Upload di file

Corso di Laboratorio di Informatica

Prof. Dei Rossi, Leonardo Essam

Cosa si intende per "storage"?

Definizione (dal Web):

"Lo storage (o archiviazione/immagazzinamento) si riferisce alla conservazione di dati digitali su dispositivi o servizi per un uso futuro, tramite varie tecnologie come lo storage a file (cartelle), a blocchi (partizioni) e a oggetti (per grandi dati non strutturati, come video e immagini). Può essere locale (HDD, SSD) o nel cloud, ed esistono diverse tipologie come il data storage primario (RAM temporanea) e secondario (persistente)."

Lo "storage" in PHP

In PHP vengono definiti due concetti che rimandano alla definizione di "storage":

- Storage nel filesystem;
- Storage di sessione.

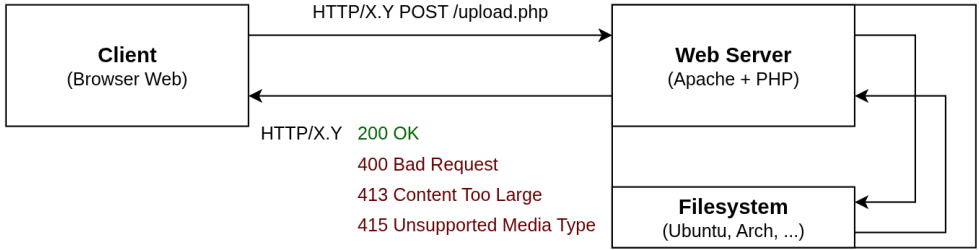
In entrambi i casi, si tratta del salvataggio di dati eterogenei relativi all'utente e/o alle sue operazioni nell'applicazione Web.

Lo storage nel filesystem (1)

Essendo PHP un linguaggio *server-side*, esiste tutta una famiglia di funzioni per gestire l'upload di file e cartelle nel filesystem del server.

Questo tipo di funzioni è utile ad esempio quando si lavora con un particolare tipo di form che richiede all'utente non più (o meglio, anche) l'inserimento di un valore testuale, ma l'upload di un file.

Lo storage nel filesystem (2)



Lo storage nel filesystem (3)

Osserviamo la presenza di diverse opzioni per il codice di stato HTTP che viene restituito all'utente.

Come ripasso:

- 1XX - Informazione (`Informational responses`);
- 2XX - Successo (`Successful responses`);
- 3XX - Inoltro (`Redirection messages`);
- 4XX - Errore client (`Client error responses`);
- 5XX - Errore server (`Server error responses`).

Lo storage nel filesystem (4)

L'utilizzo dei codici 400, 413 e 415 permettono allo sviluppatore di differenziare i messaggi di errore in base al tipo di errore generato nell'analisi della richiesta dell'utente.

HTTP/X.Y 400 Bad Request

Il codice di stato **400 Bad Request** indica che il server non ha elaborato la richiesta a causa di un errore che ha ritenuto essere un errore del client.

Il motivo di una risposta 400 è in genere dovuto a una sintassi della richiesta non corretta.

I client che ricevono una risposta 400 devono aspettarsi che ripetere la richiesta senza modifiche non vada a buon fine, generando lo stesso errore.

Lo storage nel filesystem (5)

HTTP/X.Y 413 Content Too Large

Il codice di stato **413 Content Too Large** indica che l'entità della richiesta era più grande dei limiti definiti dal server. Il server potrebbe chiudere la connessione o restituire un campo di intestazione `Retry-After`.

Prima della RFC 9110, la frase di risposta per lo stato era "Payload Too Large". Questo messaggio è ancora ampiamente utilizzato.

Lo storage nel filesystem (6)

415 Unsupported Media Type

Il codice di stato **415 Unsupported Media Type** indica che il server ha rifiutato la richiesta perché il formato del contenuto del messaggio non è supportato.

Il problema di formato potrebbe essere dovuto al `Content-Type` o al `Content-Encoding` indicati nella richiesta, oppure all'elaborazione del contenuto del messaggio di richiesta.

Alcuni server potrebbero essere rigidi riguardo al `Content-Type` previsto per le richieste.

Un primo esempio (1)

Si consideri il seguente from HTML:

```
<form action="upload.php" method="post"
    enctype="multipart/form-data">
    Seleziona file da caricare:

    <input type="file" name="file1" required>
    <button type="submit">Invia</button>
</form>
```

Un primo esempio (2)

Alcune regole da seguire:

1. Assicurarsi che la form utilizzi POST;
2. La form necessita anche del seguente attributo:
   `enctype="multipart/form-data"`

L'uso di `multipart/form-data` specifica quale tipo di contenuto utilizzare al momento dell'invio della form.

Un primo esempio (3)

Altre cose da notare:

1. L'attributo `type="file"` del tag `<input>` mostra il campo di input come un controllo di selezione file, con un pulsante "Sfoglia" accanto al nome;
2. Il modulo sopra invia i dati a un file chiamato "upload.php".

Un primo esempio (4)

Iniziamo a impostare il codice per gestire l'upload di un file:

```
<?php
$target_dir = "uploads/";

$target_file = $target_dir;
$target_file .= basename($_FILES["file1"]["name"]);
?>
```

Si noti l'uso della variabile globale `$_FILES`. Tale variabile contiene al suo interno i dati relativi ai file inviati tramite una richiesta `POST`.

Un primo esempio (5)

Definizione 1: Funzione basename([...])

La funzione `basename([...])` ammette e richiede come parametro una stringa che rappresenta un percorso nel filesystem.

La funzione restituisce quindi l'ultimo elemento (file o cartella) del percorso dato.

```
<?php
echo(basename("/var/www/html/index.php")); // index.php
echo(basename("/home/rory/Documenti"));    // Documenti
?>
```

Un file temporaneo

Quando viene fatto l'invio della POST da una form contenente dei file, tali file vengono salvati in una posizione temporanea del server.

Si può leggere tale posizione accedendo alla chiave `tmp_name`:

```
<?php
echo($_FILES["file1"]["tmp_name"]);
?>
```

Verrà restituito un percorso assoluto che punterà al file appena caricato.

Who am I? (1)

Ci sono dei casi in cui vi è la necessità di permettere l'upload di solo alcuni tipi di file, come ad esempio quando si vuole caricare una nuova foto profilo su un social network.

In PHP è possibile verificare lato server che tipo di file è stato caricato.

Ricordate il trucco più vecchio del mondo?

Who am I? (2)

Definizione 2: Media types (MIME types)

Un media type (precedentemente noto come Multipurpose Internet Mail Extensions o MIME type) indica la natura e il formato di un documento, file o insieme di byte.

I tipi MIME sono definiti e standardizzati nella RFC 6838 dell'IETF.

Alcuni MIME type più comuni: (link)

Who am I? (3)

È possibile leggere il MIME type di un file accedendo alla chiave `type`:

```
<?php
echo($_FILES["file1"]["type"]; // application/pdf
?>
```

In questo modo è possibile filtrare in modo preciso i tipi di file che sono ammessi!

Esistenza di un file

Prima di caricare un file (o anche dopo per verificare che non ci siano stati problemi), PHP mette a disposizione una funzione per verificare l'esistenza di un file.

Definizione 3: Funzione file_exists([...])

La funzione `file_exists([...])` ammette e richiede come parametro una stringa che rappresenta il percorso di un file.

Restituisce `true` se il file esiste, `false` altrimenti.

Completamento dell'upload

Per finalizzare l'upload di un file nel server, vale la funzione `move_uploaded_file([...])`.

Definizione 4: Funzione move_uploaded_file([...])

La funzione `move_uploaded_file([...])` ammette e richiede due parametri di tipo `string` dove il primo rappresenta il percorso di sorgente (`/tmp`) e il secondo la destinazione dove si vuole spostare il file.

La funzione restituisce `true` se lo spostamento è avvenuto con successo, `false` altrimenti.

Buonarroti