
Desenvolvimento WEB

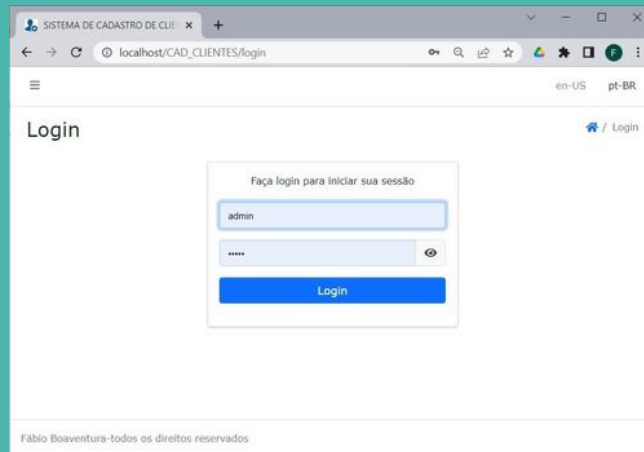
-

Full Stack Completo: Java + React

Controle de Acesso

Autenticação, Autorização e Auditoria

Parte 01



The screenshot shows a web browser window with the title 'SISTEMA DE CADASTRO DE CLIENTES'. The address bar displays 'localhost/CAD_CLIENTES/login'. The page has a header with a hamburger menu icon, language selectors for 'en-US' and 'pt-BR', and a 'Login' link. The main content area is titled 'Login' and contains a form with the instruction 'Faça login para iniciar sua sessão'. The form has two input fields: the first contains 'admin' and the second contains masked characters '****' with an eye icon for toggling visibility. Below the inputs is a blue 'Login' button. The footer of the page reads 'Fábio Boaventura-todos os direitos reservados'.

SISTEMA DE CADASTRO DE CLIENTES

localhost/CAD_CLIENTES/login

en-US pt-BR

Login

Faça login para iniciar sua sessão

admin

Login

Fábio Boaventura-todos os direitos reservados

Triplo A

- Autenticação: Quem é?
- Autorização: Pode fazer?
- Auditoria: O que fez?

Controle de Acesso :: Autenticação, Autorização e Auditoria

- **Autenticação** representa a forma de como o usuário prova quem realmente ele é: **username + password, biometrias, etc.**
- Identificação única do usuário baseada em:
 - O que o usuário sabe (senha, PIN);
 - O que o usuário possui (crachá, smart card, token);
 - O que o usuário é (impressão digital, voz, retina, íris);
 - Onde o usuário está (antes ou depois do firewall).



Controle de Acesso :: Autenticação, Autorização e Auditoria

- Para um cliente que solicita a construção de um software, é óbvio e evidente que a segurança é importante, ele não precisa dizer isso, pois é tão óbvio que você como profissional já deve saber desde o início. É como comprar um carro e dizer que você quer que seu carro tenha freios, isso é implícito.
- Então o principal objetivo é não perder muito tempo desenvolvendo a segurança de uma aplicação, para isso recorreremos a Frameworks que nos fornecem tal funcionalidade, no nosso caso, usaremos o **Spring Security**.
- O Spring Security é uma biblioteca que fornece proteção, mas também autenticação, autorização e armazenamento de senhas. Sendo que trabalha com vários protocolos para autenticação. Iremos utilizar como protocolo de autenticação o **JWT**.

Implementando a Autenticação com Spring Security

- JWT, ou JSON Web Tokens, é um padrão que, em grande parte, é usado para a segurança em APIs REST. Embora seja uma tecnologia relativamente nova, ela vem ganhando popularidade rapidamente.
- No processo de autorização do JWT, o front-end (o *client*) primeiramente envia algumas credenciais para se autenticar (nome de usuário e senha, em nosso caso).
- O servidor (a aplicação com Spring, em nosso caso), em seguida, verifica essas credenciais. Se elas forem válidas, o servidor gera um JWT e o retorna.
- Depois desse passo, o *client* precisa fornecer esse token no cabeçalho **Authorization** da solicitação, no formulário “**Bearer TOKEN**” (Token do portador). O back-end verificará a validade desse token e autorizará ou rejeitará as solicitações. O token também pode armazenar funções de usuário e autorizar as solicitações com base na autoridade fornecida.

Implementando a Autenticação com Spring Security

- No arquivo `pom.xml`, **descomente** a dependência abaixo relacionadas ao Sprint Security:

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

- Ainda no arquivo `pom.xml`, **acrescente** a dependência abaixo relacionadas ao JWT:

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.11.5</version>
</dependency>
```

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-impl</artifactId>
  <version>0.11.5</version>
</dependency>
```

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-jackson</artifactId>
  <version>0.11.5</version>
</dependency>
```

Vamos à implementação ?

- Vamos primeiro **refatorar o CRUD de cliente** para ajustar essa entidade e usarmos ela para logar na nossa aplicação.
- No nosso projeto, quem irá logar na app é um cliente e portanto, a entidade `Cliente` deve ter campos com informações para as credenciais de acesso.

Implementando a Autenticação com Spring Security

- Crie um novo pacote chamado `acesso` dentro do pacote `br.com.ifpe.oxefood.modelo`
- Crie um nova classe chamada `Usuario` dentro de `br.com.ifpe.oxefood.modelo.acesso`

Implementando a Autenticação com Spring Security

- Código da classe `Usuario`

```
package br.com.ifpe.oxefood.modelo.acesso;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import org.hibernate.annotations.SQLRestriction;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import com.fasterxml.jackson.annotation.JsonIgnore;

import br.com.ifpe.oxefood.util.entity.EntidadeNegocio;
import jakarta.persistence.Column;
import jakarta.persistence.ElementCollection;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.Table;
import jakarta.persistence.Builder;
import jakarta.persistence.Getter;
import jakarta.persistence.Setter;
import jakarta.persistence.AllArgsConstructor;
import jakarta.persistence.NoArgsConstructor;
import jakarta.persistence.Setter;

@Entity
@Table(name = "Usuario")
@SQLRestriction("habilitado = true")
@Builder
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Usuario extends EntidadeNegocio implements UserDetails {

    public static final String ROLE_CLIENTE = "CLIENTE";
```

```
@Column(nullable = false, unique = true)
private String username;

@JsonIgnore
@Column(nullable = false)
private String password;

@JsonIgnore
@ElementCollection(fetch = FetchType.EAGER)
@Builder.Default
private List<String> roles = new ArrayList<>();

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return List.of();
}

@Override
public String getUsername() {
    return username;
}

public String getPassword() {
    return password;
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return true;
}
}
```

Implementando a Autenticação com Spring Security

- Acrescente na classe `Cliente` um relacionamento com a classe `Usuario` que tem os campos para o login e senha. Acrescente o `@Fetch(FetchMode.SUBSELECT)` na lista de `EnderecoCliente`.

```
...
import org.hibernate.annotations.FetchMode;
...
public class Cliente extends EntidadeAuditavel {

    @ManyToOne
    @JoinColumn(nullable = false)
    private Usuario usuario;

    @OneToMany(mappedBy = "cliente", orphanRemoval = true, fetch = FetchType.EAGER)
    @Fetch(FetchMode.SUBSELECT)
    private List<EnderecoCliente> enderecos;

    @Column(nullable = false, length = 100)
    private String nome;

    ...
}
```

Implementando a Autenticação com Spring Security

- Acrescente `email` na `Usuario` e `password` na `Cliente` e adicione os atributos para o controle de acesso;
- Implemente o método `buildUsuario()`
- Refatore o método `buildCliente()`

```
import java.util.Arrays;
...
public class ClienteRequest {
    ...
    @NotBlank(message = "O e-mail é de preenchimento obrigatório")
    @Email
    private String email;

    @NotBlank(message = "A senha é de preenchimento obrigatório")
    private String password;
    ...

    public Usuario buildUsuario() {
        return Usuario.builder()
            .username(email)
            .password(password)
            .roles(Arrays.asList(Usuario.ROLE_CLIENTE))
            .build();
    }

    public Cliente build() {
        return Cliente.builder()
            .usuario(buildUsuario())
            .nome(nome)
            .dataNascimento(dataNascimento)
            .cpf(cpf)
            .foneCelular(foneCelular)
            .foneFixo(foneFixo)
            .build();
    }
}
```

Implementando a Autenticação com Spring Security

- No arquivo `application.properties`, acrescente o código abaixo **em vermelho**:

```
...
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=seumail@ifpe.edu.br
spring.mail.password=suasenha
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=tru

security.jwt.secret-key=3cfa76ef14937c1c0ea519f8fc057a80fcd04a7420f8e8bcd0a7567c272e007b
# 1h in millisecond
security.jwt.expiration-time=3600000
```

Implementando a Autenticação com Spring Security

- Implemente a interface chamada `UsuarioRepository` dentro de `br.com.ifpe.oxefood.modelo.acesso`

```
package br.com.ifpe.oxefood.modelo.acesso;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UsuarioRepository extends JpaRepository<Usuario, Long> {

    Optional<Usuario> findByUsername(String username);

}
```

Implementando a Autenticação com Spring Security

- Implemente a classe chamada `UsuarioService` dentro de `br.com.ifpe.oxefood.modelo.acesso`

```
package br.com.ifpe.oxefood.modelo.acesso;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import jakarta.transaction.Transactional;

@Service
public class UsuarioService implements UserDetailsService {

    @Autowired
    private UsuarioRepository repository;

    private final PasswordEncoder passwordEncoder;

    private final AuthenticationManager authenticationManager;

    public UsuarioService(UsuarioRepository userRepository, AuthenticationManager authenticationManager, PasswordEncoder passwordEncoder) {

        this.authenticationManager = authenticationManager;
        this.repository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }
}
```

continua ao lado ->

```
public Usuario authenticate(String username, String password) {

    authenticationManager.authenticate(
        new UsernamePasswordAuthenticationToken(username, password));

    return repository.findByUsername(username).orElseThrow();
}

@Transactional
public Usuario findByUsername(String username) {

    return repository.findByUsername(username).get();
}

@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

    return repository.findByUsername(username).get();
}

@Transactional
public Usuario save(Usuario user) {

    user.setPassword(passwordEncoder.encode(user.getPassword()));
    user.setHabilitado(Boolean.TRUE);
    return repository.save(user);
}
}
```

Implementando a Autenticação com Spring Security

- Acrescente o código **em vermelho** abaixo no método `save` da classe `ClienteService` para salvar o usuário que será relacionado ao cliente, no momento do cadastro do cliente:

```
...
@Autowired
private UsuarioService usuarioService;
...

@Transactional
public Cliente save(Cliente cliente) {

    usuarioService.save(cliente.getUsuario());

    cliente.setHabilitado(Boolean.TRUE);
    cliente.setVersao(1L);
    cliente.setDataCriacao(LocalDate.now());

    Cliente clienteSalvo = repository.save(cliente);
    emailService.enviarEmailConfirmacaoCadastroCliente(clienteSalvo);

    return clienteSalvo;
}
```


Implementando a Autenticação com Spring Security

- Pronto, agora que temos uma entidade (Cliente) com campos para o login no sistema, iremos iniciar o código do Spring Security para implementarmos nosso controle de acesso com JWT.

Implementando a Autenticação com JWT

- Dentro do pacote `br.com.ifpe.oxefood.modelo`, crie o pacote `seguranca`
- Dentro do pacote `seguranca`, implemente a classe `JwtAuthenticationFilter`:

```
package br.com.ifpe.oxefood.modelo.seguranca;

import java.io.IOException;

import org.springframework.lang.NonNull;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import org.springframework.web.servlet.HandlerExceptionResolver;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final HandlerExceptionResolver handlerExceptionResolver;
    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;

    public JwtAuthenticationFilter(JwtService jwtService, UserDetailsService userDetailsService,
        HandlerExceptionResolver handlerExceptionResolver) {

        this.jwtService = jwtService;
        this.userDetailsService = userDetailsService;
        this.handlerExceptionResolver = handlerExceptionResolver;
    }

    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest request, @NonNull HttpServletResponse response,
        @NonNull FilterChain filterChain) throws ServletException, IOException {

        final String authHeader = request.getHeader("Authorization");

        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }

        try {
            final String jwt = authHeader.substring(7);
            final String userEmail = jwtService.extractUsername(jwt);

            Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

            if (userEmail != null && authentication == null) {
                UserDetails userDetails = this.userDetailsService.loadUserByUsername(userEmail);

                if (jwtService.isTokenValid(jwt, userDetails)) {
                    UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
                        userDetails,
                        null,
                        userDetails.getAuthorities());

                    authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                    SecurityContextHolder.getContext().setAuthentication(authToken);
                }

                filterChain.doFilter(request, response);
            } catch (Exception exception) {
                handlerExceptionResolver.resolveException(request, response, null, exception);
            }
        }
    }
}
```

Implementando a Autenticação com JWT

- Dentro do pacote `seguranca`, implemente a classe `JwtService`:

o código da classe continua no próximo slide

```
package br.com.ifpe.oxefood.modelo.seguranca;

import java.security.Key;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;

@Service
public class JwtService {

    @Value("${security.jwt.secret-key}")
    private String secretKey;

    @Value("${security.jwt.expiration-time}")
    private long jwtExpiration;

    public String extractUsername(String token) {

        return extractClaim(token, Claims::getSubject);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {

        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    public String generateToken(UserDetails userDetails) {

        return generateToken(new HashMap<>(), userDetails);
    }

    public String generateToken(Map<String, Object> extraClaims, UserDetails userDetails) {

        return buildToken(extraClaims, userDetails, jwtExpiration);
    }

    public long getExpirationTime() {

        return jwtExpiration;
    }
}
```

Implementando a Autenticação com Spring Security

- Dentro do pacote `seguranca`, implemente a classe `JwtService`:

```
private String buildToken(Map<String, Object> extraClaims, UserDetails userDetails, long expiration) {  
  
    return Jwts  
        .builder()  
        .setClaims(extraClaims)  
        .setSubject(userDetails.getUsername())  
        .setIssuedAt(new Date(System.currentTimeMillis()))  
        .setExpiration(new Date(System.currentTimeMillis() + expiration))  
        .signWith(getSignInKey(), SignatureAlgorithm.HS256)  
        .compact();  
}  
  
public boolean isTokenValid(String token, UserDetails userDetails) {  
  
    final String username = extractUsername(token);  
    return (username.equals(userDetails.getUsername())) && !isTokenExpired(token);  
}  
  
private boolean isTokenExpired(String token) {  
  
    return extractExpiration(token).before(new Date());  
}  
  
private Date extractExpiration(String token) {  
  
    return extractClaim(token, Claims::getExpiration);  
}  
  
private Claims extractAllClaims(String token) {  
  
    return Jwts  
        .parserBuilder()  
        .setSigningKey(getSignInKey())  
        .build()  
        .parseClaimsJws(token)  
        .getBody();  
}  
  
private Key getSignInKey() {  
  
    byte[] keyBytes = Decoders.BASE64.decode(secretKey);  
    return Keys.hmacShaKeyFor(keyBytes);  
}  
}
```

Implementando a Autenticação com Spring Security

- Dentro do pacote

`br.com.ifpe.oxefood.config`,
crie uma nova classe chamada
`ApplicationConfiguration`

- Código da classe

`ApplicationConfiguration`:

```
package br.com.ifpe.oxefood.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import br.com.ifpe.oxefood.modelo.acesso.UsuarioRepository;

@Configuration
public class ApplicationConfiguration {

    private final UsuarioRepository userRepository;

    public ApplicationConfiguration(UsuarioRepository userRepository) {

        this.userRepository = userRepository;
    }

    @Bean
    UserDetailsService userDetailsService() {

        return username -> userRepository.findByUsername(username)
            .orElseThrow(() -> new UsernameNotFoundException("User not found"));
    }

    @Bean
    BCryptPasswordEncoder passwordEncoder() {

        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {

        return config.getAuthenticationManager();
    }

    @Bean
    AuthenticationProvider authenticationProvider() {

        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());

        return authProvider;
    }
}
```

Implementando a Autenticação com Spring Security

- Dentro do pacote `br.com.ifpe.oxefood.config`, crie uma nova classe chamada `SecurityConfiguration`
- Código da classe `SecurityConfiguration`:

```
package br.com.ifpe.oxefood.config;

import java.util.Arrays;
import java.util.List;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import br.com.ifpe.oxefood.modelo.acesso.Usuario;
import br.com.ifpe.oxefood.modelo.seguranca.JwtAuthenticationFilter;

@Configuration
@EnableWebSecurity
public class SecurityConfiguration {

    private final AuthenticationProvider authenticationProvider;
    private final JwtAuthenticationFilter jwtAuthenticationFilter;

    public SecurityConfiguration(JwtAuthenticationFilter jwtAuthenticationFilter, AuthenticationProvider authenticationProvider) {
        this.authenticationProvider = authenticationProvider;
        this.jwtAuthenticationFilter = jwtAuthenticationFilter;
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        http
            .csrf(c -> c.disable())
            .authorizeHttpRequests(authorize -> authorize
                .requestMatchers(HttpMethod.POST, "/api/cliente").permitAll()
                .requestMatchers(HttpMethod.POST, "/api/auth").permitAll()

                .requestMatchers(HttpMethod.GET, "/api-docs/*").permitAll()
                .requestMatchers(HttpMethod.GET, "/swagger-ui/*").permitAll()

                .anyRequest().authenticated()

            )
            .sessionManagement((session) -> session
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            )
            .authenticationProvider(authenticationProvider)
            .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

    public CorsConfigurationSource corsConfigurationSource() {

        CorsConfiguration configuration = new CorsConfiguration();

        configuration.setAllowedOrigins(Arrays.asList("http://localhost:3000"));
        configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE"));
        configuration.setAllowedHeaders(List.of("Authorization", "Content-Type"));
        configuration.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }
}
```

Implementando a Autenticação com Spring Security

- Dentro do pacote `br.com.ifpe.oxefood.api` , crie o pacote `acesso`
- Implemente a classe `AuthenticationRequest` :

```
package br.com.ifpe.oxefood.api.acesso;

import java.io.Serializable;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class AuthenticationRequest implements Serializable {

    private String username;

    private String password;

}
```

Implementando a Autenticação com Spring Security

- Ainda dentro do pacote `acesso`, crie e implemente a classe `AuthenticationController`.
- Código da classe `AuthenticationController`:

```
package br.com.ifpe.oxefood.api.acesso;

import java.util.HashMap;
import java.util.Map;

import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import br.com.ifpe.oxefood.modelo.acesso.Usuario;
import br.com.ifpe.oxefood.modelo.acesso.UsuarioService;
import br.com.ifpe.oxefood.modelo.seguranca.JwtService;

@RestController
@RequestMapping("/api/auth")
@CrossOrigin
public class AuthenticationController {

    private final JwtService jwtService;

    private UsuarioService usuarioService;

    public AuthenticationController(JwtService jwtService, UsuarioService usuarioService) {

        this.jwtService = jwtService;
        this.usuarioService = usuarioService;
    }

    @PostMapping
    public Map<Object, Object> signin(@RequestBody AuthenticationRequest data) {

        Usuario authenticatedUser = usuarioService.authenticate(data.getUsername(), data.getPassword());

        String jwtToken = jwtService.generateToken(authenticatedUser);

        Map<Object, Object> loginResponse = new HashMap<>();
        loginResponse.put("username", authenticatedUser.getUsername());
        loginResponse.put("token", jwtToken);
        loginResponse.put("tokenExpiresIn", jwtService.getExpirationTime());

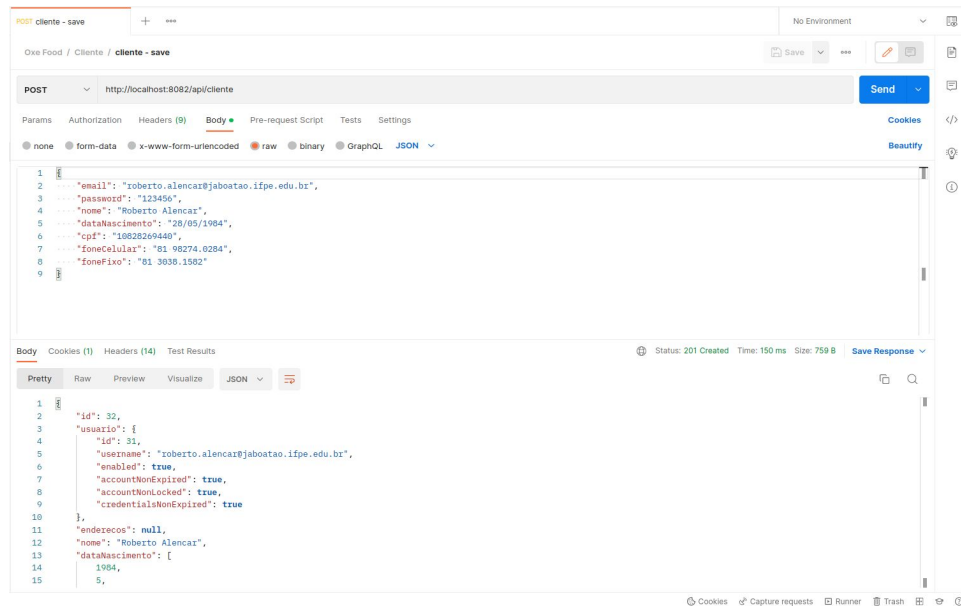
        return loginResponse;
    }
}
```


Implementando a Autenticação com Spring Security

- Faça o teste da inclusão de um novo cliente no sistema, informando o e-mail e a senha que serão utilizados para efetuar o login posteriormente.

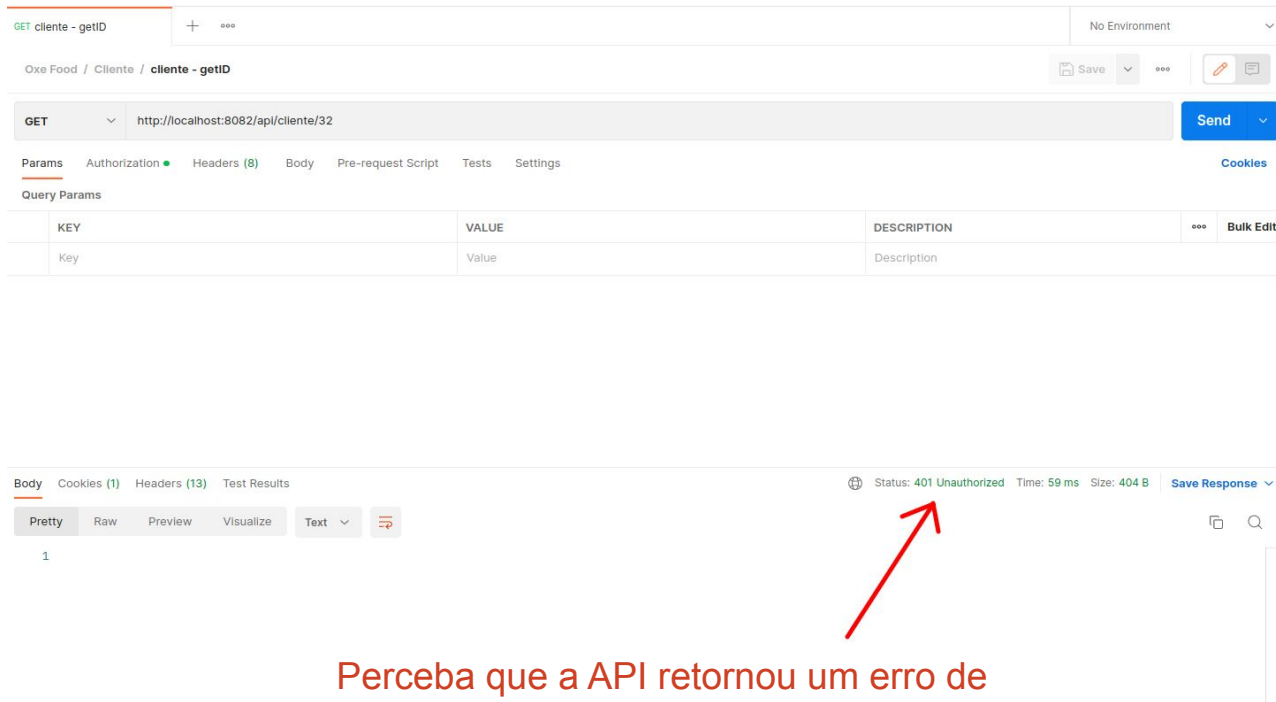
OBS.: ao tentar levantar o projeto, caso haja clientes já cadastrados no banco, irá ocorrer um erro no spring em virtude de campos obrigatórios não preenchidos.

Desta forma, é necessário limpar a tabela de clientes antes de levantar o projeto.



Implementando a Autenticação com Spring Security

- Pronto, vamos testar agora, levantar a aplicação e tente fazer uma requisição a uma rota qualquer da API:



The screenshot shows a REST client interface with a GET request to `http://localhost:8082/api/cliente/32`. The response status is **401 Unauthorized**, indicated by a red arrow pointing to the status text. The response body is empty, and the response time is 59 ms.

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies (1) Headers (13) Test Results

1

Perceba que a API retornou um erro de acesso não autorizado

Implementando a Autenticação com Spring Security

The screenshot shows a REST client interface with the following details:

- Request:** GET `http://localhost:8082/api/cliente/32`
- Authorization:** Type: Bearer Token. The token value is `eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJyb2JlcnF...`
- Response:** Status: 200 OK, Time: 26 ms, Size: 752 B. The response body is a JSON object:

```
1 {
2   "id": 32,
3   "usuario": {
4     "id": 31,
5     "username": "roberto.alencar@jaboatao.ifpe.edu.br",
6     "enabled": true,
7     "accountNonExpired": true,
8     "accountNonLocked": true,
9     "credentialsNonExpired": true
10  },
11  "enderecos": [],
12  "nome": "Roberto Alencar",
13  "dataNascimento": [
14    1984,
15    5,
```

- Teste novamente uma rota protegida, agora informando o token na aba Authorization, Type: Bearer Token.
- Ao informar um token válido, a API irá permitir acessar a rota desejada.

Dúvidas



Exercício

Implemente o controle de acesso no seu projeto do back-end.



The background features two large, solid green abstract shapes. One is a semi-circle on the left side, and the other is a more complex, organic shape on the right side.

Obrigado !