Linguaggi di scripting per il web

Scritto da Febogamedev

Come abbiamo visto, alcuni siti web hanno bisogno di comportamenti dinamici e interattivi, come la compilazione e la verifica di un modulo, l'apertura di un menu o l'avvio di un'animazione in risposta a un'azione dell'utente.

Questi comportamenti vengono realizzati mediante codice che esegue specifiche operazioni. In base al dispositivo sul quale il codice viene eseguito, possiamo distinguere tra:

- Codice lato client (client-side): viene eseguito dal browser sul dispositivo dell'utente. Permette, per esempio, di aggiornare il contenuto della pagina, controllare preliminarmente i dati inseriti in un modulo, mostrare animazioni e reagire immediatamente alle azioni dell'utente (Fielding et al., 2022; WHATWG, n.d.-b).
- Codice lato server (server-side): viene eseguito dal server prima di inviare una risposta al client. Viene utilizzato, per esempio, per accedere a un database, verificare le credenziali di un utente, elaborare i dati inviati attraverso un modulo o svolgere operazioni che non devono essere affidate al dispositivo dell'utente per ragioni di sicurezza (Fielding et al., 2022; PHP Documentation Group, 2026).

Il codice lato client e quello lato server possono collaborare: il browser raccoglie una richiesta dell'utente e la invia al server; il server la elabora e restituisce al browser i dati o il risultato dell'operazione (Fielding et al., 2022).

Lato client

JavaScript è un linguaggio di programmazione utilizzato nel Web per definire il comportamento delle pagine e delle applicazioni (Ecma International, 2026; WHATWG, n.d.-b).
Attraverso JavaScript è possibile, per esempio:

- Reagire alle azioni dell'utente.
- Leggere e modificare la struttura della pagina.
- Mostrare o nascondere elementi.
- Controllare i dati inseriti nei moduli.
- Aggiornare parti dell'interfaccia.
- Richiedere dati a un server.
- Gestire animazioni e comportamenti complessi.
- Conservare temporaneamente informazioni nel browser.

I file che contengono codice JavaScript utilizzano normalmente l'estensione .js (WHATWG, n.d.-b).

JavaScript non viene utilizzato soltanto nei browser: può essere eseguito anche su server e in altri ambienti software. In questo capitolo ci concentreremo però sul suo utilizzo lato client, cioè all'interno del browser (Ecma International, 2026).

Il nucleo del linguaggio JavaScript è standardizzato dalla specifica **ECMAScript** (Ecma International, 2026).

ECMAScript definisce elementi fondamentali come variabili, valori, funzioni, oggetti, condizioni, cicli, operazioni matematiche e gestione degli errori (Ecma International, 2026).
Il browser aggiunge al linguaggio una serie di strumenti chiamati **Web API**, attraverso i quali il programma può interagire con la pagina, con la rete e con le funzioni del dispositivo (WHATWG, n.d.-a, n.d.-b).
Tra le Web API troviamo: DOM, gestione degli eventi, richieste di rete, timer, archiviazione locale e riproduzione multimediale (WHATWG, n.d.-a, n.d.-b).

Un documento HTML può collegare un file JavaScript attraverso l'elemento <script> (WHATWG, n.d.-b):
<script src="script.js"></script>

- L'attributo src indica l'URL dal quale recuperare il programma.
- Il percorso script.js è un URL relativo. Viene risolto rispetto all'URL di base del documento HTML (WHATWG, n.d.-b).

Una soluzione comune consiste nell'inserire lo script nel head utilizzando l'attributo defer:

```
<head>
  <script src="script.js" defer></script>
</head>
```

defer permette al browser di recuperare il file mentre continua a interpretare il documento e ne rimanda l'esecuzione fino a quando la struttura HTML non è stata analizzata (WHATWG, n.d.-b).
Il *DOM* mette a disposizione interfacce, proprietà e metodi attraverso cui JavaScript può

- Cercare elementi;
- Leggerne il contenuto;
- Modificarne il testo;
- Cambiare gli attributi;
- Aggiungere o rimuovere elementi;
- Applicare o rimuovere classi;
- Reagire agli eventi (WHATWG, n.d.-a; World Wide Web Consortium [W3C], 2026).

Per esempio:

```
const titolo = document.querySelector("h1");
titolo.textContent = "Nuovo titolo";
```

- La prima istruzione "document.querySelector("h1")" cerca nel DOM il primo elemento che corrisponde al selettore h1. Il riferimento all'elemento viene conservato nella costante "titolo" (WHATWG, n.d.-a).
- La seconda istruzione "titolo.textContent = "Nuovo titolo";" modifica il contenuto testuale dell'elemento (WHATWG, n.d.-a).

Il file HTML originale non viene modificato sul server: cambia soltanto il DOM presente in quel momento nel browser (WHATWG, n.d.-a).

Un evento è una segnalazione che indica che si è verificato qualcosa (W3C, 2026).
Alcuni esempi sono riportati nella tabella seguente.

Evento	Quando si verifica
click	Si verifica quando l'utente seleziona un elemento, normalmente con un clic del mouse, un tocco o un comando equivalente da tastiera. È comune su pulsanti e collegamenti.
input	Si verifica ogni volta che il valore di un campo viene modificato dall'utente. In un campo di testo può attivarsi a ogni carattere inserito o cancellato.
change	Si verifica quando il valore di un controllo viene modificato e la modifica viene confermata. Nei campi di testo avviene normalmente quando il campo perde il focus; nelle caselle di selezione può avvenire subito.
submit	Si verifica quando viene inviato un elemento <form>, per esempio premendo il relativo pulsante di invio o il tasto Invio in un campo.
load	Si verifica quando una risorsa ha terminato il caricamento. Può riguardare la pagina, un'immagine, uno script o un'altra risorsa.
keydown	Si verifica quando l'utente preme un tasto della tastiera. Può ripetersi automaticamente se il tasto viene mantenuto premuto.

Gli eventi possono essere generati dalle azioni dell'utente oppure dal browser e dalle altre componenti della piattaforma Web (WHATWG, n.d.-b; W3C, 2026).
JavaScript può registrare una funzione che deve essere eseguita quando si verifica un determinato evento (WHATWG, n.d.-a).

Per esempio:

```
const pulsante = document.querySelector("button");

pulsante.addEventListener("click", mostraMessaggio);

function mostraMessaggio() {
```

```
  console.log("Il pulsante è stato selezionato");
}
```

- Viene salvato il riferimento al primo elemento <button> presente nel DOM
- A questo riferimento viene associata una funzione all'evento click.
- "mostraMessaggio" è la funzione associata all'evento.
- La funzione viene eseguita quando l'evento raggiunge il pulsante.

La funzione associata a un evento viene chiamata **event listener**, **event handler** o **funzione di callback** (WHATWG, n.d.-a).

Realizziamo una pagina che calcola automaticamente l'importo totale partendo da:

- Prezzo unitario.
- Quantità.

Il risultato deve aggiornarsi ogni volta che l'utente modifica uno dei due valori.
Di seguito è riportato il codice HTML corrispondente:

```
<!doctype html>
<html lang="it">

<head>
  <meta charset="utf-8">
  <meta
    name="viewport"
    content="width=device-width, initial-scale=1">

  <title>Calcolo dell'importo totale</title>

  <script src="script.js" defer></script>
</head>

<body>
  <main>
    <h1>Calcolo dell'importo totale</h1>

    <form id="calcolo-totale">
      <p>
        <label for="price">
          Prezzo unitario:
        </label>

        <input
          type="number"
          id="price"
          name="price"
          min="0"
```

```
          step="0.01"
          required>
      </p>

      <p>
        <label for="quantity">
          Quantità:
        </label>

        <input
          type="number"
          id="quantity"
          name="quantity"
          min="0"
          step="1"
          required>
      </p>

      <p>
        Importo totale:

        <output
          id="totalAmount"
          for="price quantity">
          --
        </output>
      </p>
    </form>
  </main>
</body>

</html>
```

Il documento contiene:

- Un modulo <form>.
- Due campi <input>.
- Due etichette <label>.
- Un elemento <output> per il risultato.
- Il collegamento al file script.js.

Gli attributi “for="price"” e “id="price"” mettono in relazione l’etichetta e il campo (WHATWG, n.d.-b).
Gli attributi min, step e required definiscono alcuni vincoli sui valori accettati (WHATWG, n.d.-b).
L’elemento:

```
<output
  id="totalAmount"
  for="price quantity">
```

rappresenta un risultato calcolato utilizzando i campi price e quantity.
Lo standard *HTML* prevede esplicitamente l'elemento output per rappresentare risultati di calcoli associati ai controlli di un modulo (WHATWG, n.d.-b).
Di seguito è riportato lo script JavaScript:

```
const form = document.querySelector("#calcolo-totale");
const priceInput = document.querySelector("#price");
const quantityInput = document.querySelector("#quantity");
const totalOutput = document.querySelector("#totalAmount");

const currencyFormatter = new Intl.NumberFormat("it-IT", {
  style: "currency",
  currency: "EUR",
});

function updateTotal() {
  const price = priceInput.valueAsNumber;
  const quantity = quantityInput.valueAsNumber;

  const valuesAreValid =
    form.checkValidity() &&
    Number.isFinite(price) &&
    Number.isFinite(quantity);

  if (!valuesAreValid) {
    totalOutput.value = "--";
    return;
  }

  const total = price * quantity;
  totalOutput.value = currencyFormatter.format(total);
}

form.addEventListener("input", updateTotal);
```

Le prime righe cercano gli elementi nel DOM:

```
const form = document.querySelector("#calcolo-totale");
const priceInput = document.querySelector("#price");
const quantityInput = document.querySelector("#quantity");
const totalOutput = document.querySelector("#totalAmount");
```

Il simbolo # indica che il selettore cerca un elemento attraverso il suo id (WHATWG, n.d.-a).

Per esempio “document.querySelector("#price")” cerca l’elemento con id equivalente a “price”.

I riferimenti vengono conservati attraverso const, perché durante l’esecuzione continueranno a indicare gli stessi elementi (Ecma International, 2026).

Per leggere i valori numerici vengono utilizzate le seguenti istruzioni:

```
const price = priceInput.valueAsNumber;
const quantity = quantityInput.valueAsNumber;
```

La proprietà valueAsNumber restituisce il valore del campo interpretato come numero. Quando il valore non può essere convertito correttamente, restituisce NaN, cioè Not a Number (Ecma International, 2026; WHATWG, n.d.-b).
L’HTML Standard definisce valueAsNumber per i controlli numerici (WHATWG, n.d.-b).

In seguito lo script controlla i vincoli definiti dall’HTML:

- Che il prezzo sia un numero finito.
- Che la quantità sia un numero finito.
- Che il form abbia rispettato i vincoli (WHATWG, n.d.-b).

```
const valuesAreValid =
    form.checkValidity() &&
    Number.isFinite(price) &&
    Number.isFinite(quantity);
```

Quando i valori non sono validi il risultato viene sostituito da due trattini e l’esecuzione della funzione termina.

```
if (!valuesAreValid) {
  totalOutput.value = "--";
  return;
}
```

Verificati i due input, viene eseguita l’elaborazione vera e propria: il prezzo viene moltiplicato per la quantità.

```
const total = price * quantity;
totalOutput.value = currencyFormatter.format(total);
```

Da notare che il risultato viene formattato come valuta italiana (in euro) e assegnato all’elemento <output>.

Quello che abbiamo appena descritto è la funzione “updateTotal”, che è collegata all’evento input dell'elemento con id “calcolo-totale”, salvato nel riferimento form (WHATWG, n.d.-a; W3C, 2026).

```
form.addEventListener("input", updateTotal);
```

L’evento input viene generato quando l’utente modifica il valore di un controllo (W3C, 2026).
La funzione “updateTotal” viene quindi richiamata a ogni modifica di prezzo o quantità.
Il calcolo avviene realmente in tempo reale, senza bisogno di premere un pulsante.

La sequenza dell’interazione è questa:

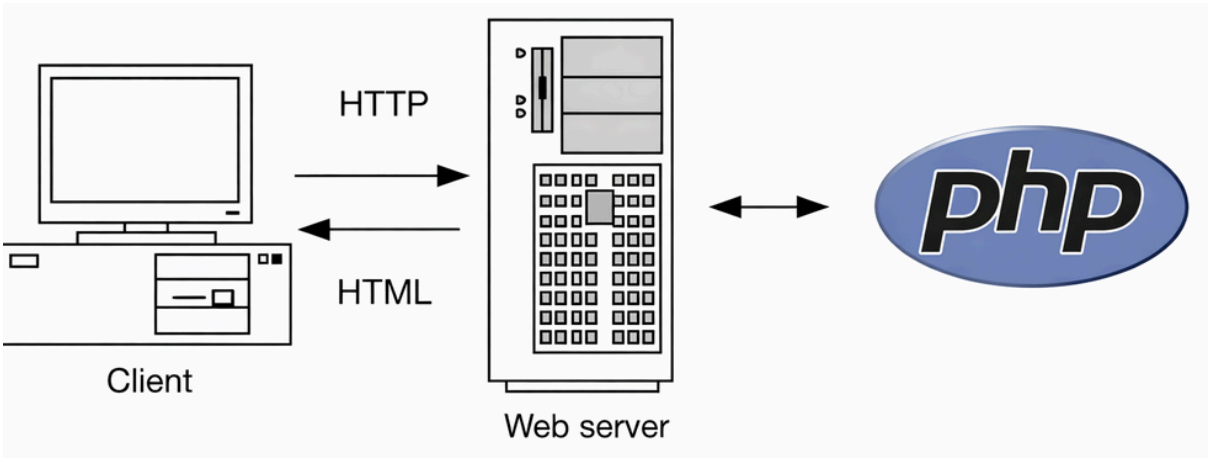
```
L'utente modifica un campo
          ↓
il browser genera l'evento input
          ↓
viene eseguita updateTotal()
          ↓
JavaScript legge i valori dal DOM
          ↓
calcola il totale
          ↓
aggiorna l'elemento output
          ↓
il browser mostra il nuovo risultato
```

Lato server

PHP, acronimo di Hypertext Preprocessor, è un linguaggio di scripting particolarmente utilizzato per sviluppare applicazioni web lato server (PHP Documentation Group, 2026).

Questo significa che Il codice *PHP* viene eseguito nell’ambiente del server e che .il browser non interpreta direttamente il codice *PHP*, ma riceve soltanto il risultato, per esempio (Fielding et al., 2022; PHP Documentation Group, 2026):

- Un documento *HTML*.
- Dati in formato *JSON*.
- Un file.
- Un reindirizzamento.
- Un codice di stato *HTTP*.
- Una risposta senza contenuto.



PHP può essere inserito all'interno di documenti contenenti *HTML* oppure utilizzato per costruire applicazioni composte da molti file e componenti. Il manuale ufficiale prevede i delimitatori <?php ... ?> e la forma abbreviata <?= ... ?> per produrre un valore nell'output (PHP Documentation Group, 2026).

I file che contengono codice *PHP* utilizzano normalmente l'estensione .php (PHP Documentation Group, 2026).
Questa estensione permette spesso al server di riconoscere che il file deve essere elaborato dall'ambiente *PHP* anziché inviato direttamente al browser.

Il percorso di un *URL* non deve corrispondere al nome o alla posizione di un file fisico presente sul server (Fielding et al., 2022).
In un'applicazione web, il server può inoltrare la richiesta a un programma *PHP* che interpreta il percorso e decide quale operazione eseguire.
Per esempio, *l'URL "*https://example.com/profilo" può attivare la funzione che recupera dal database i dati dell'utente e genera la pagina del profilo, anche se non esiste un file chiamato profilo.php.
Le regole che associano i percorsi *URL* alle funzioni dell'applicazione sono chiamate **route**.
Spesso un unico file, come index.php, riceve molte richieste differenti e le indirizza verso la parte corretta dell'applicazione.

L'applicazione PHP può:

- Ricevere informazioni dalla richiesta.
- Controllare e interpretare i dati.
- Eseguire calcoli o altre operazioni.
- Comunicare con un database.
- Decidere il codice di stato della risposta.
- Produrre HTML, JSON o altri contenuti.
- Inviare la risposta al client.

Nell'immagine sottostante è rappresentata in maniera semplificata l'interazione tra browser e PHP.

1 BROWSER



HTTP request

The browser sends an HTTP request to the server to obtain a resource (e.g. a web page).

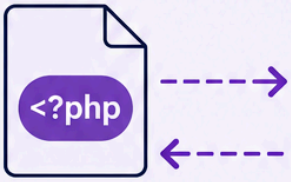
2 WEB SERVER



The web server receives the HTTP request and forwards it to the PHP application (if necessary).

3 PHP APPLICATION

The PHP application processes the request.





optional communication with a database or other services

The application can query a database or communicate with other external services to retrieve or send data.

4 PROCESSING



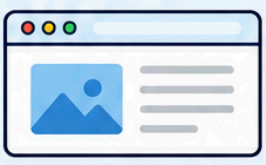
The application processes the data and prepares the result to be returned to the client.



HTTP response

The application sends the HTTP response to the web server (e.g. an HTML page, a JSON file, a redirect, etc.).

5 BROWSER



The browser receives the HTTP response and displays the content to the user.

PHP mette a disposizione variabili predefinite per accedere alle diverse informazioni provenienti dalla richiesta, come query string, dati *POST*, cookie e file caricati (PHP Documentation Group, 2026).

Supponiamo che il server possieda questo file:

```php
<?php

$nome = $_GET["nome"] ?? "visitatore";

$nomeSicuro = htmlspecialchars(
    $nome,
    ENT_QUOTES,
    "UTF-8"
);

?>

<!doctype html>
<html lang="it">

<head>
  <meta charset="utf-8">
  <title>Saluto</title>
</head>

<body>
  <h1>Ciao, <?= $nomeSicuro ?>!</h1>
</body>

</html>
```

L'utente visita: https://example.com/saluto.php?nome=Anna.

PHP legge il parametro della query string attraverso "$_GET["nome"]" e produce un documento HTML contenente: <h1>Ciao, Anna!</h1> (PHP Documentation Group, 2026)

Il browser riceve il risultato *HTML*, non il codice PHP originale. Nello specifico, ricevo questo (Fielding et al., 2022; PHP Documentation Group, 2026).

```html
<!doctype html>
<html lang="it">

<head>
  <meta charset="utf-8">
  <title>Saluto</title>
</head>
```

```
<body>
  <h1>Ciao, Anna!</h1>
</body>

</html>
```

Nota:
La funzione htmlspecialchars() converte determinati caratteri speciali nelle corrispondenti entità/rappresentazioni HTML, in modo che vengano trattati come testo nel contesto appropriato (PHP Documentation Group, 2026).

converte i caratteri che possiedono un significato particolare in HTML, così il testo inserito dall'utente può essere incluso più correttamente nel documento. L'operazione da eseguire sui dati dipende comunque dal contesto: la preparazione per l'HTML è diversa dalla preparazione per SQL, URL o altri formati (OWASP Foundation, n.d.; PHP Documentation Group, 2026).

PHP può accedere a dati provenienti da varie parti della richiesta tramite questi fonti:

- Query string
- Corpo di una richiesta POST
- Cookie e sessioni

Query string
Con un *URL* come "https://example.com/prodotti?categoria=libri"
PHP può leggere il valore attraverso "$_GET["categoria"]"
$_GET contiene i parametri presenti nella query string, indipendentemente dal metodo HTTP utilizzato (PHP Documentation Group, 2026).

Corpo di una richiesta POST
Quando un modulo viene inviato tramite POST:

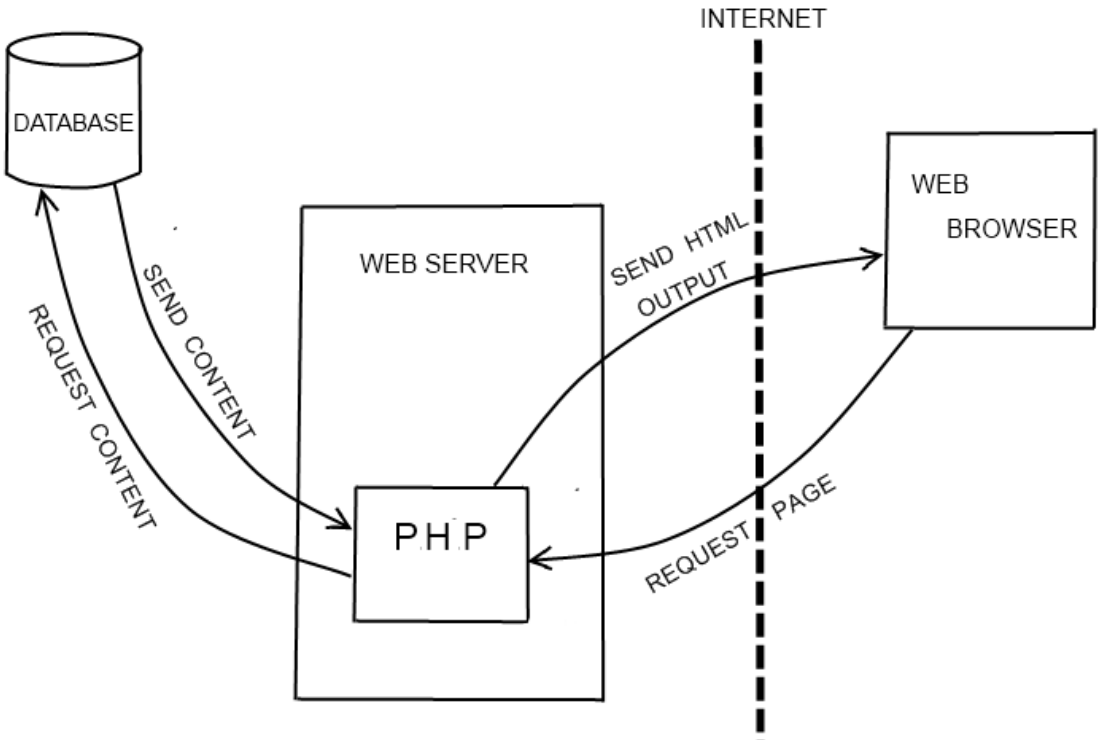
```
<form method="post" action="/login">
  ...
</form>
```

i valori dei normali campi possono essere resi disponibili attraverso "$_POST" (PHP Documentation Group, 2026).

Il corpo potrebbe però contenere anche altri formati, come JSON, che richiedono sistemi differenti di lettura e interpretazione (Fielding et al., 2022).

Cookie e sessioni
I cookie inviati dal browser possono essere letti attraverso "$_COOKIE" (PHP Documentation Group, 2026).

PHP offre inoltre un sistema di sessioni che permette di conservare dati associati a un utente tra richieste successive. Il browser conserva normalmente un identificatore di sessione in un cookie, mentre i dati effettivi della sessione vengono gestiti dal server (PHP Documentation Group, 2026).



Una parte importante di *PHP* è la sua interazione con il database.

SQL, acronimo di Structured Query Language, è un linguaggio standardizzato per operare sui dati e sulle strutture dei database relazionali (International Organization for Standardization [ISO], 2023).
Permette di eseguire operazioni come:

- Creare tabelle.
- Definire vincoli.
- Inserire dati.
- Cercare dati.
- Modificare dati.
- Eliminare dati.
- Mettere in relazione informazioni provenienti da più tabelle.

Supponiamo di possedere una tabella chiamata studenti.
Di seguito è riportata una query SQL:

```sql
SELECT nome
FROM studenti
WHERE classe = '3C';
```

L’istruzione chiede al *DBMS* di restituire il valore della colonna nome per le righe nelle quali la colonna classe ha valore 3C (ISO, 2023).
Alcune istruzioni comuni dell’*SQL* sono:

- SELECT: Per leggere dati.
- INSERT: Per inserire nuove righe.
- UPDATE: Per modificare dati esistenti.
- DELETE: Per eliminare righe.
- CREATE TABLE: per definire una nuova tabella (ISO, 2023).

MySQL è un sistema di gestione di database relazionali (Oracle, 2026).
MySQL utilizza *SQL* per permettere alle applicazioni e agli amministratori di definire, leggere e modificare i dati (ISO, 2023; Oracle, 2026).
PHP può comunicare con *MySQL* attraverso interfacce come *PDO* e *MySQLi* (Oracle, 2026; PHP Documentation Group, 2026).
L’applicazione invia un’istruzione al *DBMS*, riceve il risultato e decide come utilizzarlo nella risposta *HTTP* (Fielding et al., 2022; Oracle, 2026; PHP Documentation Group, 2026).

Sotto si vede un esempio:
Supponiamo di avere un database chiamato scuola, contenente la tabella studenti con campi: id, nome e classe.
La tabella contiene quindi questi valori:

id	nome	classe
1	Giovanni	3C
2	Michela	3C
3	Carmela	3C
4	Marco	4A

Creiamo il file studenti.php:

```php
<?php

// Dati necessari per collegarsi al database.
$host = "localhost";
$database = "scuola";
$username = "utente_app";
$password = "password";

// Il DSN descrive il database al quale vogliamo collegarci.
$dsn = "mysql:host=$host;dbname=$database;charset=utf8mb4";

try {
    // Creazione della connessione al database.
    $pdo = new PDO(
        $dsn,
```

```
        $username,
        $password,
        [
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
            PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
            PDO::ATTR_EMULATE_PREPARES => false,
        ]
    );

    // Legge la classe dalla query string.
    // Se il parametro manca, utilizza 3C.
    $classe = $_GET["classe"] ?? "3C";

    // Prepara l'istruzione SQL.
    $istruzione = $pdo->prepare(
        "SELECT nome, classe
         FROM studenti
         WHERE classe = :classe
         ORDER BY nome"
    );

    // Sostituisce :classe con il valore ricevuto.
    $istruzione->execute([
        "classe" => $classe,
    ]);

    // Recupera tutte le righe restituite dal DBMS.
    $studenti = $istruzione->fetchAll();

} catch (PDOException $errore) {
    http_response_code(500);
    exit("Non è stato possibile recuperare gli studenti.");
}

?>

<!doctype html>
<html lang="it">

<head>
  <meta charset="utf-8">
  <title>Studenti della classe</title>
</head>

<body>
  <h1>
```

```
    Studenti della classe
    <?= htmlspecialchars($classe, ENT_QUOTES, "UTF-8") ?>
  </h1>

  <?php if (count($studenti) === 0): ?>

    <p>Non è stato trovato alcuno studente.</p>

  <?php else: ?>

    <ul>
      <?php foreach ($studenti as $studente): ?>
        <li>
          <?= htmlspecialchars(
              $studente["nome"],
              ENT_QUOTES,
              "UTF-8"
          ) ?>
        </li>
      <?php endforeach; ?>
    </ul>

  <?php endif; ?>
</body>

</html>
```

PDO permette a *PHP* di creare una connessione a *MySQL* attraverso il relativo driver (Oracle, 2026; PHP Documentation Group, 2026).
La connessione viene realizzata creando un oggetto *PDO* e fornendo il *DSN*, il nome utente e la password (PHP Documentation Group, 2026).

L'utente potrebbe visitare: https://example.com/studenti.php?classe=3C
La query string contiene: classe=3C.
PHP recupera il valore attraverso: $classe = $_GET["classe"] ?? "3C".
La variabile $classe contiene quindi: 3C.

Ricevuti gli input, *PHP* prepara la query da mandare al *DBMS* con questa istruzione:

```
$istruzione = $pdo->prepare(
    "SELECT nome, classe
     FROM studenti
     WHERE classe = :classe
     ORDER BY nome"
);
```

Il parametro ":classe" è un segnaposto (PHP Documentation Group, 2026).

Il suo valore viene fornito separatamente:

```
$istruzione->execute([
    "classe" => $classe,
]);
```

L'istruzione equivale concettualmente a chiedere:

```
SELECT nome, classe
FROM studenti
WHERE classe = '3C'
ORDER BY nome;
```

Dopo l'esecuzione della query, il DBMS restituisce a PHP le righe corrispondenti (Oracle, 2026; PHP Documentation Group, 2026):

- Carmela.
- Giovanni.
- Michela.

Queste righe vengono conservate nella variabile:

```
$studenti = $istruzione->fetchAll();
```

Questa variabile può essere immaginata come una raccolta di dati:

```
[
    [
        "nome" => "Carmela",
        "classe" => "3C",
    ],
    [
        "nome" => "Giovanni",
        "classe" => "3C",
    ],
    [
        "nome" => "Michela",
        "classe" => "3C",
    ],
]
```

PHP attraversa i risultati mediante:

```
<?php foreach ($studenti as $studente): ?>
```

e produce un elemento <li> per ogni studente (PHP Documentation Group, 2026).

Infine il browser riceve il documento HTML prodotto dall'esecuzione del codice *PHP* (Fielding et al., 2022; PHP Documentation Group, 2026):

```html
<!doctype html>
<html lang="it">

<head>
  <meta charset="utf-8">
  <title>Studenti della classe</title>
</head>

<body>
  <h1>Studenti della classe 3C</h1>

  <ul>
    <li>Carmela</li>
    <li>Giovanni</li>
    <li>Michela</li>
  </ul>
</body>
</html>
```

Materiale di supporto

Esercizi

Client o server?
Una piattaforma di e-commerce deve svolgere queste operazioni:

- Mostrare immediatamente un menu quando l'utente preme un pulsante.
- Calcolare il totale provvisorio del carrello mentre l'utente modifica la quantità.
- Verificare che il campo e-mail di un modulo abbia un formato accettabile.
- Controllare che la password inserita corrisponda realmente all'account dell'utente.
- Verificare nel database la disponibilità di un prodotto.
- Aggiornare graficamente il numero di prodotti mostrato nel carrello.
- Registrare definitivamente un ordine.
- Mostrare un'animazione quando il pagamento è stato completato.

Per ciascuna operazione:

- Stabilisci se dovrebbe essere eseguita principalmente lato client, lato server oppure attraverso la collaborazione di entrambi.
- Motiva ogni scelta.
- Spiega perché alcune verifiche possono essere effettuate nel browser per migliorare l'esperienza dell'utente ma devono comunque essere ripetute sul server.
- Spiega perché non sarebbe sicuro affidare al solo JavaScript del browser la verifica delle credenziali o l'autorizzazione di un pagamento.
- Individua quali operazioni potrebbero richiedere l'accesso a un database.
- Ricostruisci infine il flusso completo di un acquisto: Utente → JavaScript → richiesta HTTP → applicazione server → database → risposta → aggiornamento dell'interfaccia.

Rendere interattivo un preventivo
Una pagina contiene due campi:

```
<input type="number" id="prezzo">
<input type="number" id="quantita">

<output id="totale">--</output>
```

Il risultato deve aggiornarsi automaticamente ogni volta che l'utente modifica uno dei due valori.

Progetta concettualmente lo script JavaScript necessario.

- Indica come JavaScript può individuare i tre elementi nel DOM.
- Spiega quale evento utilizzeresti per rilevare immediatamente le modifiche dei campi.
- Descrivi cosa deve fare la funzione associata all'evento.
- Spiega perché è necessario controllare che i valori inseriti siano effettivamente numerici e validi.
- Se prezzo = 24.50 e quantità = 3, indica quale risultato dovrebbe essere mostrato.

- Ricostruisci la sequenza: Modifica dell'utente → evento → event listener → lettura del DOM → calcolo → modifica del DOM → aggiornamento grafico.
- Spiega perché il file HTML originale presente sul server non viene modificato quando JavaScript cambia il contenuto dell'elemento <output>.
- Spiega quale vantaggio offre l'attributo defer quando il file JavaScript viene collegato all'interno di <head>.
- Immagina che lo script venga eseguito prima che gli elementi HTML esistano nel DOM. Spiega quale problema potrebbe verificarsi.
- Indica quali strumenti utilizzati nell'esercizio appartengono al linguaggio ECMAScript e quali invece sono forniti dalle Web API del browser.

Dal modulo alla risposta PHP
Un sito permette di accedere alla propria area personale attraverso questo modulo:

```
<form method="post" action="/login">
    <input type="email" name="email">
    <input type="password" name="password">
    <button type="submit">Accedi</button>
</form>
```

Quando l'utente invia il modulo, la richiesta viene gestita da un'applicazione PHP.

- Spiega dove viene interpretato il codice HTML ed eseguito JavaScript della pagina e dove viene eseguito invece il codice PHP.
- Indica attraverso quale struttura PHP possono essere normalmente letti i dati dei campi inviati tramite POST.
- Spiega perché il browser non riceve il codice PHP originale.
- Indica quali risultati potrebbe produrre PHP dopo l'elaborazione, per esempio HTML, JSON, reindirizzamento o codice di stato HTTP.
- Supponi che l'URL /login venga gestito da index.php, anche se non esiste alcun file login.php. Spiega come ciò sia possibile utilizzando il concetto di route.
- Dopo l'accesso corretto, il server deve ricordare l'identità dell'utente durante le richieste successive. Spiega come possono collaborare cookie e sessione.
- Spiega perché un dato proveniente dall'utente deve essere trattato in modo appropriato prima di essere inserito nell'HTML prodotto dal server.
- Spiega perché htmlspecialchars() è appropriato per un certo contesto HTML ma non rappresenta automaticamente una protezione adeguata quando lo stesso dato viene utilizzato in SQL.
- Ricostruisci il processo: Compilazione del modulo → richiesta POST → PHP → verifica dei dati → eventuale sessione → produzione della risposta → browser.

Dal database alla pagina Web
Una scuola possiede questa tabella:

id	nome	classe
1	Giovanni	3C

2	Michela	3C
3	Carmela	3C
4	Marco	4A

Un utente visita: https://example.com/studenti?classe=3C

L'applicazione PHP deve mostrare soltanto gli studenti della classe richiesta.

- Spiega come PHP può recuperare il valore 3C dalla query string.
- Scrivi concettualmente la query SQL necessaria per ottenere nome e classe degli studenti della 3C.
- Indica quale funzione svolge SELECT.
- Spiega la differenza tra SQL e MySQL.
- Spiega quale ruolo può svolgere PDO tra PHP e MySQL.
- Spiega perché è preferibile utilizzare una query parametrizzata con un segnaposto come :classe invece di inserire direttamente nella query il valore ricevuto dall'utente.
- Descrivi cosa restituisce il DBMS all'applicazione PHP.
- Spiega come PHP può utilizzare i risultati per generare una lista HTML composta da elementi <li>.
- Spiega perché il browser riceverà il documento HTML risultante e non la query SQL né il codice PHP che l'ha generato.
- Immagina che la richiesta sia invece "https://example.com/studenti?classe=5B" e che non esista alcuno studente della 5B. Proponi quale risposta dovrebbe produrre l'applicazione.
- Immagina infine che il database non sia raggiungibile. Spiega perché potrebbe essere appropriato restituire un errore HTTP lato server.
- Ricostruisci l'intero flusso: Browser → richiesta HTTP → applicazione/router PHP → PDO → MySQL, che esegue la query SQL → risultati → PHP → HTML → risposta HTTP → browser.

Bibliografia

Ecma International. (2026). *ECMAScript® 2026 language specification* (ECMA-262, 17th ed.). https://ecma-international.org/publications-and-standards/standards/ecma-262/

Fielding, R., Nottingham, M., & Reschke, J. (2022). *HTTP semantics* (RFC 9110). RFC Editor. https://www.rfc-editor.org/info/rfc9110

International Organization for Standardization. (2023). *Information technology—Database languages SQL—Part 1: Framework* (ISO/IEC Standard No. 9075-1:2023). https://www.iso.org/standard/76583.html

Oracle. (2026). *MySQL 8.4 Reference Manual*. https://dev.mysql.com/doc/refman/8.4/en/

OWASP Foundation. (n.d.). *Cross Site Scripting Prevention Cheat Sheet*. Retrieved September 2, 2026, from https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

PHP Documentation Group. (2026). *PHP manual*. https://www.php.net/manual/en/

WHATWG. (n.d.-a). *DOM Standard*. Retrieved September 2, 2026, from https://dom.spec.whatwg.org/

WHATWG. (n.d.-b). *HTML Standard*. Retrieved September 2, 2026, from https://html.spec.whatwg.org/

World Wide Web Consortium. (2026). *UI Events*. https://www.w3.org/TR/uievents/

Approfondimenti

- Jon Duckett, JavaScript & jQuery. Sviluppare interfacce web interattive, Apogeo, 2017 — Introduce JavaScript con un approccio fortemente visuale e affronta interattività, gestione degli eventi, manipolazione della pagina, validazione dei moduli, Ajax, API e JSON. È particolarmente adatto per approfondire la parte della dispensa dedicata a JavaScript lato client, DOM, eventi e comunicazione con il server. La sezione dedicata a jQuery è oggi meno centrale rispetto al JavaScript moderno, ma la parte fondamentale sul linguaggio resta didatticamente utile.
- David Flanagan, JavaScript. La guida definitiva. Dalle basi del linguaggio alle tecniche avanzate, 7ª ed., Apogeo, 2021 — Costituisce un approfondimento molto più completo del linguaggio JavaScript ed esamina tipi, variabili, funzioni, oggetti, classi, moduli, programmazione asincrona e API del browser. È il testo più indicato per passare dall'introduzione della dispensa a una conoscenza più rigorosa di ECMAScript e della programmazione JavaScript moderna.
- Jon Duckett, PHP & MySQL. Sviluppare applicazioni web lato server, Apogeo, 2022 — È probabilmente il libro più direttamente collegato alla seconda metà della dispensa. Spiega PHP, generazione dinamica delle pagine, elaborazione dei dati provenienti dalle richieste, utenti, sessioni, caricamento dei file e utilizzo di database MySQL. L'approccio visuale rende particolarmente chiaro il flusso browser → server → PHP → database → risposta.
- Serena Sensini, Basi di dati. Tecnologie, architetture e linguaggi per database, Apogeo, 2021 — Approfondisce la parte relativa a SQL e database relazionali, spiegando progettazione dei dati, SELECT, INSERT, UPDATE, DELETE, MySQL e PostgreSQL, oltre a tecnologie NoSQL. È quindi utile per sviluppare ciò che nella dispensa viene introdotto attraverso l'interazione tra PHP, PDO, MySQL e query SQL.

Nota editoriale

Per la realizzazione di questi materiali ho utilizzato strumenti di intelligenza artificiale generativa come supporto alla scrittura, in particolare per migliorare la forma del testo, riorganizzare i contenuti, correggere la formulazione e velocizzare alcune attività redazionali.

Lavorando autonomamente alla produzione di questi materiali, cerco infatti di automatizzare tutte le attività che possono esserlo, così da poter dedicare più tempo alla ricerca, alla progettazione e allo sviluppo dei contenuti.

L'intelligenza artificiale non determina però i contenuti dell'opera: la scelta degli argomenti, la struttura, le idee, le interpretazioni, gli esempi e l'impostazione didattica sono elaborati da me. L'IA viene quindi utilizzata principalmente come strumento di supporto alla produzione e alla revisione formale, mentre la responsabilità autoriale e progettuale dei contenuti rimane mia.

Salvo diversa indicazione, questo materiale è distribuito con licenza Creative Commons Attribuzione – Non commerciale – Condividi allo stesso modo 4.0 Internazionale (CC BY-NC-SA 4.0).

È quindi possibile condividere, ridistribuire, modificare e creare opere derivate a partire dal materiale, a condizione di attribuirne correttamente la paternità, di non utilizzarlo per scopi commerciali e di distribuire eventuali versioni modificate o derivate con la stessa licenza.

