**HTTP**

**Hyper Text Transfer Protocol**

Protocolo de comunicação

Browser – Implementar o cliente HTTP

Servidor – Host Objetos We**b**

**Arquitetura Cliente-Servidor**

**Cliente** – Envia as mensagens de **request** – Objetos WEB

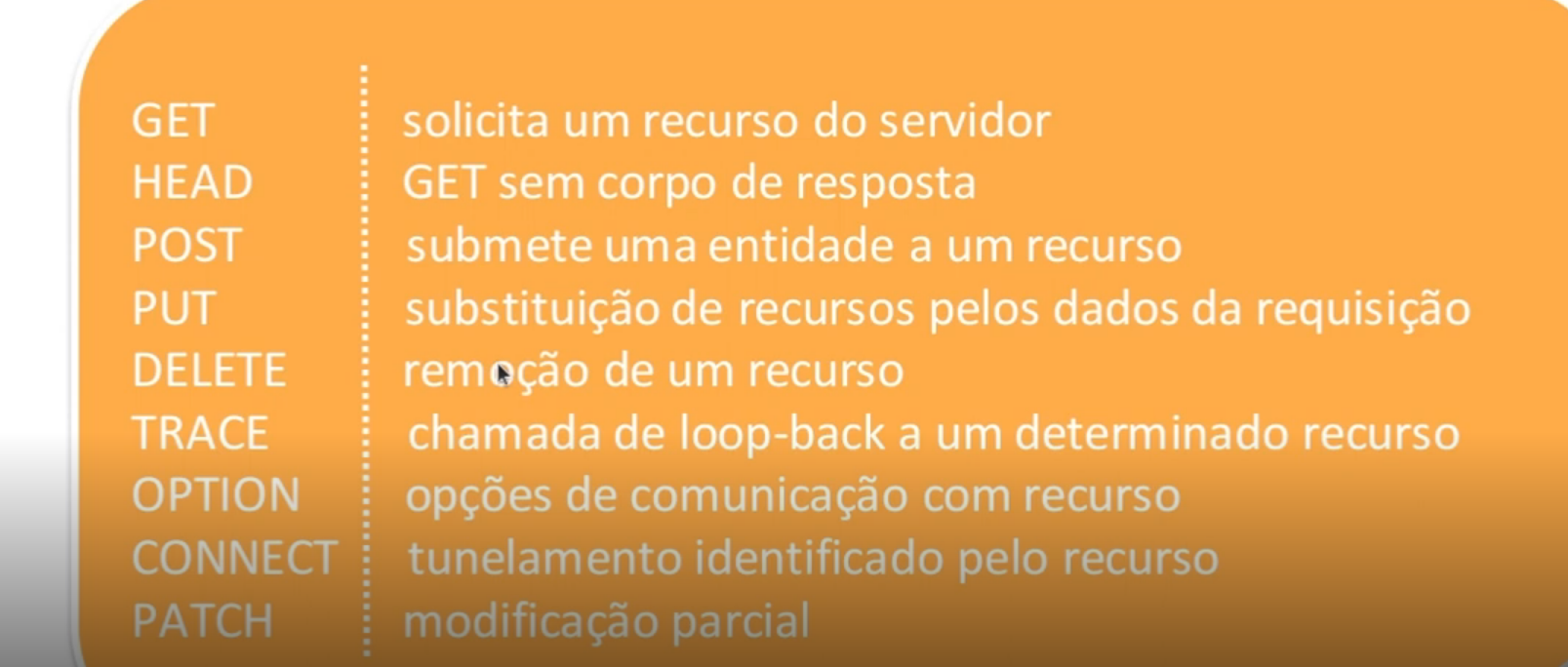
**Servidor** – Responde as mensagens de **response** – Protocolo TCP - **Stateless** – Não guarda o dado do cliente

1997 – Chegou o CORS e o Keep-Alive

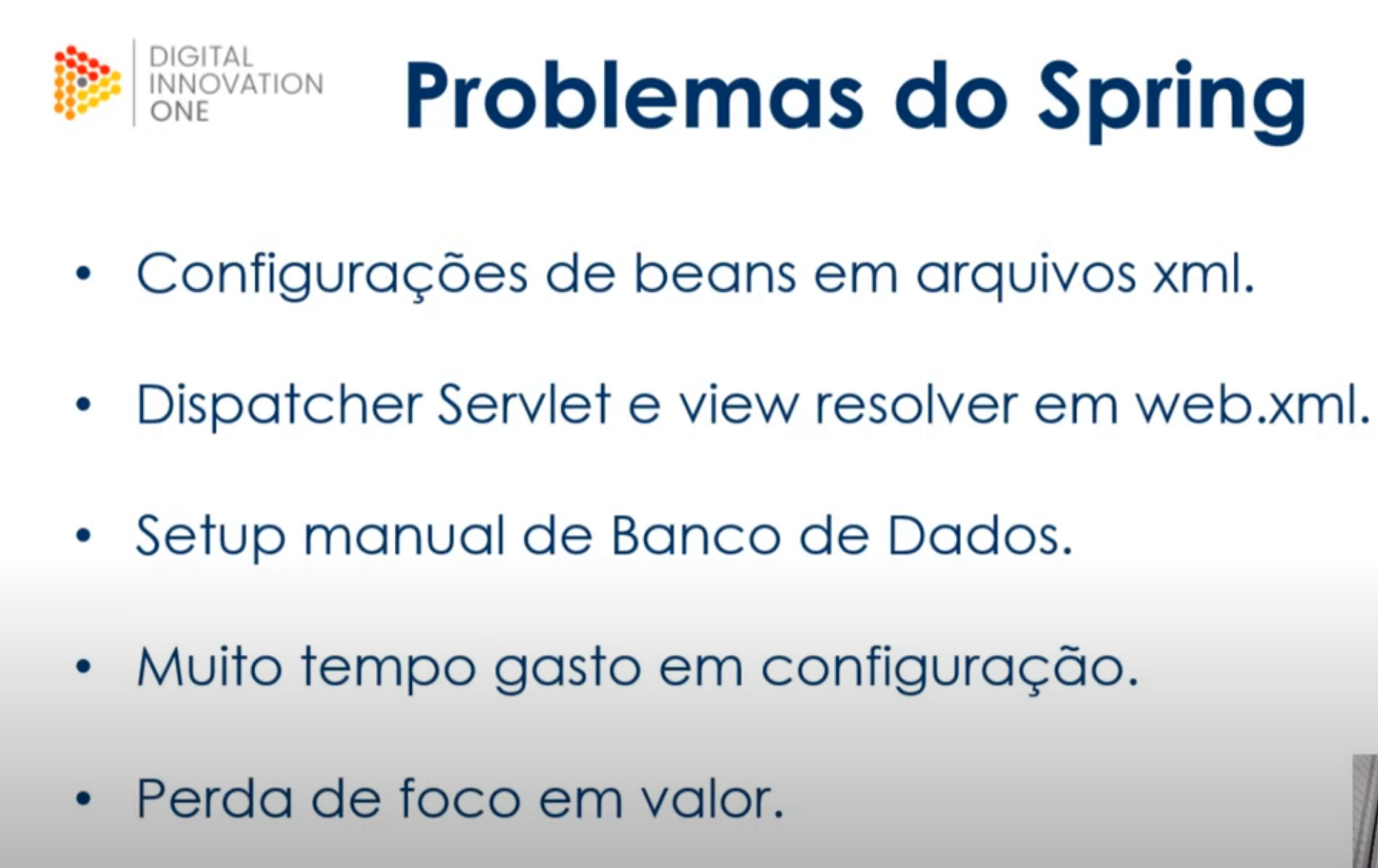
**Entity Body – XML ou JSON**

**Processo –** Armazenar, transmitir e reconstruir

**Objetivo** – Serialização



**SPRING BOOT**



Ele foi criado em 2012, seu objetivo é facilitar setup de projetos.... Sem necessidade de criar arquivos de configuração, sendo assim o foco em produtividade é maior, pois temos maior tempo de desenvolvimento em valor.

**Produtividade**: Setup simplificado de projeto.

**Starters**: Dependências autoconfiguráveis pelo Spring Boot.

**Execução** Simplificada: Sem deploy em servidor externo.

**Configuração**: Arquivo externo para configuração.

**Valor**: Maior tempo em desenvolvimento.

**Antes**

**Multiplos arquivos XML**

**Configuração manual do MVC: DIspacher Servlet, Web.xml e spring-mvc.xml**

**Configuração manual dos beans**

**Hoje – Auto Configuration**

**Starters:** Dependências simplificados e auto importadas

**Identificação e configuração** automática da dependência.

**Spring Tradicional:** War precisa de servidor de aplicação

**Dependência** de um container WEB ou servidor de aplicação

**Complexidade** para configurações

**Atualizações frequentes**, junto com versão do projeto

**Gerenciamento manual** de configurações

**FatJar/UberJar**

**Artefato do projeto pronto para execução**

**Container web embutido na geração e execução (Tomcat)**

**Deploy embarcado com outros containers são opcionais**

**Dependências principais do projeto embarcado**

****

**Profiles e Configurações**

Ambientes para desenvolvimento, teste e produção – **Múltiplos Ambientes.**

Bancos de dados para cada ambiente

Execução de testes unitários em ambiente local

Suite de testes completas em ambiente de teste

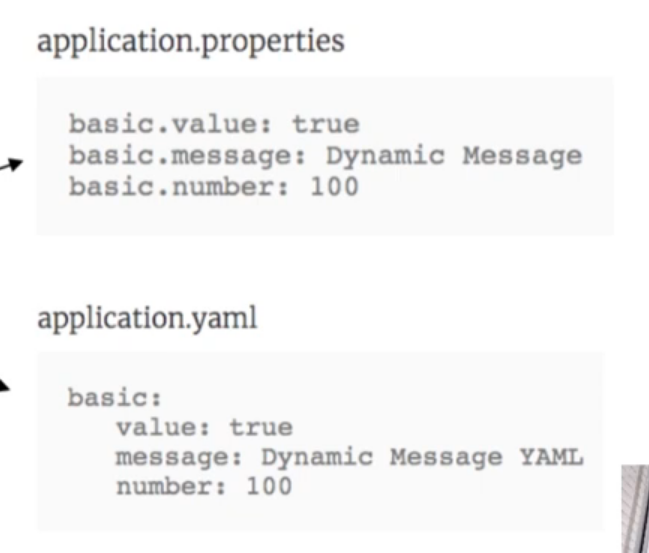
Simulação do ambiente real em staging

Deploy simplificado em produção

O **Profile** tem configurações próprias para cada ambiente.

**YAML**

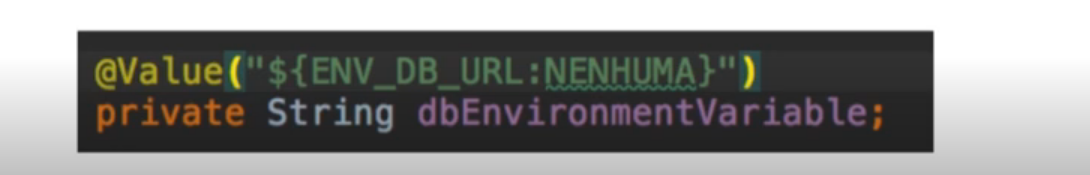
**Troca .properties para .yaml**



**Variáveis De Ambiente**

Injeção com anotação @Value({{NOME\_VARIAVEL})

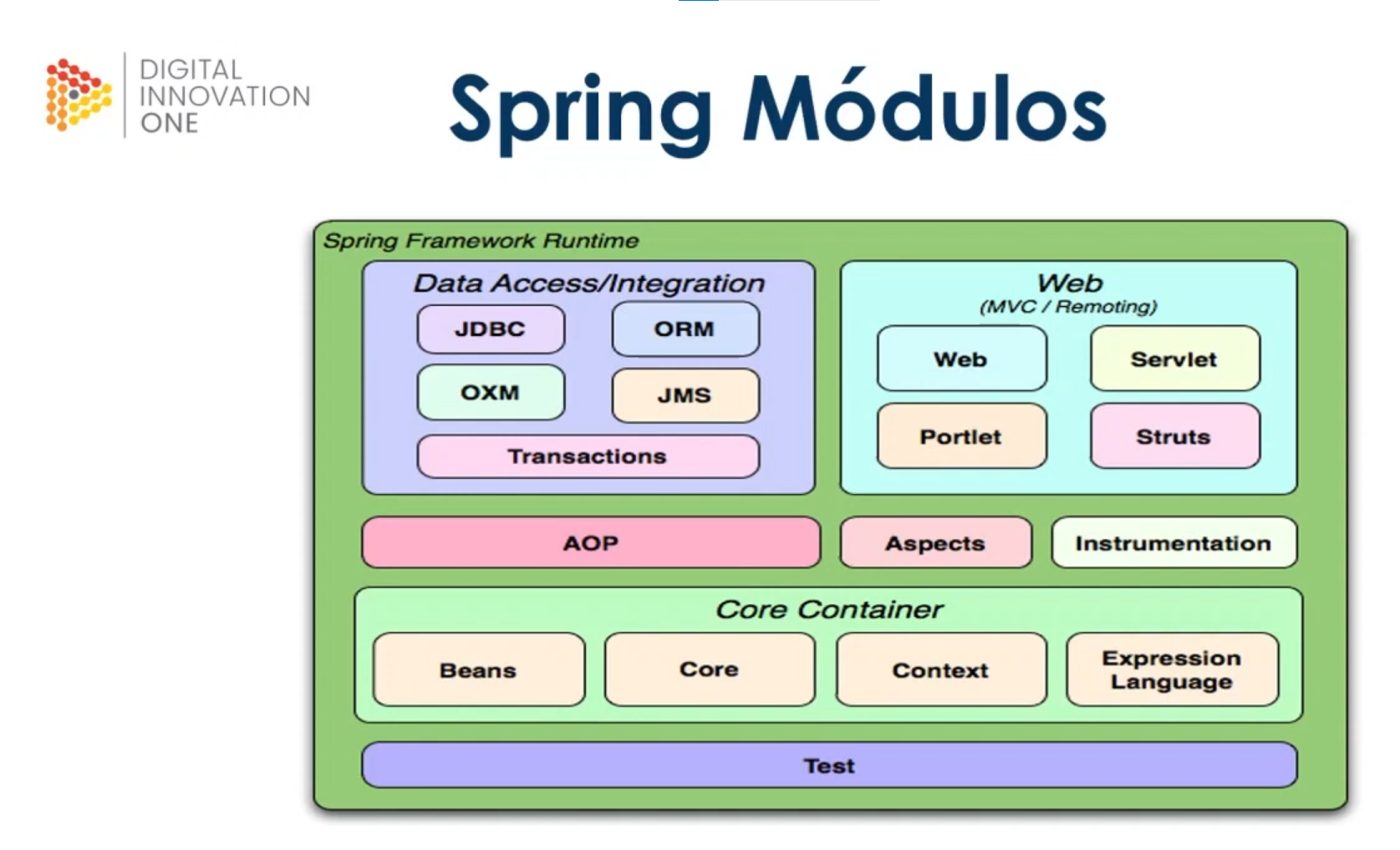
Definição de valor default quando não há variável.



**Spring Framework**

Framework baseado os padrões de projetos inversão de controle e injeção de dependências, baseado em módulos.

Core Container.



**Inversão de controle**

Redirecionamento do fluxo de execução de um código retirando parcialmente o controle sobre ele e delegando-o para um container. O Principal propósito é minimizar o acoplamento do código.

**Injeção de dependência**

Padrão de desenvolvimento com a finalidade de manter baixo o nível de acoplamento entre módulo de um sistema.

**Beans**

Objeto que é instanciado (criado), montado e gerenciado por um container através do principio da inversão de controle.

**Escopos**

**Sigleton**

Único objeto sendo compartilhado por toda minha aplicação.

**Protoype**

Criado uma nova instancia a cada requisição ao container**.**

**Request**

Um bean será criado para cada requisição HTTP (Objetos existirão enquanto a requisição estiver em execução)

**Session**

Um bean será criado para cada sessão de usuário.

**Global**

Cria um bean para o ciclo de vida do contexto da aplicação.

**Autowired**

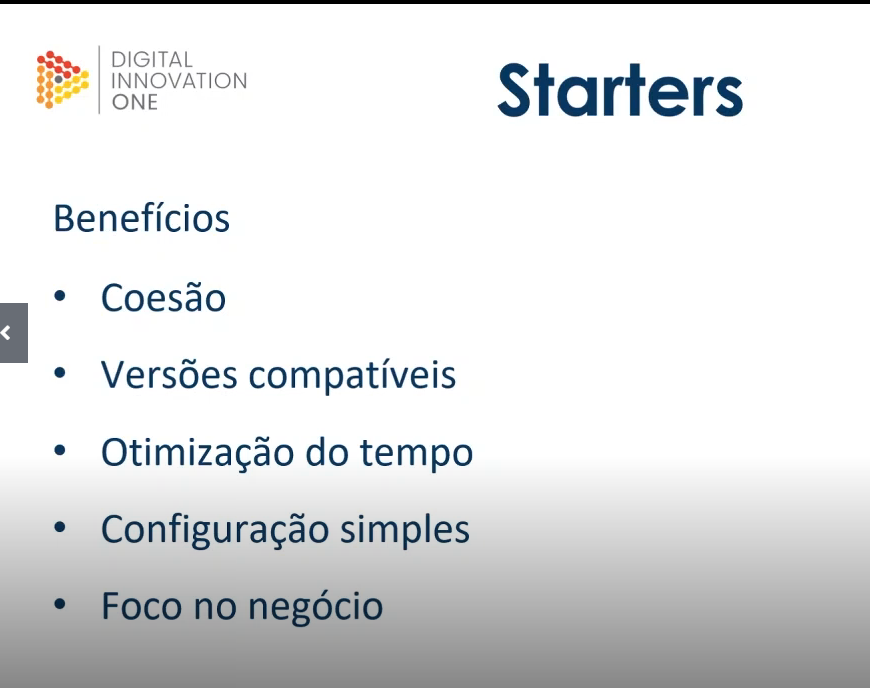
Anotação onde deverá ocorrer uma injeção automática de dependência.

**byName** – É buscado um método set que corresponde ao nome do Bean.

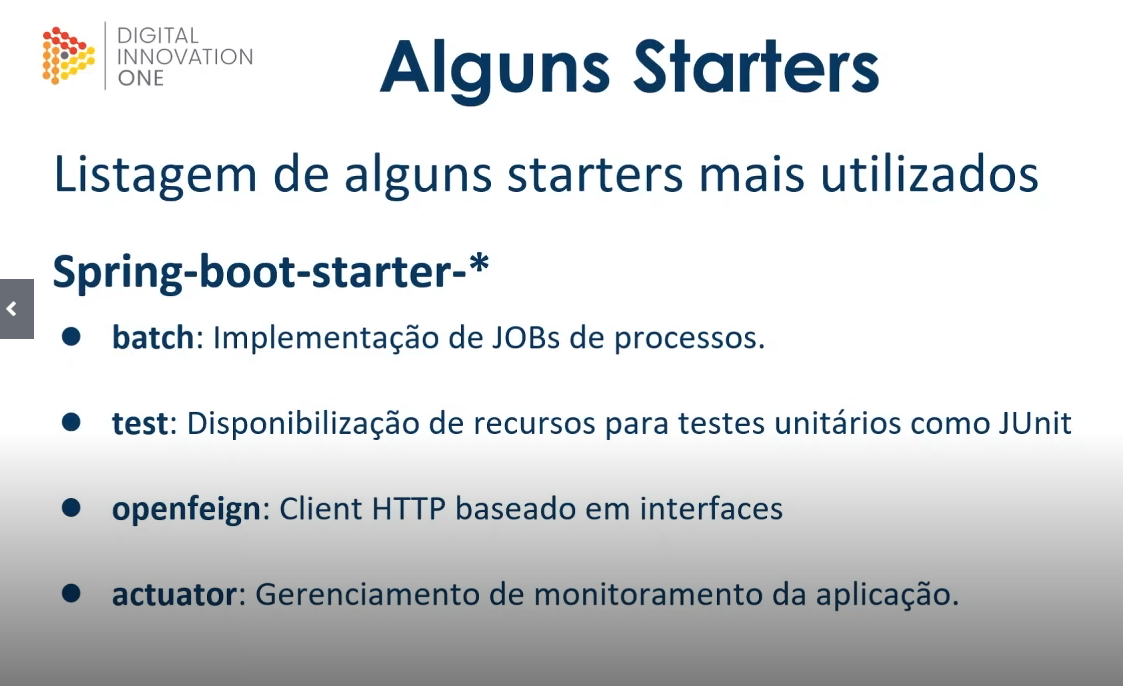
**byType** – É considerado o tipo da classe para inclusão do Bean.

**byConstrutor** – Usamos o construtor para incluir a dependência.









**Beans x Components**

**Component** quando tenho acesso ao código fonte, não vou dar new... E posso injetá-lo em qualquer ecossistema do meu spring boot.

**Bean** quando já é um arquivo de uma biblioteca externa

**Nem tudo é = --- Properties Value**

No properties = nome=Mateus

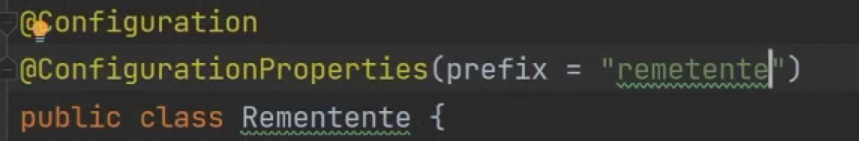
Na classe: @Value(“${nome:NoReply}”)

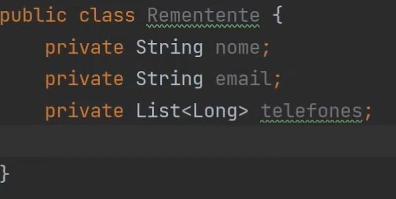
(No Reply seria o valor default caso não encontrasse uma no properties.)

**@ConfigurationProperties**

No properties = remetente.nome=Mateus

Cria uma classe rementente e nela você coloca



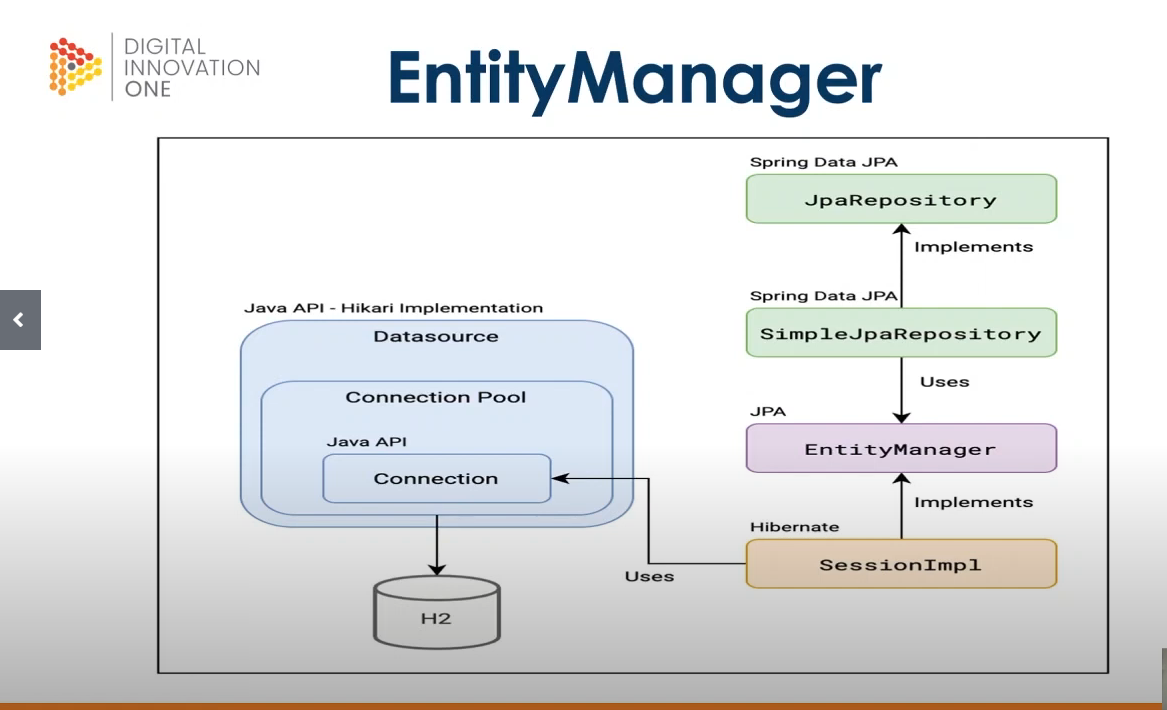


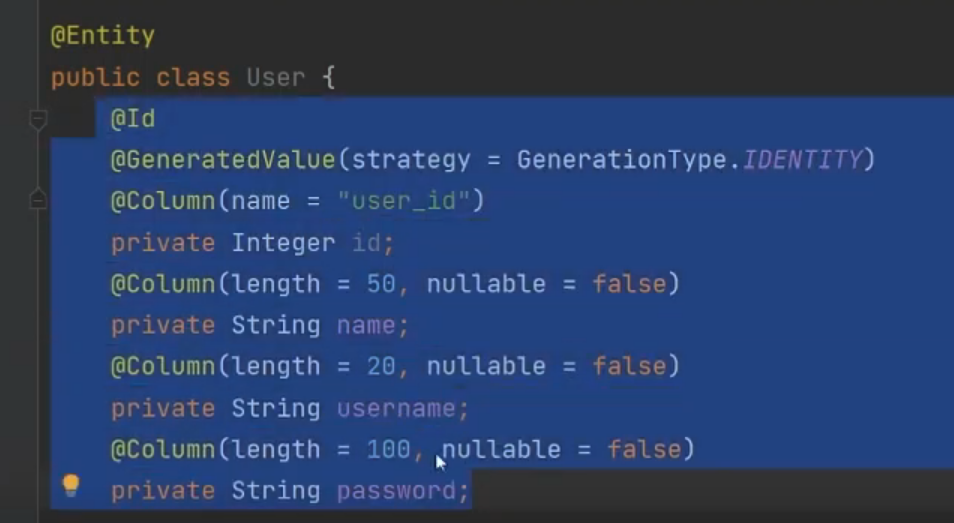
**ORM – Object-Relational Mapping.** Recurso para aproximar o paradigma da POO ao contexto de banco de dados relacional.

**JPA** é uma especificação baseada em interfaces, que através de um framework realiza operações de persistência de objetos em Java.

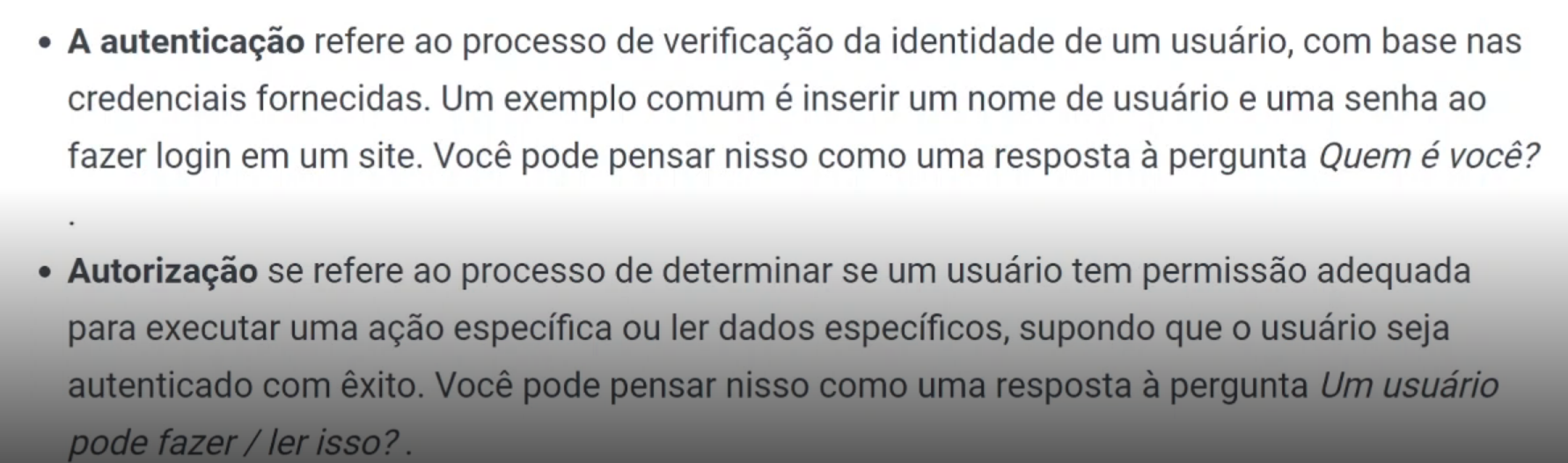
**Mapeamento**

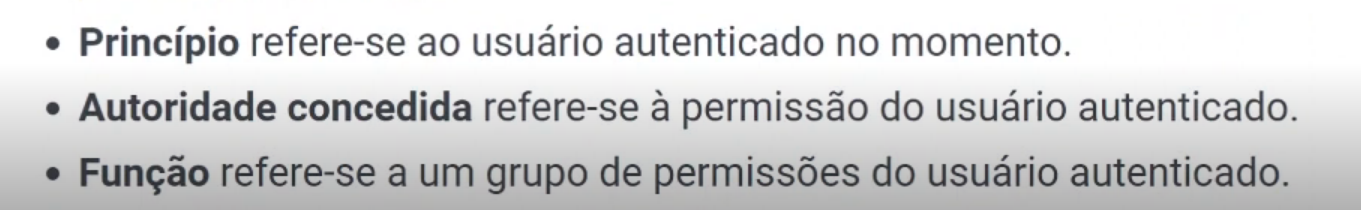
* Identificação
* Definição (Maximo de caracteres, é chave? Etc)
* Relacionamento
* Herança
* Persistência



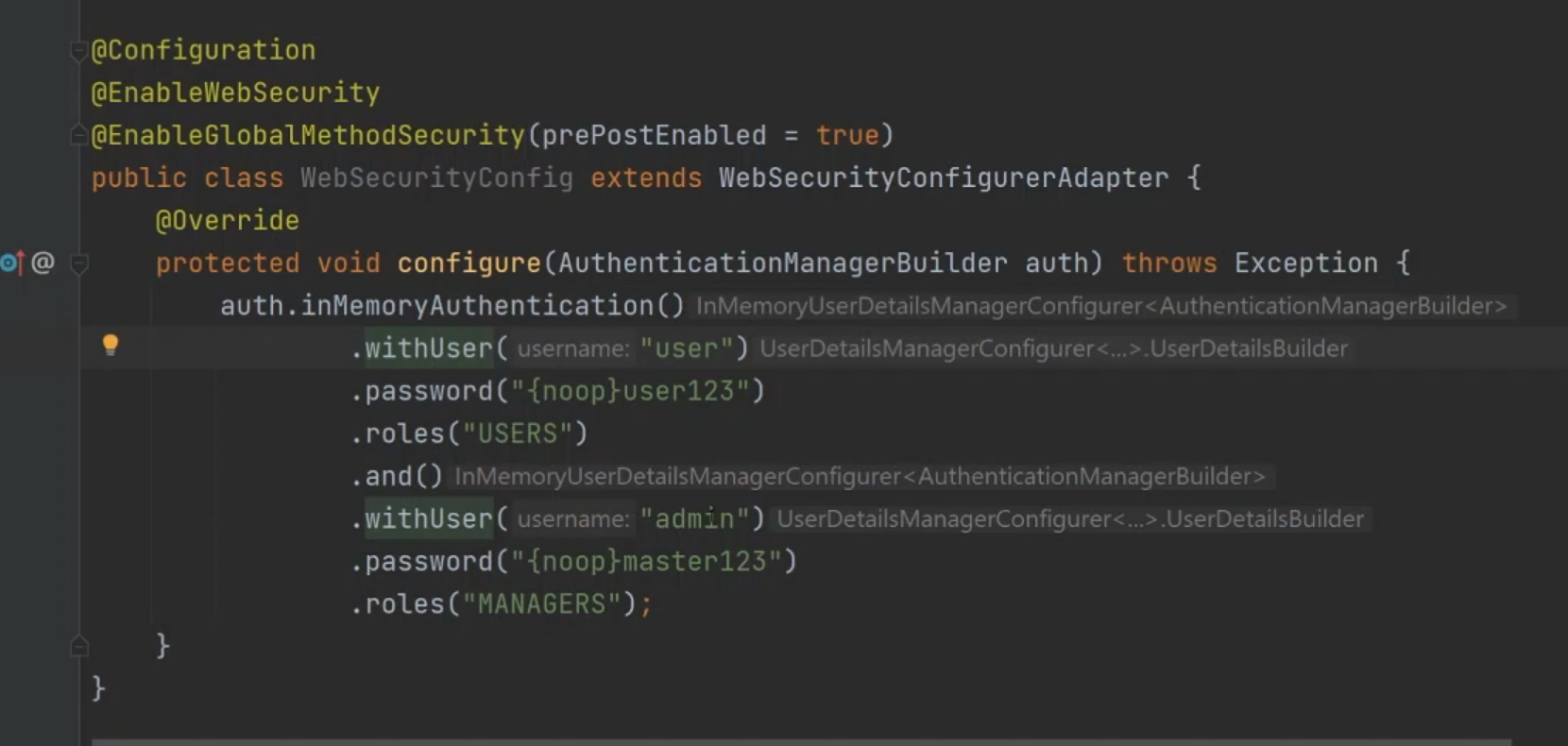


**Spring Security**





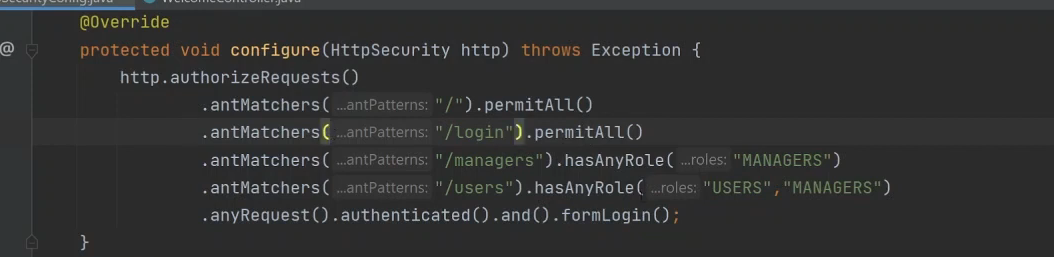
Configuração da classe de segurança



@PreAutohrize(“hasRole(‘MANAGERS)”) – configura a roles que pode acessar.

@PreAutohrize(“hasAnyRole(‘MANAGERS’,’USERS’)”) – configura a roles que pode acessar.

Recomendado criar uma classe que faça isso, uma configure Adapter:



JWT – JSON Web Token

Ele tem um Header, Payload e uma Signature (Camada de assinatura)

**Header**: Tipo de token e o algoritmo de assinatura

**Payload**: Estrutura do corpo contendo as informações de autenticação e atualização do usuário

**Signature**: Verifica a autenticidade do Token