

# DOCUMENTAZIONE PROGETTUALE



*Developed by* **TEAM BRAVO**

**Davide Cologgi**



<https://github.com/DavideCologgi>



<https://it.linkedin.com/in/davide-cologgi-588b62206>

**Flaviano Carlucci**



<https://github.com/fcarlucc>



<https://www.linkedin.com/in/flaviano-carlucci-1aa816298/>

**Alessio Buonomo**



<https://github.com/abuonom>



<https://www.linkedin.com/in/alessiobuonomo/>

**Manuele Longo**



<https://github.com/mlongo03>



<https://www.linkedin.com/in/manuele-longo-154979243/>

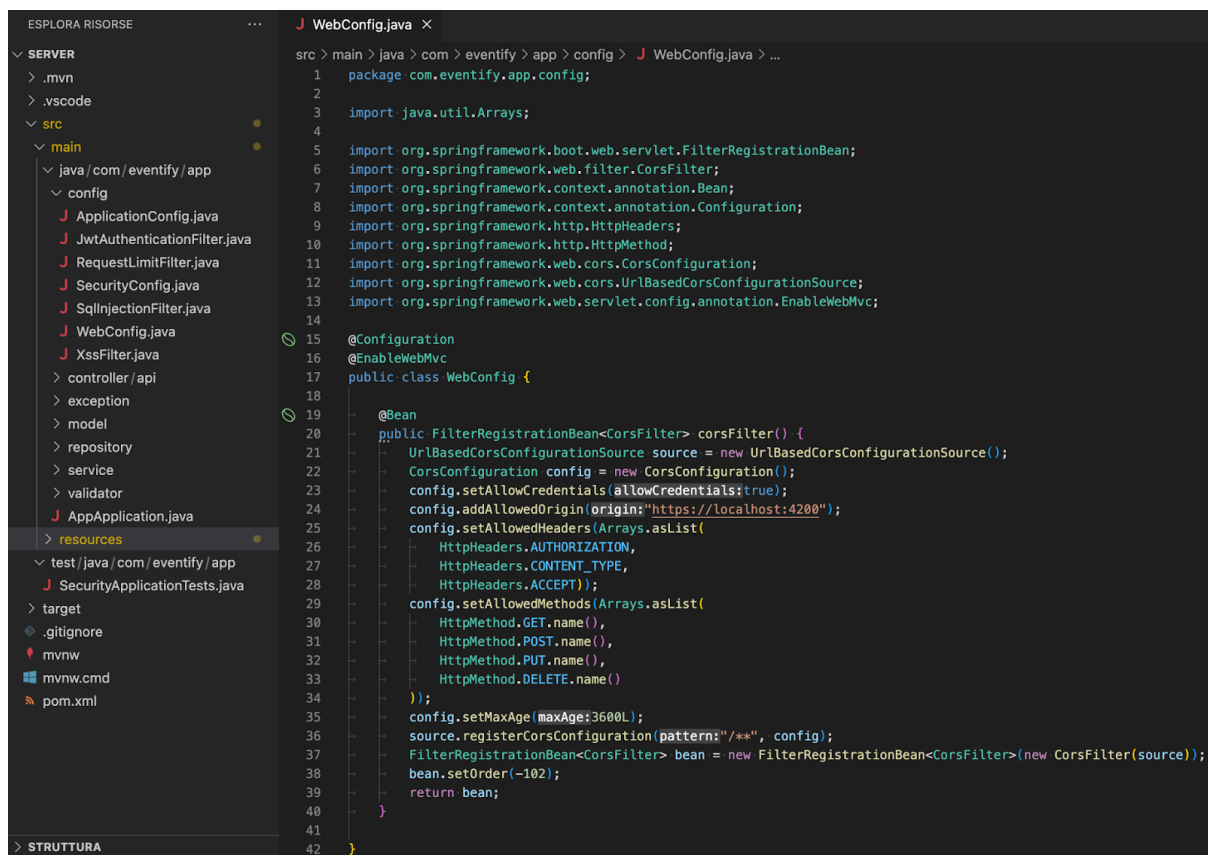
<b>Struttura del Database.....</b>	<b>5</b>
Schema "eventify" .....	6
Tabelle Principali.....	6
Tabella di Unione.....	7
Schema grafico del database.....	7
Relazioni tra le Tabelle.....	8
Utilizzo di ORM Hibernate e JPA di Spring Boot.....	8
<b>Tecnologie utilizzate.....</b>	<b>8</b>
<b>Flusso di richieste.....</b>	<b>10</b>
<b>Accesso e Autenticazione.....</b>	<b>21</b>
Gestione dei Cookie.....	21
Access Token.....	24
Refresh Token.....	24
Revocazione dell'Access Token.....	24
Logout.....	25
<b>Gestione delle Autorizzazioni.....</b>	<b>26</b>
Ruoli Utente.....	26
Dettagli dei Ruoli.....	26
Autorizzazioni Basate sulla Struttura del Sito.....	26
Modifica degli Eventi.....	26
Modifica del Profilo.....	27
Sicurezza e Controllo.....	27
<b>Protezione dalle Minacce.....</b>	<b>28</b>
Prevenzione del SQL Injection.....	28
Prevenzione di Cross-Site Scripting (XSS).....	28
Prevenzione di Cross-Site Request Forgery (CSRF).....	28
Validazione dei Dati.....	29
Memorizzazione Sicura delle Password.....	29
Minimizzazione dei Rischi OWASP Top Ten.....	29
<b>Flusso di Registrazione - sign-up.....</b>	<b>31</b>
<b>Flusso di accesso - Sign-In.....</b>	<b>33</b>
<b>Flusso di Esecuzione - Gestione degli Eventi.....</b>	<b>36</b>
1. Validazione dei Dati di Creazione dell'Evento.....	36
2. Creazione di Eventi.....	37
3. Modifica di Eventi.....	37
4. Partecipazione e Annullamento della Partecipazione.....	37
5. Eliminazione di Eventi.....	38
6. Filtri e Ricerca di Eventi.....	38
<b>Implementazione delle Notifiche in Tempo Reale con SSE.....</b>	<b>40</b>
Cos'è SSE (Server-Sent Events).....	40
Implementazione del Flusso delle Notifiche.....	41
Vantaggi di SSE.....	41
Considerazioni sulla Sicurezza.....	41

# Architettura sito web

Il nostro sito web adotta un'architettura RESTful API, che non solo separa in modo efficiente il front-end e il back-end ma promuove anche una notevole sicurezza. Questo approccio architetturale favorisce la modularità e la scalabilità, permettendo a entrambi i server di operare come entità distinte e interagire attraverso richieste HTTPS.

La separazione tra il server del front-end e il server del back-end è implementata in modo sicuro grazie a una ben definita configurazione CORS (Cross-Origin Resource Sharing). CORS è una tecnologia che regola le richieste tra domini diversi, contribuendo a proteggere l'ambiente del front-end e prevenire potenziali minacce alla sicurezza.

La nostra configurazione CORS è attentamente progettata e consente esplicitamente al front-end, che risiede su "<https://localhost:4200>," di effettuare richieste al server del back-end. La configurazione CORS è specificata nel file [WebConfig.java](#) nella directory config:



```
1 package com.eventify.app.config;
2
3 import java.util.Arrays;
4
5 import org.springframework.boot.web.servlet.FilterRegistrationBean;
6 import org.springframework.web.filter.CorsFilter;
7 import org.springframework.context.annotation.Bean;
8 import org.springframework.context.annotation.Configuration;
9 import org.springframework.http.HttpHeaders;
10 import org.springframework.http.HttpMethod;
11 import org.springframework.web.cors.CorsConfiguration;
12 import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
13 import org.springframework.web.servlet.config.annotation.EnableWebMvc;
14
15 @Configuration
16 @EnableWebMvc
17 public class WebConfig {
18
19     @Bean
20     public FilterRegistrationBean<CorsFilter> corsFilter() {
21         UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
22         CorsConfiguration config = new CorsConfiguration();
23         config.setAllowCredentials(allowCredentials:true);
24         config.addAllowedOrigin(origin:"https://localhost:4200");
25         config.setAllowedHeaders(Arrays.asList(
26             HttpHeaders.AUTHORIZATION,
27             HttpHeaders.CONTENT_TYPE,
28             HttpHeaders.ACCEPT));
29         config.setAllowedMethods(Arrays.asList(
30             HttpMethod.GET.name(),
31             HttpMethod.POST.name(),
32             HttpMethod.PUT.name(),
33             HttpMethod.DELETE.name()
34         ));
35         config.setMaxAge(maxAge:3600L);
36         source.registerCorsConfiguration(pattern:"/**", config);
37         FilterRegistrationBean<CorsFilter> bean = new FilterRegistrationBean<CorsFilter>(new CorsFilter(source));
38         bean.setOrder(-102);
39         return bean;
40     }
41
42 }
```

Viene specificato quali header e metodi sono consentiti, garantendo che solo le richieste conformi siano accettate. L'impostazione "config.setAllowCredentials(true)" consente l'inclusione di credenziali nelle richieste, garantendo un'autenticazione sicura tra i due server.

Oltre a promuovere la sicurezza, questo approccio favorisce la scalabilità, consentendo al front-end e al back-end di essere distribuiti su server diversi o su infrastrutture cloud senza mettere a rischio la sicurezza dei dati o delle transazioni. In breve, la nostra architettura RESTful API e la configurazione CORS svolgono un ruolo fondamentale nell'assicurare una separazione sicura e una comunicazione efficiente tra le componenti front-end e back-end del nostro sito web.

Inoltre, entrambi i lati del nostro sito web vantano certificazioni SSL (Secure Sockets Layer) self-signed (per lavorare in locale) per garantire una connessione sicura. Il backend, accessibile tramite "<https://localhost:8443>," è protetto da una connessione SSL che crittografa tutte le comunicazioni tra il server del backend e il client. Questa crittografia avanzata non solo protegge da potenziali intercettazioni ma garantisce anche l'integrità delle informazioni scambiate tra il front-end e il back-end.

Similmente, il frontend, raggiungibile tramite "<https://localhost:4200>," beneficia di una connessione SSL sicura. La certificazione SSL self-signed del frontend contribuisce a proteggere le comunicazioni tra il browser dell'utente e il server del frontend, assicurando che le informazioni personali siano al riparo da possibili minacce o attacchi.

L'impiego congiunto di certificazioni SSL sia per il frontend che per il backend contribuisce in modo significativo a rafforzare la sicurezza complessiva del nostro sistema. La connessione SSL garantisce l'autenticazione del server e la crittografia dei dati, riducendo al minimo il rischio di intercettazioni o manipolazioni da parte di terze parti non autorizzate durante le comunicazioni tra il front-end e il back-end.

In definitiva, l'architettura RESTful API combinata con le certificazioni SSL per entrambi i lati del nostro sito web crea un ambiente altamente sicuro in cui le comunicazioni sono protette, garantendo la confidenzialità e l'integrità dei dati degli utenti. Questa solida infrastruttura di sicurezza è fondamentale per garantire che i dati sensibili e le transazioni siano protetti da possibili minacce online.

# Struttura del Database

Il database sottostante il nostro sistema è progettato per gestire efficacemente i dati delle entità principali dell'applicazione. L'intera struttura del database è realizzata all'interno di uno schema specifico denominato "eventify", che viene creato automaticamente all'avvio dell'applicazione mediante l'esecuzione di uno script SQL personalizzato situato nella directory delle risorse (resources).

## Schema "eventify"

Lo schema "eventify" è il cuore del nostro database e contiene le tabelle principali che rappresentano le entità fondamentali dell'applicazione. L'uso di uno schema dedicato contribuisce a mantenere la separazione logica tra i dati dell'applicazione e altri dati eventualmente presenti nel database.

## Tabelle Principali

Il database comprende le seguenti tabelle principali:

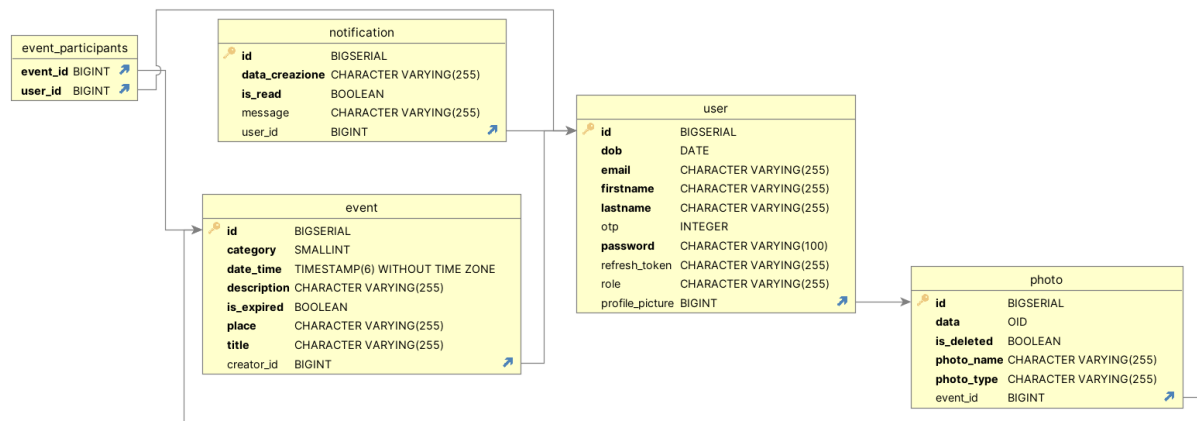
1. **User:** Questa tabella archivia i dati degli utenti registrati nell'applicazione. Contiene le seguenti informazioni:
  - id, Numero univoco che indicizza la tabella (BIGSERIAL=si incrementa automaticamente alla creazione di una nuova tabella), non è mai nullo ed è la chiave primaria
  - nome, stringa da massimo 255 caratteri, deve contenere solo caratteri alfabetici e non è mai nullo.
  - cognome, stringa da massimo 255 caratteri, deve contenere solo caratteri alfabetici e non è mai nullo.
  - email, stringa da massimo 255 caratteri, deve contenere una struttura consona ad una email, non è mai nullo ed è unico ovvero non si possono avere doppiati di email.
  - password, stringa da massimo 100 caratteri, non è mai nullo.
  - OTP, codice otp generato per identificare l'utente, field di tipo INTEGER
  - Refresh Token, refresh token generato per aggiornare l'access token dell'utente
  - role, ruolo dell'user, attualmente gli user possono avere solamente un ruolo di default USER ma è stato predisposto per future implementazione
  - profile\_picture, Riferimento alla table Photo contenente l'immagine profilo dell'user, field di tipo BIGINT (contiene l'id della table a cui è relazionato)
  - dob, date of birth, data di nascita, field di tipo DATE

2. **Notification:** La tabella "Notification" contiene informazioni sulle notifiche inviate agli utenti:
  - id, Numero univoco che indicizza la tabella (BIGSERIAL=si incrementa automaticamente alla creazione di una nuova tabella), non è mai nullo ed è la chiave primaria
  - data\_creazione, rappresenta la data di creazione della notifica
  - is\_read, field di tipo BOOLEAN che rappresenta lo stato della notifica, notifica letta=true, notifica da leggere=false
  - user\_id, Riferimento alla table dell'user a cui è destinata la notifica, field di tipo BIGINT (contiene l'id della table a cui è relazionato)
  
3. **Event:** La tabella "Event" registra i dettagli sugli eventi creati dagli utenti:
  - id, Numero univoco che indicizza la tabella (BIGSERIAL=si incrementa automaticamente alla creazione di una nuova tabella), non è mai nullo ed è la chiave primaria
  - category, contiene l'enumeratore corrispondente alla categoria dell'evento
  - date\_time, data di inizio evento
  - description, contiene la descrizione dell'evento
  - is\_expired, field di tipo BOOLEAN che consente di verificare se un evento è scaduto o meno
  - place, contiene il luogo in cui si svolgerà l'evento
  - title, contiene il titolo dell'evento
  - creator\_id, Riferimento alla table dell'user che ha creato l'evento, field di tipo BIGINT (contiene l'id della table a cui è relazionato)
  
4. **Photo:** La tabella "Photo" è responsabile della memorizzazione delle immagini associate agli eventi o agli utenti:
  - id, Numero univoco che indicizza la tabella (BIGSERIAL=si incrementa automaticamente alla creazione di una nuova tabella), non è mai nullo ed è la chiave primaria
  - data, contiene la traduzione binaria della foto
  - is\_deleted, consente di capire se una foto è stata eliminata o meno
  - photo\_name, nome della foto
  - photo\_type, tipo di foto
  - event\_id, Riferimento alla table dell'evento a cui è associata la foto, field di tipo BIGINT (contiene l'id della table a cui è relazionato)

## Tabella di Unione

1. **Event\_Participants:** Questa tabella svolge un ruolo fondamentale nell'applicazione, unendo gli eventi ai partecipanti. Consente di stabilire una relazione tra gli eventi e gli utenti che partecipano a tali eventi. Questo collegamento semplifica l'ottenimento della lista dei partecipanti di un evento specifico.

## Schema grafico del database



## Relazioni tra le Tabelle

Le tabelle all'interno dello schema "eventify" sono relazionate quando necessario. Ad esempio, l'entità "Event" può essere associata a una o più immagini dalla tabella "Photo", mentre la tabella "Event\_Participants" collega gli eventi agli utenti partecipanti. Queste relazioni consentono di recuperare e correlare i dati in base alle necessità dell'applicazione.

## Utilizzo di ORM Hibernate e JPA di Spring Boot

La struttura del database e la creazione delle tabelle sono gestite mediante l'uso di Hibernate, un framework ORM (Object-Relational Mapping), supportato dall'estensione JPA (Java Persistence API) di Spring Boot. Questo approccio consente di definire gli schemi del database utilizzando classi Java annotate con l'annotazione **@Entity**. Gli attributi di queste classi vengono tradotti direttamente in campi delle tabelle del database. Inoltre, le relazioni tra le tabelle vengono definite attraverso le annotazioni JPA.

# Tecnologie utilizzate

Il nostro sito web si avvale di un ampio set di tecnologie e framework che contribuiscono in modo significativo alla sua solidità, prestazioni e sicurezza. Di seguito, forniamo una panoramica delle principali tecnologie utilizzate nel nostro progetto:

## Backend:

### 1. Java 17 con Spring Boot 3.1.5:

- **Java 17:** L'uso della versione più recente di Java offre numerosi vantaggi in termini di sicurezza e prestazioni. Java è noto per la sua robustezza e la capacità di gestire carichi di lavoro critici. Le nuove versioni di Java includono miglioramenti significativi in termini di sicurezza e stabilità.
- **Spring Boot 3.1.5:** Spring Boot è un framework Java che semplifica la creazione di applicazioni Java, rendendo lo sviluppo più rapido ed efficiente. Spring Boot 3.1.5 è noto per il suo supporto completo alle ultime tecnologie e standard di sicurezza. Offre una serie di funzionalità di sicurezza predefinite, inclusa la protezione contro le vulnerabilità comuni come le injection SQL e i problemi di autenticazione. Inoltre, facilita la configurazione di CORS, consentendo una comunicazione sicura tra il frontend e il backend.

### 2. Database PostgreSQL v16:

- **PostgreSQL** è un sistema di gestione di database relazionali (RDBMS) open-source noto per la sua robustezza e sicurezza. La versione 16 di PostgreSQL offre numerosi miglioramenti in termini di sicurezza, tra cui la crittografia avanzata dei dati e il supporto per l'autenticazione a più fattori (MFA). Inoltre, PostgreSQL offre funzionalità di sicurezza avanzate come la gestione fine dei permessi e l'audit dei dati, contribuendo a proteggere i dati dei nostri utenti.

## Frontend:

### 1. Angular:

- **Angular** è un framework di sviluppo front-end noto per la sua robustezza e la capacità di creare interfacce utente sofisticate e reattive. Angular è particolarmente attento alla sicurezza e offre un'ampia gamma di funzionalità per mitigare le minacce comuni. Alcuni dei suoi punti di forza includono la prevenzione delle vulnerabilità XSS (Cross-Site Scripting) e il supporto per la protezione CSRF (Cross-Site Request Forgery). Queste caratteristiche si integrano perfettamente con le funzionalità di sicurezza offerte da Spring Boot, creando una combinazione potente per la protezione delle applicazioni web.

## Vantaggi in termini di Sicurezza:

- L'utilizzo di Java 17 e Spring Boot 3.1.5 garantisce un alto livello di sicurezza grazie a funzionalità predefinite di protezione, come l'Autenticazione, l'Autorizzazione e la prevenzione di vulnerabilità comuni.



- PostgreSQL v16 offre sicurezza avanzata dei dati, crittografia e controllo fine dei permessi, proteggendo i dati del nostro sito.
- Angular si concentra sulla prevenzione di vulnerabilità XSS e CSRF, riducendo il rischio di attacchi comuni.
- L'integrazione tra Angular e Spring Boot crea un ambiente coeso che semplifica la gestione della sicurezza a livello di front-end e back-end.

# Flusso di richieste

Nel nostro sito web, il flusso delle richieste segue un percorso ben definito per garantire un alto livello di sicurezza e integrità dei dati. Quando un utente compie un'azione sul frontend che richiede una comunicazione con il backend, il seguente flusso viene innescato:

## 1. Richiesta dal Frontend:

- L'utente compie un'interazione sul frontend, come fare clic su un pulsante o inviare un modulo, che genera una richiesta verso il backend.

## 2. Richiesta Axios:

- Per gestire la richiesta HTTP dal frontend al backend, utilizziamo **Axios**, una libreria JavaScript per effettuare richieste HTTP. Axios semplifica la creazione e l'invio di richieste e offre un'ampia gamma di funzionalità, inclusa la gestione automatica dei dati JSON, l'intercettazione delle richieste e delle risposte, e molto altro.

### 3. Invio della Richiesta al Backend:

- La richiesta Axios viene indirizzata verso l'endpoint specifico nel backend, che è già stato implementato come un controller. Questo endpoint è responsabile della gestione dell'azione richiesta.

#### 4. FilterChain di Spring Boot:

- Prima di raggiungere il controller, la richiesta deve passare attraverso una serie di filtri noti come **FilterChain** di Spring Boot. Questi filtri sono progettati per proteggere il sistema da varie minacce e garantire la sicurezza delle operazioni. I filtri includono:

- **SqlInjectionFilter**: la quale configurazione si può trovare nel file `SqlInjectionFilter.java` nella directory `config`:

The screenshot shows the VS Code interface. On the left, the Project Explorer displays the directory structure:

- SOURCE**: Includes folders like .mvn, .vscode, and files like pom.xml.
- SERVER**: Contains the main source code directories:
  - .main: Subdirectories include java/com/eventify/app and resources.
  - java/com/eventify/app: Contains config, controller, exception, model, repository, service, validator, and JAppApplication.java.
  - resources: test/java/com/eventify/app and SecurityApplicationTests.java.

The Editor Window on the right shows the implementation of `SqlInjectionFilter.java`:

```
src > main > java > com > eventify > app > config > J SqlInjectionFilter.java | Language Support for Java(TM) by Red Hat > 🚀 SqlInjectionFilter
package com.eventify.app.config;

import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import org.springframework.http.HttpStatus;

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

import java.io.IOException;
import java.util.regex.Pattern;
import java.util.Map;
import java.util.regex.Matcher;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Component
public class SqlInjectionFilter extends OncePerRequestFilter {

    private static final Pattern SQL_INJECTION_PATTERN = Pattern.compile(
        "(?i)(\\.\\.|'|\"|`|~|^|&#x[0-9A-F]{4}|%[0-9A-Z]{1,2})|(CREATE|DELETE|DROP|EXEC|INSERT|MERGE|SELECT|UPDATE)|\\b.*");
    private static final Logger LOGGER = LoggerFactory.getLogger(SqlInjectionFilter.class);

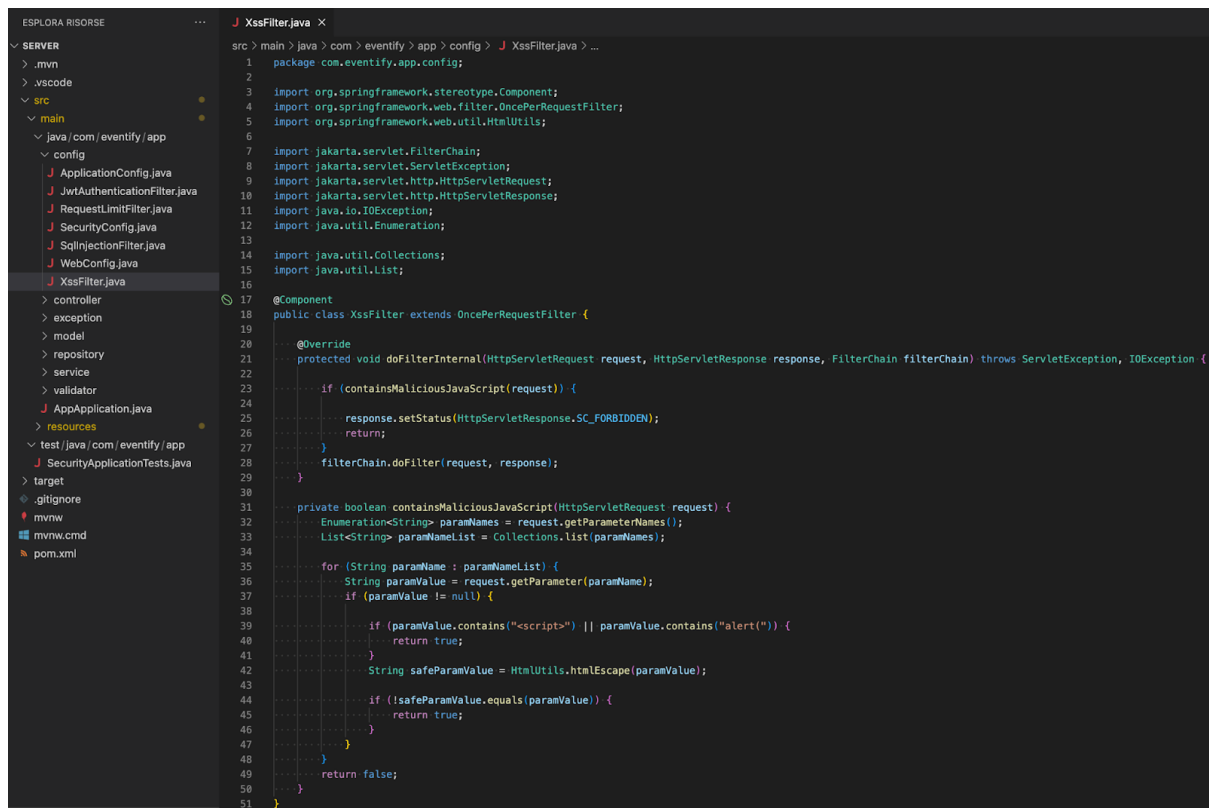
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
        Map<String, String[]> parameters = request.getParameterMap();

        for (String parameterName : parameters.keySet()) {
            String[] parameterValues = parameters.get(parameterName);
            for (String parameterValue : parameterValues) {
                Matcher matcher = SQL_INJECTION_PATTERN.matcher(parameterValue);
                if (matcher.matches()) {
                    LOG.info("SQL injection attack detected! Request: {}", request);
                    response.setStatus(HttpStatus.FORBIDDEN.value());
                    response.getWriter().write("SQL injection attack detected!");
                    return;
                }
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

Il filtro **SqlInjectionFilter** è un componente chiave per la sicurezza del tuo sistema. Questo filtro è progettato per proteggere il tuo sito web dalle vulnerabilità legate alle "injection SQL". In termini più semplici, il suo obiettivo è impedire che gli utenti malintenzionati sfruttino le richieste HTTP per eseguire query SQL dannose sul tuo database.

Ecco come funziona il filtro **SqlInjectionFilter**:

1. **Pattern di Ricerca SQL Injection:** Il filtro utilizza un pattern di espressione regolare (**Pattern.compile**) per cercare corrispondenze potenziali a segni di SQL injection nelle richieste in arrivo. Questo pattern è stato progettato per individuare stringhe che contengono segni di punteggiatura o parole chiave SQL rischiose, come "ALTER," "DELETE," "DROP," "INSERT," "SELECT," e altre.
  2. **Iterazione sui Parametri della Richiesta:** Il filtro esamina tutti i parametri presenti nella richiesta HTTP, inclusi quelli passati come parametri GET o dati POST.
  3. **Ricerca di Corrispondenze:** Per ciascun valore di parametro, il filtro verifica se contiene corrispondenze con il pattern di espressione regolare. Questo viene fatto utilizzando il metodo **matcher.matches()**. Se una corrispondenza viene trovata, il filtro rileva un potenziale tentativo di SQL injection.
  4. **Reazione alla Rilevazione:** Se una corrispondenza viene individuata, il filtro reagisce immediatamente. Registra un messaggio di avviso attraverso il logger (**LOGGER.info**) indicando che è stata rilevata un'attività sospetta legata a una possibile SQL injection.
  5. **Risposta alla Richiesta:** Successivamente, il filtro restituisce una risposta di "FORBIDDEN" (403 Forbidden) al client che ha effettuato la richiesta. La risposta contiene un messaggio che informa l'utente che è stata rilevata un'attività sospetta legata a un possibile attacco di SQL injection. Questo aiuta a prevenire l'esecuzione di comandi SQL dannosi.
  6. **Blocco della Richiesta:** Dopo aver inviato la risposta, il filtro interrompe ulteriormente l'elaborazione della richiesta e ritorna, impedendo così che la richiesta prosegua nel flusso di elaborazione standard.
- **XssFilter:** la quale configurazione si può trovare nel file `SqlInjectionFilter.java` nella directory `config`:



```
src > main > java > com > eventify > app > config > J XssFilter.java > ...
1 package com.eventify.app.config;
2
3 import org.springframework.stereotype.Component;
4 import org.springframework.web.filter.OncePerRequestFilter;
5 import org.springframework.web.util.HtmlUtils;
6
7 import jakarta.servlet.FilterChain;
8 import jakarta.servlet.ServletException;
9 import jakarta.servlet.http.HttpServletRequest;
10 import jakarta.servlet.http.HttpServletResponse;
11 import java.io.IOException;
12 import java.util.Enumeration;
13
14 import java.util.Collections;
15 import java.util.List;
16
17 @Component
18 public class XssFilter extends OncePerRequestFilter {
19
20     @Override
21     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
22
23         if (containsMaliciousJavaScript(request)) {
24             response.setStatus(HttpServletResponse.SC_FORBIDDEN);
25             return;
26         }
27         filterChain.doFilter(request, response);
28     }
29
30     private boolean containsMaliciousJavaScript(HttpServletRequest request) {
31         Enumeration<String> paramNames = request.getParameterNames();
32         List<String> paramNameList = Collections.list(paramNames);
33
34         for (String paramName : paramNameList) {
35             String paramValue = request.getParameter(paramName);
36             if (paramValue != null) {
37                 if (paramValue.contains("<script>") || paramValue.contains("alert(")) {
38                     return true;
39                 }
40                 String safeParamValue = HtmlUtils.htmlEscape(paramValue);
41                 if (!safeParamValue.equals(paramValue)) {
42                     return true;
43                 }
44             }
45         }
46         return false;
47     }
48 }
49
50
51
```

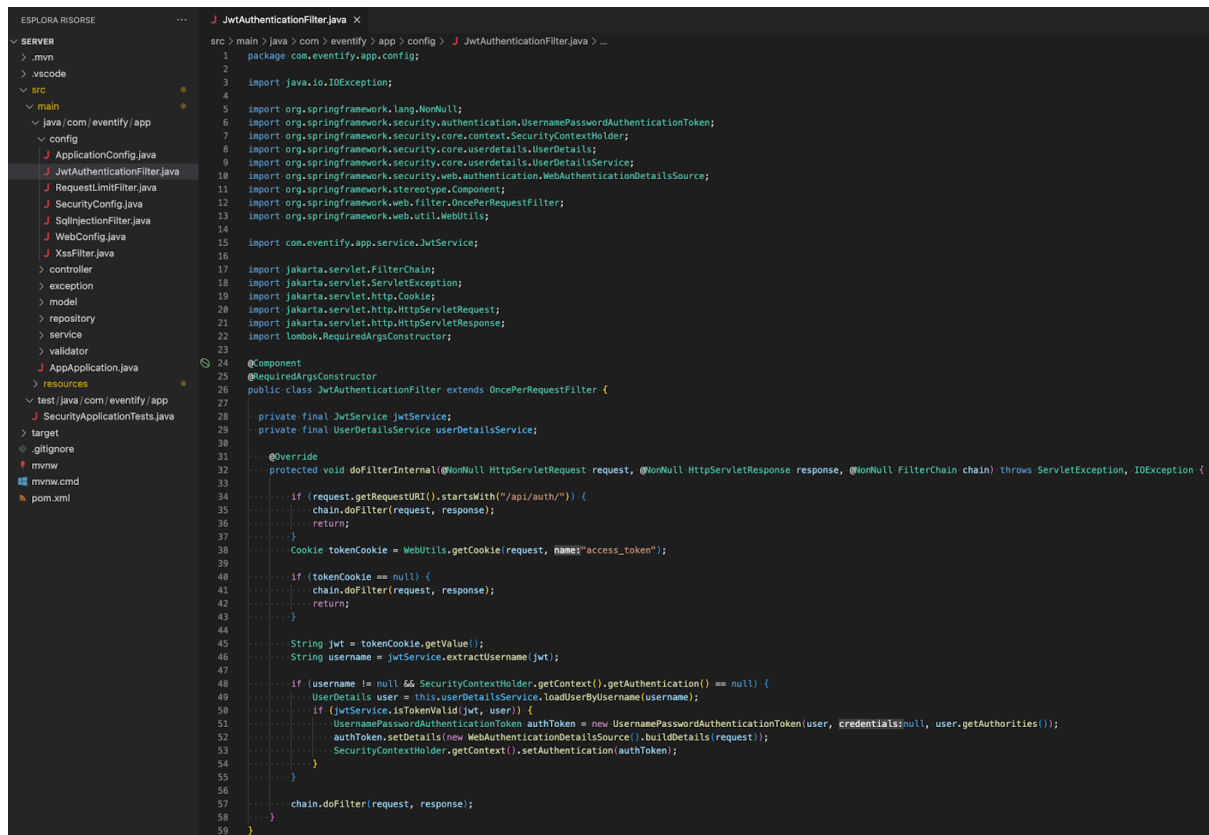
Il filtro **XssFilter** è un componente di sicurezza fondamentale che mira a mitigare le vulnerabilità legate al Cross-Site Scripting (XSS) nel tuo sito web. Il Cross-Site Scripting è una minaccia comune in cui gli attaccanti cercano di iniettare script dannosi nelle pagine web visualizzate da altri utenti. Il filtro **XssFilter** è stato progettato per rilevare e prevenire tali attacchi.

Ecco come funziona il filtro **XssFilter**:

1. **Esame dei Parametri della Richiesta:** Il filtro analizza i parametri della richiesta HTTP in arrivo, tra cui i parametri GET e POST. Questi parametri sono le informazioni inviate dal client al server tramite la richiesta.
2. **Ricerca di Contenuti Dannosi:** Per ciascun parametro, il filtro esamina il valore del parametro alla ricerca di contenuti potenzialmente dannosi. In particolare, cerca se il valore del parametro contiene sequenze di caratteri come **<script>** o **alert()**, che sono spesso associati a tentativi di eseguire script dannosi sul lato client.
3. **Prevenzione degli Attacchi XSS:** Se il filtro trova uno qualsiasi di questi contenuti potenzialmente dannosi, lo considera un segno di attacco XSS e risponde bloccando la richiesta. La risposta restituita ha uno stato "FORBIDDEN" (403 Forbidden), indicando che la richiesta è stata negata.
4. **Verifica della Sicurezza dei Dati:** Inoltre, il filtro verifica se i dati sono sicuri da utilizzare. Utilizza la funzione **HtmlUtils.htmlEscape(paramValue)** per verificare se il parametro è stato correttamente "escapato" o sanitizzato. Se il parametro contiene caratteri non sicuri, significa che potrebbe essere stato manomesso, e il filtro risponderà negando la richiesta.

5. **Flusso di Elaborazione:** Se la richiesta supera con successo tutte queste verifiche, il filtro consente alla richiesta di continuare il flusso di elaborazione standard.

- **JwtAuthenticationFilter:** la quale configurazione si può trovare nel file JwtAuthenticationFilter.java nella directory config:

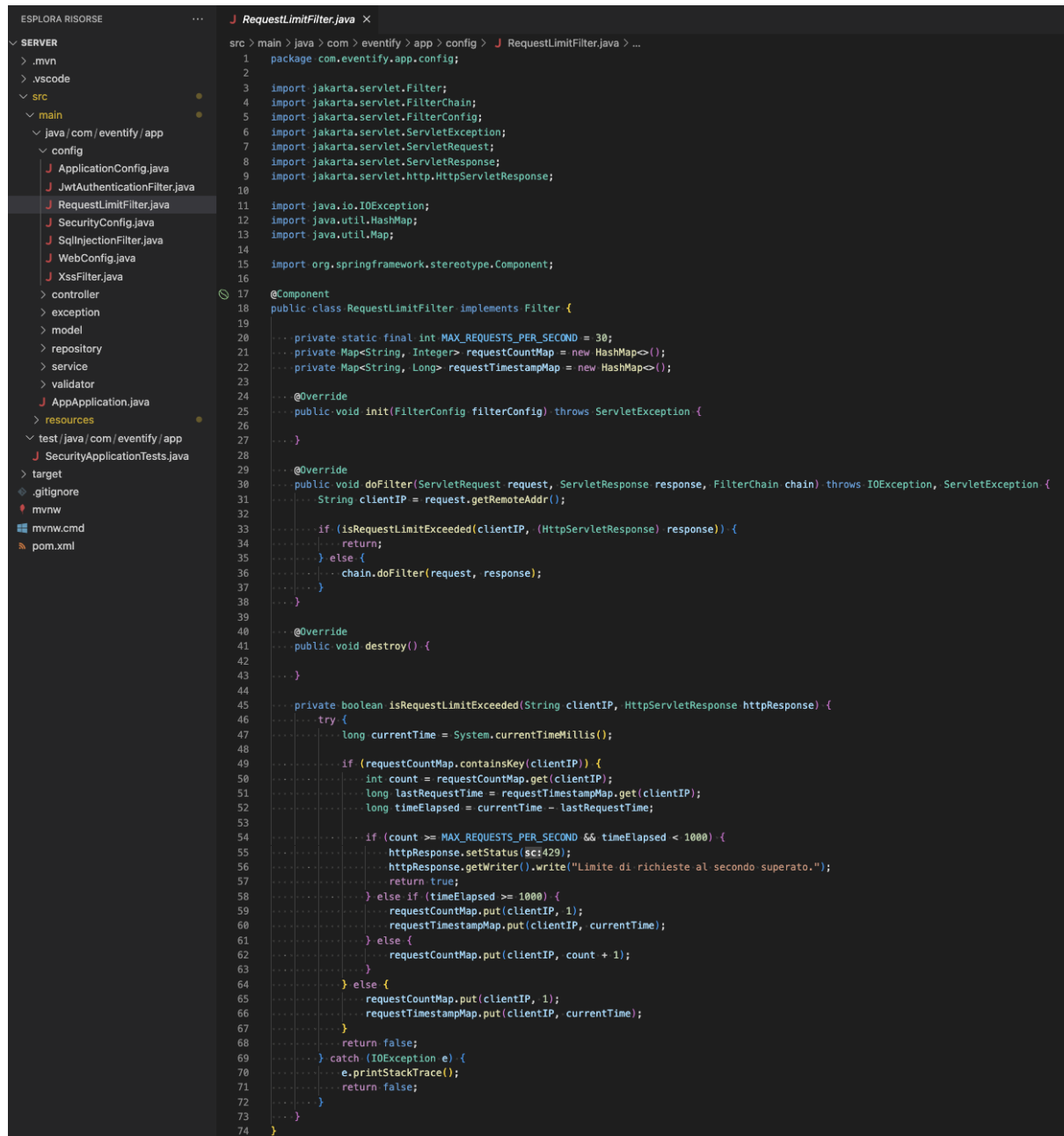


```
src > main > java > com > eventify > app > config > JJwtAuthenticationFilter.java > ...
1 package com.eventify.app.config;
2
3 import java.io.IOException;
4
5 import org.springframework.lang.NonNull;
6 import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
7 import org.springframework.security.core.context.SecurityContextHolder;
8 import org.springframework.security.core.userdetails.UserDetails;
9 import org.springframework.security.core.userdetails.UserDetailsService;
10 import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
11 import org.springframework.stereotype.Component;
12 import org.springframework.web.filter.OncePerRequestFilter;
13 import org.springframework.web.util.WebUtils;
14
15 import com.eventify.app.service.JwtService;
16
17 import jakarta.servlet.FilterChain;
18 import jakarta.servlet.ServletException;
19 import jakarta.servlet.http.Cookie;
20 import jakarta.servlet.http.HttpServletRequest;
21 import jakarta.servlet.http.HttpServletResponse;
22 import lombok.RequiredArgsConstructor;
23
24 @Component
25 @RequiredArgsConstructor
26 public class JJwtAuthenticationFilter extends OncePerRequestFilter {
27
28     private final JwtService jwtService;
29     private final UserDetailsService userDetailsService;
30
31     @Override
32     protected void doFilterInternal(@NonNull HttpServletRequest request, @NonNull HttpServletResponse response, @NonNull FilterChain chain) throws ServletException, IOException {
33
34         if (request.getRequestURI().startsWith("/api/auth/")) {
35             chain.doFilter(request, response);
36             return;
37         }
38
39         Cookie tokenCookie = WebUtils.getCookie(request, "access_token");
40
41         if (tokenCookie == null) {
42             chain.doFilter(request, response);
43             return;
44         }
45
46         String jwt = tokenCookie.getValue();
47         String username = jwtService.extractUsername(jwt);
48
49         if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
50             UserDetails user = this.userDetailsService.loadUserByUsername(username);
51             if (jwtService.isValid(jwt, user)) {
52                 UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(user, Credentials.empty(), user.getAuthorities());
53                 authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
54                 SecurityContextHolder.getContext().setAuthentication(authToken);
55             }
56
57             chain.doFilter(request, response);
58         }
59     }
60 }
```

Il filtro **JwtAuthenticationFilter** è un componente cruciale nel sistema di autenticazione del tuo sito web basato su JSON Web Tokens (JWT). Questo filtro è responsabile della gestione dell'autenticazione degli utenti attraverso i token JWT. Ecco come funziona:

1. **Controllo del Percorso della Richiesta:** All'inizio, il filtro verifica il percorso della richiesta ricevuta. Se la richiesta è destinata a un endpoint specifico che gestisce l'autenticazione (ad esempio, le richieste a "/api/auth/"), il filtro consente alla richiesta di proseguire nel flusso di elaborazione standard e non interferisce con il processo di autenticazione, questo perché tutte le richieste che riguardano l'autenticazione dell'utente come il sign-in, il sign-up, refresh token ecc... non avranno bisogno di autenticare nessun utente e di conseguenza saltano questo filtro.
2. **Ricerca del Token JWT:** Il filtro successivamente cerca un cookie chiamato "access\_token" nella richiesta HTTP. Questo cookie dovrebbe contenere un token JWT che è stato precedentemente generato e memorizzato dal tuo sistema durante l'autenticazione dell'utente.

3. **Estrazione dell'Username:** Se viene trovato un token JWT nel cookie, il filtro estrae l'username dall'interno del token. Questo username è stato precedentemente incluso nel token durante la fase di autenticazione dell'utente.
  4. **Verifica della Validità del Token:** Una volta estratto l'username, il filtro verifica se l'utente è già autenticato. Per fare ciò, utilizza il servizio **JwtService** per determinare se il token JWT è valido e non è scaduto. Inoltre, il filtro verifica se l'utente associato all'username esiste nel sistema.
  5. **Creazione dell'Autenticazione:** Se il token è valido e l'utente esiste, il filtro crea un oggetto **UsernamePasswordAuthenticationToken** per l'utente autenticato. Questo oggetto rappresenta l'autenticazione dell'utente e contiene informazioni sull'utente e i suoi ruoli.
  6. **Impostazione dell'Autenticazione nel Contesto di Sicurezza:** Infine, il filtro imposta l'oggetto **UsernamePasswordAuthenticationToken** nel contesto di sicurezza (SecurityContextHolder) del sistema. Questo indica che l'utente è stato autenticato con successo e che può accedere a risorse protette dal sistema.
  7. **Continuazione del Flusso di Elaborazione:** Dopo aver completato il processo di autenticazione, il filtro consente alla richiesta di continuare nel flusso di elaborazione standard. L'utente è ora considerato autenticato e può accedere alle risorse protette dal sistema.
- **RequestLimitFilter:** la quale configurazione si può trovare nel file RequestLimitFilter.java nella directory config:

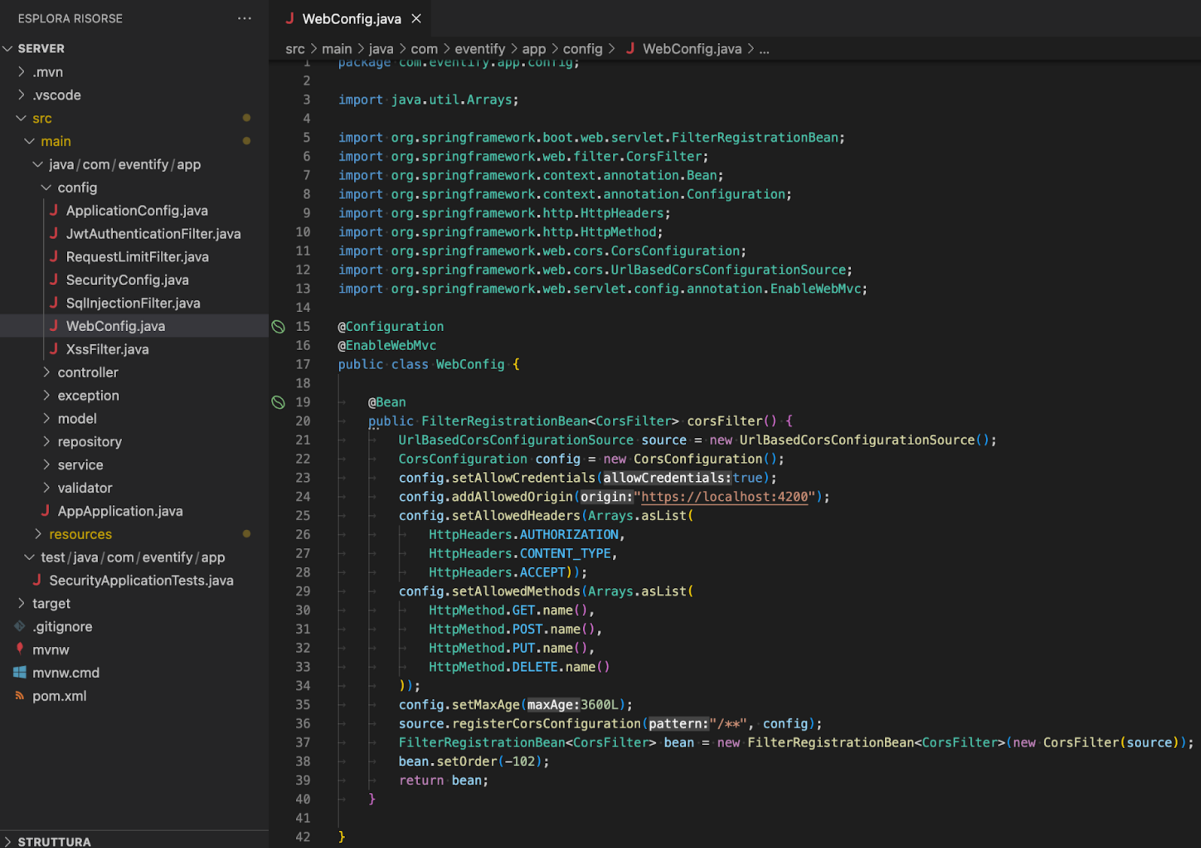


```
1 package com.eventify.app.config;
2
3 import jakarta.servlet.Filter;
4 import jakarta.servlet.FilterChain;
5 import jakarta.servlet.FilterConfig;
6 import jakarta.servlet.ServletException;
7 import jakarta.servlet.ServletRequest;
8 import jakarta.servlet.ServletResponse;
9 import jakarta.servlet.http.HttpServletRequest;
10
11 import java.io.IOException;
12 import java.util.HashMap;
13 import java.util.Map;
14
15 import org.springframework.stereotype.Component;
16
17 @Component
18 public class RequestLimitFilter implements Filter {
19
20     private static final int MAX_REQUESTS_PER_SECOND = 30;
21     private Map<String, Integer> requestCountMap = new HashMap<>();
22     private Map<String, Long> requestTimestampMap = new HashMap<>();
23
24     @Override
25     public void init(FilterConfig filterConfig) throws ServletException {
26     }
27
28     @Override
29     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
30         String clientIP = request.getRemoteAddr();
31
32         if (isRequestLimitExceeded(clientIP, (HttpServletRequest) response)) {
33             return;
34         } else {
35             chain.doFilter(request, response);
36         }
37     }
38
39     @Override
40     public void destroy() {
41     }
42
43     private boolean isRequestLimitExceeded(String clientIP, HttpServletRequest httpServletRequest) {
44         try {
45             long currentTime = System.currentTimeMillis();
46
47             if (requestCountMap.containsKey(clientIP)) {
48                 int count = requestCountMap.get(clientIP);
49                 long lastRequestTime = requestTimestampMap.get(clientIP);
50                 long timeElapsed = currentTime - lastRequestTime;
51
52                 if (count >= MAX_REQUESTS_PER_SECOND && timeElapsed < 1000) {
53                     httpServletResponse.setStatus(503);
54                     httpServletResponse.getWriter().write("Limite di richieste al secondo superato.");
55                     return true;
56                 } else if (timeElapsed >= 1000) {
57                     requestCountMap.put(clientIP, 1);
58                     requestTimestampMap.put(clientIP, currentTime);
59                 } else {
60                     requestCountMap.put(clientIP, count + 1);
61                 }
62             } else {
63                 requestCountMap.put(clientIP, 1);
64                 requestTimestampMap.put(clientIP, currentTime);
65             }
66             return false;
67         } catch (IOException e) {
68             e.printStackTrace();
69             return false;
70         }
71     }
72 }
73
74 }
```

Il filtro **RequestLimitFilter** è progettato per gestire e limitare il numero di richieste effettuate da un client in un determinato intervallo di tempo. Questo filtro è utile per prevenire il sovraccarico del server (attacchi DDOS) dovuto a un elevato numero di richieste da parte di un singolo client. Ecco come funziona:

1. **Inizializzazione del Filtro:** Durante l'inizializzazione del filtro, vengono impostate alcune variabili di configurazione, come il numero massimo di richieste consentite al secondo (**MAX\_REQUESTS\_PER\_SECOND**), e vengono inizializzate due mappe, **requestCountMap** e **requestTimestampMap**, che verranno utilizzate per tenere traccia del conteggio delle richieste e dei timestamp delle richieste precedenti per ciascun client.

2. **Elaborazione delle Richieste:** Quando una richiesta arriva al filtro, viene estratto l'indirizzo IP del client che ha effettuato la richiesta.
  3. **Controllo del Limite delle Richieste:** Il filtro verifica se il client ha superato il limite consentito di richieste al secondo. Questo controllo viene effettuato confrontando il conteggio delle richieste effettuate dal client con il limite massimo consentito.
  4. **Gestione del Limite Superato:** Se il client ha superato il limite consentito, il filtro restituirà una risposta HTTP con il codice di stato "429 Too Many Requests" e un messaggio che indica che è stato superato il limite delle richieste al secondo. In questo caso, il filtro impedirà all'ulteriore elaborazione della richiesta di continuare.
  5. **Aggiornamento del Conteggio delle Richieste:** Se il client non ha superato il limite delle richieste al secondo, il filtro aggiornerà il conteggio delle richieste e il timestamp delle richieste precedenti per il client corrente. Questo permette al filtro di tenere traccia delle richieste e verificare se sono state effettuate nel limite del tempo specificato.
  6. **Continuazione del Flusso di Elaborazione:** Se il client non ha superato il limite delle richieste, il filtro consentirà alla richiesta di proseguire nel flusso di elaborazione standard, consentendo al server di rispondere alla richiesta.
- **CorsFilter:** la quale configurazione si può trovare nel file WebConfig.java nella directory config:



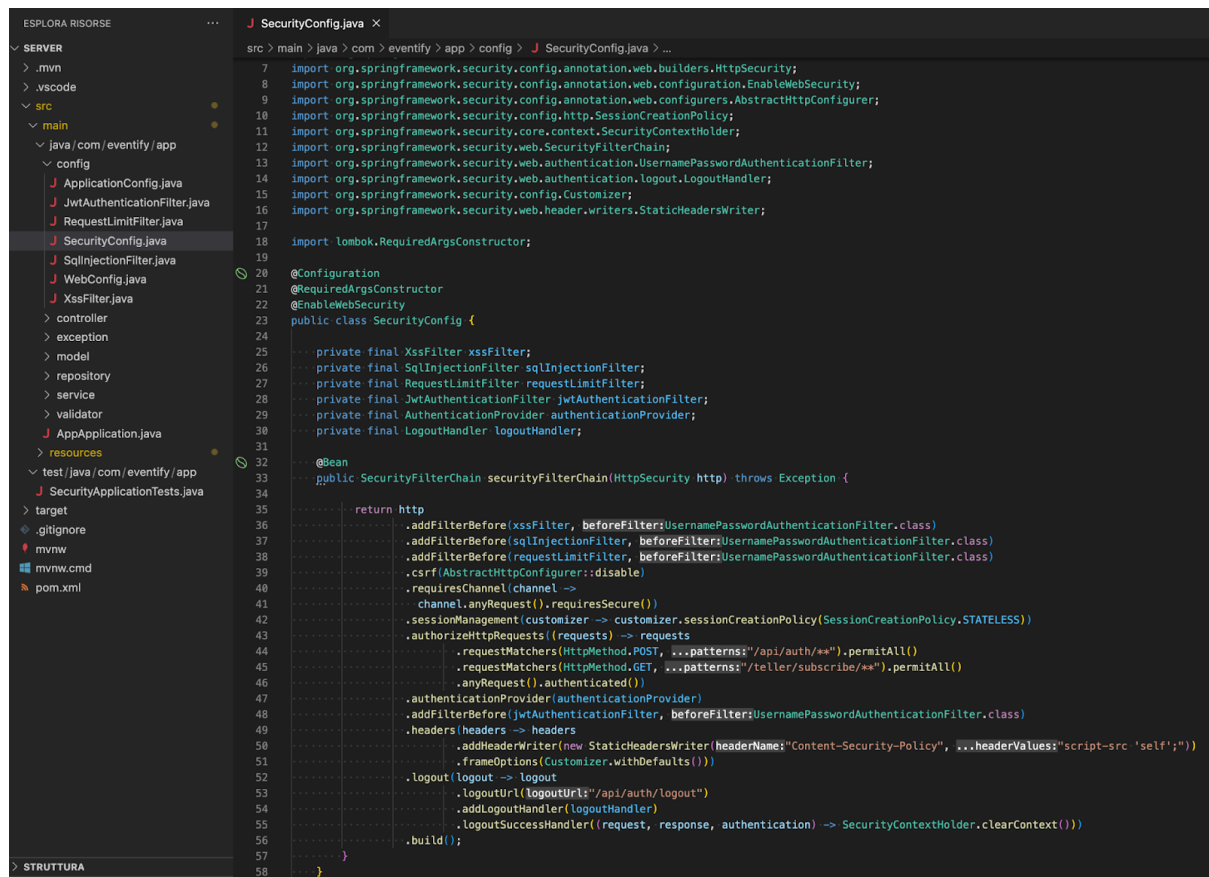
```
1 package com.eventify.app.config;
2
3 import java.util.Arrays;
4
5 import org.springframework.boot.web.servlet.FilterRegistrationBean;
6 import org.springframework.web.filter.CorsFilter;
7 import org.springframework.context.annotation.Bean;
8 import org.springframework.context.annotation.Configuration;
9 import org.springframework.http.HttpHeaders;
10 import org.springframework.http.HttpMethod;
11 import org.springframework.web.cors.CorsConfiguration;
12 import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
13 import org.springframework.web.servlet.config.annotation.EnableWebMvc;
14
15 @Configuration
16 @EnableWebMvc
17 public class WebConfig {
18
19     @Bean
20     public FilterRegistrationBean<CorsFilter> corsFilter() {
21         UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
22         CorsConfiguration config = new CorsConfiguration();
23         config.setAllowCredentials(true);
24         config.addAllowedOrigin(origin("https://localhost:4200"));
25         config.setAllowedHeaders(Arrays.asList(
26             HttpHeaders.AUTHORIZATION,
27             HttpHeaders.CONTENT_TYPE,
28             HttpHeaders.ACCEPT));
29         config.setAllowedMethods(Arrays.asList(
30             HttpMethod.GET.name(),
31             HttpMethod.POST.name(),
32             HttpMethod.PUT.name(),
33             HttpMethod.DELETE.name()));
34     });
35     config.setMaxAge(3600L);
36     source.registerCorsConfiguration(pattern: "**", config);
37     FilterRegistrationBean<CorsFilter> bean = new FilterRegistrationBean<CorsFilter>(new CorsFilter(source));
38     bean.setOrder(-102);
39     return bean;
40 }
41
42 }
```

Il filtro **CorsFilter** gestisce e configura le autorizzazioni CORS (Cross-Origin Resource Sharing) per consentire una comunicazione sicura tra il frontend e il backend. CORS è un



meccanismo di sicurezza che regola le richieste tra domini diversi, consentendo o rifiutando l'accesso alle risorse tra siti web diversi. Ecco come funziona la configurazione definita nel codice fornito:

1. **Creazione dell'istanza del Filtro:** Il codice definisce una configurazione del filtro **CorsFilter** utilizzando Spring Framework. Viene creato un bean **FilterRegistrationBean** che è un'applicazione Spring Boot standard per la registrazione dei filtri.
2. **Configurazione del Filtro CORS:** All'interno del bean del filtro, vengono configurate le autorizzazioni CORS. Questa configurazione specifica quali origini, metodi e intestazioni sono consentiti nelle richieste tra il frontend e il backend.
  - **config.setAllowCredentials(true):** Questa impostazione consente di includere le credenziali nelle richieste, il che è essenziale per l'autenticazione sicura tra il frontend e il backend.
  - **config.addAllowedOrigin("<https://localhost:4200>"):** Questo specifica che le richieste provenienti dall'origine "<https://localhost:4200>" sono consentite. Solo il frontend con questo dominio può effettuare richieste al backend.
  - **config.setAllowedHeaders(...):** Qui vengono specificate le intestazioni consentite, che includono l'autorizzazione, il tipo di contenuto e l'accettazione. Questo assicura che solo le intestazioni specificate siano autorizzate nelle richieste.
  - **config.setAllowedMethods(...):** Vengono specificati i metodi HTTP consentiti, che includono GET, POST, PUT e DELETE. Questo indica quali metodi possono essere utilizzati nelle richieste tra il frontend e il backend.
  - **config.setMaxAge(3600L):** Questo imposta il tempo massimo, in secondi, durante il quale i risultati della pre-elaborazione delle richieste CORS possono essere memorizzati nella cache del browser. In questo caso, il valore è impostato su 3600 secondi (cioè 1 ora).
3. **Registrazione del Filtro CORS:** Il filtro CORS configurato viene quindi registrato utilizzando la **UrlBasedCorsConfigurationSource** con il percorso "/", il che significa che questa configurazione si applicherà a tutte le richieste. Il filtro CORS sarà responsabile di consentire o rifiutare le richieste in base alle autorizzazioni configurate.
4. **Ordine di Esecuzione del Filtro:** Il bean del filtro viene ordinato con un valore di **-102**. L'ordine di esecuzione dei filtri è importante in Spring Security, e il valore negativo assicura che questo filtro venga eseguito prima di altri filtri.
5. **Configurazione di Sicurezza:**
  - La gestione dei filtri di sicurezza è definita all'interno di un bean chiamato **SecurityFilterChain** all'interno del file SecurityConfig definito nella directory config:



```
7 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
8 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
9 import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
10 import org.springframework.security.config.http.SessionCreationPolicy;
11 import org.springframework.security.core.context.SecurityContextHolder;
12 import org.springframework.security.web.SecurityFilterChain;
13 import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
14 import org.springframework.security.web.authentication.logout.LogoutHandler;
15 import org.springframework.security.config.Customizer;
16 import org.springframework.security.web.header.writers.StaticHeadersWriter;
17
18 import lombok.RequiredArgsConstructor;
19
20 @Configuration
21 @RequiredArgsConstructor
22 @EnableWebSecurity
23 public class SecurityConfig {
24
25     private final XssFilter xssFilter;
26     private final SqlInjectionFilter sqlInjectionFilter;
27     private final RequestLimitFilter requestLimitFilter;
28     private final JwtAuthenticationFilter jwtAuthenticationFilter;
29     private final AuthenticationProvider authenticationProvider;
30     private final LogoutHandler logoutHandler;
31
32     @Bean
33     public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
34
35         return http
36             .addFilterBefore(xssFilter, beforeFilter::UsernamePasswordAuthenticationFilter.class)
37             .addFilterBefore(sqlInjectionFilter, beforeFilter::UsernamePasswordAuthenticationFilter.class)
38             .addFilterBefore(requestLimitFilter, beforeFilter::UsernamePasswordAuthenticationFilter.class)
39             .csrf(AbstractHttpConfigurer::disable)
40             .requiresChannel(channel ->
41                 channel.anyRequest().requiresSecure())
42             .sessionManagement(customizer -> customizer.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
43             .authorizeHttpRequests((requests) -> requests
44                 .requestMatchers(HttpMethod.POST, ...patterns: "/api/auth/**").permitAll()
45                 .requestMatchers(HttpMethod.GET, ...patterns: "/teller/subscribe/**").permitAll()
46                 .anyRequest().authenticated())
47             .authenticationProvider(authenticationProvider)
48             .addFilterBefore(jwtAuthenticationFilter, beforeFilter::UsernamePasswordAuthenticationFilter.class)
49             .headers(headers -> headers
50                 .addHeaderWriter(new StaticHeadersWriter(headerName:"Content-Security-Policy", ...headerValues:"script-src 'self';"))
51                 .frameOptions(Customizer.withDefaults()))
52             .logout(logout -> logout
53                 .logoutUrl(logoutURL:"/api/auth/logout")
54                 .addLogoutHandler(logoutHandler)
55                 .logoutSuccessHandler((request, response, authentication) -> SecurityContextHolder.clearContext()))
56             .build();
57     }
58 }
```

Questa configurazione personalizzata è responsabile della gestione della sicurezza. La configurazione include anche:

- Disabilitazione del CSRF (Cross-Site Request Forgery), questo perché la prevenzione da questo attacco non la lasciamo gestire da spring boot automaticamente ma bensì attraverso le proprietà dei nostri JWT Token gli attacchi CSRF sono coperti.
- Richiedere una connessione sicura HTTPS per tutte le richieste.
- Impostare la sessione come "STATELESS" per evitare l'uso di sessioni lato server, anche qui come per il CSRF token, la gestione delle sessioni abbiamo deciso di disabilitarla in quanto vengono gestite direttamente dai JWT Token.
- Definizione delle autorizzazioni per varie richieste, come l'accesso a determinati endpoint senza autenticazione, questo perché le richieste per gli endpoint specificati non hanno bisogno di un autenticazione.
- Gestire l'ordine di esecuzione dei filtri
- Gestione del logout, qui vengono inizializzate le operazioni che il server deve eseguire in caso di logout dell'utente, specificando la classe adibita (LogoutHandler).

## 6. Elaborazione nel Controller:

- Una volta superata la FilterChain, la richiesta viene finalmente instradata al controller specifico, che gestirà l'azione richiesta. Il controller può accedere al database, eseguire logica di business, recuperare dati o effettuare qualsiasi altra operazione necessaria.

## 7. Generazione della Risposta:

- Il controller genera una risposta che può essere inviata al frontend. La risposta può essere formattata in vari modi, come dati JSON, pagine HTML o qualsiasi altro formato richiesto.

**8. Risposta al Cliente:**

- La risposta generata viene quindi restituita al frontend, che la presenta all'utente in modo appropriato. Questo può comportare l'aggiornamento dell'interfaccia utente, la visualizzazione di dati o qualsiasi altra azione prevista dalla richiesta iniziale.

**9. Gestione degli Errori:**

- Durante il processo, se si verificano errori, vengono gestiti secondo le regole definite. Vengono restituiti messaggi di errore appropriati all'utente in caso di problemi.

Questo flusso delle richieste assicura che ogni interazione con il nostro sito web sia gestita in modo sicuro e conforme alle norme di sicurezza, proteggendo i dati sensibili e prevenendo le vulnerabilità comuni. La combinazione di Axios, i filtri di sicurezza personalizzati e la configurazione di sicurezza avanzata di Spring Boot contribuisce a creare un ambiente web sicuro e affidabile per i nostri utenti.

# Accesso e Autenticazione

Gli utenti possono registrarsi e accedere al sito utilizzando un sistema di autenticazione basato su JSON Web Tokens (JWT). La sicurezza dell'accesso è una parte fondamentale della protezione dei dati sensibili e delle identità degli utenti. In questo contesto, utilizziamo i JWT per garantire l'autenticazione e l'autorizzazione in modo sicuro ed efficiente.

## Gestione dei Cookie

La gestione delle sessioni avviene tramite l'utilizzo di JWT Token, che vengono memorizzati in cookie. Questi cookie sono configurati con le seguenti proprietà:

- **HTTP ONLY:** L'attributo **HTTP ONLY** dei cookie rappresenta un elemento fondamentale della nostra strategia di sicurezza, progettato per difendere gli utenti da uno dei rischi più diffusi e pericolosi su Internet: gli attacchi di tipo XSS (Cross-Site Scripting).

L'attributo **HTTP ONLY** risolve questa minaccia bloccando qualsiasi tentativo di accesso ai cookie tramite JavaScript. Questo significa che i cookie contenenti i token di accesso rimangono inaccessibili e non possono essere letti o manipolati da script JavaScript malevoli. Gli attacchi di tipo XSS diventano così inefficaci, poiché i cookie sono inaccessibili ai potenziali aggressori.

La protezione dei token di accesso è di vitale importanza perché i cookie sono una parte cruciale dell'infrastruttura di autenticazione. Consentono agli utenti di autenticarsi in modo sicuro e di mantenere le sessioni attive. L'attributo **HTTP ONLY** garantisce che questi cookie siano inaccessibili ai potenziali aggressori, proteggendo i dati dell'utente e l'integrità del sistema.

- **SAMESITE=Strict:** L'attributo **Samesite=Strict** svolge un ruolo cruciale nella nostra strategia di sicurezza, proteggendo gli utenti da attacchi CSRF (Cross-Site Request Forgery) che potrebbero compromettere la sicurezza delle loro sessioni autenticate.

L'impostazione "Strict" di **Samesite** garantisce che i cookie siano trasmessi solo attraverso richieste provenienti dallo stesso dominio e sottodominio. Ciò significa che un cookie impostato da "<https://localhost:8443>" verrà inviato solo quando il browser effettua richieste a "<https://localhost>" e non a qualsiasi altro dominio o sottodominio. Questo livello di restrizione impedisce a un attaccante di sfruttare un utente autenticato per compiere azioni non autorizzate su un sito diverso, garantendo una maggiore sicurezza.

L'uso di **Samesite=Strict** contribuisce anche a ridurre i potenziali conflitti tra cookie quando il tuo sito web è parte di una complessa struttura di dominio, migliorando ulteriormente la sicurezza dell'autenticazione. Tuttavia, è importante

notare che in alcune situazioni, come l'accesso tramite social media, potrebbe essere necessario un approccio più flessibile. Per consentire agli utenti di effettuare il login o di interagire con il tuo sito utilizzando i loro account sui social media in modo agevole, potresti considerare l'implementazione futura di **Samesite=Lax**. Questa impostazione consente alcune eccezioni controllate, garantendo al contempo la sicurezza.

- **SECURE=TRUE:** L'impostazione **Secure=true** è un altro aspetto cruciale della nostra strategia di sicurezza, progettato per garantire la massima sicurezza durante il trasferimento dei dati sensibili attraverso i cookie.

In un mondo in cui la condivisione e il trasferimento di dati via Internet sono all'ordine del giorno, è fondamentale garantire che le informazioni personali e sensibili degli utenti siano protette durante la trasmissione tra il loro browser e il server. L'uso di una connessione sicura HTTPS è un requisito essenziale in questo contesto, e l'attributo "SECURE=TRUE" dei cookie svolge un ruolo critico per garantirlo.

Ecco perché questa impostazione è così rilevante:

1. **Sicurezza nei Trasferimenti di Dati Sensibili:** I cookie spesso contengono informazioni sensibili, come token di autenticazione o altri dati critici per l'identità dell'utente. L'attributo "SECURE=TRUE" garantisce che questi dati siano trasmessi solo su connessioni sicure HTTPS, rendendo estremamente difficile a un potenziale aggressore intercettare o manipolare queste informazioni durante la trasmissione. La cifratura dei dati sui protocolli HTTPS assicura che solo il server di destinazione possa decodificare e comprendere i dati trasmessi, offrendo una barriera invalicabile per gli attacchi di tipo "man-in-the-middle."
2. **Protezione dalla Manomissione dei Cookie:** La trasmissione dei cookie attraverso una connessione HTTPS non solo protegge i dati sensibili, ma garantisce anche che i cookie stessi rimangano invariati durante il trasferimento. Ciò significa che non possono essere alterati o sostituiti da terzi malevoli, che potrebbero cercare di manipolare i cookie per accedere a un'identità o una sessione utente.
3. **Conformità alle Migliori Pratiche di Sicurezza:** L'uso di HTTPS e l'impostazione "SECURE=TRUE" per i cookie è una pratica standard e consigliata nell'ambito della sicurezza web. È una componente essenziale della conformità alle normative sulla privacy, come il Regolamento Generale sulla Protezione dei Dati (GDPR), e aiuta a dimostrare l'impegno per la protezione dei dati sensibili degli utenti.

Il nostro sistema di autenticazione attraverso i cookie comprende quindi l'utilizzo di JWT token i quali hanno un sistema di creazione, una struttura interna e un sistema di decriptazione che di loro natura fornisce un alto grado di sicurezza, infatti i JWT Token hanno una struttura che comprende tre corpi essenziali:

1. **Header che contiene le due informazioni base:** la tipologia del token, che nel nostro caso è JWT, e quella dell'algoritmo utilizzato per la cifratura.
2. **Payload**, il blocco che contiene le informazioni di scambio tra le parti. Questo a sua volta si divide in tre fasi: **parametri registrati**, **parametri pubblici** e **parametri privati**. Con i **parametri registrati** andiamo a indicare quelle proprietà che sono predefinite e che servono a mostrare le peculiarità del token. Sono, per dirla tutta, stringhe, dati e array che servono per il corretto funzionamento del sistema. Con i **parametri pubblici** invece indichiamo altri dati che sono predefiniti nel IANA JSON Web Token Registry. Questi parametri possono essere inseriti a scelta dello user ma bisogna stare attenti in questa fase perché è possibile che si possano registrare delle problematiche se l'inserimento non è stato fatto correttamente. Con i **parametri privati** ci si può sbizzarrire: è un corpo che viene lasciato alla libera determinazione del compilatore. Questa flessibilità è propria della struttura JSON e ne ha caratterizzato il successo.
3. **Signature, la fase di cifratura delle due parti:** Payload e Header. Queste vengono dapprima unite e poi sottoposte a un'operazione di crittografia estremamente complessa. Alla fine dell'operazione viene generata una chiave che darà luogo a un token di oltre 200 caratteri.

Quindi per garantire un alto livello di sicurezza nella creazione e nella criptazione dei token, il nostro sistema adotta un rigoroso approccio di sicurezza. La scelta dell'algoritmo di criptazione e la generazione della chiave sono componenti chiave di questa strategia.

**Algoritmo di Criptazione HS256:** I nostri token vengono criptati utilizzando l'algoritmo HS256 (HMAC con SHA-256). Questo è uno degli algoritmi più affidabili e comunemente utilizzati per la criptazione di dati sensibili. La scelta di HS256 offre un elevato grado di sicurezza e garantisce che i token siano resistenti a molteplici tipi di attacchi.

**Generazione della Chiave a 256 bit Hex:** La chiave di criptazione, utilizzata sia per la firma (signature) che per la decodifica dei token, è generata con estrema attenzione alla sicurezza. È una chiave a 256 bit rappresentata in formato esadecimale (Hex). Questo rende la chiave virtualmente impossibile da indovinare o decifrare attraverso attacchi di forza bruta.

La chiave di criptazione è un segreto noto solo al server, garantendo che solo il server stesso sia in grado di creare e decodificare i token. Questo livello di sicurezza aggiuntivo è fondamentale per evitare che terze parti non autorizzate accedano ai dati contenuti nei token.

**Payload con Dati Non Sensibili:** Nel payload dei token, abbiamo fatto una scelta consapevole di non includere dati sensibili. Invece, il payload contiene informazioni che consentono di identificare un utente, come ad esempio il suo nome utente (username). Questo è un approccio che contribuisce a garantire la sicurezza, poiché i dati sensibili sono mantenuti al sicuro all'interno del server, riducendo al minimo il rischio di esposizione.

Nel cuore del nostro sistema di autenticazione attraverso i cookie si trova l'uso di due tipi di JSON Web Tokens (JWT): l'Access Token e il Refresh Token. Questi due token sono generati al momento dell'accesso dell'utente, vengono trasferiti ad ogni richiesta e svolgono

ruoli distinti ma complementari, contribuendo a migliorare l'esperienza dell'utente e rafforzare la sicurezza.

## **Access Token**

L'access token è il token che permette di riconoscere se l'utente è autenticato o meno in quanto alla sua creazione viene inserita nel payload l'email dell'utente che è un dato univoco e di conseguenza quando il token verrà successivamente ricevuto dal server quest'ultimo controllerà se l'email presente nel payload del token appena ricevuto è presente nel database confermando che l'utente è già autenticato, inoltre per aumentare l'efficienza dell'autenticazione quando un token viene riconosciuto per la prima volta viene successivamente associato al security context holder il quale farà sì che quando si ripresenterà lo stesso token non ci sarà bisogno nuovamente di estrarre il token e riconoscerlo ma bensì basterà verificare se il token è già presente le security context holder. L'access token ha una data di scadenza molto breve (5 min) e quindi il server dovrà anche verificare se il token è effettivamente scaduto e nel caso rifiutare l'accesso, se un access token è scaduto il client, al quale verrà fornito il tempo di scadenza dell'access token si occuperà di richiedere un nuovo access token attraverso il processo di refresh token.

## **Refresh Token**

Il refresh token è un token che permette di rigenerare un access token dopo la sua scadenza, il suo impiego quindi è molto meno frequente e questa combinazione permette oltre ad un'aumento dell'efficienza del sito. Il refresh token viene generato insieme all'access token in seguito all'accesso dell'utente al sito, la creazione del refresh token è identica a quella dell'access token con la differenza che il refresh token in seguito alla sua creazione viene immagazzinato nel database in particolare in un campo associato all'utente, questo è per evitare casi in cui un malintenzionato possa inviare un token malizioso da lui creato magari dopo aver scoperto la chiave segreta. Il processo di refresh token si ha quando dopo la scadenza di un access token se ne richieda un altro e a quel punto il server verifica che il refresh token ricevuto sia opportunamente registrato nella tabella dell'user con email registrata nel payload dello stesso e a quel punto restituisce il nuovo access token. Il refresh token nasce per aver una scadenza a lungo termine (1 mese) questo migliora l'usabilità del sistema, consentendo agli utenti di rimanere connessi per periodi prolungati senza dover effettuare frequenti accessi. Inoltre il refresh token nasce anche per fornire una maggiore efficienza al server il quale non avrà bisogno di confrontarsi con il database ad ogni ricevimento di access token. Per implementazioni future l'access token potrà essere utilizzato per disabilitare la possibilità di generare altri access token ad un utente (bandirlo dal sito).

## **Revocazione dell'Access Token**

È importante considerare la possibilità di revocare un Access Token in situazioni in cui un utente perde il controllo del suo token o è necessario revocare l'accesso. Questo può essere implementato utilizzando una lista nera dei token o un meccanismo di revoca centralizzato. La revoca dell'Access Token è un'importante misura di sicurezza per proteggere l'accesso non autorizzato.

## Logout

Durante la procedura di logout, entrambi gli Access Token e i Refresh Token vengono rimossi dai cookie. Il logout garantisce che l'utente non possa più accedere al sistema dopo aver effettuato l'uscita e contribuisce alla sicurezza complessiva dell'applicazione.

In conclusione, il nostro sistema di autenticazione basato su JWT e cookie offre un alto grado di sicurezza e protezione delle identità degli utenti. Tuttavia, è essenziale attenersi a rigorose best practice di sicurezza e monitorare costantemente il sistema per rilevare e affrontare eventuali minacce potenziali.



# Gestione delle Autorizzazioni

La gestione delle autorizzazioni nel nostro sistema è progettata per garantire un controllo preciso e sicuro dell'accesso alle risorse e alle funzionalità dell'applicazione. Ogni utente ha assegnati ruoli specifici che definiscono i permessi e i livelli di accesso all'interno del sistema.

## Ruoli Utente

Nel nostro sistema, sono definiti i seguenti ruoli utente:

1. **Utente Registrato:** Questo è il ruolo base assegnato a tutti gli utenti registrati. Gli utenti registrati hanno il permesso di iscriversi a eventi e creare nuovi eventi all'interno dell'applicazione.
2. **Organizzatore:** Gli utenti con il ruolo di Organizzatore hanno privilegi aggiuntivi. Possono creare eventi e gestire i dettagli degli eventi esistenti. Tuttavia, possono modificare o eliminare solo gli eventi che hanno creato.
3. **Utente Autenticato:** Gli utenti autenticati possono modificare le informazioni del proprio profilo. Possono aggiornare i dettagli del proprio account, tra cui nome, cognome, indirizzo email e password.

## Dettagli dei Ruoli

- **Utente Registrato:** Gli utenti registrati hanno il permesso di iscriversi a qualsiasi evento presente nell'applicazione. Possono anche creare nuovi eventi e partecipare a quelli creati da altri utenti. Tuttavia, non possono modificare o eliminare eventi creati da altri utenti.
- **Organizzatore:** Gli organizzatori hanno il potere di creare eventi e aggiornarne i dettagli. Possono anche eliminare eventi, ma solo quelli che hanno creato. Questo ruolo è pensato per coloro che gestiscono eventi e necessitano di un maggiore controllo sui dettagli degli eventi.
- **Utente Autenticato:** Gli utenti autenticati hanno il controllo completo sui dati del proprio profilo. Possono modificare informazioni personali come il proprio nome, cognome, indirizzo email e password. Tuttavia, non hanno il permesso di accedere a funzionalità di amministrazione o modificare eventi creati da altri utenti.

Le autorizzazioni nel nostro sistema sono strettamente legate alla struttura del sito web. La progettazione dell'interfaccia utente consente di gestire con precisione chi ha accesso a quali risorse e funzionalità. In particolare, le autorizzazioni vengono applicate in base alla sezione del sito in cui l'utente si trova.

## Autorizzazioni Basate sulla Struttura del Sito

### Modifica degli Eventi

Nel nostro sistema, gli utenti possono modificare eventi solo all'interno della sezione denominata "MyEvents". Questa sezione è dedicata esclusivamente agli eventi creati dall'utente. All'interno di "MyEvents", gli utenti vedranno i propri eventi e avranno accesso a opzioni di modifica per ciascun evento che hanno creato. Questo garantisce un controllo completo sulle operazioni di modifica, inclusa la possibilità di aggiornare dettagli, data, orario e altre informazioni relative agli eventi di loro competenza.

## **Modifica del Profilo**

L'accesso alla modifica del proprio profilo è limitato all'utente loggato. Ogni utente ha il controllo completo dei propri dati personali e delle informazioni del profilo. Possono aggiornare il proprio nome, cognome, indirizzo email e foto profilo attraverso una sezione specifica dedicata al profilo personale. Questa struttura garantisce che ciascun utente abbia il controllo esclusivo sui dati personali e sulla privacy del proprio account.

## **Sicurezza e Controllo**

La struttura delle autorizzazioni basata sulla struttura del sito garantisce un elevato livello di sicurezza e controllo all'interno del nostro sistema. Gli utenti possono operare in modo sicuro e intuitivo all'interno delle sezioni del sito web, sapendo esattamente a quali risorse e funzionalità hanno accesso in base ai loro diritti e alle loro responsabilità. Ciò contribuisce a prevenire accessi non autorizzati e garantisce che le operazioni di modifica degli eventi e del profilo siano eseguite in modo corretto e sicuro, mantenendo la privacy e l'integrità dei dati degli utenti.

# Protezione dalle Minacce

La sicurezza è una priorità fondamentale nell'architettura e nello sviluppo del nostro sito web. Al fine di proteggere l'applicazione e i dati degli utenti da attacchi comuni, sono state implementate diverse misure di sicurezza.

## Prevenzione del SQL Injection

Per prevenire gli attacchi SQL Injection, abbiamo adottato un approccio a più livelli. In primo luogo, è stato implementato un filtro dedicato chiamato "SqlInjectionFilter" che analizza le richieste in arrivo e rileva potenziali tentativi di SQL Injection. Se viene rilevato un attacco, il filtro risponde con uno stato di "FORBIDDEN" e impedisce l'accesso non autorizzato. Questo filtro è in grado di identificare stringhe di SQL malevole nei dati inviati.

In secondo luogo, abbiamo reso rigorose le proprietà dei campi del database. Questo significa che i dati che vengono inseriti nei campi devono rispettare specifiche proprietà e vincoli definiti. Qualsiasi tentativo di inserire dati non validi o fuori specifica verrà respinto dal database.

## Prevenzione di Cross-Site Scripting (XSS)

Per proteggerci da attacchi di Cross-Site Scripting (XSS), il sistema adotta una duplice strategia. In primo luogo, i dati provenienti dai form del sito passano attraverso una verifica sia lato frontend che lato backend. Il frontend verifica che i dati inseriti siano conformi alle proprietà definite e che non contengano codice JavaScript dannoso. Se la validazione fallisce, il frontend rifiuta la richiesta prima che i dati raggiungano il server.

Nel processo di validazione lato server (backend), i dati vengono nuovamente esaminati per garantire che siano sicuri e conformi. Se si rileva qualsiasi inconformità o presenza di codice malevolo, il sistema risponderà con un messaggio di errore e non immagazzinerà i dati non validi.

## Prevenzione di Cross-Site Request Forgery (CSRF)

Per prevenire gli attacchi CSRF, abbiamo adottato un'approccio basato su token. Abbiamo disabilitato il controllo CSRF automatico fornito da Spring Boot e utilizziamo invece token JWT (JSON Web Tokens) per l'autenticazione. Questi token sono memorizzati in cookie con impostazioni "SameSite=Strict", il che significa che possono essere trasmessi solo attraverso richieste dallo stesso dominio. In altre parole, i cookie non possono essere veicolati da altri domini.

Inoltre, la configurazione CORS implementata nel sistema consente solo richieste provenienti dal dominio del frontend, garantendo che solo il frontend possa interagire con il backend. Questa configurazione riduce notevolmente il rischio di CSRF, poiché gli attacchi devono provenire dalla stessa origine.

## Validazione dei Dati

I dati vengono attentamente validati sia lato frontend che lato backend. Ogni form e ogni richiesta passano attraverso processi di validazione per garantire che i dati siano corretti e conformi alle specifiche dell'applicazione. Se un dato non soddisfa i criteri di validazione, verrà rifiutato e non immagazzinato nel database. Questa pratica riduce il rischio di errori e di dati dannosi.

## Memorizzazione Sicura delle Password

Le password degli utenti vengono memorizzate in maniera sicura attraverso l'uso di algoritmi di hashing avanzati. Questo significa che le password sono convertite in una sequenza di caratteri illeggibili prima di essere memorizzate nel database. Questa pratica garantisce che le password degli utenti siano protette da accessi non autorizzati.

## Minimizzazione dei Rischi OWASP Top Ten

Nel corso dello sviluppo, ci siamo impegnati a minimizzare i rischi elencati nel documento OWASP Top Ten Standard Awareness. Le misure di sicurezza adottate, come discusse sopra, contribuiscono a mitigare molte delle minacce comuni menzionate nel documento OWASP, garantendo un ambiente online più sicuro per i nostri utenti.

## Prevenzione delle Minacce OWASP Top Ten

Nel corso dello sviluppo del nostro sistema, abbiamo adottato misure di sicurezza rigorose per mitigare le principali minacce elencate nel documento OWASP Top Ten Standard Awareness. Questo impegno è finalizzato a garantire un ambiente online sicuro e protetto per i nostri utenti.

**Filtri di Sicurezza:** Abbiamo implementato una solida "Security Filter Chain" che include diversi filtri chiave:

- **CORS Filter:** Per prevenire i rischi di Cross-Origin Resource Sharing (CORS) e proteggere la nostra applicazione da potenziali attacchi.
- **Filtro di Limitazione delle Richieste (LimitRequestFilter):** Questo filtro è progettato per mitigare gli attacchi Distributed Denial of Service (DDoS), garantendo che il nostro sistema rimanga sempre disponibile e reattivo.
- **Filtro contro l'Iniezione SQL (SQL Injection Filter):** Per prevenire le vulnerabilità di iniezione SQL, il filtro convalida e sanifica rigorosamente i dati inseriti.
- **Filtro contro Cross-Site Scripting (XSS):** Abbiamo integrato un filtro per proteggere gli utenti da attacchi basati su script dannosi, impedendo l'inserimento di script malevoli nelle pagine web.
- **Filtro di Autenticazione JWT (JWT Authentication Filter):** Questo filtro è progettato per garantire l'autenticazione sicura e l'accesso autorizzato all'applicazione.

**Protocollo HTTPS:** Abbiamo impostato l'uso del protocollo HTTPS sia dal lato client che dal lato server. Questo garantisce la crittografia dei dati trasmessi tra il client e il server, proteggendo le informazioni sensibili dagli occhi indiscreti.

**Gestione Sicura dei Cookie:** Per garantire la sicurezza dei cookie, abbiamo adottato rigorose politiche:

- I cookie utilizzati per l'accesso sono contrassegnati come "**HTTP-only**," impedendo l'accesso da parte di script lato client.
- Abbiamo impostato **SameSite=Strict** e **Secure=True** nei cookie, garantendo che i cookie vengano trasmessi solo su connessioni sicure, riducendo il rischio di attacchi basati su cookie.

**Notifiche di Autenticazione Fallita:** Abbiamo implementato un sistema di notifiche per informare gli utenti in caso di tentativi falliti di autenticazione. Questo li tiene informati delle minacce potenziali alla sicurezza del proprio account e li incoraggia a mantenere le proprie credenziali al sicuro.

**Autenticazione a Due Fattori (2FA):** Stiamo promuovendo l'uso dell'autenticazione a due fattori (2FA) per offrire un ulteriore strato di sicurezza ai nostri utenti. Il 2FA richiede un secondo metodo di verifica oltre alle normali credenziali, rendendo più difficile l'accesso non autorizzato.

Questo testo evidenzia come abbiamo adottato precauzioni specifiche per proteggere il nostro sistema dalle principali minacce OWASP Top Ten, garantendo un ambiente online sicuro e affidabile per i nostri utenti.

# Flusso di Registrazione - sign-up

## Introduzione

Il flusso di registrazione è un processo cruciale all'interno dell'applicazione, consentendo agli utenti di creare un account personale per accedere ai servizi offerti.

Questa documentazione approfondita copre gli aspetti tecnici, tra cui l'uso del framework **Spring**, la validazione dei dati, il controllo delle immagini, le politiche di sicurezza delle password e la crittografia delle password prima dell'archiviazione nel database.

## Uso di Spring Framework

Nel flusso di registrazione, facciamo ampio uso del framework Spring, sfruttando le annotazioni **@Transactional** e il processo di validazione dati.

L'annotazione **@Transactional** è fondamentale per garantire la corretta gestione delle transazioni durante la registrazione.

Tutte le operazioni all'interno del metodo **signUp** vengono eseguite all'interno di una singola transazione. In caso di errori, la transazione viene automaticamente annullata per mantenere la coerenza dei dati nel database.

## Validazione dei Dati

L'oggetto di richiesta di registrazione viene sottoposto a un rigoroso processo di validazione attraverso il metodo `validate`.

Questo passaggio è cruciale per assicurare che i dati forniti rispettino le regole e i criteri di validità stabiliti prima di procedere con la registrazione.

### Controllo dei Magic Numbers per le Immagini

Il flusso di registrazione include un controllo dei "magic numbers" per verificare la validità dei file immagine caricati. Questo processo comporta la verifica dei primi byte del file rispetto ai valori noti che identificano i formati di immagine supportati. I "magic numbers" utilizzati includono:

**JPEG** (Joint Photographic Experts Group): [0xFF, 0xD8, 0xFF, 0xE0]

**PNG** (Portable Network Graphics): [0x89, 0x50, 0x4E, 0x47]

Questo controllo garantisce che solo immagini con formati validi siano ammesse durante il processo di registrazione.

## Politiche di Sicurezza delle Password

Le politiche di sicurezza delle password sono una componente vitale del flusso di registrazione. Sono progettate per garantire che le password degli utenti siano robuste e resistenti agli attacchi. Le seguenti politiche sono implementate:

### **Lunghezza Minima della Password**

La password deve essere costituita da almeno 8 caratteri per garantire un adeguato livello di sicurezza.

### **Complessità della Password**

Le password devono soddisfare requisiti rigidi, inclusi almeno un carattere maiuscolo, almeno un carattere speciale tra "@", "#", "\$", "%", "^", "&", "+", "!", e almeno un numero. Questo contribuisce a rendere le password resistenti agli attacchi di forza bruta.

### **Conferma della Password**

L'utente è tenuto a confermare la password inserita, contribuendo a evitare errori di battitura e garantendo l'accuratezza dei dati di registrazione.

### **Crittografia e Archiviazione delle Password**

Prima dell'archiviazione nel database, le password degli utenti vengono sottoposte a crittografia. Questo processo è fondamentale per garantire che le password rimangano al sicuro anche in caso di accesso non autorizzato al database. Un algoritmo di crittografia, come bcrypt, è comunemente utilizzato per questo scopo.

### **Conclusione**

Il flusso di registrazione è una fase critica dell'applicazione, che richiede un'accurata gestione delle transazioni, la validazione dei dati, il controllo delle immagini, le politiche di sicurezza delle password e la crittografia delle stesse prima dell'archiviazione nel database. Questa documentazione offre una panoramica dettagliata di come ciascun aspetto sia implementato nell'applicazione.

# Flusso di accesso - Sign-In

Nel processo di registrazione, la fase di sign-in è fondamentale per consentire agli utenti registrati di accedere in modo sicuro all'applicazione. Questa sezione documenta il flusso di autenticazione dell'utente, evidenziando i componenti chiave del sistema.

## Configurazione dell'Applicazione

### **ApplicationConfig**

La classe **ApplicationConfig** gestisce la configurazione dell'applicazione, includendo il setup del servizio utente e la gestione dell'autenticazione.

**UserDetailsService:** Definisce il servizio per ottenere i dettagli dell'utente in base al nome utente, che viene utilizzato durante l'autenticazione.

**AuthenticationProvider:** Configura il provider di autenticazione, che utilizza il servizio utente e l'encoder delle password per verificare le credenziali degli utenti.

**PasswordEncoder:** Specifica l'encoder delle password (nel caso specifico, BCrypt) per crittografare e proteggere le password degli utenti.

## Gestione delle Richieste di Autenticazione

### **JwtAuthenticationFilter**

La classe `JwtAuthenticationFilter` è responsabile di filtrare le richieste di autenticazione. Questo filtro esegue diverse azioni chiave:

**Validazione del Token:** Verifica se un token di accesso è presente nella richiesta. Se presente, il filtro estrae il nome utente dal token.

**Caricamento dei Dettagli dell'Utente:** Utilizza il servizio `UserDetailsService` per ottenere i dettagli dell'utente basati sul nome utente estratto dal token.

**Verifica della Validità del Token:** Utilizza il servizio JWT per verificare se il token è valido per l'utente.

**Impostazione dell'Autenticazione:** Se il token è valido, il filtro imposta l'oggetto `Authentication` nel contesto di sicurezza dell'applicazione, consentendo all'utente di accedere in modo autenticato.



## Configurazione della Sicurezza

### SecurityConfig

La classe SecurityConfig gestisce la configurazione della sicurezza globale dell'applicazione. Questa configurazione include:

**Filtraggio delle Richieste:** Si applica una serie di filtri, tra cui il filtro XSS, il filtro SQL Injection e il filtro di limitazione delle richieste. Questi filtri contribuiscono a proteggere l'applicazione da attacchi comuni.

**Disabilitazione del CSRF:** Il Cross-Site Request Forgery (CSRF) è disabilitato per evitare richieste non autorizzate.

**Richiede Canale Sicuro:** Si richiede un canale sicuro per tutte le richieste, garantendo la sicurezza delle comunicazioni.

**Gestione delle Sessioni:** La gestione delle sessioni è impostata su STATELESS, il che significa che l'applicazione non mantiene sessioni utente, migliorando la sicurezza.

**Autorizzazione delle Richieste:** Si specifica quali richieste sono consentite. Ad esempio, le richieste di autenticazione e alcune richieste pubbliche sono permessi, mentre tutte le altre richieste richiedono l'autenticazione.

**Gestione dell'Autenticazione:** Si configura il provider di autenticazione e si aggiunge il filtro di autenticazione JWT per consentire l'autenticazione basata su token.

**Header e Frame Options:** Vengono impostati gli header di sicurezza, tra cui il "Content-Security-Policy" e le opzioni per la protezione dei frame.

**Gestione del Logout:** Si configura l'URL di logout e le azioni da intraprendere quando un utente esegue il logout.

### Sign-In dell'Utente

Il metodo signIn consente all'utente di effettuare l'accesso. Questo metodo esegue le seguenti azioni:

**Autenticazione:** Verifica le credenziali dell'utente utilizzando l'AuthenticationManager e imposta il contesto di sicurezza.

**Invio dell'OTP:** In caso di autenticazione riuscita, genera un One-Time Password (OTP) e lo invia all'utente per ulteriore verifica.

**Gestione degli Errori:** Gestisce gli errori possibili, ad esempio credenziali non valide o utente non registrato, restituendo un messaggio di errore appropriato.

**Restituzione dei Risultati:** Restituisce una risposta che include l'access token, il refresh token, l'eventuale messaggio di errore e la data di scadenza dell'accesso.

## **Flusso di accesso - Sign-In con 2FA (Two-Factor Authentication)**

Nel contesto del processo di registrazione, il sign-in con 2FA è un aspetto critico per garantire un accesso sicuro all'applicazione. Questa sezione documenta il flusso di autenticazione dell'utente con la 2FA, evidenziando i passaggi chiave.

### **Autenticazione Tradizionale**

Il processo di sign-in inizia con un'autenticazione tradizionale in cui l'utente inserisce le sue credenziali, ovvero l'indirizzo email e la password. Queste credenziali vengono verificate per accertare l'identità dell'utente.

**Autenticazione dell'Utente:** L'utente fornisce l'indirizzo email e la password.

**Verifica delle Credenziali:** Le credenziali vengono verificate rispetto ai dati dell'utente nel database. Se le credenziali sono valide, l'utente viene autenticato e il contesto di sicurezza viene impostato.

### **Generazione dell'OTP (One-Time Password)**

Dopo una corretta autenticazione, l'applicazione richiede all'utente di inserire un One-Time Password (OTP) generato e inviato all'utente per il processo di 2FA.

**Generazione dell'OTP:** Viene generato un OTP e inviato all'utente tramite un canale sicuro (ad esempio, via email).

Verifica dell'OTP

L'utente inserisce l'OTP ricevuto, e l'applicazione verifica la sua correttezza.

**Inserimento dell'OTP:** L'utente inserisce l'OTP ricevuto.

**Verifica dell'OTP:** L'applicazione verifica se l'OTP inserito corrisponde a quello generato e inviato all'utente. Se l'OTP è valido, l'utente può accedere; altrimenti, riceverà un messaggio di errore.

# Flusso di Esecuzione - Gestione degli Eventi

## 1. Validazione dei Dati di Creazione dell'Evento

La validazione dei dati di creazione di un evento è un passaggio cruciale per garantire che solo informazioni corrette e sicure vengano utilizzate nell'applicazione. Qui di seguito vengono illustrati i processi di validazione che si verificano prima della creazione di un evento:

### Validazione dei Campi del Form

Quando un utente cerca di creare un nuovo evento, l'applicazione verifica che tutti i campi obbligatori siano stati compilati. Ciò include il titolo, la descrizione, la data e l'ora, il luogo, la categoria e le foto dell'evento. Se mancano informazioni o i campi sono vuoti, l'applicazione restituisce un errore e impedisce la creazione dell'evento fino a quando non vengono forniti tutti i dati richiesti.

### Controllo sulla Data e l'Ora

La data e l'ora dell'evento sono soggette a un controllo per garantire che l'evento sia programmato per un momento futuro. L'applicazione confronta la data e l'ora fornite dall'utente con l'orario attuale e verifica che ci siano almeno 24 ore di anticipo rispetto alla data corrente. Se l'evento è previsto per una data passata o per meno di 24 ore dal momento attuale, viene rifiutato.

### Controllo sulle Immagini

Le immagini caricate per l'evento vengono verificate per assicurarsi che siano immagini valide. L'applicazione utilizza la tecnica dei "magic numbers" per confermare che il file è effettivamente un'immagine. Se le immagini non soddisfano i criteri, l'applicazione rifiuta il caricamento delle immagini.

### Sicurezza dei Dati Utente

In ogni fase di validazione, l'applicazione assicura che i dati utente, come l'identificazione dell'utente che crea l'evento, siano protetti. L'applicazione deve prevenire possibili attacchi, come l'inserimento di script dannosi o tentativi di manipolare i dati dell'evento.

### Comunicazione degli Errori

In caso di fallimento della validazione, l'applicazione deve comunicare chiaramente gli errori all'utente. Questo include messaggi di errore specifici che indicano quale campo deve essere corretto e perché. Una volta superata con successo la fase di validazione, l'evento può essere creato in modo sicuro e i dati associati vengono salvati nel sistema.

## 2. Creazione di Eventi

Il processo di creazione di eventi consente agli utenti di definire nuovi eventi e fornire dettagli come titolo, descrizione, data e ora, luogo, categoria e immagini associate.

Flusso:

1. L'utente invia una richiesta per creare un nuovo evento al `EventController`.
2. Il `EventController` utilizza l'`EventValidator` per convalidare i dati forniti dall'utente.
3. Se i dati sono validi, un nuovo oggetto `Event` viene creato con i dettagli forniti.
4. L'evento viene quindi salvato nel database attraverso il `IEventRepository`.
5. L'utente riceve una risposta conferma con l'ID dell'evento appena creato.

## 3. Modifica di Eventi

Gli utenti possono modificare solamente gli eventi esistenti creati da loro, facilmente individuabili nella sezione **MyEvents**, inclusi titolo, descrizione, data e ora, luogo, categoria e immagini.

Flusso:

1. Un utente autorizzato invia una richiesta di modifica di un evento specifico al `EventController`.
2. Il `EventController` verifica se l'utente ha il diritto di modifica (deve essere il creatore dell'evento).
3. Se l'utente è autorizzato, il controller esegue l'aggiornamento dell'evento, utilizzando i nuovi dati forniti dall'utente.
4. Il `EventController` esegue la validazione dei dati attraverso `EventValidator`.
5. Se i dati sono validi, l'evento viene aggiornato e le modifiche vengono salvate nel database.

## 4. Partecipazione e Annullamento della Partecipazione

Gli utenti possono registrarsi per partecipare a eventi o annullare la partecipazione.

Flusso - Partecipazione:

1. Un utente interessato invia una richiesta di partecipazione a un evento specifico al `EventController`.
2. Il controller verifica se l'utente esiste e se l'evento è valido.
3. L'utente viene aggiunto alla lista dei partecipanti dell'evento e riceve una conferma.
4. Le notifiche vengono inviate all'utente e al creatore dell'evento (tramite `NotificationService` ed `EmailService`).

Flusso - Annullamento della Partecipazione:

1. Un utente precedentemente registrato per partecipare a un evento può annullare la partecipazione inviando una richiesta al `EventController`.
2. Il controller verifica se l'utente è registrato per l'evento e procede a rimuoverlo dalla lista dei partecipanti.
3. Le notifiche vengono inviate all'utente e al creatore dell'evento.

## 5. Eliminazione di Eventi

Gli utenti possono eliminare eventi che hanno creato, con notifiche agli utenti partecipanti.

Flusso:

1. Un utente autenticato, creatore di un evento, invia una richiesta di eliminazione al `EventController`.
2. Il controller verifica se l'evento esiste e se l'utente è il creatore dell'evento.
3. Se tutti i criteri sono soddisfatti, l'evento viene eliminato dal database e le notifiche vengono inviate a tutti i partecipanti e al creatore.

## 6. Filtri e Ricerca di Eventi

L'applicazione offre funzionalità di ricerca e filtraggio degli eventi per consentire agli utenti di trovare gli eventi che soddisfano i loro interessi. Questo è un aspetto importante per migliorare l'esperienza dell'utente e aiutare gli utenti a scoprire gli eventi che sono rilevanti per loro.

### Filtraggio per Titolo

Gli utenti possono cercare eventi specifici inserendo il titolo dell'evento. L'applicazione esegue una query per trovare eventi con titoli che corrispondono alla stringa di ricerca fornita dagli utenti.

### Filtraggio per Luogo

Gli utenti possono filtrare gli eventi in base alla posizione geografica. L'applicazione utilizza servizi di geocoding per mappare gli indirizzi dei luoghi degli eventi alle coordinate geografiche. Gli utenti possono cercare eventi in una determinata area geografica o nelle vicinanze di un luogo specifico.

### Filtraggio per Categoria

È possibile filtrare gli eventi in base alla categoria, consentendo agli utenti di trovare eventi che rientrano in una categoria specifica, ad esempio "Sport," "Musica," "Cultura," ecc.

## **Ordinamento per Data e Ora**

Gli eventi sono elencati in base alla data e all'ora. Gli eventi futuri sono visualizzati prima, mentre quelli passati vengono archiviati o nascosti.

## **FILTRAGGIO DEGLI EVENTI PER TIPO DI SEZIONE/BOARD**

### **1. Event Board**

La sezione "Event Board" offre agli utenti una vista completa di tutti gli eventi disponibili nell'applicazione. In questa sezione, gli utenti possono esplorare una vasta gamma di eventi, indipendentemente dal fatto che siano il creatore dell'evento o partecipanti. Possono cercare eventi per titolo, luogo, categoria o data e visualizzare dettagli aggiuntivi sugli eventi che li interessano.

### **2. My Events**

La sezione "My Events" è una sezione dedicata agli utenti che sono creatori di eventi. Qui, gli utenti possono visualizzare e gestire gli eventi che hanno creato. Le azioni principali includono:

- **Modifica:** Gli utenti possono aprire e modificare i dettagli degli eventi che hanno creato, tra cui il titolo, la descrizione, la data, il luogo, la categoria e le foto associate all'evento.
- **Eliminazione:** Gli utenti possono rimuovere eventi che non desiderano più mantenere nell'applicazione. L'eliminazione di un evento comporta la cancellazione di tutte le informazioni correlate ad esso, comprese le foto.

### **3. Registered Events**

La sezione "Registered Events" è una sezione in cui gli utenti possono visualizzare gli eventi a cui hanno scelto di partecipare. Gli utenti possono iscriversi a eventi da "Event Board" o da altre sezioni dell'applicazione. In questa sezione, possono visualizzare tutti gli eventi futuri a cui sono registrati e accedere ai dettagli dell'evento, compresi i dettagli dell'evento e l'elenco dei partecipanti.

Le diverse sezioni di gestione degli eventi offrono agli utenti un'esperienza completa per scoprire, creare, gestire e partecipare agli eventi nell'applicazione. Ognuna di queste sezioni contribuisce a fornire una panoramica completa delle attività legate agli eventi e consente agli utenti di interagire con gli eventi in modi diversi in base ai loro ruoli e interessi.

# Implementazione delle Notifiche in Tempo Reale con SSE

## Cos'è SSE (Server-Sent Events)

SSE è una tecnologia web che consente al server di inviare in modo asincrono dati ai client attraverso una connessione HTTP persistente. A differenza di altre tecnologie di comunicazione bidirezionale come WebSockets, SSE è una soluzione basata su eventi

unidirezionale, il che lo rende ideale per situazioni in cui il server deve inviare aggiornamenti ai client in modo "push", come nel caso delle notifiche in tempo reale.

## Implementazione del Flusso delle Notifiche

1. **Generazione delle Notifiche:** Quando un utente compie un'azione che genera notifiche (ad esempio, partecipare a un evento o ricevere un messaggio), il server crea le notifiche corrispondenti e le memorizza nel database con informazioni come il messaggio, la data e lo stato (solitamente "non lette").
2. **Inizializzazione SSE:** Il server apre una connessione SSE per ciascun client che desidera ricevere notifiche in tempo reale. Questa connessione persistente rimarrà aperta fino a quando il client o il server decideranno di chiuderla.
3. **Invio di Notifiche ai Client:** Quando una nuova notifica viene creata o quando uno stato di notifica cambia (ad esempio, da "non lette" a "lette" quando l'utente le visualizza), il server invia un evento SSE al client con i dati aggiornati. Questi dati solitamente vengono inviati in formato JSON e possono contenere tutte le notifiche o solo quelle che sono state modificate.
4. **Aggiornamento del Client:** Il client riceve l'evento SSE e aggiorna dinamicamente la lista delle notifiche. Ad esempio, se un utente riceve una nuova notifica, il client può mostrare una notifica visiva o incrementare il contatore di notifiche non lette. Quando l'utente consulta la lista delle notifiche, il client invia una richiesta al server per segnare le notifiche come "lette."

## Vantaggi di SSE

- **Semplicità:** SSE è relativamente semplice da implementare sia sul lato del server che su quello del client.
- **Efficienza:** Le connessioni SSE sono leggere e richiedono meno overhead rispetto a WebSocket.
- **Compatibilità:** La maggior parte dei browser moderni supporta SSE in modo nativo.

## Considerazioni sulla Sicurezza

È importante assicurarsi che la connessione SSE sia sicura e che il server autentichi correttamente i client prima di inviare dati sensibili, infatti nel nostro caso la connessione SSE dal client avviene solo se quest'ultimo è autenticato. Inoltre, è fondamentale proteggere il server da attacchi di congestione o DoS (Denial of Service), poiché le connessioni SSE rimangono aperte per lunghi periodi, nel nostro caso è stato aggiunto un nuovo filtro apposito per gli attacchi Dos (RequestLimitFilter).

Questa implementazione permette di offrire agli utenti un'esperienza in tempo reale, garantendo che ricevano notifiche immediate delle attività pertinenti al loro utilizzo del sito web.



