Mini-Prova

WEB II

Componentes: Daniel Otaviano (20210054512)

Repositório: https://github.com/danielotaviano/miniprova

1. Ideia Principal

O Mini-Prova, tem como objetivo, ser uma plataforma simplificada para aplicações de provas online. Nossas principais entidades, professores e alunos, podem realizar ações de criação de turmas, inscrições em turmas, criação de prova e realização de determinada prova.

2. Ferramentas técnicas

Para construção do projeto, foi utilizado a linguagem de programação "Rust", junto com algumas libs de auxilios, cito como principais, Axum, tokio e minijinja.

3. Principais features

- Oauth via github como principal fonte de autenticação de usuário.
- Utilização de dois perfis distintos
  - Professor
    1. Criação de turmas
    2. Criação de Provas para turmas especificas, cada prova possui sua data de inicio e fim, bem como um titulo, questões e respostas para cada questão.
    3. Acompanhamento de provas realizadas por alunos
  - Aluno
    1. Inscrição em determinada turma
    2. Acesso a provas disponíveis em turmas matriculadas
    3. Resolução de provas
    4. Visualização de notas obtidas em provas realizadas ### 4. Critérios de Avaliação
- O sistema deve estar funcional e com funcionalidade explícita. Deve dividir, de maneira adequada, componentes importantes da aplicação, criando seus próprios pacotes e subdiretórios. Apresentação e propostas adequadas (1,00 ponto).
  - Visualização na DEMO
- O sistema deve fazer extenso uso dos recursos de persistência e mapeamento objeto-relacional. Recomendo a utilização do Spring Data Jpa com JpaRepository. Deve fazer uso de, no mínimo, 5 entidades que serão mapeadas em tabelas. Deve existir relacionamentos entre as tabelas, pelo menos um relacionamento de cada tipo: um-para-um, um-para-muitos

(muitos-para-um), e muitos-para-muitos. Pelo menos um dos relacionamentos feitos deve ser bidirecional. Sua aplicação deve habilitar todas as operações de CRUD para pelo menos 4 entidades. (4,00 pontos).
  - Utilização de banco de dados Postgres, com mapeamento de entidades e relacionamentos.
  - Entidades mapeadas: Usuário, Turma, Prova, Questao, Resposta, Além de entidades de controle como a de respostas de alunos.
  - Relacionamentos: um-para-um (Não foi utilizado), um-para-muitos (Turma -> Professor), muitos-para-muitos (Aluno -> Turma). Obs: Não foi utilizado relacionamento bidirecional.
  - Operações de CRUD: Turma, Prova, Questao, Resposta.
- Sua aplicação deve seguir a arquitetura MVC. Para cada modelo operacional, implemente o respectivo controller, que deve ser o responsável pelas requisições e direcionamento das respostas (páginas carregadas) e os services para desacoplar sua aplicação das classes repositórios ou de regras de negócio. Pelo menos duas classes de repositórios devem conter queries de busca personalizadas (2,00 pontos).
  - Utilização de arquitetura MVC, com controllers e services para cada entidade.
  - Por está utilizando sqlx, todas as queries foram personalizadas.
- Personalização e Usabilidade: Utilize recursos de javascript e css para personalizar sua aplicação. Você pode utilizar frameworks de layout como Bootstrap e Materialize para personalizar suas páginas.
  - Utilização de bootstrap para personalização de páginas.

5. Execução

Para execução do projeto, é necessário ter o docker e docker-compose instalados. Após isso, basta executar o comando `docker-compose up` na raiz do projeto para subir o banco de dados postgres. Obs: Necessita ser criado um arquivo .env na raiz do projeto seguindo o modelo do arquivo .env.example. Ao ter um serviço postgres rodando, basta executar o comando `cargo run` na raiz do projeto para subir o servidor.