Relazione progetto VMR_Condivisione_Spese

SEP, sharing expense pay

Il progetto che abbiamo voluto creare tratta di una piattaforma per la spesa, sia singola che di gruppo, questo perché per dei studenti fuori sede è importante tenere traccia delle uscite economiche e avere indietro facilmente i soldi prestati.

Abbiamo deciso di sviluppare questa SPA; single page application (la differenza delle tradizionali applicazioni web multi-pagina, una SPA carica il suo contenuto iniziale una sola volta e successivamente aggiorna dinamicamente il contenuto della pagina senza dover ricaricare l'intera pagina, ed inoltre possono comunicare con il server tramite API per ottenere dati o eseguire operazioni di backend) tramite Ionic con framework recat ed utilizzando un database MySQL.

Il sito si presenta con una pagina di apertura dove si può decidere se accedere o registrarsi (index), dopo aver fatto la scelta ed una volta eseguito l'accesso si avranno più opzioni come quella di inserire delle carte con cui saldare i debiti o vedere le spese in sospeso o la possibilità di effettuare modifiche ai dati personali come password ed email.

Mentre quanto riguarda la sicurezza questi sono gli aspetti;
Possiede più strumenti per proteggersi da attacchi hacker di più tipo.

- Protezione SQL INJECTION

Per proteggersi dalla SQL INJECTION, cioè mettere stringhe che se messe in una query da eseguire al BDMS del database comandi in modo spontaneo, il server si collega al Database MySQL con un' oggetto della classe ConnectionDB.
Al momento della creazione di questo oggetto, si passano i parametri necessari per trovare il DB ed accedervi con le dovute credenziali.

Nella classe ConnectionDB, esiste un metodo chiamato "GetQuery". Questo metodo prende 2 parametri:

1) Richiesta MySQL: la richiesta che si farà al DB. In questa richiesta, ci sono dei ? al posto dei valori
2) Lista di parametri: Valori dei parametri da passare alla nostra richiesta

Nella richiesta, ci sono dei ? al posto dei valori, per evitare di concatenare delle stringhe (ed essere vulnerabili alla SQL INJECTION). Il metodo riconosce quali parametri deve mettere in quali punti della richiesta, così da poter riconoscere quali sono le parti di richiesta e quali sono i valori.

- Protezione XSS

I parametri che mostro dentro il codice HTML sono i parametri che passo dentro il body delle mie richieste e risposte.

I parametri che vengono passati attraverso il body di una richiesta vengono verificati se contengono codice malevolo XSS attraverso un metodo di una libreria personalizzata.
Se un parametro ottenuto è pericoloso (cioè verrà mostrato in qualche pagina HTML ed ha del codice javascript al suo interno), il server rifiuta quella richiesta.

Vengono controllati solo i parametri nella registrazione e di modifica dei dati (nelle impostazioni non ancora sviluppate), poichè la registrazione e modifica dei dati sono le uniche API di inzio da cui posso entrare i dati in modo rischioso, nelle altre i dati vengono confrontati con i dati del DB e non salvati.

- Autenticazione JWT

Il server dispone di un oggetto di una classe ManagementJWT, che gestisce l' associazione JWT - Nickname e scadenza dei JWT stessi.
Ogni JWT scade dopo 30 minuti dalla sua creazione. Ogni volta che un utente vuole chiamare un API protetta, deve mostrare il suo JWT. I JWT (dal lato server che client) vengono passati dall' header "Authorization".
Ogni volta che un utente mostra un JWT valido, il server ne restituisce (ed assegna a quell' utente) un JWT diverso, che abbia però lo stesso timestamp di scadenza di quello precedente. In questo modo un hacker deve tenere traccia di tutti JWT che circola.
I JWT vengono modificati con una password casuale.

- Protezione CORS

Il server ha una protezione CORS classica, che si attiva ogni volta che viene richiesta una qualsiasi API. dentro la funzione che accetta o rifiuta la richiesta sotto forma di protezioen CORS, esiste una mappatura 'AllowOrigin'

INDIRIZZO - PORTE, che contiene tutti gli indirizzi (con porte associate) che possono accedere alle risorse del server.
Abbiamo deciso di mettere una stessa protezione CORS per tutti le API, anche se si poteva mettere una protezione CORS per ogni API diversa, ma non aveva senso per noi.

- Protezione di crittografia delle password

Le password vengono protette in questo modo:

1) il browser cripta la password con l' algoritmo SHA256
2) il server riceve la password hashata e crea una stringa casuale da 20 caratteri chiamata SALT.
3) Il server aggiunge questo SALT alla password hashata e passa il tutto dentro un' altro hash, sempre SHA256.
4) Il server si salva in un file json, sul proprio OS, tutte le coppie Nickname - SALT, in modo tale da non mandarle sul DB (e rischiare che qualche hacker la veda in rete)
5) Il server manda la password (con tutti gl' altri dati) la password hashata

Ogni volta che l' utente fa l' accesso, nel caso in cui le credenziali fossero corrette, prima di far accede l' utente alla sua area personale, il server aggiorna il SALT associato all' utente ed aggiorna la password sul Database

- Protezione Brute-force

Per evitare situazioni di brute-force, il server ha un oggetto di una classe BruteforceBlocks. Questo oggetto contiene una lista di blocchi, ad ogni blocco è associato un indirizzo IP ed una porta. La macchina che fa la richiesta ha
5 tentativi e sta al livello 0, finiti i 5 tentativi sale al livello 1 e le sue richieste non vengono accettate per 5 minuti. Finiti altri 5 tentativi ha un blocco di 10, poi 15, poi 20 etc.
Questa "scalata" di livelli viene resettata quando non ci sono più 5 tentativi failli di richieste nell' arco degl' ultimi 30 minuti dall' ultimo sblocco (da quando le richieste della macchina vengono di nuovo accettate).

Invece per quanto riguarda il lavoro di gruppo siamo riusciti a dividerci i compiti, in modo tale da farci lavorare singolarmente su alcuni aspetti, ma richiedendo sempre l’approvazione dell’altro per far si che ne uscisse un progetto di coppia sotto tutti gli aspetti.