Para criação do projeto, na aba: Criar novo projeto:

Spring Starter Project

Spring version (2.3.3 mesmo)

Procurar por dependência: Spring Web (pois não tem mais o Spring Web Starter)

No teste, iniciamos o projeto, já tem o Tomcat embutido no projeto automaticamente.

1 commit:

Entidade users e criado nosso resource, onde teremos no controle REST

2 commit:

Instalação do BD H2, ele é instalado em memória. Criando um perfil de teste.

Fazendo a instalação do H2 pelo pom.xml já temos junto o JPA automaticamente.

Entendendo as configurações do application properties:

spring.datasource.url=jdbc:h2:mem:testdb String de conexão com o banco, sendo em memória (mem)

spring.datasource.username=sa

spring.datasource.password=

spring.h2.console.enabled=true console está habilitado

spring.h2.console.path=/h2-console acesso ao adm do H2 é este

spring.jpa.show-sql=true mostrar no log da aplicação para mostrar o comando

spring.jpa.properties.hibernate.format\_sql=true formatar o comando SQL

3 commit:

Criando JPA repository, um “sub” framework do Spring, inserção

JPA Repository é uma interface.

TestConfig é uma Classe auxiliar – não e Controlador, Service ou Respository.

Database seeding acontece no TestConfig, quando instanciamos no método run.

4 commit:

Neste momento estamos começando a mexer na nossa camada de serviço, retirando o mockup que colocamos no UserResource da Maria, e chamando o findAll com uma Lista. Para isso, fazemos uma autoinstanciação do UserService para podemos aproveitar a camada de serviço.

A partir daí retornaremos uma Lista de Usuários, portanto ResponseEntity<List<User>>

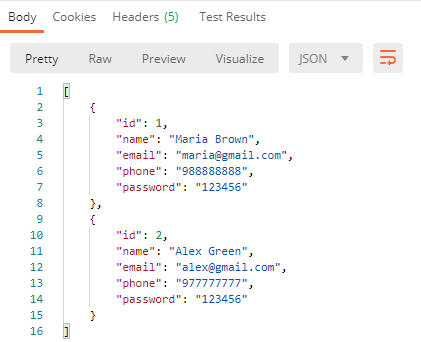
Aqui também registramos pelo framework que o UserService é um @Service, mas no UserRepository não fizemos, pois se ver no código verá que ele herda do próprio JPARepository

public interface UserRepository extends JpaRepository<User, Long>

Portanto é opcional o uso do @.

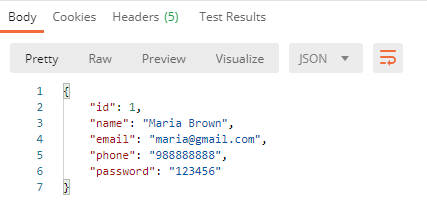
A pilha de camadas funciona aqui: resource chama o service que chama o repositor





Depois disso criamos nosso endpont para recebermos o id no get





Feito na camada de Resource, pois acessa o BD.

Entidade pedido: sem o orderStatus e +total() por enquanto

Para se criar essa entidade, temos no documento o checklist.

Instanciamos a coleção ArrayList de orders....

OBS: set list não tem porque em nenhum momento iremos trocar a lista utilizando o Set, apenas acrescentar e remover elementos.

Em Order, tivemos que fazer esta anotação (aproveitamos e fizemos pro User também):

*@Table*(name = "tb\_order") //tivemos que criar desta forma pois deu erro no build, deu conflito com o Banco de Dados, é uma palavra reservada do SQL

*@Table*(name = "tb\_user")

Nesta segunda parte do commit acima: Order, Instant, ISSO 8601

Estamos criando as classes de Java para Order e colocando o padrão 8601 (parte de Timezone conforme UTC – Greenwich)

Uma observação:

Order o1 = new Order(null, Instant.*parse*("2019-06-20T19:53:07Z"), u1); //famosa associação de objetos, passando o User (u1)

Order o2 = new Order(null, Instant.*parse*("2019-07-21T03:42:10Z"), u2);

Order o3 = new Order(null, Instant.*parse*("2019-07-22T15:21:22Z"), u1);

Estamos com o PC com o horário do Brasil, portanto aparece no h2 horário local do PC, então dá o atraso de 3 horas.

A biblioteca que faz a serialização de JSON é a Jackson, enquanto temos associação entre User e Order, temos uma associação de mão dupla...devemos colocar o ***JsonIgnore***. Em um dos lados pq senão vira um looping infinito -> user – order – user – order.... Portanto, onde colocamos o JsonIgnore ele não aparecerá

**O que faz o Jackson? Serializa o JSON, solicitando os itens que estão no banco de dados, e assim o JPA os traz.**

No exemplo abaixo o JsonIgnore está dentro da Entidade User, em cima da Order

Neste momento do desenvolvimento temos uma relação de MUITOS Orders para 1 User, o JPA faz a relação que quando queremos ver as Orders já puxa o User. Conforme o GET ***orders/1***:

{

    "id": 1,

    "moment": "2019-06-20T19:53:07Z",

    "client": {

        "id": 1,

        "name": "Maria Brown",

        "email": "maria@gmail.com",

        "phone": "988888888",

        "password": "123456"

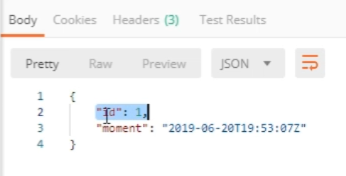
    }

}

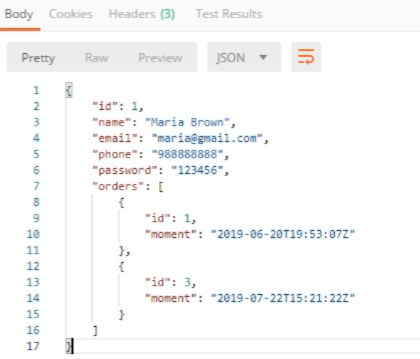
Do lado 1 para Muitos o JPA não carrega o Muitos (Lazy loding), ToMany anotação.

Se invertermos, colocando o JsonIgnore na Entidade Order, em cima da instanciação de User, teremos estes dois GET:

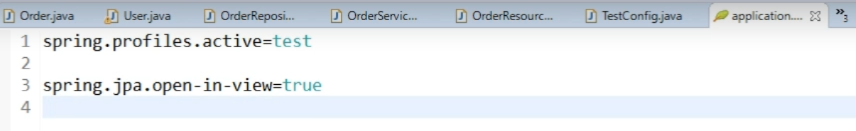
GET orders/1:



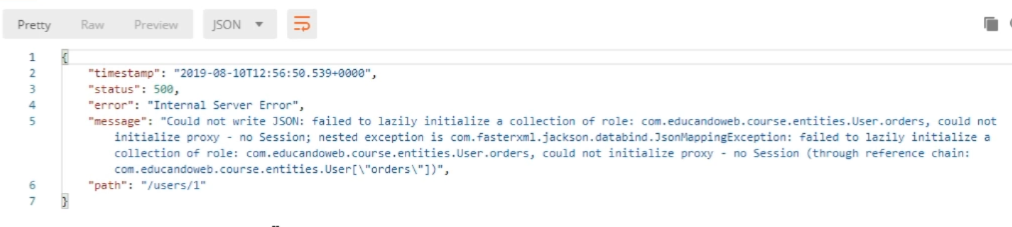
Users/1:



Como funciona esta configuração abaixo?



Está dizendo que no fim do ciclo, ele chama o Banco de Dados de novo para pegar o restante dos itens, deixando o Jackson pedir itens. Desta forma ele está chamando o JPA para mais itens, e se deixarmos false, ex: Erro 500 abaixo



Se deixamos false no fim do ciclo não é chamado o restante de itens. Deve-se atentar para quando é bom deixar false ou true, true deixa mais lento o ciclo.