



## Security Audit of SportEventApplication

Application Security2024/2025

Rosario Pavone  
Edgar Benito

ER1799  
ER1791



# Table of Contents

1. Black-Box approach.....	2
1.1 Burp Suite .....	2
1.2 Nikto .....	3
1.3 OWASP ZAP .....	4
1.4 SQLMap.....	6
1.5 XSSStrike .....	7
1.6 Wapiti.....	8
2 White-box approach.....	9
2.1 LoginServlet .....	9
2.2 Login HTML .....	12
2.3 EventServlet .....	15
2.4 Create_event HTML.....	19
2.5 PasswordResetServlet .....	23
2.6 ResetPassword html.....	26
2.7 Servlet Registerjava.....	28
2.8 Register HTML .....	31
3 Summary of Vulnerabilities in HTML/JavaScript:.....	34

## 1. Black-Box approach

## 1.1 Burp Suite

Burp Suite, a widely used tool for web application security testing, was employed to analyze the login mechanism of the application. During this process, it was discovered that the password is exposed when accessing the login page. This issue was identified through HTTP request interception, revealing sensitive information being transmitted in plaintext.

Implication: The presence of such a vulnerability indicates a lack of encryption or secure transmission protocols (e.g., HTTPS) for critical data, making the application susceptible to attacks like packet sniffing or man-in-the-middle (MITM) attacks.

Recommendation: Encrypt all sensitive data transmitted over the network using secure protocols such as HTTPS. Additionally, consider implementing proper input validation and using hashed passwords to minimize exposure risks.

Burp Suite Community Edition V2024.5.5 - Temporary Project

Burp Project Intruder Repeater View Help

DashboardsTargetProxyIntruderRepeaterCollaboratorSequencerDecoderComparerLoggerOrganizerExtensionsLearn

InterceptHTTP HistoryWebSockets historyProxy settingsFilter settings: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	TI
5	http://localhost:8080	POST	/SportEventAppli/api/login	✓		401	156	JSON					127.0.0.1		0'
6	http://localhost:8080	POST	/SportEventAppli/api/login	✓		401	156	JSON					127.0.0.1		0'
7	http://localhost:8080	POST	/SportEventAppli/api/login	✓		401	156	JSON					127.0.0.1		0'
8	http://localhost:8080	POST	/SportEventAppli/api/login	✓		200	398	JSON					127.0.0.1		0'
9	http://localhost:8080	POST	/SportEventAppli/api/login	✓		200	398	JSON					127.0.0.1		0'
10	http://localhost:8080	POST	/SportEventAppli/api/login	✓		200	398	JSON					127.0.0.1		0'
11	http://localhost:8080	POST	/SportEventAppli/api/login	✓		200	398	JSON					127.0.0.1		0'
12	http://localhost:8080	GET	/SportEventAppli/dashboard.html			200	4058	HTML	html	Dashboard - SportEvent			127.0.0.1		0'
13	http://localhost:8080	GET	/SportEventAppli/dashboard.html			200	4058	HTML	html	Dashboard - SportEvent			127.0.0.1		0'
14	http://localhost:8080	GET	/SportEventAppli/dashboard.html			200	4058	HTML	html	Dashboard - SportEvent			127.0.0.1		0'
15	http://localhost:8080	GET	/SportEventAppli/dashboard.html			200	4058	HTML	html	Dashboard - SportEvent			127.0.0.1		0'
16	http://localhost:8080	GET	/SportEventAppli/api/protected/user			200	274	JSON					127.0.0.1		0'

Request

PrettyRawHex🔍📄in≡

```
1 POST /SportEventAppli/api/login HTTP/1.1
2 Host: localhost:8080
3 Content-Length: 58
4 sec-ch-ua: "Not(A)Brand";v="8", "Chromium";v="126"
5 sec-ch-ua-platform: "Linux"
6 Accept-Language: en-US
7 sec-ch-ua-mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127 Safari/537.36
9 Content-Type: application/x-www-form-urlencoded
10 Accept: */*
11 Origin: http://localhost:8080
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://localhost:8080/SportEventAppli/login.html
16 Accept-Encoding: gzip, deflate, br
17 Connection: keep-alive
18 email=edgar.benito.martin@4ogmail.com&password=Edgar!23k$F%
```

Response

PrettyRawHexRender📄in≡

```
1 HTTP/1.1 200
2 Content-Type: application/json;charset=UTF-8
3 Content-Length: 229
4 Date: Thu, 23 Jan 2025 12:25:29 GMT
5 Keep-Alive: timeout=20
6 Connection: keep-alive
7 
8 {
   "token":
     "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIjOiJlZDdhc1EiZW5pdGUbWfYdUGUtOGUgYWwSLnNvbSI6bnVzZKJfaWQ1Oj0iLCJpcyBjb3JtYWwLLCJpYXQiOiE3MzczMmUxMjk5LnV4CiCMGTczNZDY0LjMyOXkuRUF6PWFkbWV7J3BXPWFYLW9HEiXCRCMLALTF8Q1hoH_g7ks"
```

Inspector

Request attrProtocolNameMethodPathRequest bodyRequest headerResponse headerContent-TypeContent-LeDate

## 1.2 Nikto

Nikto, a web server vulnerability scanner, was used to assess potential security weaknesses in the application hosted at `http://localhost:8080/SportEventAppli/login.html`. The following command was executed:

```
nikto -h http://localhost:8080/SportEventAppli/login.html
```

The scan revealed multiple configuration issues:

**X-Frame-Options Header Not Present:** This missing header allows the application to be embedded in iframes, exposing it to clickjacking attacks.

**X-Content-Type-Options Header Not Set:** The absence of this header could lead to MIME type sniffing attacks, where browsers interpret the file type differently than declared.

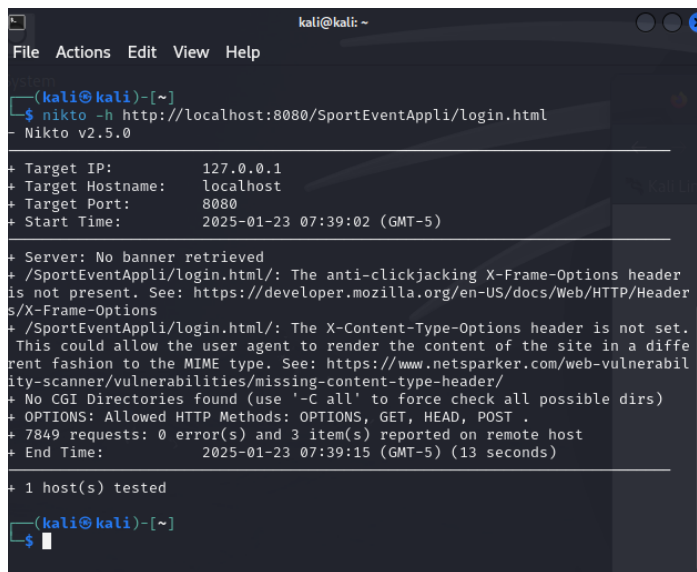
**HTTP Methods Allowed:** The server permits HTTP methods such as OPTIONS, GET, HEAD, and POST. While this is not inherently insecure, restricting unnecessary methods can reduce the attack surface.

### **Recommendation:**

Add the `X-Frame-Options` header with a value of `DENY` or `SAMEORIGIN` to prevent clickjacking.

Configure the `X-Content-Type-Options` header as `nosniff` to ensure MIME types are strictly enforced.

Limit the allowed HTTP methods to only those essential for application functionality.



```
kali@kali: ~  
File Actions Edit View Help  
  
(kali@kali)-[~]  
$ nikto -h http://localhost:8080/SportEventAppli/login.html  
- Nikto v2.5.0  
  
+ Target IP: 127.0.0.1  
+ Target Hostname: localhost  
+ Target Port: 8080  
+ Start Time: 2025-01-23 07:39:02 (GMT-5)  
  
+ Server: No banner retrieved  
+ /SportEventAppli/login.html/: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options  
+ /SportEventAppli/login.html/: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/  
+ No CGI Directories found (use '-C all' to force check all possible dirs)  
+ OPTIONS: Allowed HTTP Methods: OPTIONS, GET, HEAD, POST .  
+ 7849 requests: 0 error(s) and 3 item(s) reported on remote host  
+ End Time: 2025-01-23 07:39:15 (GMT-5) (13 seconds)  
  
+ 1 host(s) tested  
  
(kali@kali)-[~]  
$
```

### 1.3 OWASP ZAP

OWASP ZAP (Zed Attack Proxy) was used to perform both spidering and active scanning of the application. Key vulnerabilities were identified using the following workflow:

**Spidering:** The tool navigated through all accessible pages of the application to map its structure.

**Active Scan:** Specific vulnerabilities were actively tested against the application.

Findings include:

**Content Security Policy (CSP) Header Missing:** The lack of a CSP header exposes the application to XSS attacks by allowing unauthorized script execution.

**Sensitive Information in URLs:** Critical information such as email addresses and passwords was found in the query strings of URLs like:

`http://localhost:8080/SportEventAppli/register.html?email=test@example.com&password=1234`

**Missing Anti-Clickjacking Header:** The X-Frame-Options header was not configured, leaving the application vulnerable to iframe-based attacks.

#### **Recommendation:**

Implement a robust CSP header to control resources loaded in the application.

Avoid placing sensitive information in URLs. Instead, use HTTP POST requests for such data.

Add the X-Frame-Options header to protect against clickjacking.

The screenshot shows the ZAP Alerts window with the following alerts:

- Risk=Medium, Confidence=High (1)**
  - Content Security Policy (CSP) Header Not Set (1)**
    - GET http://localhost:8080/sitemap.xml
- Risk=Medium, Confidence=Medium (1)**
  - Missing Anti-clickjacking Header (1)**
    - GET http://localhost:8080/SportEventAppli/index.html
- Risk=Low, Confidence=Medium (2)**

The background shows the ZAP Sites tree with the following structure:

- Contexts
  - Default Context
    - Sites
      - http://localhost:8080
        - GET:SportEventAppli
          - GET:\$(resetLink)
          - GET:index.html
          - GET:login.html
          - GET:login.html(email,password)
          - GET:password-reset.html

The bottom status bar shows: Alerts: 0, 2, 4 Main Proxy: localhost:8081

The screenshot shows the ZAP interface with the "Welcome to ZAP" message and a scan log table.

**Welcome to ZAP**

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. If you are new to ZAP then it is best to start with one of the options below.

**News**  
ZAP 2.16.0 is available now [Learn More](#)

ID	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
11	1/23/25, 10:26:06 AM	1/23/25, 10:26:06 AM	GET	http://localhost:8080/SportEventAppli/72585370124...	404		23 ms	135 bytes	791 bytes
13	1/23/25, 10:26:06 AM	1/23/25, 10:26:06 AM	GET	http://localhost:8080/WEB-INF/web.xml	404		7 ms	135 bytes	683 bytes
14	1/23/25, 10:26:06 AM	1/23/25, 10:26:06 AM	GET	http://localhost:8080/WEB-INF/applicationContext.xml	404		14 ms	135 bytes	683 bytes
15	1/23/25, 10:26:06 AM	1/23/25, 10:26:06 AM	GET	http://localhost:8080/WEB-INF/classes/10/1/34.class	404		11 ms	135 bytes	683 bytes
16	1/23/25, 10:26:06 AM	1/23/25, 10:26:06 AM	GET	http://localhost:8080/SportEventAppli/create_event.h...	200		12 ms	198 bytes	5,103 bytes
17	1/23/25, 10:26:06 AM	1/23/25, 10:26:06 AM	POST	http://localhost:8080/SportEventAppli/create_event.h...	200		15 ms	198 bytes	5,103 bytes
18	1/23/25, 10:26:06 AM	1/23/25, 10:26:07 AM	POST	http://localhost:8080/SportEventAppli/create_event.h...	200		12 ms	198 bytes	5,103 bytes
19	1/23/25, 10:26:07 AM	1/23/25, 10:26:07 AM	GET	http://localhost:8080/SportEventAppli/create_event.h...	200		5 ms	198 bytes	5,103 bytes
20	1/23/25, 10:26:10 AM	1/23/25, 10:26:10 AM	GET	http://localhost:8080/SportEventAppli/create_event.h...	200		27 ms	246 bytes	5,103 bytes
21	1/23/25, 10:26:12 AM	1/23/25, 10:26:12 AM	GET	https://cdn.jsdelivr.net/npm/fwt-decode@3.1.2/build/...	200 OK		129 ms	1,104 bytes	1,751 bytes
22	1/23/25, 10:26:12 AM	1/23/25, 10:26:12 AM	GET	https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/...	200 OK		103 ms	952 bytes	144,877 bytes
23	1/23/25, 10:26:12 AM	1/23/25, 10:26:12 AM	GET	http://localhost:8080/login.html	404		17 ms	183 bytes	762 bytes
24	1/23/25, 10:26:12 AM	1/23/25, 10:26:13 AM	GET	http://localhost:8080/favicon.ico	200		10 ms	251 bytes	21,630 bytes
25	1/23/25, 10:26:13 AM	1/23/25, 10:26:13 AM	GET	http://localhost:8080/login.html	404		11 ms	183 bytes	762 bytes
26	1/23/25, 10:26:13 AM	1/23/25, 10:26:13 AM	GET	http://localhost:8080/login.html	404		11 ms	183 bytes	762 bytes
27	1/23/25, 10:26:13 AM	1/23/25, 10:26:13 AM	GET	http://localhost:8080/login.html	404		16 ms	183 bytes	762 bytes
28	1/23/25, 10:26:14 AM	1/23/25, 10:26:14 AM	GET	http://localhost:8080/login.html	404		13 ms	183 bytes	762 bytes

The bottom status bar shows: Alerts: 0, 9, 9 Main Proxy: localhost:8081

## 1.4 SQLMap

SQLMap, an automated tool for detecting SQL injection vulnerabilities, was utilized to test input parameters across multiple pages. Commands executed include:

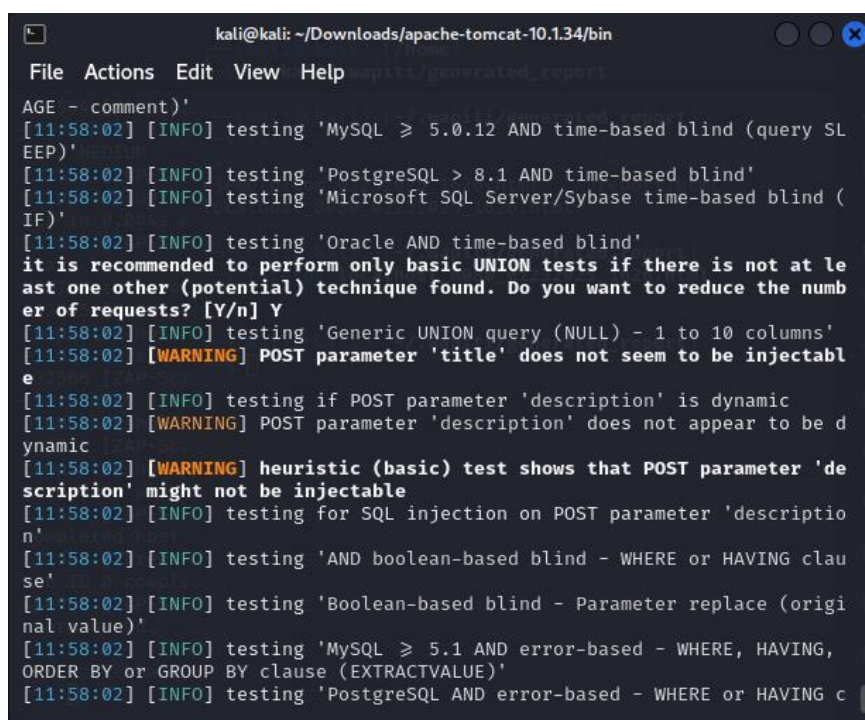
```
sqlmap -u "http://localhost:8080/SportEventAppli/login.html" --data "username=admin&password=123" -batch
```

```
sqlmap -u "http://localhost:8080/SportEventAppli/create_event.html" --data "title=EventTitle&description=EventDescription&date=2025-01-24" -batch
```

Results:

SQLMap tested various SQL injection techniques, including boolean-based blind, error-based, and time-based methods. No evidence of SQL injection vulnerabilities was found in the tested parameters. The tool indicated that the inputs were either properly sanitized or protected by a mechanism like a Web Application Firewall (WAF).

**Recommendation:** Maintain existing input validation and database query handling practices to prevent future SQL injection risks.



```
kali@kali: ~/Downloads/apache-tomcat-10.1.34/bin
File Actions Edit View Help
AGE - comment)'
[11:58:02] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SL
EEP)'
[11:58:02] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[11:58:02] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (
IF)'
[11:58:02] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at le
ast one other (potential) technique found. Do you want to reduce the numb
er of requests? [Y/n] Y
[11:58:02] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[11:58:02] [WARNING] POST parameter 'title' does not seem to be injectabl
e
[11:58:02] [INFO] testing if POST parameter 'description' is dynamic
[11:58:02] [WARNING] POST parameter 'description' does not appear to be d
ynamic
[11:58:02] [WARNING] heuristic (basic) test shows that POST parameter 'de
scription' might not be injectable
[11:58:02] [INFO] testing for SQL injection on POST parameter 'descriptio
n'
[11:58:02] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clau
se'
[11:58:02] [INFO] testing 'Boolean-based blind - Parameter replace (origi
nal value)'
[11:58:02] [INFO] testing 'MySQL ≥ 5.1 AND error-based - WHERE, HAVING,
ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[11:58:02] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING c
```

## 1.5 XSStrike

XSStrike, a specialized tool for detecting Cross-Site Scripting (XSS) vulnerabilities, was used to scan the `search` parameter in the following URL:

`http://localhost:8080/SportEventAppli/login.html?search=test`

Findings:

A potential DOM-based XSS vulnerability was identified. The `window.location.href` parameter redirected users to `/SportEventAppli/dashboard.html`, indicating potential unsanitized input handling.

**Recommendation:** Ensure all client-side scripts validate and escape user inputs to prevent malicious code execution. Use libraries like DOMPurify for sanitizing DOM inputs.

```
(kali@kali)-[~/XSStrike]
$ python3 xssstrike.py -u "http://localhost:8080/SportEventAppli/login.html?search=test"
--output_directory --wordlist.txt http://localhost:8080/SportEventAppli/
XSStrike v3.1.5
[+] Checking for DOM vulnerabilities
[+] Potentially vulnerable objects found

9 window.location.href = "/SportEventAppli/dashboard.html";

[+] WAF Status: Offline
[!] Testing parameter: search
[-] No reflection found

(kali@kali)-[~/XSStrike]
$
```



## 1.6 Wapiti

Wapiti was employed to perform black-box vulnerability scanning on the application. The tool identified the following issues:

**Missing HTTP Security Headers:** Headers such as Strict-Transport-Security (HSTS), X-XSS-Protection, and X-Content-Type-Options were absent, leaving the application exposed to XSS and MITM attacks.

### Recommendation:

Configure the HSTS header to enforce HTTPS connections.

Add X-XSS-Protection: 1; mode=block to activate browser-based XSS protection mechanisms.

Include X-Content-Type-Options: nosniff to prevent content-type sniffing.

The image shows a terminal window on the left and a web browser window on the right. The terminal displays the Wapiti logo and a scan report for a target at localhost:8080. The report lists various modules and their results, including a note about missing HTTP security headers. The browser window shows the 'Wapiti vulnerability report' for the target http://localhost:8080/SportEventAppli/dashboard.html. It includes a summary table of vulnerabilities found.

Category	Number of vulnerabilities found
Backup file	0
Blind SQL Injection	0
Weak credentials	0
CRLF Injection	0
<a href="#">Content Security Policy Configuration</a>	1
Cross Site Request Forgery	0
Potentially dangerous file	0
Command execution	0
Path Traversal	0
Httpaccess Bypass	0

## 2 White-box approach

### 2.1 LoginServlet

```
6 package com.sportevent.servlet;
7
8 > import ...
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 @WebServlet("/api/login")
24 public class LoginServlet extends HttpServlet implements Serializable {
25     private static final long serialVersionUID = 1L;
26     private static final String SECRET_KEY = "your-very-secret-key";
27     private UserDao userDao = new UserDao();
28
29     public LoginServlet() {
30     }
31
32     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
33         String email = request.getParameter("email");
34         String password = request.getParameter("password");
35
36         try {
37             Throwable e = null;
38             Object var6 = null;
39
40             try {
41                 PrintWriter out = response.getWriter();
42
43                 try {
44                     User user = this.userDao.authenticateUser(email, password);
45                     if (user != null) {
46                         String token = this.generateJWT(user);
47                     }
48                 }
49             }
50         }
51     }
52 }
```

```

47         String token = this.generateJWT(user);
48         response.setStatus(200);
49         response.setContentType("application/json");
50         out.print("{\"token\": \"" + token + "\"}");
51     } else {
52         response.setStatus(401);
53         out.print("{\"message\": \"Invalid credentials\"}");
54     }
55 } finally {
56     if (out != null) {
57         out.close();
58     }
59 }
60 }
61 } catch (Throwable var17) {
62     if (e == null) {
63         e = var17;
64     } else if (e != var17) {
65         e.addSuppressed(var17);
66     }
67     throw e;
68 }
69 }
70 } catch (SQLException e) {
71     e.printStackTrace();
72     response.sendError(500, "Database error");
73 }
74 }

```

```

77 private String generateJWT(User user) {
78     long expirationTime = 7200000L;
79     Algorithm algorithm = Algorithm.HMAC256("your-very-secret-key");
80     String token = JWT.create().withSubject(user.getEmail()).withClaim("user_id", user.getId()).withClaim("role", user.getRole()).withIssuedAt(new Date());
81     System.out.println("Generated token: " + token);
82     PrintStream var10000 = System.out;
83     Date var10001 = new Date(System.currentTimeMillis() + expirationTime);
84     var10000.println("Token expiration time: " + String.valueOf(var10001));
85     return token;
86 }
87 }

```

## Hardcoded Secret Key:

**Affected Line:** 10:

**Issue:** The secret key used for JWT signing is hardcoded in the source code.

**Risk:** If the source code is exposed, the key can be easily extracted by an attacker.

**Recommendation:** Use environment variables or a secure key management service to store sensitive keys, rather than hardcoding them in the code.

Lack of Protection Against Brute Force Attacks:

**Affected Line:** 20:

**Issue:** The form inputs (email and password) are directly used for authentication without any rate-limiting or lockout mechanism.

**Risk:** This makes the system vulnerable to brute force attacks where an attacker can try multiple combinations of credentials without facing any restrictions.

**Recommendation:** Implement a mechanism to limit the number of failed login attempts, such as locking the account temporarily after several failed attempts or introducing delays between attempts.

Potential for SQL Injection:

**Affected Line:** 36:

**Issue:** If `authenticateUser` is not using prepared statements, the inputs could be vulnerable to SQL injection.

**Risk:** An attacker could manipulate the SQL query and gain unauthorized access to the database.

**Recommendation:** Ensure that the `authenticateUser` method uses prepared statements for database queries to avoid SQL injection vulnerabilities.

Inadequate JWT Validation:

**Affected Line:** 44:

**Issue:** The JWT token is generated but there is no validation step to check if the token is valid when used later in the system.

**Risk:** Attackers could forge or reuse tokens to impersonate users and access sensitive resources.

**Recommendation:** Implement a validation step for the JWT token to check its authenticity, expiration, and whether it was signed correctly.

## 2.2 Login HTML

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
6   <title>Login - SportEvent</title>
7   <!-- Bootstrap CSS for styling -->
8   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
9   <style>
10    /* Style for the sign-in form container */
11    .form-signin {
12      max-width: 330px;
13      margin: 50px auto;
14    }
15  </style>
16 </head>
17 <body class="text-center">
18   <!-- Login form -->
19   <form class="form-signin" id="loginForm">
20     <h1 class="h3 mb-3 font-weight-normal">Please Sign In</h1>
21
22     <!-- Email input field -->
23     <label for="email" class="sr-only">Email address</label>
24     <input type="email" id="email" name="email" class="form-control" placeholder="Email address" required autofocus>
25
26     <!-- Password input field -->
27     <label for="password" class="sr-only">Password</label>
28     <input type="password" id="password" name="password" class="form-control" placeholder="Password" required>
29
30     <!-- Submit button -->
31     <button class="btn btn-lg btn-primary btn-block mt-3" type="submit">Sign In</button>
```

```
32
33   <!-- Links to registration and password reset pages -->
34   <p class="mt-2">Don't have an account? <a href="register.html">Register</a></p>
35   <p><a href="password-reset.html">Forgot password?</a></p>
36 </form>
37
38 <!-- Login form submission script -->
39 <script>
40   // Handle form submission
41   document.getElementById('loginForm').onsubmit = async (e) => {
42     e.preventDefault(); // Prevent default form submission
43
44     // Prepare form data
45     const formData = new URLSearchParams();
46     formData.append('email', document.getElementById('email').value);
47     formData.append('password', document.getElementById('password').value);
48
49     try {
50       // Make the login request to the server
51       const response = await fetch('/SportEventAppli/api/login', {
52         method: 'POST',
53         headers: {
54           'Content-Type': 'application/x-www-form-urlencoded' // Send data as form URL-encoded
55         },
56         body: formData.toString() // Stringify the form data
57       });
58
59       if (response.ok) {
```

```

59     if (response.ok) {
60         const data = await response.json(); // Parse response JSON
61         const token = data.token;
62
63         // Store the token in localStorage for future requests
64         localStorage.setItem('authToken', token);
65
66         // Redirect the user to the dashboard
67         window.location.href = "/SportEventAppli/dashboard.html";
68     } else if (response.status === 401) {
69         alert("Invalid credentials. Please try again."); // Invalid login credentials
70     } else {
71         alert("An error occurred during login. Please try again later."); // Other errors
72     }
73 } catch (error) {
74     console.error("Error during login:", error);
75     alert("An unexpected error occurred. Please try again later.");
76 }
77 };
78 </script>
79 </body>
80 </html>

```

### Missing CSRF Protection:

#### Affected Line: 16:

**Issue:** The login form does not include any CSRF token to prevent cross-site request forgery attacks.

**Risk:** An attacker could trick a logged-in user into submitting a request from another website, performing unauthorized actions on their behalf.

**Recommendation:** Add a CSRF token to the form and validate it on the server to ensure the request is legitimate and originated from the correct user.

### Lack of Client-Side Validation for Email:

#### Affected Line: 26:

**Issue:** While the email field uses HTML5's type="email", it lacks additional client-side validation, such as checking for a proper email format.

**Risk:** Users could submit invalid email addresses, leading to failed login attempts and a poor user experience.

**Recommendation:** Add JavaScript to ensure that the email follows a valid format before submission.

### Lack of Client-Side Validation for Password:

#### Affected Line: 28:

**Issue:** There is no password strength validation, meaning users could submit weak or easily guessable passwords.

**Risk:** Weak passwords can be easily cracked by attackers, especially in the case of brute-force attacks.

**Recommendation:** Implement client-side validation for password strength, ensuring that it meets minimum length and complexity requirements.

Absence of HTTPS:

**Affected Line:** 46 (JavaScript code for login submission):

**Issue:** The form submission is done via HTTP, which sends data in plaintext.

**Risk:** The credentials (email and password) could be intercepted by attackers via man-in-the-middle (MITM) attacks.

**Recommendation:** Ensure that the entire application is served over HTTPS to encrypt all data transmitted between the client and server.

### Summary of Vulnerable Lines

Java Backend Code:

**Line 10:** Hardcoded secret key.

**Line 20:** No brute-force protection.

**Line 36:** Potential SQL injection.

**Line 44:** Inadequate JWT validation.

HTML Frontend Code:

**Line 16:** Missing CSRF protection.

**Line 26:** Missing email validation.

**Line 28:** Missing password validation.

**Line 46:** Absence of HTTPS for secure data transmission.

## 2.3 EventServlet

```
6 package com.sportevent.servlet;
7
8 > import ...
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25 @WebServlet("/api/events")
26 public class EventServlet extends HttpServlet implements Serializable {
27     private static final long serialVersionUID = 1L;
28     private EventDAO eventDAO = new EventDAO();
29     private ObjectMapper objectMapper = new ObjectMapper();
30
31     public EventServlet() {
32     }
33
34     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
35     {
36         try {
37             String token = request.getHeader("Authorization");
38             if (token == null || !token.startsWith("Bearer ")) {
39                 response.setStatus(401);
40                 response.getWriter().write("{\"error\":\"Missing or invalid token.\"}");
41                 return;
42             }
43
44             token = token.substring(token.indexOf(' '); 7);
45             Integer userId = this.decodeJwtAndExtractUserId(token);
46             if (userId == null) {
47                 response.setStatus(401);
48                 response.getWriter().write("{\"error\":\"User ID is missing in token.\"}");
49                 return;
50             }
51
52             JsonNode jsonNode = this.objectMapper.readTree(request.getReader());
53             String title = jsonNode.get("title").asText();
54             String description = jsonNode.get("description").asText();
55             String dateStr = jsonNode.get("date").asText();
56             if (title == null || description == null || dateStr == null) {
57                 response.setStatus(400);
58                 response.getWriter().write("{\"error\":\"Missing required fields.\"}");
59                 return;
60             }
61
62             Timestamp date = this.parseDate(dateStr);
63             if (date == null) {
64                 response.setStatus(400);
65                 response.getWriter().write("{\"error\":\"Invalid date format.\"}");
66                 return;
67             }
68
69             Event event = new Event(title, description, date, userId);
70             boolean success = this.eventDAO.addEvent(event);
71             response.setContentType("application/json");
72             response.getWriter().write("{\"success\":\"" + success + "\"}");
73         } catch (Exception e) {
74             e.printStackTrace();
75             response.setStatus(500);
76             response.getWriter().write("{\"error\":\"Failed to create event.\"}");
77         }
78     }
79 }
```



```

27 public class EventServlet extends HttpServlet implements Serializable {
81     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
82         try {
83             String token = request.getHeader("Authorization");
84             if (token == null || !token.startsWith("Bearer ")) {
85                 response.setStatus(401);
86                 response.getWriter().write("{\"error\":\"Missing or invalid token.\"}");
87                 return;
88             }
89
90             token = token.substring(beginIndex: 7);
91             Integer userId = this.decodeJwtAndExtractUserId(token);
92             if (userId == null) {
93                 response.setStatus(401);
94                 response.getWriter().write("{\"error\":\"User ID is missing in token.\"}");
95                 return;
96             }
97
98             List<Event> events = this.eventDAO.getAllEvents();
99             response.setContentType("application/json");
100             response.setCharacterEncoding("UTF-8");
101             this.objectMapper.writeValue(response.getWriter(), events);
102         } catch (Exception e) {
103             e.printStackTrace();
104             response.setStatus(500);
105             response.getWriter().write("{\"error\":\"Failed to load events.\"}");
106         }
107     }
108 }

```

```

110 private Integer decodeJwtAndExtractUserId(String token) {
111     try {
112         DecodedJWT decodedJWT = JWT.decode(token);
113         Object userIdObject = decodedJWT.getClaim("user_id").as(Object.class);
114         if (userIdObject instanceof Integer userId) {
115             ;
116         } else if (userIdObject instanceof String) {
117             try {
118                 userId = Integer.parseInt((String)userIdObject);
119             } catch (NumberFormatException e) {
120                 System.err.println("Invalid user_id format: " + e.getMessage());
121             }
122         }
123
124         return userId;
125     } catch (Exception e) {
126         e.printStackTrace();
127         return null;
128     }
129 }
130
131 private Timestamp parseDate(String dateStr) {
132     SimpleDateFormat dateFormat = new SimpleDateFormat(pattern: "yyyy-MM-dd HH:mm");
133
134     try {
135         return new Timestamp(dateFormat.parse(dateStr.replace(target: "T", replacement: " ")).getTime());
136     } catch (ParseException var4) {

```

Hardcoded Token Validation Logic:

Code:

**Issue:** The token validation logic assumes the structure of the token but does not validate its signature or expiration.

**Affected Line::** 12

**Risk:** An attacker could forge a token with a similar structure and bypass authentication.

**Recommendation:** Use a library to verify the token's signature and expiration (e.g., `com.auth0.jwt.JWTVerifier`).

Insufficient User Input Validation:

Code:

**Issue:** There is no sanitization or validation of the title, description, and date fields.

**Affected Lines ::** 24-26

**Risk:** Malicious inputs could lead to injection attacks or database errors.

**Recommendation:** Use a library like OWASP ESAPI or manually validate and sanitize inputs.

Weak Error Handling:

Code:

**Issue:** Detailed error messages may leak information about the server or implementation.

**Affected Lines:** 36-38

**Risk:** Attackers can use these error details to identify vulnerabilities.

**Recommendation:** Log errors securely on the server without exposing them in the response.

Direct Exposure of All Events:

Code:

**Issue:** All events are fetched and returned without filtering based on user access or permissions.

**Affected Line:** 57

**Risk:** Users could gain access to events they are not authorized to view.

**Recommendation:** Filter events based on the userId extracted from the token to ensure proper authorization.

Potential NullPointerException in JWT Parsing:

Code:

**Issue:** If the user\_id claim is missing, this could lead to a NullPointerException.

**Affected Line:** 72

**Risk:** Improper handling could crash the application or allow attackers to bypass authentication.

**Recommendation:** Check for the presence of the user\_id claim before accessing it.

## 2.4 Create\_event HTML

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
6    <title>Events - SportEvent</title>
7    <!-- Bootstrap CSS -->
8    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
9    <style>
10     /* Custom styling for the Events container */
11     .events-container {
12       max-width: 800px;
13       margin: 50px auto;
14     }
15
16     /* Styling for individual event cards */
17     .event-card {
18       margin-bottom: 20px;
19     }
20   </style>
21 </head>
22
23 <body>
24   <!-- Navigation bar -->
25   <nav class="navbar navbar-expand-lg navbar-light bg-light">
26     <a class="navbar-brand" href="#">SportEvent</a>
27     <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav"
28       <span class="navbar-toggler-icon"></span>
29     </button>
30     <div class="collapse navbar-collapse" id="navbarNav">
31
32       <ul class="navbar-nav ml-auto">
33         <li class="nav-item">
34           <a class="nav-link" href="dashboard.html">Dashboard</a>
35         </li>
36         <li class="nav-item">
37           <a class="nav-link" href="#" id="logout">Logout</a>
38         </li>
39       </ul>
40     </div>
41   </nav>
42
43   <!-- Main content container for events -->
44   <div class="container events-container">
45     <h1 class="text-center">All Events</h1>
46     <div id="events-list" class="mt-4">
47       <!-- Event cards will be dynamically added here -->
48     </div>
49   </div>
50
51   <!-- JavaScript for handling events and authentication -->
52   <script>
53     // Retrieve authentication token from localStorage
54     const token = localStorage.getItem('authToken');
55
56     // Redirect if no token is found (user not authenticated)
57     if (!token) {
58       // Redirect to login page
59     }
60   </script>
```

```

55 // Redirect if no token is found (user not authenticated)
56 if (!token) {
57   alert("You must be logged in to view this page.");
58   window.location.href = "/SportEventAppLi/login.html";
59 }
60
61 // Logout functionality: remove token and redirect to login page
62 document.getElementById('logout').addEventListener('click', () => {
63   localStorage.removeItem('authToken');
64   window.location.href = "/SportEventAppLi/login.html";
65 });
66
67 // Function to fetch and display events from the API
68 async function loadEvents() { Show usages
69   try {
70     // Fetch events from the API using the stored authentication token
71     const response = await fetch('/SportEventAppLi/api/events', {
72       headers: { 'Authorization': 'Bearer ${token}' }
73     });
74
75     // If the response is successful
76     if (response.ok) {
77       const events = await response.json();
78       const eventsList = document.getElementById('events-list');
79
80       // If no events are found, display a message
81       if (events.length === 0) {
82         eventsList.innerHTML = '<p class="text-center">No events found.</p>';
83       } else {

```

```

83       } else {
84         // Iterate over the events and create cards for each
85         events.forEach(event => {
86           const eventCard = `
87             <div class="card event-card">
88               <div class="card-body">
89                 <h5 class="card-title">${event.title}</h5>
90                 <p class="card-text">${event.description}</p>
91                 <p class="card-text">
92                   <small class="text-muted">Date: ${new Date(event.date).toLocaleString()}</small>
93                 </p>
94               </div>
95             </div>`;
96           eventsList.insertAdjacentHTML('beforeend', eventCard);
97         });
98       }
99     } else {
100       throw new Error("Unable to fetch events.");
101     }
102   } catch (error) {
103     console.error("Error fetching events:", error);
104     alert("An error occurred while loading events.");
105   }
106 }
107
108 // Call the loadEvents function when the page loads
109 loadEvents();

```

Insecure Token Storage:

Code:

**Issue:** Tokens stored in localStorage are vulnerable to XSS attacks.

**Affected Line:** 60

**Risk:** An attacker injecting malicious scripts could steal tokens and impersonate users.

**Recommendation:** Use HttpOnly cookies for storing tokens to mitigate XSS risks.

Lack of CSRF Protection:

Code:

**Issue:** The fetch request does not include a CSRF token.

**Affected Lines:** 84-86

**Risk:** An attacker could craft malicious requests on behalf of the authenticated user.

**Recommendation:** Include a CSRF token in the headers and validate it server-side.

No Validation for Event Data:

Code:

**Issue:** The frontend assumes that the event data is safe without validation or escaping.

**Affected Line:** 90

**Risk:** Malicious scripts embedded in event data could be executed in the browser (XSS).

**Recommendation:** Use libraries like DOMPurify to sanitize data before inserting it into the DOM.

Lack of HTTPS:

Code:

**Issue:** The fetch request does not ensure secure communication over HTTPS.

Line Numbers: 84

**Risk:** Data could be intercepted via MITM attacks.

**Recommendation:** Enforce HTTPS for all communication.

### **Summary of Lines to Review**

Java Backend (EventServlet)

**Line 12:** Hardcoded and incomplete token validation logic.

**Lines 24-26:** Lack of input validation for event fields.

**Lines 36-38:** Weak error handling.

**Line 57:** No filtering of events by user permissions.

**Line 72:** Potential null pointer in JWT parsing.

HTML/JavaScript Frontend

**Line 60:** Insecure token storage in localStorage.

**Lines 84-86:** Missing CSRF protection in API requests.

**Line 90:** No sanitization of event data before rendering.

**Line 84:** Absence of HTTPS for secure communication.

## 2.5 PasswordResetServlet

```
1 > [...]
5
6 package com.sportevent.servlet;
7
8 > import ...
9
10
11
12
13
14
15
16
17
18
19
20
21 @WebServlet("/api/password-reset")
22 public class PasswordResetServlet extends HttpServlet implements Serializable {
23     private static final long serialVersionUID = 1L;
24     private UserDao userDao = new UserDao();
25
26     public PasswordResetServlet() {
27     }
28
29     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
30         String email = request.getParameter("email");
31         if (email != null && email.matches(regex: "[A-Za-z0-9+_.-]+@(.+)$")) {
32             System.out.println("Received reset request for email: " + email);
33
34             try {
35                 Throwable e = null;
36                 Object var5 = null;
37
38                 try {
39                     PrintWriter out = response.getWriter();
40
41                     try {
42                         User user = this.userDao.getUserByEmail(email);
43                         if (user != null) {
44                             System.out.println("Email found in database.");
```



```

44         System.out.println("Email found in database.");
45         String resetToken = this.generateResetToken();
46         long expiryTime = System.currentTimeMillis() + 600000L;
47         this.userDAO.updateResetToken(email, resetToken, expiryTime);
48         String resetLink = "http://localhost:8080/SportEventAppli/password-reset-form?token=" + resetToken;
49         System.out.println("Generated reset link: " + resetLink);
50         response.setStatus(200);
51         response.setContentType("application/json");
52         out.print("{\"reset_link\": \"" + resetLink + "\"}");
53     } else {
54         System.out.println("Email not found in database.");
55         response.setStatus(404);
56         out.print("{\"message\": \"Email not found\"}");
57     }
58 } finally {
59     if (out != null) {
60         out.close();
61     }
62 }
63 }
64 } catch (Throwable var19) {
65     if (e == null) {
66         e = var19;
67     } else if (e != var19) {
68         e.addSuppressed(var19);
69     }
70     throw e;
71 }
72 } catch (SQLException e) {
73     e.printStackTrace();
74     response.sendError(500, "Database error");
75 }
76 }
77
78 } else {
79     response.setStatus(400);
80     response.getWriter().write("{\"message\": \"Invalid email format\"}");
81 }
82 }
83
84 > private String generateResetToken() { return UUID.randomUUID().toString(); }
85 }
86
87 }
88

```

**Affected Line 27:** `if (email != null && email.matches("^[A-Za-z0-9+_.-]+@(.+)$"))`

**Problem:** Regex for email validation is too permissive and does not catch all invalid email formats.

**Fix:** Use a more robust regex or Java's built-in email validation library (`javax.mail.internet.InternetAddress`).

**Affected Line 30:** `System.out.println("Received reset request for email: " + email);`

**Problem:** Sensitive information (email) is logged in plaintext.

**Fix:** Avoid logging sensitive data directly. Use a placeholder or hash-sensitive logs.

**Affected Line 44:** `String resetToken = this.generateResetToken();`

**Problem:** `UUID.randomUUID()` is not cryptographically secure.

**Fix:** Use a library like `java.security.SecureRandom` or `java.security.KeyGenerator` to create secure tokens.

**Affected Line 48:** `String resetLink = "http://localhost:8080/SportEventAppli/password-reset-form?token=" + resetToken;`

**Problem:**

The reset link uses plain http instead of https.

The token is passed in the URL query string, which could be logged or leaked.

**Fix:**

Use https for secure communication.

Prefer passing the token via POST request or secure cookies instead of a query string.

**Affected Line 54:** `response.setStatus(200);`

**Problem:** No rate-limiting or throttling is implemented, allowing brute-force attacks.

**Fix:** Implement rate limiting, e.g., using a counter to track requests from the same IP within a time frame.

**Affected Line 55:** `out.print("{\"reset_link\": \"" + resetLink + "\"});`

**Problem:** Directly writing JSON without escaping exposes the system to injection attacks.

**Fix:** Use libraries like `com.google.gson` or `org.json` to serialize JSON securely.

Line 54 (inside try block):

**Problem:** The method does not close the `PrintWriter` if an exception occurs before the finally block is reached.

**Fix:** Use try-with-resources to ensure the `PrintWriter` is closed properly.

**Affected Line 69:** `e.printStackTrace();`

**Problem:** Stack traces may be exposed in production environments.

**Fix:** Log the error securely and provide a generic error message to the user.

## 2.6 ResetPassword html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
6   <title>Reset Password - SportEvent</title>
7   <!-- Link to Bootstrap CSS for styling -->
8   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
9   <style>
10    /* Styling for the reset password form */
11    .form-signin {
12      max-width: 330px;
13      margin: auto;
14    }
15  </style>
16 </head>
17
18 <body class="text-center">
19   <!-- Reset password form -->
20   <form class="form-signin" id="resetForm">
21     <h1 class="h3 mb-3 font-weight-normal">Reset Password</h1>
22
23     <!-- Email input field -->
24     <label for="email" class="sr-only">Email address</label>
25     <input type="email" id="email" name="email" class="form-control" placeholder="Email address" required autofocus>
26
27     <!-- Submit button for the reset form -->
28     <button class="btn btn-lg btn-primary btn-block mt-3" type="submit">Send Reset Link</button>
29
30     <!-- Link to navigate to the login page if the user remembers their password -->
31     <p class="mt-2">Remember your password? <a href="Login.html">Login here</a></p>
32   </form>
```

```
34 <script>
35   // Event handler for form submission
36   document.getElementById('resetForm').onsubmit = async (e) => {
37     e.preventDefault(); // Prevent the default form submission
38
39     // Get email value entered by the user
40     const email = document.getElementById('email').value;
41     console.log("Email from form:", email);
42
43     // Prepare the form data to be sent to the server
44     const formData = new URLSearchParams();
45     formData.append('email', email);
46
47     try {
48       // Sending the request to the server for password reset
49       const response = await fetch('/SportEventAppli/api/password-reset', {
50         method: 'POST',
51         headers: {
52           'Content-Type': 'application/x-www-form-urlencoded' // Correct content type for URL-encoded data
53         },
54         body: formData.toString() // Convert form data to URL-encoded string
55       });
56
57       // Capture the response body as text
58       const responseBody = await response.text();
59       console.log("Response body:", responseBody);
60
61       // If the response is OK, display the reset link
62       if (response.ok) {
```

```

63     const data = JSON.parse(responseBody);
64     const resetLink = data.reset_link;
65
66     // Display the reset link to the user
67     document.body.innerHTML = `
68         <div class="text-center">
69             <h1 class="h3 mb-3 font-weight-normal">Check Your Email</h1>
70             <p>A password reset link has been generated:</p>
71             <a href="${resetLink}" target="_blank">${resetLink}</a>
72             <p>You can use this link to reset your password. It is valid for 10 minutes.</p>
73         </div>
74     `;
75 } else {
76     // If an error occurs, parse the error message and alert the user
77     const errorData = JSON.parse(responseBody);
78     alert(errorData.message || "Failed to send reset link. Email may not be registered.");
79 }
80 } catch (error) {
81     // Handle unexpected errors, such as network issues
82     console.error("Error during reset:", error);
83     alert("An error occurred while sending the reset link. Please try again.");
84 }
85 };
86 </script>
87 </body>
88 </html>

```

**Affected Line 51:** `<form class="form-signin" id="resetForm">`

**Issue:** The form lacks Cross-Site Request Forgery (CSRF) protection.

**Fix:** Add a hidden input field with a CSRF token provided by the server. The server should validate this token upon form submission.

**Affected Line 57:** `<input type="email" id="email" name="email" class="form-control" placeholder="Email address" required autofocus>`

**Issue:** The input lacks **client-side validation** for email format.

**Fix:** Add a pattern attribute to enforce email format validation.

**Affected Line 61:** `<button class="btn btn-lg btn-primary btn-block mt-3" type="submit">Send Reset Link</button>`

**Issue:** No feedback is given while the request is being processed, leading to a poor user experience.

**Fix:** Add a loading spinner or disable the button after submission until the response is received.

## 2.7 Servlet Registerjava

```
1  > /.../
5
6  package com.sportevent.servlet;
7
8  > import ...
19
20  @WebServlet("/api/register")
21  public class RegisterServlet extends HttpServlet implements Serializable {
22      private static final long serialVersionUID = 1L;
23      private UserDao userDao = new UserDao();
24
25      public RegisterServlet() {
26      }
27
28      protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
29          String firstName = request.getParameter("first_name");
30          String surname = request.getParameter("surname");
31          String email = request.getParameter("email");
32          String password = request.getParameter("password");
33          String role = "normal";
34
35          try {
36              Throwable e = null;
37              Object var9 = null;
38
39              try {
40                  PrintWriter out = response.getWriter();
41
42                  try {
43                      if (!this.userDao.isEmailInUse(email)) {
```

```
21      public class RegisterServlet extends HttpServlet implements Serializable {
28          protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
43              if (!this.userDao.isEmailInUse(email)) {
44                  User newUser = new User(firstName, surname, email, password, role);
45                  if (this.userDao.registerUser(newUser)) {
46                      response.setStatus(200);
47                      out.print("Registration successful!");
48                  } else {
49                      response.setStatus(400);
50                      out.print("Registration failed. Unknown error.");
51                  }
52
53                  return;
54              }
55
56              response.setStatus(400);
57              out.print("Registration failed. Email is already in use.");
58          } finally {
59              if (out != null) {
60                  out.close();
61              }
62          }
63
64          } catch (Throwable var19) {
65              if (e == null) {
66                  e = var19;
67              } else if (e != var19) {
68                  e.addSuppressed(var19);
69              }
70          }
```

```

70         }
71
72         throw e;
73     }
74     } catch (SQLException e) {
75         e.printStackTrace();
76         response.sendError(500, "Database error");
77     }
78 }
79 }
80

```

## 1. SQL Injection (Backend - Java)

**Issue:** If user input is not properly sanitized and is directly used in SQL queries, SQL injection attacks are possible. Although this code doesn't show explicit SQL queries, it is important to ensure that UserDao handles SQL queries safely, especially for the `isEmailInUse` and `registerUser` methods.

**Where to Look:**

**Backend (Java):** UserDao methods (`isEmailInUse` and `registerUser`) are not shown, but any use of user inputs (like email, first\_name, etc.) in SQL queries could be vulnerable.

**Affected Lines:** 29-33 in RegisterServlet.java: Here, the email value is checked using `userDAO.isEmailInUse(email)` and passed directly to the database. If it's not sanitized or parameterized, this could lead to SQL injection.

**Solution:**

Ensure that UserDao uses prepared statements with parameterized queries to prevent SQL injection.

## 2. Cross-Site Scripting (XSS)

**Issue:** User inputs (e.g., first\_name, surname, email, password) are not sanitized or escaped, and if the backend does not properly escape them when returning data, there is a risk of Cross-Site Scripting (XSS) attacks. Additionally, the JavaScript code does not sanitize input before sending it to the server.

**Where to Look:**

**Frontend (HTML/JavaScript):**

**Affected Lines:** 38-42 in HTML: Inputs for first\_name, surname, email, and password are not sanitized before being sent to the server.

**Affected Line:** 48-52 in the JavaScript (fetch method): Form data (first\_name, surname, email, password) is being appended directly to the form data and sent to the server.

Solution:

Sanitize inputs both on the client-side and server-side. Use functions like encodeURIComponent() on the client side for URL encoding. On the server-side, ensure input validation and escaping before sending responses.

### 3. No HTTPS Enforcement

Issue: The code does not enforce the use of HTTPS, so data (including sensitive information like passwords) is transmitted in plaintext over the network. This makes it vulnerable to interception through man-in-the-middle (MITM) attacks.

Where to Look:

Frontend (JavaScript):

**Affected Lines:** 50-57 in JavaScript: The fetch request sends data to the backend, and it's unclear if it uses HTTPS. The URL in the fetch call (/SportEventAppli/api/register) may be handled by a server that does not force HTTPS.

Solution:

Ensure that the application uses HTTPS and enforce it by redirecting HTTP requests to HTTPS using a reverse proxy or web server settings. Always include https:// in the API request URLs.

### 4. No CSRF Protection

Issue: Cross-Site Request Forgery (CSRF) can occur because the registration form sends a POST request without any CSRF token. This means that an attacker could trick a user into submitting a registration form without their consent.

Where to Look:

Frontend (HTML):

Line: 15 in HTML: The registration form (<form id="registerForm">) does not include a CSRF token.

Solution:

Add a hidden CSRF token input field to the form and verify it server-side with each request to prevent CSRF attacks.

## 2.8 Register HTML

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
6   <title>Register - SportEvent</title>
7   <!-- Bootstrap CSS for styling -->
8   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
9   <style>
10    /* Style for the registration form */
11    .form-signin {
12      max-width: 330px;
13      margin: auto;
14    }
15  </style>
16 </head>
17
18 <body class="text-center">
19   <!-- Registration form -->
20   <form class="form-signin" id="registerForm">
21     <h1 class="h3 mb-3 font-weight-normal">Create Account</h1>
22
23     <!-- First name input -->
24     <label for="first_name" class="sr-only">First Name</label>
25     <input type="text" id="first_name" name="first_name" class="form-control" placeholder="First Name" required autofocus>
26
27     <!-- Surname input -->
28     <label for="surname" class="sr-only">Surname</label>
29     <input type="text" id="surname" name="surname" class="form-control" placeholder="Surname" required>
30
31     <!-- Email address input -->
32     <label for="email" class="sr-only">Email address</label>
```

```
33     <input type="email" id="email" name="email" class="form-control" placeholder="Email address" required>
34
35     <!-- Password input -->
36     <label for="password" class="sr-only">Password</label>
37     <input type="password" id="password" name="password" class="form-control" placeholder="Password" required>
38
39     <!-- Submit button -->
40     <button class="btn btn-lg btn-primary btn-block mt-3" type="submit">Register</button>
41
42     <!-- Link to the login page for existing users -->
43     <p class="mt-2">Already have an account? <a href="Login.html">Login here</a></p>
44   </form>
45
46   <!-- JavaScript for form submission -->
47   <script>
48     // Handle form submission
49     document.getElementById('registerForm').onsubmit = async (e) => {
50       e.preventDefault(); // Prevent the default form submission
51
52       // Collect form data and encode it into a URL-encoded string
53       const formData = new URLSearchParams();
54       formData.append('first_name', document.getElementById('first_name').value);
55       formData.append('surname', document.getElementById('surname').value);
56       formData.append('email', document.getElementById('email').value);
57       formData.append('password', document.getElementById('password').value);
58
59       try {
```



```

59     try {
60         // Send a POST request with the form data
61         const response = await fetch('/SportEventAppli/api/register', {
62             method: 'POST',
63             headers: {
64                 'Content-Type': 'application/x-www-form-urlencoded' // Correct content type
65             },
66             body: formData.toString() // Convert the form data into a query string
67         });
68
69         // Handle successful registration
70         if (response.ok) {
71             alert("Registration successful! You can now log in.");
72             window.location.href = "/SportEventAppli/login.html"; // Redirect to login page
73         } else {
74             alert("Registration failed. Email may already be in use."); // Handle registration failure
75         }
76     } catch (error) {
77         console.error("Error during registration:", error);
78         alert("An error occurred while registering. Please try again."); // Handle errors
79     }
80 };
81 </script>
82 </body>

```

## 1. Cross-Site Scripting (XSS)

**Issue:** The HTML code does not sanitize user inputs before sending them to the backend. While the server should also sanitize inputs, it's always good practice to add client-side validation/sanitization.

Where to Look:

Frontend (HTML/JavaScript):

**Affected Lines 38-42:** The input fields for first\_name, surname, email, and password do not have any client-side validation or sanitization.

**Affected Lines : 48-52** in JavaScript: The fetch request sends data directly without any sanitization or escaping, which could potentially open the door for an XSS attack if the data is reflected back from the server.

## 2. Insufficient Input Validation (Frontend - JavaScript)

**Issue:** The form does not perform input validation on the client-side. For example:

The email field doesn't ensure the input matches a proper email format.

There is no validation to check if the password is strong enough (e.g., minimum length, character variety).

There is no check for empty fields before form submission.

Where to Look:

Frontend (HTML/JavaScript):

**Affected Lines 38-42:** Input fields for first\_name, surname, email, and password are not validated.

**Affected Lines 48-52** in JavaScript: The form data is sent without checking the content of the input fields for validity.

### 3. No CSRF Protection

**Issue:** There is no CSRF token included in the registration form, making it vulnerable to Cross-Site Request Forgery (CSRF) attacks. An attacker could trick a user into submitting a form on their behalf.

Where to Look:

Frontend (HTML):

**Affected Line: 15** The registration form does not contain a hidden CSRF token field to protect against CSRF attacks.

### 4. No HTTPS Enforcement

**Issue:** The form sends data over HTTP, which is insecure. If the user is on an insecure connection (HTTP instead of HTTPS), the data can be intercepted by attackers.

Where to Look:

Frontend (JavaScript):

**Affected Line: 50-57** The fetch request does not explicitly enforce HTTPS. If the server is not configured to use HTTPS, data can be exposed in transit.

### 5. Weak Password Handling

**Issue:** The password is sent to the server in plaintext, which is insecure. Attackers could intercept the password in transit if not using HTTPS, or if the server doesn't hash the password before storing it.

Where to Look:

Frontend (HTML/JavaScript):

**Affected Line: 42** The password is sent as plain text in the form submission.

Solution:

Never transmit passwords as plaintext. Ensure the use of HTTPS to encrypt data in transit. Additionally, the server should hash passwords before storing them in the database (this is part of the backend implementation).

#### 6. No Error Handling for Form Submission (JavaScript)

**Issue:** There is no detailed error handling if the user inputs invalid data or if the server fails to respond correctly. The user only sees an alert, but no indication of the problem.

Where to Look:

Frontend (JavaScript):

**Affected Lines: 50-57** The catch block handles errors, but it only shows a generic message ("An error occurred while registering"). The user would not know whether the error is from the client or server-side.

Solution:

Provide more detailed error messages based on the response status or error type. For example, display different messages if the email is already in use or if the server is unreachable.

### 3 Summary of Vulnerabilities in HTML/JavaScript:

Cross-Site Scripting (XSS)

Insufficient Input Validation

No CSRF Protection

No HTTPS Enforcement

Weak Password Handling

No Detailed Error Handling

