Avaliação Prática: Desenvolvimento de API de Gerenciamento de Projetos Simplificada

Objetivo do Trabalho

Avaliar a capacidade do aluno de analisar uma descrição de requisitos, extrair informações pertinentes e projetar uma API Web RESTful utilizando Python, FastAPI e Pydantic. O aluno deverá demonstrar compreensão na definição de endpoints, modelos de dados, regras de negócio e códigos de status HTTP adequados.

Descrição da API a ser Desenvolvida (Texto Não Estruturado)

A "DevMasters Inc." precisa de uma nova API para um sistema interno de gerenciamento de mini-projetos. Atualmente, eles controlam tudo em planilhas e isso está se tornando caótico. A ideia é ter um sistema onde possam cadastrar cada mini-projeto com um **título** claro e uma **descrição** mais detalhada do que precisa ser feito. Para organização, cada projeto precisa ser classificado com uma **prioridade**, que eles definiram como um número de 1 (mais alta) a 3 (mais baixa). Além disso, é crucial saber o **status** atual do projeto, que pode ser "Planejado", "Em Andamento", "Concluído" ou "Cancelado".

Ao criar um novo projeto, o sistema deve automaticamente registrar a **data e hora de criação** e atribuir um **identificador único** para que ele possa ser rastreado. Os usuários da API precisarão listar todos os projetos, e seria útil se pudessem **paginar** os resultados, pois a lista pode ficar longa. Também é importante que eles possam **filtrar** os projetos listados, tanto pelo status quanto pela prioridade, para encontrar rapidamente o que procuram.

Claro, eles precisam ser capazes de visualizar os detalhes de um **projeto específico** usando seu identificador. As informações dos projetos podem mudar, então deve ser possível **atualizar** qualquer um dos campos preenchidos pelo usuário (título, descrição, prioridade, status). Não é permitido alterar o ID ou a data de criação. Finalmente, se um projeto for cadastrado por engano ou não for mais necessário, deve haver uma forma de **removê-lo** do sistema. A API precisa ser robusta, validando os dados de entrada – por exemplo, a prioridade não pode ser 0 ou 5, e o status deve ser um dos pré-definidos. Respostas rápidas e documentação clara (como a que o FastAPI gera) são esperadas.

Extração dos Requisitos Funcionais e Não-Funcionais

Requisitos Funcionais (RF)

Os requisitos funcionais descrevem o que o sistema deve fazer. Eles foram identificados a partir das funcionalidades explícitas solicitadas no texto:

1. **RF01: Criar novo projeto.**
    - **Localização:** "cadastrar cada mini-projeto com um título claro e uma descrição mais detalhada...", "Ao criar um novo projeto, o sistema deve automaticamente registrar a data e hora de criação e atribuir um identificador único..."
    - **Identificação:** A necessidade de adicionar novas entidades (projetos) ao sistema.
2. **RF02: Listar todos os projetos.**
    - **Localização:** "Os usuários da API precisarão listar todos os projetos..."

    - **Identificação:** A necessidade de visualizar um conjunto de entidades.
3. **RF03: Paginar listagem de projetos.**
    - **Localização:** "...e seria útil se pudessem paginar os resultados, pois a lista pode ficar longa."
    - **Identificação:** Um atributo da funcionalidade de listagem para lidar com grandes volumes de dados.
4. **RF04: Filtrar projetos por status.**
    - **Localização:** "Também é importante que eles possam filtrar os projetos listados, tanto pelo status..."
    - **Identificação:** A capacidade de restringir a lista de projetos com base em um critério específico.
5. **RF05: Filtrar projetos por prioridade.**
    - **Localização:** "...quanto pela prioridade, para encontrar rapidamente o que procuram."
    - **Identificação:** Outro critério de filtragem para a listagem de projetos.
6. **RF06: Visualizar detalhes de um projeto específico.**
    - **Localização:** "Claro, eles precisam ser capazes de visualizar os detalhes de um projeto específico usando seu identificador."
    - **Identificação:** A necessidade de recuperar uma entidade individual.
7. **RF07: Atualizar um projeto existente.**
    - **Localização:** "As informações dos projetos podem mudar, então deve ser possível atualizar qualquer um dos campos preenchidos pelo usuário (título, descrição, prioridade, status)."
    - **Identificação:** A capacidade de modificar dados de uma entidade existente.
8. **RF08: Deletar um projeto.**
    - **Localização:** "...se um projeto for cadastrado por engano ou não for mais necessário, deve haver uma forma de removê-lo do sistema."
    - **Identificação:** A capacidade de excluir uma entidade do sistema.

Requisitos Não-Funcionais (RNF)

Os requisitos não-funcionais descrevem como o sistema deve ser, ou qualidades que ele deve ter.

1. **RNF01: Atribuição automática de ID único.**
    - **Localização:** "...atribuir um identificador único para que ele possa ser rastreado."
    - **Identificação:** Uma característica de como os dados são gerenciados internamente, garantindo unicidade.
2. **RNF02: Registro automático de data/hora de criação.**
    - **Localização:** "...o sistema deve automaticamente registrar a data e hora de criação..."
    - **Identificação:** Outra característica de gerenciamento de dados interno.
3. **RNF03: Validação de dados de entrada.**
    - **Localização:** "A API precisa ser robusta, validando os dados de entrada – por exemplo, a prioridade não pode ser 0 ou 5, e o status deve ser um dos pré-definidos."
    - **Identificação:** Requisito de qualidade de dados e robustez do sistema.
4. **RNF04: Desempenho (respostas rápidas).**
    - **Localização:** "Respostas rápidas..."
    - **Identificação:** Uma expectativa de performance da API.
5. **RNF05: Usabilidade (documentação clara).**
    - **Localização:** "...e documentação clara (como a que o FastAPI gera) são esperadas."
    - **Identificação:** Facilidade de uso e entendimento da API, mencionando a documentação automática.

Extração das Regras de Negócio (Agrupadas por Requisito Funcional)

As regras de negócio definem ou restringem como as operações podem ser realizadas.

- **RF01: Criar novo projeto**
    - **BN01:** Um projeto deve ter um título (obrigatório).
    - **BN02:** Um projeto pode ter uma descrição (opcional).
    - **BN03:** Um projeto deve ter uma prioridade (obrigatória), que deve ser um número inteiro entre 1 e 3.
    - **BN04:** Um projeto deve ter um status (obrigatório), que deve ser um dos seguintes: "Planejado", "Em Andamento", "Concluído", "Cancelado".
    - **BN05:** Na criação, se o status não for fornecido, pode-se assumir um padrão (ex: "Planejado") ou ser obrigatório. (O texto diz "status atual do projeto", implicando que é definido. Vamos considerar obrigatório).
    - **BN06:** Um ID único deve ser gerado e atribuído ao projeto no momento da criação. [Relacionado com RNF01]
    - **BN07:** A data e hora de criação devem ser registradas automaticamente. [Relacionado com RNF02]
- **RF03: Paginar listagem de projetos**
    - **BN08:** A paginação deve permitir que o cliente especifique um número de itens por página e um deslocamento (offset/skip).
    - **BN09:** Se não especificado, deve haver um limite padrão de itens por página (ex: 10).
- **RF04: Filtrar projetos por status**
    - **BN10:** A filtragem por status deve aceitar os valores "Planejado", "Em Andamento", "Concluído", "Cancelado".
- **RF05: Filtrar projetos por prioridade**
    - **BN11:** A filtragem por prioridade deve aceitar os valores 1, 2 ou 3.
- **RF07: Atualizar um projeto existente**
    - **BN12:** O título, descrição, prioridade e status de um projeto podem ser atualizados.
    - **BN13:** As mesmas validações de dados da criação (BN01, BN03, BN04) se aplicam aos campos sendo atualizados.
    - **BN14:** O ID do projeto não pode ser alterado.
    - **BN15:** A data de criação do projeto não pode ser alterada.

Descrição das Funcionalidades da API (Endpoints)

Aqui estão descritos formalmente os endpoints a serem implementados, contendo detlhes sobre seu métodos HTTP, status codes, parâmetros e modelos Pydantic utilizados.

Definição dos Endpoints

1. **Listar todos os projetos (com filtros e paginação)**

    - **Método HTTP:** `GET`
    - **URI:** `/projects`
    - **Descrição:** Lista todos os projetos cadastrados. Permite filtros por status e prioridade, e suporta paginação.
    - **Parâmetros (Query):**
        - `skip: int = Query(0, ge=0)`: Número de registros a pular (para paginação).
        - `limit: int = Query(10, gt=0, le=100)`: Número máximo de registros a retornar.
        - `status: Optional[Literal["Planejado", "Em Andamento", "Concluído", "Cancelado"]] = Query(None)`: Filtrar por status do projeto.
        - `priority: Optional[Literal[1, 2, 3]] = Query(None)`: Filtrar por prioridade do projeto.
    - **Modelo Pydantic (Request Body):** N/A
    - **Modelo Pydantic (Response Body):** `List[ProjectResponse]`
    - **Status de Retorno Sucesso:** `200 OK`
    - **Retorno Esperado (Sucesso):** Uma lista de objetos de projeto.
    - **Status de Retorno Erro:** `422 Unprocessable Entity` (se os parâmetros de query forem inválidos).

2. **Criar novo projeto**

    - **Método HTTP:** `POST`
    - **URI:** `/projects`
    - **Descrição:** Cria um novo projeto.
    - **Parâmetros (Path):** N/A
    - **Modelo Pydantic (Request Body):** `ProjectCreate`
    - **Modelo Pydantic (Response Body):** `ProjectResponse`
    - **Status de Retorno Sucesso:** `201 Created`
    - **Retorno Esperado (Sucesso):** O objeto do projeto recém-criado.
    - **Status de Retorno Erro:** `422 Unprocessable Entity` (se os dados de entrada forem inválidos).

3. **Obter detalhes de um projeto específico**

    - **Método HTTP:** `GET`
    - **URI:** `/projects/{project_id}`
    - **Descrição:** Retorna os detalhes de um projeto específico pelo seu ID.
    - **Parâmetros (Path):**
        - `project_id: uuid.UUID`: ID do projeto a ser recuperado.
    - **Modelo Pydantic (Request Body):** N/A
    - **Modelo Pydantic (Response Body):** `ProjectResponse`
    - **Status de Retorno Sucesso:** `200 OK`
    - **Retorno Esperado (Sucesso):** O objeto do projeto solicitado.
    - **Status de Retorno Erro:** `404 Not Found` (se o projeto não existir), `422 Unprocessable Entity` (se o ID for inválido).

4. **Atualizar um projeto existente**

    - **Método HTTP:** `PUT`

- **URI:** `/projects/{project_id}`
- **Descrição:** Atualiza os dados de um projeto existente. Espera-se que todos os campos editáveis sejam fornecidos.
- **Parâmetros (Path):**
  - `project_id: uuid.UUID`: ID do projeto a ser atualizado.
- **Modelo Pydantic (Request Body):** `ProjectUpdate` (ou `ProjectCreate` se a intenção é uma substituição completa dos campos mutáveis)
- **Modelo Pydantic (Response Body):** `ProjectResponse`
- **Status de Retorno Sucesso:** `200 OK`
- **Retorno Esperado (Sucesso):** O objeto do projeto atualizado.
- **Status de Retorno Erro:** `404 Not Found` (se o projeto não existir), `422 Unprocessable Entity` (se os dados de entrada ou o ID forem inválidos).

5. **Deletar um projeto**

- **Método HTTP:** `DELETE`
- **URI:** `/projects/{project_id}`
- **Descrição:** Remove um projeto do sistema.
- **Parâmetros (Path):**
  - `project_id: uuid.UUID`: ID do projeto a ser deletado.
- **Modelo Pydantic (Request Body):** N/A
- **Modelo Pydantic (Response Body):** N/A
- **Status de Retorno Sucesso:** `204 No Content`
- **Retorno Esperado (Sucesso):** Corpo da resposta vazio.
- **Status de Retorno Erro:** `404 Not Found` (se o projeto não existir), `422 Unprocessable Entity` (se o ID for inválido).
