

BACK-END

Prof. Bruno Kurzawe



Aspectos básicos de segurança e escalabilidade

Importância da Segurança na Web

O que é Segurança na Web?

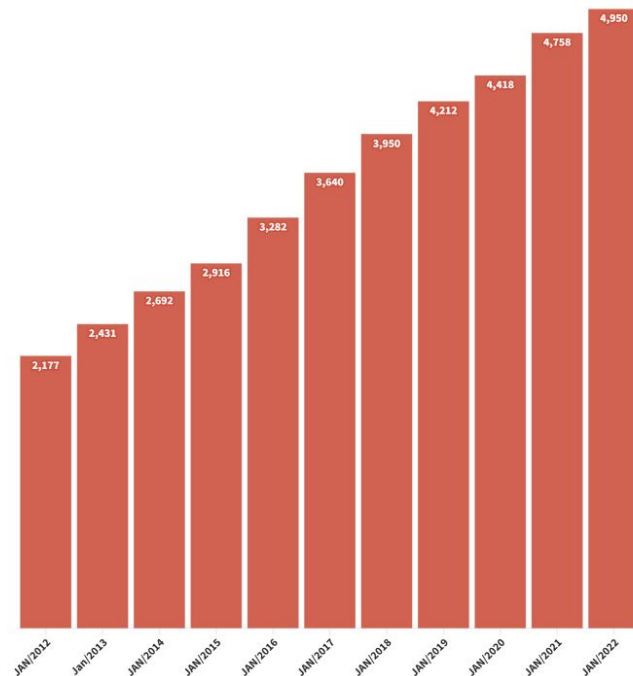
A prática de proteger aplicações, servidores e infraestrutura web contra ameaças e vulnerabilidades.

Por que é vital?

- Crescimento exponencial da internet e aplicações web.
- Maior dependência de serviços online para atividades diárias.

EVOLUÇÃO DO NÚMERO DE USUÁRIOS ATIVOS DE INTERNET

Em dez anos, número de internautas dobra no mundo (em bilhões)



Fonte: Datareportal.com (Digital 2022: Global Overview Report)

- Perda de Dados Críticos
- Dano à Reputação e Confiança
- Perdas Financeiras Diretas e Indiretas

- Informações pessoais, financeiras, propriedade intelectual.
- Consequências legais.

- Clientes podem hesitar ou evitar fazer negócios no futuro.
- Cobertura negativa na mídia.

- Custo de recuperação após uma violação.
- Multas regulatórias.
- Perda de negócios e receitas futuras.

Principais Ameaças

- Phishing
- Ataques de DDoS
- Injeção de Código

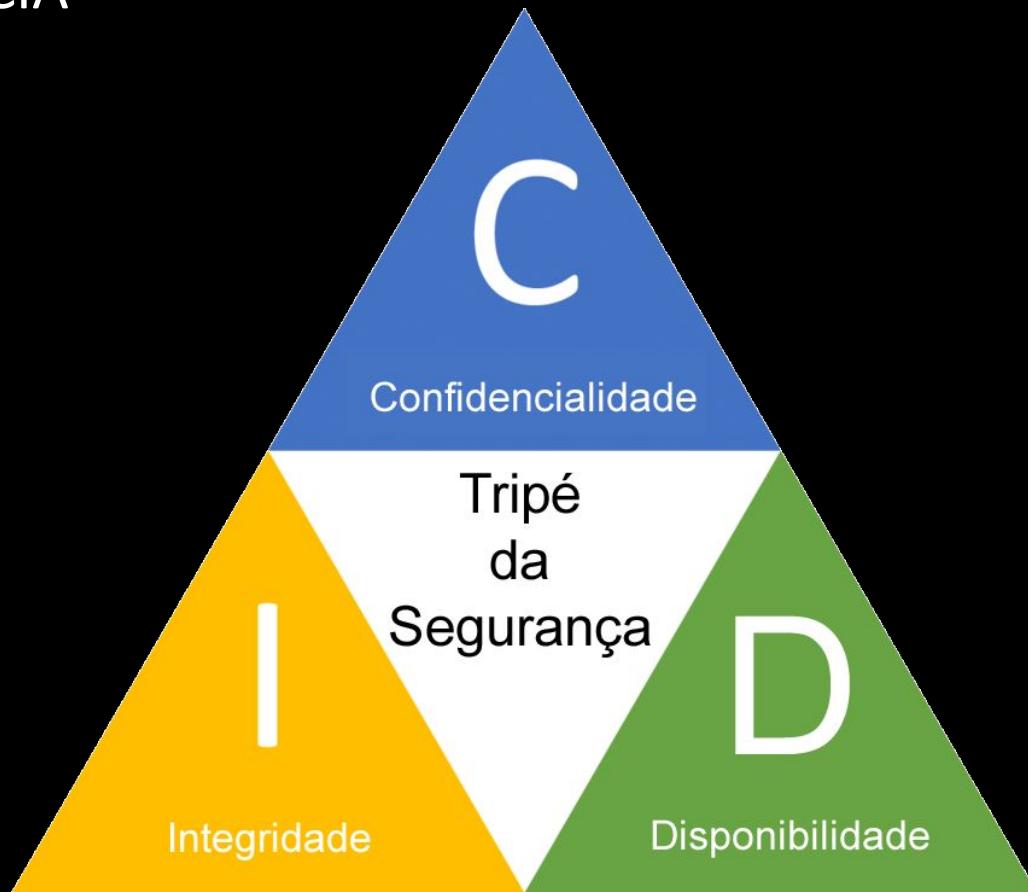
Como Proteger sua Aplicação Web

- Auditorias regulares e testes de penetração.
- Atualizações e patches frequentes.
- Educação e treinamento para funcionários.

- A segurança na web é crucial para a integridade e sucesso de aplicações e empresas.
- Ignorá-la pode resultar em consequências devastadoras.

Princípios Básicos de Segurança

Triângulo do CIA



- Garante que a informação só seja acessível a quem possui autorização.

- Garante que a informação seja mantida e transmitida sem alterações não autorizadas.

- Garante que a informação e os recursos relacionados estejam disponíveis quando necessários.

Autenticação, Autorização e Criptografia

O que são Autenticação, Autorização e Criptografia?

Por que são vitais para a segurança da informação?

Processo de confirmar a identidade de um usuário.

Processo que determina o que um usuário autenticado pode fazer.

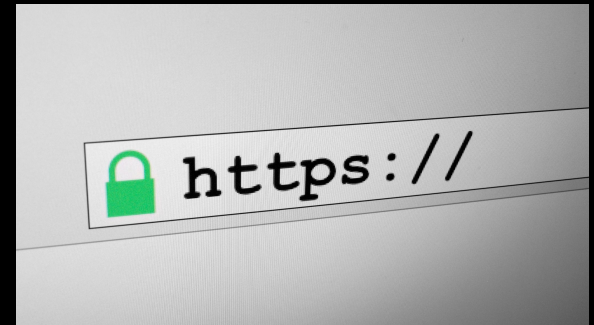
Diferença entre Autenticação e Autorização

- Autenticação: Quem é você?
- Autorização: O que você está permitido a fazer?

- Método para transformar informações de modo que só o destinatário pretendido possa entender.
- Protege dados em trânsito e em repouso.

- Protocolos de segurança para comunicações na internet.
- Garantem a privacidade e integridade dos dados transmitidos.

- Versão segura do HTTP.
- Usa SSL/TLS para proteger todas as comunicações entre o navegador e o servidor.



- Autenticação confirma a identidade.
- Autorização define permissões.
- Criptografia protege os dados.

- A combinação de Autenticação, Autorização e Criptografia é vital para uma estratégia de segurança robusta.
- Protegendo informações e garantindo acesso seguro é a chave para a confiança do usuário.

Práticas para Prevenção de Ataques

- A crescente ameaça de ataques cibernéticos.
- A necessidade de práticas robustas de prevenção.

- Ataque que insere ou "injeta" código SQL malicioso através da entrada de dados.
- Pode resultar em exposição de dados ou perda de integridade do banco de dados.

Prevenindo Injeção de SQL

- Utilizar prepared statements.
- Limitar as permissões do banco de dados.
- Validar e higienizar entradas.

XSS (Cross-Site Scripting)

- Ataque em que scripts maliciosos são injetados em sites confiáveis.
- O script é então executado no navegador da vítima.

- Validar, filtrar e higienizar entradas.
- Utilizar CSP (Content Security Policy).
- Evitar inserir dados não confiáveis diretamente no HTML.

CSRF (Cross-Site Request Forgery)

- Ataque que força o usuário a executar ações indesejadas em uma aplicação web onde ele está autenticado.

CSRF (Cross-Site Request Forgery)

Suponhamos que a vítima esteja autenticada em um site de banco e, sem encerrar a sessão, visite um site mal-intencionado. Esse site contém um script que faz uma solicitação ao site do banco para transferir dinheiro para a conta do invasor, sem que a vítima saiba. Como a vítima está autenticada no banco, a operação é realizada com sucesso.

- Utilizar tokens CSRF.
- Forçar reautenticação para ações críticas.
- Verificar cabeçalhos de referência.

- As APIs são frequentemente alvos devido à quantidade de dados que elas podem expor.

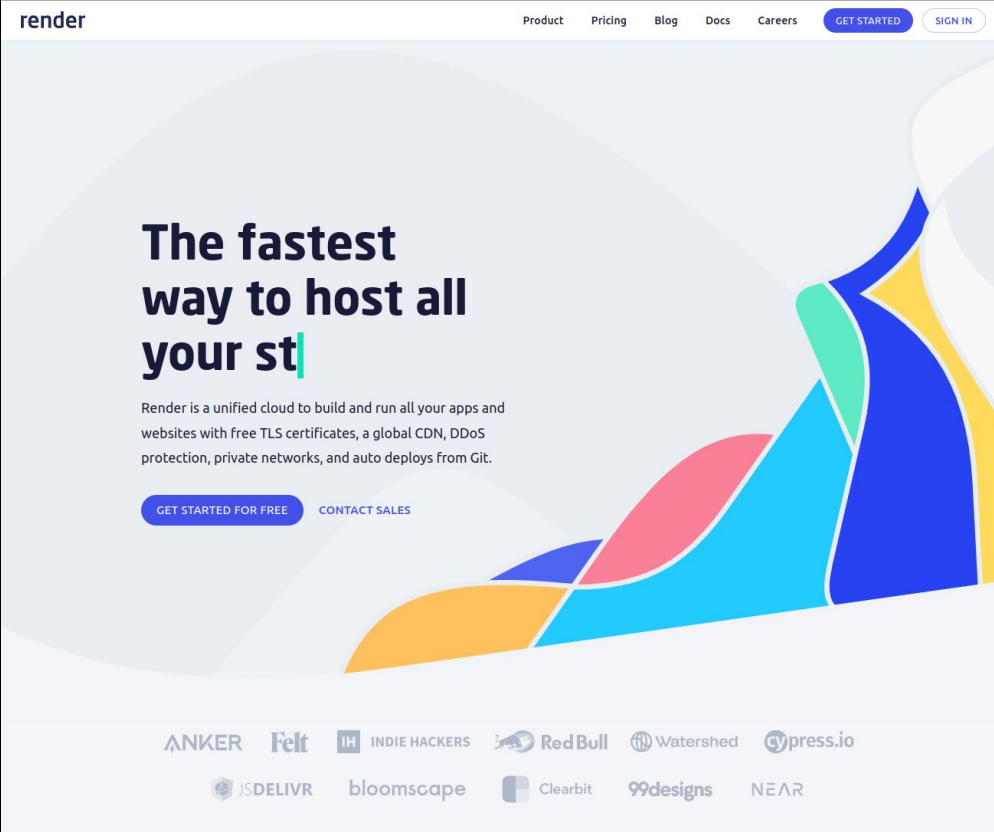
- Autenticação e autorização robustas.
- Limitar a taxa de solicitações (rate limiting).
- Validar e higienizar entradas.

- A prevenção é a primeira linha de defesa.
- A implementação de práticas recomendadas reduz significativamente os riscos.

Vamos para a parte prática

RATE-LIMIT

Criem uma conta no <https://render.com/>



The Render website landing page features a clean, modern design with a light gray background. At the top, a navigation bar includes links for Product, Pricing, Blog, Docs, and Careers, along with 'GET STARTED' and 'SIGN IN' buttons. The main headline reads 'The fastest way to host all your st', with the word 'stack' partially visible. Below this, a sub-headline states: 'Render is a unified cloud to build and run all your apps and websites with free TLS certificates, a global CDN, DDoS protection, private networks, and auto deploys from Git.' Two buttons, 'GET STARTED FOR FREE' and 'CONTACT SALES', are positioned below the text. To the right, a large, colorful abstract graphic composed of overlapping geometric shapes in blue, yellow, red, and cyan is visible. At the bottom, a row of logos for partner companies is displayed, including ANKER, Felt, IH INDIE HACKERS, Red Bull, Watershed, cypress.io, JSDELIVR, bloomscape, Clearbit, 99designs, and NEAR.

render

Product Pricing Blog Docs Careers [GET STARTED](#) [SIGN IN](#)

The fastest way to host all your st

Render is a unified cloud to build and run all your apps and websites with free TLS certificates, a global CDN, DDoS protection, private networks, and auto deploys from Git.

[GET STARTED FOR FREE](#) [CONTACT SALES](#)

ANKER Felt IH INDIE HACKERS Red Bull Watershed cypress.io

JSDELIVR bloomscape Clearbit 99designs NEAR

Painel do render

render

Dashboard

Blueprints

Env Groups

Docs

Community

Help

New +



Bruno Kurzawe



Get started in minutes



Static Sites

Static Sites are automatically served over a global CDN. Add a custom domain and get free, fully-managed SSL.

New Static Site



Web Services

Web Services include zero-downtime deploys, persistent storage and PR previews. Scale up and down with ease.

New Web Service



Private Services

Private Services are only accessible within your Render network and can speak any protocol.

New Private Service



Background Workers

Background Workers are suitable for long running processes like consumers for queues and streaming.

New Worker



Cron Jobs

With **Cron Jobs**, you can schedule any command or script to run on a regular interval.

New Cron Job



PostgreSQL

Fully-managed hosted **PostgreSQL** with internal and external connectivity, and automated daily backups.

New PostgreSQL



Redis

A cloud based in-memory key value datastore. Render offers fully managed hosted **Redis instances**.

New Redis



Blueprints

A **Blueprint** specifies your Infrastructure as Code in a single file. Use it to set up all your services at once.

New Blueprint

Redis é um armazenamento de estrutura de dados em memória, usado como um banco de dados em memória distribuído de chave-valor, cache e agente de mensagens, com durabilidade opcional.

Vamos criar um serviço de REDIS

render

Dashboard

Blueprints

Env Groups

Docs

Community

Help

New +



Bruno Kurzawe



Get started in minutes



Static Sites

Static Sites are automatically served over a global CDN. Add a custom domain and get free, fully-managed SSL.

New Static Site



Web Services

Web Services include zero-downtime deploys, persistent storage and PR previews. Scale up and down with ease.

New Web Service



Private Services

Private Services are only accessible within your Render network and can speak any protocol.

New Private Service



Background Workers

Background Workers are suitable for long running processes like consumers for queues and streaming.

New Worker



Cron Jobs

With **Cron Jobs**, you can schedule any command or script to run on a regular interval.

New Cron Job



PostgreSQL

Fully-managed hosted **PostgreSQL** with internal and external connectivity, and automated daily backups.

New PostgreSQL



Redis

A cloud based in-memory key value datastore. Render offers fully managed hosted **Redis instances**.

New Redis



Blueprints

A **Blueprint** specifies your Infrastructure as Code in a single file. Use it to set up all your services at once.

New Blueprint

Vamos criar um serviço de REDIS



render

DashboardBlueprintsEnv GroupsDocsCommunityHelp

New +

Bruno Kurzawe

New Redis Instance

Read the docs

Name
A unique name for your Redis instance.

satc-loja

Region
The [region](#) where your Redis instance runs.

Oregon (US West)

Maxmemory Policy
The [eviction policy](#) for when your Redis' memory is full.

allkeys-lru (recommended for caches)

Please [enter your payment information](#) to select an instance type with higher limits.

Instance Type	RAM	Connection Limit	Persistence	Price
<input checked="" type="radio"/> Free	25 MB	50	✗	\$0 / month
<input type="radio"/> Starter	256 MB	250	✓	\$10 / month
<input type="radio"/> Standard	1 GB	1,000	✓	\$32 / month
<input type="radio"/> Pro	5 GB	5,000	✓	\$135 / month
<input type="radio"/> Pro Plus	10 GB	10,000	✓	\$250 / month

Need a [custom instance type](#)? We support up to 512 GB RAM.

Create Redis

Vamos criar um serviço de REDIS

General

Name satc-loja [Edit](#)

Creating since a minute ago

Status ● Available

Maxmemory Policy allkeys-lru [Edit](#)

Region Oregon

Redis Instance

Instance Type

Free • RAM 25 MB Connection Limit 50

[Update](#)



A credit card is required to change instance types.

[Add payment information](#)

Vamos criar um serviço de REDIS

Connections

Internal Redis URL `redis://red-ckv5pga37rbc73es4hhg:6379`

External Redis URL



.....

Redis CLI Command



.....

Access Control

1 IP range is allowed from outside of your private network.

Sources are specified [CIDR block notation](#).

Source

0.0.0.0/0

asdsadad



Test an IP address



+ Add source

Cancel

Save

Configurar um rate-limit (limite de taxa) em uma aplicação Spring Boot é uma excelente maneira de proteger sua API contra abuso. Uma biblioteca popular que fornece essa funcionalidade é a `spring-boot-starter-data-redis`. Essa biblioteca permite implementar um rate limit baseado em Redis.

Adicione as seguintes dependências no .pom

```
<!-- Spring Boot Data Redis Starter -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>

<dependency>
  <groupId>redis.clients</groupId>
  <artifactId>jedis</artifactId>
  <version>3.7.0</version>
</dependency>
```

Vamos criar a classe RateLimitInterceptor

```
m pom.xml (satc-loja) × application.properties × RateLimitInterceptor.java ×
13 @Component
14 public class RateLimitInterceptor extends HandlerInterceptorAdapter {
15
16     private static final int MAX_REQUESTS_PER_HOUR = 100; // Ajuste conforme necessário
17     private final RedisTemplate<String, String> redisTemplate;
18
19     @Autowired
20     public RateLimitInterceptor(RedisTemplate<String, String> redisTemplate) {
21         this.redisTemplate = redisTemplate;
22     }
23
24     @Override
25     public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {
26         String clientId = getClientId(request);
27         String requestCountKey = clientId + ":requests";
28
29         Long requests = redisTemplate.opsForValue().increment(requestCountKey, 1);
30         if (requests == 1) {
31             redisTemplate.expire(requestCountKey, 1, TimeUnit.HOURS);
32         }
33
34         if (requests > MAX_REQUESTS_PER_HOUR) {
```



Registre o Interceptor, crie uma classe WebMvcConfig

```
@Configuration
public class WebMvcConfig implements WebMvcConfigurer {

    private final RateLimitInterceptor rateLimitInterceptor;

    @Autowired
    public WebMvcConfig(RateLimitInterceptor rateLimitInterceptor) {
        this.rateLimitInterceptor = rateLimitInterceptor;
    }

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(rateLimitInterceptor);
    }
}
```



Vamos adicionar os Beans do JedisConnection no application

```
@Bean
public RedisTemplate<String, Object> redisTemplate() {
    RedisTemplate<String, Object> template = new RedisTemplate<>();
    template.setConnectionFactory(jedisConnectionFactory());
    return template;
}

@Bean
public JedisConnectionFactory jedisConnectionFactory() {
    RedisStandaloneConfiguration redisConfig = new RedisStandaloneConfiguration();
    redisConfig.setHostName("");
    redisConfig.setPort(6379);
    redisConfig.setPassword("");
    redisConfig.setUsername("");
    JedisClientConfiguration.JedisClientConfigurationBuilder jedisClientConfig = JedisClientConfiguration.builder();
    jedisClientConfig.connectTimeout(Duration.ofSeconds(60));
    jedisClientConfig.useSsl();

    return new JedisConnectionFactory(redisConfig, jedisClientConfig.build());
}
```



Onde obter as credenciais

Connections

Internal Redis URL `redis://red-ckv5pga37rbc73es4hhg:6379` 

External Redis URL



.....

Redis CLI Command



.....



Vamos rodar a aplicação, e se tudo der certo...

429 Too Many Requests

REQUEST

METHOD: GET | SCHEME://HOST[:PORT][PATH[?QUERY]] | http://localhost:8080/api/locacoes | length: 34 byte(s)

QUERY PARAMETERS: + Add query parameter

HEADERS: + Add header | Add auth | Form | BODY: XHR does not allow payloads for GET request.

Response

429

Cache: Detected - Elapsed Time: 1.47s

HEADERS: pretty | BODY: Content

Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Length: 0 byte
Date: Sun, 29 Oct 2023 18:09:56 GMT -1s

HISTORY | ASSERTIONS | HTTP | DESCRIPTION

<input type="checkbox"/>	Oct 29, 2023, 3:09:55 PM	GET	http://localhost:8080/api/locacoes	429	cache	1.47s
<input type="checkbox"/>	Oct 29, 2023, 3:09:53 PM	GET	http://localhost:8080/api/locacoes	200	cache	10x 1.46s

Autenticação e Autorização

Dependências

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

```
<dependency>  
  <groupId>io.jsonwebtoken</groupId>  
  <artifactId>jjwt</artifactId>  
  <version>0.9.1</version>  
</dependency>
```

```
<dependency>  
  <groupId>org.springframework.security</groupId>  
  <artifactId>spring-security-test</artifactId>  
  <scope>test</scope>  
</dependency>
```

No pacote enterprise vamos criar **TokenRefreshException**

```
@ResponseStatus(HttpStatus.FORBIDDEN)
public class TokenRefreshException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public TokenRefreshException(String token, String message) {
        super(String.format("Failed for [%s]: %s", token, message));
    }
}
```

Dentro do model, vamos criar um pacote **security**

```
package com.satc.satcloja.model.security;  
  
public enum ERole {  
    ROLE_USER,  
    ROLE_MODERATOR,  
    ROLE_ADMIN  
}
```

Dentro do security criar a enum ERole

No pacote security, criar RefreshToken

```
@Entity(name = "refreshtoken")
public class RefreshToken {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @OneToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User user;

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

    @Column(nullable = false)
    private Instant expiryDate;

    public long getId() { return id; }

    public void setId(long id) { this.id = id; }

    public User getUser() { return user; }
```



No pacote security, criar Role

```
@Entity
@Table(name = "roles")
public class Role {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Enumerated(EnumType.STRING)
    @Column(length = 20)
    private ERole name;

    public Role() {

    }

    public Role(ERole name) { this.name = name; }

    public Integer getId() { return id; }

    public void setId(Integer id) { this.id = id; }
```



No pacote security, criar User

```
@Entity
@Table( name = "users",
        uniqueConstraints = {
            @UniqueConstraint(columnNames = "username"),
            @UniqueConstraint(columnNames = "email")
        })
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank
    @Size(max = 20)
    private String username;

    @NotBlank
    @Size(max = 50)
    @Email
    private String email;

    @NotBlank
```



No pacote security, criar RefreshTokenRepository

```
package com.satc.satcloja.repository.security;

import com.satc.satcloja.model.security.RefreshToken;
import com.satc.satcloja.model.security.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface RefreshTokenRepository extends JpaRepository<RefreshToken, Long> {
    Optional<RefreshToken> findByToken(String token);

    @Modifying
    int deleteByUser(User user);
}
```



No pacote security, criar RefreshTokenService

```
@Service
public class RefreshTokenService {
    @Value("${senac.app.jwtRefreshExpirationMs}")
    private Long refreshTokenDurationMs;

    @Autowired
    private RefreshTokenRepository refreshTokenRepository;

    @Autowired
    private UserRepository userRepository;

    public Optional<RefreshToken> findByToken(String token) { return refreshTokenRepository.findByToken(token); }

    public RefreshToken createRefreshToken(Long userId) {
        RefreshToken refreshToken = new RefreshToken();

        refreshToken.setUser(userRepository.findById(userId).get());
        refreshToken.setExpiryDate(Instant.now().plusMillis(refreshTokenDurationMs));
        refreshToken.setToken(UUID.randomUUID().toString());

        refreshToken = refreshTokenRepository.save(refreshToken);
        return refreshToken;
    }
}
```



No pacote security, criar **RoleRepository**

```
package com.satc.satcloja.repository.security;

import ...

@Repository
public interface RoleRepository extends JpaRepository<Role, Long> {
    Optional<Role> findByName(ERole name);
}
```



No pacote security, criar UserRepository

@Repository

```
public interface UserRepository extends JpaRepository<User, Long> {  
    Optional<User> findByUsername(String username);  
  
    Boolean existsByUsername(String username);  
  
    Boolean existsByEmail(String email);  
}
```



No pacote controller, crie um pacote **security**

```
package com.satc.satcloja.resource.security;

import ...

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    AuthenticationManager authenticationManager;

    @Autowired
    UserRepository userRepository;

    @Autowired
    RoleRepository roleRepository;

    @Autowired
    PasswordEncoder encoder;

    @Autowired
    RefreshTokenService refreshTokenService;

    @Autowired
    JwtUtils jwtUtils;
```

Crie a classe AuthController



No pacote security, crie **payload // request**

```
public class LoginRequest {  
    @NotBlank  
    private String username;  
  
    @NotBlank  
    private String password;  
  
    public String getUsername() { return username; }  
  
    public void setUsername(String username) { this.username = username; }  
  
    public String getPassword() { return password; }  
  
    public void setPassword(String password) { this.password = password; }  
}
```



No pacote security, crie `payload // request`

```
public class SignupRequest {  
    @NotBlank  
    @Size(min = 3, max = 20)  
    private String username;  
  
    @NotBlank  
    @Size(max = 50)  
    @Email  
    private String email;  
  
    private Set<String> role;  
  
    @NotBlank  
    @Size(min = 6, max = 40)  
    private String password;  
  
    public String getUsername() { return username; }  
  
    public void setUsername(String username) { this.username = username; }
```



No pacote security, crie **payload // request**

```
package com.satc.satcloja.resource.security.payload.request;

import javax.validation.constraints.NotBlank;

public class TokenRefreshRequest {

    @NotBlank
    private String refreshToken;

    public String getRefreshToken() { return refreshToken; }

    public void setRefreshToken(String refreshToken) { this.refreshToken = refreshToken; }

}
```



No pacote security, crie `payload // response`

```
package com.satc.satcloja.resource.security.payload.response;

import java.util.List;

public class JwtResponse {
    private String token;
    private String type = "Bearer";
    private String refreshToken;
    private Long id;
    private String username;
    private String email;
    private List<String> roles;

    public JwtResponse(String accessToken, String refreshToken, Long id, String username, String email, List<String> roles) {
        this.token = accessToken;
        this.id = id;
        this.username = username;
        this.email = email;
        this.roles = roles;
        this.refreshToken = refreshToken;
    }
}
```



No pacote security, crie **payload // response**

```
package com.satc.satcloja.resource.security.payload.response;

public class MessageResponse {
    private String message;

    public MessageResponse(String message) { this.message = message; }

    public String getMessage() { return message; }

    public void setMessage(String message) { this.message = message; }
}
```



No pacote security, crie `payload // response`

```
public class TokenRefreshResponse {  
  
    private String accessToken;  
    private String refreshToken;  
    private String tokenType = "Bearer";  
  
    public TokenRefreshResponse(String accessToken, String refreshToken) {  
        this.accessToken = accessToken;  
        this.refreshToken = refreshToken;  
    }  
  
    public String getAccessToken() { return accessToken; }  
  
    public void setAccessToken(String accessToken) { this.accessToken = accessToken; }  
  
    public String getRefreshToken() { return refreshToken; }  
  
    public void setRefreshToken(String refreshToken) { this.refreshToken = refreshToken; }  
  
    public String getTokenType() { return tokenType; }  
  
    public void setTokenType(String tokenType) { this.tokenType = tokenType; }  
}
```



No nível de application, crie um pacote **security**
Dentro do pacote security, criem o pacote **jwt**
Crie a classe **WebSecurityConfig**

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    UserDetailsServiceImpl userDetailsService;

    @Autowired
    private AuthEntryPointJwt unauthorizedHandler;

    @Bean
    public AuthTokenFilter authenticationJwtTokenFilter() { return new AuthTokenFilter(); }

    @Override
    public void configure(AuthenticationManagerBuilder authenticationManagerBuilder) throws Exception {
        authenticationManagerBuilder.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}
```



No nível de application, crie um pacote **security**
Dentro do pacote security, criem o pacote **jwt**
Crie a classe **AuthTokenFilter**

```
public class AuthTokenFilter extends OncePerRequestFilter {  
    @Autowired  
    private JwtUtils jwtUtils;  
  
    @Autowired  
    private UserDetailsServiceImpl userDetailsService;  
  
    private static final Logger logger = LoggerFactory.getLogger(AuthTokenFilter.class);  
  
    @Override  
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)  
        throws ServletException, IOException {  
        try {  
            String jwt = parseJwt(request);  
            if (jwt != null && jwtUtils.validateJwtToken(jwt)) {  
                String username = jwtUtils.getUserNameFromJwtToken(jwt);  
  
                UserDetails userDetails = userDetailsService.loadUserByUsername(username);  
                UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(  
                    userDetails, null, userDetails.getAuthorities());  
                authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));  
  
                SecurityContextHolder.getContext().setAuthentication(authentication);  
            }  
        }  
    }  
}
```



No nível de application, crie um pacote **security**
Dentro do pacote security, criem o pacote **jwt**
Crie a classe **JwtUtils**

```
@Component
public class JwtUtils {
    private static final Logger logger = LoggerFactory.getLogger(JwtUtils.class);

    @Value("${senac.app.jwtSecret}")
    private String jwtSecret;

    @Value("${senac.app.jwtExpirationMs}")
    private int jwtExpirationMs;

    public String generateJwtToken(Authentication authentication) {

        UserDetailsImpl userPrincipal = (UserDetailsImpl) authentication.getPrincipal();

        return Jwts.builder()
            .setSubject((userPrincipal.getUsername()))
            .setIssuedAt(new Date())
            .setExpiration(new Date((new Date()).getTime() + jwtExpirationMs))
            .signWith(SignatureAlgorithm.HS512, jwtSecret)
            .compact();
    }

    public String getUserNameFromJwtToken(String token) {
        return Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody().getSubject();
    }
}
```



No nível de application, crie um pacote **security**
Dentro do pacote security, criem o pacote **services**
Crie a classe **UserDetailsImpl**

```
public class UserDetailsImpl implements UserDetails {  
    private static final long serialVersionUID = 1L;  
  
    private Long id;  
  
    private String username;  
  
    private String email;  
  
    @JsonIgnore  
    private String password;  
  
    private Collection<? extends GrantedAuthority> authorities;  
  
    public UserDetailsImpl(Long id, String username, String email, String password,  
        Collection<? extends GrantedAuthority> authorities) {  
        this.id = id;  
        this.username = username;  
        this.email = email;  
        this.password = password;  
        this.authorities = authorities;  
    }  
}
```



No nível de application, crie um pacote **security**
Dentro do pacote security, criem o pacote **services**
Crie a classe **UserDetailsServiceImpl**

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired
    UserRepository userRepository;

    @Override
    @Transactional
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByUsername(username)
            .orElseThrow(() -> new UsernameNotFoundException("User Not Found with username: " + username));

        return UserDetailsImpl.build(user);
    }
}
```



No application.properties configure os parametros JWT

```
# App Properties
senac.app.jwtSecret= bezKoderSecretKey
senac.app.jwtExpirationMs= 86400000
senac.app.jwtRefreshExpirationMs= 86400000
```



Vamos rodar a aplicação, e se tudo der certo...

Autenticação

DRAFT

METHOD

GET

Scheme :// Host [":" Port] [Path ["?" Query]]

http://localhost:8080/api/clientes

QUERY PARAMETERS

+ Add query parameter

HEADERS

+ Add header

Add authorization

Form

BODY

XHR does not allow payloads for GET request.

Response

401

HEADERS

pretty

Vary: Origin


Vary: Access-Control-Request-Method


Vary: Access-Control-Request-Headers

BODY

Criando um usuário

DRAFT

Save as 

METHOD **POST** 




SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]


http://localhost:8080/api/auth/signup



length: 37 byte(s)



▼ QUERY PARAMETERS

+ Add query parameter

HEADERS   **Form** 

☒ Content-Type : application/json 

+ Add header  Add authorization 

BODY  **Text** 

```
1 {  
2   "username": "admin",  
3   "password": "123456",  
4   "role": ["ADMIN"],  
5   "email": "bruno.kurzawe@betha.com.br"  
6 }
```

Criando um usuário

Response

Cache Detected - Elapsed Time: 178ms

200

HEADERS ⓘ

pretty ▼

Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
X-Content-Type-Options: nosniff

BODY ⓘ

pretty ▼


```
{  
  message: "User registered successfully!"  
}
```

[lines](#) [nums](#) [copy](#)

length: 43 bytes

Realizar login

DRAFT

Save as  ▼

METHOD **POST** ▼ SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

http://localhost:8080/api/auth/signin length: 37 byte(s)

▼ QUERY PARAMETERS

+ Add query parameter

HEADERS [?] ⚙ **Form** ▼

☒ Content-Type : application/json ×

+ Add header 🔗 Add authorization 🗑

BODY [?] **Text** ▼

```
1 {  
2   "username": "admin",  
3   "password": "123456"  
4 }
```

Realizar login

Response

Cache Detected - Elapsed Time: 293ms

200

HEADERS [?]

pretty ▼

BODY ⑦

pretty ▼

```
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 29 Oct 2023 19:50:11 GMT
```

```
{
  refreshToken: "f5b0879c-1e89-4734-8059-920e688b2f",
  id: 4,
  username: "admin",
  email: "admin@teste.com",
  roles: [
    "ROLE_USER"
  ],
  accessToken: "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG9",
  tokenType: "Bearer"
}
```

[lines nums](#) [copy](#)

length: 341 bytes

Realizar login

DRAFT

Save as

METHOD

SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

GET

http://localhost:8080/api/clientes

length: 34 byte(s)

Send

QUERY PARAMETERS

+ Add query parameter

HEADERS

Form

BODY

☒ Authorizati : Bearer eyJhbGciOiJI

+ Add header

Add authorization

XHR does not allow payloads for GET request.

Response

Cache Detected - Elapsed Time: 108ms

200

HEADERS

pretty

BODY

pretty

Vary: Origin

Vary: Access-Control-Request-Metho
d

Vary: Access-Control-Request-Heade
rs

X-Content-T... nosniff

{

content: [

{id: 29, nome: "Cliente Zero um da silva"

{id: 30, nome: "Cliente Dois um da silva"


{id: 31, nome: "Cliente Tres um da silva"

Falando um pouco de autorização

Criamos três roles ADMIN, MODERATOR e USER

Vamos alterar a classe **ClienteController**

```
}  
  
@GetMapping  
@PreAuthorize("hasRole('ADMIN') or hasRole('MODERATOR')")  
public ResponseEntity findAll(@RequestParam(required = false) String filter,  
                              @RequestParam(defaultValue = "0") int page,  
                              @RequestParam(defaultValue = "10") int size) {  
    Page<Cliente> clientes = service.buscaTodos(filter, PageRequest.of(page, size));  
    Page<ClienteDTO> clienteDTOS = ClienteDTO.fromEntity(clientes);  
    return ResponseEntity.ok(clienteDTOS);  
}
```



Vamos criar um usuário tipo USER

DRAFT

Save as ▼

METHOD: **POST** ▼

SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

http://localhost:8080/api/auth/signup

length: 37 byte(s)

▼ QUERY PARAMETERS

+ Add query parameter

HEADERS ⓘ ⚙

Form ▼

☒ Content-Type : application/json ×

+ Add header 🔍 Add authorization 🗑

BODY ⓘ

Text ▼

```
1 {  
2   "username": "normal",  
3   "password": "123456",  
4   "role": ["USER"],  
5   "email": "normal@teste.com"  
6 }
```

Vamos fazer o login do normal

DRAFT

Save as ▼

METHOD: POST

SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

http://localhost:8080/api/auth/signin

length: 37 byte(s)

▼ QUERY PARAMETERS

+ Add query parameter

HEADERS ② ⚙

Form ▼

☒ Content-Type : application/json ×

+ Add header 🔍 Add authorization 🗑

BODY ②

Text ▼

```
1 {  
2   "username": "normal",  
3   "password": "123456",  
4 }
```

Vamos fazer o login do normal

DRAFT

Save as ▾

METHOD: **GET** | SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

http://localhost:8080/api/clientes length: 34 byte(s) **Send** ▾

▼ QUERY PARAMETERS

+ Add query parameter

HEADERS ⓘ **Form** ▾ | BODY ⓘ

☒ Authorization: Bearer eyJhbGciOiJIU... ✕ 🔗

+ Add header 🔗 Add authorization 🗑

Response Elapsed Time: 380ms

403

Escalabilidade e Performance

Capacidade de um sistema, rede ou processo de lidar com o aumento da carga de trabalho ou demanda.

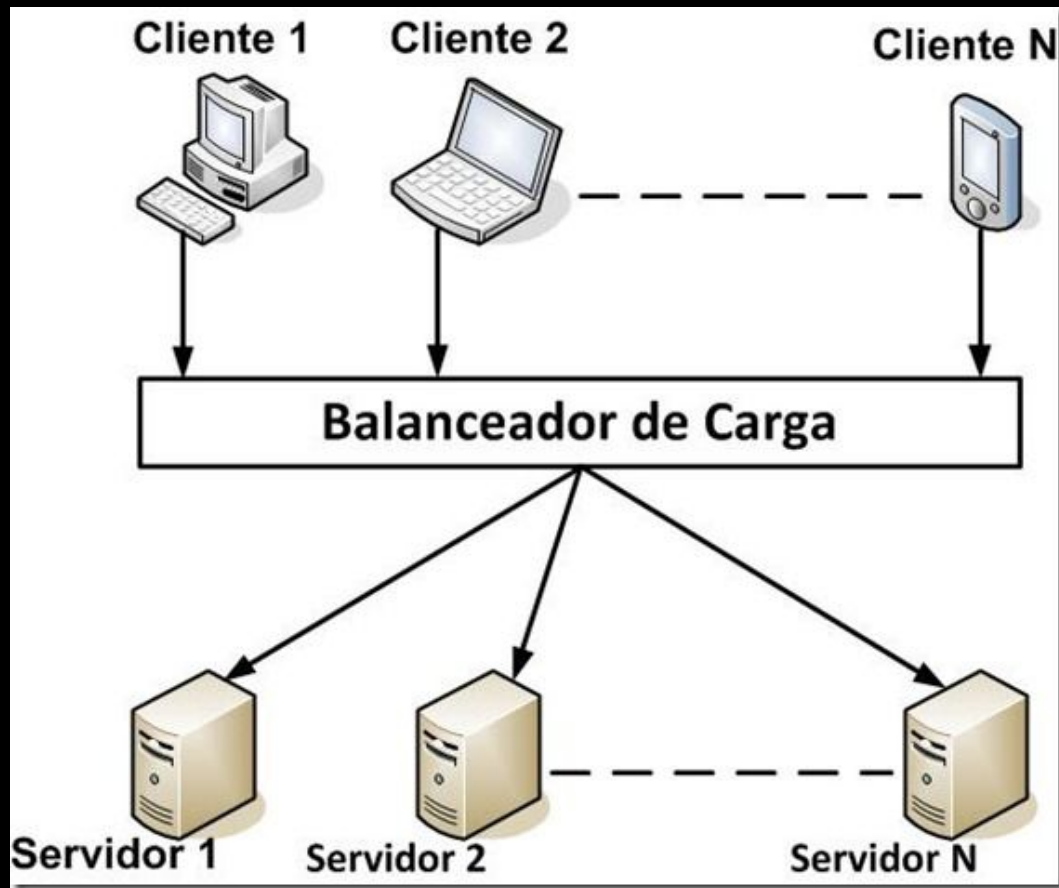
Por que a Escalabilidade é Importante?

- **Demanda Variável:** A carga em sistemas pode variar ao longo do tempo.
- **Usuários em Crescimento:** Preparação para um aumento no número de usuários.
- **Requisitos de Negócios:** Expansão para novos mercados ou funcionalidades.

- Escalabilidade Vertical: Adicionar mais recursos a uma única entidade (por exemplo, mais RAM ou CPU a um servidor).
- Escalabilidade Horizontal: Adicionar mais entidades ao sistema (por exemplo, mais servidores).

- Definição: Distribuição inteligente do tráfego de entrada entre múltiplos servidores.
- Benefícios: Aumento da disponibilidade, resiliência e capacidade de lidar com grandes volumes de tráfego.
- Ferramentas populares: NGINX, HAProxy, AWS Elastic Load Balancing.

Balanceamento de Carga



Eficiência com que um sistema atende às demandas e responde às solicitações.

- Monitore sua aplicação com ferramentas como New Relic, Datadog ou Prometheus para identificar gargalos.
- Profile seu código usando ferramentas específicas da linguagem/framework para encontrar os trechos mais lentos.

- Índices: Certifique-se de que as consultas mais frequentes estão otimizadas com índices apropriados.
- Normalização e Desnormalização: Normalizar onde a integridade é crucial; desnormalizar onde a performance é mais importante.
- Cache de Query: Use sistemas de cache como Redis ou Memcached para armazenar resultados de consultas frequentes.

- Cache de Dados: Armazena dados frequentemente usados em cache para evitar operações caras de recuperação.
- Cache de Página/Web: Utilize ferramentas como Varnish ou utilize CDNs para armazenar versões em cache de páginas da web inteiras

- Para aplicações web, minimize e compacte recursos como JavaScript, CSS e imagens.
- Use ferramentas como Webpack, UglifyJS ou PurifyCSS.

Otimização de Imagens:

- Use formatos modernos, como WebP.
- Redimensione imagens para o tamanho necessário.
- Utilize ferramentas de compressão sem perdas.

- Carregue apenas o conteúdo necessário e recupere o restante sob demanda, especialmente útil para imagens em sites ou listas em aplicações móveis.

- Use threads, processos ou técnicas assíncronas para executar tarefas em paralelo, especialmente se uma tarefa não depender da outra.

Reduza Redirecionamentos e Solicitações HTTP:

- Para aplicações web, cada redirecionamento ou solicitação HTTP adicional acrescenta latência.

Use CDNs (Content Delivery Networks)

- Distribui o conteúdo globalmente e serve o usuário a partir do ponto mais próximo.

- **Microserviços:** Decompor uma aplicação monolítica em microserviços pode ajudar a escalar partes individuais de uma aplicação.
- **Particionamento de Dados (Sharding):** Em bancos de dados, dividir os dados em diferentes servidores/databases para melhorar a performance.

- SSDs vs. HDDs: Os SSDs oferecem tempos de acesso mais rápidos.
- Otimização de Rede: Use conexões de rede rápidas, especialmente em ambientes de data center.

- SSDs vs. HDDs: Os SSDs oferecem tempos de acesso mais rápidos.
- Otimização de Rede: Use conexões de rede rápidas, especialmente em ambientes de data center.

- Algoritmos Eficientes: Escolha algoritmos adequados para o problema em questão.
- Estruturas de Dados Apropriadas: Uma estrutura de dados adequada pode fazer uma grande diferença na performance.

Vamos para a parte prática

No nível de application, crie uma classe **CacheConfig**

```
@Configuration
@EnableCaching
public class CacheConfig {

    @Autowired
    private JedisConnectionFactory jedisConnectionFactory;

    @Bean
    public RedisCacheManager cacheManager() {
        RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig()
            .entryTtl(Duration.ofMinutes(2)); // Define a duração do cache para 2 minutos

        return RedisCacheManager.builder(jedisConnectionFactory)
            .cacheDefaults(config)
            .build();
    }
}
```



Vamos adicionar o Serializable na `entityId`

```
@MappedSuperclass
public class EntityId implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @Column(name = "id", nullable = false)
    private Long id;

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }
}
```



Para ver funcionando, vamos alterar a **ClienteController**

```
@GetMapping("/{id}")
public ResponseEntity findById(@PathVariable("id") Long id) {
    System.out.println("(CONTROLLER) Buscando cliente por id: " + id);
    Cliente produto = service.buscaPorId(id);
    return ResponseEntity.ok(produto);
}
```



Para ver funcionando, vamos alterar a **ClienteService**

```
@Cacheable(value = "clienteIdCache", key = "#id")  
public Cliente buscaPorId(Long id) {  
    System.out.println("(BANCO) Buscando cliente por id: " + id);  
    return repository.findById(id).orElse(null);  
}
```



Vamos rodar e testar o findById de cliente

```
2023-10-29 18:25:37.738 INFO 1309863 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
(CONTROLER) Buscando cliente por id: 29
(BANCO) Buscando cliente por id: 29
Hibernate: select cliente0_.id as id1_0_0_, cliente0_.email as email2_0_0_, cliente0_.endereco as endereco3_0_0_, cliente0_.nome as nome4_0_0_, cliente0_.rg as rg7_0_0_ from cliente cliente0_ where cliente0_.id=?
(CONTROLER) Buscando cliente por id: 29
(CONTROLER) Buscando cliente por id: 29
```



Fim da aula 10...