**Construindo uma API REST com Java e Spring**

1. Como base para esse projeto vou utilizar as tecnologias:

* **Spring Boot v2.4.2:**

Para facilitar na criação e configuração inicial do projeto;

* **Spring Web:**

Com esse módulo tenho acesso a um servidor de aplicação chamado Tomcat embutido no framework, então não precisamos configurar um servidor de aplicação para executar o projeto;

* **Spring Validation:**

Disponibiliza inúmeras anotações para validação de dados de entrada da API e a criação de anotações personalizadas;

* **Spring Data JPA:**

O Spring Data usa Hibernate como provedor de implementação JPA padrão, oferece um grande suporte na implementação de camadas que acessam o banco de dados ex. (DAO), recursos como implementação de um poderoso repositório com métodos CRUD, paginação, ordenação, suporte a consultas mais complexas com JPQL e muito mais;

* **Spring test:**

fornece todo o ecossistema Spring facilitando na criação de testes de integração, mock objects,

* **Modelmapper:**

Uma biblioteca muito útil para a conversão de classes POJO ou DTO para classes que representam entidades do banco de dados;

* **Springdoc - openapi:**

Uma biblioteca que automatiza a criação da documentação de toda a API;

* **PostgreSql:**

Um banco de dados relacional;

* **Docker:**

Será usado Docker e Docker-compose para encapsular tanto a aplicação como o banco de dados;

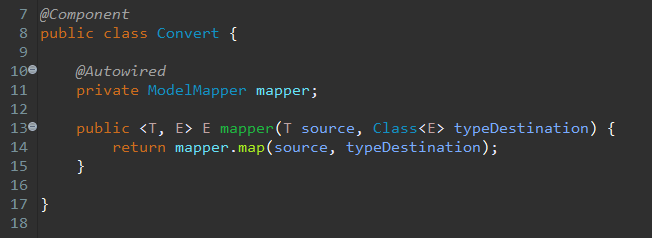
1. O projeto:

Como de costume começo pela borda mais externa da aplicação isso ajuda a ter um resultado inicial mais rapidamente, simplesmente utilizando o Postman para enviar requisições e ver se já estão chegando ao controller, então assim que crio o projeto, a primeira classe a ser criada é a UserController segue abaixo a implementação:



Este é o único controller da aplicação, possui 4 métodos e um construtor, no construtor recebe dois atributos por injeção de dependência, a classe da camada de serviço e uma classe com um método genérico de conversão, então podemos converter tanto classes de Entidade para DTOs como também DTOs para entidade, merge(User user) é o único método privado dessa classe, sua função é converter a Entidade User para um DTO UserRs e todos os Tickets para TicketRs, que são objetos de resposta da aplicação;

Segue abaixo as classes recebidas pelo construtor:



Classe Convert anotada com @Component para ser gerenciada pelo Spring, possui um único método que recebe uma entidade T e um E que é o tipo de destino e retorna uma instancia da entidade do tipo de destino já com tudo mapeado;



@Transactional????