API Paypal: come implementare un pagamento online? Le procedure specifiche (4/6)

Proseguiamo con la guida riguardante l'implementazione di pagamenti tramite le **API Paypal**. Come avrai avuto modo di leggere nei precedenti articoli, in questa guida stiamo realizzando un'applicazione grazie alla quale é possibile attivare degli account a pagamento (*istant access*, ovvero le risorse sono disponibili immediatamente dopo il pagamento).

Nell'articolo precedente abbiamo concluso la preparazione della classe *IPNListener*, la quale ci fornisce lo strumento per verificare se è possibile procedere (*isReadyTransaction*) all'attivazione dell'account.

Come abbiamo visto questa classe **sarà utilizzabile in qualunque tipo di applicazione che necessita di un pagamento**. Si tratta infatti di una raccolta di tutte le procedure necessarie alla verifica della notifica di pagamento.

Ora dovremo sviluppare il resto, ovvero la classe *YIIListener* che sarà un'estensione di IPN*Listener* e conterrà i metodi che si occuperanno della creazione dell'account come pure della definizione dei metodi astratti contenuti nella classe parent.

API Paypal: La classe YIIListener per gestire il pagamento online

Creiamo il file *YIIListener.php* ed iniziamo ad includere IPN*Listenr.php*, a dichiarare la classe e la proprietà *$conn* che conterrà la risorsa di connessione del database.

Ed ora scriviamo il metodo che provvederà alla connessione al database:

A questo punto implementiamo il metodo astratto *isVerifiedAmmount()*.

Nel nostro caso sarà molto semplice. Come detto abbiamo un solo prodotto ed un solo prezzo.

Verificheremo dunque che il pagato (senza deduzione per le commissioni) corrisponda a quanto abbiamo indicato nella costante *AMMOUNT*.

Ed ora procediamo ad implementare il secondo metodo astratto *isNotProcessed()*, che si occupa di verificare che la transazione non sia già stata processata. Per fare questo controlleremo la presenza dell'id della transazione (txn_id) nel database:

Ora sviluppiamo un piccolo metodo grazie al quale otterremo una password casuale da attribuire al nuovo utente.

Come vedi con questo metodo andiamo a scegliere 10 caratteri casuali compresi tra il carattere 40 ed il carattere 126 del codice ASCII. Partiamo dal 40 in modo da evitare fastidiosi apici e virgolette.

Ora sviluppiamo il metodo *sendLoginData()* resposabile di inviare all'utente i dati del suo nuovo account. Questo metodo sarà invocato poi all'interno di un altro metodo che si occuperà di inserire il nuovo account nel database.

Anche in questo caso, iniziamo con il verificare se ci trovaimo in ambiente di simulazione. In caso affermativo aggiungiamo la parola *simulazione* all'oggetto dell'email sempre per non fare confusione. Inoltre, in simulazione, faremo in modo che l'email venga inviata a noi stessi.

Infatti, l'email dell'utente di Sandbox non esiste, o meglio può ricevere messaggi solo dall'interno.

In ogni caso non è utilizzabile. Dunque, per verificare l'effettivo e corretto invo dell'email con i dati di autenticazione, farò in modo che venga inviata a me. Mentre nell'ambiente di produzione sarà correttamente inviata al pagante (payer_email).

Siamo così giunti alla definizione dell'ultimo metodo (*insertNewUser*) che, dopo aver verificato la bontà della notifica, procederà all'inserimento del nuovo utente nel database ed all'invio dell'email con i dati di autenticazione.

Come prima cosa, chiaramente verificheremo l'esito del metodo *isReadyTransaction()*. Solo se l'esito è positivo, generiamo una password casule con il metodo *getRandPassword()*.

Creiamo l'hash di questa password da inserire nel database.

Scriviamo la query di inserimento e la eseguiamo.

Ed infine inviamo i dati di autenticazione tramite il metodo *sendLoginData()* avendo cura di passare la password (in chiaro ovviamente).

Come vedi non ho fatto nessun escape delle stringhe in entrata. Infatti, a questo punto siamo certi (in quanto lo abbiamo verificato) che i dati ci arrivano da **PayPal** e dunque li considero dati sicuri (non credo proprio che PayPal si diverta ad inviarci delle sql injection).

Non ci resta che instanziare la classe ed invocare questo ultimo metodo. Il risultato finale sarà questo:

Questo è il file al quale dovrà puntare PayPal che avevamo genericamente chiamato *lettoreIPN.php*. Ora modifichiamolo con il nome e percorso corretto così come descritto all'inizio del secondo articolo di questa guida.

Conclusione

Se hai seguito correttamente gli articoli fino ad ora, il tuo sistema di pagamento realizzato sfruttanto le **API PayPal** dovrebbe funzionare senza problemi. Potrai eseguire il pagamento con l'utente *utente* ed il tutto dovrebbe funzionare. Dovrebbe. Nel prossimo articolo procederemo quindi ad un'accurata sessione di test.
Fino ad ora é tutto chiaro? Sei riuscito a seguire e a comprendere i vari passaggi anche se oggettivamente complessi?

Usare le API Paypal: Articoli di questa guida

1. API PayPal: Preparazione
2. API PayPal: Chiarirsi le idee
3. API PayPal: Le procedure generali
4. **API PayPal: Le procedure specifiche**
5. API PayPal: Testare l'applicazione
6. API PayPal: Creare dinamicamente i pulsanti di pagamento