

DOCUMENTAZIONE DI PROGETTO

Università degli Studi di Bari Aldo Moro

CdLM in Sicurezza Informatica

Secure Web App

Corso: Sicurezza nelle Applicazioni

Docenti: Prof. Donato Malerba, Prof. Paolo Mignone

Studente: Giacomo Pagliara

Matricola: 799628

Sommario

Introduzione e Obiettivi di Progetto	3
Analisi Statica	3
1 - Cookie e gestione del token "remember me"	3
Implementazione sicura dei cookie	4
Gestione del token lato Server	5
Validazione del token	6
Pulizia automatica dei token scaduti	6
2 - Sessione HTTP e Autenticazione	7
Configurazione della sessione nel web.xml e Protezione HTTPS obbligatoria	7
Implementazione del filtro di autenticazione	8
Gestione sicura del timeout di sessione	10
Gestione del logout	11
3 - Caricamento file e validazione	12
4 – Prevenzione SQL Injection.....	16
Isolamento delle operazioni di database.....	19
5 – Chiavi e Crittografia AES	19
Implementazione AES nell'applicazione	20
Gestione sicura delle chiavi	20
Operazioni di cifratura e decifratura.....	21
Utilizzo di AES nell'applicazione.....	22
6 – Gestione sicura delle password.....	23
Implementazione nell'applicazione.....	23
Integrazione nel sistema di autenticazione	26
7 – Lifetime dei dati sensibili	26
8 - Thread-safety.....	28
Analisi Dinamica.....	31
Test d'uso	31
Test d'abuso	34

Introduzione e Obiettivi di Progetto

Il progetto “Secure Web App” richiedeva le seguenti funzionalità:

- Registrazione con immagine di profilo
 - L'applicazione consente di registrarsi con e-mail, password, retype password, foto profilo (ad esempio .png, .jpeg...)
- Login
 - L'applicazione consente di chiedere all'utente di voler effettuare il login con o senza il meccanismo dei Cookie
- Caricamento di una proposta progettuale in formato .txt
- Visualizzazione delle altre proposte progettuali all'interno della sessione https e/o della validità dei Cookie
- Logout
 - Invalida la sessione Http e i Cookie
- L'applicazione è inoltre in grado di fornire feedback all'utente dipendenti dal tipo di esito delle richieste (Registrazione OK, Registrazione KO, Immagine non valida, Errore login, Sessione scaduta...)

Analisi Statica

1 - Cookie e gestione del token "remember me"

I cookie sono piccoli file di testo che vengono salvati sul dispositivo dell'utente quando quest'ultimo visita un sito web. I cookie contengono informazioni che possono essere utilizzate dal sito per vari scopi, tra cui mantenere lo stato della sessione, oppure ricordare le preferenze dell'utente, ecc.

All'interno dell'app viene implementato un sistema di autenticazione persistente “remember me” che consente all'utente di mantenere l'accesso anche dopo la chiusura del browser, evitando così di memorizzare direttamente credenziali sensibili nei cookie.

Quando l'utente effettua il login può scegliere se premere sulla checkbox “Ricordami”, come si vede qui sotto nell'immagine.

Accedi

Nome utente:

Password:

☐ Ricordami

Accedi

Non hai un account? [Registrati](#)

Implementazione sicura dei cookie

Anziché memorizzare direttamente le credenziali all'interno dei cookie, perché potrebbe essere una pratica rischiosa se gestita male e senza approfondimenti, viene utilizzato un approccio più sicuro e in un certo senso “certo”, con token cifrati:

```
/**
 * Crea e aggiunge i cookie di autenticazione alla risposta.
 */
@param username Nome utente
@param response Risposta HTTP
*/
private void createAuthCookies(String username, HttpServletResponse response) {
    try {
        // Genera un token di autenticazione
        TokenManager.TokenResult tokenResult = TokenManager.generateRememberToken(username);
        if (tokenResult != null) {
            // Creiamo un cookie che contiene sia il token cifrato che l'UUID
            String encryptedToken = AesEncryption.encryptToBase64(tokenResult.getPlainToken());
            String cookieValue = tokenResult.getUuid() + ":" + encryptedToken;

            // Crea e configura il cookie
            Cookie rememberMeCookie = new Cookie(REMEMBER_TOKEN_COOKIE, cookieValue);
            rememberMeCookie.setHttpOnly(true);
            rememberMeCookie.setSecure(true);
            rememberMeCookie.setMaxAge(COOKIE_MAX_AGE); // 1 giorno in secondi

            // Aggiungi il cookie alla risposta
            response.addCookie(rememberMeCookie);

            System.out.println("Cookie di autenticazione creato con successo: " + tokenResult.getUuid());
        }
    } catch (Exception e) {
        System.out.println("Errore nella creazione del cookie di autenticazione: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Questa implementazione permette di implementare un approccio:

- **token-based:** Il token è un valore casuale generato tramite **SecureRandom**, quindi non correlato direttamente alle credenziali dell'utente;
- **Token cifrato:** Viene utilizzato l'**AES (Advanced Encryption Standard)** per cifrare il token prima di salvarlo nel cookie;
- **Identificatore Univoco:** Chiamato anche UUID, il cookie contiene sia il token cifrato che un UUID univoco, impedendo eventuali attacchi di clonazione (*si riferiscono a diverse tecniche con cui un aggressore cerca di duplicare informazioni o risorse, con l'obiettivo di ingannare le vittime o ottenere accesso non autorizzato a dati sensibili*);
- **Attributi di sicurezza:**
 - **HttpOnly:** Previene l'accesso al cookie tramite JavaScript, proteggendo contro attacchi XSS;
 - **Secure:** Limita la trasmissione del cookie alle sole connessioni HTTPS;
 - **MaxAge:** Imposta un timeout di 24 ore sul cookie.

Questa struttura contiene:

- L'UUID del token (non cifrato) per l'identificazione rapida all'interno del database
- Il token stesso, cifrato con AES, per la verifica dell'autenticità

La combinazione di UUID e token cifrato previene attacchi di manipolazione dei cookie e tentativi di indovinare i token.

Gestione del token lato Server

I token sono memorizzati in una tabella dedicata nel database con le seguenti caratteristiche:

- Username associato al token;
- Il token cifrato;
- UUID univoco per il token;
- Data di scadenza (expiry_date);
- Data di creazione (created_at);

Questa struttura consente:

- Di invalidare i token scaduti automaticamente grazie alla data di scadenza;
- Identificare e validare i token in modo sicuro;
- Mantenere un registro delle sessioni attive.

Validazione del token

Nel momento in cui un utente ritorna sul sito, il sistema verifica automaticamente la presenza e la validità del cookie di autenticazione:

```
public static String validateToken(String encryptedToken, String uuid) {
    if (encryptedToken == null || encryptedToken.isEmpty() || uuid == null || uuid.isEmpty()) {
        return null;
    }

    Connection connection = null;
    try {
        connection = DatabaseConnection.getConnectionRead();

        // Query modificata per cercare per UUID
        String query = DatabaseQueries.getCheckRememberTokenQuery();
        try (PreparedStatement stmt = connection.prepareStatement(query)) {
            stmt.setString(1, uuid);
            try (ResultSet resultSet = stmt.executeQuery()) {
                if (resultSet.next()) {
                    String username = resultSet.getString("username");
                    String storedEncryptedToken = resultSet.getString("token");

                    try {
                        // Decrypta entrambi i token
                        String plainStoredToken = AesEncryption.decryptFromBase64(storedEncryptedToken);
                        String plainCookieToken = AesEncryption.decryptFromBase64(encryptedToken);

                        // Confronta i token in chiaro
                        if (plainStoredToken.equals(plainCookieToken)) {
                            return username;
                        }
                    } catch (Exception e) {
                        System.out.println("Errore nella decriptazione dei token: " + e.getMessage());
                    }
                }
            }
        }
    } catch (SQLException e) {
        MessageUtils.showErrorMessage("Errore durante la validazione del token");
        e.printStackTrace();
    } finally {
        closeConnection(connection);
    }

    return null;
}
```

Questo processo di validazione permette di:

- Verificare l'esistenza dell'UUID, usando quest'ultimo come identificatore, all'interno del database;
- Controllare che il token non sia scaduto grazie a "getCheckRememberTokenQuery()" che contiene (expiry_date > NOW());
- Decifrare il token memorizzato e decifrare il token presente nel cookie per confrontarli per poter autenticare l'utente;
- Infine, restituire lo username solo se tutti i controlli sono passati, permettendo così all'utente di poter riaccedere alla propria "dashboard".

Pulizia automatica dei token scaduti

È stato implementato un meccanismo di pulizia automatica dei token scaduti tramite un listener di contesto "ServletContextListener" (Un ServletContextListener è un tipo di "listener" in Java Servlet che permette di reagire agli eventi di ciclo di vita del ServletContext. In

pratica, consente di eseguire codice quando l'applicazione web viene avviata, arrestata o si verificano altri eventi a livello di applicazione. Questo è utile per attività come la configurazione iniziale, la gestione delle risorse o la pulizia all'arresto dell'applicazione.):

```
/**
 * Listener per eseguire pulizia periodica dei token scaduti.
 */
@WebListener
public class TokenCleanupListener implements ServletContextListener {

    private ScheduledExecutorService scheduler;

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        scheduler = Executors.newSingleThreadScheduledExecutor();

        // Pianifica la pulizia dei token scaduti ogni 24 ore
        scheduler.scheduleAtFixedRate(new Runnable() {
            @Override
            public void run() {
                TokenManager.cleanExpiredTokens();
            }
        }, 0, 24, TimeUnit.HOURS);
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        if (scheduler != null) {
            scheduler.shutdownNow();
        }
    }
}
```

Questo sistema garantisce che i token scaduti vengano rimossi periodicamente dal database, ogni 24 ore, riducendo così il rischio di potenziali attacchi e soprattutto per mantenere il database efficiente.

2 - Sessione HTTP e Autenticazione

Le sessioni http rappresentano un meccanismo fondamentale per mantenere lo stato tra diverse richieste di un utente al server web. A differenza dei cookie, i dati di sessione vengono memorizzati sul server anziché sul browser dell'utente, offrendo un livello superiore di sicurezza per le informazioni sensibili. È necessario, dunque, optare per una corretta implementazione dei meccanismi di sessione e autenticazione per prevenire attacchi come *session hijacking* e *session fixation*.

Configurazione della sessione nel web.xml e Protezione HTTPS obbligatoria

La configurazione del file **web.xml** implementa diverse misure di sicurezza:

```
<!-- Timeout della sessione a 15 minuti per maggiore sicurezza -->
<session-config>
  <session-timeout>15</session-timeout>
</session-config>
```

Il timeout di 15 minuti garantisce che le sessioni inattive vengano terminate automaticamente andando a ridurre la finestra temporale durante la quale un attaccante potrebbe sfruttare una sessione abbandonata. La scelta a 15 minuti è una scelta che cerca di creare un compromesso tra sicurezza e usabilità, offrendo un tempo sufficiente all'utente per poter completare varie operazioni ordinarie senza dover effettuare nuovamente l'accesso molto spesso.

Inoltre, tutte le comunicazioni relative alle sessioni sono protette tramite HTTPS, come definito sempre nel file web.xml:

```
<!-- Forzatura HTTPS per l'intera applicazione -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>HTTPSOnly</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Infatti, è importante perché viene offerta protezione contro l'intercettazione dei cookie di sessione, previene attacchi man-in-the-middle, va a garantire l'integrità dei dati trasmessi ed è conforme alle best practice di sicurezza moderne. Nel nostro caso, infatti, l'utilizzo di HTTPS è fondamentale per proteggere l'ID di sessione durante la trasmissione, impedendo gli attacchi citati poc'anzi.

Implementazione del filtro di autenticazione

Il controllo dell'accesso alle risorse protette viene implementato, all'interno dell'app, attraverso un filtro di autenticazione “*AuthenticationFilter.java*” che protegge le pagine che richiedono l'autenticazione:

```
/**
 * Filtro di autenticazione migliorato con debug approfondito
 */
@WebFilter(urlPatterns = {"/benvenuto.jsp", "/progetti.jsp"})
public class AuthenticationFilter implements Filter {
```

Il seguente filtro permette di andare a verificare la presenza di una sessione http valida. Viene poi controllato se l'utente è autenticato tramite l'attributo di sessione, inoltre c'è

la verifica della presenza di un cookie “rememberToken” valido in assenza di sessione e poi c’è il reindirizzamento al login degli utenti non autenticati.

Il seguente filtro implementa una strategia di autenticazione a due livelli:

1. Verifica primaria attraverso la sessione http;
2. Verifica secondaria attraverso il token “ricordami” cifrato

```
// Verifica se l'utente è già autenticato
HttpSession session = httpRequest.getSession(false);
boolean isLoggedIn = (session != null && session.getAttribute("login") != null &&
    (Boolean) session.getAttribute("login"));

Logger.info("Stato sessione: " + (session != null ? "Esistente" : "Non esistente"));
Logger.info("Stato login: " + (isLoggedIn ? "Loggato" : "Non loggato"));

if (isLoggedIn) {
    // Utente già autenticato, prosegui
    Logger.info("Utente autenticato via sessione, accesso consentito");
    chain.doFilter(request, response);
    return;
}

// Verifica se c'è un cookie rememberToken
String cookieValue = null;
Cookie[] cookies = httpRequest.getCookies();

if (cookies != null) {
    for (Cookie cookie : cookies) {
        if ("rememberToken".equals(cookie.getName())) {
            cookieValue = cookie.getValue();
            Logger.info("Cookie rememberToken trovato: " + cookieValue);
            break;
        }
    }
}

if (cookieValue != null) {
    // Estrai UUID e token dal valore del cookie
    String[] parts = cookieValue.split(":", 2);
    if (parts.length != 2) {
        Logger.warning("Formato cookie non valido: " + cookieValue);
        removeCookie(httpResponse);
        httpResponse.sendRedirect(httpRequest.getContextPath() + "/login.jsp?unauthorized=true");
        return;
    }

    String uuid = parts[0];
    String encryptedToken = parts[1];
}
```

```

// Verifica il token |
String username = TokenManager.validateToken(encryptedToken, uuid);
Logger.info("Validazione token per username: " + username);

if (username != null) {
    // Token valido, crea una nuova sessione
    HttpSession newSession = httpRequest.getSession(true);
    newSession.setAttribute("login", true);
    newSession.setAttribute("nomeUtente", username);
    newSession.setMaxInactiveInterval(15*60); // 15 minuti

    Logger.info("Autenticazione via token riuscita, nuova sessione creata per: " + username);

    // Continua con la richiesta
    chain.doFilter(request, response);
    return;
} else {
    Logger.info("Token non valido, rimuovo il cookie");
    // Token non valido, rimuovi il cookie
    removeCookie(httpResponse);
}
} else {
    Logger.info("Nessun cookie rememberToken trovato");
}

// Nessuna autenticazione valida, reindirizza al login
Logger.warning("ACCESSO NEGATO: reindirizzamento a login.jsp");
httpResponse.sendRedirect(httpRequest.getContextPath() + "/login.jsp?unauthorized=true");
}

/**
 * Rimuove il cookie rememberToken
 *
 * @param response Risposta HTTP
 */
private void removeCookie(HttpServletResponse response) {
    Cookie cookie = new Cookie("rememberToken", "");
    cookie.setMaxAge(0);
    cookie.setPath("/");
    response.addCookie(cookie);
}

```

Gestione sicura del timeout di sessione

Oltre alla configurazione lato server all'interno del file web.xml che abbiamo descritto poc'anzi, la gestione del timeout di sessione avviene anche lato client attraverso JavaScript nelle pagine protette (benvenuto.jsp e progetti.jsp):

```

<script>
    // Timeout di 15 minuti in millisecondi
    var sessionTimeoutMillis = 900000;

    // Funzione da eseguire quando la sessione scade
    function sessionExpired() {
        // Chiamata sincrona al LogoutServlet per assicurarsi che la sessione sia invalidata
        // prima di procedere con l'alert e il reindirizzamento
        var xhr = new XMLHttpRequest();
        xhr.open('GET', 'LogoutServlet?timeout=true', false); // Sincrona!
        xhr.send();

        // Mostra l'alert
        alert("La tua sessione è scaduta per inattività.");

        // Reindirizza alla pagina di login
        window.location.href = 'login.jsp?timeout=true';
    }

    // Imposta il timer di timeout
    var sessionTimer = setTimeout(sessionExpired, sessionTimeoutMillis);

    // Resetta il timer quando l'utente interagisce con la pagina
    function resetTimer() {
        clearTimeout(sessionTimer);
        sessionTimer = setTimeout(sessionExpired, sessionTimeoutMillis);
    }
</script>

```

Questo doppio approccio garantisce che, se il server termina la sessione, al successivo tentativo di accesso l'utente sarà reindirizzato al login mentre se il client rileva l'inattività prima del server, l'utente riceve una notifica sul web e viene reindirizzato.

Gestione del logout

Nella nostra app, la servlet di logout “LogoutServlet.java” implementa meccanismi di sicurezza per terminare correttamente la sessione dell’utente:

```
@WebServlet("/LogoutServlet")
public class LogoutServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    // Pagina di reindirizzamento dopo il logout
    private static final String LOGIN_PAGE = "login.jsp";

    /**
     * Gestisce le richieste GET.
     * Esegue il logout dell'utente invalidando la sessione e i cookie.
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Verifica se è una richiesta di timeout
        boolean isTimeout = request.getParameter("timeout") != null &&
            request.getParameter("timeout").equals("true");

        // Ottieni informazioni sulla sessione e username
        HttpSession session = request.getSession(false);
        String username = null;

        if (session != null) {
            username = (String) session.getAttribute("nomeUtente");

            // Invalida la sessione
            session.invalidate();
            System.out.println("Sessione invalidata");
        }

        if (!isTimeout) {
            // Caso di logout normale (non timeout)

            // Rimuovi i cookie
            removeCookies(request, response);

            // Elimina i token associati all'utente
            if (username != null) {
                TokenManager.deleteTokensByUsername(username);
                System.out.println("Token eliminati per l'utente: " + username);
            }

            // Notifica all'utente
            MessageUtils.showInfoMessage("Logout effettuato con successo!");

            // Reindirizza alla pagina di login
            response.sendRedirect(LOGIN_PAGE);
        } else {
            // Caso di timeout di sessione

            // Verifica se l'utente aveva un cookie rememberToken
            boolean hasRememberToken = false;
            Cookie[] cookies = request.getCookies();
            if (cookies != null) {
                for (Cookie cookie : cookies) {
                    if ("rememberToken".equals(cookie.getName())) {
                        hasRememberToken = true;
                        break;
                    }
                }
            }

            // Se NON aveva il cookie di rememberToken, tratta come un logout comple
            if (!hasRememberToken && username != null) {
                // Rimuovi i cookie comunque per sicurezza
                removeCookies(request, response);

                // Elimina i token associati all'utente
                TokenManager.deleteTokensByUsername(username);
                System.out.println("Token eliminati per l'utente: " + username);
            }

            // Invia solo un codice di successo, il resto è gestito via JavaScript
            response.setStatus(HttpServletResponse.SC_OK);
        }
    }
}
```

Le misure di sicurezza implementate includono che ci sia:

- Un’invalidazione della sessione http esistente;

- Una rimozione di tutti i cookie, ovviamente con particolare attenzione al cookie “rememberToken”;
- Una eliminazione dei token di autenticazione associati, dal database;
- Una gestione separata del timeout di sessione e del logout esplicito.

Durante il logout, infatti, i cookie vengono invalidati con metodi sicuri:

```
private void removeCookies(HttpServletRequest request, HttpServletResponse response) {
    Cookie[] cookies = request.getCookies();

    if (cookies != null) {
        for (Cookie cookie : cookies) {
            // Per il cookie rememberToken, eliminalo anche dal database
            if ("rememberToken".equals(cookie.getName())) {
                String cookieValue = cookie.getValue();
                if (cookieValue != null && !cookieValue.isEmpty()) {
                    String[] parts = cookieValue.split(":", 2);
                    if (parts.length == 2) {
                        String uuid = parts[0];
                        TokenManager.deleteTokenByUuid(uuid);
                        System.out.println("Token eliminato con UUID: " + uuid);
                    }
                }
            }

            // IMPORTANTE: Crea un nuovo cookie con gli stessi attributi
            Cookie killCookie = new Cookie("rememberToken", "");
            killCookie.setMaxAge(0);
            killCookie.setPath("/"); // Usa lo stesso path del cookie originale

            // Aggiungi gli stessi attributi usati in LoginServlet
            killCookie.setHttpOnly(true);
            killCookie.setSecure(true);

            // Se l'applicazione usa un context path
            if (request.getContextPath() != null && !request.getContextPath().isEmpty()) {
                killCookie.setPath(request.getContextPath());
            }

            response.addCookie(killCookie);
            System.out.println("Cookie rememberToken rimosso con parametri completi");
        }
    } else {
        // Per altri cookie, usa il metodo standard
        cookie.setPath("/");
        cookie.setMaxAge(0);
        response.addCookie(cookie);
    }
}
```

L’approccio utilizzato garantisce che la sessione non possa essere riutilizzata dopo il logout, che i token “ricordami” vengano invalidati sia nel browser che nel database e che nessun dato di autenticazione persista dopo il logout.

3 - Caricamento file e validazione

Il caricamento di file potrebbe rappresentare una funzionalità rischiosa in un’applicazione web poiché può essere un vettore per numerosi attacchi. All’interno della nostra app vengono gestiti due tipi di upload: immagini di profilo durante la registrazione e proposte progettuali in formato testo “.txt”

Nell’app è stata implementata una strategia di validazione a più livelli:

1. **Validazione dell’estensione del file:** Si va a verificare che l’estensione corrisponda ai formati consentiti;
2. **Controllo della dimensione:** Viene limitata la dimensione massima;

3. **Verifica del contenuto reale:** Viene effettuato con Apache Tika, si va a identificare il tipo MIME effettivo del file, indipendentemente dall'estensione.
4. **Sanitizzazione del contenuto:** Questo avviene per i file di testo delle proposte progettuali dove vengono rimossi i contenuti potenzialmente dannosi come gli script di JavaScript.

Validazione delle immagini di profilo:

```
// Costanti per i controlli sui file
private static final long MAX_IMAGE_SIZE = 5 * 1024 * 1024; // 5 MB
private static final Set<String> ALLOWED_IMAGE_EXTENSIONS = new HashSet<>(
    Arrays.asList("jpeg", "jpg", "png"));
private static final String IMAGE_MIME_PREFIX = "image/";

/**
 * Verifica se un file caricato è un'immagine valida (jpg, jpeg, png).
 * Controlla estensione, dimensione e tipo MIME del file.
 *
 * @param filePart Parte del file caricato tramite multipart/form-data
 * @return true se il file è valido, false altrimenti
 * @throws IOException Se si verifica un errore durante la lettura del file
 */
public static boolean isValidImageFile(Part filePart) throws IOException {
    // Controllo se il file esiste e non è vuoto
    if (filePart == null || filePart.getSize() <= 0) {
        MessageUtils.showErrorMessage("Nessun file caricato");
        return false;
    }

    String fileName = Paths.get(filePart.getSubmittedFileName()).getFileName().toString();
    String fileExtension = fileName.substring(fileName.lastIndexOf(".") + 1).toLowerCase();

    // Controllo dell'estensione del file
    if (!ALLOWED_IMAGE_EXTENSIONS.contains(fileExtension)) {
        MessageUtils.showErrorMessage("Estensione del file non supportata. Formati accettati: jpg, jpeg, png");
        return false;
    }

    // Controllo della dimensione del file
    if (filePart.getSize() > MAX_IMAGE_SIZE) {
        MessageUtils.showErrorMessage("L'immagine selezionata supera la dimensione massima consentita di 5 MB");
        return false;
    }

    // Controllo del tipo MIME usando Apache Tika
    Tika tika = new Tika();
    String contentType = tika.detect(filePart.getInputStream());

    if (contentType == null || !contentType.startsWith(IMAGE_MIME_PREFIX)) {
        MessageUtils.showErrorMessage("Il file non è un'immagine valida");
        return false;
    }
}
```

Questo metodo va a implementare il controllo delle estensioni permesse tramite whitelist (jpeg, jpg, png), va a verificare che la dimensioni non superi il limite impostato (5 MB) e poi viene utilizzato Apache Tika per determinare il tipo reale del file, impedendo attacchi di MIME-spoofing.

Validazione delle proposte progettuali:

```

public static boolean isValidTextFile(Part filePart) throws IOException {
    // Implementazione per i file di testo (per la parte delle proposte progettuali)
    // Simile alla validazione dell'immagine ma per file .txt

    if (filePart == null || filePart.getSize() <= 0) {
        MessageUtils.showErrorMessage("Nessun file caricato");
        return false;
    }

    String fileName = Paths.get(filePart.getSubmittedFileName()).getFileName().toString();
    String fileExtension = fileName.substring(fileName.lastIndexOf(".") + 1).toLowerCase();

    if (!"txt".equals(fileExtension)) {
        MessageUtils.showErrorMessage("Il file deve essere in formato .txt");
        return false;
    }

    // Controllo del tipo MIME
    Tika tika = new Tika();
    String contentType = tika.detect(filePart.getInputStream());

    if (contentType == null || !contentType.contains("text/plain")) {
        MessageUtils.showErrorMessage("Il file non è un documento di testo valido");
        return false;
    }

    return true;
}
}

```

Per le proposte progettuali, oltre ai controlli base, viene implementata una sanitizzazione del contenuto HTML, per prevenire Cross-Site Scripting (XSS)

```

public static String processFileContent(Part filePart) {
    try {
        // Verifica dimensione file
        if (filePart.getSize() > MAX_FILE_SIZE) {
            MessageUtils.showErrorMessage(
                "Il file supera la dimensione massima consentita. Il file può essere massimo di 20 MB");
            return null;
        }

        // Verifica tipo MIME
        Tika tika = new Tika();
        String contentType = tika.detect(filePart.getInputStream());

        if (!TEXT_MIME_TYPE.equals(contentType) && !HTML_MIME_TYPE.equals(contentType)) {
            MessageUtils.showErrorMessage("Il file contiene del testo non valido.");
            return null;
        }

        // Leggi il contenuto del file
        String content = readFileContent(filePart);

        // Sanitizza il contenuto HTML
        return sanitizeHtml(content);
    } catch (IOException e) {
        e.printStackTrace();
        MessageUtils.showErrorMessage("C'è stato un problema con il caricamento del file.");
        return null;
    }
}
}

```

```

private static String sanitizeHtml(String content) {

    // Usa Jsoup per sanitizzare l'HTML
    Document document = Jsoup.parse(content);

    // Rimuovi elementi pericolosi
    document.select("script, [type=application/javascript], [type=text/javascript]").remove();

    // Rimuovi attributi pericolosi
    document.select("[onclick], [onload], [onerror], [onfocus], [onblur]").forEach(element -> {
        element.removeAttr("onclick");
        element.removeAttr("onload");
        element.removeAttr("onerror");
        element.removeAttr("onfocus");
        element.removeAttr("onblur");
    });

    // Pulisci ulteriormente con Safelist
    String cleanedHtml = Jsoup.clean(document.body().html(), Safelist.relaxed());

    return cleanedHtml;
}

```

Viene limitata la dimensione massima dei file a 20 MB, viene verificato che il contenuto sia effettivamente testo mediante Apache Tika e infine viene sanitizzato il contenuto HTML con Jsoup che rimuove completamente i tag <script> e altri tag Javascript, va a eliminare attributi potenzialmente pericolosi come “onclick” e “onload” e soprattutto applica una whitelist di tag HTML consentiti tramite “Safelist.relaxed()”.

La gestione dell’**upload dei file** lato servlet avviene tramite “ProjectServlet.java”, quest’ultima gestisce il caricamento dei file:

```

@WebServlet("/ProjectServlet")
@MultipartConfig
public class ProjectServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    // Costanti per i parametri delle richieste
    private static final String PROJECT_FILE_PARAM = "Proposta progettuale";
    private static final String USERNAME_PARAM = "nomeUtente";
}

```

```

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    System.out.println("Nome utente ricevuto: " + request.getParameter("nomeUtente"));
    System.out.println("File ricevuto: " + (request.getPart("Proposta progettuale") != null));

    Part filePart = request.getPart(PROJECT_FILE_PARAM);
    String username = request.getParameter(USERNAME_PARAM);

    // Controllo input
    if (username == null || username.trim().isEmpty()) {
        sendErrorResponse(response, HttpServletResponse.SC_BAD_REQUEST, "Nome utente non specificato");
        return;
    }

    // Validazione file
    if (!ProjectFileValidator.isValidProjectFile(filePart, getServletContext())) {
        sendErrorResponse(response, HttpServletResponse.SC_BAD_REQUEST, "File non valido");
        return;
    }

    // Processa il contenuto del file
    String sanitizedHtml = ProjectFileValidator.processFileContent(filePart);
    if (sanitizedHtml == null) {
        sendErrorResponse(response, HttpServletResponse.SC_BAD_REQUEST, "Impossibile processare il file");
        return;
    }

    String fileName = ProjectFileValidator.getFileName(filePart);
    byte[] htmlBytes = sanitizedHtml.getBytes(StandardCharsets.UTF_8);

    try {
        // Carica il file nel database
        if (ProjectDao.uploadProject(username, htmlBytes, fileName)) {
            MessageUtils.showInfoMessage("La proposta è stata correttamente caricata!");

            // Reindirizza alla pagina di visualizzazione delle proposte
            response.sendRedirect("progetti.jsp?action=view&success=true");
        } else {
            sendErrorResponse(response, HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
                "Non è stato possibile caricare il file della proposta");
        }
    }
}

```

C'è l'annotazione “@MultipartConfig” per il corretto processing dei dati multipart/form-data, i parametri di input vengono validati, viene sanitizzato il contenuto prima del salvataggio nel database.

4 – Prevenzione SQL Injection

Tutte le query effettuate all'interno dell'app per recuperare ed inserire i dati del/nel database sono gestite mediante Prepared Statement. La principale ragione di tale utilizzo è la prevenzione da attacchi di tipo SQL Injection; infatti, le Prepared Statement forniscono un meccanismo sicuro per inserire i parametri nelle query SQL. Invece di concatenare direttamente i valori degli input nelle query, vengono utilizzati dei segnaposto (“?”) successivamente sostituiti con valori validi in modo sicuro.

All'interno di **DatabaseQueries.java** c'è la centralizzazione delle query in SQL, qui sotto si può vedere quella relativa al Login:

```

public class DatabaseQueries {
    // Costanti per i nomi delle proprietà
    private static final String USER_LOGIN_QUERY = "db.query_userLogin";
}

```



```

/**
 * Ottiene la query per il login dell'utente.
 *
 * @return Query SQL per il login
 */
public static String getLoginQuery() {
    return getQueryProperty(USER_LOGIN_QUERY);
}

```

Sempre all'interno dello stesso file c'è il metodo che permette di recuperare una query specifica dal file di configurazione

```

/**
 * Recupera una query specifica dal file di configurazione.
 *
 * @param propertyName Nome della proprietà da recuperare
 * @return Valore della proprietà o null in caso di errore
 */
private static String getQueryProperty(String propertyName) {
    try {
        Properties properties = ConfigManager.getProperties();
        return properties.getProperty(propertyName);
    } catch (IOException e) {
        System.err.println("Errore nel recupero della query " + propertyName);
        e.printStackTrace();
        return null;
    }
}
}

```

La query viene letta dal file “config.ini” che viene caricato all'interno della classe “ConfigManager.java”

```

public class ConfigManager {
    // Percorso del file di configurazione
    private static final String CONFIG_FILE = "config.ini";
    // Proprietà caricate dal file
    private static Properties properties = null;

    /**
     * Ottiene le proprietà di configurazione, caricandole se necessario.
     *
     * @return Oggetto Properties con le configurazioni
     * @throws IOException Se si verifica un errore durante il caricamento
     */
    public static synchronized Properties getProperties() throws IOException {
        if (properties == null) {
            properties = loadProperties();
        }
        return properties;
    }

    /**
     * Carica le proprietà dal file di configurazione.
     *
     * @return Oggetto Properties con le configurazioni caricate
     * @throws IOException Se si verifica un errore durante il caricamento
     */
    private static Properties loadProperties() throws IOException {
        Properties props = new Properties();

        try (InputStream input = Thread.currentThread().getContextClassLoader()
            .getResourceAsStream(CONFIG_FILE)) {

            if (input == null) {
                throw new IOException("File di configurazione " + CONFIG_FILE + " non trovato");
            }

            props.load(input);
            return props;
        }
    }
}

```

Ritornando al Login, nel file **AuthDao.java**, come si può vedere qui sotto vengono verificate le credenziali dell'utente nel Database. Viene creato il Prepared Statement che richiama il metodo "getLoginQuery" dove vengono passati i parametri username e hashedPassword necessari per poter eseguire la query.

Come si può vedere c'è anche il Binding sicuro dei parametri volto a prevenire l'escape dei valori

```

/**
 * Verifica le credenziali dell'utente nel database.
 *
 * @param username Nome utente
 * @param hashedPassword Password hashata
 * @param connection Connessione al database
 * @return true se le credenziali sono valide, false altrimenti
 * @throws SQLException Se si verifica un errore SQL
 */
private static boolean checkCredentials(String username, byte[] hashedPassword, Connection connection)
    throws SQLException {
    try (PreparedStatement stmt = connection.prepareStatement(DatabaseQueries.getLoginQuery())) {
        stmt.setString(1, username);
        stmt.setBytes(2, hashedPassword);

        try (ResultSet resultSet = stmt.executeQuery()) {
            return resultSet.next();
        }
    }
}

```

Qui sotto si può vedere la query del Login che si trova all'interno del file config.ini:

```

# Query SQL
db.query_userLogin=SELECT * FROM users WHERE username=? AND password=?

```

Isolamento delle operazioni di database

L'app implementa il principio del minimo privilegio attraverso:

- **Connessioni separate per lettura e scrittura in *DatabaseConnection.java*:**

```

public static Connection getConnectionRead() throws SQLException {
    return DriverManager.getConnection(DATABASE_URL, READ_USERNAME, READ_PASSWORD);
}

public static Connection getConnectionWrite() throws SQLException {
    return DriverManager.getConnection(DATABASE_URL, WRITE_USERNAME, WRITE_PASSWORD);
}
}

```

- **Separazione dei privilegi utente nel database:**

- Utente “read” con permessi minimi di sola lettura;
- Utente “write” con permessi solo sulle tabelle necessarie

```

# Configurazione Database
db.url=jdbc:mysql://localhost:3306/webapp_pagliara?serverTimezone=UTC&useSSL=true&requireSSL=true
db.username_read=pagliara_read
db.password_read=R3ad#Usr$72!9pL
db.username_write=pagliara_write
db.password_write=Writ3#U$r@93!5qT

```

5 – Chiavi e Crittografia AES

La crittografia serve a proteggere informazioni sensibili attraverso la trasformazione dei dati in un formato illeggibile, noto come “testo cifrato”, che può essere decifrato solo da chi possiede la chiave di decrittazione corrispondente.

All'interno dell'app viene utilizzato l'Algoritmo di crittografia simmetrica AES (Advanced Encryption Standard) secondo la modalità CBC (Cipher Block Chaining) che utilizza un vettore di inizializzazione (IV) e collega i blocchi.

Le caratteristiche principali dell'AES sono:

- Algoritmo a chiave simmetrica (stessa chiave per cifrare e decifrare);
- Opera su blocchi di dati di dimensione fissa (128 bit);
- Supporta chiavi di lunghezza 128, 192 o 256 bit;
- È considerato sicuro contro attacchi di forza bruta con la tecnologia attuale.

La gestione sicura delle chiavi e dei vettori di inizializzazione è fondamentale per garantire la sicurezza della crittografia implementata da AES.

Implementazione AES nell'applicazione

L'app implementa la crittografia AES tramite la classe “**AesEncryption.java**”:

```
public class AesEncryption {  
    // Costanti per l'algoritmo e i parametri  
    private static final String ALGORITHM = "AES/CBC/PKCS5Padding";  
    private static final String KEY_SPEC = "AES";  
    private static final int IV_SIZE = 16; // 128 bit  
  
    private static SecretKey secretKey;  
    private static String ivString;
```

L'implementazione utilizza:

- **Algoritmo:** AES/CBC/PKCS5Padding
 - AES come cifrario;
 - CBC (Cipher Block Chaining) come modalità operativa, che richiede un vettore di inizializzazione (IV);
 - PKCS5Padding come schema di riempimento per blocchi incompleti.
- **Dimensione IV:** 16 byte (128 bit), conforme alle specifiche AES
- **Chiave segreta:** Caricata dinamicamente dalle configurazioni dell'applicazione dal file config.ini.

Gestione sicura delle chiavi

Infatti, le chiavi crittografiche sono gestite in modo sicuro:

```

private static void initialize() throws Exception {
    if (secretKey == null) {
        String aesKey = ConfigManager.getProperty("aes.key");
        if (aesKey == null || aesKey.isEmpty()) {
            throw new IllegalStateException("Chiave AES non trovata nelle configurazioni");
        }
        secretKey = new SecretKeySpec(Base64.getDecoder().decode(aesKey), KEY_SPEC);
    }

    if (ivString == null) {
        // Ottieni l'IV dalla configurazione o genera un nuovo IV sicuro
        ivString = ConfigManager.getProperty("aes.iv");
        if (ivString == null || ivString.isEmpty()) {
            // Se non è configurato, genera un IV casuale sicuro
            byte[] iv = new byte[IV_SIZE];
            new SecureRandom().nextBytes(iv);
            ivString = Base64.getEncoder().encodeToString(iv);
        }
    }
}
}

```

```

# Configurazione AES
aes.key=E5Vy1X[REDACTED]

```

Operazioni di cifratura e decifratura

Le funzioni di cifratura e decifratura sono così definite:

```

/**
 * Cripa un array di byte usando AES.
 *
 * @param data Dati da criptare
 * @return Dati criptati
 * @throws Exception Se si verifica un errore durante la crittografia
 */
public static byte[] encrypt(byte[] data) throws Exception {
    if (data == null) {
        return null;
    }

    initialize();

    Cipher cipher = Cipher.getInstance(ALGORITHM);
    cipher.init(Cipher.ENCRYPT_MODE, secretKey,
        new IvParameterSpec(Base64.getDecoder().decode(ivString)));
    return cipher.doFinal(data);
}

/**
 * Decripa un array di byte usando AES.
 *
 * @param encryptedBytes Dati criptati
 * @return Dati deciptati
 * @throws Exception Se si verifica un errore durante la decrittografia
 */
public static byte[] decrypt(byte[] encryptedBytes) throws Exception {
    if (encryptedBytes == null) {
        return null;
    }

    initialize();

    Cipher cipher = Cipher.getInstance(ALGORITHM);
    cipher.init(Cipher.DECRYPT_MODE, secretKey,
        new IvParameterSpec(Base64.getDecoder().decode(ivString)));
    return cipher.doFinal(encryptedBytes);
}

```

Utilizzo di AES nell'applicazione

La crittografia AES, nell'app, è utilizzata principalmente per proteggere i token di autenticazione persistente attraverso la cifratura e decifratura di quest'ultimi:

1. Cifratura dei token "remember me":

```
// Creiamo un cookie che contiene sia il token cifrato che l'UUID
String encryptedToken = AesEncryption.encryptToBase64(tokenResult.getPlainToken());
String cookieValue = tokenResult.getUuid() + ":" + encryptedToken;
```

In questo modo anche se il database venisse compromesso, i token cifrati non potrebbero essere utilizzati senza la chiave AES. Inoltre, il sistema mantiene la capacità di verificare i token in modo sicuro.

2. Decifratura durante la validazione del token:

```
try {
    // Decrypta entrambi i token
    String plainStoredToken = AesEncryption.decryptFromBase64(storedEncryptedToken);
    String plainCookieToken = AesEncryption.decryptFromBase64(encryptedToken);

    // Confronta i token in chiaro
    if (plainStoredToken.equals(plainCookieToken)) {
        return username;
    }
}
```

Poi c'è l'Encoding Base64 per rappresentare in modo sicuro i dati binari cifrati:

```
/* Cripta una stringa e la restituisce come stringa Base64.
 *
 * @param data Stringa da criptare
 * @return Stringa criptata codificata in Base64
 * @throws Exception Se si verifica un errore durante la crittografia
 */
public static String encryptToBase64(String data) throws Exception {
    if (data == null) {
        return null;
    }
    byte[] encrypted = encrypt(data.getBytes());
    return Base64.getEncoder().encodeToString(encrypted);
}

/**
 * Decrypta una stringa Base64 e la restituisce come testo normale.
 *
 * @param encryptedBase64 Stringa Base64 criptata
 * @return Stringa decryptata
 * @throws Exception Se si verifica un errore durante la decrittografia
 */
public static String decryptFromBase64(String encryptedBase64) throws Exception {
    if (encryptedBase64 == null) {
        return null;
    }
    byte[] decodedBytes = Base64.getDecoder().decode(encryptedBase64);
    byte[] decrypted = decrypt(decodedBytes);
    return new String(decrypted);
}
```

Queste funzioni svolgono un ruolo fondamentale nella gestione e trasformazione dei dati cifrati.

encryptToBase64 -> Trasformazione dati-testo per la trasmissione sicura: Questo metodo fornisce una modalità per permettere di cifrare dati testuali e ottenere un risultato in formato stringa che può essere facilmente memorizzato all'interno di database, inserito in cookie http, trasmesso in JSON o XML e incluso in URL o altri contesti che richiedono stringhe.

decryptFromBase64 -> Ripristino dei dati originali: Questo metodo permette di recuperare i dati originali da una stringa Base64 cifrata, completando il ciclo di vita dei dati protetti.

L'**encoding Base64** è fondamentale nella crittografia tramite AES per diverse ragioni:

- **Problema dei caratteri non stampabili:**
 - La cifratura AES produce byte binari che possono includere qualsiasi valore da 0 a 255;
 - Molti di questi valori rappresentano caratteri non stampabili o di controllo;
 - Questi caratteri causerebbero problemi se inseriti direttamente in:
 - Cookie HTTP (potrebbero terminare prematuramente la stringa);
 - Database (potrebbero interferire con la sintassi SQL);
 - JSON (potrebbero interrompere la struttura JSON);
- **Problema della codifica dei caratteri:**
 - I byte cifrati non appartengono a nessuna codifica di caratteri specifica (UTF-8, ISO-8859, ecc.);
 - Trattarli direttamente come stringhe porterebbe a errori di codifica/decodifica.

6 – Gestione sicura delle password

La gestione delle password è un meccanismo importante per salvaguardare le credenziali dell'utente.

Implementazione nell'applicazione

L'app implementa un sistema di gestione delle password attraverso la classe **PasswordManager.java**:

```

public class PasswordManager {

    // Costanti per i requisiti delle password
    private static final int MIN_PASSWORD_LENGTH = 8;
    private static final String SPECIAL_CHARACTERS = "!@#$%^&*()-_+=<>?.";
    private static final String HASH_ALGORITHM = "SHA-256";

```

I salt sono generati in modo sicuro per ogni utente:

```

    public static byte[] generateRandomBytes(int saltLength) {
        byte[] salt = new byte[saltLength];
        SecureRandom secureRandom = new SecureRandom();
        secureRandom.nextBytes(salt);
        return salt;
    }

```

Il seguente approccio garantisce che ogni utente abbia un salt diverso, impedendo attacchi con tabelle rainbow. Attraverso l'utilizzo di SecureRandom vengono generati salt imprevedibili. Inoltre, i salt sono sufficientemente lunghi per prevenire eventuali attacchi. Quindi prima che la password venga memorizzata sul database viene generata una stringa di 16 bytes casuale che viene concatenata alla password e poi hashata in maniera tale che risulta impossibile recuperare la password in chiaro a partire dall'hash.

Hashing delle password: le password vengono sottoposte a hashing combinando password e salt:

```

    public static byte[] concatenateAndHash(byte[] password, byte[] salt) {
        try {
            // Alloca un nuovo array di byte con dimensioni totali
            byte[] concatenatedData = new byte[password.length + salt.length];

            // Copia i dati dal primo array di byte al nuovo array
            System.arraycopy(password, 0, concatenatedData, 0, password.length);

            // Copia i dati dal secondo array di byte al nuovo array
            System.arraycopy(salt, 0, concatenatedData, password.length, salt.length);

            // Ottieni un'istanza di MessageDigest con l'algoritmo SHA-256
            MessageDigest digest = MessageDigest.getInstance(HASH_ALGORITHM);

            // Calcola l'hash dell'array di byte concatenato
            return digest.digest(concatenatedData);
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("Algoritmo di hashing non disponibile", e);
        }
    }

```

Durante la fase di Login, alla password inserita dall'utente che risulta essere in chiaro, viene applicato lo stesso procedimento. Prima viene recuperata dal database la stringa casuale generata durante la registrazione, viene concatenata alla password e successivamente la nuova stringa ottenuta viene hashata. Il confronto per il login verrà

quindi fatto tra l'hash memorizzato sul database e il nuovo hash ottenuto durante il login.

Validazione delle password: L'app impone criteri di complessità per garantire password robuste:

```
public static boolean isStrongPassword(byte[] password) {
    if (password == null) {
        return false;
    }

    String passwordString = new String(password);

    return isLengthValid(passwordString) &&
        containsUpperCase(passwordString) &&
        containsDigit(passwordString) &&
        containsSpecialCharacter(passwordString);
}
```

I criteri includono:

- Lunghezza minima di 8 caratteri;
- Presenza di almeno una lettera maiuscola;
- Presenza di almeno un numero;
- Presenza di almeno un carattere speciale.

Verifica sicura delle password: la verifica della password è implementata effettuando una protezione contro timing attacks:

```
public static boolean verifyPassword(byte[] inputPassword, byte[] storedHash, byte[] salt) {
    byte[] hashedInput = concatenateAndHash(inputPassword, salt);

    // Controllo se gli hash hanno la stessa lunghezza
    if (hashedInput.length != storedHash.length) {
        return false;
    }

    // Confronto time-constant per evitare timing attacks
    int result = 0;
    for (int i = 0; i < hashedInput.length; i++) {
        result |= hashedInput[i] ^ storedHash[i];
    }

    return result == 0;
}
```

Questo metodo applica lo stesso processo di hashing alla password inserita dall'utente e infatti va a confrontare gli hash in modo time-constant (tempo di esecuzione indipendente dai dati), per andare a prevenire timing attacks (analizzando il tempo di risposta per dedurre informazioni).

Integrazione nel sistema di autenticazione

La classe **PasswordManager.java** è integrata nel sistema di autenticazione dell'applicazione:

- **Durante la registrazione:**
 - Viene verificata la robustezza della password;
 - Viene generato un salt casuale;
 - La password viene sottoposta a hashing con il salt
 - Hash e salt vengono archiviati separatamente nel database.
- **Durante il login:**
 - Il salt dell'utente viene recuperato dal database;
 - La password inserita viene sottoposta a hashing con lo stesso salt;
 - L'hash risultante viene confrontato con quello archiviato.

7 – Lifetime dei dati sensibili

Il “lifetime” o ciclo di vita dei dati sensibili è un aspetto critico della sicurezza delle applicazioni che spesso viene trascurato.

Lifetime delle password e salt: durante la fase di **registrazione** in RegistrationDao.java, vengono svuotate le varie variabili utilizzate per manipolare la password e il salt dell'utente.

```
    } finally {  
        // Cancella in modo sicuro i dati sensibili dalla memoria  
        if (hashedPassword != null) {  
            PasswordManager.clearBytes(hashedPassword);  
        }  
        if (salt != null) {  
            PasswordManager.clearBytes(salt);  
        }  
    }
```

In PasswordManager.java, abbiamo il metodo “clearBytes()” dove tutte le posizioni dell'array vengono impostate a “0”.

```

/**
 * Cancella in modo sicuro il contenuto di un array di byte.
 * Utilizzato per rimuovere dati sensibili dalla memoria.
 *
 * @param sensitiveData Array di byte da cancellare
 */
public static void clearBytes(byte[] sensitiveData) {
    if (sensitiveData != null) {
        for (int i = 0; i < sensitiveData.length; i++) {
            sensitiveData[i] = 0;
        }
    }
}

```

Anche in RegistrationServlet.java, tramite il metodo qui sotto, vengono cancellati in modo sicuro tutti i dati sensibili. Infatti, viene richiamato il metodo “clearBytes()” di PasswordManager.java

```

/**
 * Cancella in modo sicuro tutti i dati sensibili.
 *
 * @param sensitiveData Array di array di byte contenenti dati sensibili
 */
private void clearSensitiveData(byte[][] sensitiveData) {
    for (byte[] data : sensitiveData) {
        if (data != null) {
            PasswordManager.clearBytes(data);
        }
    }
}

```

La password e il salt vengono puliti in maniera esplicita dal buffer di memoria

```

// Array per tenere traccia dei dati sensibili da cancellare
byte[][] sensitiveData = new byte[3][];
sensitiveData[0] = password;
sensitiveData[1] = confirmPassword;

sensitiveData[2] = salt;

} finally {
    // Pulizia dei dati sensibili
    clearSensitiveData(sensitiveData);
    username = null;
}

```

Durante la fase di Login, in AuthDao.java:

```
// Pulisci i dati sensibili
Arrays.fill(password, (byte) 0);
Arrays.fill(salt, (byte) 0);
Arrays.fill(hashPassword, (byte) 0);
```

In LoginServlet.java nel metodo “doPost”:

```
// Pulisci i dati sensibili
PasswordManager.clearBytes(password);
username = null;
```

In TokenManager.java, viene implementato il metodo di pulizia che rimuove i token scaduti dal database:

```
/**
 * Elimina tutti i token scaduti dal database.
 *
 * @return true se l'operazione ha successo, false altrimenti
 */
public static boolean cleanExpiredTokens() {
    Connection connection = null;
    try {
        connection = DatabaseConnection.getConnectionWrite();
        try (PreparedStatement stmt = connection.prepareStatement(DatabaseQueries.getDeleteExpiredTokensQuery())) {
            stmt.executeUpdate();
            return true;
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    } finally {
        closeConnection(connection);
    }
}
```

8 - Thread-safety

Le applicazioni web sono intrinsecamente multi-thread, infatti accessi concorrenti non gestiti correttamente possono causare corruzione dei dati. La mancanza di thread-safety può compromettere l'integrità dell'applicazione. La classe `LogoutServlet` è dichiarata con l'annotazione `@ThreadSafe`, indicando che è stata progettata e implementata in modo da essere sicura per l'accesso concorrente da parte di più thread. Alcune caratteristiche della classe che contribuiscono a garantire la thread-safety:

1. **Utilizzo di HttpSession:** l'accesso all'oggetto `HttpSession` è realizzato in modo sicuro, poiché la variabile `session` è dichiarata all'interno del metodo `doGet` e non è condivisa tra più chiamate ai metodi. Inoltre, il metodo `invalidate()` viene chiamato per terminare la sessione corrente.

```

@ThreadSafe
@WebServlet("/LogoutServlet")
public class LogoutServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    // Pagina di reindirizzamento dopo il logout
    private static final String LOGIN_PAGE = "login.jsp";

    /**
     * Gestisce le richieste GET.
     * Esegue il logout dell'utente invalidando la sessione e i cookie.
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Verifica se è una richiesta di timeout
        boolean isTimeout = request.getParameter("timeout") != null &&
            request.getParameter("timeout").equals("true");

        // Ottieni informazioni sulla sessione e username
        HttpSession session = request.getSession(false);
        String username = null;

        if (session != null) {
            username = (String) session.getAttribute("nomeUtente");

            // Invalida la sessione
            session.invalidate();
            System.out.println("Sessione invalidata");
        }

        if (!isTimeout) {
            // Caso di logout normale (non timeout)

            // Rimuovi i cookie
            removeCookies(request, response);

            // Elimina i token associati all'utente
            if (username != null) {
                TokenManager.deleteTokensByUsername(username);
                System.out.println("Token eliminati per l'utente: " + username);
            }

            // Notifica all'utente
            MessageUtils.showInfoMessage("Logout effettuato con successo!");

            // Reindirizza alla pagina di login
            response.sendRedirect(LOGIN_PAGE);
        } else {
            // Caso di timeout di sessione

            // Verifica se l'utente aveva un cookie rememberToken
            boolean hasRememberToken = false;
            Cookie[] cookies = request.getCookies();
            if (cookies != null) {
                for (Cookie cookie : cookies) {
                    if ("rememberToken".equals(cookie.getName())) {
                        hasRememberToken = true;
                        break;
                    }
                }
            }

            // Se NON aveva il cookie di rememberToken, tratta come un logout completo
            if (!hasRememberToken && username != null) {
                // Rimuovi i cookie comunque per sicurezza
                removeCookies(request, response);

                // Elimina i token associati all'utente
                TokenManager.deleteTokensByUsername(username);
                System.out.println("Token eliminati per l'utente: " + username);
            }

            // Invia solo un codice di successo, il resto è gestito via JavaScript
            response.setStatus(HttpServletResponse.SC_OK);
        }
    }
}

```

2. **Gestione sicura dei cookie:** Il metodo `removeCookies()` manipola i cookie in modo thread-safe. Infatti, ogni richiesta http ha il proprio oggetto “request” e “response”, quindi non c’è rischio di condivisione impropria tra thread diversi.

```
private void removeCookies(HttpServletRequest request, HttpServletResponse response) {
    Cookie[] cookies = request.getCookies();

    if (cookies != null) {
        for (Cookie cookie : cookies) {
            // Per il cookie rememberToken, eliminato anche dal database
            if ("rememberToken".equals(cookie.getName())) {
                String cookieValue = cookie.getValue();
                if (cookieValue != null && !cookieValue.isEmpty()) {
                    String[] parts = cookieValue.split(":", 2);
                    if (parts.length == 2) {
                        String uuid = parts[0];
                        TokenManager.deleteTokenByUuid(uuid);
                        System.out.println("Token eliminato con UUID: " + uuid);
                    }
                }
            }

            // IMPORTANTE: Crea un nuovo cookie con gli stessi attributi
            Cookie killCookie = new Cookie("rememberToken", "");
            killCookie.setMaxAge(0);
            killCookie.setPath("/"); // Usa lo stesso path del cookie originale

            // Aggiungi gli stessi attributi usati in LoginServlet
            killCookie.setHttpOnly(true);
            killCookie.setSecure(true);

            // Se l'applicazione usa un context path
            if (request.getContextPath() != null && !request.getContextPath().isEmpty()) {
                killCookie.setPath(request.getContextPath());
            }

            response.addCookie(killCookie);
            System.out.println("Cookie rememberToken rimosso con parametri completi");
        } else {
            // Per altri cookie, usa il metodo standard
            cookie.setPath("/");
            cookie.setMaxAge(0);
            response.addCookie(cookie);
        }

        System.out.println("Nome del cookie da rimuovere: " + cookie.getName());
    }
    System.out.println("Cookie invalidati");
}
```

3. **Le operazioni chiave sono progettate per essere idempotenti** ovvero, possono essere eseguite più volte senza effetti collaterali:

```
// Invalida la sessione
session.invalidate();
```

```

// IMPORTANTE: Crea un nuovo cookie con gli stessi attributi
Cookie killCookie = new Cookie("rememberToken", "");
killCookie.setMaxAge(0);
killCookie.setPath("/"); // Usa lo stesso path del cookie originale

// Aggiungi gli stessi attributi usati in LoginServlet
killCookie.setHttpOnly(true);
killCookie.setSecure(true);

// Se l'applicazione usa un context path
if (request.getContextPath() != null && !request.getContextPath().isEmpty()) {
    killCookie.setPath(request.getContextPath());
}

response.addCookie(killCookie);
System.out.println("Cookie rememberToken rimosso con parametri completi");
} else {
    // Per altri cookie, usa il metodo standard
    cookie.setPath("/");
    cookie.setMaxAge(0);
    response.addCookie(cookie);
}

```

Questa caratteristica è importante in un ambiente concorrente dove potrebbero verificarsi richieste duplicate o ripetute.

Analisi Dinamica

Durante l'analisi dinamica, sono state testate tutte le funzionalità richieste, analizzando sia esiti negativi che positivi.

Test d'uso

Caso d'uso 1 – Registrazione effettuata correttamente -> l'utente inserisce correttamente tutti i dati, viene reindirizzato alla pagina di login.

Registrazione

Nome utente:
demo00

Password:
.....

La password deve contenere almeno 8 caratteri, includere una lettera maiuscola, un numero e un carattere speciale.

Conferma password:
.....

Immagine profilo:
Scegli file | groot.jpg

Formati accettati: JPG, JPEG, PNG. Dimensione massima: 5MB.

Registrati

Hai già un account? [Accedi](#)

Accedi

Nome utente:

Password:

☐ **Ricordami**

Accedi

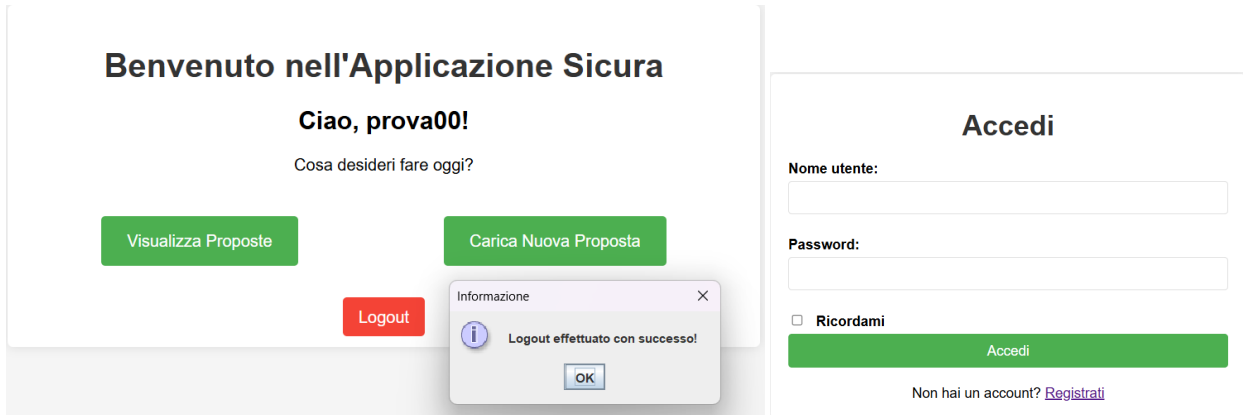
Non hai un account? [Registrati](#)

Informazione

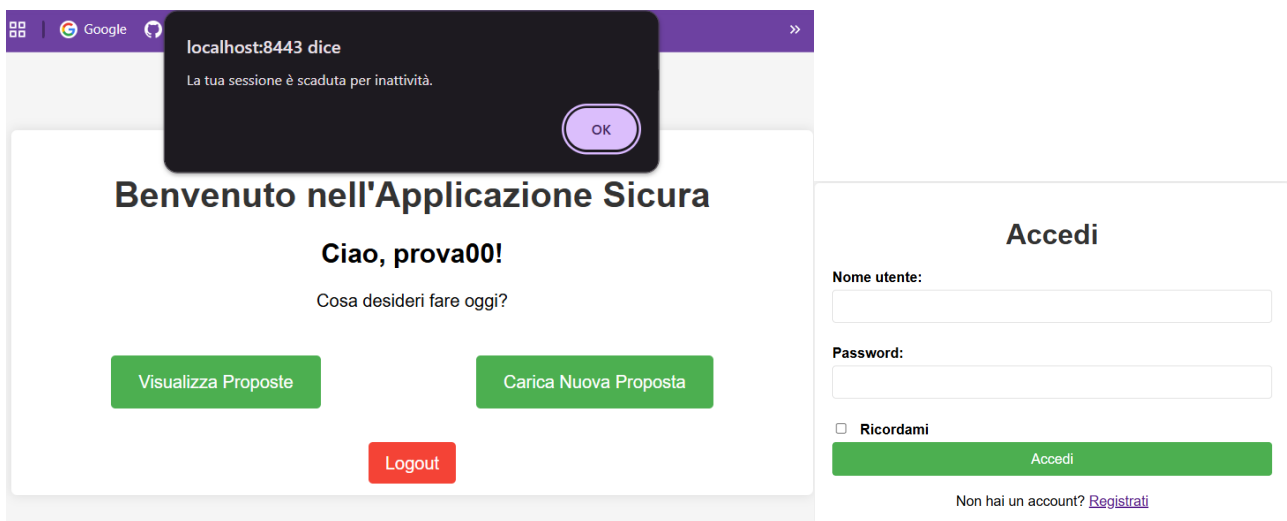
Registrazione effettuata con successo!

OK

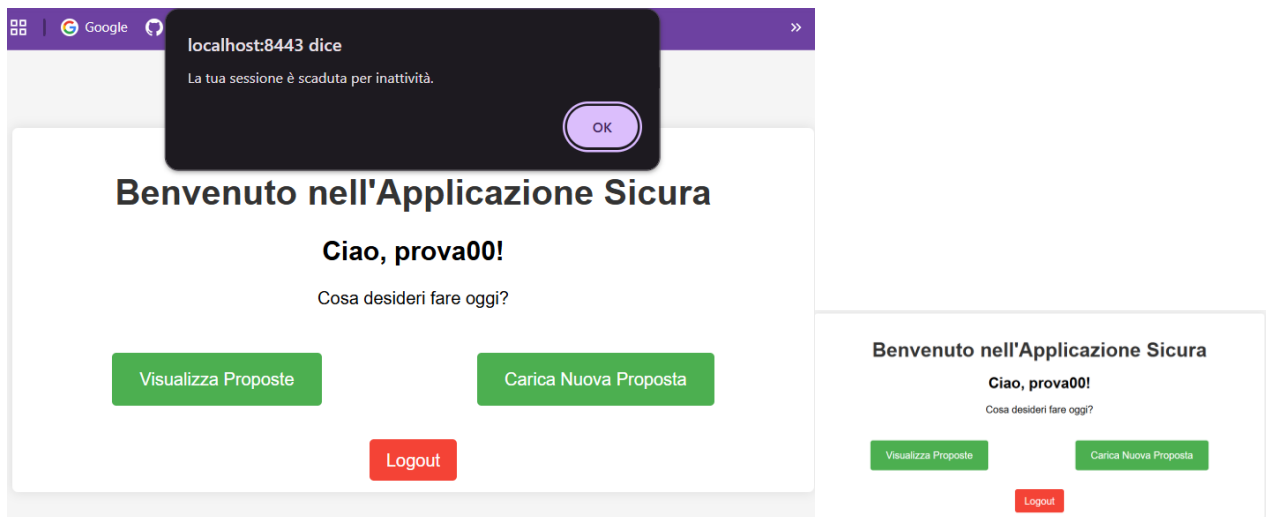
Caso d'uso 2 – Logout manuale effettuato correttamente-> L'utente effettua manualmente il logout, viene reindirizzato alla pagina di Login, la sessione viene invalidata e i cookie rememberMe, se presenti, vengono distrutti all'interno del dispositivo e poi cancellati dal database.



Caso d'uso 3 – Logout automatico di un utente loggato senza “ricordami” effettuato correttamente -> L'utente non interagisce per 15 minuti all'interno della Web App. Viene reindirizzato alla pagina di Login e la sessione viene invalidata



Caso d'uso 4 – Logout automatico di un utente loggato con “ricordami” effettuato correttamente -> L'utente non interagisce per 15 minuti all'interno della Web App. Viene reindirizzato di nuovo nella sua dashboard perché viene rilevato il cookie remember me



Caso d'uso 5: l'utente carica un file txt valido. Il file viene elaborato con l'HTML ed è correttamente visualizzato.



Caso d'uso 6: l'utente visualizza tutte le proposte progettuali presente nell'elenco e anche il loro contenuto e da chi è stata caricata

[← Torna alla pagina principale](#)

Proposte Progettuali

Elenco delle Proposte

Utente: demo00

File: proposta_new2025.txt

• Obiettivo: Sviluppare un'applicazione web sicura utilizzando Java EE, MySQL e Apache Tomcat, con protezione contro attacchi comuni come SQL Injection, XSS e CSRF. • Funzionalità principali: ☒ Registrazione e login sicuri con hashing delle password (Bcrypt) ☒ Upload e gestione file .txt con validazione ☒ Gestione sessioni sicure con token di autenticazione ☒ Logging delle attività e monitoraggio sicurezza

• Stack tecnologico: Backend: Java EE, Servlet, JDBC Database: MySQL Web Server: Apache Tomcat Sicurezza: Hashing password, crittografia AES, gestione cookie sicura

• Sicurezza implementata:
♦ SQL Injection Protection → Prepared Statements
♦ Cross-Site Scripting (XSS) Protection → Sanitizzazione input/output
♦ Cross-Site Request Forgery (CSRF) Protection → Token CSRF nei form
♦ Session Management → Timeout sessione, cookie HttpOnly e Secure

• Conclusione: SecureWebApp sarà un'applicazione progettata con best practice di sicurezza, garantendo un ambiente affidabile per gli utenti.

Utente: demo00

File: malevolo.txt

Proposta progettuale allettante!!!!

Test d'abuso

Caso d'uso 7: utente già registrato (nome già presente nel database)

Registrazione

Nome utente:

manu97

Password:

.....

La password deve contenere almeno 8 caratteri, includere una lettera maiuscola, un numero e un carattere speciale.

Conferma password:

.....

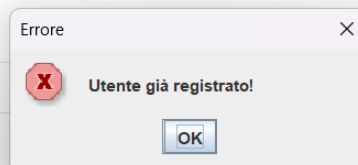
Immagine profilo:

Scegli file groot.jpg

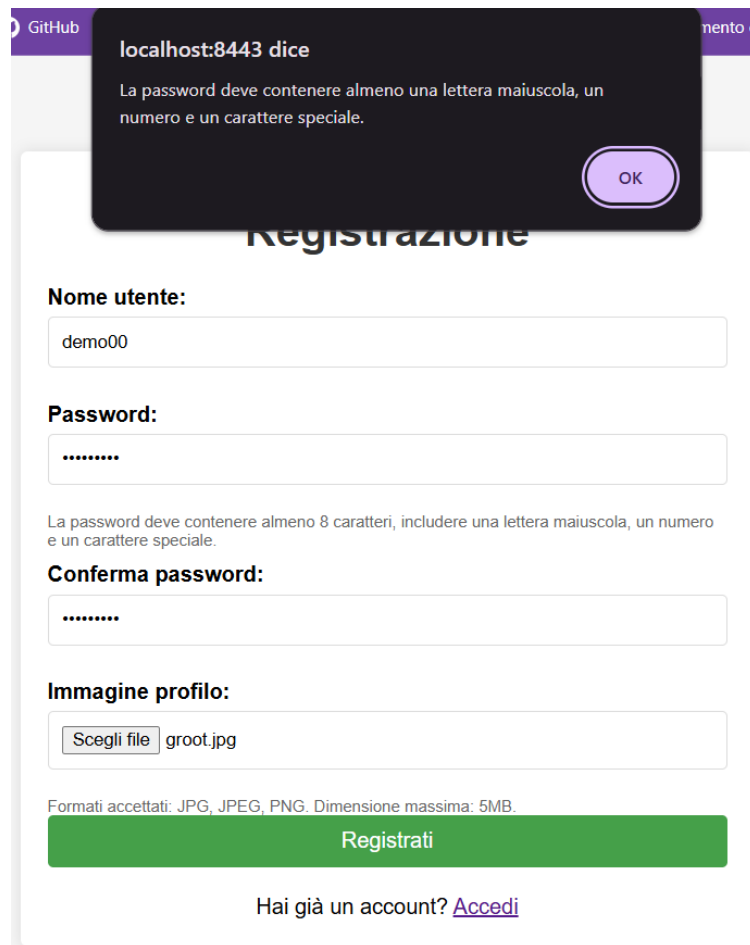
Formati accettati: JPG, JPEG, PNG. Dimensione massima: 5MB.

Registrati

Hai già un account? [Accedi](#)



Caso d'uso 8: l'utente inserisce una password debole. In questo caso è stata inserita la password "CiaoMondo". Per essere considerata valida la password dovrebbe avere almeno 8 caratteri, una lettera maiuscola, un numero ed un carattere speciale.



The screenshot shows a web application interface for user registration. At the top, a dark modal box displays an error message from 'localhost:8443 dice' stating: 'La password deve contenere almeno una lettera maiuscola, un numero e un carattere speciale.' with an 'OK' button. Below this, the registration form is titled 'Registrazione'. It contains the following fields: 'Nome utente:' with the value 'demo00'; 'Password:' with masked characters '.....'; 'Conferma password:' also with masked characters '.....'; and 'Immagine profilo:' with a file selection button 'Scegli file' and the filename 'groot.jpg'. Below the image field, it specifies 'Formati accettati: JPG, JPEG, PNG. Dimensione massima: 5MB.' At the bottom of the form is a large green 'Registrati' button. Below the button, there is a link: 'Hai già un account? [Accedi](#)'.

Caso d'uso 9: l'utente carica un'immagine profilo non valida. In questo caso è stato caricato un file .txt invece che un'immagine.

GitHub

mento c

localhost:8443 dice

Formato file non supportato. Utilizzare JPG, JPEG o PNG.

OK

Nome utente:

Password:

La password deve contenere almeno 8 caratteri, includere una lettera maiuscola, un numero e un carattere speciale.

Conferma password:

Immagine profilo:

Scegli file

proposta_new2025.txt

Formati accettati: JPG, JPEG, PNG. Dimensione massima: 5MB.

Registrati

Hai già un account? [Accedi](#)

Caso d'uso 10: Registrazione con immagine profilo avente estensione valida ma formato non coerente -> L'utente effettua la registrazione caricando un file con estensione apparentemente supportata ma non valida, tutti i campi vengono puliti subito dopo

Registrazione

Nome utente:

Password:

La password deve contenere almeno 8 caratteri, includere una lettera maiuscola, un numero e un carattere speciale.

Conferma password:

Immagine profilo:

Scegli file

fake_foto.jpg

Formati accettati: JPG, JPEG, PNG. Dimensione massima: 5MB.

Registrati

Hai già un account? [Accedi](#)

Errore

✖

Il file non è un'immagine valida

OK

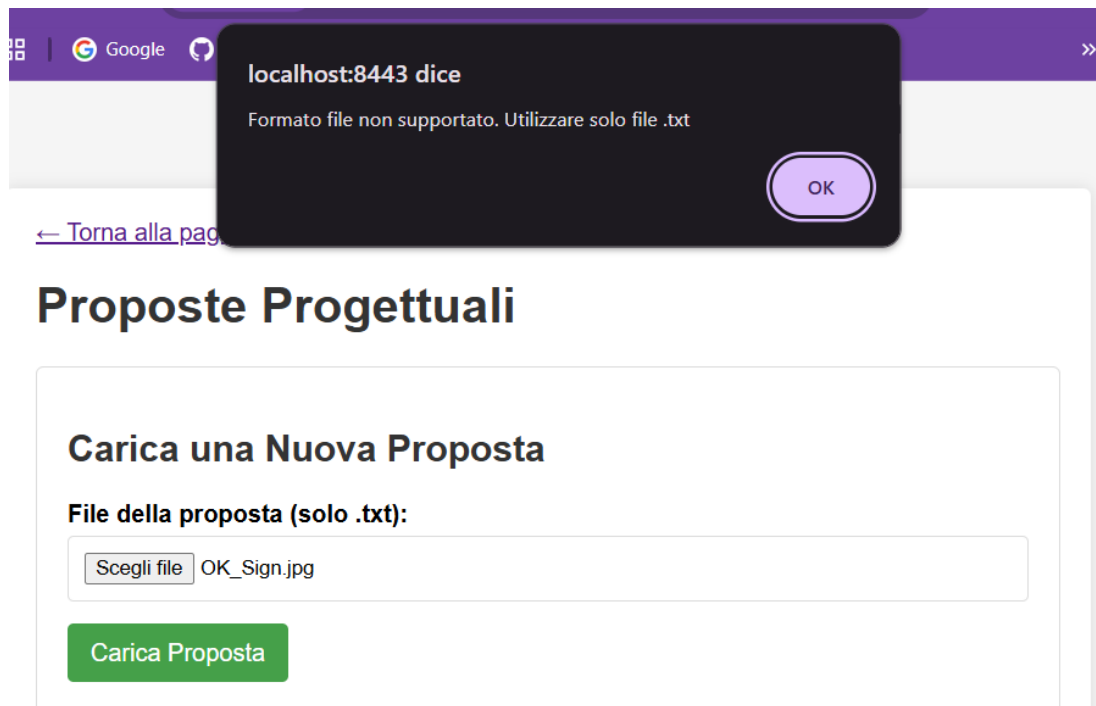
Caso d'uso 11: l'utente inserisce un username che non è registrato.

The screenshot shows a login form titled "Accedi". It has two input fields: "Nome utente:" with the value "prova00" and "Password:" with masked characters ".....". Below the password field is a checkbox labeled "Ricordami". A green button labeled "Accedi" is at the bottom of the form. Below the button, there is a link "Non hai un account? [Registrati](#)". An error dialog box is displayed in the foreground with the title "Errore", a red "X" icon, and the message "Utente non trovato". The dialog has an "OK" button.

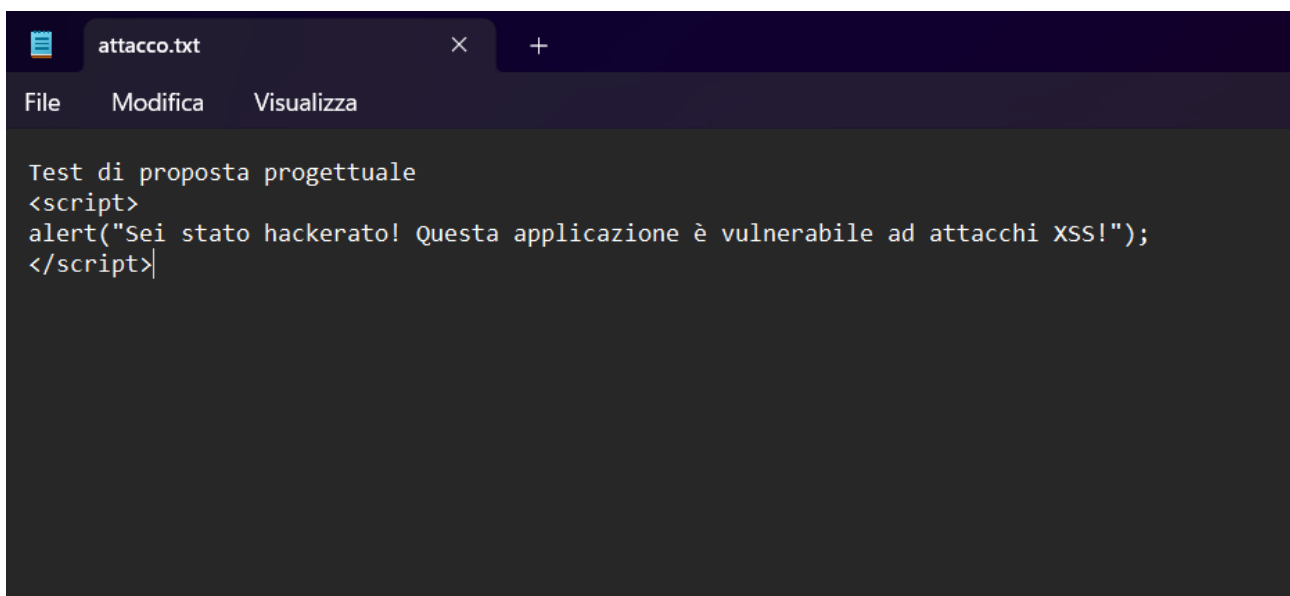
Caso d'uso 12: l'utente inserisce un username valido, ma una password errata.

The screenshot shows the same login form as in Case 11, but with the username "demo00" and password "....". The "Accedi" button is green. Below the button, there is a link "Non hai un account? [Registrati](#)". An error dialog box is displayed in the foreground with the title "Errore", a red "X" icon, and the message "Credenziali non valide. Riprova.". The dialog has an "OK" button.

Caso d'uso 13: l'utente prova a caricare un file (proposta) con formato diverso da .txt



Caso d'uso 14: Caricamento della proposta con script incorporato -> L'utente effettua il caricamento di una proposta in formato "txt" con uno script malevolo incorporato



[← Torna alla pagina principale](#)

Proposte Progettuali

Carica una Nuova Proposta

File della proposta (solo .txt):

Scegli file attacco.txt

Carica Proposta

Informazione



La proposta è stata correttamente caricata!

OK

[← Torna alla pagina principale](#)

Proposte Progettuali

Elenco delle Proposte

Utente: prova00

File: attacco.txt

Test di proposta progettuale