# Web Scripting Languages

**Written by [Febogamedev]**

As we have seen, some websites need dynamic and interactive behavior, such as filling in and validating a form, opening a menu or starting an animation in response to a user action.

This behavior is implemented through code that performs specific operations. Depending on the device on which the code runs, we can distinguish between:

- Client-side code (client-side): runs in the browser on the user's device. It allows, for example, page content to be updated, preliminary checks on data entered in a form, animations to be displayed and immediate responses to user actions (Fielding et al., 2022; WHATWG, n.d.-b).
- Server-side code (server-side): runs on the server before a response is sent to the client. It is used, for example, to access a database, check a user's credentials, process data submitted through a form or perform operations that must not be entrusted to the user's device for security reasons (Fielding et al., 2022; PHP Documentation Group, 2026).

Client-side and server-side code can work together: the browser collects a user request and sends it to the server; the server processes it and returns the data or the result of the operation to the browser (Fielding et al., 2022).

## Client Side

**JavaScript** is a programming language used on the Web to define the behavior of pages and applications (Ecma International, 2026; WHATWG, n.d.-b).
JavaScript can be used, for example, to:

- Respond to user actions.
- Read and modify the page structure.
- Show or hide elements.
- Check data entered in forms.
- Update parts of the interface.
- Request data from a server.
- Manage animations and complex behavior.
- Temporarily store information in the browser.

Files containing JavaScript code normally use the .js extension (WHATWG, n.d.-b).

JavaScript is not only used in browsers: it can also run on servers and in other software environments. In this chapter, however, we will focus on its client-side use, that is, within the browser (Ecma International, 2026).

The core of the JavaScript language is standardized by the **ECMAScript** specification (Ecma International, 2026).

ECMAScript defines fundamental elements such as variables, values, functions, objects, conditions, loops, mathematical operations and error handling (Ecma International, 2026).
The browser adds a set of tools called **Web APIs** to the language, through which the program can interact with the page, the network and the device's functions (WHATWG, n.d.-a, n.d.-b).
Web APIs include: DOM, event handling, network requests, timers, local storage and media playback (WHATWG, n.d.-a, n.d.-b).

An HTML document can link a JavaScript file through the <script> element (WHATWG, n.d.-b):
<script src="script.js"></script>

- The src attribute indicates the URL from which to retrieve the program.
- The script.js path is a relative URL. It is resolved against the base URL of the HTML document (WHATWG, n.d.-b).

A common solution is to place the script in the head using the defer attribute:

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

defer allows the browser to retrieve the file while continuing to interpret the document and postpones its execution until the HTML structure has been parsed (WHATWG, n.d.-b).
The *DOM* provides interfaces, properties and methods through which JavaScript can

- Find elements;
- Read their content;
- Modify their text;
- Change attributes;
- Add or remove elements;
- Apply or remove classes;
- Respond to events (WHATWG, n.d.-a; World Wide Web Consortium [W3C], 2026).

For example:

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

- The first statement, "document.querySelector("h1")", searches the DOM for the first element matching the h1 selector. The reference to the element is stored in the constant "titolo" (WHATWG, n.d.-a).
- The second statement, "titolo.textContent = "New heading";", modifies the element's text content (WHATWG, n.d.-a).

The original HTML file is not modified on the server: only the DOM currently in the browser changes (WHATWG, n.d.-a).

**An event** is a signal indicating that something has happened (W3C, 2026).
Some examples are shown in the following table.

| Event | When it occurs |
|---|---|
| click | Occurs when the user selects an element, usually with a mouse click, a tap or an equivalent keyboard command. It is common on buttons and links. |
| input | Occurs whenever the user changes the value of a field. In a text field, it can fire for every character entered or deleted. |
| change | Occurs when the value of a control is changed and the change is committed. In text fields, this usually happens when the field loses focus; in selection boxes, it can happen immediately. |
| submit | Occurs when a <form> element is submitted, for example by pressing its submit button or the Enter key in a field. |
| load | Occurs when a resource has finished loading. It can concern the page, an image, a script or another resource. |
| keydown | Occurs when the user presses a key on the keyboard. It can repeat automatically if the key is held down. |

Events can be generated by user actions or by the browser and other components of the Web platform (WHATWG, n.d.-b; W3C, 2026).
JavaScript can register a function to be executed when a particular event occurs (WHATWG, n.d.-a).

For example:

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

pulsante.addEventListener("click", mostraMessaggio);

function mostraMessaggio() {
  console.log("The button has been selected");
}
```

- The reference to the first <button> element in the DOM is saved
- A function is associated with the click event on this reference.
- “mostraMessaggio” is the function associated with the event.
- The function is executed when the event reaches the button.

The function associated with an event is called an **event listener**, **event handler** or **callback function** (WHATWG, n.d.-a).

Let us create a page that automatically calculates the total amount from:

- Unit price.
- Quantity.

The result must update whenever the user changes either of the two values.
The corresponding HTML code is shown below:

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

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

  <title>Calculating the total amount</title>

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

<body>
  <main>
    <h1>Calculating the total amount</h1>

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

        <input
          type="number"
          id="price"
          name="price"
          min="0"
          step="0.01"
          required>
      </p>

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

        <input
```

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

      <p>
        Total amount:

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

</html>
```

The document contains:

- A <form> form.
- Two <input> fields.
- Two <label> labels.
- An <output> element for the result.
- The link to the script.js file.

The “for="price"” and “id="price"” attributes associate the label with the field (WHATWG, n.d.-b).
The min, step and required attributes define some constraints on the accepted values (WHATWG, n.d.-b).
The element:

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

represents a result calculated using the price and quantity fields.
The *HTML* standard explicitly provides the output element to represent calculation results associated with form controls (WHATWG, n.d.-b).
The JavaScript script is shown below:

```
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);
```

The first lines find the elements in the DOM:

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

The # symbol indicates that the selector searches for an element through its id (WHATWG, n.d.-a).

For example, “document.querySelector("#price")” searches for the element whose id is equal to “price”.

The references are stored using const because they will continue to refer to the same elements during execution (Ecma International, 2026).

The following statements are used to read the numeric values:

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

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

The valueAsNumber property returns the field's value interpreted as a number. When the value cannot be converted correctly, it returns NaN, meaning Not a Number (Ecma International, 2026; WHATWG, n.d.-b).
The HTML Standard defines valueAsNumber for numeric controls (WHATWG, n.d.-b).

The script then checks the constraints defined by the HTML:

- That the price is a finite number.
- That the quantity is a finite number.
- That the form has met the constraints (WHATWG, n.d.-b).

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

When the values are invalid, the result is replaced by two hyphens and the function's execution ends.

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

Once the two inputs have been checked, the actual processing takes place: the price is multiplied by the quantity.

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

Note that the result is formatted as Italian currency (in euros) and assigned to the <output> element.

What we have just described is the "updateTotal" function, which is linked to the input event of the element with id "calcolo-totale", stored in the form reference (WHATWG, n.d.-a; W3C, 2026).

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

The input event is generated when the user changes the value of a control (W3C, 2026).
The "updateTotal" function is therefore called whenever the price or quantity changes.
The calculation actually happens in real time, without the need to press a button.

The interaction sequence is as follows:

```
The user changes a field
          ↓
```

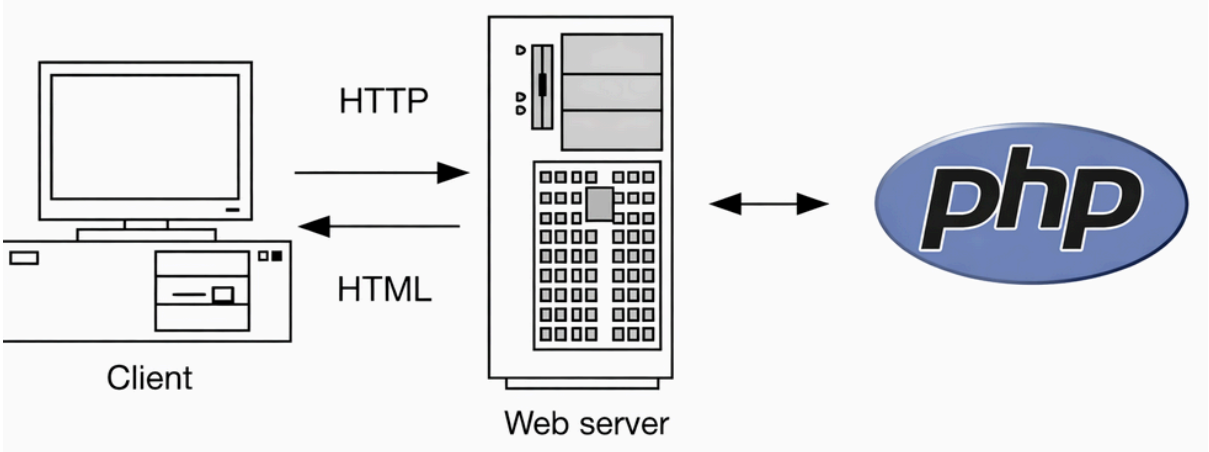
```
the browser generates the input event
          ↓
updateTotal() is executed
          ↓
JavaScript reads the values from the DOM
          ↓
calculates the total
          ↓
updates the output element
          ↓
the browser displays the new result
```

## Server Side

*PHP*, an acronym for Hypertext Preprocessor, is a scripting language widely used to develop server-side web applications (PHP Documentation Group, 2026).

This means that *PHP* code runs in the server environment and that the browser does not directly interpret *PHP* code, but receives only the result, for example (Fielding et al., 2022; PHP Documentation Group, 2026):

- An *HTML* document.
- Data in *JSON* format.
- A file.
- A redirect.
- An *HTTP* status code.
- A response with no content.




*PHP* can be embedded in documents containing *HTML* or used to build applications made up of many files and components. The official manual provides the delimiters <?php ... ?> and the shorthand form <?= ... ?> to produce a value in the output (PHP Documentation Group, 2026).

Files containing *PHP* code normally use the .php extension (PHP Documentation Group, 2026).
This extension often allows the server to recognize that the file must be processed by the *PHP* environment instead of being sent directly to the browser.

The path of a *URL* does not have to match the name or location of a physical file on the server (Fielding et al., 2022).
In a web application, the server can forward the request to a *PHP* program that interprets the path and decides which operation to perform.
For example, *the URL "*https://example.com/profilo" can trigger the function that retrieves the user's data from the database and generates the profile page, even if no file called profilo.php exists.
The rules that associate *URL* paths with application functions are called **routes**. Often a single file, such as index.php, receives many different requests and directs them to the correct part of the application.

The PHP application can:

- Receive information from the request.
- Check and interpret the data.
- Perform calculations or other operations.
- Communicate with a database.
- Decide the response status code.
- Produce HTML, JSON or other content.
- Send the response to the client.

The image below shows a simplified representation of the interaction between the browser and PHP.

## 1 BROWSER



### HTTP request

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

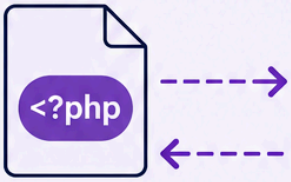
## 2 WEB SERVER



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



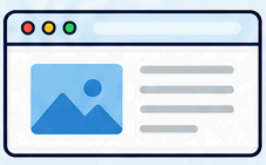
The application processes the data and prepares the result to be returned to the client.




### 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* provides predefined variables to access different information from the request, such as query strings, *POST* data, cookies and uploaded files (PHP Documentation Group, 2026).

Suppose the server has this file:

```php
<?php

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

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

?>

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

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

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

</html>
```

The user visits: https://example.com/saluto.php?nome=Anna.

PHP reads the query string parameter through “$_GET["nome"]” and produces an HTML document containing: <h1>Hello, Anna!</h1> (PHP Documentation Group, 2026)

The browser receives the *HTML* result, not the original PHP code. Specifically, I receive this (Fielding et al., 2022; PHP Documentation Group, 2026).

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

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

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

</html>
```

Note:
The htmlspecialchars() function converts certain special characters into the corresponding HTML entities/representations, so that they are treated as text in the appropriate context (PHP Documentation Group, 2026).

converts characters that have a special meaning in HTML, so that the text entered by the user can be included more correctly in the document. The operation to perform on the data still depends on the context: preparation for HTML differs from preparation for SQL, URLs or other formats (OWASP Foundation, n.d.; PHP Documentation Group, 2026).

PHP can access data from various parts of the request through these sources:

- Query string
- Body of a POST request
- Cookies and sessions

**Query string**
With a *URL* such as “https://example.com/prodotti?categoria=libri”
*PHP* can read the value through “$_GET["categoria"]”
$_GET contains the parameters in the query string, regardless of the HTTP method used (PHP Documentation Group, 2026).

**Body of a POST request**
When a form is submitted via POST:

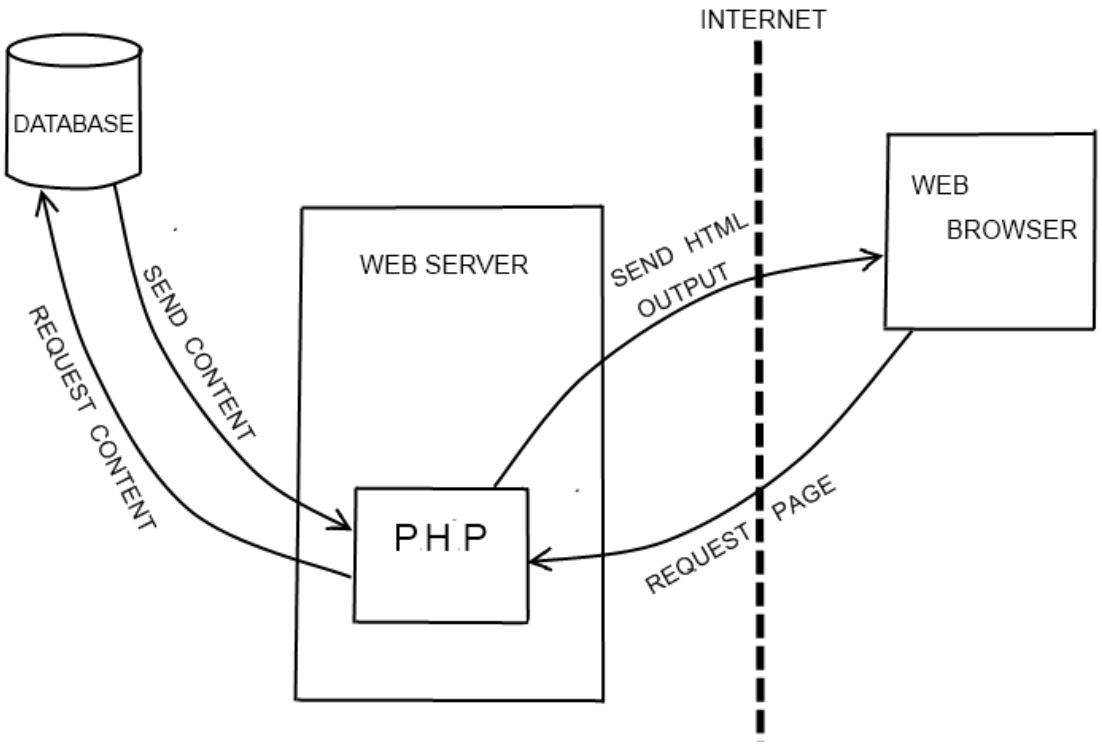
```
<form method="post" action="/login">
  ...
</form>
```

the values of ordinary fields can be made available through “$_POST” (PHP Documentation Group, 2026).

However, the body could also contain other formats, such as JSON, which require different systems for reading and interpretation (Fielding et al., 2022).

**Cookies and sessions**
Cookies sent by the browser can be read through “$_COOKIE” (PHP Documentation Group, 2026).

PHP also provides a session system that allows data associated with a user to be retained across successive requests. The browser normally stores a session identifier in a cookie, while the actual session data is managed by the server (PHP Documentation Group, 2026).




An important part of *PHP* is its interaction with the database.

***SQL***, an acronym for Structured Query Language, is a standardized language for working with data and structures in relational databases (International Organization for Standardization [ISO], 2023).
It allows operations such as:

- Creating tables.
- Defining constraints.
- Inserting data.
- Searching for data.
- Modifying data.
- Deleting data.
- Relating information from multiple tables.

Suppose we have a table called studenti.
An SQL query is shown below:

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

The statement asks the *DBMS* to return the value of the nome column for rows in which the classe column has the value 3C (ISO, 2023).

Some common *SQL* statements are:

- SELECT: To read data.
- INSERT: To insert new rows.
- UPDATE: To modify existing data.
- DELETE: To delete rows.
- CREATE TABLE: to define a new table (ISO, 2023).

***MySQL*** is a relational database management system (Oracle, 2026).
*MySQL* uses *SQL* to allow applications and administrators to define, read and modify data (ISO, 2023; Oracle, 2026).
*PHP* can communicate with *MySQL* through interfaces such as *PDO* and *MySQLi* (Oracle, 2026; PHP Documentation Group, 2026).
The application sends a statement to the *DBMS*, receives the result and decides how to use it in the *HTTP* response (Fielding et al., 2022; Oracle, 2026; PHP Documentation Group, 2026).

An example is shown below:
Suppose we have a database called scuola, containing the studenti table with the fields: id, nome and classe.
The table therefore contains these values:

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

Let us create the studenti.php file:

```php
<?php

// Data needed to connect to the database.
$host = "localhost";
$database = "scuola";
$username = "utente_app";
$password = "password";

// The DSN describes the database we want to connect to.
$dsn = "mysql:host=$host;dbname=$database;charset=utf8mb4";

try {
    // Creating the database connection.
    $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,
        ]
    );

    // Reads the class from the query string.
    // If the parameter is missing, uses 3C.
    $classe = $_GET["classe"] ?? "3C";

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

    // Replaces :classe with the received value.
    $istruzione->execute([
        "classe" => $classe,
    ]);

    // Retrieves all rows returned by the DBMS.
    $studenti = $istruzione->fetchAll();

} catch (PDOException $errore) {
    http_response_code(500);
    exit("The students could not be retrieved.");
}

?>

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

<head>
  <meta charset="utf-8">
  <title>Students in class</title>
</head>

<body>
  <h1>
    Students in class
```

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

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

    <p>No students were found.</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* allows *PHP* to create a connection to *MySQL* through the corresponding driver (Oracle, 2026; PHP Documentation Group, 2026).
The connection is established by creating a *PDO* object and providing the *DSN*, username and password (PHP Documentation Group, 2026).

The user could visit: https://example.com/studenti.php?classe=3C
The query string contains: classe=3C.
PHP retrieves the value through: $classe = $_GET["classe"] ?? "3C".
The variable $classe therefore contains: 3C.

Having received the inputs, *PHP* prepares the query to send to the *DBMS* with this statement:

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

The parameter “:classe” is a placeholder (PHP Documentation Group, 2026).

Its value is supplied separately:

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

The statement is conceptually equivalent to asking:

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

After the query is executed, the DBMS returns the matching rows to PHP (Oracle, 2026; PHP Documentation Group, 2026):

- Carmela.
- Giovanni.
- Michela.

These rows are stored in the variable:

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

This variable can be thought of as a collection of data:

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

*PHP* iterates through the results using:

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

and produces an <li> element for each student (PHP Documentation Group, 2026).

Finally, the browser receives the HTML document produced by executing the *PHP* code (Fielding et al., 2022; PHP Documentation Group, 2026):

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

<head>
  <meta charset="utf-8">
  <title>Students in class</title>
</head>

<body>
  <h1>Students in class 3C</h1>

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

# Supplementary material

## Exercises

**Client or Server?**

An e-commerce platform must perform these operations:

- Immediately display a menu when the user presses a button.
- Calculate the provisional cart total while the user changes the quantity.
- Check that the e-mail field in a form has an acceptable format.
- Check that the entered password actually matches the user's account.
- Check a product's availability in the database.
- Visually update the number of products shown in the cart.
- Permanently record an order.
- Display an animation when payment has been completed.

For each operation:

- Decide whether it should mainly run on the client side, on the server side or through both working together.
- Justify each choice.
- Explain why some checks can be performed in the browser to improve the user experience but must still be repeated on the server.
- Explain why it would be unsafe to entrust credential verification or payment authorization solely to JavaScript in the browser.
- Identify which operations might require access to a database.
- Finally, reconstruct the complete purchase flow: User → JavaScript → HTTP request → server application → database → response → interface update.

**Making a Quote Interactive**

A page contains two fields:

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

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

The result must update automatically whenever the user changes either of the two values.

Conceptually design the JavaScript script needed.

- State how JavaScript can find the three elements in the DOM.
- Explain which event you would use to immediately detect changes to the fields.
- Describe what the function associated with the event must do.
- Explain why it is necessary to check that the entered values are actually numeric and valid.
- If price = 24.50 and quantity = 3, state what result should be displayed.
- Reconstruct the sequence: User change → event → event listener → reading the DOM → calculation → modifying the DOM → visual update.

- Explain why the original HTML file on the server is not modified when JavaScript changes the content of the <output> element.
- Explain the advantage of the defer attribute when the JavaScript file is linked inside <head>.
- Imagine that the script runs before the HTML elements exist in the DOM. Explain what problem could occur.
- State which tools used in the exercise belong to the ECMAScript language and which are instead provided by the browser's Web APIs.

**From the Form to the PHP Response**
A website allows users to access their personal area through this form:

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

When the user submits the form, the request is handled by a PHP application.
- Explain where the page's HTML code is interpreted and its JavaScript is executed, and where the PHP code is executed.
- State through which PHP structure the data from fields submitted via POST can normally be read.
- Explain why the browser does not receive the original PHP code.
- State what results PHP could produce after processing, for example HTML, JSON, a redirect or an HTTP status code.
- Suppose that the URL /login is handled by index.php, even though no login.php file exists. Explain how this is possible using the concept of routes.
- After a successful login, the server must remember the user's identity during subsequent requests. Explain how cookies and a session can work together.
- Explain why data from the user must be handled appropriately before it is inserted into the HTML produced by the server.
- Explain why htmlspecialchars() is appropriate for a certain HTML context but does not automatically provide adequate protection when the same data is used in SQL.
- Reconstruct the process: Filling in the form → POST request → PHP → data verification → possible session → response generation → browser.

**From the Database to the Web Page**
A school has this table:

| id | nome | classe |
|---|---|---|
| 1 | Giovanni | 3C |
| 2 | Michela | 3C |
| 3 | Carmela | 3C |

| | | |
|---|---|---|
| 4 | Marco | 4A |

A user visits: https://example.com/studenti?classe=3C

The PHP application must display only the students in the requested class.

- Explain how PHP can retrieve the value 3C from the query string.
- Conceptually write the SQL query needed to obtain the nome and classe of the students in 3C.
- State what function SELECT performs.
- Explain the difference between SQL and MySQL.
- Explain what role PDO can play between PHP and MySQL.
- Explain why it is preferable to use a parameterized query with a placeholder such as :classe instead of inserting the value received from the user directly into the query.
- Describe what the DBMS returns to the PHP application.
- Explain how PHP can use the results to generate an HTML list made up of <li> elements.
- Explain why the browser will receive the resulting HTML document and neither the SQL query nor the PHP code that generated it.
- Imagine that the request is instead “https://example.com/studenti?classe=5B” and that there are no students in 5B. Suggest what response the application should produce.
- Finally, imagine that the database cannot be reached. Explain why it might be appropriate to return a server-side HTTP error.
- Reconstruct the entire flow: Browser → HTTP request → PHP application/router → PDO → MySQL, which executes the SQL query → results → PHP → HTML → HTTP response → browser.

# Bibliography


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/Consortium.

# Further reading

- Marijn Haverbeke, Eloquent JavaScript: A Modern Introduction to Programming, 4th ed., No Starch Press, 2024 — Probably the best introductory reference for the client-side section. It explains JavaScript fundamentals, functions, objects, events, asynchronous programming, DOM manipulation, and browser APIs. Its practical examples are particularly relevant to understanding how JavaScript responds to user actions and dynamically updates web interfaces. The fourth edition includes dedicated chapters on the DOM, event handling, HTTP, and forms.
- David Flanagan, JavaScript: The Definitive Guide, 7th ed., O'Reilly Media, 2020 — A comprehensive technical reference for JavaScript and the browser programming environment. It covers variables, functions, objects, modules, asynchronous programming, the DOM, events, networking, storage, and browser APIs. It is particularly useful for understanding the distinction between the ECMAScript language itself and the additional capabilities provided by the browser, a distinction explicitly developed in the handout.
- Jon Duckett, PHP & MySQL: Server-side Web Development, Wiley, 2022 — The most directly relevant book for the server-side section. It explains PHP programming, processing information received from browsers, generating dynamic HTML, handling forms, working with cookies and sessions, managing errors, and connecting applications to MySQL databases. Its visual approach and complete application examples make it particularly suitable for students learning how PHP, SQL, and HTTP work together.
- Alan Beaulieu, Learning SQL, 3rd ed., O'Reilly Media, 2020 — Provides a systematic introduction to relational databases and SQL. It examines tables, relationships, queries, filtering, sorting, joins, and data manipulation using statements such as SELECT, INSERT, UPDATE, and DELETE. It is particularly relevant to the handout's explanation of the relationship between SQL as a database language, MySQL as a DBMS, and PHP as an application that communicates with the database.
- Malcolm McDonald, Web Security for Developers, No Starch Press, 2020 — An accessible introduction to security in web applications, including vulnerabilities related to user input, authentication, sessions, and database interactions. It is particularly useful for understanding why client-side validation cannot replace server-side verification, why user input must be handled according to its destination, and how vulnerabilities such as cross-site scripting and SQL injection can be prevented.

# Editorial notes

In producing these materials, I have used generative artificial intelligence tools to support the writing process, particularly to improve the form of the text, reorganize content, refine wording, and speed up certain editorial tasks.

As I work independently on the production of these materials, I try to automate any activities that can reasonably be automated, so that I can devote more time to research, design, and content development.

However, artificial intelligence does not determine the content of this work: the choice of topics, structure, ideas, interpretations, examples, and teaching approach are developed by me. AI is therefore used primarily as a tool to support production and formal revision, while authorship and responsibility for the design and content remain mine.

Unless otherwise indicated, this material is distributed under the Creative Commons Attribution–NonCommercial–ShareAlike 4.0 International license (CC BY-NC-SA 4.0).

You are therefore free to share, redistribute, adapt, and create derivative works based on this material, provided that appropriate credit is given, the material is not used for commercial purposes, and any modified or derivative versions are distributed under the same license.

