1 Lezione del 25-11-24

1.1 Superglobali

1.1.1 $_SERVER

`$_SERVER` rappresenta un'array associativa che contiene opzioni di configurazione per PHP e il server Apache, e gli header delle richieste HTTP inviate dai client.

Ad esempio, possiamo trovare le chiavi:

- `SERVER_NAME`: il nome del sito che è stato richiesto;
- `SERVER_ADDR`: l'indirizzo IP di tale server;
- `DOCUMENT_ROOT`: la directory da cui stiamo eseguendo lo script;
- `SCRIPT_NAME`: una chiave (il nome) dello script in esecuzione;
- `REQUEST_METHOD`: restituisce se il metodo di accesso con cui si è fatto accesso alla pagina è GET, HEAD, POST o PUT;
- `REMOTE_ADDR`: l'indirizzo IP del richiedente; curioso
- `HTTP_USER_AGENT`: sistema operativo e browser del client;
- `HTTP_REFERER`: l'indirizzo IP della pagina (se esiste) che conteneva il link usato per raggiungere la pagina corrente.

Ricordiamo che, come sempre, **non** possiamo fidarci del client, e quindi i valori di `HTTP_USER_AGENT` e `HTTP_REFERER` potrebbero essere falsificati.

1.2 Gestione di file caricati dall'utente

`$_FILES` rappresenta una variabile associativa che contiene oggetti che sono stati caricati dalla richiesta corrente, ciascuno come una coppia chiave-valore. Ricordiamo quindi che la trasmissione di file dal client si fa attraverso la richiesta POST. A questo punto, possiamo quindi creare un form che ottiene file dall'utente come segue:

```
<form enctype="multipart/form-data" method="post">
  <input type="file" name="file1" id="file1"/>
  <input type="submit"/>
</form>
```

Ogni elemento associato alla chiave per ogni file sarà un'array che conterrà i le chiavi:

- `name`: il nome del file sulla macchina del client;
- `type`: il tipo MIME (l'estensione) del file. Potremmo voler limitare i tipi di file supportati: questo si può fare lato server controllando l'estensione di ogni elemento di `$_FILES` caricato. Questo si può fare agevolmente con la funzione `explode()`, che segmenta la stringa in base a delimitatori specificati (nel nostro caso il punto "."), o direttamente controllando il campo `type` di ogni elemento di `$_FILES`.
- `tmp_name`: il nome del file sul server, che è una locazione temporanea;
- `error`: un intero che codifica diversi stati, fra cui ricordiamo UPLOAD_ERR_OK con valore 0 (che significa operazione andata a buon fine).

- `size`: un intero che rappresenta la dimensione in byte del file caricato. Potremmo voler limitare le dimensioni dei file inviati al nostro server. Possiamo fare ciò attraverso 3 meccanismi:
  - HTML nel form di input, cioè inserendo un elemento `input` nascosto con una coppia chiave valore `MAX_FILE_SIZE` e il valore in byte della dimensione massima dei file che il browser dovrà inviare. Questo meccanismo può essere manomesso dall'utente, e quindi va verificato nuovamente lato server. Notiamo inoltre che la maggior parte dei browser in commercio non supportano questo elemtno, ma è il PHP che solitamente si occupa di trasformare la sua inclusione in una legge che impedisce all'utente di caricare file più grandi del dovuto; chiarisci
  - JavaScript nel form di input, cioè controllando il file in uno script lato client. In particolare, l'elemento `input` di tipo `file` contiene un'array `files`, dove ogni file ha un campo `size`. Anche questo meccanismo è facilmente manomesso dall'utente.
  - PHP, controllando come nell'esempio precedente le dimensioni, ma stavolta lato server, cioè usando direttamente l'array `$_FILES` e il campo `size` dei suoi elementi,

1.2.1 Spostare i file

Possiamo spostare i file caricati dall'utente attraverso la funzione `move_uploaded_file()`, che prende come primo argomento la locazione temporanea `tmp_name` del file o altre locazioni dove esso si potrebbe trovare, e come secondo argomento l'indirizzo di destinazione nel filesystem del server.

1.3 Leggere e scrivere file

Ci sono due modi di leggere e scrivere file in PHP:

- **Accesso in stream:** il file viene letto una porzione alla volta, attraverso il concetto di stream, implementato analogamente ad altri linguaggi (come ad esempio il C).
- **Accesso all-in-memory:** il file viene caricato completamente in memoria, rendendo più facile la sua elaborazione.

Chiaramente, il primo approccio si presta a file di grandi dimensioni o a contesti dove le prestazioni sono fondamentali, mentre il secondo approccio è più agevole da usare, ma solo su file di piccole dimensioni.

1.3.1 Accesso in stream

Per aprire un file in modalità stream si usa la funzione `fopen()`. Da qui in poi il file è aperto come uno stream sequenziale, con un cursore che indica la posizione corrente, e sono disponibili le `fread()`, `fgets()`, `fwrite`, `fclose()`, ecc... a cui siamo abituati dal C.

1.3.2 Accesso all-in-memory

Nell'accesso all-in-memory, possiamo usare le funzioni `file()` che mette l'intero file in un array di stringhe rappresentanti ogni riga del file, `file_get_contents()`, che legge l'intero file in una variabile stringa, e `file_put_contents`, che scrive i contenuti di una variabile stringa in un file.