**Como eu fiz: autenticação na API**

Primeiramente, você precisará adicionar o ***Spring Security*** no projeto, incluindo essas dependências no pom.xml:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId

<artifactId>spring-security-test</artifactId>

<scope>test</scope>

</dependency>

Depois disso, você precisará criar as classes Usuario, UsuarioRepository e AutenticacaoService no projeto, conforme demonstrado a seguir:

@Table(name = "usuarios")

@Entity(name = "Usuario")

@Getter

@NoArgsConstructor

@AllArgsConstructor

@EqualsAndHashCode(of = "id")

public class Usuario implements UserDetails {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String login;

private String senha;

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

return List.of(new SimpleGrantedAuthority("ROLE\_USER"));

}

@Override

public String getPassword() {

return senha;

}

@Override

public String getUsername() {

return login;

}

@Override

public boolean isAccountNonExpired() {

return true;

}

@Override

public boolean isAccountNonLocked() {

return true;

}

@Override

public boolean isCredentialsNonExpired() {

return true;

}

@Override

public boolean isEnabled() {

return true;

}

}COPIAR CÓDIGO

public interface UsuarioRepository extends JpaRepository<Usuario, Long> {

UserDetails findByLogin(String login);

}COPIAR CÓDIGO

@Service

public class AutenticacaoService implements UserDetailsService {

@Autowired

private UsuarioRepository repository;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

return repository.findByLogin(username);

}

}

Você também deve criar uma nova *migration* no projeto para a criação da tabela de usuários (**IMPORTANTE:** lembre-se de parar o projeto antes de criar a nova *migration*!):

create table usuarios(

id bigint not null auto\_increment,

login varchar(100) not null,

senha varchar(255) not null,

primary key(id)

);

Além disso, você também vai precisar criar a classe com as configurações de segurança da API:

@Configuration

@EnableWebSecurity

public class SecurityConfigurations {

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

return http.csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and().build();

}

@Bean

public AuthenticationManager authenticationManager(AuthenticationConfiguration configuration) throws Exception {

return configuration.getAuthenticationManager();

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

Por fim, você precisará criar uma classe Controller e um DTO para lidar com as requisições de autenticação na API:

@RestController

@RequestMapping("/login")

public class AutenticacaoController {

@Autowired

private AuthenticationManager manager;

@PostMapping

public ResponseEntity efetuarLogin(@RequestBody @Valid DadosAutenticacao dados) {

var token = new UsernamePasswordAuthenticationToken(dados.login(), dados.senha());

var authenticaon = manager.authenticate(token);

return ResponseEntity.ok().build();

}

}

public record DadosAutenticacao(String login, String senha) {

}

Para testar a autenticação, você precisará inserir um registro de usuário em seu banco de dados, na tabela de usuários:

insert into usuarios values(1, 'ana.souza@voll.med',