Planejamento do Backend para o Sistema de Gerenciamento de Biblioteca

Visão Geral

O sistema de gerenciamento de biblioteca será composto por quatro módulos principais: Autores, Livros, Locatários e Alugueis. Cada módulo terá funcionalidades específicas para cadastro, atualização, exclusão, e listagem de dados, seguindo as regras de negócio estabelecidas.

Módulo de Autores

Funcionalidades

1. **Cadastro de Autor**
    - A API deve expor um endpoint POST para receber um JSON com os seguintes campos obrigatórios:
        - Nome.
        - Ano de Nascimento.
        - CPF, que deve ser único e validado.
    - Campos opcionais:
        - Sexo.
    - Validação para que o CPF seja único no sistema.
2. **Atualização de Autor**
    - A API deve expor um endpoint PUT para atualização de todos os campos, exceto CPF.
    - O CPF não pode ser alterado após o cadastro.
3. **Exclusão de Autor**
    - A API deve expor um endpoint DELETE para a remoção de um autor a partir de seu ID.

- O Autor só pode ser excluído se não houver livros associados.

4. Listagem ded Autores
    - A API deve expor um endpoint GET que retorne o Autor pelo seu nome.
    - A API deve expor um endpoint GET que retorne o Autor pelo seu ID.
    - A API deve expor um endpoint GET que retorne todos os Autores do sistema.

Estrutura de Story para Implementação

- **História de Usuário: Cadastro de Autor**
    - *Como usuário do sistema de biblioteca, eu quero cadastrar novos autores, para que eu possa associá-los a livros existentes ou futuros.*
    - **Critérios de Aceitação:**
        - O JSON de cadastro deve contemplar todos os campos obrigatórios.
        - O CPF deve ser único e deve estar no tamanho correto (11 dígitos, indiferente a caracteres especiais) para que seja aceito.
        - Caso o CPF informado já exista, a API deve retornar um 400 com a seguinte mensagem: "Já existe um autor registrado para o CPF '{cpf}'".
        - Caso um CPF esteja em formato incorreto, a API deve retornar um 400 com a seguinte mensagem: "CPF inválido. Por favor cadastre um CPF com 11 dígitos.'".
        - A API deve aceitar CPF tanto com caracteres especiais ('.' e '-') quanto apenas os dígitos.
        - Para salvar um CPF no banco, todos os caracteres que não forem numéricos deverão ser removidos. Exemplo: o cpf '987.654.321-00' será salvo no banco como '98765432100'.
        - A API deve impedir o cadastro de autores com dados incompletos, destacando os campos obrigatórios não fornecidos.
        - Caso o nome do Autor informado já exista, a API deve retornar um 400 com a seguinte mensagem: "Autor já cadastrado para o nome

‘{nome}’”.

- **História de Usuário: Atualização de Autor**
  - *Como usuário do sistema, eu quero atualizar as informações dos autores existentes, para que as informações estejam sempre corretas e atualizadas.*
  - **Critérios de Aceitação:**
    - O JSON de autalização pode conter um ou mais campos para serem atualizados.
    - Caso seja informado apenas um campo, os demais que não foram informados não devem ter suas informações alteradas.
    - Não deve ser permitido alterar o CPF do Autor.
    - As alterações devem ser salvas somente após a validação completa dos dados.
    - Deve ser realizada a mesma validação para o nome do Autor que é feita no cadastro, não sendo possível atualizar o nome de um autor para outro que já tenha no sistema.
- **História de Usuário: Exclusão de Autor**
  - *Como usuário do sistema, eu quero excluir autores que não têm livros associados, para que eu mantenha o sistema organizado e atualizado.*
  - **Critérios de Aceitação:**
    - O sistema deve verificar a existência de livros associados antes de permitir a exclusão.
    - Caso o autor que está sendo excluído possua livros associados, a API deve retornar um 400 com a seguinte mensagem: “Não foi possível remover o autor de ID ‘{id}’ pois os livros dos seguintes ID’s estão associados a ele: {inserir uma lista com os IDs dos livros que ele é o autor}”.
    - Caso não seja localizado nenhum autor com o ID informado, a API deve retornar um 404 com a seguinte mensagem: “Não foi encontrado nenhum autor para o ID: #{id}”.

- **História de Usuário: Listagem de Autores**

- *Como usuário do sistema, eu quero buscar autores pelo nome, para que eu possa encontrar rapidamente o autor que estou procurando.*
- **Critérios de Aceitação:**
  - A API deve retornar o autor de acordo com o nome informado no corpo da requisição.
  - O usuário não precisa passar o nome completo do autor para encontrá-lo. Por exemplo: caso eu esteja buscando o autor 'George Orwell' mas no corpo da requisição eu informe apenas 'George', a API deve retornar todos os autores que se chamem 'George', independente do seu sobrenome.
  - Caso nenhum autor seja encontrado para o nome informado, a API deve retornar um 404 com a seguinte mensagem: "Não foi encontrado nenhum autor com o nome '{nome}'".
  - A API deve retornar o autor de acordo com o ID informado no path da requisição.
  - Caso nenhum autor seja encontrado para o ID informado, a API deve retornar um 404 com a seguinte mensagem: "Não foi encontrado nenhum autor para o ID: #'{id}'".
  - A API deve retornar todos os autores cadastrado no sistema.

Módulo de Livros

Obs.: As categorias de livros serão definidas de forma estática, não sendo necessário implementar um endpoint para gerenciá-las

Funcionalidades

1. **Cadastro de Livro**

- A API deve expor um endpoint POST para receber um JSON com os seguintes campos obrigatórios:
  - Nome.
  - ISBN, que é tipo um RG dos livros.
  - Data de Publicação, que deve estar no formato YYYY-MM-DD.

    - Autores, que deve ser uma lista contendo o ID de cada autor daquele livro.
    - Categoria, que representa o gênero do livro.
  - Validação para que ISBN seja válido.

2. **Atualização de Livro**
   - A API deve expor um endpoint PUT para atualização de todos os campos.
3. **Exclusão de Livro**
   - A API deve expor um endpoint DELETE para a remoção de um livro a partir de seu ID.
   - O livro só pode ser excluído se não estiver alugado
4. **Listagem de Livros**
   - A API deve expor um endpoint GET para listagem de todos os livros.
   - A API deve expor um endpoint GET que retorne um determinado livro de acordo com o ID informado.
   - A API deve expor um endpoint GET que retorne livros de acordo com o título informado.
   - A API deve expor um endpoint GET que retorne um determinado livro de acordo com a categoria informada.
   - A API deve expor um endpoint GET que retorne livros de acordo com o ID do Autor que foi informado.

Estrutura de Story para Implementação

- **História de Usuário: Cadastro de Livro**
  - *Como usuário do sistema de biblioteca, eu quero cadastrar novos livros, para que possam ser disponibilizados para aluguel ou consulta no sistema.*
  - **Critérios de Aceitação:**
    - O JSON de cadastro deve contemplar todos os campos obrigatórios.

        - Caso seja informado o ID de algum autor que não exista no sistema, a API deve retornar um 404 com a seguinte mensagem: “Não foi localizado nenhum autor para o ID: #{id}”
        - Os livros terão um campo denominado STATE que será um enum com os valores DISPONÍVEL (0) e ALUGADO (1). Por padrão, todos os livros quando cadastrados terão o state = 0.
        - O ISBN deve ser verificado para garantir o formato correto (13 dígitos, indiferente a caracteres especiais) para que seja aceito.
        - Caso o ISBN informado esteja em um formato incorreto, a API deve retornar um 400 com a seguinte mensagem: “ISBN inválido. Por favor cadastre um livro com ISBN de 13 dígitos.”.
        - A API deve aceitar ISBN tanto com caracteres especiais (’-’) quanto apenas os dígitos.
        - Para salvar um ISBN no banco, todos os caractéres que não forem numéricos deverão ser removidos. Exemplo: o ISBN ‘978-85-3591-484-9’ será salvo no banco como ‘9788535914849’.
        - O sistema deve impedir o cadastro de livros com dados incompletos, destacando os campos obrigatórios não preenchidos.
        - Caso a categoria informada não esteja dentro da lista de categorias pré-definidas, a API deve retornar um 400 com a seguinte mensagem: “A categoria do livro deve ser uma das seguintes opções: {citar as categorias que ainda serão definidas”.
- **História de Usuário: Atualização de Livro**
    - *Como usuário do sistema, eu quero atualizar as informações dos livros existentes, para que as informações estejam sempre corretas e atualizadas.*
    - **Critérios de Aceitação:**
        - O JSON de autalização pode conter um ou mais campos para serem atualizados.
        - Não deve ser possível atualizar o state de um livro via endpoint PUT. Essa funcionalidade será automática quando o livro for devolvido no aluguel.

        - Caso seja informado apenas um campo, os demais que não foram informados não devem ter suas informações alteradas.
        - A validação do ISBN deve ser a mesma que no cadastro de novos livros, tendo em vista que o ISBN pode ser alterado.
        - As alterações devem ser salvas somente após a validação completa dos dados.
- **História de Usuário: Exclusão de Livro**
    - *Como usuário do sistema, eu quero excluir livros que não estão alugados, para que eu mantenha o sistema organizado e atualizado.*
    - **Critérios de Aceitação:**
        - O sistema deve verificar se o livro não está alugado, antes de excluí-lo
        - Caso o livro que está sendo excluído esteja relacionado a um aluguel ativo, a API deve retornar um 400 com a seguinte mensagem: “Não foi possível remover o livro pois ele está alugado no aluguel de ID: #{id}”
- **História de Usuário: Listagem de Livros**
    - *Como usuário do sistema, eu quero visualizar uma lista de livros disponíveis e alugados, para que eu possa gerenciar os livros e entender a sua disponibilidade.*
    - **Critérios de Aceitação:**
        - O endpoint de listagem de todos os livros, deve ter como opção um Query Param nomeado “state” onde será passado ‘0’ para filtrar apenas os livros disponíveis ou ‘1’ para filtrar apenas livros alugados.
        - Caso não seja informado o state, todos os livros do banco de dados devem ser retornados.
        - Deve ser possível buscar livros pelo título. A busca se assemelha com a busca de autores pelo nome, onde não é necessário informar todo o título do livro para encontrá-lo.
        - Deve ser possível buscar um livro pelo ID.
        - Deve ser possível buscar livros pela categoria.

        - Caso nenhum livro seja encontrado (tanto por ID quanto por título), a API deve retornar um 404 com a seguinte mensagem: “Não foi localizado nenhum livro para o {título ou ID} ‘{título ou ID}’ “.
        - Deve ser possível buscar livros de acordo com o ID do Author informado, retornando todos os livros de um autor específico.
        - Os filtros devem funcionar juntos, ou seja, deve ser possível buscar, por exemplo, os livros disponíveis de um Autor de ID X e que tenham o título Y e seja da categoria Z. (verificar o design pattern Specification)

Módulo de Locatários

Funcionalidades

1. **Cadastro de Locatário**
    - A API deve expor um endpoint POST para receber um JSON com os seguintes campos obrigatórios:
        - Nome.
        - Telefone.
        - Email, que deve ser único.
        - Data de Nascimento, que deve seguir o formato YYYY-MM-DD
        - CPF, que deve ser único.
    - Campos opcionais:
        - Sexo.
    - Validação para que o CPF seja único no sistema.
    - Validação para que o email seja único no sistema.
    - Validação no formato da data.
2. **Atualização de Locatário**
    - A API deve expor um endpoint PUT para atualização de todos os campos, exceto o CPF.
    - O CPF não pode ser alterado após o cadastro.
3. **Exclusão de Locatário**

- A API deve expor um endpoint DELETE para a remoção de um locatário a partir de seu ID.
- O locatário só pode ser excluído se não tiver alugueis pendentes de devolução.

4. **Listagem de Locatários**
   - A API deve expor um endpoint GET para visualizar uma lista de locatários.
   - A API deve expor um endpoint GET para buscar um locatário pelo ID.
5. **Listagem de Aluguéis do Locatário**
   - A API deve expor um endpoint GET para visualizar uma lista de todos os aluguéis feitos por um locatário específico a partir de seu ID.
   - A lista deve incluir informações detalhadas sobre cada aluguel, como a data de início, a data de devolução e o estado do aluguel.

Estrutura de Story para Implementação

- **História de Usuário: Cadastro de Locatário**
  - *Como usuário do sistema de biblioteca, eu quero cadastrar novos locatários com informações detalhadas, para que eu possa gerenciar os empréstimos de forma eficiente.*
  - **Critérios de Aceitação:**
    - O JSON de cadastro deve incluir todos os campos obrigatórios.
    - Ao tentar salvar um locatário com dados incompletos, a API deve impedir o envio e destacar os campos que precisam ser preenchidos.
    - O CPF e o email devem ser únicos e não devem permitir duplicidade no sistema.
    - Caso o CPF informado já exista, a API deve retornar um 400 com a seguinte mensagem: “Já existe um locatário registrado para o CPF ‘{cpf}’”.
    - Caso o email informado já exista, a API deve retornar um 400 com a seguinte mensagem: “Já existe um locatário registrado para o email ‘{email}’”.

        - Caso um CPF esteja em formato incorreto, a API deve retornar um 400 com a seguinte mensagem: “CPF inválido. Por favor cadastre um CPF com 11 dígitos.’”.
        - A API deve aceitar CPF tanto com caracteres especiais (‘.’ e ‘-’) quanto apenas os dígitos.
        - Para salvar um CPF no banco, todos os caracteres que não forem numéricos deverão ser removidos. Exemplo: o cpf ‘987.654.321-00’ será salvo no banco como ‘98765432100’.
        - A API não deve aceitar emails inválidos (sem @ ou com caracteres não suportados).
        - Semelhante ao CPF, a api deve aceitar numeros de telefone com caracteres especiais, porém ao salvar no banco deve removê-los. Por exemplo, o telefone ‘(51) 99999-0990’ será salvo no banco como ‘51999990990’.
        - A data de nascimento deve estar no formato YYYY-MM-DD.
        - Deve haver uma validação para a formatação da data, caso a data informada não esteja no formato correto a API deve retornar um 400 com a seguinte mensagem: “Formato de data incorreto. Favor informar uma data no formato ‘YYYY-MM-DD’.
- **História de Usuário: Atualização de Locatário**
    - *Como usuário do sistema, eu quero atualizar os dados dos locatários existentes, para que as informações estejam sempre corretas e atualizadas.*
    - **Critérios de Aceitação:**
        - O JSON de atualização pode conter um ou mais campos para serem atualizados.
        - Caso seja informado apenas um campo, os demais que não foram informados não devem ter suas informações alteradas.
        - O CPF não pode ser alterado após o cadastro.
        - Todas as validações para os campos realizadas no endpoint de cadastro, devem ser seguidas neste endpoint de atualização.
        - Alterações devem ser salvas somente após passar por todas as validações.

- **História de Usuário: Exclusão de Locatário**
  - *Como usuário do sistema, eu quero excluir locatários que não têm livros pendentes, para que eu mantenha o sistema organizado e atualizado.*
  - **Critérios de Aceitação:**
    - O sistema deve verificar se há alugueis pendentes antes de permitir a exclusão.
    - Caso o locatário que está sendo excluído possua alugueis pendentes, a API deve retornar um 400 com a seguinte mensagem: “Não foi possível remover o locatário de ID ‘{id}’ pois os alugueis dos seguintes ID’s estão pendentes de devolução: {inserir uma lista com os IDs dos alugueis pendentes}”.
- **História de Usuário: Listagem de Locatários**
  - *Como usuário do sistema, eu quero ver uma lista dos locatários registrados para que eu possa gerenciar os cadstros.*
  - **Critérios de Aceitação:**
    - Deve ser possível buscar todos os locatários do sistema.
    - Deve ser possível buscar por apenas um locatário informando o seu ID.
    - Caso seja buscado um locatário por um ID não registrado no banco, a API deve retornar um 404 com a seguinte mensagem: “Não foi possível localizar nenhum locatário para o ID: #{id}”.
- **História de Usuário: Listagem de Aluguéis do Locatário**
  - *Como locatário, eu quero visualizar meu perfil e meu histórico de empréstimos, para que eu possa acompanhar minhas atividades na biblioteca.*
  - **Critérios de Aceitação:**
    - Deve ser possível buscar todos os alugueis de um determinado locatário pelo seu ID.

Módulo de Aluguéis

Funcionalidades

1. **Cadastro de Aluguel**
    - A API deve expor um endpoint POST para criar um novo aluguel.
    - O JSON de requisição deve incluir os seguintes campos obrigatórios:
        - ID do Locatário (associado ao aluguel).
        - Lista de IDs dos Livros (um ou mais livros) que estão sendo alugados.
    - A data de retirada pode ser informada. Caso não seja, deve ser considerado o dia atual.
    - A data de devolução pode ser informada. Caso não seja, deve ser considerado por padrão para dois dias após a data de retirada.
2. **Retorno de Aluguel**
    - A API deve expor um endpoint PUT para retornar todos os livros de um aluguel a partir de seu ID.
    - O registro do aluguel retornado deverá permanecer na base de dados, sendo alterado apenas um campo booleano para mantermos o histórico. Por exemplo, um campo 'isReturned'.
3. **Listagem de Aluguéis**
    - A API deve expor um endpoint GET para listar todos os aluguéis.
    - A API deve expor um endpoint GET para listar um aluguel de acordo com o seu ID.

Estrutura de Story para Implementação

- **História de Usuário: Cadastro de Aluguel**
    - *Como usuário do sistema de biblioteca, eu quero registrar novos aluguéis de livros para os locatários, para que eles possam usufruir dos livros disponíveis.*
    - **Critérios de Aceitação:**
        - O JSON de cadastro deve incluir todos os campos obrigatórios.
        - Caso não sejam informadas as datas de retirada e de devolução, a API deve setar a data de retirada como o dia atual e a data de devolução como 2 dias após a retirada.

        - Deve verificar a disponibilidade dos livros antes de confirmar o aluguel.
        - Caso um ou mais livros estejam indisponíveis, a API deve retornar um 400 com uma mensagem clara indicando os livros indisponíveis.
        - Caso um ou mais livros não forem encontrados, a API deve retornar um 404 com uma mensagem clara indicando quais livros que não foram encontrados.
        - Caso seja informado o ID de um locatário inexistente, a API deve retornar um 404 com a seguinte mensagem: “Não foi localizado um locatário para o ID: #{id}”.
- **História de Usuário: Retorno de Aluguel**
    - *Como usuário do sistema, eu quero retornar os livros de um determinado aluguel, para que os livros voltem a estar disponíveis.*
    - **Critérios de Aceitação:**
        - Verificar se o ID informado no Path é de um aluguel existente.
        - Caso não exista um aluguel com o ID informado, a API deve retornar um 404 com a seguinte mensagem: “Não foi localizado um aluguel com o ID: #{id}”.
        - Após o retorno do aluguel, todos os livros daquele aluguel devem estar novamente disponíveis e o aluguel definido como retornado (campo booleano deve ser atualizado).
        - Alterações devem ser salvas somente após a validação completa dos dados.
- **História de Usuário: Listagem de Aluguéis**
    - *Como usuário do sistema, eu quero ver uma lista de todos os aluguéis. Quero poder filtrar por ativos e inativos, para que eu possa gerenciar melhor os retornos e a disponibilidade dos livros.*
    - **Critérios de Aceitação:**
        - O endpoint de listagem de todos os alugueis, deve ter como opção um filtro onde deve ser informado a busca por alugueis já devolvidos ou alugueis que ainda estão pendentes.

- Caso não seja informado o filtro, todos os alugueis do banco de dados deve ser retornado.
- Deve ser possível buscar um aluguel pelo seu ID.
- Caso nenhum aluguel seja encontrado pelo ID, a API deve retornar um 404 com a seguinte mensagem: “Não foi localizado nenhum aluguel para o ID: # {id} “.