Appunti di Programmazione PHP

Prof. Luca Campion

13 novembre 2025

Indice

Prefazione

Questi appunti di PHP adottano una struttura modulare allineata agli altri corsi. L'obiettivo è fornire spiegazioni tecniche accessibili, esempi pratici e rimandi incrociati, con attenzione a sicurezza, affidabilità e manutenzione.

Capitolo 1

Form HTML

Mappa del capitolo

Introduzione, Creazione di form, Gestione dati, Validazione, Sicurezza XSS, Esempi pratici, Caso di studio, Diagrammi, Esercizi, Verifica, Riepilogo, Riferimenti.

1.1 Obiettivi di apprendimento

- Progettare form ben strutturati per GET e POST.
- Gestire correttamente input lato server con normalizzazione e escaping.
- Applicare PRG quando si modifica stato e prevenire XSS.

1.2 Teoria

I form HTML sono la principale modalità con cui un client invia dati al server. PHP riceve i dati tramite le superglobali `$_GET`, `$_POST` e `$_FILES`. La scelta del metodo dipende dal caso d'uso: **GET** per richieste idempotenti e query string, **POST** per invio di dati sensibili o variazioni di stato.

1.3 Creazione di form (GET/POST)

```
<!-- Esempio GET -->
<form method="get" action="processa.php">
  <label>Query: <input type="text" name="q"></label>
  <button type="submit">Cerca</button>
</form>

<!-- Esempio POST -->
<form method="post" action="processa.php">
  <label>Email: <input type="email" name="email" required></label>
  <label>Password: <input type="password" name="password" required></label>
  <button type="submit">Invia</button>
</form>
```

1.4 Gestione dei dati inviati

```php
<?php
// processa.php
$metodo = $_SERVER['REQUEST_METHOD'];
$email  = $_POST['email'] ?? '';
$q      = $_GET['q'] ?? '';

if ($metodo === 'POST') {
    if ($email === '') {
        echo 'Email non valida';
        exit;
    }
    echo 'OK';
} else {
    echo 'Ricerca: ' . htmlspecialchars($q, ENT_QUOTES | ENT_SUBSTITUTE, '
        UTF-8');
}
```

1.5 Validazione degli input

- Validazione lato client (HTML5: `required`, `type=email`) e lato server (sempre necessaria). - Usare regex e normalizzazione dei dati.

```php
<?php
// Normalizzazione e validazione
$name = trim((string)($_POST['name'] ?? ''));
if ($name === '' || mb_strlen($name) > 100) {
    echo 'Nome obbligatorio (<=100)';
    exit;
}
```

1.6 Sicurezza: prevenzione XSS

- **XSS**: effettuare sempre escaping in output con `htmlspecialchars`.

```php
<?php
// Esempio di escaping in output
// Usare htmlspecialchars per prevenire XSS quando si rende input utente
echo 'Ricerca: ' . htmlspecialchars($q, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8
    ');
```

Best practice

- Usare POST per dati sensibili e operazioni che modificano stato. - Escaping sistematico dell'output (XSS). - Validare e normalizzare sempre lato server. - Impostare `Content-Type` e charset corretti nelle risposte.

Errori comuni

- Fidarsi della sola validazione lato client. - Non effettuare escaping dell'output. - Mescolare GET/POST senza gestire i casi distinti.

1.7 Esempi pratici

```php
<?php
// Contatto semplice con normalizzazione e escape
$name  = trim((string)($_POST['name'] ?? ''));
$email = trim((string)($_POST['email'] ?? ''));
$msg   = trim((string)($_POST['message'] ?? ''));

$errors = [];
if ($name === '' || mb_strlen($name) > 100) { $errors[] = 'Nome obbligatorio (<=100)'; }
if (!preg_match('/^[^@\s]+@[^@\s]+\.[^@\s]+$/', $email)) { $errors[] = 'Email non valida'; }
if ($msg === '') { $errors[] = 'Messaggio obbligatorio'; }

if ($errors) {
    // PRG: mostra errori su pagina GET
    header('Location: /contatto_errore.php', true, 303);
    exit;
}

echo 'Grazie, ' . htmlspecialchars($name, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
?>
```

1.8 Caso di studio

Progettazione di un form di login minimale con gestione errori e PRG. Si valida l'input, si normalizza e si evita qualsiasi echo diretto di dati utente senza escape.

```php
<?php
$u = trim((string)($_POST['username'] ?? ''));
$p = (string)($_POST['password'] ?? '');
if ($u === '' || $p === '') {
  header('Location: /login.php?err=1', true, 303);
  exit;
}
// Verifica credenziali (simulata)
header('Location: /dashboard.php', true, 303);
exit;
?>
```

1.9 Diagrammi

Flusso PRG: *POST* (valida/aggiorna) → *303 See Other* → *GET* pagina di conferma.

1.10 Esercizi

- Implementa un form di contatto con validazione lato server e PRG.
- Aggiungi un campo textarea e normalizza gli spazi e le nuove linee.
- Progetta un form di ricerca con GET e escaping dell'output.

1.11 Verifica

- Vero/Falso: dopo `header('Location',...)` è obbligatorio `exit`.
- Quale metodo usare per mostrare risultati di ricerca? GET o POST e perché?

1.12 Riepilogo

Form ben progettati separano responsabilità tra input, elaborazione e output. La validazione lato server e l'escaping sono indispensabili; PRG evita ri-submit del POST.

1.13 Riferimenti

- Manuale PHP — superglobali: https://www.php.net/manual/en/reserved.variables.php
- OWASP XSS: https://owasp.org/www-community/attacks/xss/
- HTTP Semantics (Redirect): https://www.rfc-editor.org/rfc/rfc9110

Capitolo 2

Campi Hidden

Mappa del capitolo

Teoria, Utilizzo pratico, Scenari di applicazione, Sicurezza, Esempi estesi, Caso di studio, Esercizi, Verifica, Riferimenti.

2.1 Obiettivi di apprendimento

- Comprendere il ruolo dei campi *hidden* nei form.
- Applicare validazioni lato server per dati non visibili.
- Integrare hidden con sessione e flussi multi-step mantenendo integrità.

2.2 Teoria

I campi *hidden* sono input non visibili all'utente ma inviati con il form. Utili per trasmettere metadati, token di sicurezza, identificatori di stato.

2.3 Utilizzo pratico

```
<!-- Trasmettere un ID ordine e metadati -->
<form method="post" action="checkout.php">
  <input type="hidden" name="order_id" value="12345">
  <button type="submit">Conferma</button>
</form>
```

Esempio esteso: token di integrità (concettuale)

```
<?php
// Generazione lato server (pagina precedente)
$orderId = 12345;
$secret  = 'chiave_server';
$token   = sha1($orderId . '|' . $secret); // firma semplice
?>
<form method="post" action="checkout.php">
  <input type="hidden" name="order_id" value="<?php echo (int)$orderId; ?>"
      >
  <input type="hidden" name="token" value="<?php echo $token; ?>">
```

```
  <button type="submit">Conferma</button>
</form>

<?php
// Verifica lato server (checkout.php)
$orderId = isset($_POST['order_id']) ? (int)$_POST['order_id'] : 0;
$token   = isset($_POST['token']) ? (string)$_POST['token'] : '';
$secret  = 'chiave_server';
if ($orderId <= 0) {
    echo 'order_id non valido';
    exit;
}
$expected = sha1($orderId . '|' . $secret);
if ($token !== $expected) {
    echo 'Token non valido';
    exit;
}
// OK: procedere con checkout
?>
```

2.4 Scenari di applicazione

- Stato dell'applicazione (wizard multi-step, ID risorsa). - Parametri non modificabili lato client.

2.5 Caso di studio: wizard di checkout a più step

- Step 1: selezione prodotti; salvataggio in sessione.
- Step 2: indirizzo e spedizione; hidden con `order_id`.
- Step 3: pagamento; verifica integrità di `order_id` con token.
- Step 4: conferma; controllo finale e generazione ordine.

2.6 Considerazioni sulla sicurezza

- Non fidarsi dei valori hidden: sono modificabili lato client; validare sempre lato server. - Usare **token** firmati o verificabili (es. HMAC) se necessario. - Non inserire mai dati sensibili in chiaro; preferire sessione lato server.

```
<?php
// Verifica lato server senza funzioni rimosse
$orderId = isset($_POST['order_id']) ? (int)$_POST['order_id'] : 0;
if ($orderId <= 0) {
    echo 'order_id non valido';
    exit;
}
// Continuare elaborazione
```

Best practice

- Usare hidden per metadati non sensibili; mai per segreti. - Convalidare tutti i campi hidden lato server. - Abbinare hidden a sessione o tracciamento lato server per integrità.

Errori comuni

- Presumere che hidden sia sicuro/inviolabile. - Esporre informazioni sensibili (es. ruoli, prezzi calcolati) in hidden. - Mancare la validazione server-side dei hidden.

2.7 Esercizi

- Progetta un form multi-step che usa hidden solo per ID non sensibili, mantenendo i dati in sessione.
- Implementa una verifica di integrità per un ID trasmesso via hidden.

2.8 Verifica

- Perché i campi hidden non sono affidabili come meccanismo di sicurezza?
- Quali dati vanno mantenuti in sessione anziché in hidden?

2.9 Riferimenti

- MDN — HTML input hidden: https://developer.mozilla.org/
- OWASP — Tampering dei parametri: https://owasp.org/

Capitolo 3

Redirect mediante Header Location

Mappa del capitolo

Teoria, Implementazione corretta, Header già inviati, Best practice, Codici di stato, Esempi pratici, Caso di studio, Esercizi, Verifica, Riferimenti.

3.1 Teoria

Il redirect HTTP si effettua impostando l'header `Location` e un codice di stato appropriato. Dopo aver inviato l'header, l'esecuzione deve terminare con `exit`.

3.2 Implementazione corretta

```php
<?php
// Redirect dopo una creazione (POST)
// Usare 303 See Other per evitare ri-submit del POST
header('Location: /success.php', true, 303);
exit;
```

3.3 Gestione degli header già inviati

Se è già stato generato output, l'invio degli header fallisce. Evitare output prima degli header o utilizzare buffer di output se necessario.

3.4 Best practice per i redirect

- Post/Redirect/Get (PRG) per evitare doppio submit. - Usare codici 302/303/307 in base al contesto. - Impostare sempre `exit` dopo l'header `Location`. - Evitare output prima degli header: attivare output buffering o controllare flussi.

3.5 Codici di stato appropriati

- **302 Found**: redirect temporaneo generico. - **303 See Other**: dopo POST per puntare a risorsa GET. - **307 Temporary Redirect**: preserva il metodo (POST resta POST). - **301 Moved Permanently**: redirect permanente (SEO, attenzione ai cache).

Errori comuni

- Dimenticare `exit` dopo il redirect. - Inviare contenuto prima degli header. - Usare sempre 302 anche dopo POST (meglio 303 per PRG).

3.6 Esempi pratici

```php
<?php
// 307: preserva il metodo
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  header('Location: /conferma.php', true, 303); // PRG
  exit;
}

// 301: migrazione URL permanente
header('Location: https://www.esempio.it/nuovo-percorso', true, 301);
exit;
?>
```

3.7 Caso di studio

Workflow di registrazione utente: POST /register → valida e crea → 303 See Other su /welcome.

3.8 Esercizi

- Implementa PRG su un form di contatto.
- Esegui un redirect 307 per ripetere POST verso un endpoint differente.

3.9 Verifica

- Qual è il codice preferibile dopo POST nel pattern PRG?
- Cosa accade se invii output prima degli header?

3.10 Riferimenti

- RFC 9110: Semantics of HTTP — Redirect status codes.
- Manuale PHP — `header`: https://www.php.net/header

Capitolo 4

Array

Mappa del capitolo

Teoria, Creazione e manipolazione, Funzioni principali, Iterazione, Array multidimensionali, Esempi pratici, Caso di studio, Esercizi, Verifica, Riepilogo, Riferimenti.

4.1 Teoria

Gli array in PHP sono strutture flessibili che possono contenere valori di qualunque tipo. Esistono funzioni native per creare, manipolare e iterare.

4.2 Creazione e manipolazione

```php
<?php
$nums = [1, 2, 3];
$mix  = ['a', 10, true];
array_push($nums, 4); // [1,2,3,4]
$nums = array_merge([0], $nums); // [0,1,2,3,4]
unset($mix[1]); // rimuove elemento
```

4.3 Funzioni principali

- `count`, `array_push`, `array_pop`, `array_shift` - `in_array`, `array_search`, `array_key_exists`

```php
<?php
// Esempi senza funzioni di ordine superiore
// Quadrati
$squared = [];
foreach ($nums as $x) { $squared[] = $x * $x; }
// Pari
$even = [];
foreach ($nums as $x) { if ($x % 2 === 0) { $even[] = $x; } }
// Somma
$sum = 0;
foreach ($nums as $x) { $sum += $x; }
```

4.4 Iterazione

```php
<?php
for ($i=0; $i<count($nums); $i++) { echo $nums[$i]; }
foreach ($nums as $n) { echo $n; }
```

4.5 Array multidimensionali

```php
<?php
$matrix = [ [1,2], [3,4] ];
echo $matrix[1][0]; // 3
```

Best practice

- Usare `foreach` per semplicità e leggibilità. - Evitare rimuovere elementi senza reindicizzare se serve l'indice.

Errori comuni

- Presumere che gli indici si ricalcolino automaticamente dopo `unset`. - Dimenticare che PHP consente tipi misti: gestire coerenza dei dati.

4.6 Esempi pratici

```php
<?php
// Reindicizzazione dopo rimozione
$a = [10,20,30];
unset($a[1]);            // [0=>10, 2=>30]
$a = array_values($a);   // [0=>10, 1=>30]

// Aggregazioni
$prezzi = [10.5, 7.2, 13.0];
$totale = 0.0;
foreach ($prezzi as $p) { $totale += $p; }
?>
```

4.7 Caso di studio

Gestione di un carrello: aggiunta, rimozione, calcolo totale e sconti senza funzioni di ordine superiore.

4.8 Esercizi

- Dato un array di interi, costruisci un nuovo array di quadrati usando solo cicli.
- Reindicizza correttamente dopo rimozioni multiple e verifica gli indici.
- Implementa un carrello con totale e sconto percentuale per importi > 100.

4.9 Verifica

- Cosa restituisce `count` su un array con buchi di indici?
- Quando è necessario usare `array_values`?

4.10 Riepilogo

Gli array in PHP sono dinamici e flessibili: usa cicli per trasformazioni e filtri, e reindicizza quando necessario.

4.11 Riferimenti

- Manuale PHP — Arrays: https://www.php.net/array

Capitolo 5

Array Associativi

Mappa del capitolo

Teoria, Confronto con array numerici, Funzioni utili, Esempi pratici, Conversioni, Caso di studio, Esercizi, Verifica, Riferimenti.

5.1 Obiettivi di apprendimento

- Modellare dati semplici con array associativi in PHP.
- Applicare funzioni standard per manipolare chiavi e valori.
- Integrare array con JSON e query string in modo sicuro.

5.2 Teoria

Gli array associativi usano chiavi stringa per mappare valori, ideali per rappresentare oggetti semplici o record.

5.3 Differenze con array numerici

- Chiave esplicita vs indice numerico. - Iterazione con `foreach ($arr as $k=>$v)`.

```php
<?php
$user = ['id'=>10, 'name'=>'Alice', 'role'=>'admin'];
echo $user['name'];
```

5.4 Funzioni specifiche

- `array_keys`, `array_values`, `array_merge`, `array_replace`

```php
<?php
$users = [
  ['id'=>1,'name'=>'A'],
  ['id'=>2,'name'=>'B'],
];
// Estrazione manuale di una colonna
$names = [];
foreach ($users as $row) { $names[] = $row['name']; } // ['A','B']
```

Esempi pratici aggiuntivi

```php
<?php
// Merge con precedenza del secondo array sulle stesse chiavi
$base = ['host'=>'localhost','port'=>3306,'debug'=>false];
$override = ['debug'=>true];
$cfg = array_replace($base, $override); // ['host'=>..., 'port'=>..., 'debug'=>true]

// Ordinamento per chiave (k) e per valore (v)
$map = ['z'=>3, 'a'=>1, 'm'=>2];
ksort($map);  // ['a'=>1, 'm'=>2, 'z'=>3]
asort($map);  // ['a'=>1, 'm'=>2, 'z'=>3] (ordinato per valore crescente)

// Ridenominazione chiavi tramite costruzione manuale
$user = ['id'=>10,'name'=>'Alice'];
$renamed = [];
foreach ($user as $k=>$v) {
    $newKey = ($k==='name') ? 'nome' : $k;
    $renamed[$newKey] = $v;
}
```

5.5 Conversione da/verso altri formati

- JSON: `json_encode` / `json_decode(true)` (associativo) - Query string: costruzione manuale (`urlencode` + `implode`)

```php
<?php
$payload = json_encode($user, JSON_UNESCAPED_UNICODE);
$assoc   = json_decode($payload, true);
// Costruzione manuale di query string
$pairs = [];
foreach ($assoc as $k=>$v) { $pairs[] = urlencode($k).'='.urlencode((string)$v); }
$qs = implode('&', $pairs);
```

5.6 Caso di studio: impostazioni applicative

Rappresentazione di una configurazione come mappa chiave→valore con override per ambiente (sviluppo/produzione) e esportazione in query string per diagnosticare lo stato.

5.7 Esercizi

- Dato un elenco di utenti, costruisci `id=>nome` e ordinane le chiavi alfabeticamente.
- Implementa una funzione che rinomini una chiave in un array associativo senza perdere l'ordine.

5.8 Verifica

- Qual è la differenza tra `array_merge` e `array_replace`?
- Come si impone l'ordinamento per chiave rispetto a per valore?

5.9 Riferimenti

- Manuale PHP — Array: https://www.php.net/array
- JSON in PHP: https://www.php.net/json

Best practice

- Validare chiavi attese e valori prima dell'uso. - Per oggetti complessi preferire classi/DTO; usare array associativi per strutture semplici.

Errori comuni

- Confondere array associativi e oggetti stdClass dopo `json_decode` (usare `true`). - Usare `array_merge` senza considerare override di chiavi.

Capitolo 6

Funzioni

Mappa del capitolo

Teoria, Definizione e chiamata, Parametri e ritorni, Scope, Esempi pratici, Caso di studio, Esercizi, Verifica, Riferimenti.

6.1 Obiettivi di apprendimento

- Incapsulare logica riutilizzabile con funzioni chiare.
- Comprendere parametri, valori di ritorno e scope delle variabili.
- Applicare pattern di funzioni pure e gestire effetti collaterali.

6.2 Teoria

Le funzioni incapsulano logica riutilizzabile. In PHP moderno è consigliato usare type hints per parametri e tipo di ritorno.

6.3 Definizione e chiamata

```php
<?php
function greet(string $name): string {
    return "Ciao, $name";
}
echo greet('Mondo');
```

Stile alternativo senza type hints (compatibilità)

```php
<?php
function greet2($name) {
    $name = (string)$name;
    return 'Ciao, ' . $name;
}
echo greet2('Mondo');
```

6.4 Parametri e valori di ritorno

- Parametri con default, tipi unione, passaggio per riferimento se necessario.

```php
<?php
function add(int $a, int $b = 0): int { return $a + $b; }
function normalize(?string $s): string { return trim($s ?? ''); }
```

Passaggio per riferimento e variadiche

```php
<?php
function pushValue(array &$arr, $value) { $arr[] = $value; }
function sumAll(...$nums) {
    $tot = 0;
    foreach ($nums as $n) { $tot += (int)$n; }
    return $tot;
}
$a = [];
pushValue($a, 10); // $a = [10]
```

6.5 Variabili globali e locali

- Evitare l'uso di `global`; preferire passaggio parametri o dipendenze esplicite.

```php
<?php
$counter = 0;
function inc(): int {
    static $c = 0; // persiste tra chiamate
    $c++; return $c;
}
```

6.6 Esempi pratici

```php
<?php
// Funzione pura
function twice($n) { return (int)$n * 2; }

// Closure con cattura di contesto
$prefixer = function($p) {
    return function($s) use ($p) { return $p . (string)$s; };
};
$hello = $prefixer('Hello ');
echo $hello('world');
```

6.7 Caso di studio: libreria di utilità

Raccolta di funzioni pure per normalizzare input (trim, lower/upper, sostituzioni) e funzioni impure per logging controllato, con documentazione delle pre/post-condizioni.

6.8 Esercizi

- Implementa una funzione `rename_key($arr, $old, $new)` che rinomini una chiave se presente.
- Scrivi una funzione `compose($f,$g)` che restituisca una nuova funzione composizione.

6.9 Verifica

- Quando è opportuno usare variabili `static` all'interno delle funzioni?
- Quali benefici e limiti portano i type hints in PHP?

6.10 Riferimenti

- Manuale PHP — Funzioni: https://www.php.net/functions
- PSR — Best practices di coding: https://www.php-fig.org/

Best practice

- Tipizzare parametri e ritorni per chiarezza e robustezza. - Funzioni pure quando possibile; gestione degli effetti collaterali ben delimitata. - Documentare pre/post-condizioni e casi limite.

Errori comuni

- Abuso di variabili globali. - Mancata validazione degli argomenti. - Ignorare i tipi e restituire valori eterogenei.

Capitolo 7

File di Testo

Mappa del capitolo

Teoria, Lettura e scrittura, Permessi, Bloccaggio, CSV, Esempi pratici, Caso di studio, Esercizi, Verifica, Riferimenti.

7.1 Teoria

La gestione dei file include apertura, lettura/scrittura, permessi e lock per evitare corruzione in accessi concorrenti.

7.2 Lettura e scrittura

```php
<?php
$path = __DIR__ . '/dati.txt';
file_put_contents($path, "Prima riga\n", FILE_APPEND | LOCK_EX);
$contenuto = file_get_contents($path);
echo $contenuto;
```

7.3 Gestione dei permessi

- Verificare `is_readable`, `is_writable` e gestire errori con messaggi chiari.

```php
<?php
if (!is_writable($path)) {
    echo 'File non scrivibile';
}
```

7.4 Bloccaggio dei file

```php
<?php
$fp = fopen($path, 'c+');
if (flock($fp, LOCK_EX)) {
    fwrite($fp, "Riga sicura\n");
    fflush($fp);
    flock($fp, LOCK_UN);
}
fclose($fp);
```

7.5 Operazioni su CSV

```php
<?php
$fp = fopen(__DIR__ . '/utenti.csv', 'r');
while (($row = fgetcsv($fp, 0, ';')) !== false) {
    // $row è un array di campi
}
fclose($fp);

// Scrittura
$fp = fopen(__DIR__ . '/out.csv', 'w');
fputcsv($fp, ['id','name'], ';');
fclose($fp);
```

Best practice

- Usare `LOCK_EX` o `flock` per scritture concorrenti. - Gestire gli errori con codici HTTP e log. - Validare i dati prima di scrivere su disco.

Errori comuni

- Ignorare permessi e fallimenti di I/O. - Non chiudere i file o non flushare i buffer. - Usare separatori inconsistenti in CSV.

7.6 Esempi pratici

```php
<?php
// Lettura riga-per-riga con controllo errori
$fp = fopen($path, 'r');
if (!$fp) { echo 'Apertura fallita'; }
while (($line = fgets($fp)) !== false) {
    echo htmlspecialchars($line, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
}
fclose($fp);
?>
```

7.7 Caso di studio

Semplice logger applicativo su file con rotazione: quando la dimensione supera una soglia, rinomina il file con timestamp e riparte.

7.8 Esercizi

- Implementa una funzione che scrive JSON su file con lock e controlla permessi.
- Leggi un CSV e costruisci un array associativo indicizzato per chiave.

7.9 Verifica

- Qual è la differenza tra `LOCK_EX` e `LOCK_SH`?
- Perché è importante impostare un encoding coerente?

7.10 Riferimenti

- Manuale PHP — File system: https://www.php.net/manual/en/book.filesystem.php

Capitolo 8

Sessioni

Mappa del capitolo

Teoria, Avvio e configurazione, Memorizzazione, Sicurezza, Sessioni e cookies, Esempi pratici, Caso di studio, Diagramma, Esercizi, Verifica, Riferimenti.

8.1 Teoria

Le sessioni mantengono stato tra richieste. Si basano tipicamente su cookie `PHPSESSID` che identifica un contesto sul server.

8.2 Avvio e configurazione

```php
<?php
ini_set('session.cookie_httponly', '1');
ini_set('session.use_strict_mode', '1');
session_start();
```

8.3 Memorizzazione dati

```php
<?php
$_SESSION['user_id'] = 123;
$_SESSION['cart'] = ['book'=>2, 'pen'=>1];
```

8.4 Sicurezza delle sessioni

- **HttpOnly** e **Secure** per i cookie (in HTTPS). - Rigenerare l'ID sessione dopo login: `session_regenerate_id(true)`. - Limitare durata e percorso del cookie; validare user agent/IP se necessario.

```php
<?php
session_start();
// Dopo autenticazione
session_regenerate_id(true);
```

8.5 Sessioni e cookies

Le sessioni usano cookie per identificare il client; evitare memorizzare dati sensibili nel cookie, preferire lo storage lato server.

Best practice

- Attivare `session.use_strict_mode` e cookie `HttpOnly`. - Rigenerare ID dopo cambiamenti di privilegi. - Invalidare sessione al logout: `session_destroy()` e cancellare cookie.

Errori comuni

- Non avviare la sessione prima di usarla. - Lasciare sessioni troppo durature senza invalidazione. - Non proteggere il cookie di sessione (manca HttpOnly/Secure).

8.6 Esempi pratici

```php
<?php
// Configurazione cookie
session_set_cookie_params([
  'lifetime' => 0,
  'path' => '/',
  'domain' => '',
  'secure' => isset($_SERVER['HTTPS']),
  'httponly' => true,
  'samesite' => 'Strict',
]);
session_start();
?>
```

8.7 Caso di studio

Login e logout: al login si rigenera l'ID, al logout si invalida la sessione e si resetta il cookie.

8.8 Diagramma

Flusso: *Login POST → session_regenerate_id → set cookie HttpOnly+SameSite → accesso area riservata → logout.*

8.9 Esercizi

- Implementa una funzione di logout che cancella cart e invalida la sessione.
- Aggiungi scadenza breve e verifica comportamento su browser.

8.10 Verifica

- Perché rigenerare l'ID dopo login?
- Quali attributi cookie riducono i rischi di attacco rubando la sessione?

8.11 Riferimenti

- Manuale PHP — Sessioni: https://www.php.net/session
- OWASP Session Management: https://cheatsheetseries.owasp.org/

Capitolo 9

Database con MySQLi

Mappa del capitolo

Teoria, Connessione, Query preparate, Transazioni, Gestione errori, Ottimizzazione, Esempi pratici, Caso di studio, Esercizi, Verifica, Riferimenti.

9.1 Teoria

L'estensione `mysqli` fornisce API procedurali e a oggetti per interagire con MySQL/MariaDB, con supporto a query preparate e transazioni.

9.2 Connessione al database

```php
<?php
$mysqli = new mysqli('localhost', 'user', 'pass', 'db');
if ($mysqli->connect_errno) {
    error_log('Connessione fallita: ' . $mysqli->connect_error);
    exit('DB non disponibile');
}
```

9.3 Query preparate

```php
<?php
$stmt = $mysqli->prepare('SELECT id, name FROM users WHERE email = ?');
$stmt->bind_param('s', $email);
$stmt->execute();
$result = $stmt->get_result();
while ($row = $result->fetch_assoc()) {
    echo $row['name'];
}
$stmt->close();
```

9.4 Query non sicure (senza bind) e SQL injection

```php
<?php
// ESEMPIO NON SICURO: concatenazione di input utente
$email = isset($_GET['email']) ? (string)$_GET['email'] : '';
```

```php
$sql = "SELECT id, name FROM users WHERE email = '" . $email . "'";
$res = $mysqli->query($sql);
if ($res) {
    while ($row = $res->fetch_assoc()) { echo $row['name']; }
}
// Input malevolo: x' OR '1'='1  -> restituisce TUTTI gli utenti
// $sql diventa: SELECT id, name FROM users WHERE email = 'x' OR '1'='1'
?>
```

Rifacimento sicuro con parametri bind

```php
<?php
// VERSIONE SICURA: placeholder e bind_param
$email = isset($_GET['email']) ? (string)$_GET['email'] : '';
$stmt = $mysqli->prepare('SELECT id, name FROM users WHERE email = ?');
if (!$stmt) { exit('Errore prepare'); }
$stmt->bind_param('s', $email);
$stmt->execute();
$result = $stmt->get_result();
while ($row = $result->fetch_assoc()) { echo $row['name']; }
$stmt->close();
?>
```

Dimostrazione

La versione non sicura consente l'iniezione di frammenti SQL in `$email`, alterando la logica della WHERE e restituendo risultati non previsti. Con i parametri bind, il valore viene inviato separatamente dall'istruzione SQL e trattato come dato, eliminando la vulnerabilità.

9.5 Transazioni

```php
<?php
$mysqli->begin_transaction();
try {
    $stmt = $mysqli->prepare('UPDATE accounts SET balance = balance - ?
        WHERE id = ?');
    $stmt->bind_param('di', $amount, $fromId);
    $stmt->execute();

    $stmt = $mysqli->prepare('UPDATE accounts SET balance = balance + ?
        WHERE id = ?');
    $stmt->bind_param('di', $amount, $toId);
    $stmt->execute();

    $mysqli->commit();
} catch (Throwable $e) {
    $mysqli->rollback();
    error_log($e->getMessage());
}
```

9.6 Gestione errori

- Controllare `$mysqli->errno` e `$mysqli->error`; loggare errori e non esporre dettagli sensibili all'utente. - Gestire eccezioni e timeouts; impostare charset `utf8mb4`.

```php
<?php
$mysqli->set_charset('utf8mb4');
if (!$mysqli->query('SET NAMES utf8mb4')) {
    error_log($mysqli->error);
}
```

9.7 Ottimizzazione delle query

- Indici appropriati, evitare SELECT * nelle tabelle grandi. - Limitare i risultati, usare paginazione. - Misurare con `EXPLAIN` e profilarne i piani.

Best practice

- Usare query preparate per prevenire SQL injection. - Gestire transazioni per operazioni atomiche. - Impostare charset `utf8mb4` e collation coerente. - Centralizzare la gestione della connessione e pooling se necessario.

Errori comuni

- Concatenare input utente nelle query. - Dimenticare `commit/rollback` e lasciare transazioni aperte. - Ignorare settaggio del charset e problemi di encoding.

9.8 Esempi pratici

```php
<?php
// Inserimento utente con validazioni minime
$stmt = $mysqli->prepare('INSERT INTO users(username,email,password_hash)
    VALUES (?,?,?)');
$stmt->bind_param('sss', $username, $email, $hash);
$stmt->execute();
if ($stmt->errno) { error_log($stmt->error); }
$stmt->close();
?>
```

9.9 Caso di studio

Registrazione e login: creazione utente, ricerca per username/email e verifica hash. Discutere error handling e messaggi user-friendly.

9.10 Esercizi

- Implementa paginazione server-side con LIMIT/OFFSET.
- Aggiungi indici e confronta piani di esecuzione con `EXPLAIN`.

9.11 Verifica

- Quali rischi introduce la concatenazione di input utente?
- Perché `utf8mb4` è preferibile a `utf8`?

9.12 Riferimenti

- Manuale PHP — MySQLi: https://www.php.net/mysqli
- MySQL Doc — Prepared Statements: https://dev.mysql.com/doc/

Appendice — Quick Reference

Questa appendice raccoglie schede riassuntive per i capitoli principali. Si veda anche il documento dedicato in `quick_reference/main.tex`.

Bibliografia

- Manuale ufficiale PHP: https://www.php.net/
- Linee guida PSR: https://www.php-fig.org/
- OWASP per riferimento alla sicurezza applicativa: https://owasp.org/