**Documentação Do Projeto**

**Nome: Matheus Santos De Lima Brito**

*Um texto na documentação que contém uma explicação (para iniciantes) sobre as características que tornam essa aplicação um software back-end web?*

-No software tem 4 classes criadas, sao elas a Cliente,Livro,Emprestimo,Consulta. Cada uma delas tem seu proprio id.

-Na classe Cliente no model tem os atributos id, nome, cpf, telefone, email, no pacote banco eu crie Cliente banco onde criei uma List dentro que tinha a classe Cliente, com isso consegue fazer método insert (para voce adicionar um cliente na sua lista), findOne (retorna o cliente), FindAll(retorna todos os clientes) e por ultimo o delete (deleta o cliente).

-Com isso criei dentro do pacote Controller a classe ClienteController, nela eu instanciei o ClienteBanco(cb), e usei para chamar os metodos para organizar esses metodos e chamar na view, os metodos getAllClientes (mostrar os clientes), save (registrar), update (atualiza), delete (deletar o usuario).

-Apos isso no pacote view eu criei o ClienteView, eu só instancie o ClienteController e fiz o GET, PUT, POST e o DELETE chamando os métodos do controller.

-Eu segui assim com todas as classes o mesmo padrao que eu crie o cliente, crie mais 3 banco LivroBanco,EmprestimoBanco e ConsultaBanco

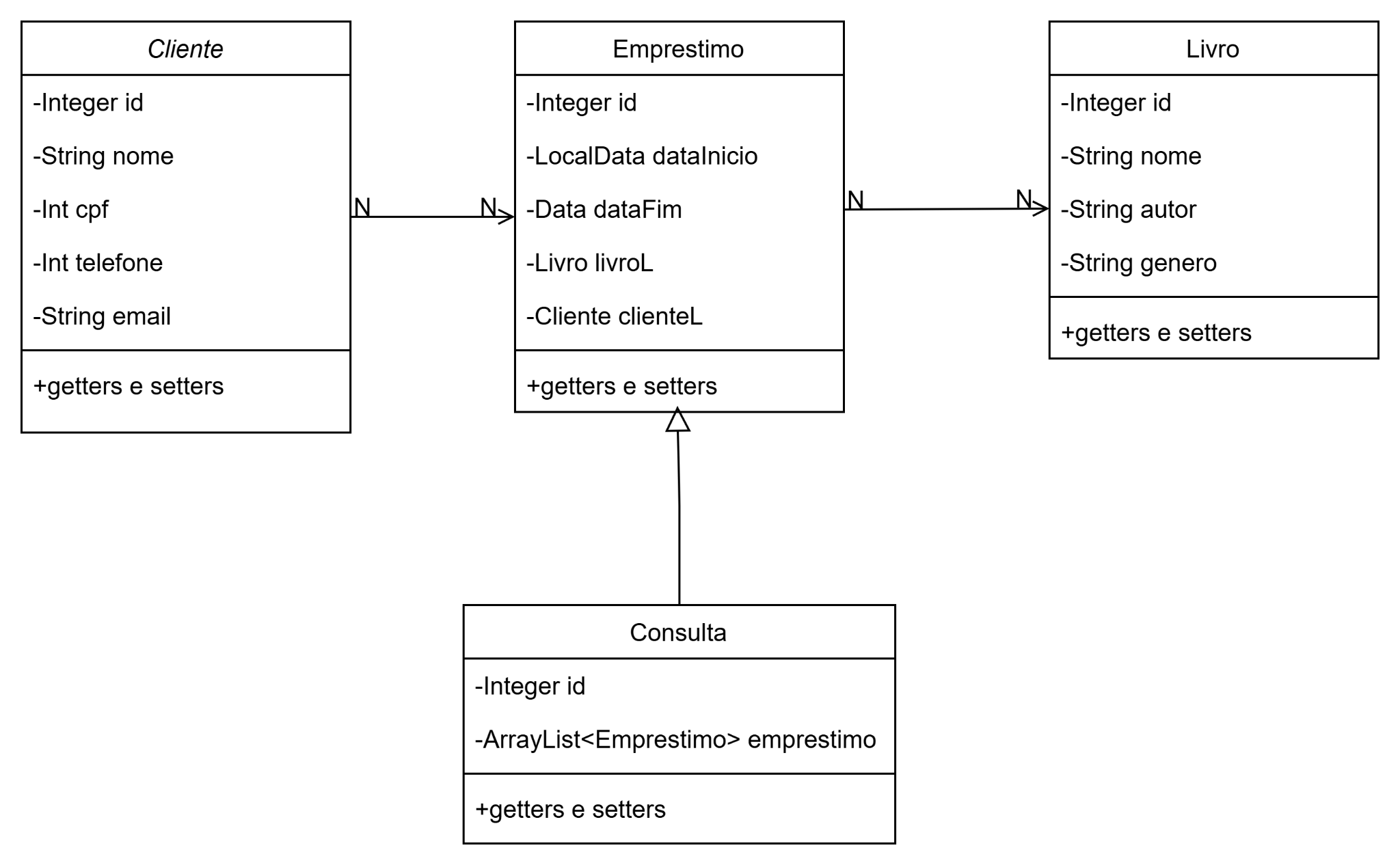
No pacote Controller mais 3 : LivroController, EmprestimoController e Consulta Controller.

No pacote view tambem mais 3 : LivroView, EmprestimoView e ConsultaView.

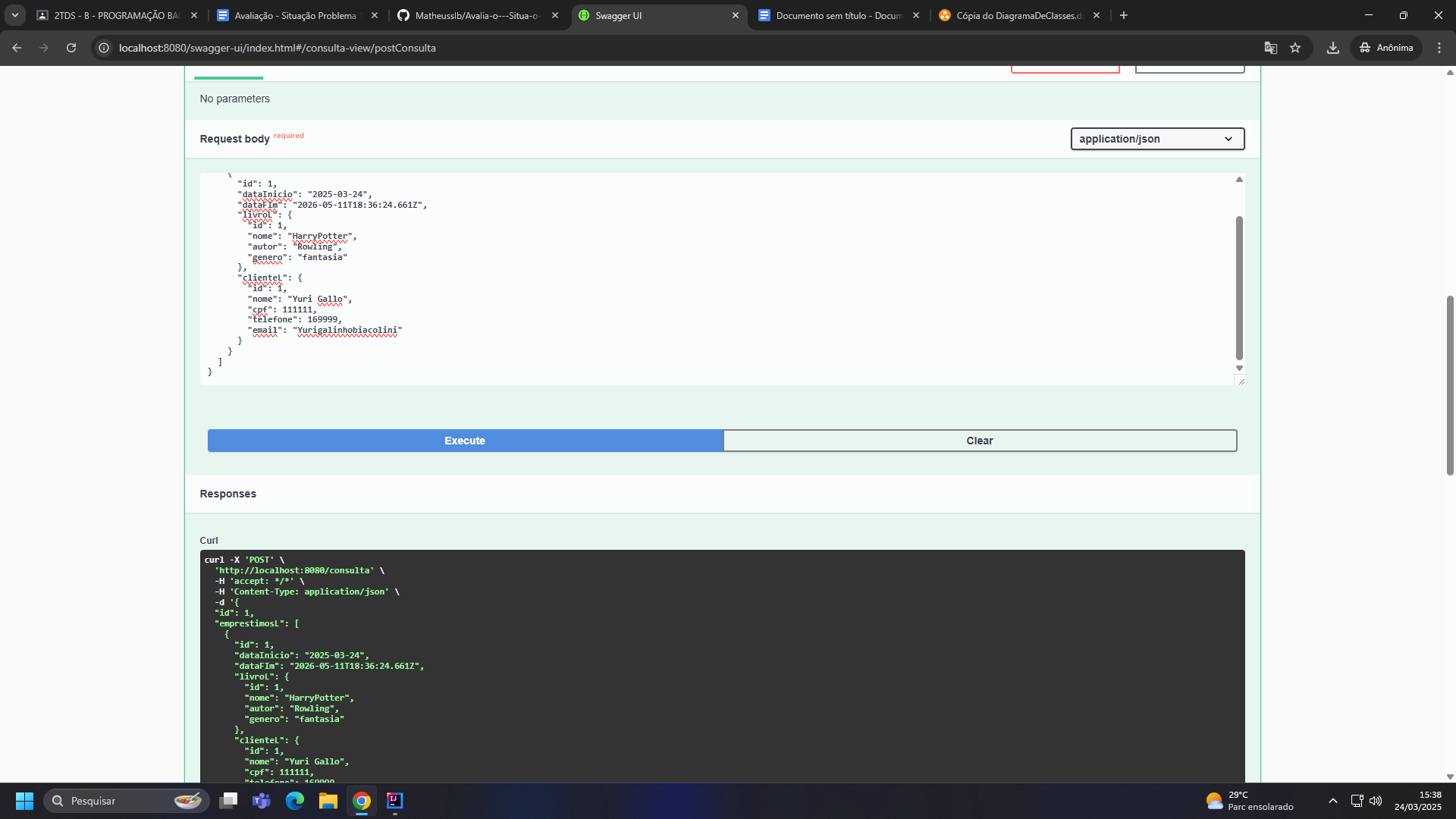
-Em Empréstimo eu fiz que quando registrado, registra o cliente e o livro do cliente junto, e a data de início que seria automática quando fosse registrada, mas a data de fim teria que ser atualizada no put.

-Na Consulta eu passei um ArrayList com emprestimo ja que nele ja esta tudo registrado os dados do cliente e o livro, então quando vc for dar uma consultada no get e so passa o id que ja vai mostar o emprestimo do cliente.

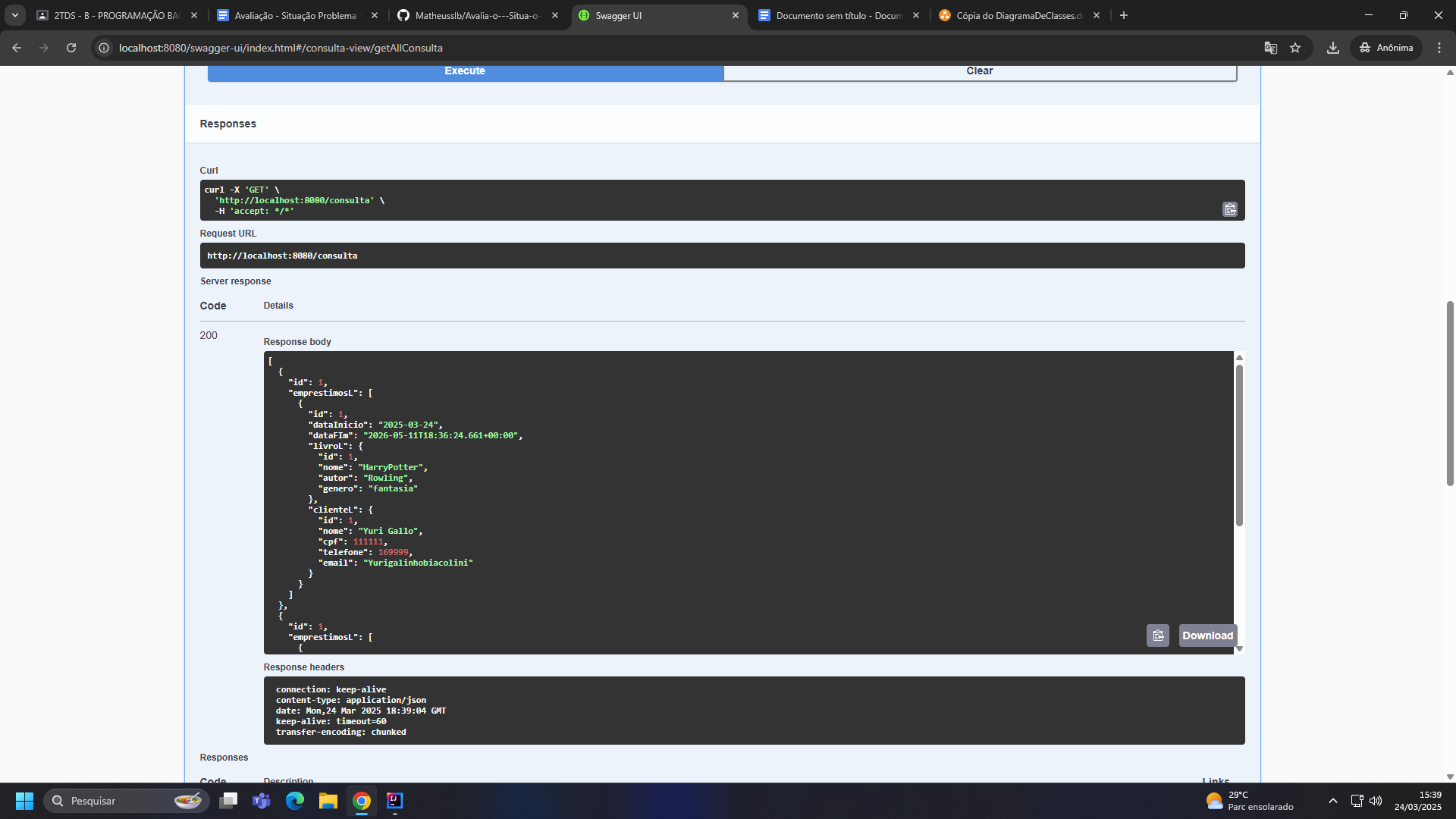
*Diagrama explicando a arquitetura usada (baseada em MVC)*

**

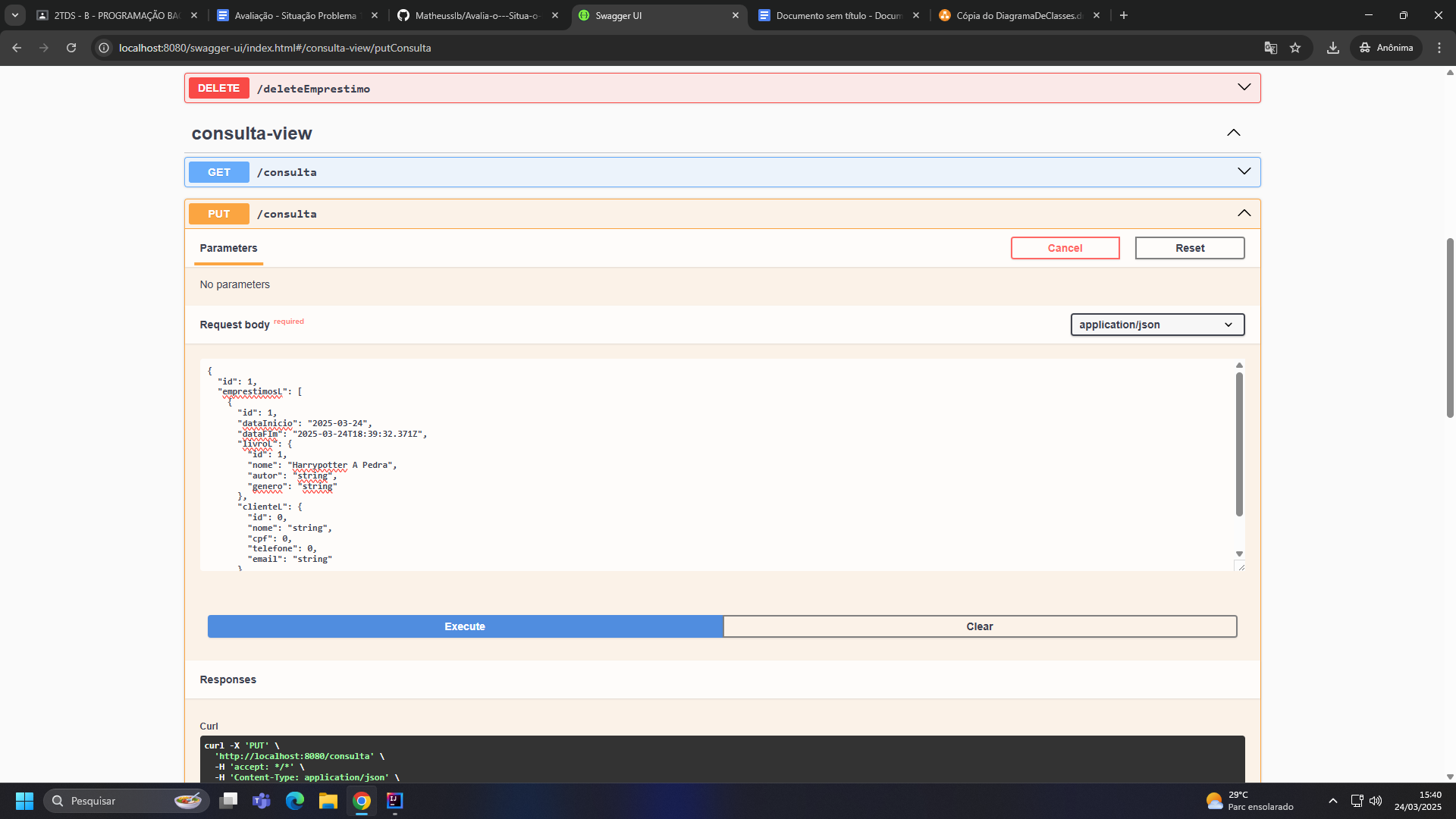
*Um texto contendo uma explicação dos elementos de entrada (endpoints, dados enviados) , processamento e saída para a programação da aplicação e adicionar evidências dessa interação (screenshots).*

**

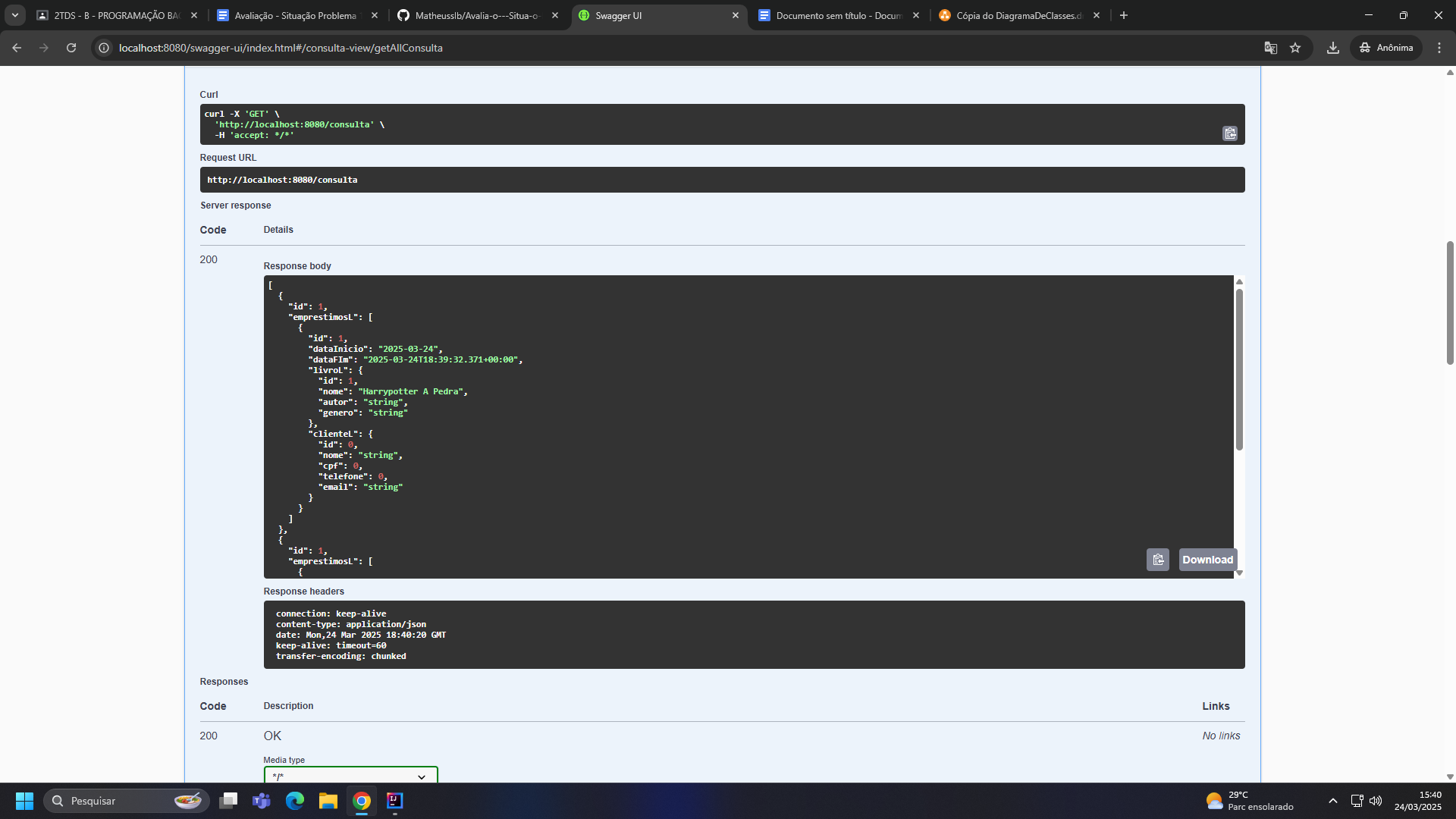
-Aqui eu usei como exemplo a consulta, eu registrei um empréstimo onde o cliente era o Yuri Gallo,e ele está com o livro do Harry Potter.



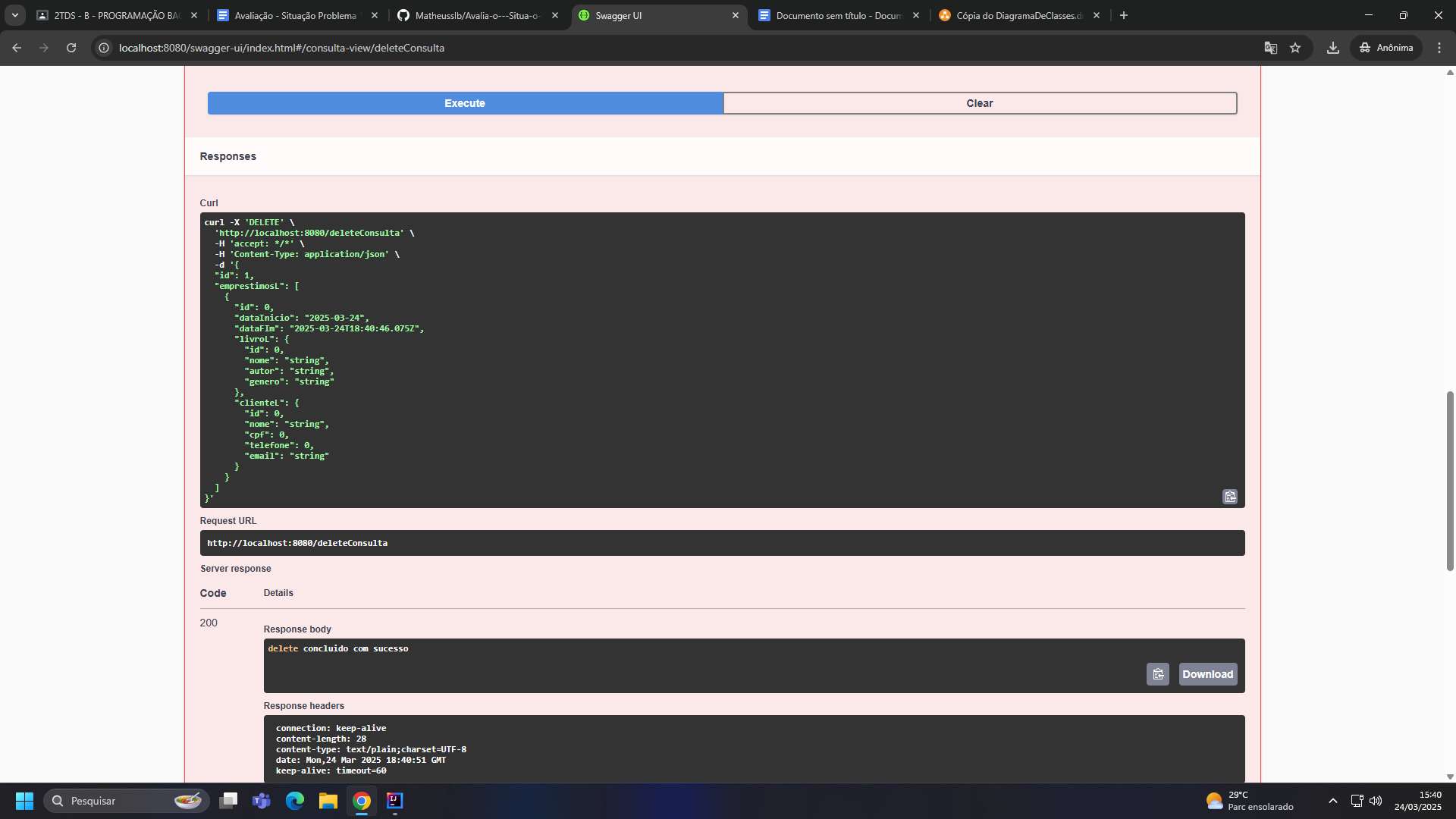
-Após isso eu dei um GET para conferir se ele foi criado mesmo e retornou isso.



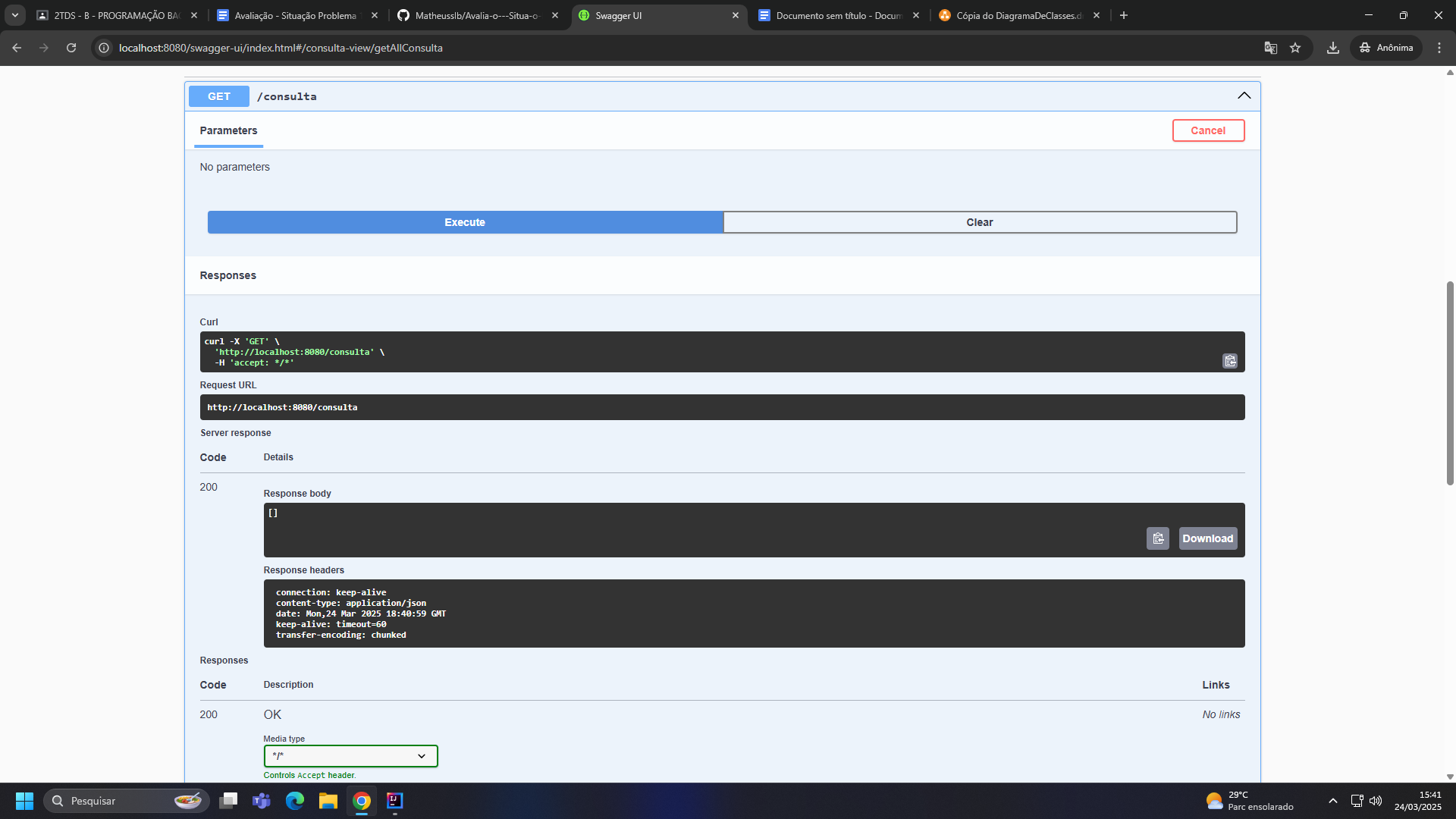
-Com isso eu dei um put que só colocando o id do livro, eu atualizei o nome do livro que o Yuri Gallo estava para Harry Potter a Pedra.



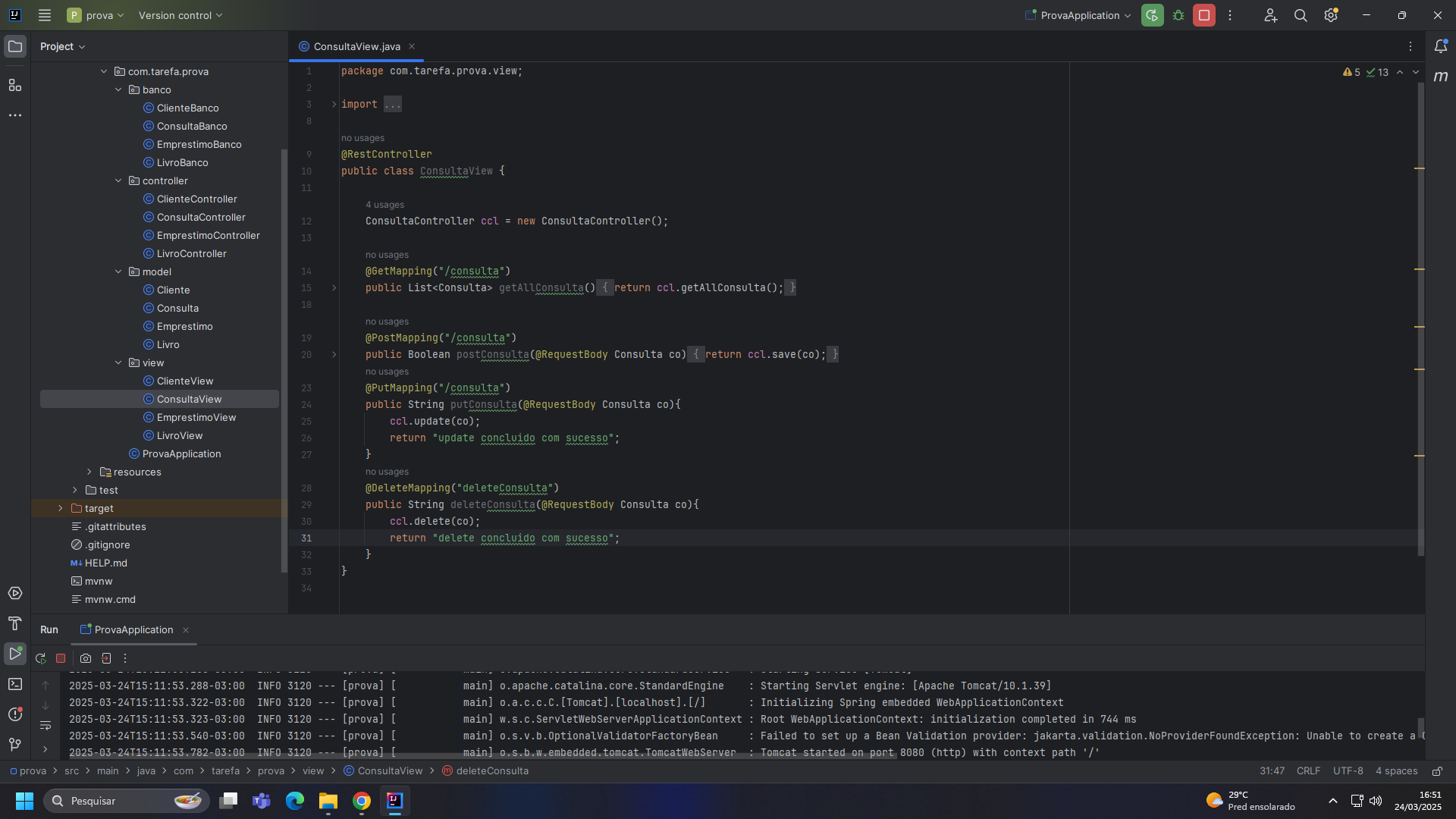
-Executei um GET para ter certeza que a atualização foi concluída, e sim foi atualizado.



-Agora eu queria deletar, eu só passei o id e deletei essa consulta.



-Agora esta aqui um GET que comprova que a consulta foi deletada.



-Assim que ficou em código.

*Um texto contendo uma lista de quais padrões de design existem e uma e breve explicação.*

**Factory Method**

O Factory Method é um padrão criacional, cria objetos em uma superclasse

**Abstract Factory**

O Abstract Factory é um padrão de projeto criacional não precisa especificar classe quando você faz família em objetos relacionados

**Adapter**

O Adapter é um padrão de projeto estrutural que permite objetos com interfaces incompatíveis colaborarem entre si.

**Bridge**

O Bridge é um padrão de projeto estrutural permite hierarquia em classes separadas

**Chain of Responsibility**

O Chain of Responsibility é um padrão de projeto comportamental que vai passando pedido por handler e ele processa e passa pra outro.

**Command**

O Command é um padrão de projeto comportamental que transforma um pedido em um objeto independente que contém toda a informação sobre o pedido.

**Iterator**

Iterator é um padrão de projeto comportamental permite que você veja elementos sem expor ele

**Mediator**

O Mediator é um padrão de projeto comportamental reduz dependências caóticas entre objetos

**Builder**

O Builder é um padrão de projeto criacional que permite a você construir objetos complexos passo a passo

**Prototype**

O Prototype é um padrão de projeto criacional ele por meio de uma protótipo faz um clone igual o prototipo.

**Composite**

O Composite é um padrão de projeto estrutural objetos separados trabalhem com ele isolado pense numa arvore.

**Decorator**

O Decorator é um padrão de projeto estrutural colocar comportamentos dentro de comportamentos

**Memento**

O Memento é um padrão de projeto comportamental restaurar um objeto sem expor.

**Observer**

O Observer é um padrão de projeto comportamental ele notifica objetos do que está acontecendo.

**State**

O State é um padrão de projeto comportamental altera comportamento quando seu estado interno muda.

**Strategy**

Strategy é um padrão de projeto comportamental defini familia de algoritmos.

**Singleton**

O Singleton é um padrão de projeto criacional que permite a você garantir que uma classe tenha apenas uma instância,

**Facade**

Facade é um padrão de projeto estrutural um simplifica framework.

**Flyweight**

O Flyweight é um padrão de projeto estrutural aumenta a quantidade de objetos para colocar na RAM.

**Template Method**

O Template Method é um padrão de projeto comportamental que define o esqueleto de um algoritmo na superclasse

**Visitor**

O Visitor é um padrão de projeto comportamental que permite que você separe algoritmos dos objetos nos quais eles operam.

**Proxy**

O Proxy é um padrão de projeto estrutural pode fornecer um substituto para outro objeto.

*Um texto contendo quais padrões poderiam ajudar nesse projeto.*

-Adapter seria de grande ajuda para todos os objetos poderem ter uma colaboração entre si.

-Factory Method seria um pouco bom também para só 1 classe ter todos os métodos, talvez possa otimizar mais o código diminuindo a linhas.

-Iterator seria legal ter uma diminuição de elementos que podem aparecer na sua tela.

-Proxy seria bom quando acontecesse alguma coisa com o livro e tivesse que substituí lo.