Protocollo di comunicazione

Documentazione dei servizi

https://documenter.getpostman.com/view/23741702/2s93eR4vYa

Repository pubblica github
https://github.com/DanielSan5/BLASP_Project_WebService

Implementazioni necessarie lato client

XMLHttpRequest, utilizzo token per l'autenticazione, gestione formato JSON

Gestione formato JSON

Oggetto semplice - trasformazione JSON

Per prima cosa dobbiamo definire il mezzo per la creazione di un **oggetto** in javascript come per esempio espone il codice seguente:

```
let dati = { // let: dichiarazione variabile che non può essere ridichiarata
    "Username": usernameInput.value,
    "Password": passwordInput.value,
};
```

In questo segmento di codice si definisce una variabile chiamata "*dati*" e le si assegna un oggetto. Un **oggetto** in JavaScript è una collezione di **proprietà**, dove ogni proprietà ha una **chiave** ed un **valore** (nella trasformazione in JSON sentirete parlare di coppia "*chiave*" - "*valore*").

Nel caso qui rappresentato l'oggetto ha due proprietà: "Username" e "Password". Il valore di queste proprietà viene invece preso dai campi di input contenuti nel form HTML, recuperando il loro valore. Quindi in sostanza questo pezzo di codice crea un oggetto che contiene le informazioni dell'utente (inserite in input) e le associa alle proprietà denominate "Username" e "Password".

Passiamo ora alla trasformazione di questo oggetto in una stringa **JSON**, come da seguente:

```
var Datijson = JSON.stringify(dati);
```

Questa riga di codice converte l'**oggetto** "dati" in una **stringa JSON**, un formato di scambio dati utilizzato nelle comunicazioni tra client e server in applicazioni web.
La funzione "*JSON.stringify()*" prende un **oggetto JavaScript** come argomento e restituisce una **stringa JSON** che rappresenta l'oggetto.
Come annunciato in precedenza abbiamo coppie di **chiavi** e **valori** caratterizzate dalla sintassi:

- "*chiave*":"*valore*"
- Separati dalla virgola ","

In conclusione la nostra variabile "Datijson" potrà quindi essere visualizzata in questo modo
{"Username":"**DanielSan5**","Password":"**Ciao123456**"}

Oggetto con array - trasformazione JSON

A questa tipologia di formato possono essere associati, come coppia di chiave valore, anche degli **array**, analizziamo infatti l'esempio seguente:

```
let numeri = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let dati = {
    "Username": usernameInput.value,
    "Password": passwordInput.value,
    "Numeri": numeri,
};
```

In questo caso abbiamo ripreso l'**oggetto** precedentemente analizzato e abbiamo solamente dichiarato un **array** di numeri (che va dall'1 al 10) preventivamente per poi andarlo ad aggiungere come **valore** della **proprietà** denominata "*Numeri*" dell'oggetto "*dati*".
In questo caso, applicando la stessa funzione vista precedentemente:

```
var Datijson = JSON.stringify(dati);
```

Come risultato avremo una stringa **JSON** di questo tipo:

{
"Username":"**DanielSan5**",
"Password":"**Ciao123456**",
"Numeri":**[1,2,3,4,5,6,7,8,9,10]**
}

Oggetto contenente un altro oggetto - trasformazione JSON

A questo punto analizziamo il caso in cui il messaggio in formato JSON contenga un **oggetto**, ecco l'esempio pratico:

```
let indirizzo = {
   "Via": IndirizzoInput.value,
   "Civico": NumeroCivicoInput.value,
   "Citta": CittaInput.value,
};
let dati = {
    "Username": usernameInput.value,
    "Password": passwordInput.value,
    "Indirizzo": indirizzo,
};
```

Come possiamo notare la **sintassi** dell'oggetto "**dati**" non varia e viene semplicemente aggiunta la nuova coppia chiave-valore: **"Indirizzo"**: ***indirizzo***, dove il valore viene rappresentato dalla variabile ***indirizzo***. Non e' un caso che abbiamo scelto proprio l'indirizzo come esempio, poiché l'indirizzo e' infatti composto da più campi (via, numero civico e città), e' bene considerare tutti i dati che rappresentano un indirizzo all'interno di un **oggetto a parte**. N.B.: questo non si applica per una data ad esempio, sempre scomponibile, poiché esiste un formato standard utilizzato dalla maggior parte dei sistemi che permette di rappresentare la data in un'unica stringa (formato: "YYYY-MM-DD", year-month-day).

Per convertire quindi l'oggetto "*dati*" in una stringa JSON basta semplicemente applicare la funzione dei due casi precedenti, ovvero:

```
var Datijson = JSON.stringify(dati);
```

Ed il risultato che si ottiene in questo ultimo caso è quanto segue:

```
{
  "Username": "DanielSan5",
  "Password": "ciao12345",
  "Indirizzo":{
        "Via": IndirizzoInput.value,
        "Civico": NumeroCivicoInput.value,
        "Citta": CittaInput.value,
   }
}
```

Oggetto contenente array di oggetti - trasformazione JSON

A questo punto analizziamo l'ultimo caso, forse quello un po' più complesso, in cui abbiamo una struttura di dati in **JavaScript** formata da un **oggetto** contenente a sua volta un **array di oggetti**, ecco l'esempio pratico:

```
let numeri = [
  { "tipo": "numero", "valore": 1 },
  { "tipo": "numero", "valore": 2 },
  { "tipo": "numero", "valore": 3 }
];
let dati = {
    "Username": usernameInput.value,
    "Password": passwordInput.value,
    "Numeri": numeri,
};
```

Anche qui la **sintassi** dell'oggetto "**dati**" resta invariata, si aggiunge la coppia chiave-valore: Numeri: numeri , nella quale il valore e' in realtà composto da un array di **3 istanze** (oggetti), ciascuno con **proprietà** "*tipo*" uguale a "*numero*" e una **proprietà** "*valore*" uguale a *1, 2, 3* (in scala).
Per convertire quindi l'oggetto rappresentato dalla variabile "*dati*" in una stringa JSON basta semplicemente applicare la funzione dei tre casi precedenti, ovvero:

```
var Datijson = JSON.stringify(dati);
```

Ed il risultato che si ottiene in questo ultimo caso è quanto segue:

```
{
  "Username": "DanielSan5",
  "Password": "ciao12345",
  "Numeri": [
            {
            "tipo": "numero",
            "valore": 1
            },
            {
            "tipo": "numero",
            "valore": 2
            },
            {
            "tipo": "numero",
            "valore": 3
            }
  ]
}
```

Richiesta ad un server web e gestione della risposta

Come effettuare la richiesta

Analizziamo per prima cosa come effettuare una **richiesta** ad un server web e soprattutto cosa si intende per **server web**.
Il compito principale del server web è quello di gestire le richieste inviate da parte dei client e fornire ad essi le risposte appropriate in base a quello che si chiede. Quando un client (banalmente un browser) invia una richiesta HTTP al server web, quest'ultimo elabora la richiesta ricevuta, esegue eventuali script ed invia al client la risposta (sempre in formato JSON).
Ora, definito che per client si intende l'*utente* che effettua la richiesta tramite il browser e che per **server web** si intende il *servizio* da noi elaborato, contenente **applicazioni web** che gestiscono la **richiesta** ed elaborano una **risposta**, specifichiamo il funzionamento per effettuare la **richiesta** tramite JavaScript.
Per fare questo utilizzeremo l'oggetto fornito da Javascript: **XMLHttpRequest,** il quale permette di interagire con il server inviando richieste *(POST* o *GET)* tramite HTTP, e gestirne la risposta.
Questo oggetto viene inizializzato in questo modo:

```
const xml = new XMLHttpRequest();
```

In questo esempio abbiamo creato un **oggetto** di tipo **XMLHttpRequest** che ha la denominazione "xml".
Ora è **obbligatorio**, dato che potrebbe essere generato un **errore** nella richiesta HTTP, utilizzare il blocco `try{}catch{}` per gestire eventuali errori, in questo modo:

```
try{
   // blocco di codice che può generare un errore
   // qui dentro inseriremo TUTTO quello che riguarda la richiesta
}catch(error){
   // gestione dell'errore
   console.log(error);
}
```

Per procedere nell'effettuare la richiesta bisogna specificare l'**URL del server**, per far si che il nostro oggetto **XMLHttpRequest** sappia a *chi* inoltrare la richiesta, ecco come fare:

```
let url ="http://www.server.com/ServizioWeb/RegistrazioneUtenti";
```

Si può notare come molto banalmente si tratti di una **stringa** contenente l'**indirizzo del server web**, niente di più e niente di meno (l'indirizzo del servizio sarà fornito da noi responsabili del back end). Ora bisogna effettuare l'effettiva richiesta HTTP specificando la tipologia, POST o GET.

Vediamo ora come aprire una richiesta al server web utilizzando l'oggetto prima specificato.
Analizzando la seguente riga di codice:

```
xml.open('POST', url);
```

Possiamo notare che:

- Il metodo `open()` è stato richiamato sulla variabile precedentemente dichiarata di tipo **XMLHttpRequest**
- All'interno del metodo vanno inseriti come **parametri**:
    - Il **tipo** della richiesta (POST o GET).
    - L'**indirizzo** del server web a cui effettuare la richiesta, precedentemente dichiarato nella variabile *url*.

Dopo aver **INIZIALIZZATO** la richiesta dobbiamo definire cosa dovrà eseguire l'applicazione una volta che la richiesta sarà andata a **buon fine** ed avrà quindi ricevuto una risposta, nel seguente modo:

```
xml.onload = function() { gestisciRisposta(this); };
```

Nello specifico viene richiamata la **proprietà** `onload` sulla variabile di tipo XMLHttpRequest, la quale prevede la dichiarazione di una funzione da eseguire quando la richiesta HTTP, che effettueremo nella prossima riga di codice, verrà **completata** con successo. In questo specifico caso, se la richiesta andrà a buon fine, verrà eseguita la funzione `gestisciRisposta(this)`, dichiarata in un **segmento** del codice JavaScript a parte (lo vedremo meglio nei passaggi del *come gestire la risposta*).
N.B.: il termine "this" si riferisce a **questa** richiesta che stiamo effettuando, praticamente e' come se, quando la risposta arriva, essa venga "passata" alla funzione gestisciRisposta per essere effettivamente gestita.

Prima di ricevere la risposta ovviamente la richiesta deve essere effettivamente aperta ed inviata, per farlo utilizziamo la funzione `send()`...
Attenzione, in questo caso, trattandosi di una richiesta di tipo POST, la funzione ha come parametro la **stringa** in formato **JSON** precedentemente creata. Nello specifico questo è il funzionamento:

```
xml.send(Datijson);
```

Notiamo come, anche questa funzione, viene richiamata sull'**oggetto** XMLHttpRequest, come i precedenti. In questo caso la funzione `send()` invia la richiesta al **server web** specificando, trattandosi di una richiesta **POST**, i dati necessari e precedentemente definiti nella variabile `Datijson,` contenente i dati in formato JSON.
Ora, se tutto è andato a buon fine, possiamo procedere con la gestione della risposta.
N.B.: la funzione send() deve essere richiamata dopo la funzione onload(), poiché la funzione onload() serve al browser per far corrispondere quella specifica richiesta ad un eventuale gestione della risposta.
N.B.: i metodi PUT e DELETE non richiedono informazioni aggiuntive nella richiesta, basterà solo specificare nel metodo open() di che metodo si tratta.

Come gestire la risposta

Dopo aver visto come effettuare una richiesta HTTP dobbiamo ora capire come **gestire** la risposta ricevuta dal server.
Gestire la risposta è molto più semplice di **effettuare la richiesta**, ma anche in questa parte è necessario tenere presente alcuni accorgimenti.
Per prima cosa **implementiamo** la già vista funzione `gestisciRisposta(risposta)` che viene per l'appunto definita al fine di gestire la risposta del **server web** ed invocata quando la risposta arriva.
Per prima cosa bisogna controllare se lo *stato* della risposta HTTP è uguale a 200, in questo modo:

```
function gestisciRisposta(risposta) {
    if (risposta.status == 200) {
      // codice da eseguire
    }
}
```

Questo perché viene fatto e cosa indica?
Nello specifico lo **stato** della risposta HTTP è un valore numerico che rappresenta appunto lo stato (completata con successo/errori vari).
Questo valore viene restituito dal server web nella proprietà '**status**' della risposta, la quale rappresenta uno speciale oggetto con varie proprietà (voi lato front end non dovrete preoccuparvi dei vari tipi di status ma solamente che questo sia uguale a 200). In questo caso, nel caso la '`risposta`' sia uguale a '`200`' significherà che la richiesta HTTP è stata completata con **successo** ed il server web ha restituito una risposta **valida**.
Come ultima parte vediamo come '*trattare*' questa **risposta**, per capire in che formato di dati è la suddetta e cosa contiene basta fare:

```
let rispostaInStringa = risposta.responseText;
console.log(rispostaInStringa);
```

In questo modo, all'interno della **console**, verrà stampato il messaggio della **risposta** ricevuta dal **server web** in modo che poi, a seconda dei controlli che preferite o manipolandola come preferite, possiate completare la vostra parte di front end.
Un esempio di messaggio della risposta potrebbe essere ad esempio il ticket di un determinato tutor o i dati di un utente.

N.B: il termine **responseText** si riferisce ad un'altra proprietà dell'oggetto risposta, la quale presenta l'effettivo messaggio inviato dal server in formato stringa.
Per poter manipolare il messaggio la prima cosa da fare e' trasformare la stringa del messaggio (`rispostaInStringa`) in un oggetto Javascript, in modo da poterlo utilizzare riferendosi alle sue chiavi.
In questo modo:

```
let x = JSON.parse(rispostaInStringa)
```

Ora ci si potrà riferire ai valori della variabile x a partire dalle chiavi.
Come ottenere il valore date queste varie casistiche di coppie:
chiave con valore: `x.chiave;`
chiave con valore array normale: `x.chiave[posizione];`
chiave con valore oggetto: `x.chiave.chiaveOggetto;`
chiave con valore array di oggetti: `x.chiave[posizione].chiaveOggetto`;

Utilizzo token per l'autenticazione

La tecnologia di autenticazione utilizzata sarà quella dei token. Questa tecnologia prevede che la prima volta che un utente si logga all'interno del sistema quest'ultimo utilizzi username e password per farsi riconoscere, e quando il server autorizza per la prima volta l'utente allo stesso tempo genera questa speciale stringa, la quale viene mandata al client nella risposta successiva.
Questa stringa dovrà essere salvata lato client attraverso i cookie ed essere utilizzata all'interno di ogni successiva richiesta, nello specifico nell'intestazione.
Quello che contiene il token non serve approfondirlo (https://jwt.io/introduction per chi fosse interessato), ci basti solo sapere che si tratta di un modo abbastanza sicuro per mantenere attiva una sessione, in modo da non dover fare il login ad ogni richiesta verso il server.
Detto ciò vediamo come si può ottenere il token dall'intestazione della risposta:
sull'oggetto **XMLHttpRequest** della **risposta** si dovrà invocare il metodo:

```
let token = risposta.getResponseHeaders("Set-cookie");
```

Ora che si e' ottenuto il token dalla risposta, ci basterà salvarlo nei cookies del browser attraverso javascript, in particolare con la libreria JS cookies, in questo modo:

```
Cookies.set('login_info', token, { expires: …});
```

Una volta salvato nei cookie il token sarà accessibile attraverso questo metodo, sempre appartenente alla libreria JS cookie:

```
let login_info = Cookies.get('login_info');
```

Quindi alla prossima richiesta la variabile `login_info` contenente il token per autenticarsi, dovrà essere inserita nell'intestazione, attraverso questo metodo javascript, notando che la variabile su cui stiamo applicando il metodo e' proprio l'oggetto **XMLHttpRequest** spiegato in precedenza :

```
xml.setRequestHeader("Authorization", login_info);
```

N.B.: questo metodo viene invocato prima di mandare la richiesta, ma comunque dopo il metodo open()

Test Cases

Tipi di errori che si possono ricevere

- **400:** quando il client invia un messaggio con sintassi sbagliata (ad esempio campi vuoti o non conformi al protocollo);
- **401:** quando l'utente non può essere autenticato e quindi non e' autorizzato ad accedere ad una risorsa;
- **403**: quando l'utente può essere autenticato ma non può più accedere (quando e' stato bannato)
- **404:** quando la risorsa a cui si sta inviando la richiesta non esiste;
- **405:** quando il servizio non implementa il metodo richiamato nella richiesta;
- **500:** quando c'è stato un errore nel server.

N.B.: per gestire questi errori lato client ci si ricordi della proprietà **status** dell'oggetto della risposta.

400	401	403	404	405	500
HTTP/3 400 Bad request **{** "**stato**":"*errore client*", "**desc**":"*sintassi errata nella richiesta"* **}**	**HTTP/3 401 Unauthorized** **{** "**stato**":"*errore client*", "**desc**":"credenziali non valide*"* **}**	**HTTP/3 403 Forbidden** **{** "**stato**":"*errore client*", "**desc**":"utente non autorizzato*"* **}**	**HTTP/3 404 Not found** **{** "**stato**":"*errore client*", "**desc**":"*risorsa inesistente"* **}**	**HTTP/3 405 Method not allowed** **{** "**stato**":"*errore client*", "**desc**":"*operazione non accettata"* **}**	**HTTP/3 500 Internal server error** **{** "**stato**":"*errore server*", "**desc**":"*problema nell'elaborazione della richiesta"* **}**

Glossario

- **GET:** La richiesta **GET** viene utilizzata per **richiedere** una risorsa al **server web**, essa viene inviata dal client al server specificando l'URL della risorsa, viene utilizzato questo tipo di richiesta ad esempio quando si vuole visualizzare la pagina principale di un sito web. Questa tipologia di richiesta può inoltre includere **parametri** all'interno dell'URL, in modo che il server possa prenderli ed elaborare una risposta in base ai medesimi.

- **POST**: La richiesta **POST** invece viene utilizzata per **inviare** dati dal client al **server web**, essa viene utilizzata quando il client deve inviare informazioni al server, ad esempio quando si invia un form, molto banalmente quando ci si **registra ad un sito web**. In parole semplici se devo modificare dei dati, aggiungere dei dati, togliere dei dati utilizzo una richiesta POST mentre se devo semplicemente visualizzare dei dati utilizzo una richiesta GET.

- **HTTP:** e' il protocollo utilizzato nella comunicazione client-server con architettura REST e prevede due principali elementi: la richiesta e la risposta. Quando si tratta di risposta, essa può essere di vari tipi (come detto prima, POST o GET sono ciò che ci interessa) e in base al tipo può essere utilizzata per determinate funzionalita'. Per quanto riguarda la risposta, essa viene inviata dal server dopo aver elaborato la richiesta e può contenere anch'essa delle informazioni. Restituisce inoltre sempre uno **stato** il quale rappresenta, sotto la forma di codici, l'andatura della richiesta, ad esempio se e' andata a buon fine, se c'è stato un errore, se la pagina richiesta non esiste (il famoso error 404) ecc..

- **Token per l'autenticazione:** e' una stringa codificata con determinati algoritmi che rappresenta un utente specifico, essa viene generata dal server ed inviata al client dopo aver autenticato il client nel sistema. Il client una volta ricevuta questa stringa, anche detta **token,** ogni volta che dovra' fare una richiesta utilizzerà quest'ultima per autenticarsi, inserendola nell'header della richiesta.