Documentazione autenticazione LBD2023

Table Of Contents

Descrizione Completa

L'idea è quella di tenere l'ID della sessione all'interno di un cookie che poi viene recuperato attraverso l'utilizzo della funzione **recupera_sessione**.
Tutte le funzioni di autenticazione hanno una procedura e una controparte API, chiamabile attraverso le fetch di JS.
Puo' essere utile per poter fare il login senza dover ricaricare la pagina.
Per semplicita' il login viene fatto attraverso il codice fiscale, non e' difficile pero' ampliare il sistema con username e password invece che CF, e poi cambiare le chiamate GET in chiamate POST per non mostrare la password in chiaro.
Perche` le ho volute fare come API? IL debug mi e' risultato piu' comodo e volevo un attimo rispolverare il mio js 😃

Funzioni di utilità

Queste sono funzioni pensate per l'utilizzo delle sessioni durante lo sviluppo di una qualunque procedura.

- utente_da_Username

  E' una funzione di utility che dato un username ritorna la riga dell'utente

- utente_da_sessione

  E' una procedura che wrappa la query per selezionare un utente dal suo **IDUtente** preso da una riga sessione, data come parametro, nel caso in cui la sessione non esista lancia l'eccezione **SESSIONE_NON_TROVATA** da gestire all'interno della procedura chiamante, anche se non dovrebbe mai sollevarsi!

- recupera_sessione

Questa è l'unica azione tipica per la gestione di sessioni che non ha una sua procedura, perche` sembrerebbe ridondante, anche se ha una sua API,
La funzione non prende parametri e ritorna una riga sessione
Agisce recuperando dal cookie l'idsessione, che nel caso in cui il cookie non esista sollevera' l'eccezione **SESSIONE_NON_TROVATA** al momento delle query, in automatico inoltre cambia il valore del timestamp della sessione, che *forse* servirà a pulire le sessioni troppo vecchie.

API

- crea_sessione

  Un API che si trovera' nel path
  `/apex/DB_UTENTE.authenticate.crea_sessione`
  Restituisce un pacchetto HTTP
  con header il cookie **CrociereSessione** con all'interno l'id della sessione
  con body un oggetto json del tipo `{"IDSESSIONE":val}`
  o nel caso l'utente non sia registrato `{"message":"Utente non registrato"}`

- cancella_sessione

  Un API che si trovera' nel path
  `/apex/DB_UTENTE.authenticate.cancella_sessione`
  Restituisce un pacchetto HTTP
  con header l'eliminazione del cookie creato da *crea_sessione*
  con body `{"message":"successo"}`
  Nel caso di fallimento il codice del messaggio sara' **400** con status text **"Login Richiesto"** e body `{"message":"Login Richiesto"}`

- aggiorna_sessione

  Un API che si trovera' nel path:
  `/apex/DB_UTENTE.authenticate.aggiorna_sessione`
  Restituisce un pacchetto HTTP con body `{"message":"successo"}`.
  Nel caso di fallimento il codice del messaggio sara' **400** con status text **"Login Richiesto"** e body `{"message":"Login Richiesto"}`.
  La procedura richiama **recupera_sessione**.

Procedure

- Login

  La procedura di login da chiamare se non si vuole usare il fetch di JS.
  Ha il parametro **p_callback** che di default è settato alla home page (per ora quella di testing) che serve per poi reindirizzare dopo il login.
  Lo script JS richiama **crea_sessione**.

- Logout

La procedura di logou da chiamare se non si vuole usare il fetch di JS. Ha il parametro **p_callback** che di default è settato alla pagina di login, che serve per reindirizzare dopo il logout o in caso di fallimento, che dovrebbe capitare solo se si è provatp a effettuare il log out senza aver effettuato log in.
Lo script JS rhichiama **cancella_sessione**.

Lista_sessioni

E' una procedura per dare un esempio di come potrebbe essere usato il pacchetto authenticate, restituisce una lista di sessioni filtrate per nome e cognome.
possibile aggiunta: controllare che l'utente abbia i permessi per vedere le sessioni di tutti

- ### Show_Session

  E' un api che serve a lista_sessioni per poter funzionare. Ha due parametri
  p_nome il nome dell'utente da cercare (se null tutti i nomi)
  p_cognome il cognome dell'utente da cercare (se null tutti i cognomi)
  La sessione viene recuperata per vedere se e' stato effettuato il login, in caso negativo viene restituito un pacchetto http con stato 400, status text Login Richiesto e body `{"message":"effettuare login"}`

- ### Lista_Sessioni

  Una procedura che al caricamento, dopo aver visto se l'utente e' loggato, ha solo 2 select e 1 button, che quando premuto popola la tabella.
  Nel caso l'utente non sia loggato viene eseguito lo script js **login_prompt**
  Nel caso l'utente sia loggato si recuperano i nomi di tutti gli utenti e si mettono nel select nome, e poi stessa cosa col cognome.
  Viene poi creato il bottone per mostrare le sessioni e la tabella in cui verranno caricate dallo script JS.

TEST_HOMEPAGE

E' una procedura temporanea che serve a mostrare il funzionamento del pacchetto autenticazione, con all'interno il recupero della sessione e tre pulsanti che richiamano le altre procedure.

JS Scripts

Dovendo salvare gli script di js in variabili varchar per poterle caricare in modo semplici sono scritte in js compresso, quindi non molto leggibile per questo il codice degli script viene riportato con una piccola spiegazione.

- ### loginJS

```
window.addEventListener("DOMContentLoaded", ()=>{
  let loginButton = document.querySelector("#login");
  let CF = document.querySelector("#cf");
  let output = document.querySelector("#output");

  loginButton.addEventListener("click", ()=>{
```

```
        fetch("'||API_LOGIN_URL||'?p_CF="+CF.value, {
            method:"GET",
            headers:{
                "content-type":"application/json"
        }})
        .then(response =>{
            if(response.ok) return response.json();
            throw new Error(response.statusText);
        })
        .then(data => window.location.href=url)
        .catch(errMsg=>output.innerHTML=errMsg);
    })})
```

- logoutJS

```
fetch("'||API_LOGOUT_URL||'", {
  method:"GET",
  headers:{
      "content-type":"application/json"
  }})
  .then(response =>{
      window.location.href=url;
})
```

- login_promptJS

```
  login_prompt = s=>{
      fetch("'||API_LOGIN_URL||'?p_CF="+prompt(s+"Login richiesto"), {
          method:"GET",
          headers:{
              "content-type":"application/json"
          }
      })
      .then(data=>window.location.reload())
      .catch(error=>login_prompt(error+" "));
  };
  login_prompt("");
```

Password e tabelle

```
Si discutono le tabelle utilizzate dal pacchetto e il metodo di storage della
password
```

- Tabella BE_Utente

```
CREATE TABLE be_UTENTE(
    IDUtente INT PRIMARY KEY,
    CF CHAR(16) NOT NULL UNIQUE,
    Nome VARCHAR2(255) NOT NULL,
    Cognome VARCHAR2(255) NOT NULL,
    InformazioniPagamento VARCHAR2(255) NOT NULL,
    EPrivilegiato NUMBER(1) NOT NULL,
    SALT RAW(32) NOT NULL,
    username VARCHAR2(64) NOT NULL UNIQUE,
    password RAW(32) NOT NULL
);
```

La tabella be_UTENTE contiente tutte le informazioni che sono proprie dell'utente,
Gli attributi sono:

- IDUtente : un intero incrementale
- CF, Nome, Cognome, InformazioniPagamento : informazioni "del cliente"
- EPrivilegiato : Un Flag che dice se effettivamente l'utente ha un ruolo oppure e' un utente normale
- SALT: 32 byte generati casualmente
- username : una stringa unica che indica l'utente
- password : 32 byte per la password vedi password

- Tabella BE_Sessione

```
CREATE TABLE be_UTENTE(
    IDSessione INT PRIMARY KEY,
    IDUtente INT NOT NULL,
    EPrivilegiato NUMBER(1) NOT NULL,
    TStamp TIMESTAMP(4) NOT NULL,
    PEPPER RAW(32) NOT NULL,
    FOREIGN KEY(IDUtente) REFERENCES BE_Utente(IDUtente)
);
```

- IDsessione : un intero incrementale
- IDUtente : chiave esterna per be_utente
- EPrivilegiato: e' una copia del campo in be_utente, e' ripetuto ma non e' un problem perche` la sessione e' uno stadio piu' in la' della registrazione
- TStamp : l'ultimo momento in cui e' stata fatta un'operazione
- PEPPE : 32 byte vedi password

- Vista Utente e Sessione

  Sono state create perche` la parte della password e' stata inserita dopo, quindi servono a mantenere la stessa interfaccia di utilizzo

- password

Il salvataggio della password e' un'operazione abbastanza coinvolta quindi si divide in fasi, registrazione, login e logout.
Registrazione :

```
DECLARE
        v_salt RAW(32) := DBMS_CRYPTO.RANDOMBYTES(32);
        v_pwd RAW(128) := UTL_I18N.STRING_TO_RAW(DATA  => 'admin', --
PASSWORD
                                                 DST_CHARSET  => 'AL32UTF8');
        v_concats RAW(128) := UTL_RAW.CONCAT(R1  => v_pwd /*IN RAW*/,
                                             R2  => v_salt /*IN RAW*/);
BEGIN

        INSERT INTO be_UTENTE VALUES (1, 'AAAAAAAAAAAAAAAA', 'ADMIN',
'ADMIN', 'carta', 1, v_salt, 'admin', DBMS_CRYPTO.HASH(v_concats,
DBMS_CRYPTO.HASH_SH256)); -- SALE,USERNAME,PASSWORD
END;
/
```

Per registrare un utente vanno preparate prima alcuni dati : v_salt si inizializza con 32 byte casuali v_pwd, si inizializzza con la codifica della password in al32utf8 (un charset) v_concats, che avra' la concatenazione di password e sale dopo si prosegue con l'insert, con nulla di particolare tranne la chiamata a dbms_crypto.hash che hasha la concatenazione di password e sale con HASH_SH256

Login: Quando si crea una nuova sessione, per vedere se la password e' giusta, si prende l'ultimo pepper che e' stato messo nell'ultima sessione (piu' dopo), e si hasha la concatenazione di password, pepe e sale, e si controlla che siano uguali l'hash di password pepe e sale nel database e l'hash di password fornita e stessi pepe e sale poi si genera un nuovo pepe, e si salva nella tabella be_utente la nuova password come hash della concatenazione password, pepe nuovo e sale, **Logout**: L'importante e' che non si cancellino le ultime sessioni di ogni utente dato che son quelle che contengono il pepe, se si dovesse perdere l'hash dentro be_utente non sarebbe recuperabile

Utilizzo tipico

Se non si deve fare niente di particolare il mio consiglio per usare velocemente la libreria e' scrivere la procedura in questo modo

```
PROCEDURE foo([...])
IS
    [...]
    v_SESSIONE SESSIONE%ROWTYPE;
    v_UTENTE UTENTE%ROWTYPE;
BEGIN
    v_SESSIONE := authenticate.RECUPERA_SESSIONE;
    v_UTENTE := authenticate.UTENTE_DA_SESSIONE(v_SESSIONE);
    [...]
EXCEPTION
    WHEN authenticate.SESSIONE_NON_TROVATA THEN
```

```
        HTP.P('<script>window.location.href="||Authenticate.LOGIN_URL||?
P_CALLBACK={indirizzo callback}";</script>');
    [...]
END foo;
```

dopo aver messo queste rige la pagina cerchera' l'**IDSessione** nei cookie,
se non lo trova va nel ramo exception e stampa uno script che reindirizza alla pagina di login, e una volta fatto la pagina di login reindirizza nell'url che avete messo al posto di `{indirizzo callback}`,
se invece trova il cookie in `v_Sessione` avrete la riga della sessione corrente e in `v_UTENTE` la riga con le informazioni dell'utente.

Potenziali criticita'

Ci sono delle falle nella sicurezza dell'autenticazione,

- 1 : non si firma il cookie, quindi si potrebbe falsare
- 2 : la sessione non ha scadenza NOTA : il pepe che cambia per ogni sessione rende piu' forte il salvataggio della password,
  un attacco in cui si conosce la firma del cookie si hanno effettivamente solo scadenza del cookie minuti a sessione