

│ ├── app.py

│ ├── cadastro.py

│ └── avaliacao.py

├── templates/

│ ├── avaliacao.html

│ ├── cadastro.html

│ ├── dashboard.html

│ ├── login.html

│ └── relatorios.html

├── tests/

│ ├── \_\_init\_\_.py

│ ├── test\_app.py

│ ├── test\_avaliacao.py

│ └── test\_cadastro.py

├── package.json

├── pytest.ini

├── relatorio\_avaliativo\_db.sql

└── README.md

### Recomendações:

* **Segurança**: Não foi detectado um arquivo de política de segurança (SECURITY.md). Recomenda-se a criação deste arquivo para definir diretrizes de segurança e procedimentos para relatar vulnerabilidades.
* **Documentação**: Melhorar a documentação, especialmente o README, para fornecer uma visão mais detalhada do projeto, instruções de configuração,

Próximos Passos para o Desenvolvimento (15/5)

1. **Integração de Banco de Dados**: Garantir que as tabelas no arquivo relatorio\_avaliativo\_db.sql estejam atualizadas e configuradas corretamente no ambiente de desenvolvimento. Isso inclui a verificação de todas as tabelas e suas relações para que estejam de acordo com os requisitos do sistema.
2. **Desenvolvimento de Funcionalidades**: Continuar a implementação das funcionalidades principais no diretório src, incluindo o desenvolvimento de novos módulos conforme necessário e a melhoria dos existentes.
3. **Testes**: Ampliar a cobertura de testes no diretório tests, criando casos de teste para todas as funcionalidades críticas. Isso garantirá que o código seja robusto e livre de erros antes da implantação.
4. **Documentação e Suporte**: Atualizar a documentação no diretório Materiais suporte para refletir as mudanças recentes no código e garantir que todos os membros da equipe estejam alinhados com as últimas atualizações.
5. **Ambiente Virtual e Dependências**: Manter o ambiente virtual atualizado e garantir que o requirements.txt contenha todas as bibliotecas necessárias para o funcionamento do projeto. Verificar a compatibilidade de versões e resolver conflitos, se houver.

dashboard.html

O código do dashboard.html está bem estruturado, utilizando Bootstrap para a estilização e organização dos elementos. A inclusão de modais para adicionar, editar e excluir alunos é uma boa prática.

**Sugestões de Melhoria:**

1. **Carregamento de Dados do Estudante:** No momento, a lógica para carregar os detalhes do estudante selecionado está incompleta. A função studentList.onchange precisa ser aprimorada para buscar as informações do estudante selecionado no backend (através de uma requisição à API) e exibir os dados no student-info.
2. **Tratamento de Erros:** Implementar tratamento de erros nas requisições à API (POST, PUT, DELETE) para informar o usuário sobre possíveis falhas, como aluno não encontrado ou erro de validação.
3. **Melhoria da Interface do Usuário:**
   * **Ordenação da Lista de Alunos:** Ordenar a lista de alunos alfabeticamente para facilitar a busca.
   * **Filtro de Alunos:** Adicionar um campo de busca para filtrar a lista de alunos por nome ou série.
   * **Feedback Visual:** Adicionar feedback visual (como um spinner) durante o carregamento dos dados para indicar que a requisição está em andamento.
   * **Mensagens de Sucesso/Erro:** Exibir mensagens de sucesso (ex: "Aluno cadastrado com sucesso!") ou erro (ex: "Erro ao excluir aluno.") para informar o usuário sobre o resultado das operações.

### app.py

O código do app.py demonstra uma boa estrutura com a utilização do Flask, SQLAlchemy e Flask-RESTX para a criação da API. A implementação dos endpoints para gerenciar alunos está correta, mas algumas melhorias podem ser feitas.

**Sugestões de Melhoria:**

1. **Segurança:**
   * **CSRF Protection:** O CSRFProtect já está importado e configurado, porém não está sendo aplicado em nenhuma rota. É necessário adicionar a proteção CSRF às rotas que modificam dados (POST, PUT, DELETE).
   * **Validação de Entrada:** Reforçar a validação dos dados de entrada para evitar vulnerabilidades como injeção de SQL. Utilize bibliotecas como Marshmallow para auxiliar na validação.
2. **Tratamento de Erros Personalizado:** Criar um tratamento de erros personalizado para lidar com erros específicos da aplicação, como aluno não encontrado, erro de validação, etc., e retornar respostas mais informativas ao usuário.
3. **Outras Melhorias:**
   * **Autenticação:** Implementar um sistema de autenticação mais robusto, como tokens JWT, para proteger as rotas da API e garantir que apenas usuários autorizados possam acessar os dados.
   * **Documentação da API:** Aprimorar a documentação da API utilizando o Swagger UI para facilitar o entendimento dos endpoints e seus parâmetros.
   * **Logs:** Aprimorar os logs para registrar informações mais detalhadas sobre as operações realizadas na API, como dados de entrada e saída, erros, etc.