Una classe per gestire l'upload

Nel precedente articolo, abbiamo visto le basi della procedura di upload di un file. Oggi svilupperemo una classe robusta e configurabile in grado di gestire questo processo in modo sicuro.
Utilizzeremo delle proprietà avanzate della programmazione ad oggetti in modo da ottenere un lavoro finale di alta qualità. Sei pronto?

Warm up

Iniziamo a progettare la classe nella teoria, si tratterà di una classe astratta (ovvero che dovrà essere estesa obbligatoriamente e che conterrà dei metodi astratti) vedremo subito perché.
Iniziamo ad elencare le proprietà che ci serviranno:

- **error**: sarà un array nel quale inseriremo gli eventuali errori che si sono verificati nel corso del processo.
- **opt**: sarà anch'esso un array che conterrà le opzioni di configurazione.
- **filename**: conterrà il nome del file (sarà una stringa casuale).
- **extension**: conterrà l'estensione del file caricato.
- **file**: valorizzeremo questa proprietà con il contenuto dell'array $_FILES.
- **name**: il valore dell'attributo *name* del form di upload.
- **uploadAbort**: verrà settato a FALSE in fase di inizializzazione, ogni volta che verrà rilevato un errore verrà settato a TRUE. In questo modo, alla fine, sarà semplice sapere se il processo è andato a buon fine.

Abbiamo visto che, per evitare sorprese, è meglio porre alcune limitazioni e fare alcuni controlli. In particolare sulle dimensioni del file e sull'estensione, come pure verificare che il file sia veramente un file caricato da protocollo HTTP POST.
Vediamo dunque i metodi:

- **getExtension**: Con questo metodo andremo a rilevare l'estensione del file che inseriremo nella proprietà *extension*.
- **getSafeName**: Questo metodo andrà a valorizzare la proprietà *filename* creando una stringa casuale e sicura.
- **allowedExtension**: Questo metodo verifica che l'estensione del file caricato faccia parte delle estensioni ammesse.
- **allowedSize**: Questo metodo verificherà che la dimensione del file non superi la dimensione massima ammessa.
- **isUploadedFile**: Grazie a questo metodo potremo verificare che il file sia stato inviato tramite protocollo HTTP POST.
- **parseError**: Questo metodo andrà a verificare una serie di eventuali errori del processo di upload gestiti da PHP e contenuti nell'array $_FILES all'indice error.

- **uploadProcess**: E' il metodo che, una volta superati tutti i controlli, provvederà a spostare il file dalla cartella temporanea a quella di destinazione.
- **onAbort, onSuccess**: Saranno due metodi astratti e verranno invocati rispettivamente in caso di errore o di successo. Li dichiareremo astratti in quanto sono le uniche due cose che possono variare da situazione a situazione. L'unica cosa che sappiamo è che in caso di successo verrà invocato il metodo *onSuccess* ed in caso di errore il metodo *onAbort*. Dichiararli come astratti corrisponde a fare la "promessa" che li implementeremo nell'estensione.

Classe upload - il costruttore di classe

Creiamo un nuovo file e chiamalo *upload.class.php*. Iniziamo a dichiarare la classe (astratta) e prima del costruttore l'elenco delle proprietà così come abbiamo appena visto:

Ora passiamo al costruttore. Prenderà un unico parametro che sarà un array associativo contenente le opzioni di configurazione.

Adesso definiamo le opzioni di default. Ci servono 4 opzioni configurabili:

- *uploadDir*: la cartella di destinazione (default: './')
- *allowedExtensions*: la lista delle estensioni ammesse; utilizzeremo la virgola come separatore (default: false - tutte le estensioni ammesse)
- *name*: il valore dell'attributo name del form di upload (default: 'upload_name').
- *maxSize*: la grandezza massima del file in byte (default: false - nessun limite alla dimensione se non quanto previsto dalla configurazione di php.ini).

A questo punto uniamo l'array che abbiamo passato al costruttore ($options) con l'array $defaults tramite la funzione array_merge().

Questa funzione fa in modo che l'array passato come primo parametro sia modificato da quello passato come secondo in questo modo:

Se in *$options* vengono dichiarati dei valori che esistono in *$defaults* questi vengono riscritti
Se in *$options* vengono dichiarati dei valori che non esistono in *$defualts*, questi vengono aggiunti.
Per il resto fa stato quanto scritto in *$defaults*.

Quindi se volessimo configurare la classe in modo che salvi i file nella cartella *upload* ed in modo che ammetta unicamente le estensioni *png* e *jpg*, dovremmo passare al costruttore questo array:

Gli altri valori resterebbero inalterati rispetto alle impostazioni che abbiamo definito di default.

Continuiamo con il costruttore. Inseriamo nella proprietà file il contenuto dell'array *$_FILES*, in questo modo.

Così facendo evitiamo di scrivere continuamente a che indice facciamo riferimento. Ad esempio per conoscere il nome del file temporaneo non dovremo scrivere:

Ma semplicemente:

Ora inizializziamo *uploadAbort* a *FALSE*

Ed in seguito eseguiamo i metodi *getExtension* e *getSafeName* (che non abbiamo ancora scritto) in modo da valorizzare le proprietà *filename* ed *extension*.

Infine invochiamo il metodo *uploadProcess* (che non abbiamo ancora scritto):

Il nostro costruttore adesso è finito:

Dichiariamo ora i due metodi astratti, semplicemente così:

Come rilevare l'estensione del file

Abbiamo visto che l'indice *type* dell'array *$_FILES* contiene il mime del file caricato. Questa informazione è però generata dal browser (se supportata) e comunque rimane un'informazione lato client quindi non affidabile.
La via migliore mi sembra quella di spezzare il nome del file sui punti ed andare a prendere l'ultimo elemento. In questo modo:

Come vedi ho passato il nome del file per la funzione *explode* inserendo l'array generato nella variabile *$part*. In seguito ho valorizzato la proprietà *extension* con l'ultimo elemento di questo array, cioé l'estensione del file.

Poi passo la stringa alla funzione strtolower. Infatti mi è già capitato di vedere estensioni tipo .JPG o . Jpg; così facendo “normalizziamo” anche l’estensione.

Come creare un nome file sicuro?

Quello che faccio di solito in questo caso è concatenare l’hash del nome file con l’hash del micro timestamp, in questo modo:

Ottenendo una stringa casuale con un bassissimo rischio di collisione.

Come verificare le estensioni ammesse?

Vediamo come sviluppare il metodo che utilizzeremo per verificare se il file caricato ha un’estensione ammessa.

Come prima cosa valutiamo che vi sia un elenco di estensioni ammesse (di default *allowedExtensions* è su false, che significa che tutte le estensioni sono ammesse).
In seguito creo un array (*$ext*) contenente le estensioni ammesse (che abbiamo separato con la virgola).
Quindi verifichiamo se in questo array sia presente l’estensione del file. Se non lo è valorizziamo *uploadAbort* a *TRUE* ed aggiungo all’array *error* il messaggio di errore corrispondente.
Come detto nell’articolo precedente, è importante limitare le estensioni ammesse;
in particolare **sono assolutamente da evitare i files eseguibili**. Bastano poche righe di codice come in questo esempio ed un qualsiasi lamer può cancellare l’intero contenuto di una cartella.

Come verificare che la dimensione del file sia ammessa?

Sviluppiamo ora il metodo che ci permetterà di verificare la dimensione del file. Nell’indice *size* dell’array $_FILES troviamo questa dimensione. In questo caso non sono certo se questo valore viene rilevato lato server o, come nel caso del mime, lato client. Nel dubbio andremo a leggere la dimensione del file temporaneo con la funzione filesize().

Anche in questo caso, se la verifica è negativa, setto uploadAbort a TRUE ed aggiungo all’array *error* il messaggio di errore corrispondente.
Se il parametro *maxSize* dovesse restare FALSE come da default, questo controllo non verrebbe eseguito. Farebbe comunque stato quanto impostato nel php.ini.

Come verificare la legittimità del file?

Come abbiamo visto, la funzione *is_uploaded_file()* ci permette di verificare che il file sia stato inviato tramite protocollo HTTP POST.

Come valutare gli altri errori?

Abbiamo visto che PHP è in grado di rilevare una serie di errori nel processo di upload e di tenerne traccia nell'indice *error* dell'array *$_FILES*.
Quindi andremo a valutare la corrispondenza tra il contenuto dell'indice *error* e le costanti predefinite. In caso di corrispondenza, valorizzeremo *uploadAbort* a *TRUE* ed aggiungeremo il corrispondente messaggio di errore alla proprietà *error*.

Come finalizzare il processo?

A questo punto disponiamo di tutti gli strumenti per scrivere il metodo *uploadProcess* (invocato alla fine del costruttore).
Iniziamo a dichiararlo e a invocare tutti i metodi di controllo, in questo modo:

A questo punto, se non si sono verificati errori, possiamo spostare il file nella cartella di destinazione. Come sappiamo se non vi sono stati errori? Semplice, in assenza di errori, *uploadAbort* sarà *FALSE*.
Dunque procediamo in questo modo:

Dopo avere verificato che *uploadAbort* sia *FALSE*, procediamo allo spostamento utilizzando l'apposita funzione. Se *move_uploaded_file* non dovesse funzionare, è previsto un messaggio di errore da inserire nell'array *error*.

Giunti alla fine, siamo pronti per invocare i metodi astratti, in questo modo:

In questo modo invochiamo il metodo *onAbort* in caso di errori ed il metodo *onSuccess* in caso di riuscita. Questi due metodi sono astratti e non contengono nulla (giustamente, in quanto quello che faremo in caso di successo o di errore dipende dai casi).
Ma essendo *Upload* una classe astratta, siamo costretti ad estenderla e ad implementare questi due metodi.

Conclusione

In questo lungo e fitto articolo abbiamo sviluppato una classe piuttosto avanzata. Questa classe contiene tutto quanto è necessario per gestire un processo di upload. Quello che rimane da fare è estendere la classe e definire le operazioni da svolgere in caso di successo o di errore con la possibilità di disporre di tutti gli strumenti messi a disposizione dalla classe genitore.
E questo è quello che vedremo nel prossimo tutorial.