

## Sumário

O que é uma API .....	2
REST – documentação do que precisa fazer (boas praticas) .....	2
REST FULL – capacidade de aplicar todas as funções atribuídas em REST .....	2
o que necessario para ser uma API REST .....	3
Nivel 0 Ausencia de Regras .....	3
Nivel 1 Aplicação de Recursos GET POST PUT DELETE .....	3
Nivel 2 Implementação de verbos HTTP .....	3
Nivel 3 HATEOAS → Hippermedia as the Engine of Application State .....	3
E REST FULL.....	4
Iniciando Projeto .....	4
Rest Controller .....	4
Requisições .....	5
RequestBody – converte um JSON para um objeto para poder manipulalo .....	5
PathVariable – passado uma informação na URI, e quero saber o que foi passado .....	5
Rest Model.....	5
@PathVariable → Organizar a Busca.....	5
@RequestMapping → Para evitar LOCALHOST:8080/USERS.....	6
SWAGGER .....	6
@Confiv .....	7
@Document.....	7
Handlers .....	8
Spring Security .....	9
Autorização.....	9
Principio.....	9
Autoridade permitida .....	9
Função .....	9
Primeiros passos com Spring Security.....	9
Por Default.....	9
Chaves de Acesso JWT .....	10
Em Memoria .....	10
Tipos de Criptografia.....	12
Autenticação com Banco de Dados – Auth Database .....	12
Dependencia do Potgress/H2/MySql.....	12
Dependencia do Data JPA .....	12
Agora é necessario ter uma classe de usuário com tipo de acesso em Role .....	13
Seguindo a documentação, criando mais uma classe de configuração, fica assim .....	13

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io

## Desenvolver coma API REST SPRING e Documentar em SWAGGER

### Desenvolver a Api

Manipular informações com Controller

Documentar recursos com SWAGGER

Tratar exceções com Handlers, quando erros não esperados na aplicação

## Contexto Web

Em Web há 2 formas, a MVC e a API

Dentro de API, aprofundaremos em API REST com SPRING BOOT, já explicado em aulas passadas a intro

## O que é uma API

Codigo programável

A ponte de um código a um conjunto de objetos

REST – documentação do que precisa fazer (boas praticas)

REST FULL – capacidade de aplicar todas as funções atribuídas em REST

Quais seus princípios

- cliente-servidor – significa aprimorar a portabilidade entre várias plataformas de interface do usuário e do servidor, permitindo uma evolução independente do sistema;
- interface uniforme – representa uma interação uniforme entre cliente e servidor. Para isso, é preciso ter uma interface que identifique e represente recursos, mensagens autodescritivas, bem como hypermedia (HATEOAS);
- stateless – indica que cada interação via API tem acesso a dados completos e compreensíveis;
- cache – necessário para reduzir o tempo médio de resposta, melhorar a eficiência, desempenho e escalabilidade da comunicação;
- camadas – permite que a arquitetura seja menos complexa e altamente flexível.

Interação entre minha aplicação e servidor

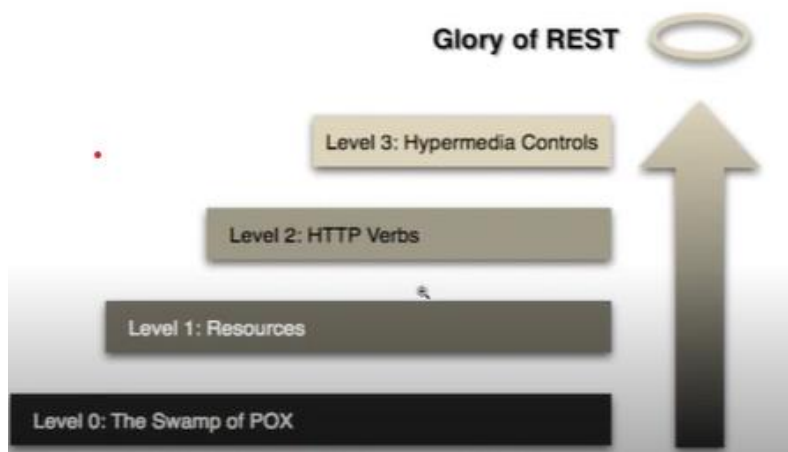
Interface que tenha sempre o mesmo tipo de estrutura para resposta para o cliente, independente de que interface ele utilize

Stateless cada uma das requisições, dao uma resposta de acordo com seu contexto

Cache tem capacidade de recursos para reduzir o tempo de resposta

Camadas ...!

o que necessário para ser uma API REST



Dessa forma o

*Nível 0 Ausencia de Regras*

Implementa os recursos

Verbo HTTP	URI	Operação
POST	/getUsuario	Pesquisar Usuario
POST	/salvarUsuario	Salvar
POST	/alterarUsuario	Alterar
POST	/excluirUsuario	Deletar

Tem os métodos mas eles não estão sendo aplicados

*Nível 1 Aplicação de Recursos GET POST PUT DELETE*

Como já estudado, há os tipos primordiais de requisições (crud)

E com elas as chamadas por URI, através do POSTMAPPING (“...”) determinado no Back

Verbo HTTP	URI	Operação
GET	/usuarios/1	Pesquisar Usuario
POST	/usuarios	Salvar
PUT	/usuarios/1	Alterar
DELETE	/usuarios/1	Deletar

Nível 2 Implementação de verbos HTTP

Encarregado da validação de se os métodos estão operando corretamente

Observa as Respostas HTTP, se estão operando como de acordo

Nível 3 HATEOAS → Hippermedia as the Engine of Application State

Disponibiliza uma estrutura dentro das respostas

Implementa e fornece aos clientes, os links para mostrar como fazer a navegação de seus recursos

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io

```

{
  "id": 1,
  "nome": "John",
  "sobrenome": "Doe",
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8080/clientes/1"
    },
    {
      "rel": "alterar",
      "href": "http://localhost:8080/clientes/1"
    },
    {
      "rel": "excluir",
      "href": "http://localhost:8080/clientes/1"
    }
  ]
}

```

## E REST FULL

Que comporta todos os níveis citados acima

Lembrando que quando criado uma aplicação Spring, o TOMCAT vem junto, pois é o compilador que torna a aplicação ser WEB

## Iniciando Projeto

Ir no site do Spring START.SPRING.IO, configurar os campos, dependência WEB somente

```

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

```

## Rest Controller

Todas as operações são feitas através de um controller

Todas as funções e logicas de negócios são feitas no controler e repassado para o protocolo HTTP

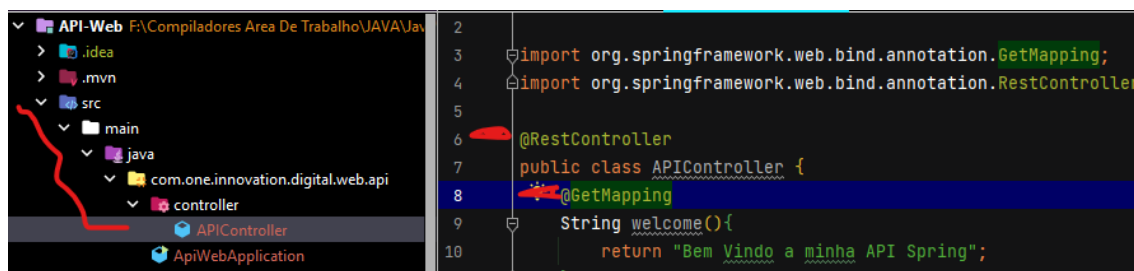
Em seguida o SERVICE vai fazer isso e repassar para o HTTP se no controler quiser deixar so como uma engrenagem

Quando iniciar a classe que queira ser de controle

Ter a anotação de Injeção de Controle @RestController

Sempre que declarado algum método, em cima do método o *Bean* que mostre que ele é uma requisição GET/POST/PUT/DELETE/...;

Dessa forma



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a package named `com.one.innovation.digital.web.api` with a sub-package `controller` containing `APIController` and `ApiWebApplication`. The code editor shows the following code:

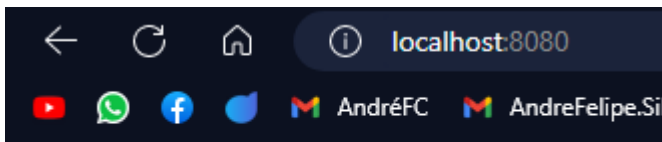
```

1
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 public class APIController {
8     @GetMapping
9     String welcome(){
10         return "Bem Vindo a minha API Spring";
11     }
12 }

```

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io

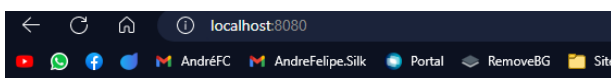


Bem Vindo a minha API Spring

Para definir que deve acessar esse método através de outro link, dessa forma

```
@RestController
public class APIController {
    @GetMapping("/welcome")
    String welcome(){
        return "Bem Vindo a minha API Spring";
    }
}
```

Desse modo, só aparecerá no link indicado, deixando a tela inicial sem nenhuma informação, dessa forma a seguir...

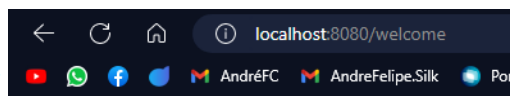


## Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Nov 15 02:24:33 BRT 2022

There was an unexpected error (type=Not Found, status=404).



Bem Vindo a minha API Spring

## Requisições

Como funciona essas requisições

- **@RestController**: Responsável por designar o bean de component que surporta requisições HTTP com base na arquitetura REST.
- **@RequestMapping("prefix")**: Determina qual a URI comum para todos os recursos disponibilizados pelo Controller.
- **@GetMapping**: Determina que o método aceitará requisições HTTP do tipo GET.
- **@PostMapping**: Determina que o método aceitará requisições HTTP do tipo POST.
- **@PutMapping**: Determina que o método aceitará requisições HTTP do tipo PUT.
- **@DeleteMapping**: Determina que o método aceitará requisições HTTP do tipo DELETE.
- **@RequestBody**: Converte um JSON para o tipo do objeto esperado como parâmetro no método.
- **@PathVariable**: Consegue determinar que parte da URI será composta por parâmetros recebidos nas requisições.

*RequestBody – converte um JSON para um objeto para poder manipulalo*

*PathVariable – passado uma informação na URI, e quero saber o que foi passado*

## Rest Model

Dentro dela que são atribuídas as Entidades, os objetos do projeto

Seja usuários, inovações, automóvel...;

**@PathVariable** → Organizar a Busca

Quando quero buscar um dado específico, além de buscar por /username

Devo utilizar do PatchVariable para definir o parâmetro

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io

```
@GetMapping("/users/{username}")
public Usuario getOne(@PathVariable("username") String username){
    return repository.findByUsername(username);
}
```

Dessa forma

[@RequestMapping](#) → Para evitar LOCALHOST:8080/USERS...

Quando utiliza do [@RequestMapping](#) ele organiza para todas as requisições dentro da Classe partirem da USUARIOS  
→ se assim declarado

Entao em vez de

```
@GetMapping("/usuarios") // GET ALL
public List<UsuarioModel> getUsers(){return userRepo.findAll();}
```

```
@GetMapping("/usuarios/{username}") // Busca por Nome
public UsuarioModel getOne(@PathVariable("username")String username){return
userRepo.findByUsername(username);}
```

```
@GetMapping("/usuarios/{id}") // Busca por Id
public UsuarioModel getId(@PathVariable("id") Integer id){return userRepo.findById(id);}
```

Fazer da seguinte forma

```
@RequestMapping("/usuarios")
```

Então ...

```
@GetMapping("/{username}") // Busca por Nome
public UsuarioModel getOne(@PathVariable("username")String username){return
userRepo.findByUsername(username);}
```

```
@GetMapping("/{id}") // Busca por Id
public UsuarioModel getId(@PathVariable("id") Integer id){return userRepo.findById(id);}
```

```
@DeleteMapping("/{id}")
public void deleteUser(@PathVariable("id") Integer id){userRepo.deleteById(id);}
```

## SWAGGER

Quando faço requisições, não se sabe ao certo que tipo de dados tenho que inserir de fato

Se tenho que colocar ID

Se faltou algum campo

Se o campo que coloquei esta correto

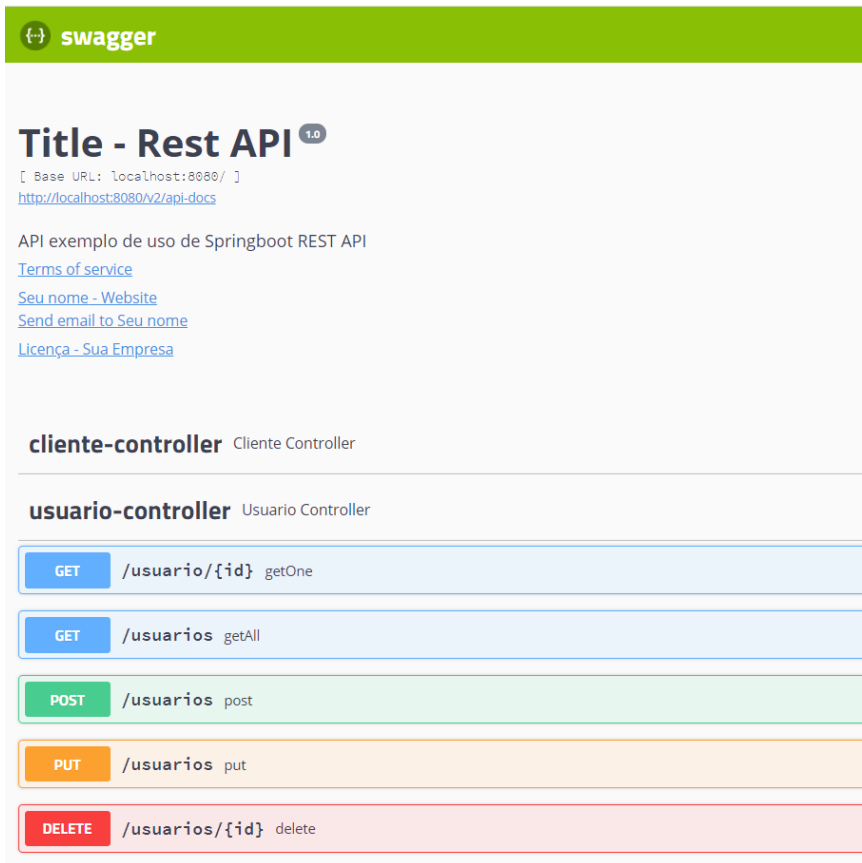
VENDO ISSO, FOI CRIADO UMA FERRAMENTA PARA AUXILIAR NA HORA DE UMA REQUISIÇÃO HTTP → SWAGGER

É a capacidade de documentar uma API

Linguagem de interface para API REST

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](#)

start.spring.io



Para isso, deve incluir a dependência do swagger no POM.XML

[Swagger - Spring Framework \(gitbook.io\)](https://gitbook.io/swagger-spring-framework)

```
<!-- SWAGGER DOCUMENTACAO -->
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.9.2</version>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.9.2</version>
</dependency>
```

Logo após, ler documentação para implementa-la da maneira correta

<https://glysns.gitbook.io/spring-framework/spring-web/swagger>

@Confiv

Como boas praticas, criar uma pasta Config para configurar as informações da Dependencia

@Document

Após feito todas as configurações, criar uma classe para Documentar e quando Inicializado o projeto, buscar antes de tudo, a documentação, implementada através do Bean @Document

Ter esse documento configurado para Consumir Json, auxilia na declaração na hora de validação de um programa

Atraves dessas linhas cadastradas

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io

O Swagger poderá em vez de scanear toda a aplicação

Ir a um local indicado diretamente, para fazer essas implementações

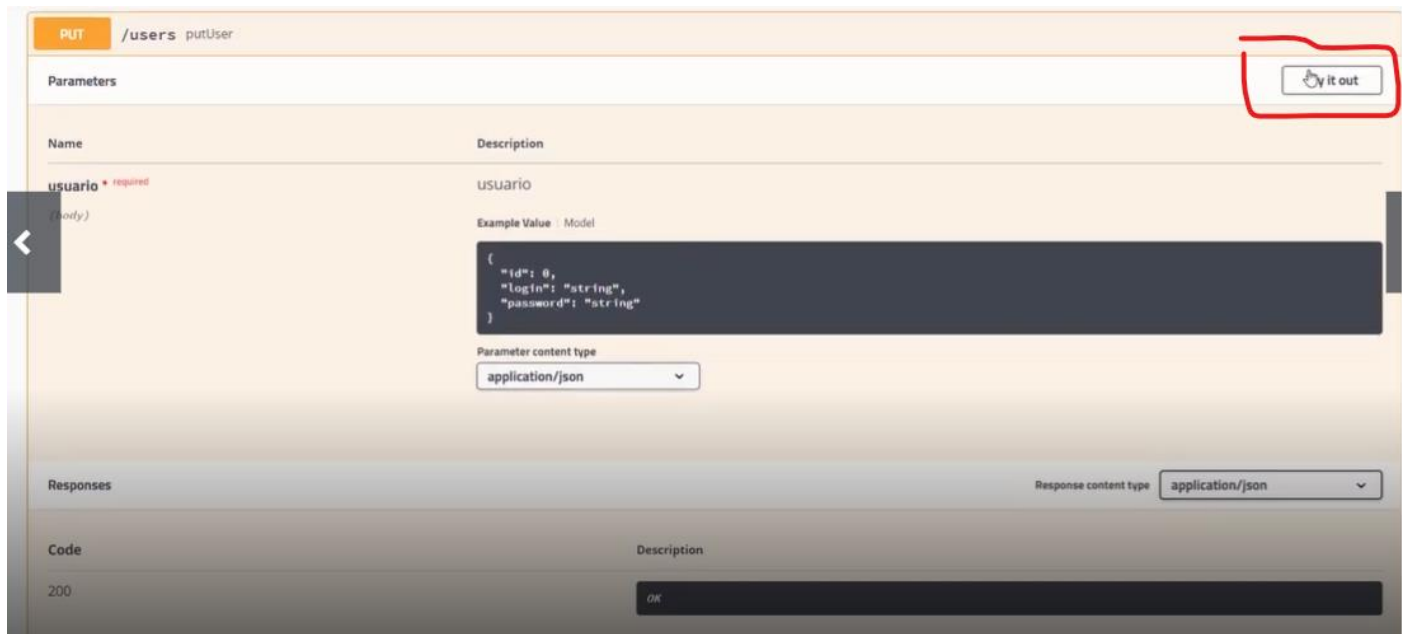
```
.apis(RequestHandlerSelectors.basePackage("pacote.comseus.controllers"))  
.paths(PathSelectors.any())  
.build()  
.apiInfo(this.informacoesApi().build())  
.consumes(new HashSet<String>(Arrays.asList("application/json")))  
.produces(new HashSet<String>(Arrays.asList("application/json"))).
```

Nessa parte `pacote.comseus.controllers`

Alterar para o pacote e classe do controller, desse modo

```
docket  
    .select() ApiSelectorBuilder  
    .apis(RequestHandlerSelectors.basePackage("com.one.innovation.digital.web.api.controller"))  
    .paths(PathSelectors.any())
```

Então acessando o Swagger pelo localhost, fica dessa forma



Aqui é um exemplo de PUT, mas da para fazer as demais requisições igualmente

## Handlers

Quando as requisições apresentam erro de Console

Aplicar tratativa de exceções

IOException

... Já falado sobre isso anteriormente

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io



# Spring Security

Grupo de filtros para auxiliar na implementação de segurança no projeto Web

## Autenticação

Verificação dos dados de um usuário, com base nas credenciais fornecidas → Usuario e Senha compatível com as credenciais solicitadas pelo sistema

## Autorização

Se o usuário tem permissão, e que grau de permissão para acesso ao sistema, impactando a níveis de acesso

## Princípio

Referencia se o usuário esta conectado no momento

## Autoridade permitida

Se o usuário tem permissão para acessar tal documento

## Função

Quantidades de permissões ao usuário autenticado

## Como habilitar segurança

Atraves da dependência Security

Ou se um projeto do zero, baixar com a dependência JPA, web e security

## Primeiros passos com Spring Security

Assim que iniciar o projeto feito com a dependência Security

É solicitado um usuário e senha, toda vez que entra na LOCALHOST:8080

## Por Default

Por default, essa senha gera a cada Run da aplicação e esta no Terminal

O usuário é por default user

## Autenticação Simples

De forma a definir um novo padrão

É possível definir um Usuario e Senha padrão, sem necessitar de BD, e burocracias JWT

Para isso, é preciso definir no APPLICATION PROPERTIES

```
Spring.security.user.name=andre
Spring.security.user.password=admin123
Spring.security.user.roles=USERS
```

Spring.security.user.name=user

Spring.security.user.password=user123

Spring.security.user.roles=USERS

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://start.spring.io)

start.spring.io

Isso definido um baixo nível de segurança, pois não há um nível de acesso declarando nesse modo

Todos acessam tudo e todos

## Chaves de Acesso

Tendo em vista o baixo nível de segurança

Foi pensado no JWT que lida com chaves de acesso para lidar com níveis de segurança, e distribuição de senhas para os demais usuários

Como no método acima so é uma boa pratica definir um usuário

Devera excluir as informações antes colocadas para aprendizagem

Em Memoria

E agora definir desse modo ...

Olhando na Documentação de Security: [Autenticação Simples - Spring Framework \(gitbook.io\)](https://gitbook.io/spring-security-5/pt-br/5.2.0/quickstart/quickstart-configuration)

```
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration // Classe de configuração
@EnableWebSecurity // Habilitando Segurança Web De Forma Manual
@EnableGlobalMethodSecurity(prePostEnabled = true) // Recursos globais de configuração
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception { /*Cria cadeia de usuarios
        auth.inMemoryAuthentication().inMemoryUserDetailsManagerConfigurer<AuthenticationManagerBuilder>
            .withUser( username: "user" ) UserDetailsManagerConfigurer<...>.UserDetailsBuilder
            .password("{noop}user123") /*NOOP nível de criptografia*/
            .roles("USERS")
            .and() InMemoryUserDetailsManagerConfigurer<AuthenticationManagerBuilder>
            .withUser( username: "andre" ) UserDetailsManagerConfigurer<...>.UserDetailsBuilder
            .password("{noop}admin123")
            .roles("MANAGERS");
    }
}
```

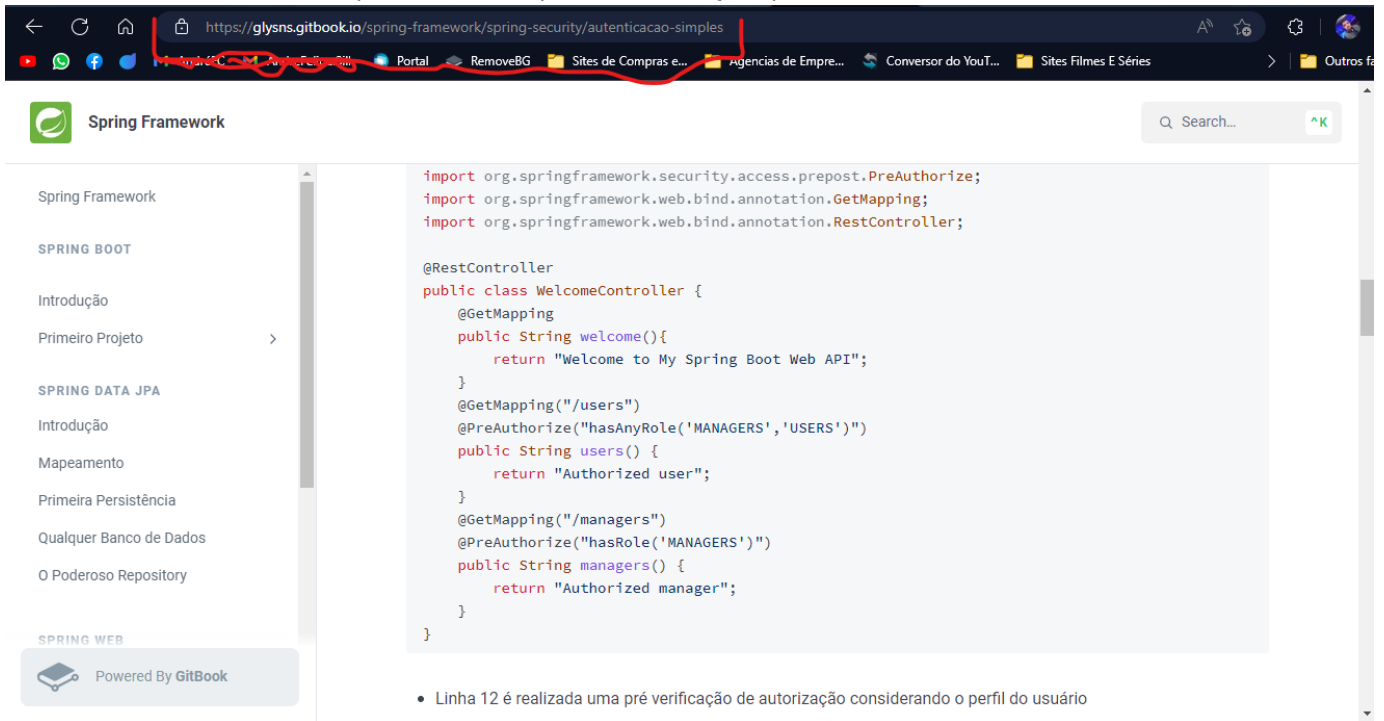
Com esses tipos de acessos, define Roles de Usuarios e Gerentes

No entanto ainda todos estão vendo tudo

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io

Dessa forma, para definir o tipo de visualização que cada um terá, será feito dessa forma:



**Lembrando!!! SEMPRE O WEB CONFIG DEVE ESTAR ALINHADO COM O WELCOME CONTROLLER, PARA NÃO HAVER ERRO**

**Dessa forma, se o Usuario tentar acessar o gerente, vai aparecer pagina não encontrada**

**Detalhar melhor isso em uma exceção**

Configure Adapter

Quando define os métodos acima, tende a assumir responsabilidade de configurar todos os usuários em Sistema

Em Configure Adapter, ele transfere os níveis de visibilidade de código e gerencia isso de forma automática

Atraves do Configure Security Adapter, conforme na documentação

**@Override** **Sobrescreve o Extends**

```
protected void configure(HttpSecurity http) throws Exception {
```

```
    http.authorizeRequests()
```

```
        .antMatchers("/").permitAll();
```

**na tela inicial todos tem acesso (PERMIT ALL)**

```
        .antMatchers("/login").permitAll();
```

**na tela inicial todos tem acesso (PERMIT ALL)**

**É POSSIVEL EM LOGIN, COLOCAR QUE SÓ PODEM FAZER HttpMethod Post, definindo envio de dados somente**

```
        .antMatchers("/managers").hasAnyRole("MANAGERS");
```

**Nas paginas atribuídas a Managers, SÓ MANAGERS!**

```
        .antMatchers("/users").hasAnyRole("USERS","MANAGERS");
```

**nas paginas users, TODOS PODEM ACESSAR!**

```
        .anyRequest().authenticated().and().formLogin();
```

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io

## Tipos de Criptografia

- Use {bcrypt} for BCryptPasswordEncoder (mais comum)
- Use {noop} for NoOpPasswordEncoder
- Use {pbkdf2} for Pbkdf2PasswordEncoder
- Use {scrypt} for SCryptPasswordEncoder
- Use {sha256} for StandardPasswordEncoder

## Autenticação com Banco de Dados – Auth Database

Para acessar banco de dados, é necessário dependência do banco de dados que ira implementar

Dependencia do Potgress/H2/MySql...

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

Definir dentro de Application Properties

```
##H2 Database Connection Properties
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=sa
spring.jpa.show-sql: true
spring.jpa.hibernate.ddl-auto=update
spring.h2.console.enabled=true
```

database-platform → Plataforma

nome do banco

usuário e senha

definindo que quero visualizar toda a aplicação

definindo que o H2 vai estar definido via Console Web

Dependencia do Data JPA

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io

Agora é necessário ter uma classe de usuário com tipo de acesso em Role

```
@Entity //Declarando uma entidade
@Table(name = "tab_user") //Definindo Tabela
public class User {
    @Id //Definindo PK
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_user")
    private Integer id;
    @Column(length = 50, nullable = false) // Nao pode ser nula
    private String name;
    @Column(length = 20, nullable = false)
    private String username;
    @Column(length = 100, nullable = false)
    private String password;
    @ElementCollection(fetch = FetchType.EAGER)
    @CollectionTable(name = "tab_user_roles", joinColumns = @JoinColumn(name = "user_id"))
    //^^-> dizendo que essa lista vai ser a lista de Roles de um usuario, que pode ser editada, por ser uma list
    @Column(name = "role_id")
    private List<String> roles = new ArrayList<>();

    public User(){
```

Explorando o Spring Data JPA temos uma query override, que busca um usuário sua respectiva lista de perfis (roles).

## UserDetailsService

A interface *UserDetailsService* é usada para recuperar dados relacionados ao usuário. Ele possui um método denominado *loadUserByUsername()* que pode ser substituído para personalizar o processo de localização do usuário.

Vamos criar uma classe **SecurityDatabaseService.java** que implementará a *UserDetailsService* para retornar um usuário para contexto de segurança conforme nosso banco de dados.

Seguindo a documentação, criando mais uma classe de configuração, fica assim

```
@Service
public class SecurityDatabaseService implements UserDetailsService { /*retorna a implementação de Spring Security*/

    @Autowired
    private UserRepository userRepository; // como o Repositorio tem a JPA Repository, ja tem os metodos implementados

    @Override
    public UserDetails loadUserByUsername(String username) {
        User userEntity = userRepository.findByUsername(username); /*retorna um usuario atraves de seu username*/
        if (userEntity == null) {
            throw new UsernameNotFoundException(username);
        }
        Set<GrantedAuthority> authorities = new HashSet<GrantedAuthority>(); //Gerado atraves do Username
        userEntity.getRoles().forEach(role -> { //Vincula o Username com a Role p
            authorities.add(new SimpleGrantedAuthority("ROLE_" + role));
        });
        UserDetails user = new org.springframework.security.core.userdetails.User(userEntity.getUsername(),
            userEntity.getPassword(),
            authorities);
        return user;
    }
}
```

Dessa forma ele controla as Roles

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)  
start.spring.io

## Definir Roles Padrão no sistema

Se quiser deixar algum Usuário como Default em sistema, dá para salvá-lo e defini-lo em memória, antes de subir aplicação e adicionar mais Roles!

Dessa Forma:

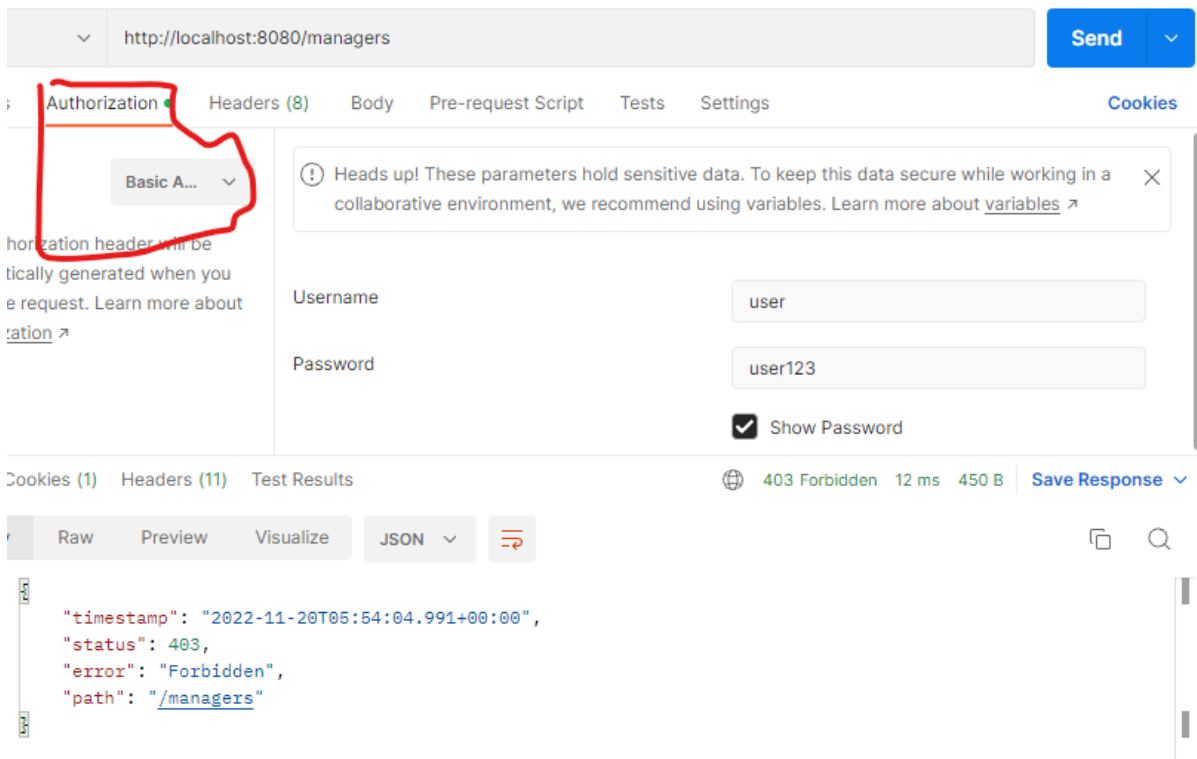
```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
```

```
@Component
public class StartApplication implements CommandLineRunner {
    @Autowired
    private UserRepository repository;
    @Transactional
    @Override
    public void run(String... args) throws Exception {
        User user = repository.findByUsername("admin");
        if(user==null){
            user = new User();
            user.setName("Andre");
            user.setUsername("santos");
            user.setPassword("admin123");
            user.getRoles().add("MANAGERS");
            repository.save(user);
        }
        user = repository.findByUsername("user");
        if(user ==null){
            user = new User();
            user.setName("USER");
            user.setUsername("user");
            user.setPassword("user123");
            user.getRoles().add("USERS");
            repository.save(user);
        }
    }
}
```

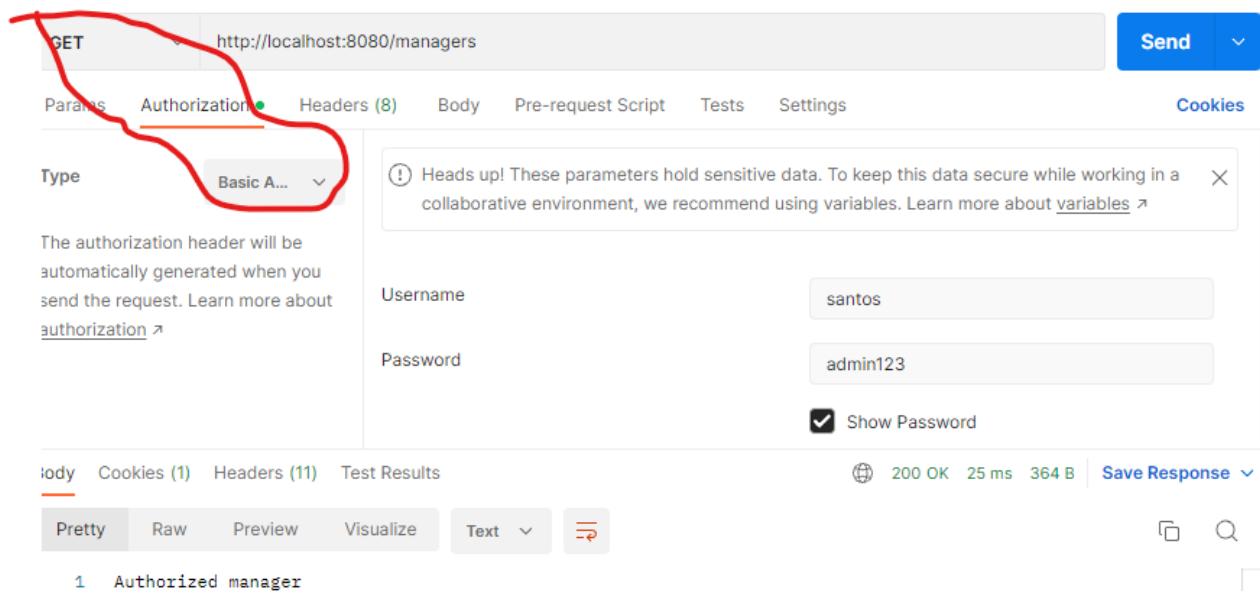
Validando implementações com POSTMAN

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io



Quando tento logar com User nos Managers, ele não tem acesso e apresenta erro, quando Login ADM no Managers, sucesso



## JWT - JSON Web Token

É uma dependência do JSON

Tecnologia independente que pode ser usado em qualquer framework

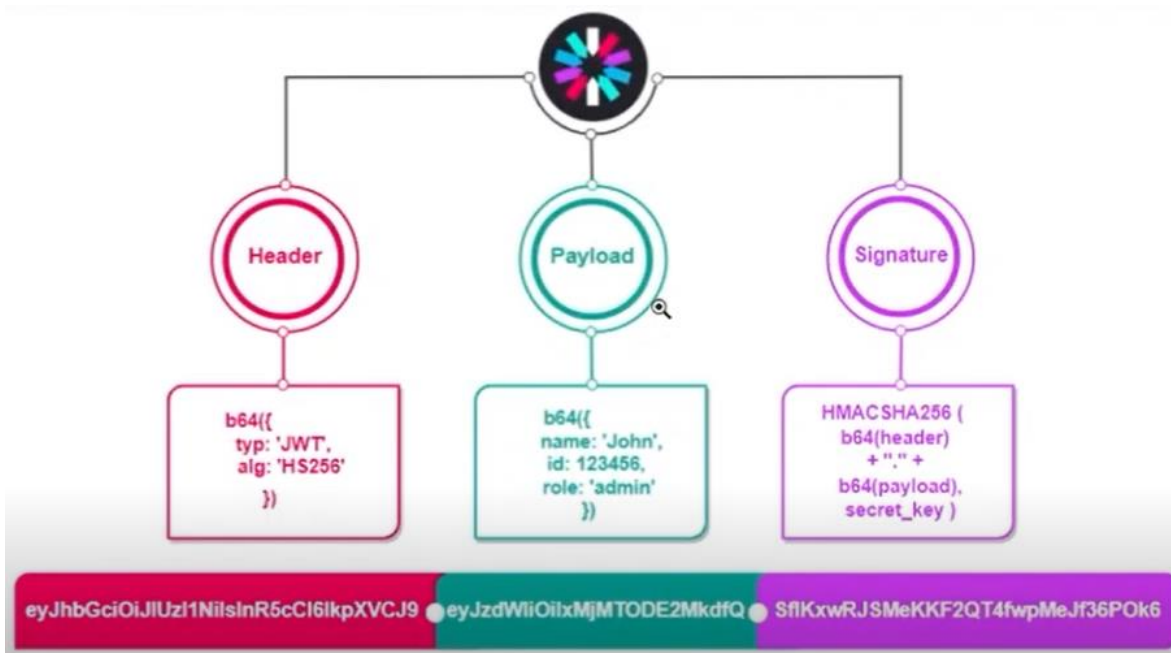
É um padrão de internet para proteção de dados e criptografia

Atraves desse arquivo JSON

Há as credencias como ID, Usuario, Perfis de acesso...

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)  
start.spring.io

Constituído



Ele é composto por esses 3 elementos

O Header

É o tipo de criptografia

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload

Estrutura do corpo contendo informações de Autenticação e Autorização do usuário

```
{
  "sub": "glysns",
  "name": "GLEYSON SAMPAIO",
  "roles": ["USERS", "MANAGERS"]
}
```

Assinatura – Assinatura

Valida a autenticidade de toda a estrutura

O JWT é um Token, e o Spring faz as validações e controla ele

Quero validar se esse usuário está expirado, se está conseguindo utilizar mesmo com ele expirado

O JWT é um objeto, que convertido, forma um Token

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)  
start.spring.io



Por ser um objeto, deveria ser declarado e configurado

Dependências com JWT

WEB – SECURITY – JPA – H2/POSTGRESS...;

Lembrando de configurar conforme dito acima, em título Dependência com Banco de Dados

Alem disso baixar a dependência do JSON JWT → JJWT (achara em MVN Repositorios)



Configurando Token

Logo após feito essas configurações, é necessário Definir uma classe usuário,

Apartir dela, seu repositório e seu serviço que vai definir suas regras de negócio e seu controller

[JWT - JSON Web Token - Spring Framework \(gitbook.io\)](https://gitbook.io/jwt-spring-framework)

Esse controller vai fazer a requisição via Post e cadastrar um novo usuário

Quando definido tendo uso de JWT

Configura os filtros

Acompanhando as requisições

CONFIGURANDO TOKEN

Ir na camada segurança, e definir o token, como declarado pela documentação

```
public class JWTObject {
    private String subject; //nome do usuario
    private Date issuedAt; //data de criação do token
    private Date expiration; // data de expiração do token
    private List<String> roles; //perfis de acesso

    //getters e setters

    public void setRoles(String... roles){
        this.roles = Arrays.asList(roles);
    }
}
```

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io

A partir de um Creator, ele faz as requisições dessa Classe, para fazer a validação

Desse modo ele gera um token

```
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Configuration
@ConfigurationProperties(prefix = "security.config")
public class SecurityConfig {
    public static String PREFIX;
    public static String KEY;
    public static Long EXPIRATION;

    public void setPrefix(String prefix){
        PREFIX = prefix;
    }
    public void setKey(String key){
        KEY = key;
    }
    public void setExpiration(Long expiration){
        EXPIRATION = expiration;
    }
}
```

- security.config.prefix= prefixo do token
- security.config.key= sua chave privada
- security.config.expiration= tempo de expiração do token

*Definir no APPLICATION PROPERTIES, as configurações finais*

```
security.config.prefix=Bearer
security.config.key=SECRET_KEY
security.config.expiration=3600000
```

Prefix – Por padrão Bearer, mas pode editar

Key – A chave que vai ser utilizada na aplicação

Expiration – O tempo em que a chave vai ser expirada, em Milissegundos

Logo após, é criado uma classe

Para criar o token ou pegar o criado, para converter para Objeto e validar a credencial

JWTCreator: Responsável por gerar o Token

Para criar o token ou pegar o criado, para converter para Objeto e validar a credencial

O método pega

o Prefixo,

a chave **que vem de Properties**,

objeto que é gerado por algum usuário, através do Banco de Dados

Criando agora o Token

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io

Ele vai pegar a partir da Classe JWTObject

Data de criação, a chave, e a data de expiração, e o ultimo marcado → algoritmo de criptografia

```
String create(String prefix, String key, JWTObject jwtObject) {  
    token = Jwts.builder().setSubject(jwtObject.getSubject()).setIssuedAt(jwtObject.getIssuedAt()).setExpiration(jwtObject.getExpiration()).claim(ROLES_AUTHORITIES, checkRoles(jwtObject.getRoles())).signWith(SignatureAlgorithm.HS512, key).compact();  
    return prefix + " " + token;  
}
```

Após isso, cria o prefixo, e o token

JWTFilter – dentro de segurança

Essa classe tem a função de avaliar as estruturas do Token

Pronto

Tendo feito todo o passo a passo, bora testar a aplicação para conferir se o Passo a Passo esta Ok

Teste com Postman e h2-console

Quando feito todo o passo a passo, ate agora, confirmar se esta tudo ok

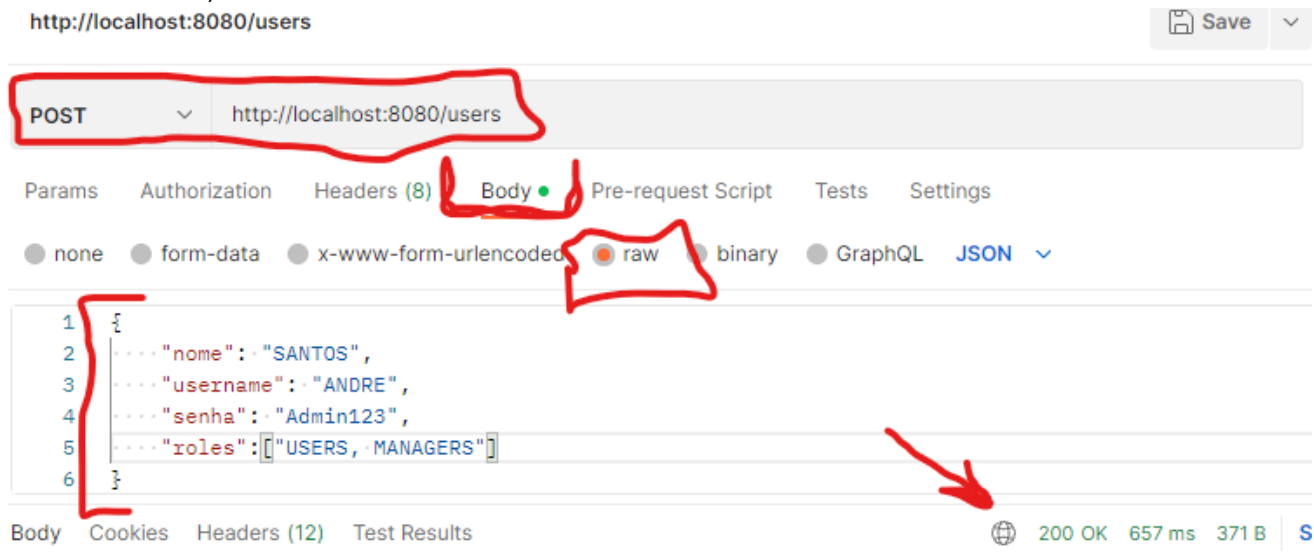
Para isso, quando acessar o LOCALHOST:8080

ele dá acesso negado

Deve Acessar o h2-console, através dele, consegue ver quantos usuários cadastrados na aplicação

Para inserir usuários, necessario acesso ao Postman

Com isso, definir usuários desse modo:



No h2 aparece desse modo:

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)  
start.spring.io

RunRun SelectedAuto completeClear

SQL statement:

SELECT \* FROM TAB\_USER

SELECT \* FROM TAB\_USER;

ID_USER	NOME	SENHA	USERNAME
1	SANTOS	\$2a\$10\$FzzNHpdKSwQWD6PoAhQWJO2y9cyH91HWwj2Qajtlm/Jtkg5.oOyVa	ANDRE

(1 row, 2 ms)

Edit

## Implementações Adicionais

Ter classe que representa solicitação de Login

Atraves desses parâmetros, realiza a geração do Token para as próximas requisições

Ter classe que representa solicitação de sessão

Nessa sessão, é determinado qual o login e senha ➔ username e token

Após criados as classes Login e Sessao, criar o LoginController

Nela, implementando os códigos, se a senha do login e usuário estiver ok, ele retorna a criação da sesao e token JWT

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

start.spring.io

Senao, ele retorna, erro ao tentar fazer o login

```
@Autowired
private UsuarioRepository repository;

@PostMapping("/login")
public Sessao login(@RequestBody Login login){
    Usuario user = repository.findByUsername(login.getUsername());
    if(user!=null) {
        boolean passwordOk = encoder.matches(login.getPassword(), user.getSenha());
        if (!passwordOk) {
            throw new RuntimeException("Senha inválida para o login: " + login.getUsername());
        }
        //Estamos enviando um objeto Sessão para retornar mais informações do usuário
        Sessao sessao = new Sessao();
        sessao.setLogin(user.getUsername());

        JWTObject jwtObject = new JWTObject();
        jwtObject.setIssuedAt(new Date(System.currentTimeMillis()));
        jwtObject.setExpiration((new Date(System.currentTimeMillis() + SecurityConfig.EXPIRATION)));
        jwtObject.setRoles(user.getRoles());
        sessao.setToken(JWTCreator.create(SecurityConfig.PREFIX, SecurityConfig.KEY, jwtObject));
        return sessao;
    }else {
        throw new RuntimeException("Erro ao tentar fazer login");
    }
}
```

Por fim

Por fim, para definir quais classes o User não tem acesso e o Manager tem

Criar o WelcomeController (uma vez já criada mas para outro projeto nessa aula)

So para confirmar se o Usuario teria acesso a classe atribuída como Manager

Pois lembrando, que como foi declarado “/manager”, , todos a partir deste usuário poderão acessa-lo

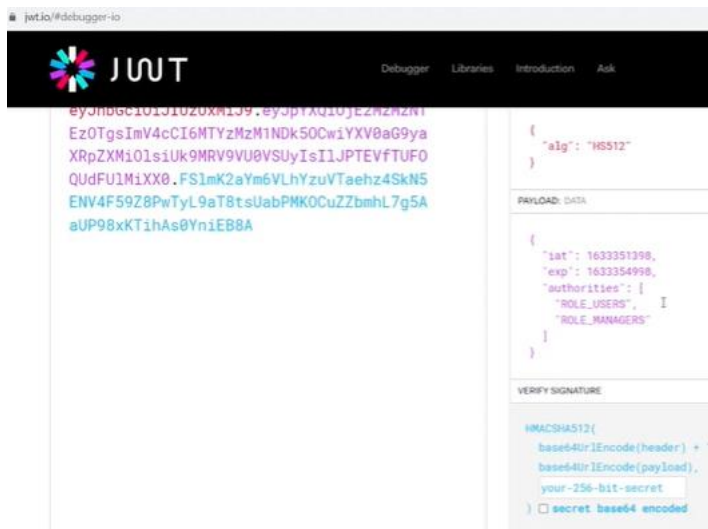
```
.addFilterAfter(new JWTFilter(), UsernamePasswordAuthenticationFilter.class)
.authorizeRequests() { ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
.antMatchers(SWAGGER_WHITELIST).permitAll()
.antMatchers(...antPatterns: "/h2-console/**").permitAll()
.antMatchers(HttpMethod.POST, ...antPatterns: "/login").permitAll()
.antMatchers(HttpMethod.POST, ...antPatterns: "/users").permitAll() /*Todas as implementações ate aki, ser
.antMatchers(HttpMethod.GET, ...antPatterns: "/users").hasAnyRole(...roles: "USERS", "MANAGERS") /*A partir daq
.antMatchers(...antPatterns: "/managers").hasAnyRole(...roles: "MANAGERS")
.anyRequest().authenticated()
```

Tradução do Token gerado

É possível visualizar o token e sua estrutura descriptografada, através do site JWT.IO

[Repositório Maven: Pesquisar/Procurar/Explorar \(mvnrepository.com\)](https://mvnrepository.com)

[start.spring.io](https://start.spring.io)



## Cadastrando usuário e realizando Login

O cadastro de Usuario é realizado através do Postman

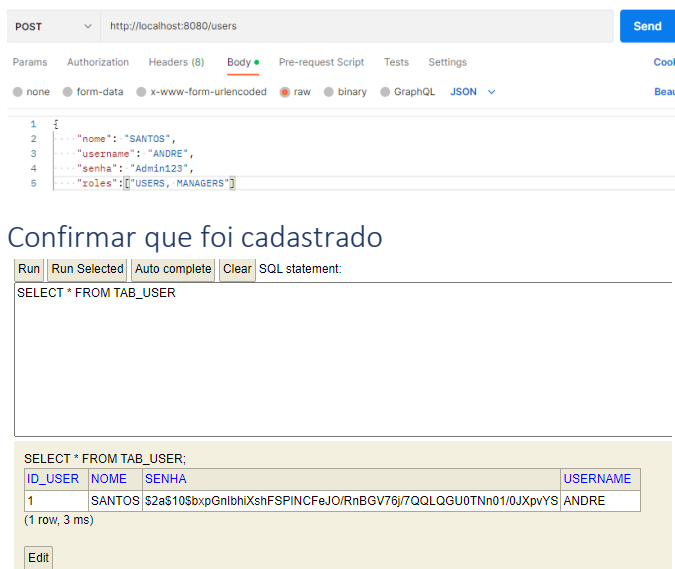
Dando Run no projeto, não vai dar para acessalo sem antes logar

Para logar, é necessário estar conectado

Para estar conectado, primeiro é necessário **ter o usuário de fato**

## Cadastrar Usuario

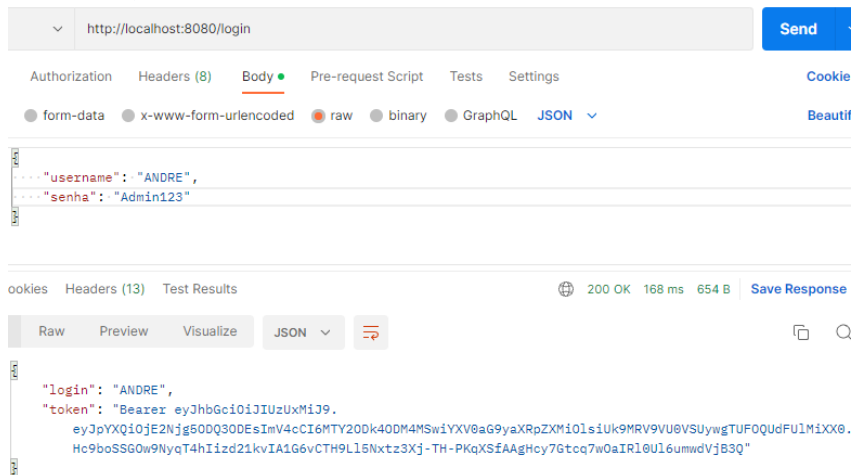
Para isso, ir no Postman, pegar o caminho para ir em Usuarios:



Repositório Maven: Pesquisar/Procurar/Explorar ([mvnrepository.com](https://mvnrepository.com))

[start.spring.io](http://start.spring.io)

## Realizar login



## GUARDE ESSE TOKEN

### Acessar Paginas, Testando Níveis de Acesso – VIA POSTMAN

Agora para visualizar as paginas, é necessario entrar com o usuário na pagina determinada

O usuário tem que entrar com o nome e o token de acesso

Atraves de AUTHORIZATION – BEARER TOKEN – TOKEN (copiar token recebido)

Se tiver a autorização de Managers, poderá acessar essa pagina, e todas vinculadas a ela

