Software Design Document

Projeto: **Biblioteca universitária**

Data	05/08/2024
Responsável	Igor Campos de Borba
Autor	Igor Campos de Borba
Doc ID	1
Localização	C:\Users\igor_\OneDrive\Documentos\INFNET\3 bimestre\1- Desenvolvimento de aplicações Java com Spring Boot\Trabalho
Versão do template	0.0.1

Histórico de revisão

Data	Versão	Autor	Descrição
05/08/2024	0.0.1	Igor Campos de Borba	Redação de: - Descrição do sistema - Público e Stakeholders - Regras de negócio - Requisitos funcionais e não funcionais - Limitações de software - Visão funcional e visão técnica - Definição de linguagens - Resumo da arquitetura de software Diagramas e ferramentas administrativas para análise de requisitos: - Descrição de História de usuário - Diagrama de Cenários Diagramas para: - Back-end - Diagramas UML: de classes e de objetos. - Banco de dados - Diagrama entidade-relacionamento - Wireframe de interface (UI)

Revisor

Nome	Papel	Data
Igor Campos de Borba	Desenvolvedor de software	05/08/2024

Sumário

Introdução

Sistema de gerenciamento de biblioteca universitária. O projeto visa ao desenvolvimento de um sistema para registrar empréstimos de livros e de multas. O público-alvo são estudantes e colaboradores (professores, técnicos, etc).

Projeto disponível no GitHub:

Nota: no atual momento (05/08) o desenvolvedor de software pré-implementou o JPA.

Público e stakeholders

Públicos-alvo:
Entende-se por públicos-alvo o consumidor que o projeto almeja atingir.

1. Estudantes: alunos da universidade que desejam pegar um livro emprestado. Empréstimo sujeito a multa de atraso.
2. Professores: funcionários da universidade que desejam pegar um livro emprestado. Não estão sujeitos a multa de R$ 4 por dia.
3. Administrador do sistema: profissional que consulta os livros e clientes da biblioteca.

Stakeholders:
Entende-se por stakeholder funcionários e investidores envolvidos no projeto.

1. Administrador do sistema.
2. Desenvolvedor de software.

Stakeholders – descrições e responsabilidades

No âmbito do cliente, pode-se citar:

Quadro 1 – Stakeholders – descrições e responsabilidades

Nome	Descrição	Responsabilidade
Administrador do sistema	É quem gerencia os dados no sistema	• Opera o CRUD de clientes da biblioteca. • Entra em contato com quem pegou um livro emprestado.

Fonte: Elaborado pelo autor.

Recursos do software

Estes recursos resumidos compõem as páginas e funcionalidades do sistema que influenciam no back-end e banco de dados.

- **Cadastro de usuários**: estudantes e funcionários para gerenciar no sistema (GET, POST, PUT, DELETE).
- **Empréstimo de livros**: registro do empréstimo num período fechado (GET, POST).
- **Cadastro de livros**: gerenciar os livros no sistema (GET, POST, PUT, DELETE).
- **Biblioteca**: representação da biblioteca que contém os livros. Porta de entrada para gerenciamento.

Visão funcional

Sistema que permite gerenciar livros, usuários, emprestar livros e registrar multas.O sistema obterá os dados do banco de dados e cadastrará dados da biblioteca, livros, usuários, empréstimos e multas dinamicamente.

Visão técnica

BACK-END se comunica com o Front via API REST e com banco de dados via Hibernate (que implementa o padrão da JPA de requisições). Este back-end organiza a informação via paradigma de orientação a objetos para obtenção dos dados do banco de dados e cadastros.
Tecnologias: Java 17 e Spring Boot 3.3.2.

BANCO DE DADOS: O SGBD MySQL Workbench (paradigma relacional) armazena em tabelas (entidades). O Administrador do sistema consulta o banco de dados para verificar se há um novo pedido de adotante de um pet.
Tecnologia: MySQL.

SERVIDOR: hospedagem em servidor compartilhado e gratuito da Railway em sistema operacional Linux, com interface no navegador. Espaço de 1GB em disco, tráfego por hora 500 horas por mês, servidor com 512mb de RAM, CPU com 2 núcleos. Path do endereço .railway.app

Regras de negócio

Quadro 2 – Regras de negócio

ID	Regra de negócio	Descrição
RN1	Catálogo de livros	Os livros podem ser cadastrados em conjunto ou individualmente.
RN2	Cadastro de usuários	Estudantes e associados são registrados no sistema. Apenas estudantes podem ser multados por atraso. Associados são apenas identificados sobre sua atuação na universidade.
RN3	Empréstimo de livros	Livros podem ser emprestados por um período fechado (início e fim), podendo ser renovados quantas vezes necessárias.

Fonte: Elaborado pelo autor.

Precedência e prioridade de caso de uso

Quadro 3 – Precedência e prioridade de caso de uso

Caso de uso	Precedência	Prioridade
Catálogo de livros implementado no back e banco de dados e para os ativos da universidade estarem catalogados.	1	Alta
Cadastro de usuários devem registrar estudantes ou associados, não genericamente somente como usuários.	2	Baixa
Empréstimo de livros para estudantes ou associados.	3	Alta
Cálculo da multa diária registrado no banco de dados.	4	Média

Fonte: Elaborado pelo autor.

Requisitos funcionais

Requisitos funcionais

Quadro 7 – Requisitos funcionais

ID	Requisito funcional	Descrição	Caso de uso resumido
RF1	Catálogo de livros	- Back-end e banco de dados: 1) receber requisição GET, POST, PUT, DELETE para obter/cadastrar/alterar/deletar livros no banco de dados. 2) os livros podem ser gerenciados em conjunto ou individualmente via requisição POST, PUT, DELETE.	Gerenciar informações de livros.
RF2	Cadastro de usuários	- Back-end e banco de dados: 1) receber requisição GET, POST, PUT, DELETE para obter/cadastrar/alterar/deletar usuários (estudantes e associados) no banco de dados. 2) calcular valor da multa em caso de atraso de devolução de livro pelo aluno E atualizar no banco de dados diariamente via Spring Boo.	Gerenciar informações de usuários.
RF3	Empréstimo de livros	Back-end e banco de dados: Receber requisição GET, POST para obter/cadastrar empréstimos de livros.	Gerenciar informações de empréstimos.

Fonte: Elaborado pelo autor.

Escopo não contemplado

O que o software não possui de funcionalidade e alternativas:

- Não será criado o front-end.
- Não terá fila de espera para alugar livros.

Linguagens

Versões LTS para evitar incompatibilidade entre linguagens e incrementos do sistema. O website será feito apenas para desktop.

- SpringBoot 3.3.2 e Java 17
- MySQL 5.7.41
- Angular 14 e TypeScript 4.7.2

Motivo da escolha das linguagens e frameworks.

- SpringBoot e Java: orientada a objetos e configuração simplificada com as annotations e fácil integração com biblioteca Hibernate (mapeamento objeto-relacional).
- MySQL: paradigma relacional proporciona confiabilidade na persistência correta dos dados.
- Angular e TypeScript: orientada a objetos e tipada.

Diagramas

Estrutura do sistema

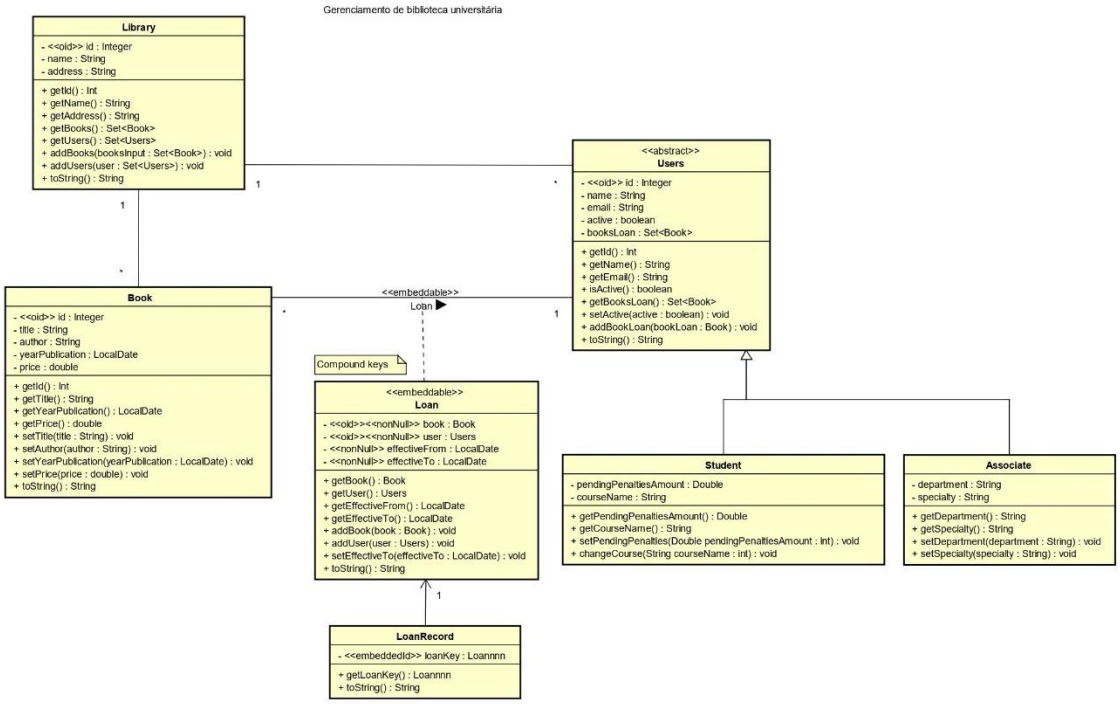
O objetivo destes diagramas é estruturar a arquitetura do software lógica e visual:

Back-end – Diagramas UML de classes e de objetos.

Diagrama UML de classes

Figura 4 – Diagrama de classe: catálogo de pets (GET do HTTP)

Anotações:



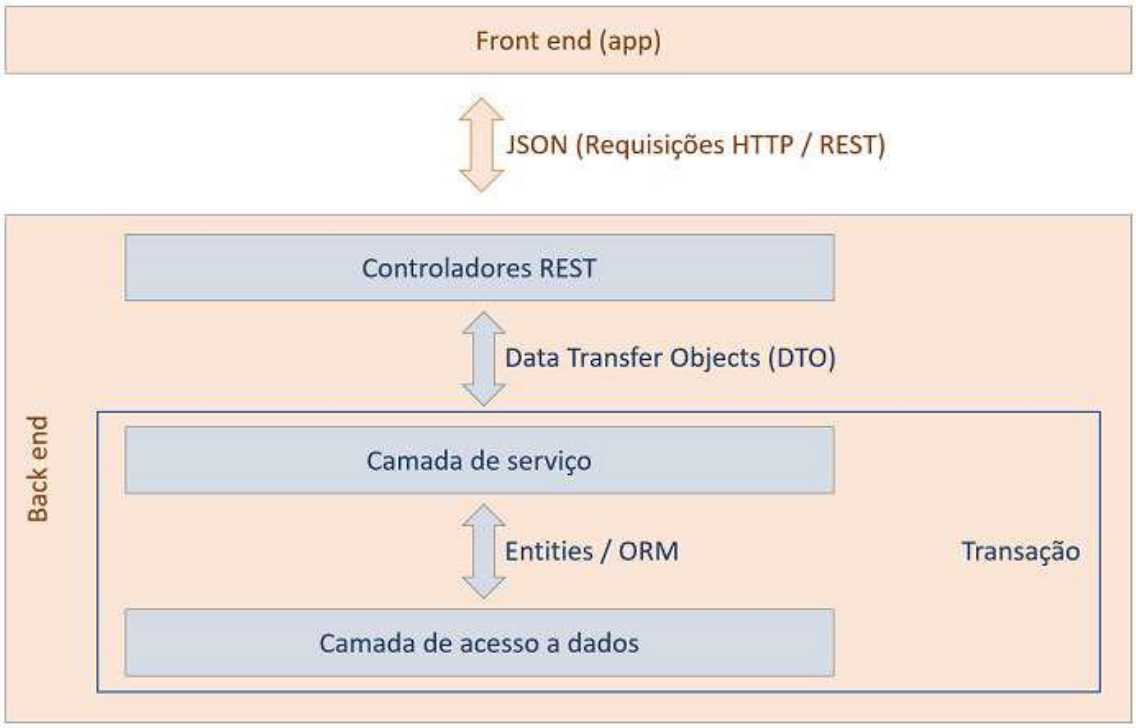
- Users se relaciona com Library porque Library tem livros e usuários para gerenciar.
- Users é uma classe abstrata porque eu quero orientar a identificação específica de qual tipo de usuário será instanciado (Student ou Associate).
- Loan é uma classe associativa para gerenciar os empréstimos de livros (Book) para o usuário (Users). Por ela ser uma chave composta (Book e User), precisa-se de uma classe externa que mantém essa chave composta (LoanRecord) e a annotation embeaddable do JPA para indicar que o gerenciamento da chave primária é pelo LoanRecord e que os seus atributos serão persistidos no banco de dados.
Usou-se uma associação direcionada de LoanRecord para Loan porque apenas LoanRecord precisa se relacionar com Loan para fornecer as chaves primárias.
- Usou-se herança para identificar um Student e um Associate. Apenas o estudante pode receber uma multa por atraso na devolução do livro. O Associado apenas é identificado sobre a sua atuação na universidade (departamento e especialidade).
- o esteriótipo <<oid>> indica chave primária, <<nonNull>> significa atributo obrigatório e que não pode ser nulo no momento da instanciação do objeto.

Fonte: Elaborado pelo autor.

- **BACK-END**: Java usa padrão de projeto que implementa DTO para eu escolher os campos que quero do banco, JPA para mapeamento objeto relacional no lado do Java quando pego um dado do banco, padrão JPA é implementado pelo Hibernate que simplifica requisições SQL porque já estão pré-configuradas. Padrão REST para requisições, usando o protocolo HTTP.
- **BANCO DE DADOS**: MySQL armazena em entidades (tabelas) os dados interagidos com o back-end.
- **BACK-END**: Arquitetura de projeto MVC.

Padrão de 3 camadas (interface, lógica e persistência):

Figura 25 – Padrão de 3 camadas



Fonte: Elaborado pelo autor.

- **JSON**: formato que encapsula objetos com notação JavaScript para transmitir dados entre sistemas. Exemplo: um JSON é retornado como resposta para o GET (do HTTP) do catálogo no front-end.
- **Controller REST (resource)**: expõe os endpoints para receber as requisições do front-end do protocolo HTTP.
- **DTO** (data transfer objects): eu defino quais dados do banco serão transmitidos ao front-end, por exemplo eu posso ignorar um atributo. E separa a camada lógica do service com a apresentação do front.
- **Service**: parte lógica das regras de negócio, por exemplo a manipulação dos dados recebidos sobre os usuários.

- **Entities (classes de domain) / ORM**: Mapeamento objeto-relacional. Mapeia a entity que é um objeto no Java para uma entity relacional do MySQL.
- **Camada de acesso a dados** (**repository**): Java faz a comunicação com o banco de dados por meio da implementação da interface JPARepository para ter acesso a métodos que facilitam o uso de operações básicas de CRUD, como o Create, Read, Update e Delete que eu utilizo para gerenciar livro, usuários e empréstimos.

Planejamento de testes

Teste de classe de equivalência: testar casos válidos e inválidos de inputs e verificar resultados dos outputs.

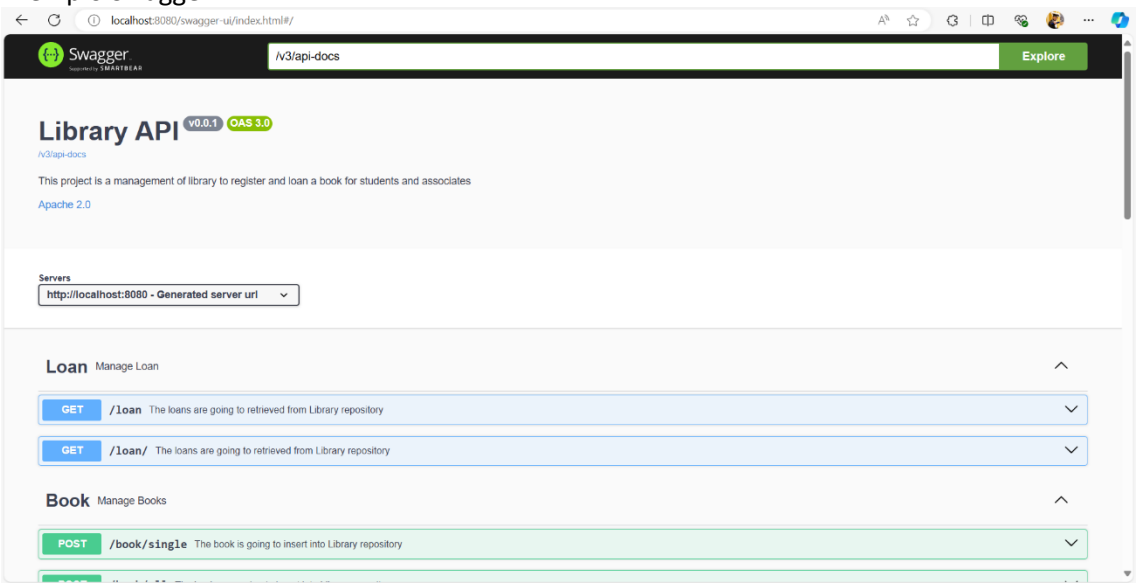
Teste unitário: testar componentes individuais do software com uma cobertura de no mínimo 70%.

Validações:

1) Empréstimo já existente de livro para o mesmo usuário.
2) Sobreposição de empréstimos para o mesmo período.
3) Aviso de back-end fora do ar.
4) Campos obrigatórios. Veja quais são os campos no diagrama de classes da UML.

Readme de execução

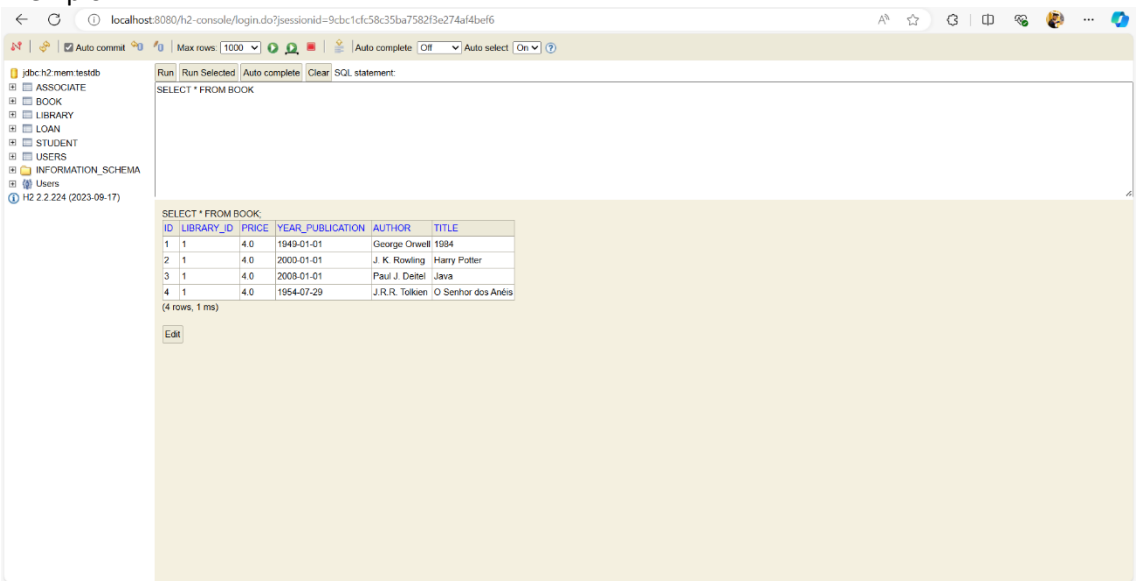
- Testar endpoints pelo Swagger: http://localhost:8080/swagger-ui/index.html
Exemplo Swagger:



- Banco de dados H2 pelo perfil de testes: http://localhost:8080/h2-console

Mantenha estas configurações:
Saved Settings: Generic H2 (Embedded)
Setting name: Generic H2 (Embedded)
Driver class: org.h2.Driver
JDBC URL: jdbc:h2:mem:testdb
User name: as
Password:
* password é em branco

Exemplo H2:



- Testar endpoints pelo postman (colocar url na requisição e texto do body no campo body): OU importar no postman: https://github.com/igorcamposdeborba/library-backend/blob/main/Library.postman_collection.json

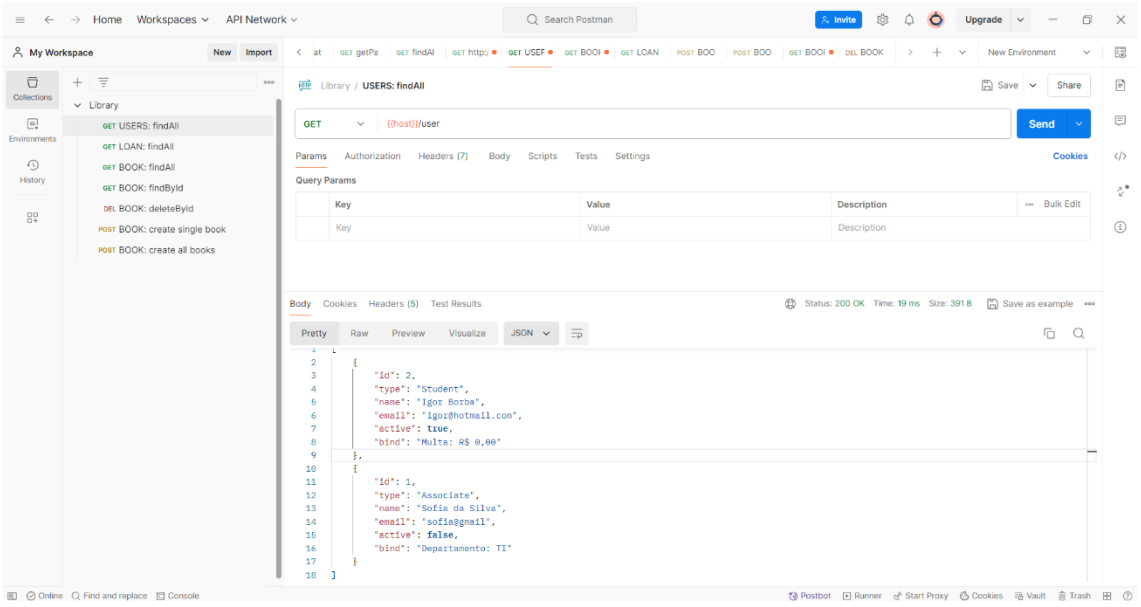
Requisição	URI
GET: BOOKS - findAll	http://localhost:8080/book
GET: USERS - findAll	http://localhost:8080/user
GET: LOAN - findAll	http://localhost:8080/loan

POST: CREATE – criar um livro individualmente	http://localhost:8080/book/single
	
	Body:
	{
	"title": "O Senhor dos Anéis",
	"author": "J.R.R. Tolkien",
	"yearPublication": "1954-07-29",
	"price": 4.00
	}

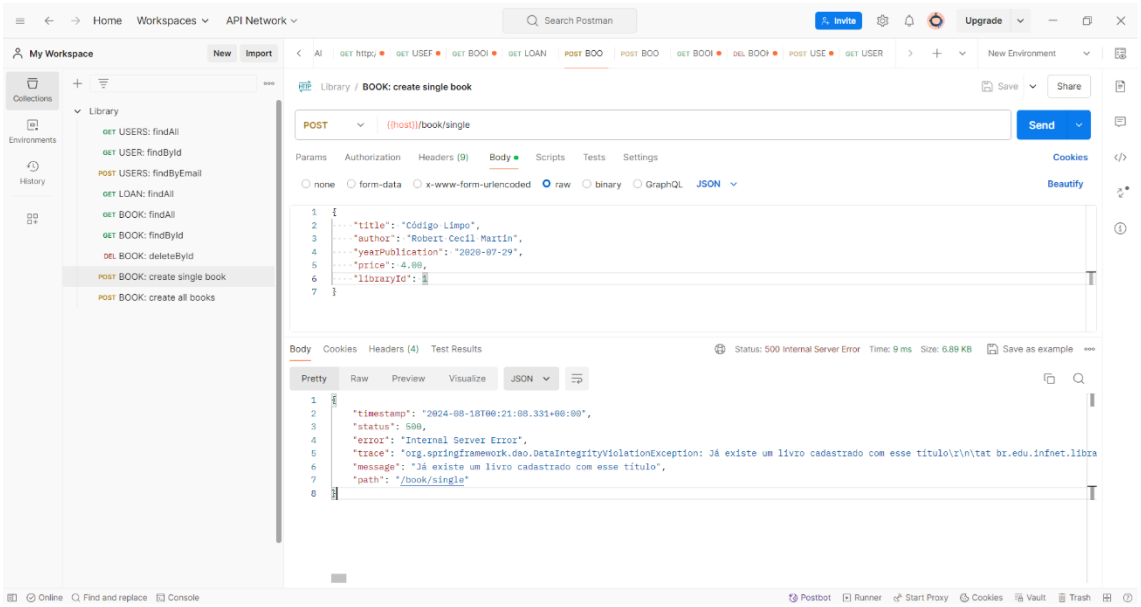
POST: CREATE – criar um livro individualmente	http://localhost:8080/book/all Body: [{ "title": "O Senhor dos Anéis", "author": "J.R.R. Tolkien", "yearPublication": "1954-07-29", "price": 4.00 }, {

	"title": "Administração de Marketing", "author": "Philip Kotler", "yearPublication": "2008-06-25", "price": 4.00 }]

Exemplo Postman – buscar usuários:



Exemplo de exception – livro já cadastrado:



Front-end no Angular:

- Exemplo de livros: